Relatório: Trabalho Final - MIPS Pipeline

Luize Cunha Duarte - GRR20221232
Gabriel Lisboa Conegero - GRR20221255
Pedro Folloni Pesserl - GRR20220072
Rebeca Soares de Oliveira - GRR20221260
Departamento de Informática
Universidade Federal do Paraná - UFPR
Curitiba, Brasil
lcd22@inf.ufpr.br, glc22@inf.ufpr.br,
pfp22@inf.ufpr.br, rso22@inf.ufpr.br

Resumo

Este relatório documenta o processo de implementação do processador MIPS Pipeline de 16 bits para FPGA (Field Programmable Gate Array) através da linguagem VHDL (VHSIC Hardware Description Language). O projeto do processador possui uma memória de dados e uma memória de instruções de 128 KiB cada, e um banco de registradores com quatro registradores.

1 As instruções.

As divisões dos bits para os endereços e funcionalidades das instruções foi pensada visando evitar bits soltos e manter um espaço considerável para cada funcionalidade. Por isso, as instruções seguem a base do MIPS, tendo sua divisão reorganizada para 16 bits.

- Tipo R (opcode: 3 bits; rs: 2 bits; rt: 2 bits; rd: 2 bits; shamt: 4 bits; func: 3 bits).

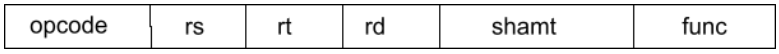

Figura 1: Formato das instruções do tipo R.

- Tipo I. (opcode: 3 bits; rs: 2 bits; rt: 2 bits; imediato: 9 bits).

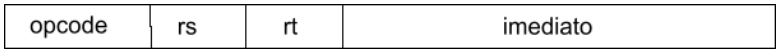

Figura 2: Formato das instruções do tipo I.

As instruções suportadas são:

- ADD
- AND
- OR
- NOR
- Set Less Than
- Shift Left Logical
- Shift Right Logical
- ADDI
- ANDI
- Display
- ORI
- Branch On Equal
- Load Word
- Store Word

2 A arquitetura e sua implementação.

Buscando uma arquitetura que suportasse todas as instruções decididas, sua organização deu-se da maneira representada abaixo. Nesse sentido, cada componente foi implementado utilizando a linguagem VHDL.

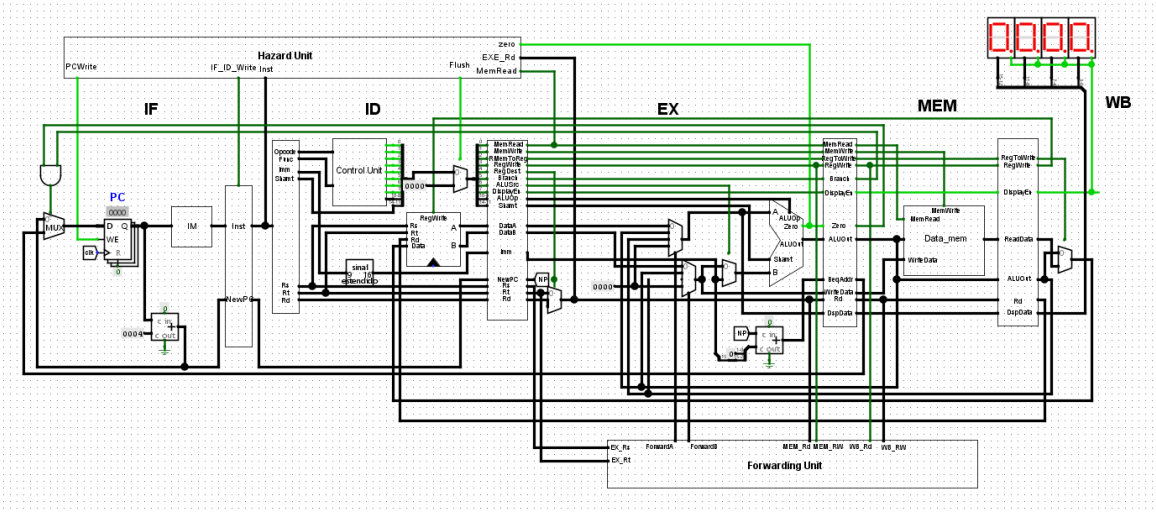

Figura 3: Bloco operacional do MIPS Pipeline.

3 Os estágios e componentes.

1. **Instruction Fetch**.

   Sendo a primeira parte da execução de uma instrução, é onde o ponteiro para a memória de instruções (Instruction Memory) é calculado e ela é lida, guardando tal instrução e sua localização no registrador intermediário IF_ID. O PC pode ser atualizado para a próxima instrução, somando um*, ou para o endereço do Branch.

   *OBS: Em VHDL, a memória de instruções foi implementada como uma array de vetores de 16 bits; portanto, ela é indexada pela palavra e não pelo byte. Dessa forma, a próxima posição do vetor é simplesmente a posição atual adicionada de 1.

2. **Instruction Decode**.

   Após lida, a instrução será decodificada, gerando seus sinais de controle através da Control Unit, a qual recebe os três primeiros bits e retorna os sinais necessários ligados. Além disso, os registradores usados para realizar a operação desejada são lidos no Register Bank, que possui quatro registradores de 16 bits de capacidade, sendo o primeiro sempre 0. Por fim, tem-se também nesse estágio a Hazarding Unit, a qual reconhece se a instrução sendo executada na proxima etapa irá desviar o fluxo de instruções, zerando as instruções que não serão mais executadas, um processo denominado "flush". No flush, os sinais de controle nos registradores IF_ID, ID_EX e EX_MEM são zerados, efetivamente descartando as três instruções anteriores ao branch que causou um desvio no código.

3. **Execute**.

   Tendo os dados necessários para realizar a operação, nessa etapa temos a ULA, controlada pela ALUControl, a qual verifica se é uma instrução do tipo R para realizar o "func" — últimos três bits da instrução — ou então uma soma, subtração, AND ou OR.

Nessa parte, também é calculado o endereço do Branch On Equal, bastando somar ao novo PC o imediato estendido com sinal, passado pelos registradores intermediários. Por fim, um componente importante para o desempenho do pipeline, a Forwarding Unit, é implementado nessa etapa. Esse componente é responsável por evitar stalls no processador ao receber dados de etapas seguintes, não sendo necessária a espera da escrita nos registradores — detalhada mais à frente — das instruções anteriores.

4. **Memory**.

   O único componente presente é a memória de dados (Memory Data), o componente mais lento do Pipeline. Usada apenas nas funções de Load Word e Store Word, a memória possui 512 Bytes de armazenamento com 16 bits cada linha. Por fim, os dados carregados são passados para o registrador intermediário MEM_WB.

5. **Write-Back**.

   Sendo a última etapa do MIPS Pipeline, ela guarda no banco de registradores o dado de saída da ALU ou a palavra carregada da Memory Data. Também é nesse estágio que o display acessa o registrador que foi determinado na instrução `dsp` e decodifica seu conteúdo para ligar os segmentos necessários no componente.

4 O clock.

O Pipeline opera a uma frequência diferente do FPGA, tendo um clock mais lento que facilite a visualização de seu funcionamento. Para garantir isso, foi criado um componente para calcular o novo clock que será utilizado por todo o circuito — o divisor de clock.

5 O programa de teste.

Foi escrito um programa simples como teste das funcionalidades do MIPS Pipeline. O programa calcula os primeiros números da sequência de Fibonacci, e a lógica interna cria sinais de saída para displays de sete segmentos, para cada dígito dos resultados (que são armazenados em registradores no processador).

6 Como compilar e carregar o software no FPGA.

O MIPS foi desenvolvido usando a plataforma Xilinx ISE, e seu código foi compilado para o FPGA Digilent Nexys 3. Para carregar o código na placa, é necessário primeiro baixar a plataforma e compilá-lo. As instruções para instalação do Xilinx ISE e do software Digilent Adept (usado para carregar o código) estão descritas no arquivo `install.md`, entregue no pacote `pipeline-vhdl.tar.gz`.

Após instalados, deve-se abrir um novo projeto na ISE, com as configurações do FPGA utilizado (no caso, *Family*: Spartan6, *Device*: XC6SLX16, *Package*: CSG324), carregar todos os arquivos-fonte do pipeline e realizar o mapeamento das suas portas de E/S — isso é feito também pelo Xilinx ISE, da seguinte forma:

- Página *Design* → *User Constraints* → *Create Timing Constraints*;
- Se for importado o arquivo `Pipeline.ucf`, o mapeamento das portas de E/S já estará feito. Caso contrário, é necessário realizá-lo por meio de *I/O Pin Planning* — o programa Plan-Ahead será aberto;
- *Implement Design*;

- *Generate Programming File*: vários arquivos serão gerados no diretório do projeto. O arquivo `Pipeline.bit` é o arquivo a ser carregado no FPGA.

Depois de gerado o arquivo `.bit` necessário, será utilizado o pacote Digilent Adept para carregá-lo na placa. Após conectar o FPGA no computador por um cabo USB, em um terminal do Linux, use os seguintes comandos:

`djtgcfg enum` → Listar as placas reconhecidas pela porta USB;

`djtgcfg init -d Nexys3` → Inicializar o dispositivo Nexys3;

`djtgcfg prog -i 0 -d Nexys3 -f Pipeline.bit` → Carregar o arquivo `Pipeline.bit` no dispositivo Nexys3.

Com o arquivo carregado, o FPGA começará a executar o programa.