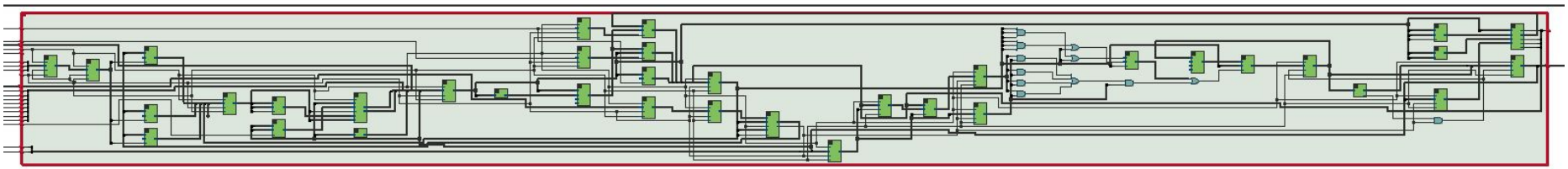
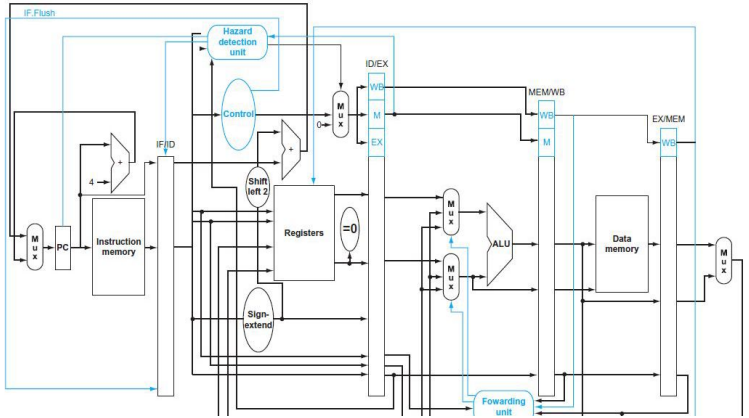
Grupo E: Implementação do subconjunto de instruções LEGv8



Visão Geral

- 37 instruções para o fluxo de dados principal.
- 19 instruções para a unidade auxiliar (ponto flutuante).
- Ponto de partida: Pipeline do livro



CORE INSTRUCTION FORMATS

R	opcode	Rm	shamt	Rn	Rd
	31 21	20 16	15 10	9 5	4 0
I	opcode	ALU_immediate		Rn	Rd
	31 22	21 10		9 5	4 0
D	opcode	DT_address	op	Rn	Rt
	31 21	20 12	11 10	9 5	4 0
B	opcode	BR_address			
	31 26	25 0			
CB	Opcode	COND_BR_address			Rt
	31 24	23 5			4 0
IW	opcode	MOV_immediate			Rd
	31 21	20 5			4 0

Divisão principal

- Projeto da unidade de controle
- Adaptações do datapath

Projeto da UC

Unidade de Controle Principal

“As simple as possible”

Novos sinais com valor zero para instruções que já funcionavam

Transcrição direta de tabela de instruções para componente

Menor quantidade de codificação possível

Trecho da tabela de instruções

						Sinais iniciais									Extensões								
1	Instrução	Versão DP	Classe	Mneumonico	Código binário	Reg2Loc	Uncondbranch	Branch	MemRead	MemtoReg	ALU control lines	MemWrite	ALUSrc	RegWrite	SetFlags	bcond	blink	bregister	bnz	zeroext0	zeroext1	zeroext2	exclusive
2	Add	Core	Aritimética	ADD	10001011000	0	0	0	0	0	0010	0	0	1	0	0	0	0	0	0	0	0	0
3	Add Immediate	Core	Aritimética	ADDI	1001000100 1001000100	X	0	0	0	0	0010	0	1	1	0	0	0	0	0	0	0	0	0
4	Add Immediate & Set flags	Core	Aritimética	ADDIS	1011000100 1011000100	X	0	0	0	0	0010	0	1	1	1	0	0	0	0	0	0	0	0
5	Add & Set Flags	Core	Aritimética	ADDS	10101011000	0	0	0	0	0	0010	0	0	1	1	0	0	0	0	0	0	0	0
6	AND	Core	Aritimética	AND	10001010000	0	0	0	0	0	0000	0	0	1	0	0	0	0	0	0	0	0	0

Unidade de controle da ULA

Divide as instruções aritméticas em 4 grupos

AluOp 00: Soma

AluOp 01: Subtração

AluOp 10: Cópia, e lógico, soma, subtração, ou inclusivo, ou exclusivo

AluOp 11: E lógico, MovK, soma, ou inclusivo, deslocamento para esquerda, deslocamento para direita, ou exclusivo, subtração.

Divisão gerada a partir do agrupamento do campo func (Instr[31 - 21]).

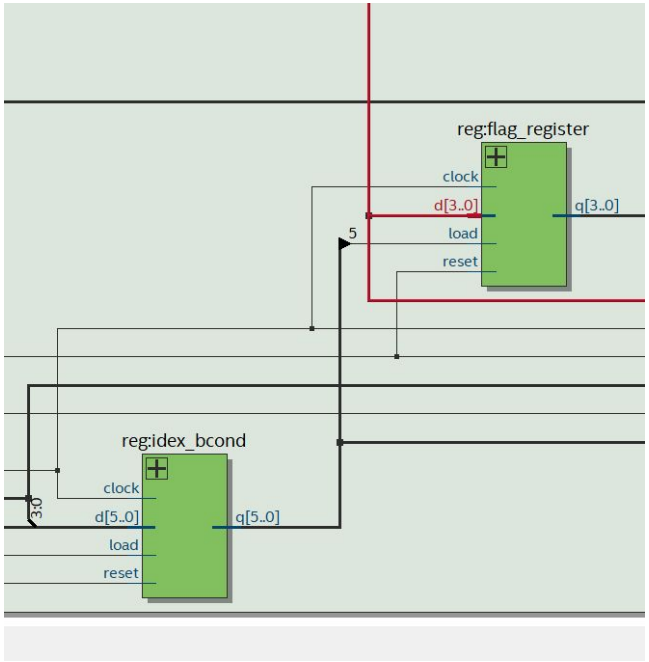
Projeto do DP

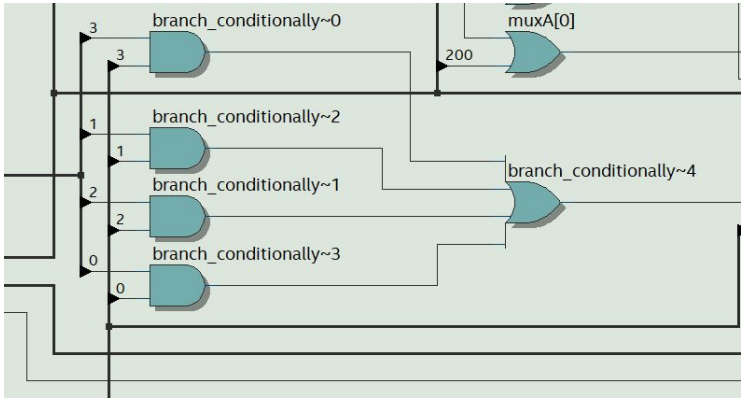
Divisão das instruções

Divididas conforme necessidade de alterações no DP

- Instruções aritméticas
- Instruções de Branch
- Instruções de Load
- Instruções de Store

Aritméticas

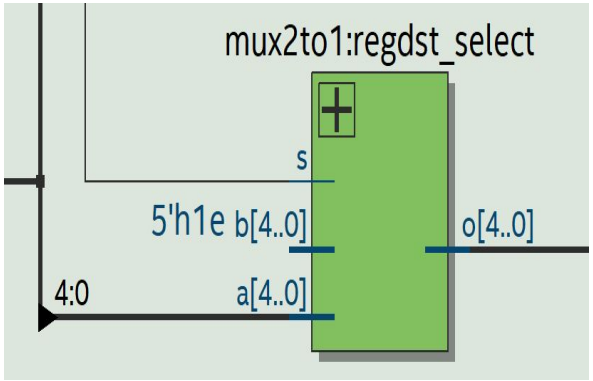




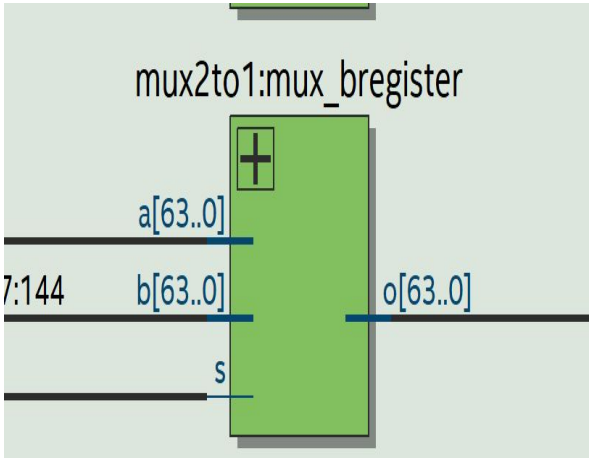
- Registrador para armazenar Flags geradas pela ULA, habilitado por novo sinal SetFlags.
- Unidade para definir sinal de branch condicional (bcond & flags != 0)
- ULA suporta todas as operações.

Branch

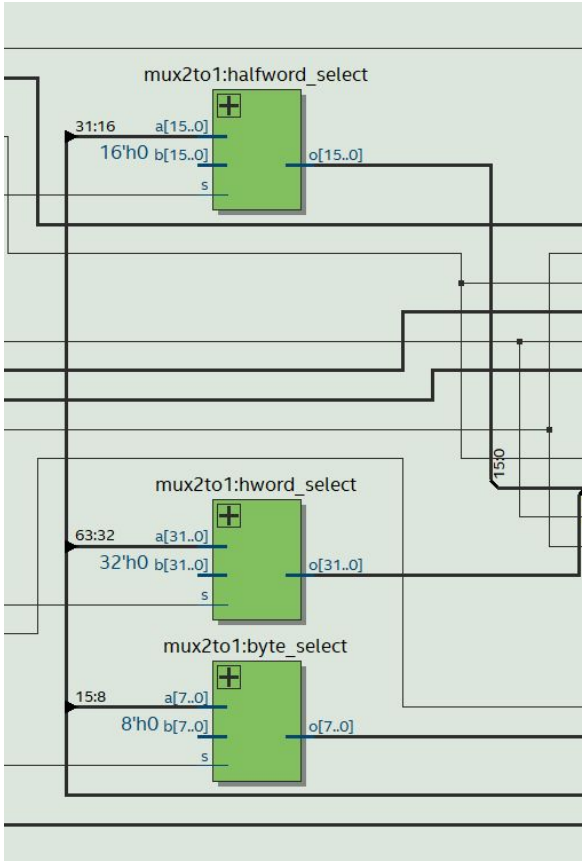
Branch and Link



Branch Register



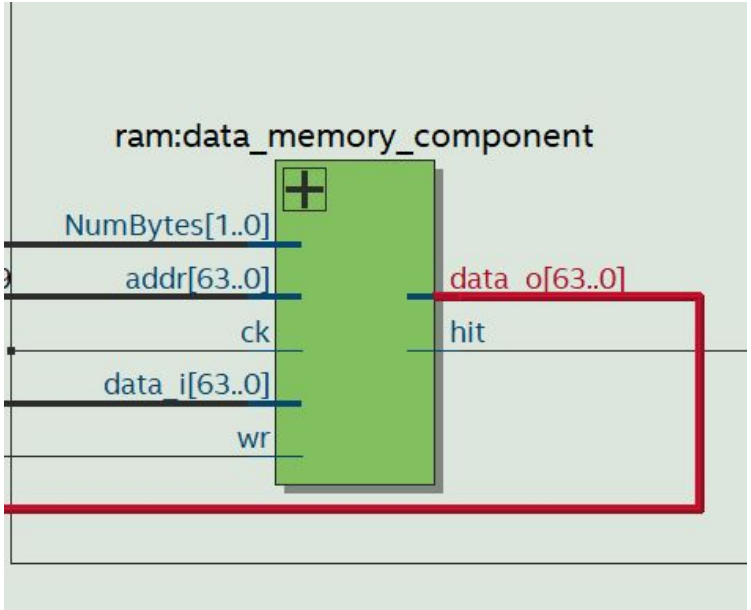
Load



- Recebe palavra de 64 bits da memória.

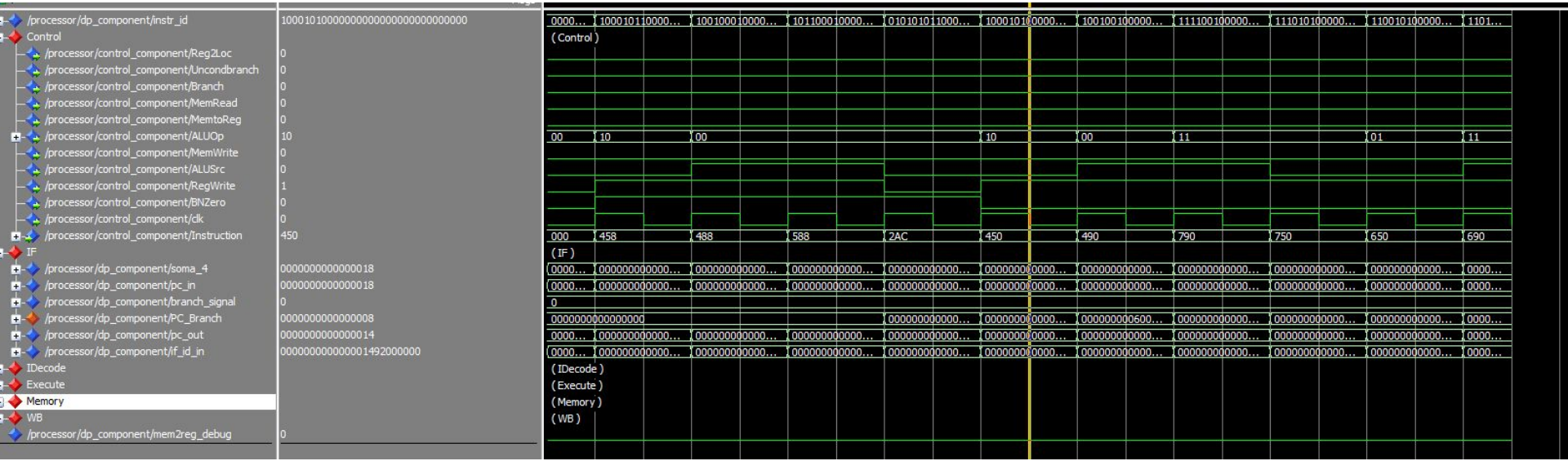
- Faz extensão de zeros a partir dos resultados dos multiplexadores.

Store



- Mudança na lógica do componente de memória.
- Codificação de NumBytes:
- 00: Escreve palavra de 64 bits
- 01: Escreve palavra de 32 bits
- 10: Escreve palavra de 16 bits
- 11: Escreve palavra de 08 bits

Testes


/processor/dp_component/instr_id
Control
/processor/control_component/Reg2Loc
/processor/control_component/Uncondbranch
/processor/control_component/Branch
/processor/control_component/MemRead
/processor/control_component/MemtoReg
/processor/control_component/ALUOp
/processor/control_component/MemWrite
/processor/control_component/ALUSrc
/processor/control_component/RegWrite
/processor/control_component/BNZero
/processor/control_component/clk
/processor/control_component/Instruction
IF
/processor/dp_component/soma_4
/processor/dp_component/pc_in
/processor/dp_component/branch_signal
/processor/dp_component/PC_Branch
/processor/dp_component/pc_out
/processor/dp_component/if_id_in
IDecode
Execute
Memory
WB
/processor/dp_component/mem2reg_debug
1000101000000000000000000000000
0
0
0
0
0
10
0
0
1
0
0
450
0000000000000018
0000000000000018
0
0000000000000008
0000000000000014
00000000000001492000000
0
(Control)
00
10
00
10
00
11
01
11
000
458
488
588
2AC
450
490
790
750
650
690
(IF)
(IDecode)
(Execute)
(Memory)
(WB)

Testes

```
when "111110" =>
    report "Entrei no 111110";
    if (Instruction(22) = '1') then
        --AluOp 00
        -- Load Register Unscaled offset == 11111000010
        report "load";
        Reg2Loc      <= '0';
        Uncondbranch <= '0';
        Branch       <= '0';
        MemRead      <= '1';
        MemtoReg     <= '1';
        ALUOp        <= "00";
        MemWrite     <= '0';
        ALUSrc       <= '1';
        RegWrite     <= '1';
        bcond        <= '0';
        setflags     <= '0';
        bregister    <= '0';
        blink        <= '0';
        zeroext0     <= '0';
        zeroext1     <= '0';
        zeroext2     <= '0';
        exclusive    <= '0';
        numBytes     <= "00";
        fp           <= '0';
```

Uso das cláusulas <<report>> e <<assert>>

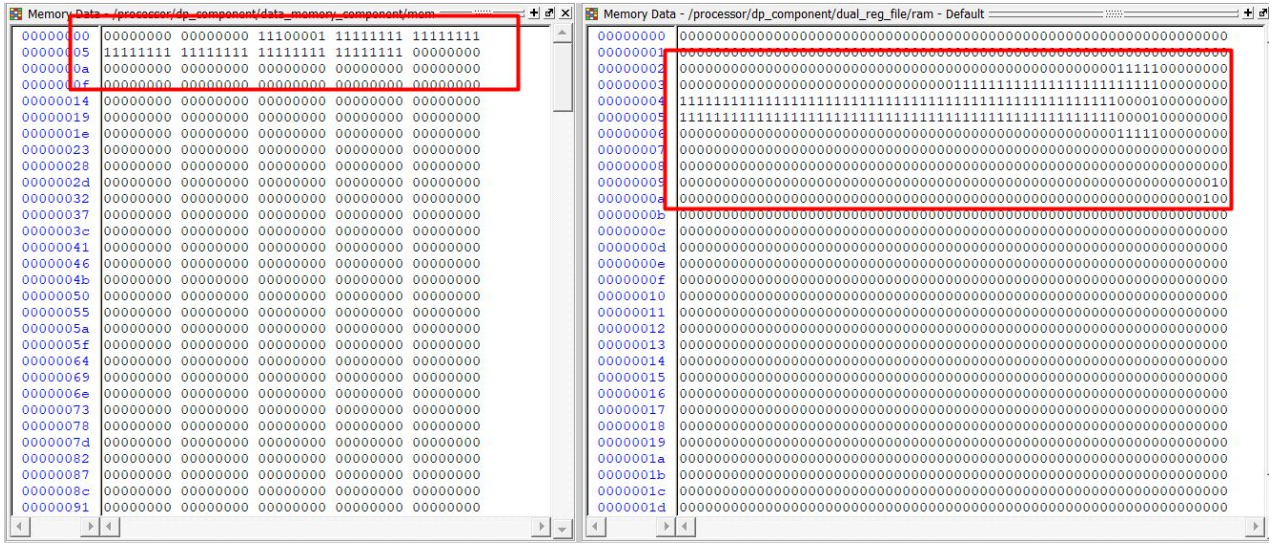
```
Instruction <= '10111000100'; -- load word unscaled register half
run 10 ns;
assert Reg2Loc = '0'       report "falha em LDUR: Reg2Loc inesperado"      severity error;
assert Uncondbranch = '0'  report "falha em LDUR: Uncondbranch inesperado" severity error;
assert Branch = '0'        report "falha em LDUR: Branch inesperado"       severity error;
assert MemRead = '1'       report "falha em LDUR: MemRead inesperado"      severity error;
assert MemtoReg = '1'      report "falha em LDUR: MemtoReg inesperado"     severity error;
assert ALUOp = "00"        report "falha em LDUR: ALUOp inesperado"        severity error;
assert MemWrite = '0'      report "falha em LDUR: MemWrite inesperado"     severity error;
assert ALUSrc = '1'        report "falha em LDUR: ALUSrc inesperado"       severity error;
assert RegWrite = '1'      report "falha em LDUR: RegWrite inesperado"     severity error;
assert bcond = '0'         report "falha em LDUR: bcond inesperado"        severity error;
assert setflags = '0'      report "falha em LDUR: setflags inesperado"     severity error;
assert bregister = '0'     report "falha em LDUR: bregister inesperado"    severity error;
assert blink = '0'         report "falha em LDUR: blink inesperado"        severity error;
assert zeroext0 = '0'      report "falha em LDUR: zeroext0 inesperado"     severity error;
assert zeroext1 = '0'      report "falha em LDUR: zeroext1 inesperado"     severity error;
assert zeroext2 = '1'      report "falha em LDUR: zeroext2 inesperado"     severity error;
assert exclusive = '0'     report "falha em LDUR: exclusive inesperado"    severity error;
assert numBytes = "01"     report "falha em LDUR: numBytes inesperado"     severity error;
```

Testes

Pequenos programas carregados na ROM para execução

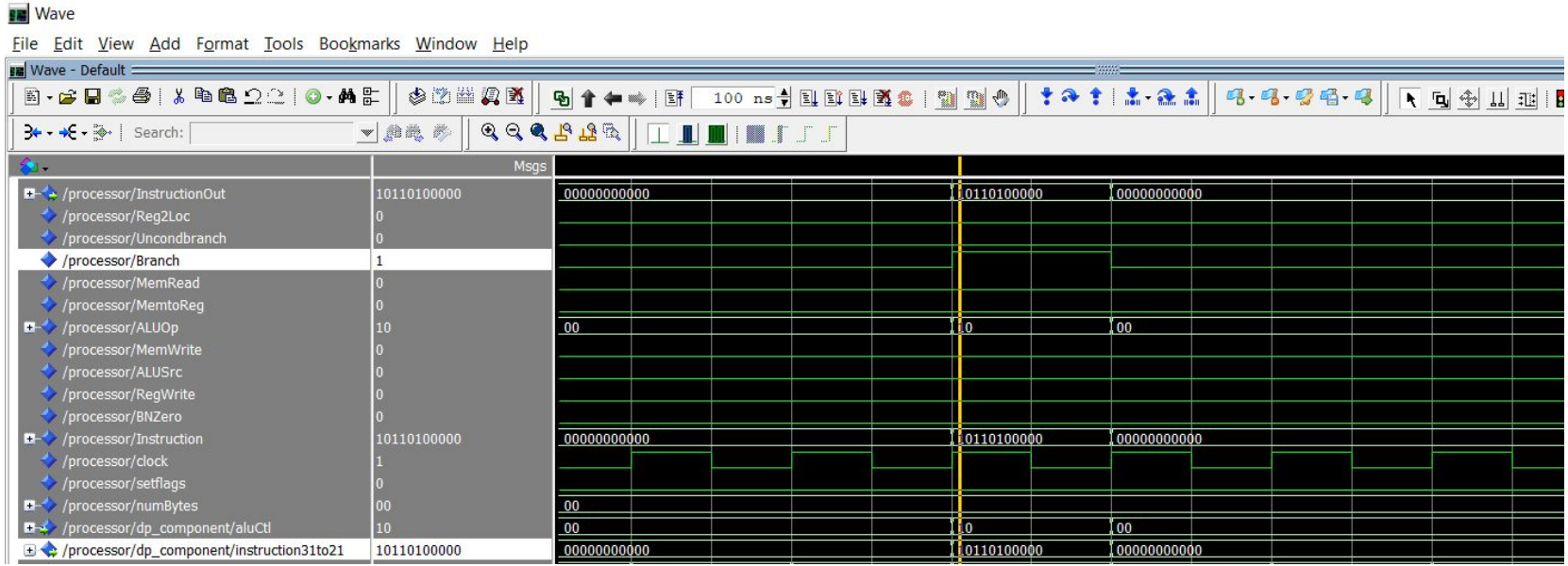
```
21  type rom_data is array (0 to 70) of bit_vector ( wordSize - 1 downto 0 );
22  constant rom : rom_data := (
23      x"F8400142",  --   LDUR r2, [r10]         1   Carrega em r2 o conteúdo da memória na posição dada por r10
24      x"00000000",
25      x"00000000",
26      x"00000000",
27      x"00000000",
28      x"F8401143",  --   LDUR r3, [r10, #1]     2   Carrega em r3 o conteúdo da memória na posição dada por (r10+1)
29      x"00000000",
30      x"00000000",
31      x"00000000",
32      x"00000000",
33      x"CB020064",  --   SUB r4, r3, r2         3   Subtrai r2 de r3, e armazena o conteúdo em r4
34      x"00000000",
35      x"00000000",
36      x"00000000",
37      x"00000000",
38      x"8B020065",  --   ADD r5, r3, r2         4   Soma r3 e r2, e armazena o conteúdo em r5
39      x"00000000",
40      x"00000000",
41      x"00000000",
42      x"00000000",
43      x"B4000041",  --   CBZ r1, #2             5   Se o conteúdo de r1 é zero, tomar desvio para PC + 2
44      x"00000000",
45      x"00000000",
46      x"00000000",
```

Testes



Carregamento de diferentes valores de memória para verificar comportamentos com diferentes entradas

Testes



Avaliação de resultados das por análise das waves