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 o suficiente para trabalhar com todos os modos de endereçamento e operações suportadas pelo processador (aritmética, memória de dados e controle).

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

Todas essas instruções são executadas pela ULA do nosso processador.

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.

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): vazio.
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’.

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): vazio.
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): vazio.
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.

O valor armazenado nessas flags será o resultado da última instrução CMP executada pela ULA.

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

OP1 (fonte): vazio.
OP2 (fonte): vazio.
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 está ativa.

OP1 (fonte): vazio.
OP2 (fonte): vazio.
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): vazio.
OP2 (fonte): vazio.
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** - Nas instruções que envolvem registradores, a parte separada para o registrador conterá o seu endereço no qual se encontra o valor a ser trabalhado na operação.

2. **Direct addressing mode** - Na instrução de memória, o endereço de memória será 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. 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 JUMP estiver sendo executada e o salto acontecer. 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 uma instrução influenciar na outra e pela parada e pipeline cleaning quando houver uma instrução de JMP, pela sua dificuldade de executar paralelamente com outras instruções.

O pipeline terá 4 estágios: fetch de instrução, obtenção da instrução (memória de instruções), decodificação da instrução, além da execução combinada com escrita do resultado no banco de registradores (write back).

O pipeline é “travado” até que uma instrução de ULA ou leitura de memória seja finalizada, o que inclui a execução e o write back 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 do 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.
- A atualização é mediante a permissão do controle, o qual enviará sinais booleanos de enable e write.
- **Entradas**: clock.
  - Sinais recebidos de componentes:
  - **Controle (Controle -> Reg. de Pipeline)**:
    - Enable (booleano, ativar componente).
    - Write (booleano, permitir a escrita de novos valores no componente).
  - **Decodificador (Decodificador -> Reg. de Pipeline)**:
    - 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 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 os 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 (valor do seletor).

        - **Memória de dados**:
            - enable_mem_dados (booleano, ativar a memória)
            - write_mem_dados (booleano, permitir a escrita)

        - seletor_mux_mem_dados (sc_uint<2>, selecionar o endereço de entrada).

    - **Registradores de Pipeline:**
        - pipeline_reg_enable (booleano, ativar o componente).
        - pipeline_reg_write (booleano, permitir a escrita de novos valores nos registradores).

    - **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 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 enviados pelo banco de registradores, identificador de operação (opcode) enviado pelo controle.
    - **Saídas:** resultado da operação, 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.
  - **Entradas:**
    - **MUX da memória de dados**:
      - Endereço de memória (leitura ou escrita)
    - **Banco de registradores**:
      - Valor de 32 bits (escrita).
  - **Saída:** valor (leitura).

Diagramas de estados e datapath

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.

Instruções de salto:

A memória de instruções envia a instrução para o decodificador, que parte a instrução em OPCODE e ENDEREÇO DO JUMP, enviando a informação para a unidade de controle, caso seja um salto incondicional, o processador é pausado, o pipeline limpo e o

JUMP é executado, caso contrário, a unidade de controle checa as flags N e Z da ULA, assim decidindo se ocorrerá execução, caso ocorra, é executado o mesmo processo do JUMP incondicional.

3.2 Algoritmos

Algoritmo 1 = Somador de 4 números, usado para testar se a execução está ocorrendo na ordem correta.

```
memset 1 1
memset 2 2
memset 3 3
memset 4 4
LD 1 0 1
LD 2 0 2
LD 3 0 3
LD 4 0 4
ADD 1 2 2
ADD 2 3 3
ADD 3 4 4
```

```
Numero de ciclos: 44
Banco de Registradores:
0 1 3 6 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Memoria de Instrucoes:
8388645 16777285 25165925 33554565 16810023 25215047 33620071 15 0 0 0
0 0 0 0 0 0 0 0 0
Memoria de dados:
0 1 2 3 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Algoritmo 2 = Algoritmo feito para testar o jump incondicional

```
memset 1 1
memset 2 2
memset 3 3
memset 4 4
LD 1 0 1
LD 2 0 2
LD 3 0 3
LD 4 0 4
J 0 0 8
ADD 1 2 2
ADD 2 3 3
```

```
ADD 3 4 4
ADD 2 3 5
```

```
Numero de ciclos: 46
Banco de Registradores:
0 1 2 3 4 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Memoria de Instrucoes:
8388645 16777285 25165925 33554565 67108865 16810023 25215047 33620071
41992263 15 0 0 0 0 0 0 0 0 0 0
Memoria de dados:
0 1 2 3 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Algoritmo 3 = Algoritmo feito para calcular 2^x, onde x é o valor de memória na posição 2 da memória,

```
memset 1 1
memset 2 4
memset 3 1
LD 1 0 1
LD 2 0 2
LD 3 0 3
ADD 1 1 1
SUB 2 3 2
SUB 4 2 5
JN 0 0 3
```

```
Banco de Registradores:
0 32 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Memoria de Instrucoes:
8388645 16777285 25165925 8405031 16826440 41975944 25165827 15 0 0 0 0
Memoria de dados:
0 1 4 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

O resultado ficará na posição 2 do banco de registradores. Feito em 124 clock cycles.

Foi requerido um elevado número 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, enquanto as instruções de jump fazem com que certas instruções sejam descartadas.

3.2 Análise de desempenho

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