UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
DEPARTAMENTO DE SISTEMAS DE COMPUTAÇÃO

FERNANDO LUCAS VIEIRA SOUZA - 12703069
ARTUR OLIVEIRA ARRAES- 14745532

SSC0108 - PRÁTICAS EM SISTEMAS DIGITAIS

TRABALHO FINAL
8 bit FPGA CPU

SÃO CARLOS
2024

Indice

1. Introdução

Este projeto implementa uma CPU de 8 bits com arquitetura básica, incluindo registradores, barramentos, memória, módulo de E/S (entrada e saída) e unidade lógica e aritmética (ALU). A CPU é projetada para executar instruções de forma síncrona com um clock e suporta um conjunto básico de operações**.**

2. Estrutura do Projeto

A CPU é composta pelos seguintes módulos principais:

1. CPU: Contém a lógica principal, controle dos estados e integração dos outros módulos.
2. ALU: Responsável pelas operações aritméticas e lógicas.
3. Módulo de Entrada/Saída (IO Module): Permite comunicação com dispositivos externos, incluindo switches e LEDs.
4. Memória: Armazena os dados e as instruções.

A comunicação entre os módulos é feita através dos barramentos.

3. Arquitetura do Sistema

CPU (Unidade de Controle e Processamento)

- Registradores:
    - PC (Program Counter): Aponta para o próximo endereço de instrução.
    - IR (Instruction Register): Armazena a instrução atualmente em execução.
    - A, B, R: Registradores gerais e de resultado da ALU.

- Barramentos:

    - Barramento de endereço: Transporta endereços para a memória.
    - Barramento de dados: Transporta dados entre memória, E/S e a CPU.
    - Barramento de controle: Sinaliza operações como leitura, escrita e habilitação de dispositivos.

- Estados da CPU:
    - fetch: Busca a próxima instrução da memória.
    - decode: Decodifica a instrução buscada.
    - execute: Executa a instrução decodificada.
    - wait_read: Espera o ciclo de leitura ser concluído.
    - Outros estados relacionados às instruções específicas.

- **Sinais de Controle:**
    - read, mem_enable, alu_enable, io_input_enable, io_output_enable.

ALU (Unidade Lógica e Aritmética)

- Operações suportadas:
    - Aritméticas: Soma (ADD), subtração (SUB).
    - Lógicas: AND, OR, NOT.
- Entradas:
    - operand_a e operand_b: Operandos para as operações.
    - alu_op: Código da operação a ser realizada.
- Saídas:
    - result: Resultado da operação.
    - zero_flag: Indica se o resultado é zero.

Módulo de Entrada/Saída (IO Module)

- Funções:
    - Lê o estado das entradas externas (SW e BTN).
    - Escreve dados nos LEDs.
- Entradas:
    - SW: Entradas de switches.
    - BTN: Entrada de botão.
- Saídas:
    - LED: Controle de LEDs externos.
    - data_out: Dados lidos do hardware externo.
    - button_status: Indica se o botão foi pressionado.

Memória

- Especificações:
    - Tipo: Memória de 256 palavras de 8 bits.
    - Interface:
        - address: Endereço para leitura/escrita.
        - data: Dados a serem lidos/escritos.
        - wren: Habilitação de escrita.
        - q: Dados lidos.

- Configuração:
    - Inicializada com um arquivo .mif (Memory Initialization File).com o nome “memoria_dados.mif” no diretorio raiz do projeto.

4. Fluxo de Operação da CPU

1. Busca da Instrução:
    - O registrador PC fornece o endereço da próxima instrução.
    - O conteúdo do endereço é carregado no IR.

2. Decodificação:
    - A CPU interpreta o código da operação (opcode) no IR.

3. Execução:
    - Dependendo do opcode, a CPU executa operações que podem incluir:
        - Interação com a memória.
        - Operações na ALU.
        - Comunicação com o módulo de E/S.

4. Incremento do PC:
    - Após a execução, o PC é atualizado para apontar para a próxima instrução.

5. Instruções

Sobre as instruções

As instruções são de 8 bits e os 4 bits mais significativos representam o código da instrução, os 4 bits menos significativos podem conter os parametros especificos da instrução.

Algumas instruções tem parametro na palavra seguinte, por exemplo:

```
LOAD R, 255
```

O endereço 255 não cabe nos 4 bits menos significativos, no entanto, eles são suficientes para indicar o primeiro operando (R). Assim, o endereço 255 fica na próxima instrução e serve como parametro. Quando isso acontece, o PC é incrementado mais uma vez.

Esses detalhes são abstraidos do programador na hora de escrever os programas em assembly, o assembler fará todo o trabalho de conversão.

Detalhamento das instruções

“0000” - NOP

Não faz nada.

“0001” - LOAD Reg1, addr

Carrega o valor do endereço addr em Reg1.

Parametros:

1. Reg1: Registrador A, B ou R
2. addr: endereço em decimal, referente ao endereço do valor que quer ser carregado em Reg1

Observação importante: o endereço deve ser um número **DECIMAL** de 0 a 255.

“0010” - CMP Reg1, Reg1

Compara Reg1 e Reg2, atualizando os registradores de flags. Reg1 e Reg2 em vez de registradores, podem ser a próxima palavra. Se forem iguais, o sinal de controle “equal_flag” vai receber ‘1’, diferentes ‘0’. Se Reg1 é menor que Reg2, o sinal de controle “sign_flag” recebe ‘1’, caso contrário, recebe ‘0’.

Parametros:

1. Reg1: registradores (A, B ou R) ou valor em decimal que deseja comparar
2. Reg2: registradores (A, B ou R) ou valor em decimal que deseja comparar

Obs: não é permitido operações do tipo CMP A, A, onde os dois operandos são os mesmos registradores.

“0011” - JMP addr

Salta incondicionalmente para o endereço que estiver na próxima palavra.
Parametros:

1. addr: endereço para o salto em DECIMAL (estará na palavra seguinte ao código da instrução).

“0100” - JEQ addr

Salta para o endereço da próxima palavra caso o resultado da última comparação seja igual.
Parametros:

1. addr: endereço para o salto em DECIMAL (estará na palavra seguinte ao código da instrução).

“0101” - JGR addr

Salta para o endereço da próxima palavra caso na última comparação o primeiro operando era maior que o segundo.
Parametros:

1. addr: endereço para o salto em DECIMAL (estará na palavra seguinte ao código da instrução).

“0110” - STORE Reg1, addr

Armazena o valor de Reg1 no endereço addr
Parametros:

1. Reg1: registrador A, B ou R
2. addr - será o endereço onde será salvo o valor do registrador. Este endereço estara na palavra seguinte à instrução. O endereço deve ser escrito em DECIMAL.

“0111” - MOV Reg1, Reg2

Move o valor de Reg2 para Reg1. O valor Reg2 não precisa necessariamente estar em um registrador, ele pode estar na palavra seguinte à instrução.

Parametros:

1. Reg1: primeiro registrador (A, B ou R).
2. Reg2: segundo registrador (A, B ou R) ou um valor em decimal

Obs: Não é permitido operações do tipo MOV B, B, ou seja, operações nas quais os dois operandos sejam iguais.

“1000” - ADD | “1001” - SUB | “1010” - AND | “1011” - OR | “1100” - NOT

Essas instruções utilizam a ULA e foram implementadas de maneira similar.

Parametros:

3. Reg1: primeiro registrador (A, B ou R) ou valor da operação em decimal
4. Reg2: segundo registrador (A, B ou R) ou valor da operação em decimal

Depois de selecionados os operandos, eles serão colocados como entrada para a ULA e o resultado será salvo no registrador R.

“1101” - IN Reg1

Lê o valor das chaves do FPGA e, quando o botão for pressionado, armazena em Reg1.
Parametros:

1. Reg1 será o registrador A, B ou R

“1110” - OUT Reg1

Exibe o valor de Reg1 nos LEDs da FPGA.
Parametros:

1. Reg1 será o registrador A, B ou R

“1111” - WAIT

Espera que o botão da FPGA seja pressionado para continuar.

6. Assembler

Foi desenvolvido um assembler compatível com a CPU apresentada. Ele suporta rótulos para referências a endereços de memória, instruções básicas de controle, operações aritméticas/lógicas e comentários no código iniciados com “;”. O assembler inclui verificações de sintaxe e retorna erros explicativos caso ocorram. O assembler vai gerar um arquivo .mif com o programa desejado.

Uso do Assembler

Execução

1. Certifique-se de que o Python está instalado em seu sistema.
2. Salve o código do assembler, ele estará no diretorio “assembler” em um arquivo chamado ***assembler.py***
3. Execute o assembler com o seguinte comando:

```
python assembler.py <arquivo_de_entrada> <arquivo_de_saida>
```

4. Depois que o arquivo .mif foi gerado, mova-o para o diretório “/cpu” e modifique seu nome para “memoria_dados.mif" para ser reconhecido pela CPU. Quando a CPU for iniciada, o programa será carregado na memória automaticamente.

Parâmetros

- `<arquivo_de_entrada>`: Nome do arquivo de entrada contendo o código assembly com a extensão .asm.
- `<arquivo_de_saida>`: Nome do arquivo de saída onde o código de máquina será salvo com a extensão .mif.

Formato do Código Assembly

- Cada linha contém uma instrução ou rótulo.
- Rótulos devem terminar com `:`
- Comentários iniciam com `;` e são ignorados pelo assembler.
- Operandos de instruções são separados por espaços ou vírgulas.

Exemplo:

```
; Exemplo de código assembly
START:
    LOAD A, 10  ; Carrega o valor que está no end 10 no registrador A
    CMP A, B    ; Compara os valores de A e B
    JEQ END     ; Salta para END se A for igual a B
    ADD A, B    ; Soma A com B
END:
    STORE A, 20 ; Armazena o valor de A no endereço 20
    NOP         ; Nenhuma operação
```

Conjunto de Instruções

Instruções Suportadas

As instruções suportadas são as instruções apresentadas anteriormente, porém, aqui está uma tabela resumida das instruções. Caso seja necessário, verifique o detalhamento das instruções na seção anterior.

Instrução	Opcode	Parâmetros	Descrição
NOP	0	-	Nenhuma operação
LOAD	1	Registrador, End.	Carrega valor da memória para registrador
CMP	2	Registrador, Reg.	Compara dois registradores
JMP	3	Endereço	Salta para um endereço
JEQ	4	Endereço	Salta se igual
JGR	5	Endereço	Salta se maior

`STORE`	6	Registrador, End.	Armazena valor do registrador na memória
`MOV`	7	Registrador, Reg.	Move valor entre registradores
`ADD`	8	Registrador, Reg.	Soma dois registradores
`SUB`	9	Registrador, Reg.	Subtrai dois registradores
`AND`	A	Registrador, Reg.	Operação AND entre dois registradores
`OR`	B	Registrador, Reg.	Operação OR entre dois registradores
`NOT`	C	Registrador	Operação NOT em um registrador
`IN`	D	Registrador	Carrega o valor das chaves no registrador
`OUT`	E	Registrador	Exibe o valor do registrador nos LEDs
`WAIT`	F	-	Aguarda o botão ser pressionado.

Processo de Tradução

1. **Primeira Passagem**
    - Processar rótulos e identificar endereços simbólicos.
    - Armazenar rótulos com seus endereços correspondentes.
2. **Geração de Código de Máquina**
    - Traduzir instruções e operandos para código hexadecimal.
    - Substituir rótulos por endereços temporários.
3. **Resolução de Endereços**
    - Ajustar endereços temporários após remoção de rótulos.
    - Substituir endereços temporários por endereços finais.