Universidade Federal de Pernambuco

Centro de Informática

Relatório do Projeto RISC-V Pipeline

Gabriel Alves Gadelha Melo - gagm
Lucas Emanuel Sabino de Souza Lima - lessl
Mayres F. da Silva Santos - mfss2

Data de entrega: 08/08/2024

Sumário

1 - Introdução

1.1 - Objetivos

Tendo já implementadas previamente as instruções BEQ, LW, SW, ADD e AND, o objetivo deste projeto é implementar as seguintes instruções RISC-V:

- LB (Load Byte)
- LH (Load Halfword)
- LBU (Load Byte Unsigned)
- SB (Store Byte)
- SH (Store Halfword)
- OR (OR)
- XOR (Exclusive OR)
- SUB (Subtract)
- ADDI (Add Immediate)
- SLLI (Shift Left Logical Immediate)
- SRLI (Shift Right Logical Immediate)
- SRAI (Shift Right Arithmetic Immediate)
- SLTI (Set Less Than Immediate)
- SLT (Set Less Than)
- LUI (Load Upper Immediate)
- BNE (Branch if Not Equal)
- BLT (Branch if Less Than)
- BGE (Branch if Greater or Equal)
- JAL (Jump and Link)
- JALR (Jump and Link Register)
- HALT (Stop PC)

1.2 - Visão Geral

Com o uso da arquitetura RISC-V, implementamos cada instrução, as quais foram categorizadas de acordo com o seu formato (I-type, R-type, B-type, U-type, J-type). A implementação

incluiu a definição dos sinais de controle necessários para a correta execução de cada instrução e a simulação dessas instruções para garantir sua funcionalidade.

2 - Descrição da Implementação das classes de operações

2.1 - I-type

As instruções do tipo I incluem LB, LH, LBU, SLTI, ADDI, SLLI, SRLI, SRAI, JALR. Estas instruções utilizam um operando imediato. A implementação envolve a decodificação do operando imediato e a execução da operação específica.

- Sinais de controle
    - RegWrite
    - MemtoReg (Load)
    - ALUsrc
    - MemRead (Load)
    - ALUOp (Load / Aritméticas)
    - JalrSel (Jalr)
    - RWSel (Jalr)
- Implementação

  Os sinais de controle acima são setados no Controller.sv de acordo com o opcode das instruções, que posteriormente são encaminhados para o controlador da ALU para coordenar a realização de suas operações correspondentes, de modo que: LOADs e JALR realizam apenas a operação de adição de seu imediato com o endereço do registrador ‘r1’, enquanto as instruções aritméticas são encaminhadas para a operação específica que cada uma realiza, sendo mapeadas por um número de 4 bits que seleciona a operação na ALU.
    - **LOADs:** Após a etapa da ALU, o endereço resultante da operação, o valor do registrador e o endereço calculado são usados para acessar a posição correta de memória no datamemoty.sv e, de acordo com o número de bits especificados pela instrução (LB = 8 bits, LH = 16 bits), esse intervalo de bits é carregado da memória para o registrador.
    - **JALR:** Após a etapa da ALU, o ‘rd’ salva o endereço do PC+4 com o auxílio do mux4 ‘wrsmux’, que usa o RWSel (existe no png do pipeline mas não existia no código) como seletor de 2 bits e usa o JalrSel (existe no png do pipeline mas não existia no código) na BranchUnit.sv para coordenar a tomada do branch para o PC = Endereço calculado.
    - **Aritméticas com imediato:** Após o cálculo da ALU, não ocorre acesso à memória e o valor WrmuxSrc é selecionado do mux (RWSel = 00) para ser escrito de volta no registrador.

2.2 - S-type

As instruções do tipo S incluem SB e SH. Estas instruções guardam valores na memória.

- Sinais de controle
    - ALUSrc
    - MemWrite
    - ALUOp
- Implementação

  Os sinais de controle acima são setados no Controller.sv de acordo com o opcode das instruções, que posteriormente são encaminhados para o controlador da ALU para coordenar a realização de suas operações correspondentes, de modo que STOREs realizam apenas a operação de adição de seu imediato com o endereço do registrador ‘r1’. Após o endereço ser calculado ele é usado para acessar a posição da memória em datamemory.sv e escrever o

valor de ‘rd’ nessa posição de memória de acordo com a quantidade de bits especificada pela instrução (SB = 8 bits, SH = 16 bits).

2.3 - R-type

As instruções do tipo R incluem SUB, SLT, XOR, OR. Estas instruções utilizam dois operandos de registrador. A implementação requer a leitura dos registradores, a execução da operação aritmética ou lógica e a escrita do resultado de volta no registrador.

- Sinais de controle
    - RegWrite
    - ALUsrc
    - ALUOp
- Implementação

  Os sinais de controle acima são setados no Controller.sv de acordo com o opcode das instruções, que posteriormente são encaminhados para o controlador da ALU para coordenar a realização de suas operações correspondentes. Na ALU são realizadas as operações de acordo com cada instrução, usando um mapeamento com um número de 4 bits único para cada operação.

2.4 - B-type

As instruções do tipo B incluem BNE, BLT, BGE. Estas instruções são usadas para desvio condicional. A implementação envolve a comparação dos operandos e o cálculo do endereço de destino.

- Sinais de controle
    - Branch
    - PcSel
    - ALUOp
- Implementação

  Os sinais de controle acima são setados no Controller.sv de acordo com o opcode das instruções, que posteriormente são encaminhados para o controlador da ALU para coordenar a realização de suas operações correspondentes. Na ALU são realizadas as comparações correspondentes de cada instrução e de acordo com o ALUResult é feita a decisão de se vai haver branch ou não. Se houver branch, ocorre desvio para o endereço do PC + imm.

2.5 - U-type

As instruções do tipo U incluem LUI. Estas instruções utilizam um operando imediato para carregar um valor alto no registro. A implementação requer a decodificação do operando imediato e a atualização do registro de destino.

- Sinais de controle
    - RegWrite
    - ALUOp
    - RWSel
- Implementação

  Os sinais de controle acima são setados no Controller.sv de acordo com o opcode das instruções, que posteriormente são encaminhados para o controlador da ALU para coordenar a realização de suas operações correspondentes. A ALU retorna 1 para deixar a decisão do desvio apenas para o Branch e gera o imediato no imm_Gen.sv nos 20 bits mais significativos, para posteriormente, carregá-lo no ‘rd’ com RWSel = 10 na fase de write-back.

2.6 - J-type

As instruções do tipo J incluem JAL. Estas instruções são usadas para desvios incondicionais. A implementação envolve o cálculo do endereço de destino e a atualização do contador de programa (PC).

- Sinais de controle
    - RWSel
    - RegWrite
    - Branch
- Implementação
  Os sinais de controle acima são setados no Controller.sv de acordo com o opcode das instruções, que posteriormente são encaminhados para o controlador da ALU para coordenar a realização de suas operações correspondentes. A ALU retorna 1 para deixar a decisão do desvio apenas para o Branch, que também é 1 no JAL. Dessa forma, o branch é tomado para o PC + imm e, no Datapath.sv, o WrmuxSrc (RWSel = 01) seleciona o PC + 4 para ser salvo no 'rd'.

2.7 - HALT

A instrução HALT é usada para para a execução do programa. A implementação envolve a definição de um sinal de controle que interrompe o contador de programa.

- Sinais de controle
    - HaltSel
- Implementação
  Foi criado um seletor 'HaltSel' específico para a instrução de HALT, que detecta no Controller.sv se o opcode da instrução corresponde ao do HALT (foi adotado o opcode 7'b1111111). O HaltSel é passado a partir do registrador B do pipeline adiante e usado no BranchUnit.sv para parar de atualizar o PC se HaltSel = 1. Além disso, no Datapath.sv, se houver HALT, todos os sinais de controle e instruções anteriores são resetadas. Saindo dos módulos de SystemVerilog, foi necessário implementar a tradução do HALT no assembler.py, adicionando a nova instrução no dicionário INSTRUCTION e adicionando um condicional para tratar o "halt" na função 'translate_instruction'.

3 - Descrição das Simulações Realizadas

Obs.: Codar por classes de instruções
Obs.: Listar por classes as instruções do que tivemos que fazer

3.1 Metodologia das Simulações

Para validar a implementação das instruções, realizamos simulações utilizando um conjunto de testes para cada instrução. Sendo feito um código simples com os possíveis casos e vendo como o código se comportava dentro do ModelSIM, quando executava lá.

- Simulação LOAD
    - Valores menores que 8 bits.
        - Código

```
addi x2,x0,1
addi x3,x0,-8
addi x4,x0,4
sw x4,0(x2)
sw x3,8(x2)
lb x5,0(x2)
lh x6,0(x2)
lw x7,0(x2)
lbu x8,0(x2)
```

```
lb x9,8(x2)
lh x10,8(x2)
lw x11,8(x2)
lbu x12,8(x2)
```

- Explicação

  Põe valores menores que 8 bits na memória a partir do sw (que já estava implementado), positivo e negativo. Cada tipo de load é testado para os valores guardados na memória.

- Valores maiores que 16 bits
  - Código

```
addi x2,x0,1
addi x3,x0,-2001
addi x4,x0,2001
slli x3,x3,7
slli x4,x4,7
sw x4,0(x2)
sw x3,8(x2)
lb x5,0(x2)
lh x6,0(x2)
lw x7,0(x2)
lbu x8,0(x2)
lb x9,8(x2)
lh x10,8(x2)
lw x11,8(x2)
lbu x12,8(x2)
```

  - Explicação

    Põe valores maiores que 16 bits na memória a partir do sw (que já estava implementado), positivo e negativo. Cada tipo de load é testado para os valores guardados na memória.

- Simulação STORES
  - Valores menores que 8 bits
    - Código

```
addi x3,x0,1
addi x4,x0,-8
addi x6,x0,4
sb x6,0(x3)
sh x6,8(x3)
sw x6,16(x3)
lw x10,0(x3)
lw x11,8(x3)
lw x12,16(x3)
sb x4,0(x3)
sh x4,8(x3)
sw x4,16(x3)
lw x13,0(x3)
```

```
lw x14,8(x3)
lw x15,16(x3)
```

- Explicação

  Guarda valores menores que 8 bits na memória a partir de cada store, positivo e negativo. O lw (que já estava implementado) verifica se foram guardados os valores certos na memória.

- Valores maiores que 16 bits
  - Código

```
addi x3,x0,1
addi x4,x0,-2001
addi x6,x0,2001
slli x4,x4,7
slli x6,x6,7
sb x6,0(x3)
sh x6,8(x3)
sw x6,16(x3)
lw x10,0(x3)
lw x11,8(x3)
lw x12,16(x3)
sb x4,0(x3)
sh x4,8(x3)
sw x4,16(x3)
lw x13,0(x3)
lw x14,8(x3)
lw x15,16(x3)
```

  - Explicação

    Guarda valores maiores que 16 bits na memória a partir de cada store, positivo e negativo. O lw (que já estava implementado) verifica se foram guardados os valores certos na memória.

- Simulação Branches
  - Desvios
    - Código

```
addi x3,x0,1
addi x5,x0,0
addi x10,x0,0
addi x15,x0,0
addi x6,x0,2
addi x7,x0,-2
addi x5,x5,1
bge x6,x5,-4
sub x10,x10,x3
bne x7,x10,-4
add x15,x15,x3
blt x15,x6,-4
```

        - Explicação

          O addi põe valores nos registradores e a partir deles cada tipo de branch é testado por mini-loops. O bge vai ser verdadeiro até que x5 (inicialmente 0) ficar com o valor maior que x6 (com o valor 2). O bne vai ser verdadeiro até x10 (inicialmente 0) ficar com o valor igual a x7 (com o valor -2). O blt vai ser verdadeiro até x15 (inicialmente 0) ficar com o valor maior que x6 (com o valor 2).

- Simulação do JAL e JALR
  - Um desvio do JAL
    - Código

```
beq x0,x0,16
add x5,x5,x3
blt x5,x6,-4
beq x0,x0,20
addi x3,x0,1
addi x5,x0,0
addi x6,x0,2
jal x10,-24
addi x29,x0,6
```

    - Explicação

      Inicialmente, ele vai para a linha 5 por causa do beq na primeira linha. Depois, ele guarda alguns valores nos registradores. Após isso, ele faz o desvio para a linha 2, roda um pequeno loop e vai para a última linha, por causa do beq.

  - Um desvio do JALR
    - Código

```
beq x0,x0,16
add x5,x5,x3
blt x5,x6,-4
beq x0,x0,24
addi x3,x0,1
addi x5,x0,0
addi x6,x0,2
addi x8,x0,32
jalr x10,x8,-28
addi x29,x0,6
```

    - Explicação

      Inicialmente, ele vai para a linha 5 por causa do beq na primeira linha. Depois, ele guarda alguns valores nos registradores. Após isso, ele faz o desvio para a linha 2, roda um pequeno loop e vai para a última linha, por causa do beq.

- Simulação ALU
  - Operações
    - Código

```
addi x1,x0,1
addi x2,x0,2
addi x3,x0,3
addi x21,x0,-1
addi x22,x0,-2
addi x23,x0,-3
or x4,x1,x2
or x4,x4,x0
or x24,x21,x22
xor x5,x2,x1
xor x5,x3,x2
xor x25,x22,x21
slt x6,x2,x1
slt x6,x1,x2
slt x26,x22,x21
slt x26,x21,x22
sub x7,x1,x3
sub x7,x7,x3
sub x27,x21,x23
sub x27,x27,x23
srai x8,x2,1
srai x28,x22,1
srli x9,x2,1
srli x29,x22,1
slli x10,x2,1
slli x30,x22,1
slti x11,x2,1
slti x12,x1,2
slti x31,x22,-1
slti x31,x21,-2
```

    - Explicação

      Inicialmente guarda valores positivos e negativos em alguns registradores e para cada instrução da ALU é feita sua operação com números positivos e negativos.

- Simulação LUI
  - Operação
    - Código

```
lui x3,3
lui x23,-3
lui x1,1
lui x21,-1
```

    - Explicação

      Testa valores possíveis para o LUI, positivos e negativos.

- Simulação Halt
  - Teste
    - Código

```
addi x1,x0,1
addi x2,x0,2
halt
addi x3,x0,3
```

    - Explicação

      A intenção é que a última linha não seja executada por causa do HALT.

3.2 Resultados das Simulações

- Simulação LOAD
  - Valores menores que 8 bits
    - Resultado

```
 55: Register [ 2] written with value: [00000001] | [          1]
 65: Register [ 3] written with value: [fffffff8] | [         -8]
 75: Register [ 4] written with value: [00000004] | [          4]
 75: Memory [  1] written with value: [00000004] | [          4]
 85: Memory [  9] written with value: [fffffff8] | [         -8]
 95: Memory [  1] read with value: [00000004] | [          4]
105: Register [ 5] written with value: [00000004] | [          4]
105: Memory [  1] read with value: [00000004] | [          4]
115: Register [ 6] written with value: [00000004] | [          4]
115: Memory [  1] read with value: [00000004] | [          4]
125: Register [ 7] written with value: [00000004] | [          4]
125: Memory [  1] read with value: [00000004] | [          4]
135: Register [ 8] written with value: [00000004] | [          4]
145: Memory [  9] read with value: [fffffff8] | [         -8]
155: Register [ 9] written with value: [fffffff8] | [         -8]
155: Memory [  9] read with value: [fffffff8] | [         -8]
165: Register [10] written with value: [fffffff8] | [         -8]
165: Memory [  9] read with value: [fffffff8] | [         -8]
175: Register [11] written with value: [fffffff8] | [         -8]
175: Memory [  9] read with value: [000000f8] | [        248]
185: Register [12] written with value: [000000f8] | [        248]
```

- Valores maiores que 16 bits
  - Resultado

```
 55: Register [ 2] written with value: [00000001] | [          1]
 65: Register [ 3] written with value: [fffff82f] | [      -2001]
 75: Register [ 4] written with value: [000007d1] | [       2001]
 85: Register [ 3] written with value: [fffc1780] | [    -256128]
 95: Register [ 4] written with value: [0003e880] | [     256128]
 95: Memory [  1] written with value: [0003e880] | [     256128]
105: Memory [  9] written with value: [fffc1780] | [    -256128]
115: Memory [  1] read with value: [00000080] | [        128]
125: Register [ 5] written with value: [00000080] | [        128]
125: Memory [  1] read with value: [0000e880] | [      59520]
135: Register [ 6] written with value: [0000e880] | [      59520]
135: Memory [  1] read with value: [0003e880] | [     256128]
145: Register [ 7] written with value: [0003e880] | [     256128]
145: Memory [  1] read with value: [00000080] | [        128]
155: Register [ 8] written with value: [00000080] | [        128]
165: Memory [  9] read with value: [ffffff80] | [       -128]
175: Register [ 9] written with value: [ffffff80] | [       -128]
175: Memory [  9] read with value: [ffff1780] | [     -59520]
185: Register [10] written with value: [ffff1780] | [     -59520]
185: Memory [  9] read with value: [fffc1780] | [    -256128]
195: Register [11] written with value: [fffc1780] | [    -256128]
195: Memory [  9] read with value: [00000080] | [        128]
205: Register [12] written with value: [00000080] | [        128]
```

- Simulação STORE
  - Valores maiores que 8 bits
    - Resultado

```
 55: Register [ 3] written with value: [00000001] | [          1]
 65: Register [ 4] written with value: [fffffff8] | [         -8]
 75: Register [ 6] written with value: [00000004] | [          4]
 75: Memory [  1] written with value: [00000004] | [          4]
 85: Memory [  9] written with value: [00000004] | [          4]
 95: Memory [ 17] written with value: [00000004] | [          4]
105: Memory [  1] read with value: [00000004] | [          4]
115: Register [10] written with value: [00000004] | [          4]
115: Memory [  9] read with value: [00000004] | [          4]
125: Register [11] written with value: [00000004] | [          4]
125: Memory [ 17] read with value: [00000004] | [          4]
135: Register [12] written with value: [00000004] | [          4]
135: Memory [  1] written with value: [fffffff8] | [         -8]
145: Memory [  9] written with value: [fffffff8] | [         -8]
155: Memory [ 17] written with value: [fffffff8] | [         -8]
165: Memory [  1] read with value: [fffffff8] | [         -8]
175: Register [13] written with value: [fffffff8] | [         -8]
175: Memory [  9] read with value: [fffffff8] | [         -8]
185: Register [14] written with value: [fffffff8] | [         -8]
185: Memory [ 17] read with value: [fffffff8] | [         -8]
195: Register [15] written with value: [fffffff8] | [         -8]
```

- Valores maiores que 16 bits
  - Resultado

```
 55: Register [ 3] written with value: [00000001] | [          1]
 65: Register [ 4] written with value: [fffff82f] | [      -2001]
 75: Register [ 6] written with value: [000007d1] | [       2001]
 85: Register [ 4] written with value: [fffc1780] | [    -256128]
 95: Register [ 6] written with value: [0003e880] | [     256128]
 95: Memory [  1] written with value: [0003e880] | [     256128]
105: Memory [  9] written with value: [0003e880] | [     256128]
115: Memory [ 17] written with value: [0003e880] | [     256128]
125: Memory [  1] read with value: [00000080] | [        128]
135: Register [10] written with value: [00000080] | [        128]
135: Memory [  9] read with value: [0000e880] | [      59520]
145: Register [11] written with value: [0000e880] | [      59520]
145: Memory [ 17] read with value: [0003e880] | [     256128]
155: Register [12] written with value: [0003e880] | [     256128]
155: Memory [  1] written with value: [fffc1780] | [    -256128]
165: Memory [  9] written with value: [fffc1780] | [    -256128]
175: Memory [ 17] written with value: [fffc1780] | [    -256128]
185: Memory [  1] read with value: [ffffff80] | [       -128]
195: Register [13] written with value: [ffffff80] | [       -128]
195: Memory [  9] read with value: [ffff1780] | [     -59520]
205: Register [14] written with value: [ffff1780] | [     -59520]
205: Memory [ 17] read with value: [fffc1780] | [    -256128]
215: Register [15] written with value: [fffc1780] | [    -256128]
```

- Simulação Branch
  - Desvios
    - Resultado

```
 55: Register [ 3] written with value: [00000001] | [          1]
 65: Register [ 5] written with value: [00000000] | [          0]
 75: Register [10] written with value: [00000000] | [          0]
 85: Register [15] written with value: [00000000] | [          0]
 95: Register [ 6] written with value: [00000002] | [          2]
105: Register [ 7] written with value: [fffffffe] | [         -2]
115: Register [ 5] written with value: [00000001] | [          1]
155: Register [ 5] written with value: [00000002] | [          2]
195: Register [ 5] written with value: [00000003] | [          3]
215: Register [10] written with value: [ffffffff] | [         -1]
255: Register [10] written with value: [fffffffe] | [         -2]
275: Register [15] written with value: [00000001] | [          1]
315: Register [15] written with value: [00000002] | [          2]
```

- Simulação do JAL e JALR
  - Um desvio do JAL
    - Resultado

```
 85: Register [ 3] written with value: [00000001] | [          1]
 95: Register [ 5] written with value: [00000000] | [          0]
105: Register [ 6] written with value: [00000002] | [          2]
115: Register [10] written with value: [00000020] | [         32]
145: Register [ 5] written with value: [00000001] | [          1]
185: Register [ 5] written with value: [00000002] | [          2]
235: Register [29] written with value: [00000006] | [          6]
```

  - Um desvio do JALR
    - Resultado

```
 85: Register [ 3] written with value: [00000001] | [          1]
 95: Register [ 5] written with value: [00000000] | [          0]
105: Register [ 6] written with value: [00000002] | [          2]
115: Register [ 8] written with value: [00000020] | [         32]
125: Register [10] written with value: [00000024] | [         36]
155: Register [ 5] written with value: [00000001] | [          1]
195: Register [ 5] written with value: [00000002] | [          2]
245: Register [29] written with value: [00000006] | [          6]
```

- Simulação ALU
  - Operações
    - Resultado

```
 55: Register [ 1] written with value: [00000001] | [          1]
 65: Register [ 2] written with value: [00000002] | [          2]
 75: Register [ 3] written with value: [00000003] | [          3]
 85: Register [21] written with value: [ffffffff] | [         -1]
 95: Register [22] written with value: [fffffffe] | [         -2]
105: Register [23] written with value: [fffffffd] | [         -3]
115: Register [ 4] written with value: [00000003] | [          3]
125: Register [ 4] written with value: [00000003] | [          3]
135: Register [24] written with value: [ffffffff] | [         -1]
145: Register [ 5] written with value: [00000003] | [          3]
155: Register [ 5] written with value: [00000001] | [          1]
165: Register [25] written with value: [00000001] | [          1]
175: Register [ 6] written with value: [00000000] | [          0]
185: Register [ 6] written with value: [00000001] | [          1]
195: Register [26] written with value: [00000001] | [          1]
205: Register [26] written with value: [00000000] | [          0]
215: Register [ 7] written with value: [fffffffe] | [         -2]
225: Register [ 7] written with value: [fffffffb] | [         -5]
235: Register [27] written with value: [00000002] | [          2]
245: Register [27] written with value: [00000005] | [          5]
255: Register [ 8] written with value: [00000001] | [          1]
265: Register [28] written with value: [ffffffff] | [         -1]
275: Register [ 9] written with value: [00000001] | [          1]
285: Register [29] written with value: [7fffffff] | [ 2147483647]
295: Register [10] written with value: [00000004] | [          4]
305: Register [30] written with value: [fffffffc] | [         -4]
315: Register [11] written with value: [00000000] | [          0]
325: Register [12] written with value: [00000001] | [          1]
335: Register [31] written with value: [00000001] | [          1]
345: Register [31] written with value: [00000000] | [          0]
```

- Simulação LUI
  - Operação
    - Resultado

```
55: Register [ 3] written with value: [00003000] | [      12288]
65: Register [23] written with value: [fffd0000] | [     -12288]
75: Register [ 1] written with value: [00001000] | [       4096]
85: Register [21] written with value: [fffff000] | [      -4096]
```

- Simulação HALT
  - Teste
    - Resultado

```
55: Register [ 1] written with value: [00000001] | [          1]

65: Register [ 2] written with value: [00000002] | [          2]
```

4- Conclusão

4.1 - Resultados obtidos

Até o momento (08/08/2024) foram implementadas, simuladas no ModelSim e verificadas 21 das 21 instruções do projeto.

4.2 - Análise se os objetivos foram alcançados

A partir dos testes realizados, todos os objetivos foram alcançados.