

### Sumário

1) Apresente o código fonte, em Verilog, dos módulos que descrevem os componentes do seu nRisc implementados nesta prática. ....	2
Extensor.....	2
MUX.....	2
2) Configure simulações que demonstrem o correto funcionamento de todos os componentes implementados nesta prática. Apresente o código fonte desse(s) módulo(s) de simulação. ....	3
Extensor.....	3
MUX.....	3
3) Demonstre e explique o funcionamento das simulações acima, usando fotos da tela ( <i>screenshots</i> ) da aplicação ModelSim em execução.....	4
Extensor:.....	4
MUX:.....	4
4) Apresente o código fonte, em Verilog, do módulo que descreve a interligação dos componentes do seu nRisc.....	5
5) Apresente o código fonte, em Verilog, do módulo que descreve a simulação do funcionamento do seu processador. ....	5
6) Demonstre e explique o funcionamento da simulação do programa definido por você no início do projeto usando fotos da tela ( <i>screenshots</i> ) da aplicação ModelSim em execução.....	6

1) Apresente o código fonte, em Verilog, dos módulos que descrevem os componentes do seu nRisc implementados nesta prática.

Os módulos que faltavam são: Extensor de sinal e o Multiplexador (MUX), os códigos estão em anexo.

Extensor:

```
1  /*
2  Implementação em Verilog dos componentes do nRisc que armazenam estado.
3  Aluno: Alexandre Roque.
4  Disciplina: Arquitetura e Organização de Computadores.
5  Professor: Mateus Felipe Tymburibá Ferreira.
6  */
7
8  module Extensor (Not_Estendido, Estendido);
9      input  [2:0] Not_Estendido;
10     output [7:0] Estendido;
11     assign Estendido = Not_Estendido;
12 endmodule
13
```

MUX:

```
1  /*
2  Implementação em Verilog dos componentes do nRisc que armazenam estado.
3  Aluno: Alexandre Roque.
4  Disciplina: Arquitetura e Organização de Computadores.
5  Professor: Mateus Felipe Tymburibá Ferreira.
6  */
7
8  module MUX(ResultA, ResultB, Chave, ResultFinal);
9
10     input [7:0] ResultA, ResultB;
11     input Chave;
12     output wire [7:0] ResultFinal;
13
14     assign ResultFinal = (Chave) ? ResultB : ResultA;
15
16 endmodule
17
```

2) Configure simulações que demonstrem o correto funcionamento de todos os componentes implementados nesta prática. Apresente o código fonte desse(s) módulo(s) de simulação.

Os códigos estão em anexo.

Extensor:

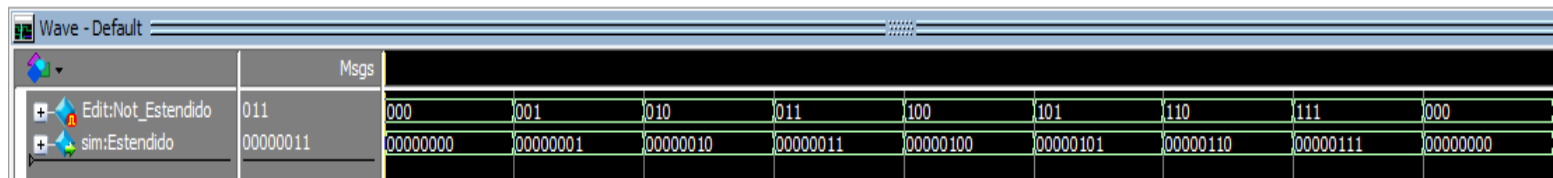
```
view wave  
wave clipboard store  
  
wave create -driver freeze -pattern counter -startvalue 000 -endvalue 111 -type Range -  
direction Up -period 50ps -step 1 -repeat forever -range 2 0 -starttime 0ps -endtime 1000ps  
sim:/Extensor/Not_Estendido  
  
WaveExpandAll -1  
WaveCollapseAll -1  
  
wave clipboard restore
```

MUX:

```
WaveRestoreZoom {100 ps} {1100 ps}  
view wave  
wave clipboard store  
  
wave create -driver freeze -pattern random -initialvalue zzzzzzzz -period 50ps -random_type Uniform -seed 5 -range  
7 0 -starttime 0ps -endtime 1000ps sim:/MUX/ResultA  
  
WaveExpandAll -1  
  
wave create -driver freeze -pattern random -initialvalue zzzzzzzz -period 50ps -random_type Uniform -seed 1 -range  
7 0 -starttime 0ps -endtime 1000ps sim:/MUX/ResultB  
  
WaveExpandAll -1  
  
wave create -driver freeze -pattern clock -initialvalue HiZ -period 150ps -dutycycle 50 -starttime 0ps -endtime 1000ps  
sim:/MUX/Chave  
  
WaveCollapseAll -1  
  
wave clipboard restore
```

3) Demonstre e explique o funcionamento das simulações acima, usando fotos da tela (*screenshots*) da aplicação ModelSim em execução.

Extensor:

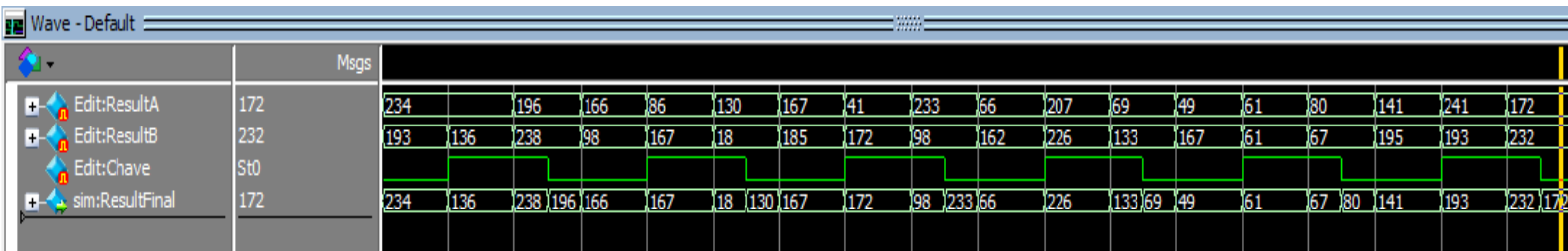


The screenshot shows the ModelSim Wave window with two signals: 'Edit:Not\_Estendido' and 'sim:Estendido'. The 'Edit:Not\_Estendido' signal has a value of 011. The 'sim:Estendido' signal has a value of 00000011. The waveforms show the signals over time, with the 'sim:Estendido' signal being a 3-bit extension of the 'Edit:Not\_Estendido' signal.

Signal	Value
Edit:Not_Estendido	011
sim:Estendido	00000011

Como podemos ver na simulação acima, o extensor de sinal, a partir de um valor de 3 bits não estendido (Not\_Estendido), transforma-o em um valor estendido de 8 bits (Estendido). A partir dessa ideia, fiz mais dois extensores de 2 bits e de 5 bits.

MUX:



The screenshot shows the ModelSim Wave window with four signals: 'Edit:ResultA', 'Edit:ResultB', 'Edit:Chave', and 'sim:ResultFinal'. The 'Edit:ResultA' signal has a value of 172. The 'Edit:ResultB' signal has a value of 232. The 'Edit:Chave' signal has a value of St0. The 'sim:ResultFinal' signal has a value of 172. The waveforms show the signals over time, with the 'sim:ResultFinal' signal being the output of the MUX operation.

Signal	Value
Edit:ResultA	172
Edit:ResultB	232
Edit:Chave	St0
sim:ResultFinal	172

Para o MUX, temos como entrada dois valores (ResultA e ResultB) e a partir da chave colocada, sendo 0 ou 1, o multiplexador irá escolher um dos resultados para colocar no resultado final (ResultFinal).

4) Apresente o código fonte, em Verilog, do módulo que descreve a interligação dos componentes do seu nRisc.

O código da implementação está em anexo. O arquivo principal é o “processadorNrisc.v”.

5) Apresente o código fonte, em Verilog, do módulo que descreve a simulação do funcionamento do seu processador.

O código está em anexo, o arquivo para o modulo de simulação é o “simula.v”.

6) Demonstre e explique o funcionamento da simulação do programa definido por você no início do projeto usando fotos da tela (*screenshots*) da aplicação ModelSim em execução.

```
# ++++++
# PC E INSTRUCAO
# Instrucao = xxxxxxxx
# Clock = 0
# PC = 00010100
# -----
# Controle
# EscPC = x
# FonteReg = x
# Jump = x
# EscreverMemoria = x
# LerMemoria = x
# ULAFonte = x
# BEQ = x
# ULAOp = xxx
# -----
# Registradores
# Reg00 = 0
# Reg01 = 0
# Reg02 = 0
# Reg03 = 0
# Sinal RegWrite = x
# REG 1 = x
# REG 2 = x
# DadoPraEscrever = x
# RegEscrito = x
# Dado1 = x
# Dado2 = x
# -----
# ULA
# DADO 1 = x
# DADO 2 = x
# RESULT = x
# ZERO = x
# ++++++
```

Para a simulação, primeiramente instanciei todos os componentes utilizados e atribui os valores que serão utilizados. O passo mais importante foi inserir as instruções no modulo de simulação, os detalhes de tradução e execução estão mais abaixo no relatório.

A figura mostra o início do programa, somente os registradores estão inicializados com 0, todos os outros estão como “x”.

```

# ++++++
# PC E INSTRUCAO
# Instrucao = 00000011
# Clock = 1
# PC = 00010100
# -----
# Controle
# EscPC = 1
# FonteReg = 0
# Jump = 0
# EscreverMemoria = 0
# LerMemoria = 0
# ULAFonte = 1
# BEQ = 0
# ULAOp = 000
# -----
# Registradores
# Reg00 = 0
# Reg01 = 0
# Reg02 = 0
# Reg03 = 0
# Sinal RegWrite = 1
# REG 1 = 0
# REG 2 = 1
# DadoPraEscrever = 3
# RegEscrito = 0
# Dado1 = 0
# Dado2 = 0
# -----
# ULA
# DADO 1 = 0
# DADO 2 = 3
# RESULT = 3
# ZERO = x
# ++++++

```

Em seguida temos a primeira instrução, como podemos ver está de acordo com o código em binário que traduzi, sendo uma instrução de adição com imediato.

```
00000011  addi(000) $s0(00)  imm(011)
```

```
nrisc.PC_module.PC = 20;
```

```

memIns.memoria[20] = 8'b00000011;
memIns.memoria[21] = 8'b00001111;
memIns.memoria[22] = 8'b01101011;
memIns.memoria[23] = 8'b00110000;
memIns.memoria[24] = 8'b01001001;
memIns.memoria[25] = 8'b10010110;
memIns.memoria[3]  = 8'b11000000;

```

```

# ++++++
# PC E INSTRUCAO
# Instrucao = 11000000
# Clock = 1
# PC = 00000011
# -----
# Controle
# EscPC = 0
# FonteReg = 0
# Jump = 0
# EscreverMemoria = 0
# LerMemoria = 0
# ULAFonte = 0
# BEQ = 0
# ULAOp = 000
# -----
# Registradores
# Reg00 = 3
# Reg01 = 0
# Reg02 = 21
# Reg03 = 0
# Sinal RegWrite = 0
# REG 1 = 0
# REG 2 = 0
# DadoPraEscrever = 6
# RegEscrito = 0
# Dado1 = 3
# Dado2 = 3
# -----
# ULA
# DADO 1 = 3
# DADO 2 = 3
# RESULT = 6
# ZERO = 1
# ++++++

```

Essa é a iteração do processador, onde temos a instrução “11000000”, que é o “halt”. No registrador 02 podemos ver o resultado da multiplicação que é 21, resultado de  $3+3+3+3+3+3+3$ , ou seja,  $3 \times 7$ .



Para a elaboração desse código, irei utilizar 3 registradores e 6 instruções, listadas abaixo. O objetivo dele é calcular uma multiplicação a partir da soma, por exemplo, 3 vezes 2, seria a soma de 3+3.

```
.main

# Atribuindo valores para as variáveis

li $s0, 0          #$s0 recebe o valor do multiplicador
addi $s1, $s1, 7    #$s1 recebe o valor do multiplicando

LOOP:
    beq $s1, $zero, Sair #se $s1 for 0, vá para o sair

    add $s2, $s2, $s0    #adiciona no $s2 o valor de $s0, aux+=aux

    subi $s1,$s1,1       #subtrai um de $s1

    sw $s2, 1           #memoria[1] recebe o valor de $s2

    j LOOP               #volta no Loop

Sair: #END            #Encerra o programa
```

As instruções suportadas:	Representação	OpCode(3 bits)
- addi	addi rd, rd, imm	000
- add	add rd, rd, rt	001
- subi	subi rd, rd, imm	010
- beq	beq rs, \$zero, label	011
- sw	sw rs, imm	101
- li	li rs, imm	111
- j	j label	101
- h	(halt → PC)	110

Registradores	Representação (2bit)
- \$s0	00
- \$s1	01
- \$s2	10

Instrução	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
addi	0	0	0	rd		imm		
add	0	0	1	rd		rt		x
subi	0	1	0	rd		imm		
beq	0	1	1	rs		\$zero	endereço	
sw	1	0	1	rs		imm		
li	1	1	1	rs		imm		
j	1	0	0	endereço				
h	1	1	0	(halt → PC)				

Código em binário linha por linha:

CÓDIGO	OPCODE	REGIST	imm/endereço/regis
11100011	li(111)	\$s0(00)	imm(000)
00001111	addi(000)	\$s1(01)	imm(111)
011010&&	beq(011)	\$s1(01)	endereço(11)
0011000x	add(001)	\$s2(10)	\$s0(00)
01001001	subi(010)	\$s1(01)	imm(001)
10110001	sw(101)	\$s0(00)	imm(001)
100&&&&	j(100)	endereço(10110)	
110xxxxx	(halt→PC)	Sair	

## DataPath Atualizado

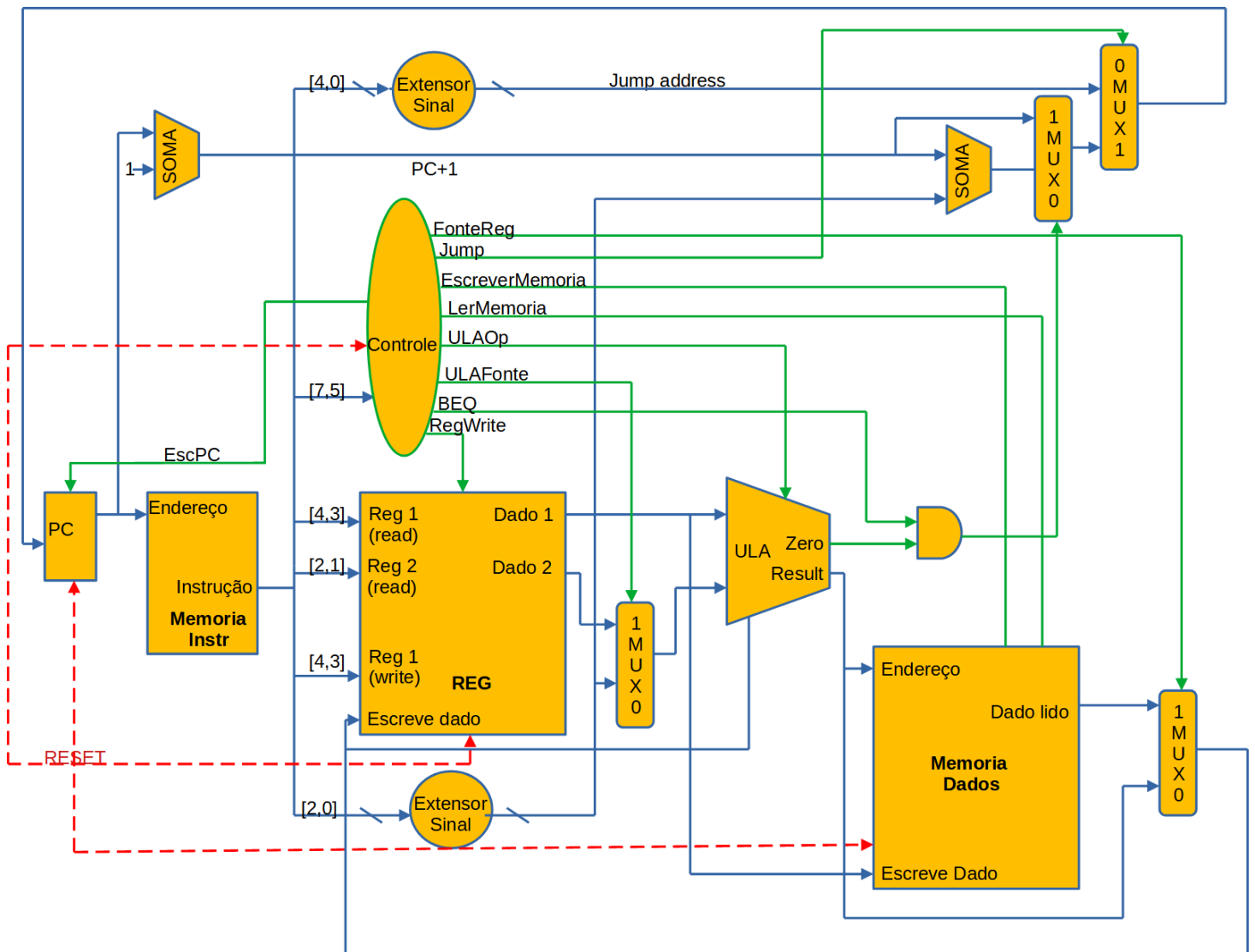


Tabela de sinais para cada instrução atualizado.

Instruções	Sinais								
	EscPC	FonteReg	Jump	Escrever Memoria	Ler Memoria	ULA Fonte	BEQ	Reg Write	ULAOp
<b>addi</b>	1	0	0	0	0	1	0	1	000
<b>add</b>	1	0	0	0	0	0	0	1	001
<b>subi</b>	1	0	0	0	0	1	0	1	010
<b>beq</b>	1	0	0	0	0	1	1	0	011
<b>J</b>	1	0	1	0	0	1	1	0	110
<b>h</b>	0	0	0	0	0	0	0	0	111