Universidade Federal de Roraima
Centro de Ciência e Tecnologia
Departamento de Ciência da Computação

UFRR

ProLAG

Processador RISC monociclo de 16 bits

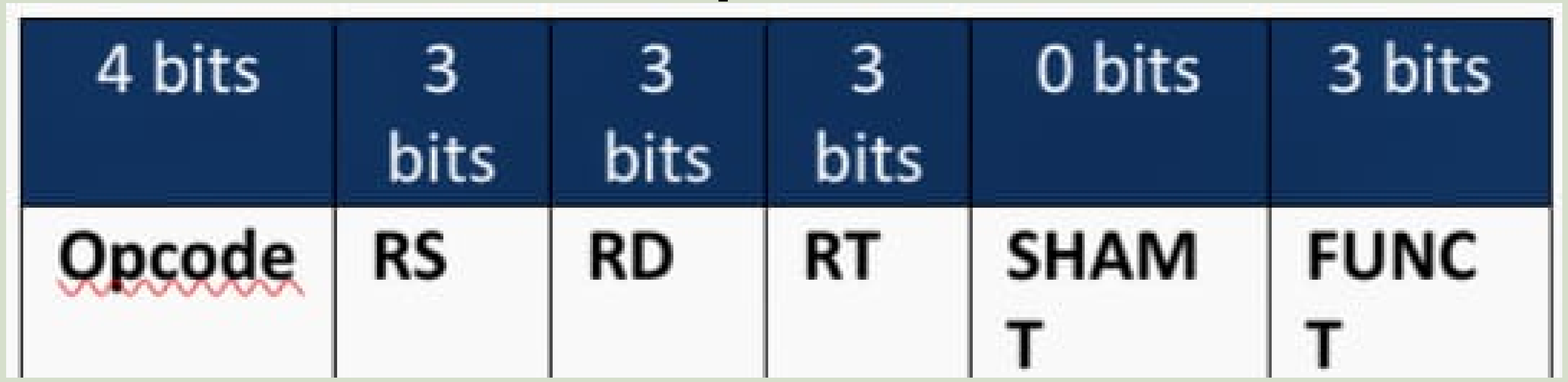
Integrantes: Ângelo Garcia Fernandez, Guilherme Almeida da Luz e Leonam de Sousa Silva
Disciplina: DCC301 - Arquitetura e Organização de Computadores (2023.2)
Docente: Dr. Herbert Oliveira Rocha
30 de Novembro de 2023
Boa Vista - RR

Registradores

Nome	Endereço
$zero	'000’
$t0	'001’
$t1	'010’
$t2	'011’
$s0	'100’
$s1	'101’
$s2	'110’
$s3	'111’

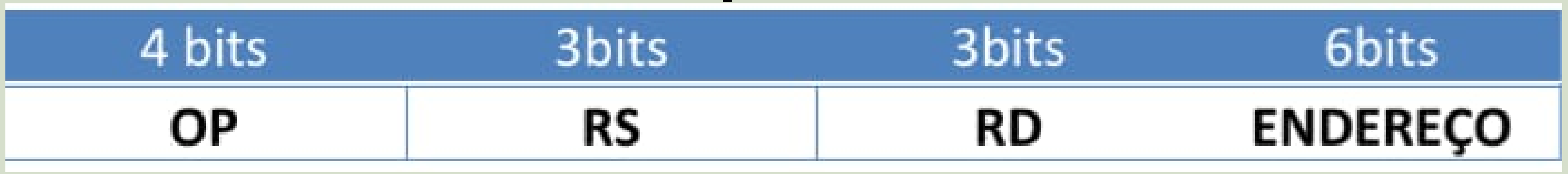
Conjunto de Instruções

Tipo R



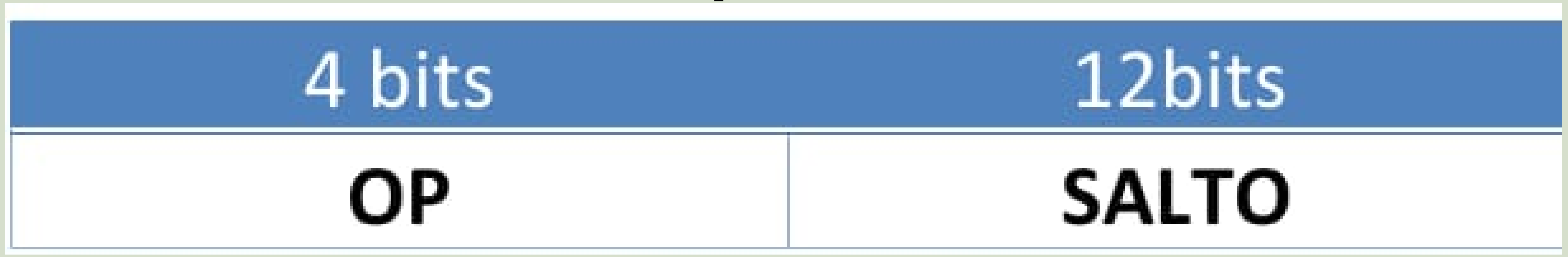
4 bits	3 bits	3 bits	3 bits	0 bits	3 bits
Opcode	RS	RD	RT	SHAMT	FUNCT

Tipo I



4 bits	3bits	3bits	6bits
OP	RS	RD	ENDEREÇO

Tipo J

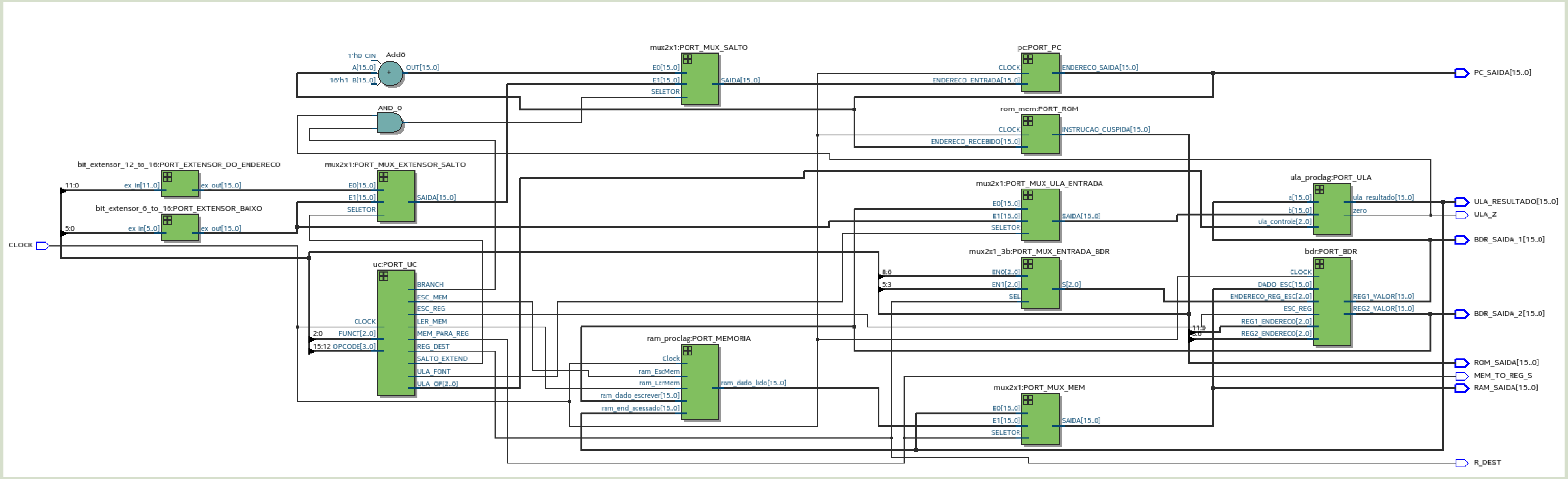


4 bits	12bits
OP	SALTO

Conjunto de instruções

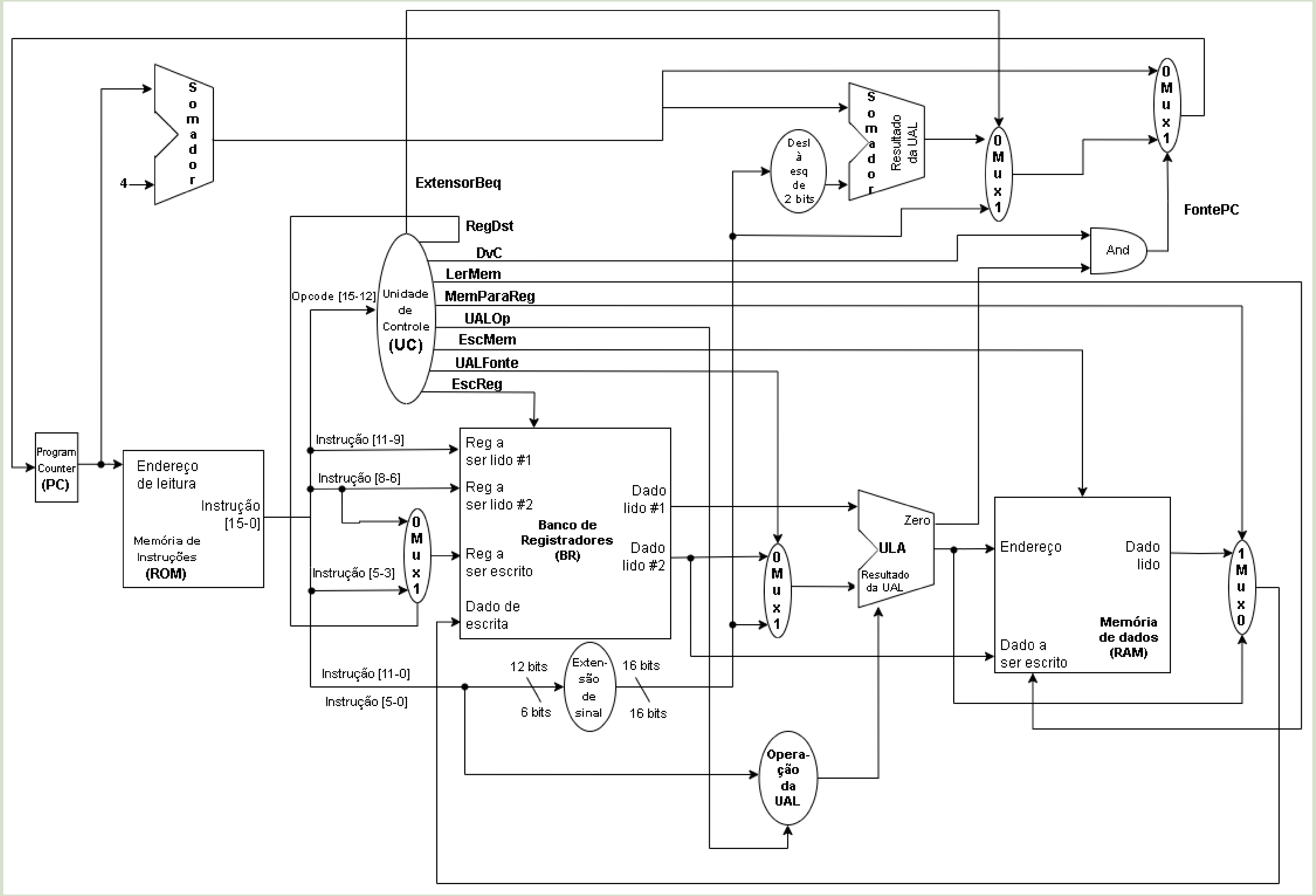
OPCODE	Funct	Nome	Formato	Sintaxe
'0001’	'000’	add	R	add $r1, $r2
'0001’	'001’	sub	R	sub $r1, $r2
'0010’	xxxx	addi	I	addi $r1, CONSTANTE
'0011’	xxxx	subi	I	subi $r1, CONSTANTE
'0100’	xxxx	lw	I	lw $r1, CONSTANTE, $rb
'0101’	xxxx	sw	I	sw $t0 CONSTANTE, $rb
'0110’	xxxx	beq	I	beq $r1, $r2, SALTO
'0111’	xxxx	j	J	j ENDEREÇO

RTL Viewer



mux2x1:PORT_MUX_SALTO
pc:PORT_PC
Add0
1'h0 CIN
A[15..0]
16'h1 B[15..0]
OUT[15..0]
E0[15..0]
E1[15..0]
SELETOR
SAIDA[15..0]
CLOCK
ENDERECO_ENTRADA[15..0]
ENDERECO_SAIDA[15..0]
PC_SAIDA[15..0]
AND_0
rom_mem:PORT_ROM
ENDERECO_RECEBIDO[15..0]
INSTRUCAO_CUSPIDA[15..0]
bit_extensor_12_to_16:PORT_EXTENSOR_DO_ENDERECO
mux2x1:PORT_MUX_EXTENSOR_SALTO
11:0
ex_in[11..0]
ex_out[15..0]
bit_extensor_6_to_16:PORT_EXTENSOR_BAIXO
5:0
ex_in[5..0]
CLOCK
mux2x1:PORT_MUX_ULA_ENTRADA
ula_proclag:PORT_ULA
a[15..0]
b[15..0]
ula_controle[2..0]
ula_resultado[15..0]
zero
ULA_RESULTADO[15..0]
ULA_Z
BDR_SAIDA_1[15..0]
mux2x1_3b:PORT_MUX_ENTRADA_BDR
8:6
5:3
EN0[2..0]
EN1[2..0]
SEL
S[2..0]
bdr:PORT_BDR
DADO_ESC[15..0]
ENDERECO_REG_ESC[2..0]
ESC_REG
REG1_ENDERECO[2..0]
REG2_ENDERECO[2..0]
REG1_VALOR[15..0]
REG2_VALOR[15..0]
11:9
8:6
BDR_SAIDA_2[15..0]
uc:PORT_UC
BRANCH
ESC_MEM
ESC_REG
LER_MEM
MEM_PARA_REG
REG_DEST
SALTO_EXTEND
ULA_FONT
ULA_OP[2..0]
2:0
FUNCT[2..0]
15:12
OPCODE[3..0]
ram_proclag:PORT_MEMORIA
Clock
ram_EscMem
ram_LerMem
ram_dado_escrever[15..0]
ram_end_acessado[15..0]
ram_dado_lido[15..0]
mux2x1:PORT_MUX_MEM
ROM_SAIDA[15..0]
MEM_TO_REG_S
RAM_SAIDA[15..0]
R_DEST

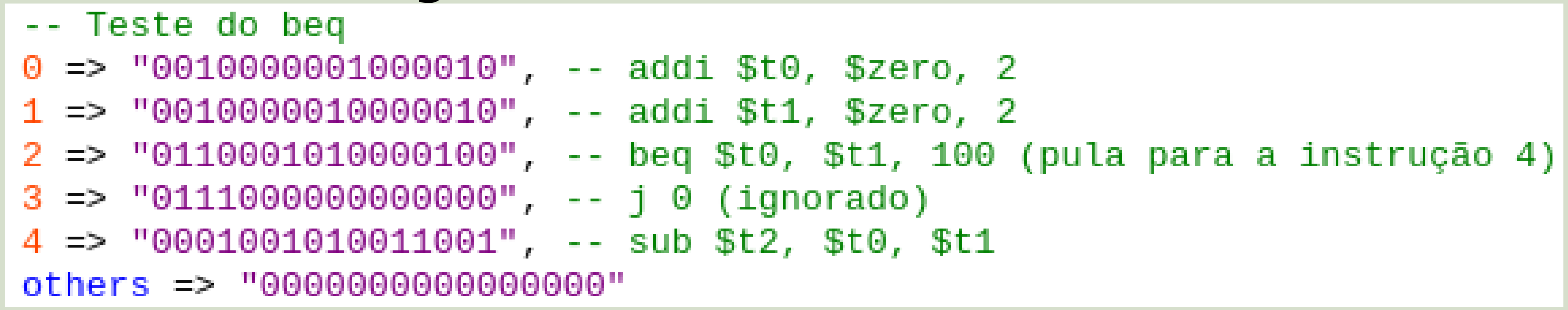
Datapath



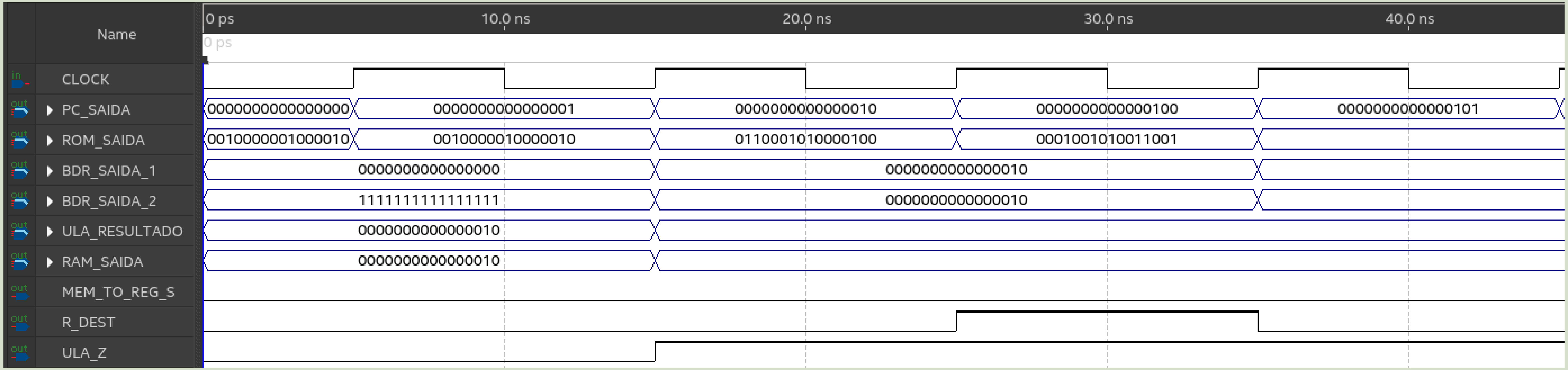
Somador
4
ExtensorBeq
RegDst
DvC
LerMem
MemParaReg
UALOp
EscMem
UALFonte
EscReg
Opcode [15-12]
Unidade de Controle (UC)
Desl à esq de 2 bits
Somador
Resultado da UAL
0 Mux 1
0 Mux 1
FontePC
And
Program Counter (PC)
Endereço de leitura
Instrução [15-0]
Memória de Instruções (ROM)
Instrução [11-9]
Instrução [8-6]
Instrução [5-3]
0 Mux 1
Reg a ser lido #1
Reg a ser lido #2
Reg a ser escrito
Dado de escrita
Banco de Registradores (BR)
Dado lido #1
Dado lido #2
0 Mux 1
ULA
Zero
Resultado da UAL
Endereço
Dado lido
Dado a ser escrito
Memória de dados (RAM)
1 Mux 0
Instrução [11-0]
Instrução [5-0]
12 bits
6 bits
Extensão de sinal
16 bits
16 bits
Operação da UAL

Testes:

1 - Instruções aritméticas e beq

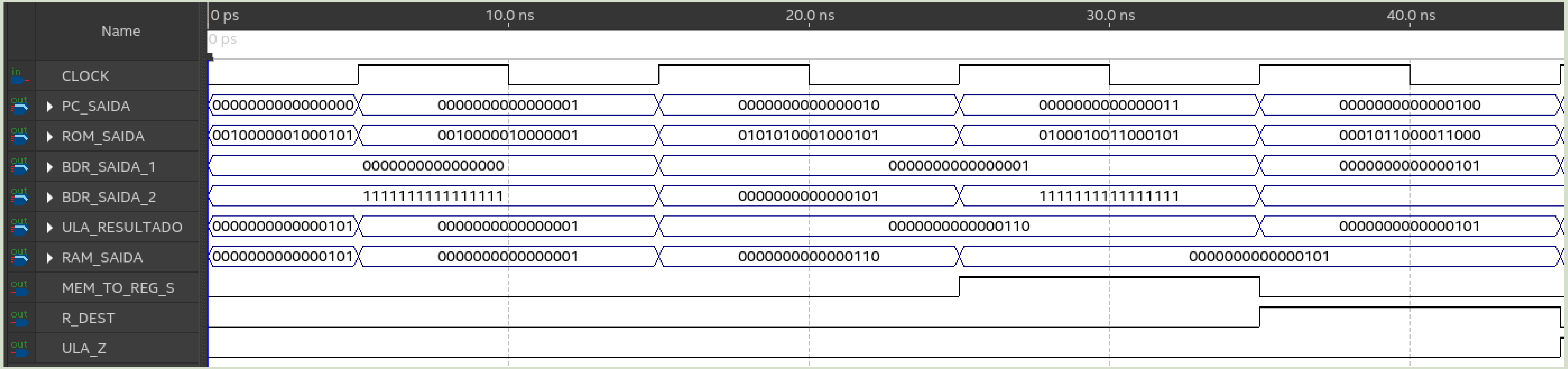


```
-- Teste do beq
0 => "0010000001000010", -- addi $t0, $zero, 2
1 => "0010000010000010", -- addi $t1, $zero, 2
2 => "0110001010000100", -- beq $t0, $t1, 100 (pula para a instrução 4)
3 => "0111000000000000", -- j 0 (ignorado)
4 => "0001001010011001", -- sub $t2, $t0, $t1
others => "0000000000000000"
```

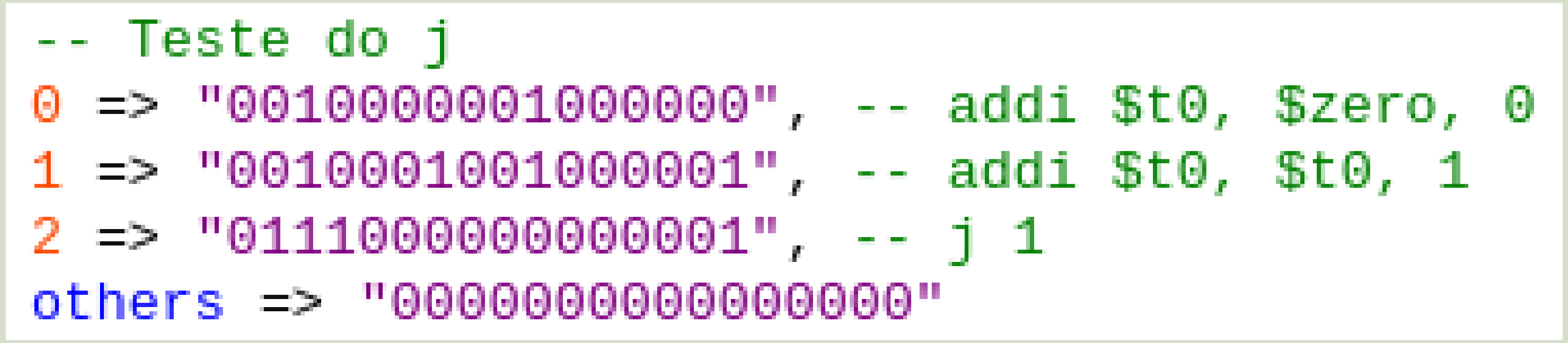


2 - Instruções de Load e Store

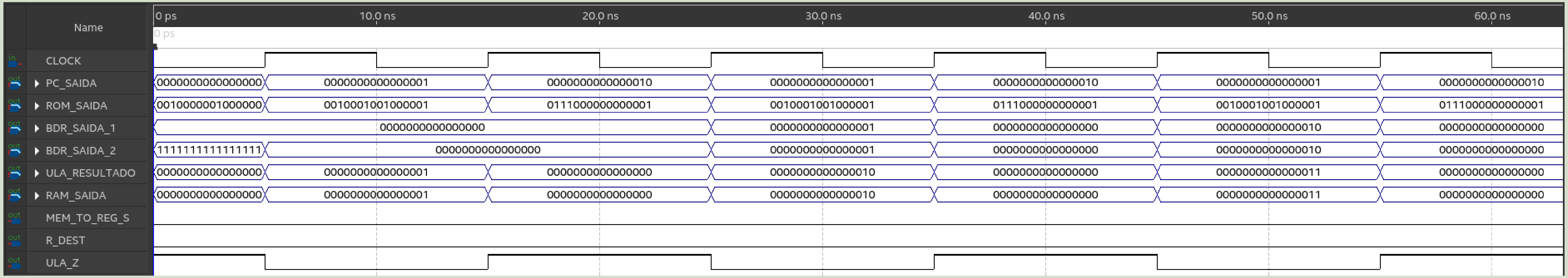
```
-- Teste Load e Store
0 => "0010000001000101", -- addi $t0, $zero, 5
1 => "0010000010000001", -- addi $t1, $zero, 1
2 => "0101010001000101", -- sw $t0, 5, $t1 (joga $t0 em 5 + $t1)
3 => "0100010011000101", -- lw $t2, 5, $t1 (carrega 5 + $t1 em $t2)
4 => "0001011000011000", -- add $t2, $t2, $zero
others => "0000000000000000"
```


Name
0 ps
10.0 ns
20.0 ns
30.0 ns
40.0 ns
CLOCK
PC_SAIDA 0000000000000000 0000000000000001 0000000000000010 0000000000000011 0000000000000100
ROM_SAIDA 0010000001000101 0010000010000001 0101010001000101 0100010011000101 0001011000011000
BDR_SAIDA_1 0000000000000000 0000000000000001 0000000000000101
BDR_SAIDA_2 1111111111111111 0000000000000101 1111111111111111
ULA_RESULTADO 0000000000000101 0000000000000001 0000000000000110 0000000000000101
RAM_SAIDA 0000000000000101 0000000000000001 0000000000000110 0000000000000101
MEM_TO_REG_S
R_DEST
ULA_Z

3 - Instrução de Salto

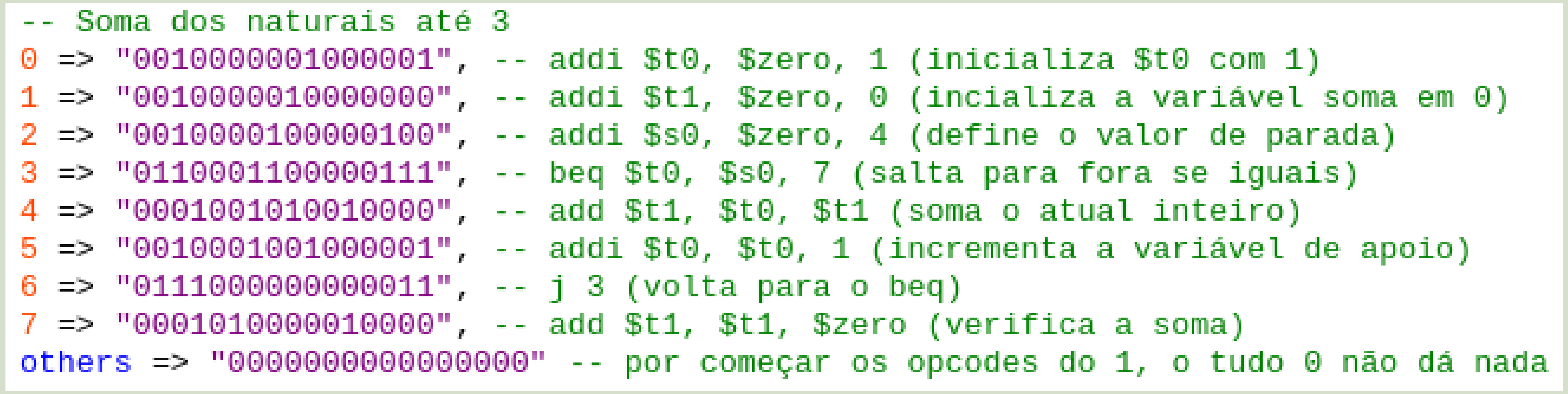


```
-- Teste do j
0 => "0010000001000000", -- addi $t0, $zero, 0
1 => "0010001001000001", -- addi $t0, $t0, 1
2 => "0111000000000001", -- j 1
others => "0000000000000000"
```

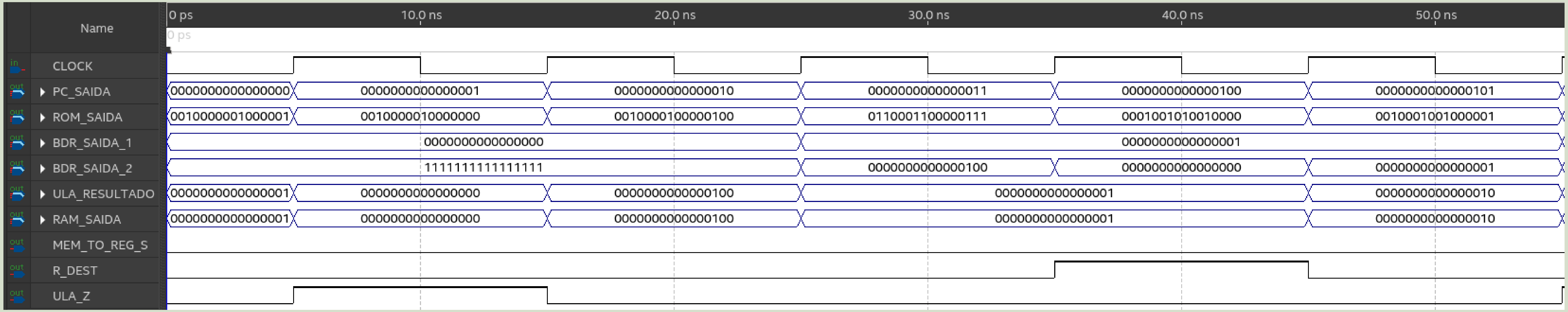


```
PC_SAIDA      0000000000000000 | 0000000000000001 | 0000000000000010 | 0000000000000001 | 0000000000000010 | 0000000000000001 | 0000000000000010
ROM_SAIDA     0010000001000000 | 0010001001000001 | 0111000000000001 | 0010001001000001 | 0111000000000001 | 0010001001000001 | 0111000000000001
BDR_SAIDA_1   0000000000000000 | 0000000000000001 | 0000000000000000 | 0000000000000010 | 0000000000000000
BDR_SAIDA_2   1111111111111111 | 0000000000000000 | 0000000000000001 | 0000000000000000 | 0000000000000010 | 0000000000000000
ULA_RESULTADO 0000000000000000 | 0000000000000001 | 0000000000000000 | 0000000000000010 | 0000000000000000 | 0000000000000011 | 0000000000000000
RAM_SAIDA     0000000000000000 | 0000000000000001 | 0000000000000000 | 0000000000000010 | 0000000000000000 | 0000000000000011 | 0000000000000000
```

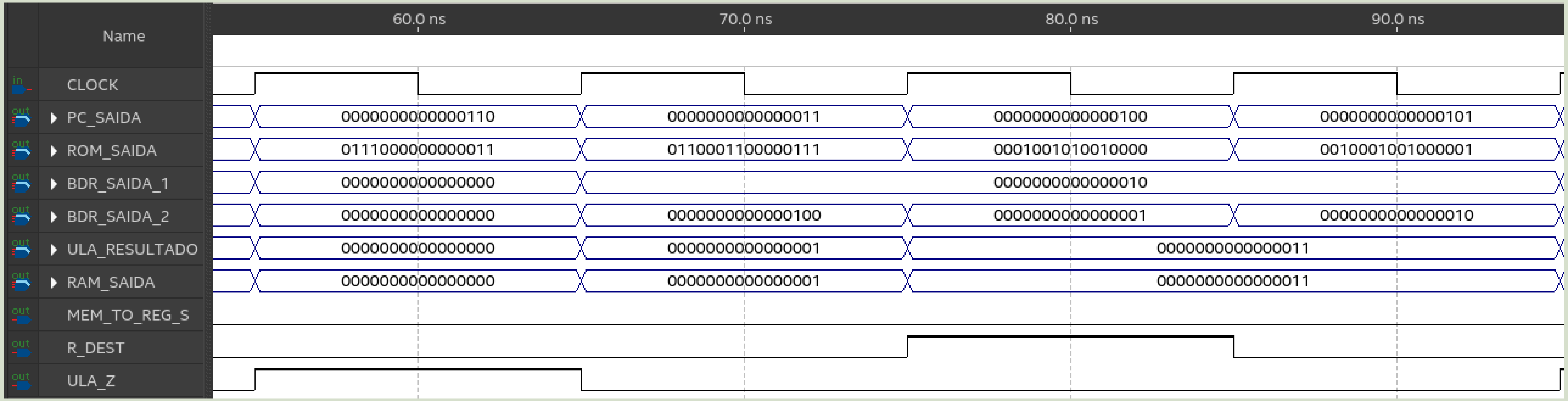
4 - Soma dos naturais até o 3



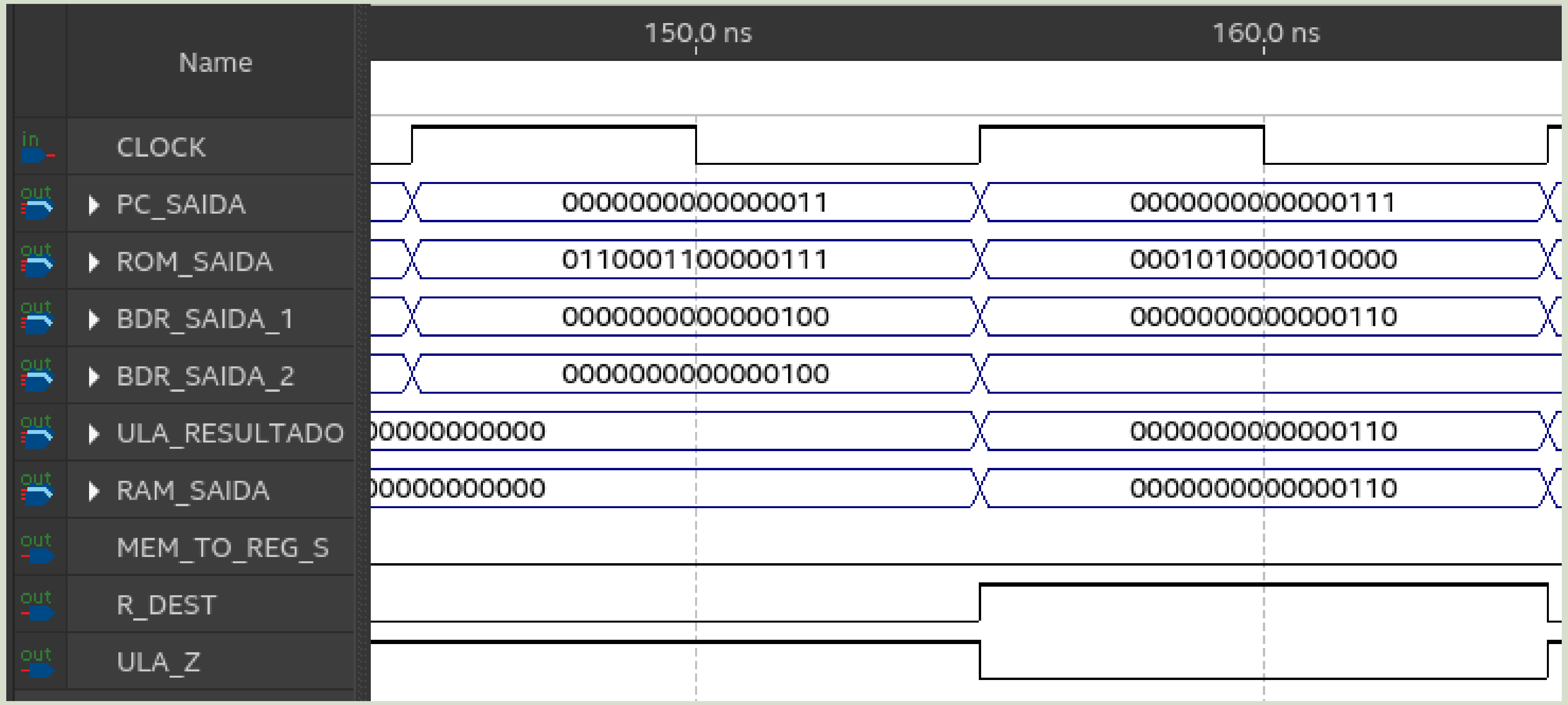
```
-- Soma dos naturais até 3
0 => "0010000001000001", -- addi $t0, $zero, 1 (inicializa $t0 com 1)
1 => "0010000010000000", -- addi $t1, $zero, 0 (incializa a variável soma em 0)
2 => "0010000100000100", -- addi $s0, $zero, 4 (define o valor de parada)
3 => "0110001100000111", -- beq $t0, $s0, 7 (salta para fora se iguais)
4 => "0001001010010000", -- add $t1, $t0, $t1 (soma o atual inteiro)
5 => "0010001001000001", -- addi $t0, $t0, 1 (incrementa a variável de apoio)
6 => "0111000000000011", -- j 3 (volta para o beq)
7 => "0001010000010000", -- add $t1, $t1, $zero (verifica a soma)
others => "0000000000000000" -- por começar os opcodes do 1, o tudo 0 não dá nada
```



```
Name            0 ps               10.0 ns             20.0 ns             30.0 ns             40.0 ns             50.0 ns
CLOCK
PC_SAIDA        0000000000000000   0000000000000001    0000000000000010    0000000000000011    0000000000000100    0000000000000101
ROM_SAIDA       0010000001000001   0010000010000000    0010000100000100    0110001100000111    0001001010010000    0010001001000001
BDR_SAIDA_1     0000000000000000                                           0000000000000001
BDR_SAIDA_2     1111111111111111                                           0000000000000100    0000000000000000    0000000000000001
ULA_RESULTADO   0000000000000001   0000000000000000    0000000000000100    0000000000000001                        0000000000000010
RAM_SAIDA       0000000000000001   0000000000000000    0000000000000100    0000000000000001                        0000000000000010
MEM_TO_REG_S
R_DEST
ULA_Z
```


Name
60.0 ns
70.0 ns
80.0 ns
90.0 ns
CLOCK
PC_SAIDA
0000000000000110
0000000000000011
0000000000000100
0000000000000101
ROM_SAIDA
0111000000000011
0110001100000111
0001001010010000
0010001001000001
BDR_SAIDA_1
0000000000000000
0000000000000010
BDR_SAIDA_2
0000000000000000
0000000000000100
0000000000000001
0000000000000010
ULA_RESULTADO
0000000000000000
0000000000000001
0000000000000011
RAM_SAIDA
0000000000000000
0000000000000001
0000000000000011
MEM_TO_REG_S
R_DEST
ULA_Z

***


Name
150.0 ns
160.0 ns
CLOCK
PC_SAIDA
0000000000000011
0000000000000111
ROM_SAIDA
0110001100000111
0001010000010000
BDR_SAIDA_1
0000000000000100
0000000000000110
BDR_SAIDA_2
0000000000000100
ULA_RESULTADO
00000000000
0000000000000110
RAM_SAIDA
00000000000
0000000000000110
MEM_TO_REG_S
R_DEST
ULA_Z

Referências

Organização e Arquitetura de Computadores - 4a ed. David A. Patterson e John L. Hennessy

8 Bit Microprocessor Design Using VHDL. Disponível em: <https://www.youtube.com/watch?v=tTlhIDgAGYY>. Acesso em: 30 nov. 2023.

HENRIQUE, E. Projeto CPU EK. Disponível em: <https://github.com/ed-henrique/8-bit-CPU/tree/main>. Acesso em: 30 nov. 2023.

FPGA4student.com. VHDL code for MIPS Processor. Disponível em: https://www.fpga4student.com/2017/09/vhdl-code-for-mips-processor.html. Acesso em: 24 nov. 2023.