Universidade Federal de Pernambuco
(UFPE)
Centro de Informática (CIn)

Relatório de Projeto

Aluno:
Gabriel Soares (gss12)

Data de Entrega: 21 de fevereiro de 2025

Relatório de Projeto: Implementação de um Processador RISC-V em Pipeline

22 de fevereiro de 2025

Sumário

1 Introdução

Este relatório descreve o desenvolvimento e validação de uma implementação em pipeline de um processador baseado no conjunto de instruções RISC-V. O objetivo do projeto foi projetar e testar um processador funcional que executa um subconjunto de instruções RISC-V, garantindo sua correta operação por meio de simulações no ModelSim.

2 Metodologia

A implementação foi realizada utilizando a linguagem de descrição de hardware **SystemVerilog**, com suporte a pipeline para execução eficiente das instruções. O projeto foi validado por meio de simulações no ModelSim, garantindo que as instruções implementadas fossem corretamente executadas.

3 Implementações

A implementação do processador incluiu suporte a diferentes tipos de instruções, categorizadas conforme feito no repositório do GitHub.

3.1 Instruções Registrador - Registrador

Para implementar as funcionalidades dessa categoria, foi necessário ajustar o controlador da ALU para considerar as partes do OPcode específicas de cada instrução, como funct3 e funct7, garantindo seu correto tratamento na ALU. Neste componente, a execução da operação correspondente à instrução é realizada com base no sinal gerado pelo controlador.

- SUB
  Considei o caso em que a operação é (0110) e definimos que a saída da ALU seria ALUResult = SrcA - SrcB.

- SLT
  Considerei o caso em que a operação é (1100) e dividimos a saída em três situações:

  Quando o primeiro registrador possui um valor negativo e o segundo um valor positivo, o primeiro sempre será menor, portanto ALUResult = 1; Quando o primeiro registrador tem um valor positivo e o segundo um valor negativo, o primeiro nunca será menor que o segundo, logo ALUResult = 0; Para qualquer outra combinação diferente das duas anteriores, utilizei um operador ternário para verificar se ($SrcA < SrcB$) e atribui o resultado a ALUResult.

- XOR
  Considerei o caso em que a operação é (0101) e definimos que a saída será $SrcA \oplus SrcB$.

- OR
  Considerei o caso em que a operação é (0001) e estabelecemos que a saída será ALUResult = $SrcA \vee SrcB$.

3.2 Instruções Registrador - Imediato

Para implementar a instrução que requer um imediato, modifiquei o Gerador de Imediato para reconhecer a instrução I-Type e o controlador para permitir o uso da ALU e da escrita do Registrador. Adicionei na ALU um novo caso para a instrução. No ALUController, usei o código da operação, a Funct3 e Funct7 de cada instrução para encaminhar para o caso da ALU correspondente à instrução. O registrador da instrução fica no fio "SrcA"e o imediato fica no fio "SrcB".

- ADDI
  Considerei o caso em que a operação é (0010) e defini que a saída da ALU será ALUResult = SrcA + SrcB.

- SLTI
  Considerei o caso em que a operação é (1100) e dividi a saída em três situações:

  Quando o registrador possui um valor negativo e o imediato um valor positivo, o registrador será sempre menor, portanto, ALUResult = 1; Quando o registrador tem um valor positivo e o imediato é negativo, o registrador nunca será menor, logo, ALUResult = 0; Para qualquer outra configuração diferente das anteriores, utilizei um operador ternário que verifica se ($SrcA < SrcB$) e atribui o resultado a ALUResult.

- SLLI
  Considerei o caso em que a operação é (0111) e utilizei o operador de shift lógico à esquerda do SystemVerilog, atribuindo o resultado a ALUResult na ALU.

- SRLI
  Considerei o caso em que a operação é (1111) e apliquei o operador de shift lógico à direita do SystemVerilog, atribuindo o resultado a ALUResult na ALU.

- SRAI
  Considerei o caso em que a operação é (1110) e empreguei o operador de shift aritmético à direita do SystemVerilog, atribuindo o resultado a ALUResult na ALU.

- LUI
  Considerei o caso em que a operação é (1011), retornei o valor de SrcB, defini 0 nos últimos 12 bits e atribui o resultado a ALUResult na ALU.

3.3 Instruções de Transferência de Controle

Para implementar as funções de desvio incondicional, o controller foi modificado para reconhecer as instruções correspondentes e enviar dois sinais ao data path: um indicando que se trata de um desvio incondicional e outro determinando que o resultado da busca nos registradores deve ser utilizado.

O sinal é processado na segunda etapa do pipeline e armazenado no banco B de registradores. Na terceira etapa, foi adicionado um multiplexador (mux) que seleciona o valor de sourceA com base nesse sinal. Esse valor, que será armazenado no registrador, pode ser tanto o conteúdo do registrador quanto o próprio PC + 4.

Além disso, a Branch Unit foi ajustada para considerar os valores obtidos dos registradores e os sinais das instruções do tipo J. No caso de um JALR, o próximo valor do PC é atualizado para o valor armazenado no registrador somado ao imediato (Imm).

- JAL
  Quando o sinal jal está ativo, o valor de PC + 4 é armazenado no registrador, e o próximo PC é atualizado para PC + Imm.

- JALR
  Quando os sinais jal e jalr estão ativos, o valor de PC + 4 é armazenado no registrador, e o próximo PC é atualizado para FAmuxResult + Imm.

- BNE
  Considerei o caso em que a operação é (1001) e utilizei o operador lógico != do SystemVerilog para verificar se os valores armazenados nos registradores são diferentes.

- BLT
  Usei o mesmo caso de operação para SLT e SLTI, uma vez que suas operações na ALU são equivalentes.

- BGE
  Considerei o caso em que a operação é (1010) e dividi a saída em três situações, de forma semelhante ao que foi feito para verificar se o valor de SrcA é menor que SrcB:

  Quando o primeiro registrador tem um valor negativo e o segundo registrador tem um valor positivo, o primeiro registrador nunca será maior que o segundo, portanto, ALUResult = 0; Quando o primeiro registrador tem um valor positivo e o segundo registrador possui um valor negativo, o primeiro registrador sempre será maior que o segundo, logo, ALUResult = 1; Para qualquer outra configuração diferente das anteriores, utilizamos um operador ternário que verifica se ($SrcA >= SrcB$) e atribui o resultado a ALUResult.

3.4 Instruções de Load e Store

As funções de load envolvem a captura de diferentes números de bits. A escolha de implementação foi capturar a palavra completa (32 bits) da memória e resetar os bits que não são necessários, estendendo o bit de sinal (exceto no caso de lbu, que desconsidera o sinal). Dessa forma, apenas os bits necessários são escritos.

- LB
  Leva em consideração o sinal e o byte menos significativo.

- LH
  Leva em consideração o sinal e os 2 bytes menos significativos.

- LBU
  Não leva em consideração o sinal e pega o byte menos significativo.

Para implementar os stores, alteramos o sinal "Wr"enviado para determinar qual byte será escrito na memória. No caso de uma operação half, desativamos o sinal de escrita para os dois bytes mais significativos, de forma que apenas os dois bytes menos significativos sejam gravados.

- SB
  Grava o byte menos significativo da palavra fornecida na memória.

- SH
  Grava os 2 bytes menos significativo da palavra fornecida na memória.

3.5 Pseudo-Instrução

- HALT
  O halt foi implementado por meio de um novo módulo chamado HaltUnit. Esse módulo é responsável por controlar o opcode enviado ao controller, sendo que o opcode para o halt foi definido como 0000001. Quando um halt é detectado, a HaltUnit começa a transformar todos os opcodes enviados para o controller em 0000000, que não possui nenhuma função. Como o assembler não estava reconhecendo o halt, testei a modificação diretamente no arquivo instruction.mif.

4 Simulações e Validações

As simulações foram realizadas no **ModelSim**, onde a execução das instruções foi analisada para verificar a correta operação do pipeline. Foram validadas todas as categorias de instruções, conforme detalhado abaixo:

4.1 Simulações de Referência

- ADD, OR, ADDI, SLL, SRL, SRA, SLT, SLTI, SLTIU, SLLI, SRLI, SRAI, XORI, ORI, ANDI, XOR

```
addi x0,x0,0
addi x1,x0,8
addi x2,x0,4
or x3,x1,x2
or x4,x2,x0
add x6,x4,x2
addi x4,x0,2
addi x5,x0,-2
sll x18,x1,x4
srl x19,x5,x4
sra x20,x5,x4
slt x21,x1,x2
slt x22,x2,x1
sltu x23,x5,x1
sltu x24,x1,x5
slti x25,x1,8
slti x26,x1,16
addi x5,x0,-4
sltiu x27,x1,-2
sltiu x28,x5,-2
slli x29,x5,1
srli x30,x5,1
srai x31,x5,1
xori x6,x1,10
ori x7,x1,2
andi x8,x1,10
xor x9,x1,x2
```

Figura 1: Fornecido

```
45: Register [ 0] written with value: [00000000] | [         0]
55: Register [ 1] written with value: [00000008] | [         8]
65: Register [ 2] written with value: [00000004] | [         4]
75: Register [ 3] written with value: [0000000c] | [        12]
85: Register [ 4] written with value: [00000004] | [         4]
95: Register [ 6] written with value: [00000008] | [         8]
105: Register [ 4] written with value: [00000002] | [         2]
115: Register [ 5] written with value: [fffffffe] | [4294967294]
125: Register [18] written with value: [00000020] | [        32]
135: Register [19] written with value: [3fffffff] | [1073741823]
145: Register [20] written with value: [ffffffff] | [4294967295]
155: Register [21] written with value: [00000000] | [         0]
165: Register [22] written with value: [00000001] | [         1]
175: Register [23] written with value: [00000000] | [         0]
185: Register [24] written with value: [00000001] | [         1]
195: Register [25] written with value: [00000000] | [         0]
205: Register [26] written with value: [00000001] | [         1]
215: Register [ 5] written with value: [fffffffc] | [4294967292]
225: Register [27] written with value: [00000001] | [         1]
235: Register [28] written with value: [00000001] | [         1]
245: Register [29] written with value: [fffffff8] | [4294967288]
255: Register [30] written with value: [7ffffffe] | [2147483646]
265: Register [31] written with value: [fffffffe] | [4294967294]
275: Register [ 6] written with value: [00000002] | [         2]
285: Register [ 7] written with value: [0000000a] | [        10]
295: Register [ 8] written with value: [00000008] | [         8]
305: Register [ 9] written with value: [0000000c] | [        12]
```

Figura 2: Esperado

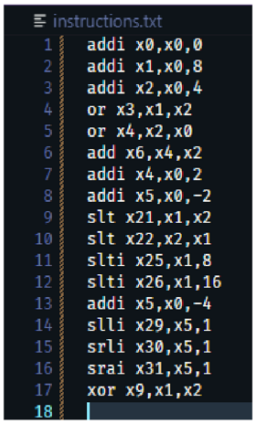

Figura 3: Depois de tirar as instruções que não tem descrição

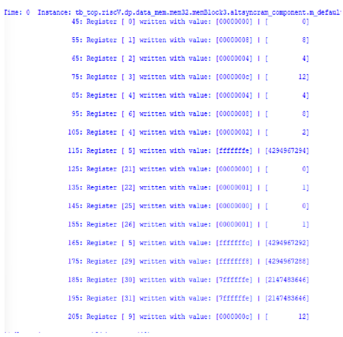

Figura 4: Resultado

- SUB, AND, LUI

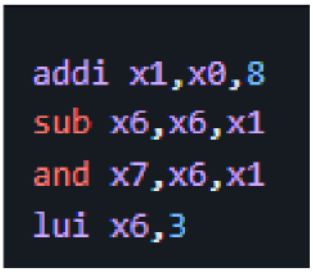

Figura 5: Fornecido

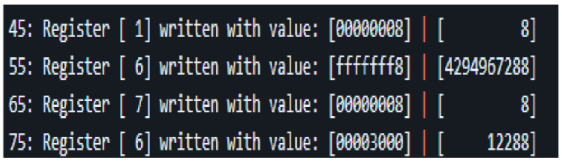

Figura 6: Esperado

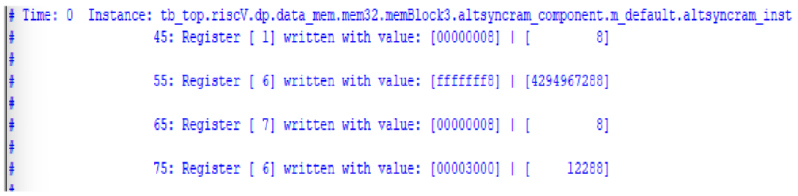

Figura 7: Obtido

4.2 Simulações de LOAD

- LB, LH, LW

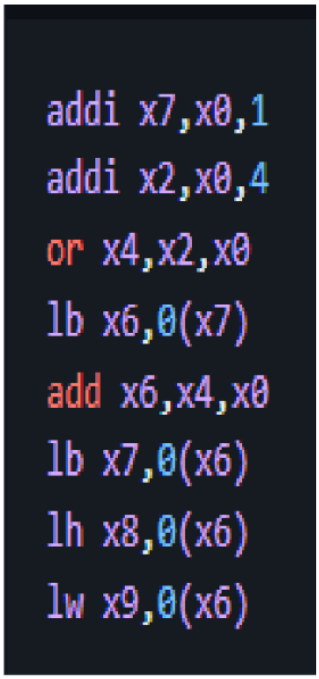

Figura 8: Fornecido

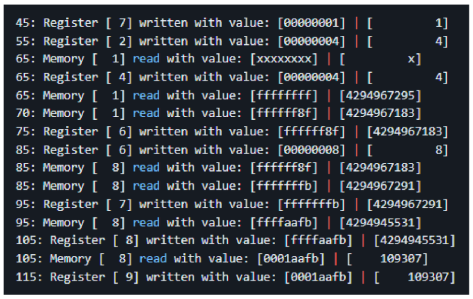

Figura 9: Esperado

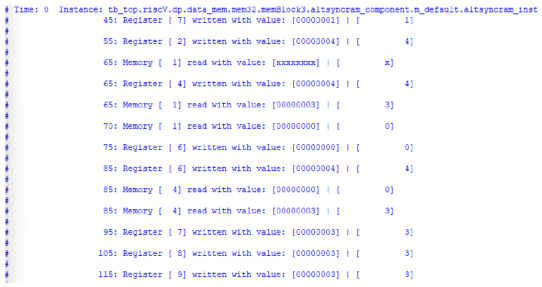

Figura 10: Obtido

- LBU, LHU

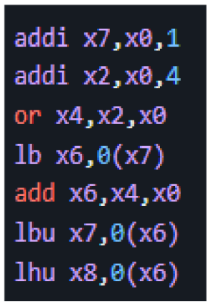

Figura 11: Fornecido

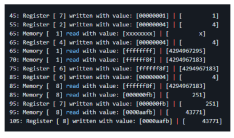

Figura 12: Esperado

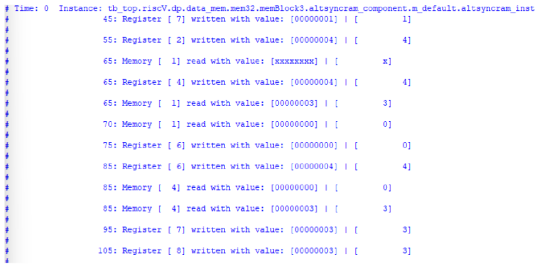

Figura 13: Obtido

4.3 Simulações de Branch

- JAL,BEQ (TAKEN)

```
addi x7,x0,1
addi x2,x0,4
jal x10,8
or x4,x2,x0
add x6,x4,x2
addi x7,x0,1
addi x8,x0,2
beq x7,x7,-8
or x4,x2,x0
```

Figura 14: Fornecido

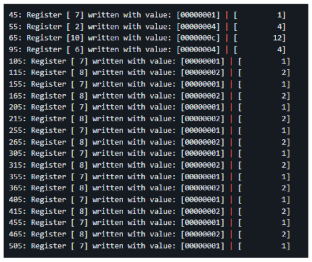

Figura 15: Esperado

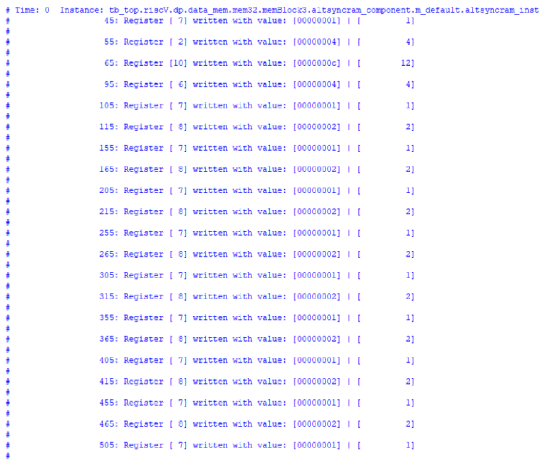

Figura 16: Obtido

- BEQ (NOT TAKEN)

```
addi x7,x0,1
addi x2,x0,4
jal x10,8
or x4,x2,x0
add x6,x4,x2
addi x7,x0,1
addi x8,x0,2
beq x8,x7,-8
or x4,x2,x0
```

Figura 17: Fornecido

```
45: Register [ 7] written with value: [00000001] | [        1]
55: Register [ 2] written with value: [00000004] | [        4]
65: Register [10] written with value: [0000000c] | [       12]
95: Register [ 6] written with value: [00000004] | [        4]
105: Register [ 7] written with value: [00000001] | [        1]
115: Register [ 8] written with value: [00000002] | [        2]
135: Register [ 4] written with value: [00000004] | [        4]
```

Figura 18: Esperado

```
# Time: 0  Instance: tb_top.riscV.dp.data_mem.mem32.memBlock3.altsyncram_component.m_default.altsyncram_inst
#                  45: Register [ 7] written with value: [00000001] | [        1]
#
#                  55: Register [ 2] written with value: [00000004] | [        4]
#
#                  65: Register [10] written with value: [0000000c] | [       12]
#
#                  95: Register [ 6] written with value: [00000004] | [        4]
#
#                 105: Register [ 7] written with value: [00000001] | [        1]
#
#                 115: Register [ 8] written with value: [00000002] | [        2]
#
#                 135: Register [ 4] written with value: [00000004] | [        4]
#
```

Figura 19: Obtido

- BNE (TAKEN)

```
addi x7,x0,1
addi x2,x0,4
jal x10,8
or x4,x2,x0
add x6,x4,x2
addi x7,x0,1
addi x8,x0,2
bne x8,x7,-8
or x4,x2,x0
```

Figura 20: Fornecido

```
45: Register [ 7] written with value: [00000001] | [        1]
55: Register [ 2] written with value: [00000004] | [        4]
65: Register [10] written with value: [0000000c] | [       12]
95: Register [ 6] written with value: [00000004] | [        4]
105: Register [ 7] written with value: [00000001] | [        1]
115: Register [ 8] written with value: [00000002] | [        2]
155: Register [ 7] written with value: [00000001] | [        1]
165: Register [ 8] written with value: [00000002] | [        2]
205: Register [ 7] written with value: [00000001] | [        1]
215: Register [ 8] written with value: [00000002] | [        2]
255: Register [ 7] written with value: [00000001] | [        1]
265: Register [ 8] written with value: [00000002] | [        2]
305: Register [ 7] written with value: [00000001] | [        1]
315: Register [ 8] written with value: [00000002] | [        2]
355: Register [ 7] written with value: [00000001] | [        1]
365: Register [ 8] written with value: [00000002] | [        2]
405: Register [ 7] written with value: [00000001] | [        1]
415: Register [ 8] written with value: [00000002] | [        2]
455: Register [ 7] written with value: [00000001] | [        1]
465: Register [ 8] written with value: [00000002] | [        2]
505: Register [ 7] written with value: [00000001] | [        1]
```

Figura 21: Esperado

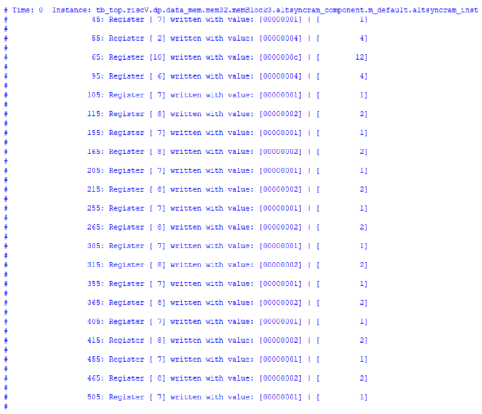

Figura 22: Obtido

- BLT (TAKEN)

```
addi x7,x0,1
addi x2,x0,4
jal x10,8
or x4,x2,x0
add x6,x4,x2
addi x7,x0,2
addi x8,x0,1
blt x8,x7,-8
or x4,x2,x0
```

Figura 23: Fornecido

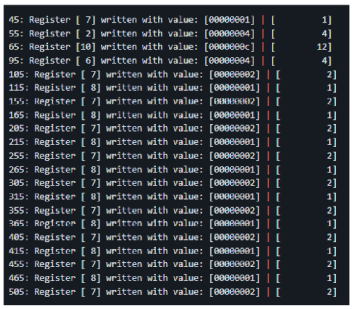

Figura 24: Esperado

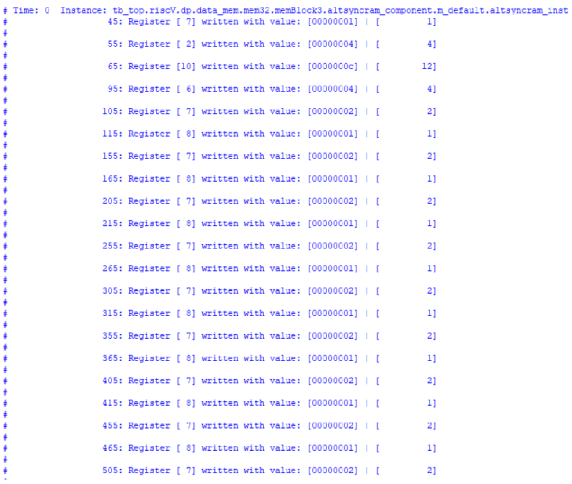

Figura 25: Obtido

- BGE (TAKEN)

```
addi x7,x0,1
addi x2,x0,4
jal x10,8
or x4,x2,x0
add x6,x4,x2
addi x7,x0,2
addi x8,x0,1
bge x7,x8,-8
or x4,x2,x0
```

Figura 26: Fornecido

```
45: Register [ 7] written with value: [00000001] | [         1]
55: Register [ 2] written with value: [00000004] | [         4]
65: Register [10] written with value: [0000000c] | [        12]
95: Register [ 6] written with value: [00000004] | [         4]
105: Register [ 7] written with value: [00000002] | [         2]
115: Register [ 8] written with value: [00000001] | [         1]
155: Register [ 7] written with value: [00000002] | [         2]
165: Register [ 8] written with value: [00000001] | [         1]
205: Register [ 7] written with value: [00000002] | [         2]
215: Register [ 8] written with value: [00000001] | [         1]
255: Register [ 7] written with value: [00000002] | [         2]
265: Register [ 8] written with value: [00000001] | [         1]
305: Register [ 7] written with value: [00000002] | [         2]
315: Register [ 8] written with value: [00000001] | [         1]
355: Register [ 7] written with value: [00000002] | [         2]
365: Register [ 8] written with value: [00000001] | [         1]
405: Register [ 7] written with value: [00000002] | [         2]
415: Register [ 8] written with value: [00000001] | [         1]
455: Register [ 7] written with value: [00000002] | [         2]
465: Register [ 8] written with value: [00000001] | [         1]
505: Register [ 7] written with value: [00000002] | [         2]
```

Figura 27: Esperado

```
# Time: 0  Instance: tb_top.riscV.dp.data_mem.mem32.memBlock3.altsyncram_component.m_default.altsyncram_inst
#                  45: Register [ 7] written with value: [00000001] | [         1]
#
#                  55: Register [ 2] written with value: [00000004] | [         4]
#
#                  65: Register [10] written with value: [0000000c] | [        12]
#
#                  95: Register [ 6] written with value: [00000004] | [         4]
#
#                 105: Register [ 7] written with value: [00000002] | [         2]
#
#                 115: Register [ 8] written with value: [00000001] | [         1]
#
#                 155: Register [ 7] written with value: [00000002] | [         2]
#
#                 165: Register [ 8] written with value: [00000001] | [         1]
#
#                 205: Register [ 7] written with value: [00000002] | [         2]
#
#                 215: Register [ 8] written with value: [00000001] | [         1]
#
#                 255: Register [ 7] written with value: [00000002] | [         2]
#
#                 265: Register [ 8] written with value: [00000001] | [         1]
#
#                 305: Register [ 7] written with value: [00000002] | [         2]
#
#                 315: Register [ 8] written with value: [00000001] | [         1]
#
#                 355: Register [ 7] written with value: [00000002] | [         2]
#
#                 365: Register [ 8] written with value: [00000001] | [         1]
#
#                 405: Register [ 7] written with value: [00000002] | [         2]
#
#                 415: Register [ 8] written with value: [00000001] | [         1]
#
#                 455: Register [ 7] written with value: [00000002] | [         2]
#
#                 465: Register [ 8] written with value: [00000001] | [         1]
#
#                 505: Register [ 7] written with value: [00000002] | [         2]
```

Figura 28: Obtido

- JALR

```
addi x7,x0,-1
sw x7,0(x0)
lw x9,0(x0)
or x4,x2,x0
add x6,x4,x2
jalr x12,x0,12
```

Figura 29: Fornecido

```
45: Memory [  0] written with value: [ffffffff] | [4294967295]
45: Register [ 7] written with value: [ffffffff] | [4294967295]
55: Memory [  0] read with value: [xxxxxxxx] | [         x]
55: Memory [  0] read with value: [00000000] | [         0]
60: Memory [  0] read with value: [ffffffff] | [4294967295]
65: Register [ 9] written with value: [ffffffff] | [4294967295]
75: Register [ 4] written with value: [00000000] | [         0]
85: Register [ 6] written with value: [00000000] | [         0]
95: Register [12] written with value: [00000018] | [        24]
125: Register [ 4] written with value: [00000000] | [         0]
135: Register [ 6] written with value: [00000000] | [         0]
145: Register [12] written with value: [00000018] | [        24]
175: Register [ 4] written with value: [00000000] | [         0]
185: Register [ 6] written with value: [00000000] | [         0]
195: Register [12] written with value: [00000018] | [        24]
225: Register [ 4] written with value: [00000000] | [         0]
235: Register [ 6] written with value: [00000000] | [         0]
245: Register [12] written with value: [00000018] | [        24]
275: Register [ 4] written with value: [00000000] | [         0]
285: Register [ 6] written with value: [00000000] | [         0]
295: Register [12] written with value: [00000018] | [        24]
325: Register [ 4] written with value: [00000000] | [         0]
335: Register [ 6] written with value: [00000000] | [         0]
345: Register [12] written with value: [00000018] | [        24]
375: Register [ 4] written with value: [00000000] | [         0]
385: Register [ 6] written with value: [00000000] | [         0]
395: Register [12] written with value: [00000018] | [        24]
425: Register [ 4] written with value: [00000000] | [         0]
435: Register [ 6] written with value: [00000000] | [         0]
445: Register [12] written with value: [00000018] | [        24]
475: Register [ 4] written with value: [00000000] | [         0]
485: Register [ 6] written with value: [00000000] | [         0]
495: Register [12] written with value: [00000018] | [        24]
```

Figura 30: Esperado

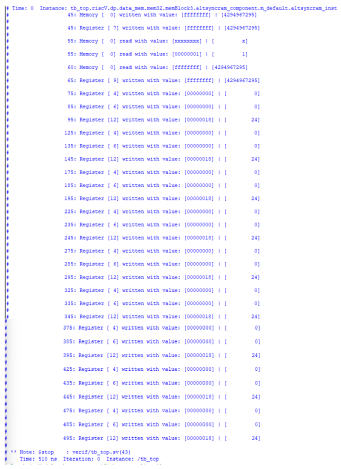

Figura 31: Obtido

4.4 Simulações de STORE

- SW

```
addi x7,x0,-1
sw x7,0(x0)
lw x9,0(x0)
```

Figura 32: Fornecido

```
45: Memory [  0] written with value: [ffffffff] | [4294967295]
45: Register [ 7] written with value: [ffffffff] | [4294967295]
55: Memory [  0] read with value: [xxxxxxxx] | [          x]
55: Memory [  0] read with value: [ffffaa80] | [4294945408]
60: Memory [  0] read with value: [ffffffff] | [4294967295]
65: Register [ 9] written with value: [ffffffff] | [4294967295]
```

Figura 33: Esperado

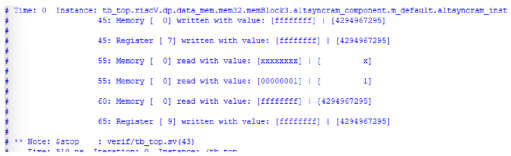

Figura 34: Obtido

- SB, SH

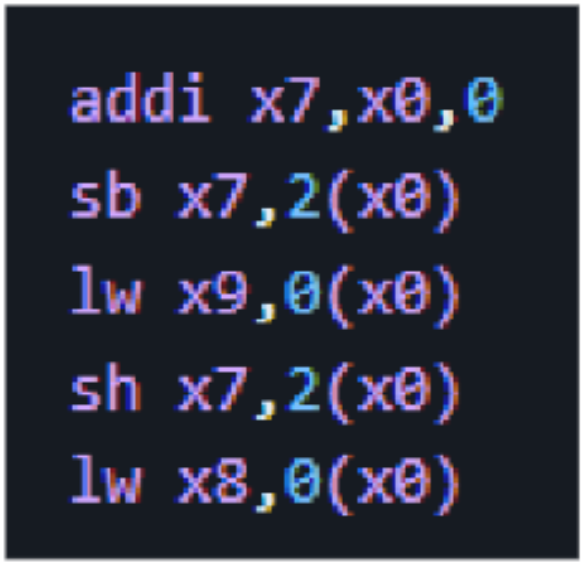

Figura 35: Fornecido

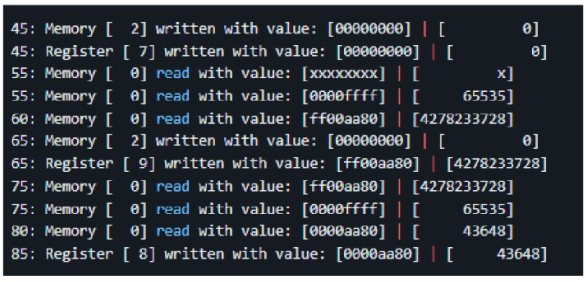

Figura 36: Esperado

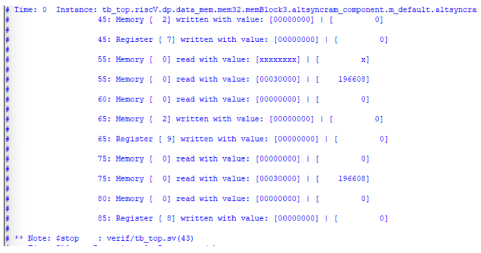

Figura 37: Obtido

4.5 HALT

O comportamento da pseudo-instrução **HALT** foi validado, garantindo que o processador interrompesse sua execução corretamente.

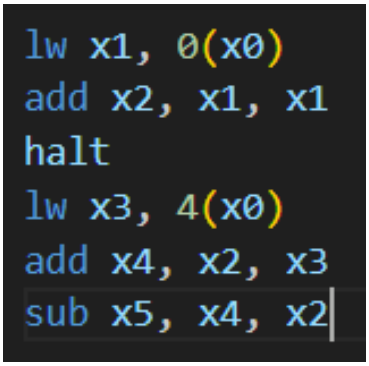

Figura 38: Utilizado

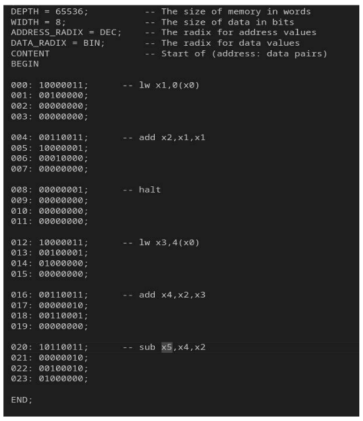

Figura 39: .MIF

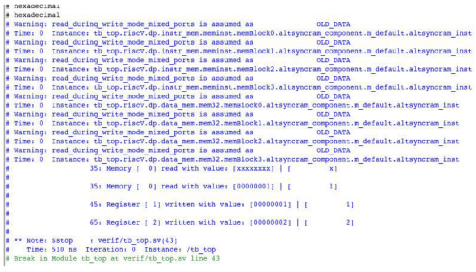

Figura 40: Resultdo

5 Conclusão

A implementação de um processador RISC-V em pipeline foi concluída com sucesso, validando sua operação por meio do **ModelSim**. O projeto permitiu compreender em detalhes o funcionamento do pipeline e os desafios associados à sua implementação, como dependências de dados e controle de fluxo. Futuras melhorias podem incluir otimizações no pipeline e a adição de novas instruções para ampliar as funcionalidades do processador.