Processador RISC-V
Projetos Digitais e Microprocessadores

Selton Miranda Rolim
GRR20235688

I. Introdução

O objetivo do trabalho foi desenvolver um processador de single cycle de 32 bits [1] compatível com um subconjunto específico das instruções RISC-V [2]. Para a validação e testes do projeto, foi fornecido, no enunciado, um código de referência.

II. Metodologia

Inicialmente, foram desenvolvidas e implementadas os componentes básicos que compõem o processador, cada um sendo submetido a testes apropriados para garantir sua funcionalidade. Em seguida, procedeu-se com a criação da unidade de controle e a codificação do *opcode*. Finalmente, o código fornecido foi testado utilizando flags de depuração para verificar a correta operação do processador.

III. Gerador de Imediato

Cada valor do imediato foi processado conforme o formato específico de sua instrução, extraindo os bits individuais desta e realocando-os adequadamente para formar o imediato correspondente. No fim, um multiplexador é utilizado para a correta escolha do imediato, a qual é definida com um sinal da unidade de controle.

IV. Banco de Registradores

O banco de registradores é composto por 32 registradores, com seis entradas: Reg1 , Reg2, RegD, Resultado da ULA, RegWrite, Write Enable e Clock. Entre as 9 saídas, duas são destinadas às saídas de dados RS1out e RS2out, enquanto as demais são exclusivamente para fins de depuração.

V. Unidade de Lógica e Aritmética

A Unidade Lógica e Aritmética recebe três entradas, das quais uma é sinalizada pela unidade de controle para determinar o tipo de operação a ser realizada.

VI. Unidade de Controle

A unidade de controle foi implementada utilizando memórias ROM. Ela recebe o *opcode* como endereço de memória para extrair os sinais de controle necessários para o processador. O conteúdo de cada endereço de memória é composto por 12 bits, que são decodificados da seguinte maneira:

- **Bit 0**: Indica se a operação é uma instrução de branch.
- **Bit 1**: Indica se a operação é uma instrução de jump.
- **Bit 2**: Determina se a operação envolve escrita na memória.
- **Bit 3**: Determina se a operação envolve escrita em um registrador.
- **Bit 4**: Especifica se deve ser selecionado o valor de `RS2out` ou o valor imediato.
- **Bits 5 e 6**: O bit 5 não é diretamente utilizado a partir do conteúdo da memória ROM, sendo fornecido pelo controle do branch. Juntos, esses bits formam o sinal que define o fluxo do programa.
- **Bits 7 e 8**: Definem qual resultado deve ser armazenado no banco de registradores.
- **Bits 9, 10 e 11**: Indicam a construção do valor imediato.

Além disso, existe um sinal de controle separado que determina a operação que deve ser realizada pela Unidade Lógica e Aritmética (ULA). Para isso, foram criadas memórias ROM específicas para cada formato ou variante de instrução. Essas memórias ROM utilizam os campos *funct3* e *funct7* (aplicável somente ao formato R) como endereços de acesso. O *opcode* é utilizado para selecionar qual entrada do multiplexador deve ser retornada.

VII. Datapath

O registrador PC (Program Counter) é, na realidade, implementado como um flip-flop do tipo D, que aponta para o próximo endereço de instrução no programa. Ele recebe como entrada a saída de um multiplexador, que determina o próximo endereço com base no sinal de controle gerado pela unidade de controle. As funcionalidades desse multiplexador são definidas da seguinte maneira:

- **PcSrc = 00:** Incrementa o PC em 4 (PC + 4).
- **PcSrc = 01:** Atualiza o PC com o valor do PC atual somado a um imediato (PC + Imediato).
- **PcSrc = 10:** Define o valor do PC como 0 (PC + 0).
- **PcSrc = 11:** Atualiza o PC com o valor calculado pelo Jump and Link Register (JALR).

Após o PC ser atualizado para apontar para o novo endereço, a instrução correspondente é obtida da memória ROM, onde o programa está armazenado. A instrução é então decomposta em suas partes constituintes: *opcode*, *RD*, *funct3*, *RS1*, *RS2* e *funct7*.

A Unidade Lógica e Aritmética (ULA) possui um multiplexador na entrada *SrcB*, encarregado de selecionar entre o valor do registrador *RS2*, proveniente do banco de registradores, ou um valor imediato.

No estágio final do caminho de dados, são utilizados apenas os primeiros 24 bits do resultado da ULA devido à

limitação do simulador DIGITAL. Um multiplexador adicional determina qual valor deve ser armazenado no banco de registradores, com as seguintes opções:

- **ResultSrc = 00:** O resultado da ULA.
- **ResultSrc = 01:** O valor imediato.
- **ResultSrc = 10:** O valor armazenado na memória RAM.
- **ResultSrc = 11:** O valor do PC incrementado em 4 (PC + 4).

O programa utilizado para testar o processador foi convertido do dump do binário, obtido pelo simulador EGG, para hexadecimal utilizando um script criado especificamente para processar o binário no formato requerido pelo simulador DIGITAL. O script utilizado para essa conversão é apresentado a seguir:

```
#include <stdio.h>

#define DEFAULT_NAME "mod0.bin.hex"

int main(int argc, char **argv)
{
        if (argc < 2) {
                fprintf(stderr, "Uso: %s <arquivo binario>\
                     n", argv[0]);
                return 1;
        }

        FILE *entrada, *saida;
        const char *nomeSaida;
        unsigned char buf[4];
        size_t bytes;

        entrada = fopen(argv[1], "rb");
        if (!entrada) {
                fprintf(stderr, "Nao foi possivel abrir o
                     arquivo %s\n", argv[1]);
                return 1;
        }

        if (argc == 3) nomeSaida = argv[2];
        else nomeSaida = DEFAULT_NAME;

        saida = fopen(nomeSaida, "wb");
        if (!saida) {
                fprintf(stderr, "Nao foi possivel escrever
                     no arquivo %s\n", nomeSaida);
                fclose(entrada);
                return -1;
        }

        fprintf(saida, "v2.0 raw\n");
        while ((bytes = fread(buf, 1, 4, entrada)) > 0) {
                for (size_t i = 0; i < bytes; i++)
                        fprintf(saida, "%02X", buf[bytes -
                             1 - i]);
                fprintf(saida, "\n");

                for (int j = 0; j < 3; j++)
                        fprintf(saida, "0\n");
        }

        fclose(entrada);
        fclose(saida);

        return 0;
}
```

VIII. HALT

Para encerrar a execução do programa, foi empregado o opcode da instrução `ebreak`, que foi codificado de maneira a direcionar a saída do multiplexador do PC para `PC + 0`, impedindo assim a continuidade da execução do programa. Dessa forma, a execução é concluída de maneira controlada, permitindo que o processador pare de buscar e executar novas instruções.

IX. Conclusão

Os requisitos do trabalho foram atendidos, com o processador executando adequadamente o código fornecido no enunciado do projeto.

References

[1] D. A. P. e John L Hennessy, *Computer Organization Design: The Hardware/Software Interface*. Morgan Kaufmann, 2014.

[2] E. Borin, *An Introduction to Assembly Programming with RISC-V*. 2021.