Relatório Projeto Final
Processador MIPS Uniciclo em VHDL

Ian Nery Bandeira, 17/0144739
Guilherme de Oliveira Silva, 11/0012496
Éden de Lucas C. V. Medeiros, 12/0151898

Dep. Ciência da Computação - Universidade de Brasília (UNB)
CiC 116394 - Organização e Arquitetura de Computadores - Turma C

Objetivo:

O projeto da disciplina consiste no desenvolvimento de uma versão do processador MIPS Uniciclo em FPGA, utilizando a linguagem VHDL. A plataforma de desenvolvimento é Altera. Foi utilizado o kit de desenvolvimento DE2-35. As ferramentas utilizadas para o desenvolvimento foram o Quartus II e ModelSimAltera.

Descrição:

A arquitetura utilizada para implementação do processador MIPS uniciclo foi utilizando primordialmente a linguagem de descrição de hardware VHDL, e diagrama de blocos para implementar os módulos utilizados para unir o processador com as saídas do display de 7 segmentos presentes na placa DE2-35. Segue uma descrição detalhada de cada módulo utilizado no desenvolvimento do trabalho:

➔ Arquivos Não-vhdl.

- ***Decoder7.v:*** O arquivo tem a função de receber 4 bits de entrada e retornar 7 de saída, utilizando a lógica de decodificação em 7 segmentos, sendo os números possíveis, de 0 a F.
- ***Processador_mips_final.bdf:*** O arquivo tem a função de unir o bloco do processador, feito em VHDL, com os decodificadores de 7 segmentos, mencionados acima.

➔ Arquivos vhdl.

- ***Mips_pkg.vhd:*** O arquivo tem a função de unir todas as entidades presentes nos arquivos VHDL do projeto, com o intuito de modularizar os processos feitos mais de uma vez. Além disso, este contém as definições de operações da ULA utilizando a lógica de label.
- ***Processador.vhd:*** Este arquivo é o módulo principal dos arquivos VHDL do processador, sendo sua função a de aplicar e interligar os módulos do processador entre si, e fazer o contato input-output com a placa.
    - ***Fetch.vhd:*** Este é o arquivo que contém o registrador PC (Program Counter), a memória de instrução e um mux que determina em qual registrador será escrito o resultado do *write back*, rt ou rd.
        - ***Pc.vhd:*** O arquivo contém um registrador simples, regido unicamente pela subida do clock.
        - ***Memoria_instrucoes.vhd:*** O arquivo contém uma ROM, regida por um clock específico (Utilizado na implementação o CLOCK_50 da placa). Retorna a instrução determinada pelo arquivo instrucoes.mif, e já é dividida em *opcode, rs, rt, rd, shamt, func, k16* e *k26.*
    - ***Bregula.vhd:*** Este arquivo contém o banco de registradores; o controle da ULA, alguns multiplexadores pontuais para execução de instruções como LUI, por exemplo; e a ULA.
        - ***Breg.vhd:*** Este é o arquivo do banco de registradores, o qual possui 32 registradores de 32 bits cada; sendo o registrador 0 o único que não pode ser mudado, possuindo valor fixo zero.
        - ***C_ula.vhd:*** O arquivo é o controle da ULA. Tal controle utiliza das saídas *op_ula,* proveniente do módulo de controle principal; e do *func* proveniente da memória de instruções, caso este seja necessário (instruções do tipo R). Sua saída *ctr_ula* repassa à ULA a função cuja a qual ela executará.
        - ***Ula.vhd:*** O arquivo possui quatro entradas, sendo elas A, B as que recebem os valores para operação; ulop sendo o valor recebido do *ctr_ula* proveniente do arquivo ***c_ula.vhd*** e define qual operação será executada; e shamt, que recebe o sinal do módulo ***fetch.vhd.*** Suas saídas são *aluout*, para o resultado da operação; *zero*, para caso a operação tenha dado 0; e *ovfl,* o qual é 1 caso ocorra overflow nas operações de ADD, ADDI e SUB.
    - ***Controle.vhd:*** O módulo de controle recebe o *opcode* proveniente da memória de instruções, e repassa *op_ula* para o controle da ULA; e diversos sinais que controlam os multiplexadores presentes no

processador, ou enable de outros módulos. Os sinais presentes serão explicados abaixo:

- ***Reg_dst:*** *Se o registrador de escrita é rt ou rd.*
- ***Orig_alu:*** *Se a segunda entrada na ula vem do imediato ou não.*
- ***Mem_para_reg:*** *O valor que vem da memória de dados para se escrita no registrador.*
- ***Escreve_reg:*** *Permite escrever na memória de registradores.*
- ***Le_mem:*** *Permite a leitura da memória.*
- ***Escreve_mem:*** *Permite a escrita na memória.*
- ***Branch:*** *Ligado caso haja uma instrução de branch equal.*
- ***Branchn:*** *Ligado caso haja uma instrução de branch not equal.*
- ***Jump:*** *Ligado caso seja instruções de jump.*
- ***Jal:*** *Ligado caso seja instrucao de link.*
- ***Luictr:*** *Ligado caso seja instrucao de lui.*
- ***Jerror:*** *Ligado caso ocorra um exceção de overflow.*

- ***Mem_final.vhd:*** O módulo possui a memória de dados, que é uma RAM 1-port, e o multiplexador de *write back*, que determina o dado que será escrito nos registradores, caso seja escrito algo.
- ***Mux_two_to_one.vhd:*** O módulo possui a funcionalidade de um MUX 2 para 1 de 32 bits.
- ***Mux_two_to_one_5.vhd:*** O módulo possui a funcionalidade de um MUX 2 para 1 de 5 bits.
- ***Mux_four_to_one.vhd:*** O módulo possui a funcionalidade de um MUX 4 para 1 de 32 bits.
- ***Somador.vhd:*** O módulo possui funcionalidade de um somador simples. Utilizado para as operações de PC + 4 e PC + 4 + Endereço de branch.
- ***Epc.vhd:*** O módulo consiste de um registrador simples de 32 bits, porém com um enable, este que vem do controle. Tem a funcionalidade de guardar o PC + 4 para reproduzí-lo após a rotina de tratameto do overflow.

Testes e resultados:

O teste realizado foi proveniente de um arquivo disponibilizado pelo professor, o qual testa todas as funções implementadas no processador. O resultado final foi satisfatório, visto que todas as instruções implementadas funcionaram corretamente. Ocorreram alguns erros durante o desenvolvimento; sendo aqueles que conseguimos resolver: Erros em multiplexadores, visto a sua complexidade de abstração em ver se as entradas e saídas estão certas; e erros nos sinais de controle, que também possuem certa complexidade por lidar com diversos sinais. Existem alguns

erros que não conseguimos identificar sua procedência, e portanto não conseguimos resolver. Eles são: Erro de PC que às vezes volta para o endereço 0x00000000 quando deveria ir de 0x00000018 para 0x0000001c, porém o problema só ocorre em certas ocasiões, com nenhum padrão aparente. O mesmo ocorre para a ligação de alguns LED's enquanto o programa ocorre; os quais ligam e desligam sem qualquer razão ou padrão ao passar as instruções.

Antes de testar na FPGA, utilizamos um arquivo ***Waveform.wvf*** para debugar esses problemas pontuais.

Conclusão:

Após os diversos problemas enfrentados na implementação, visto que há uma complexidade grande ao trabalhar com os diversos sinais do controle e com os mux's; conseguimos, após muito tempo, uma fluência na linguagem de VHDL e um grande conhecimento do processador uniciclo, tanto na forma abstrata de escrita em VHDL quanto em sua representação gráfica, visto que sempre ao implementar alguma função não-trivial do processador (LUI, JAL, JR, etc) tívemos que desenhar o processador inteiro nos quadros brancos do LINF e do aquário do CiC. Portanto, o trabalho nos auxiliou tanto na prova 2 quanto no melhor entendimento sobre a matéria como um todo.