Universidade Federal do Rio Grande do Norte (UFRN)

Relatório - Processador RISC

Daniel Sehn Colao
João Victor Malheiros

Natal
2022

1. Instruções suportadas

AND —> operação booleana AND
OR —> operação booleana OR
XOR —> operação booleana XOR
NOT —> operação booleana NOT
CMP —> comparação
ADD —> soma
SUB —> subtração

LD —> leitura em memória
ST —> armazenamento em memória
LI —> leitura de imediato

J —> salto incondicional
JN —> salto condicional; salta se (N)egativo
JZ —> salto condicional; salta se (Z)ero

2. Decisões do Projeto

2.1- **Tamanho da palavra do processador**

O tamanho da palavra do processador foi definido como sendo 32 bits. Este valor foi escolhido por ser tanto um múltiplo de 8, como também comum em processadores.

2.2- **Formato da palavra de instrução**

A instrução será composta por 4 campos: opcode, operando fonte 1 (op1), operando fonte 2 (op2) e operando destino (op3). O opcode terá 5 bits, enquanto os demais, 9 bits.

A quantidade de bits destinada ao campo do opcode é suficiente para trabalhar com todos os modos de endereçamento e operações suportadas pelo processador (aritmética, memória de dados e controle).

Abaixo, a tabela indica a associação entre valor e sua respectiva instrução associada.

Tabela 1: Opcodes das instruções suportadas

OPCODE	INSTRUÇÃO
1	J
2	JZ
3	JZ
4	LI
5	LD

6	ST
7	ADD
8	SUB
9	XOR
10	AND
11	OR
12	NOT
13	CMP

2.2.1 Instrução Lógica/Aritmética: AND, OR, XOR, NOT, CMP, ADD e SUB.

Todas essas instruções são executadas pela ULA do nosso processador, componente que recebe os valores armazenados nos operandos 1 e 2 (fonte) e envia o resultado para que seja salvo no endereço do operando destino.

O resultado da operação é enviado para o mux de banco de registradores para que este selecione o valor apropriado para escrever no banco de registradores.

Especificamente, a instrução CMP apenas altera o valor das flags zero e negativo com base nos operandos 1 e 2 (valores da saída do banco de registradores). Esta instrução não produz nenhum outro resultado. Esta instrução permite comparar quaisquer valores armazenados no banco de registradores. As demais instruções, além de realizarem a computação do resultado, também atualizam os valores das flags zero e negativo.

Durante a execução dessas instruções, o pipeline será pausado, não havendo a propagação de instruções, para que o conflito de dados seja evitado.

A atribuição de valor às flags acima dá-se pela seguinte maneira:
Flag negativo (booleano) : OP1 < OP2.
Flag zero (booleano) : OP1 == OP2.

Essas flags serão utilizadas durante a execução de uma instrução de jump condicional. Dessa maneira, deverão ser enviadas para a unidade de controle.

Operandos da instrução:
OP 1 (fonte): endereço de registrador.
OP 2 (fonte): endereço de registrador.
OP 3 (destino): endereço de registrador.

Exemplo: ADD 2 1 5

Esta instrução é de adição (opcode = ADD). Serão buscados os valores armazenados nos registradores de endereço ‘2’ e ‘1’. Após isso, realiza-se a soma desses, e o resultado será salvo no endereço de registrador 5.

2.2.2 Instruções de Memória: LD, ST e LI

Leitura (LD): O endereço da memória será de onde iremos ler o valor armazenado. O valor lido será salvo no endereço do registrador contido na instrução.

OP 1 (fonte): endereço de memória.

OP 2 (fonte): qualquer valor, pois será ignorado.

OP 3 (destino): endereço de registrador.

Exemplo: LD 1 0 4.

Esta instrução é de leitura de memória (opcode = LD). Primeiramente, será acessada a memória de dados para obter o valor armazenado no endereço de memória '1', o controle envia o seletor com valor 'true' para obter o endereço presente no OP1. Após isso, o controle envia para o mux do banco de registradores o valor igual a 1 (seletor) para que seja escolhido o valor oriundo da leitura de memória.

Finalmente, o resultado será salvo no endereço de registrador '4'.

O valor '0', na instrução, é ignorado.

Escrita (ST): O endereço da memória será onde iremos salvar o valor armazenado no endereço do registrador contido na instrução.

OP 1 (fonte): endereço de registrador.

OP 2 (fonte): qualquer valor, pois será ignorado.

OP 3 (destino): endereço de memória.

Leitura de imediato (LI): Iremos escrever o valor imediato contido no campo OP 1 da instrução no endereço de registrador presente no campo OP 3 (destino).

OP 1 (fonte): Valor a ser armazenado.

OP 2 (fonte): qualquer valor, pois será ignorado.

OP 3 (destino): endereço de registrador.

2.2.3 Instruções de Controle: J, JN e JZ.

O endereço de salto estará presente no operando OP 3 (destino). Para os saltos condicionais, o componente "controle" utilizará as flags "zero" e "negativo" para decidir se deverá ou não realizar o salto. Tais flags serão resetadas quando um jump condicional acontecer.

Jump if Negative (JN): Ocorre a instrução de jump se e somente se a flag N da ULA estiver ativa.

OP1 (fonte): qualquer valor, pois será ignorado.

OP2 (fonte): qualquer valor, pois será ignorado.

OP3 (destino): endereço da instrução (memória de instruções) a ser executada.

Jump if Zero (JZ): Ocorre a instrução de jump se e somente se a flag Z da ULA estiver ativa.

OP1 (fonte): qualquer valor, pois será ignorado.

OP2 (fonte): qualquer valor, pois será ignorado.

OP3 (destino): endereço da instrução (memória de instruções) a ser executada.

Jump incondicional (J): Instrução de jump sem verificação das flags citadas acima.
OP1 (fonte): qualquer valor, pois será ignorado.
OP2 (fonte): qualquer valor, pois será ignorado.
OP3 (destino): endereço da instrução (memória de instruções) a ser executada.

2.3 - **Modos de endereçamento de operandos**

O modo de endereçamento é a maneira pela qual o operando de uma instrução é especificado. São as diferentes formas de especificar a localização ou o valor de um operando na instrução.

1. **Register addressing mode** - As instruções lógicas/aritméticas trabalham com este tipo de endereçamento. Todos os seus operandos são referenciados por meio de endereços do banco de registradores.
   As instruções ST, LD e LI também utilizam esse modo endereçamento.

2. **Direct addressing mode** - Nas instruções de acesso à memória de dados, os operandos da instrução contém o endereço físico da memória onde iremos ler/escrever um valor.

3. **Immediate addressing mode** - O campo OP1 da instrução contém o valor a ser trabalhado, não sendo necessário acessar nenhuma memória para obtê-lo. A instrução LI trabalha com este tipo de endereçamento.

2.4 e 2.5 - Tamanho do banco de registradores, das memórias de instruções e de dados;

O banco de registradores terá 512 registradores de 32 bits.
Memória de instruções: 512 registradores de 32 bits.
Memória de dados: 512 registradores de 32 bits.
Como as palavras possuem 32 bits, então o tamanho dos registradores e das memórias deve ser igual a 32 bits também, a fim de armazenar os valores e instruções.
O tamanho de 512 foi escolhido por ser um múltiplo de 8.

2.6 - Número e tipos de barramentos (ou canais dedicados) da parte operativa;

PC: 5 barramentos internos do processador.
Barramento de controle: 3 (clock, enable, pc_jump).
pc_jump booleano que indica a necessadidade realizar um jump.
Barramento de endereço: 2 (endereço da próxima instrução e endereço do jump).

ULA: 7 barramentos internos do processador.

Barramento de dados: 4 (opcode, valor do op1, valor do op2 e o resultado).
Barramento de controle: 3 (reset_zn, zero e negativo).

Banco de registradores: 9 barramentos internos do processador.
Barramento de dados: 3.
Valores armazenados nos endereços de OP1 e OP2.
Valor que vem da ULA, leitura de memória ou imediato.
Barramento de endereços: 3 (endereços OP1, OP2 e OP3).
Barramento de controle: 3 (clock, write e enable).

Memória de dados: 6 barramentos.
Barramento de dados: 2.
Valor de entrada (escrita) e valor de saída (leitura).
Barramento de endereços: 1 (endereço)
Barramento de controle: 3 (clock, write e enable).

Memória de instruções: 4 barramentos ao todo.
Barramento de controle: 2 (clock e enable).
Barramento de endereços: 2 (entrada e saída).

2.7- Organização do pipeline.

Hazard de dados: Incompatibilidade de dados. Ocorre quando se obtém o valor de um registrador que não teve seu conteúdo atualizado após uma operação da ULA. Assim, estaremos realizando cálculos em cima de valores incorretos.

Solução: No momento em que uma operação de ULA for executada, o pipeline será travado até que esta tenha sido executada e o valor tenha sido armazenado no banco de registradores (write back). Os registradores de pipeline não serão sobrescritos, o decodificador é travado para que não sejam enviados os campos da instrução para esses registradores. O decodificador será destravado no momento em que as operações, que necessitem da escrita de um resultado no banco de registradores, forem finalizadas, evitando, assim, a manipulação de valores incorretos.

Hazard de controle: Quando uma instrução é obtida, o PC incrementa em uma unidade para obter a próxima instrução. O hazard de controle acontece quando uma instrução de salto (J, JZ e JN) estiver sendo executada e o salto aconteça. No momento em que a instrução de jump estiver sendo executada, outras instruções estão em estágios anteriores no pipeline. No entanto, não necessariamente a próxima instrução (PC + 1) será executada, possibilitando a execução em ordem incorreta de instruções no processador.

Solução: Pipeline cleaning. As instruções em estágios anteriores serão excluídas do pipeline. É utilizada uma flag, chamada flag “pipeline_reset”, que fará com que o controle do processador volte para o estado 0, evitando a execução de uma instrução incorreta.

Dessa maneira, a organização do pipeline será pela criação de bolhas quando houver uma instrução de leitura ou lógica/aritmética, além do cleaning quando houver uma instrução de salto.

O pipeline implementado possui 2 estágios: fetch de instrução (PC) e execução da instrução.

Os componentes PC, memória de instruções e o decodificador são travados no momento em que houver a execução de uma instrução de leitura ou operação lógica/aritmética. Dessa maneira, é evitado o conflito de dados, já que sempre estará sendo escrito o resultado de uma dessas operações no banco de registradores antes da leitura do valor armazenado na posição do destino. O destravamento acontece após a escrita (write back) do resultado no banco de registradores.

3. Composição do processador

- Program Counter

- Este componente será responsável por armazenar o endereço da próxima instrução a ser executada. Tal endereço será enviado para a memória de instruções.
- Possui um contador interno que é incrementado toda vez que for enviado o endereço da instrução a ser executada pelo processador, assim como quando houver uma instrução de jump.

- **Entradas**: Clock, posição da próxima instrução - salto (in)condicional (a unidade de controle apenas envia o sinal se o JUMP ocorrer).
    - Sinais oriundos do controle:
        - **PC_enable**: indicador para incremento do contador de instrução.
        - **PC_jump**: indicador para realizar o jump.
        - **Instrução_jump**: Endereço da instrução de jump (9 bits).

- **Saída**: Endereço da instrução a ser executada.

- Memória de instruções

- Este componente conterá um banco de instruções e será responsável por enviar, ao decodificador, a instrução armazenada no endereço enviado pelo componente PC.
- **Entradas:** Endereço da instrução a ser executada, enable (sinal do controle), clock.
- **Saída:** Instrução (inteiro de 32 bits).

- Decodificador

- Receberá uma instrução e irá quebrá-la para obtenção do opcode e operandos.
- Entrará em execução toda vez que receber uma instrução.

- **Entrada:** Clock**.**
    - Sinais recebidos de componentes:
    - **Memória de Instruções**:
        - Instrução como um inteiro de 32 bits.

    - **Controle:**
        - Enable (booleano), enviado pelo controle.
- **Saídas:**
    - OPCODE (5 bits).
    - operando fonte 1 (OP1, 9 bits).
    - operando fonte 2 (OP1, 9 bits).
    - operando destino (OP3, 9 bits).

- **Registradores de Pipeline:**

    - Este componente é responsável por receber os campos da instrução, enviados pelo decodificador, e enviá-los para os componentes “controle” e “banco de registradores”.
    - Este componente possui campos internos para armazenar os campos da instrução a ser executada, de modo que estágios do pipeline sejam desacoplados.
    - A atualização (escrita) acontece mediante a permissão do controle, o qual enviará sinais booleanos de enable e write.
    - **Entradas**: clock.
        - Sinais recebidos de outros componentes:
        - **Controle**:
            - Enable (booleano, ativar componente).
            - Write (booleano, permitir a escrita de novos valores no componente).
        - **Decodificador**:
            - Opcode (5 bits).
            - Operando fonte da instrução (OP1, 9 bits).
            - Operando fonte da instrução (OP2, 9 bits).
            - Operando destino da instrução (OP3, 9 bits).
    - **Saídas**: OPCODE, OP1, OP2 e OP3.

- **Controle**
    - Será responsável por coordenar a execução de uma instrução no processador com base em seu OPCODE. Para isso, este componente enviará sinais aos componentes do datapath. Consiste na implementação da máquina de estados presente no diagrama de estados.
    - **Entradas:** clock.
        - Sinais recebidos de outros componentes:
        - **ULA**:
            - Flag zero (booleano)
            - Flag negativo (booleano)

- **Registradores de Pipeline**:
    - Opcode.
    - Operando fonte da instrução (OP1).
    - Operando fonte da instrução (OP2).
    - Operando destino da instrução (OP3).

- **Saídas**:
    - Sinais que são enviados pelo controle para outros componentes:
    - **PC**:
        - enable_pc (booleano, continuar ou parar o incremento)
        - pc_jump (booleano, realizar salto)
        - pc_jump_value (sc_uint<9>, endereço da instrução para saltar).

    - **Memória de instruções**:
        - enable_mem_instrucoes (booleano, ativar a memória de instruções).

    - **Decodificador:**
        - enable_decodificador (booleano, ativar o decodificador).

    - **ULA:**
        - flag zero (booleano).
        - flag negativo (booleano).
        - enable_ula (booleano, ativar a ula) .
        - reset_zn (booleano, resetar as flags zero e negativo).
        - opcode_ula (sc_uint<5>, operação a ser realizada).

    - **Banco de registradores**:
        - enable_banco_reg (booleano, ativar o banco de registradores),
        - write_banco_reg (booleano, permissão de escrita)
        - seletor_mux_banco_reg (sc_uint<2>, valor do seletor do mux do banco de registradores).

    - **Memória de dados**:
        - enable_mem_dados (booleano; ativar a memória de dados).
        - write_mem_dados (booleano, permitir a escrita na memória de dados).
        - seletor_mux_mem_dados (booleano; selecionar o endereço OP1 ou OP3 que seja de memória, no multiplexador da memória de dados).

    - **Registradores de Pipeline:**
        - pipeline_reg_enable (booleano; ativar o registrador de pipeline).
        - pipeline_reg_write (booleano, permitir a escrita de novos valores no registrador de pipeline).

- **Multiplexador - Memória de Dados:**
  - seletor_mux_mem_dados (booleano; selecionar a entrada: endereço OP1 ou endereço OP3)

- **Multiplexador - Banco de Registradores**:
  - seletor_mux_banco_reg (sc_uint<2>, selecionar qual valor será armazenado no registrador)
  - valor_imediato (sc_uint<9>, valor imediato contido no OP1 da instrução LI)
  - endereco_imediato (sc_uint<9>, endereço de registrador no qual iremos armazenar o valor_imediato)

- Banco de registradores

- Componente que representa o banco de registradores de um processador.
- **Entradas**:
  - Sinais de controle (ver item acima em “saídas -> Banco de registradores).
  - **Mux do banco de registradores:**
    - Valor a ser armazenado, oriundo de uma leitura de memória, ULA ou imediato, no endereço indicado pelo OP3.
  - **Registradores de Pipeline**:
    - Endereços de registradores (OP1, OP2 e OP3).

- **Saídas**: Valores (inteiros 32 bits) armazenados nos endereços de registradores indicados pelos operandos fonte OP1 e OP2 .

- ULA

- Responsável pela execução das instruções lógica-aritmética.
- **Entradas:** Valores do OP1 e OP2, ambos inteiros de 32 bits, enviados pelo banco de registradores, além do identificador de operação (opcode) enviado pelo controle.
- **Saídas:** resultado da operação (inteiro de 32 bits), flag zero (op1 == op2) e flag negativo (op1 < op2).

- Multiplexador - Banco de Registradores

- Responsável por selecionar o valor a ser armazenado no endereço de registrador destino (OP 3).
- O valor pode ser oriundo de uma leitura de memória, execução de uma operação na ULA ou um valor imediato.
- **Entradas:**
  - Sinais recebidos de componentes:
  - **ULA**:
    - Resultado de uma operação da ULA (32 bits).
  - **Memória de dados**:

        - Valor oriundo de uma leitura (32 bits)
    - **Controle**:
        - Seletor (2 bits).
        - Valor imediato (9 bits).

- **Saídas:** Valor a ser armazenado (32 bits) no endereço de registrador presente no OP3.

- **Multiplexador - Memória de Dados**
    - Responsável por selecionar o endereço de memória a ser consumido pela memória de dados.
    - Leitura de memória: o endereço de memória estará no OP 1.
    - Escrita de memória: o endereço de memória estará no OP 3.
    - **Entradas:** seletor e dois endereços de memória (OP 1 e OP 3). Enviados pelo controle.
    - **Saída:** endereço de memória.

- **Memória de dados**
    - Este componente é a memória RAM do computador que contém os dados a serem trabalhados.
    - **Entradas:**
        - **MUX da memória de dados**:
            - Endereço de memória (leitura ou escrita)
        - **Banco de registradores**:
            - Valor de 32 bits (escrita).
    - **Saída:** valor (leitura).

4. Diagramas de estados e organização do processador

Estão presentes, em pdf, no arquivo deste projeto.
Foram omitidos vários sinais do controle no diagrama do datapath por causa de poluição visual causada pela quantidade de fios, que prejudica o entendimento do datapath.

Cada estado, do diagrama de estados, possui os sinais que o componente de controle envia para os demais componentes, coordenando suas respectivas operações. A transição de estado é mediante a subida de clock. Os sinais presentes neste diagrama são detalhados nos componentes do processador, no item 3.

O diagrama da organização do processador contém todos os componentes do processador e suas respectivas interligações.

5. Sequência de execução

Detalhamento acerca da execução das instruções do processador.

5.1 Operações lógicas/aritméticas

1. O contador de programa (PC) obtém o endereço da próxima instrução e envia-o para a memória de endereços.
2. A memória de instruções recebe do PC o endereço, realiza a busca da instrução e envia-a para o decodificador.
3. O decodificador recebe a instrução, inteiro de 32 bits, da memória de instruções e quebra nos 4 campos que a compõem: opcode, operando fonte 1 (OP1), operando fonte 2 (OP2) e operando destino (OP3). Estes campos são enviados para os registradores do pipeline.
4. O registrador do pipeline recebe a instrução decodificada e envia os campos para o controle do processador, assim como para o banco de registradores.
5. O controle verifica o opcode da instrução e determina qual operação deve ser feita pela ULA. Com isso, o controle enviará sinais para os demais componentes do processador, para coordenar a execução. Logo, a propagação do pipeline é pausada.
6. O banco de registradores recebe os operandos fonte dos registradores de pipeline, e envia para a ULA os valores armazenados neles. O operando destino contém o endereço no qual será armazenado o resultado de uma operação de leitura ou lógica/aritmética, que também é lido dos registradores de pipeline.
7. A ULA recebe do banco de registradores os valores armazenados nos operandos OP1 e OP2, para então calcular o resultado, que será enviado para o multiplexador do banco de registradores.
8. O multiplexador do banco de registradores selecionará qual valor deverá ser armazenado no endereço do OP3. O controle envia o valor do seletor.
9. O banco de registradores recebe, então, o valor a ser armazenado e salvo no endereço de destino (OP3).
10. Instrução finalizada, a propagação do pipeline é resumida.

5.2 Instruções de acesso à memória de dados

Os passos 1 ao 5 destas instruções são iguais aos passos descritos no item 5.1.

6. O controle envia os operandos fonte (OP1) e destino (OP3) para o multiplexador da memória de dados, o qual será responsável por selecionar o endereço de memória de acordo com o seletor enviado pelo controle.

7. Na instrução de escrita na memória, o banco de registradores recebe o endereço do operando fonte 1 (OP1), do registrador de pipeline, e envia o valor armazenado para a memória de dados.

8. Leitura: a memória de dados recebe o endereço de memória, busca o valor armazenado e envia-o para o multiplexador do banco de registradores. O controle enviará um sinal específico para selecionar a entrada deste multiplexador.

8. Escrita: a memória de dados recebe o endereço de memória e o valor a ser armazenado, que vem do banco de registradores.

9. Execução finalizada, a propagação do pipeline será resumida.

5.3 Instruções de salto (jump)

Os passos 1 ao 5 destas instruções são similares aos passos descritos no item 5.1.

A diferença é que no lugar de pausar a propagação do pipeline, o controle envia o endereço presente no operando destino da instrução para o contador de programa junto com o sinal pc_jump, indicando que deve ser feito o salto.

Isso é feito para a instrução de salto incondicional (J). Caso seja instrução de salto condicional (JN e JZ) são verificadas as flags para determinação se é ou não para realizar o salto.

Caso não seja feito o salto condicional, o controle envia o sinal pc_jump para o contador de programa com valor falso, indicando a não necessidade do jump. O processador continuará a execução das microinstruções que estão no pipeline.

6. Compilação e criação de algoritmos

Verificar arquivo README.md encaminhado juntamente com o projeto.

7. Algoritmos e resultados

Algoritmo 1: Somador de 4 números, usado para testar se a execução está ocorrendo na ordem correta.
Instruções e resultado da execução do algoritmo 1:

algo3.txt

```
1  memset 1 1
2  memset 2 2
3  memset 3 3
4  memset 4 4
5  LD 1 0 1
6  LD 2 0 2
7  LD 2 0 2
8  LD 3 0 3
9  ADD 1 2 2
10 ADD 2 3 3
11 ADD 3 4 4
```

```
        ALL RIGHTS RESERVED
EXECUCAO: [5|1|0|1]
EXECUCAO: [5|1|0|1]
EXECUCAO: [5|2|0|2]
EXECUCAO: [5|2|0|2]
EXECUCAO: [5|3|0|3]
EXECUCAO: [7|1|2|2]
EXECUCAO: [7|2|3|3]
EXECUCAO: [7|3|4|4]
INSTRUÇÃO STOP ACIONADA!
FIM DA EXECUÇÃO

Info: /OSCI/SystemC: Simulation stopped by user.

Numero de ciclos de clock: 44

Banco de Registradores  Memoria de Instrucoes   Memoria de Dados
[0]: 0                  [0]: 8388645                    [0]: 0
[1]: 1                  [1]: 16777285                   [1]: 1
[2]: 3                  [2]: 16777285                   [2]: 2
[3]: 6                  [3]: 25165925                   [3]: 3
[4]: 6                  [4]: 16810023                   [4]: 4
[5]: 0                  [5]: 25215047                   [5]: 0
[6]: 0                  [6]: 33620071                   [6]: 0
[7]: 0                  [7]: 15                 [7]: 0
[8]: 0                  [8]: 0                  [8]: 0
[9]: 0                  [9]: 0                  [9]: 0
[10]: 0                 [10]: 0                 [10]: 0
[11]: 0                 [11]: 0                 [11]: 0
[12]: 0                 [12]: 0                 [12]: 0
[13]: 0                 [13]: 0                 [13]: 0
[14]: 0                 [14]: 0                 [14]: 0
[15]: 0                 [15]: 0                 [15]: 0
[16]: 0                 [16]: 0                 [16]: 0
[17]: 0                 [17]: 0                 [17]: 0
[18]: 0                 [18]: 0                 [18]: 0
[19]: 0                 [19]: 0                 [19]: 0
```

Algoritmo 2: Algoritmo feito para testar o jump incondicional
Instruções e execução do algoritmo 2:

algo6.txt

```
1   memset 1 1
2   memset 2 2
3   memset 3 3
4   memset 4 4
5   LD 1 0 1
6   LD 2 0 2
7   LD 3 0 3
8   LD 4 0 4
9   J 0 0 8
10  ADD 1 2 2
11  ADD 2 3 3
12  ADD 3 4 4
13  ADD 2 3 5
```

```
        ALL RIGHTS RESERVED
EXECUCAO: [5|1|0|1]
EXECUCAO: [5|1|0|1]
EXECUCAO: [5|2|0|2]
EXECUCAO: [5|3|0|3]
EXECUCAO: [5|4|0|4]
EXECUCAO: [1|0|0|8]
APAGANDO [1|0|0|8]
EXECUCAO: [7|2|3|5]
EXECUCAO: [7|2|3|5]
INSTRUÇÃO STOP ACIONADA!
FIM DA EXECUÇÃO

Info: /OSCI/SystemC: Simulation stopped by user.

Numero de ciclos de clock: 46

Banco de Registradores  Memoria de Instrucoes   Memoria de Dados
[0]: 0                  [0]: 8388645                            [0]: 0
[1]: 1                  [1]: 16777285                           [1]: 1
[2]: 2                  [2]: 25165925                           [2]: 2
[3]: 3                  [3]: 33554565                           [3]: 3
[4]: 4                  [4]: 67108865                           [4]: 4
[5]: 5                  [5]: 16810023                           [5]: 0
[6]: 0                  [6]: 25215047                           [6]: 0
[7]: 0                  [7]: 33620071                           [7]: 0
[8]: 0                  [8]: 41992263                           [8]: 0
[9]: 0                  [9]: 15                         [9]: 0
[10]: 0                 [10]: 0                         [10]: 0
[11]: 0                 [11]: 0                         [11]: 0
[12]: 0                 [12]: 0                         [12]: 0
[13]: 0                 [13]: 0                         [13]: 0
[14]: 0                 [14]: 0                         [14]: 0
[15]: 0                 [15]: 0                         [15]: 0
[16]: 0                 [16]: 0                         [16]: 0
[17]: 0                 [17]: 0                         [17]: 0
[18]: 0                 [18]: 0                         [18]: 0
[19]: 0                 [19]: 0                         [19]: 0
```

Algoritmo 3: Algoritmo feito para realizar operações lógicas (and, not,xor e or) e escrever na memória de dados.
Instruções e execução do algoritmo 3:

algo4.txt

```
1   memset 0 0
2   memset 1 1
3   LD 0 0 0
4   LD 1 0 1
5   NOT 0 0 2
6   NOT 1 0 3
7   AND 0 1 4
8   OR 0 1 5
9   XOR 0 1 6
10  ST 5 0 0
11  ST 6 0 1
12  ST 4 0 2
```

```
       ALL RIGHTS RESERVED
EXECUCAO: [5|0|0|0]
EXECUCAO: [5|0|0|0]
EXECUCAO: [5|1|0|1]
EXECUCAO: [12|0|0|2]
EXECUCAO: [12|1|0|3]
EXECUCAO: [10|0|1|4]
EXECUCAO: [11|0|1|5]
EXECUCAO: [9|0|1|6]
EXECUCAO: [6|5|0|0]
EXECUCAO: [6|6|0|1]
EXECUCAO: [6|4|0|2]
INSTRUÇÃO STOP ACIONADA!
FIM DA EXECUÇÃO

Info: /OSCI/SystemC: Simulation stopped by user.

Numero de ciclos de clock: 59

Banco de Registradores  Memoria de Instrucoes   Memoria de Dados
[0]: 0                  [0]: 5                  [0]: 1
[1]: 1                  [1]: 8388645                    [1]: 1
[2]: -1                 [2]: 16777228                   [2]: 0
[3]: -2                 [3]: 25165868                   [3]: 0
[4]: 0                  [4]: 33570826                   [4]: 0
[5]: 1                  [5]: 41959435                   [5]: 0
[6]: 1                  [6]: 50348041                   [6]: 0
[7]: 0                  [7]: 166                        [7]: 0
[8]: 0                  [8]: 8388806                    [8]: 0
[9]: 0                  [9]: 16777350                   [9]: 0
[10]: 0                 [10]: 15                        [10]: 0
[11]: 0                 [11]: 0                 [11]: 0
[12]: 0                 [12]: 0                 [12]: 0
[13]: 0                 [13]: 0                 [13]: 0
[14]: 0                 [14]: 0                 [14]: 0
[15]: 0                 [15]: 0                 [15]: 0
[16]: 0                 [16]: 0                 [16]: 0
[17]: 0                 [17]: 0                 [17]: 0
[18]: 0                 [18]: 0                 [18]: 0
[19]: 0                 [19]: 0                 [19]: 0
```

Foi requerido um número elevado de ciclos de clocks para executar os algoritmos acima. Isso se deve ao fato das medidas tomadas para controlar os hazards do pipeline. Instruções de leitura de memória e operações lógicas/aritméticas causam um stop na propagação do pipeline.

A forma que resolvemos medir o desempenho para cada instrução foi rodando-a 32 vezes seguidas, computando aproximadamente a quantidade necessária para cada instrução.

Operação	Clock cycles aproximado	32 instruções seguidas
AND	5	169
OR	5	169
XOR	5	169
NOT	5	169
CMP	5	169
ADD	5	169
SUB	5	169
LD	5	169
ST	5	169
J	7	228
JN	7	228
JZ	7	228