Universidade de Brasília
INSTITUTO DE CIÊNCIAS EXATAS – IE
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO – CIC

Disciplina: CIC 116394 – Organização e Arquitetura de Computadores – Turma C
Semestre: 1/2017
Professor: Marcelo Grandi Mandelli

PROJETO FINAL – MIPS UNICICLO EM VHDL

OBJETIVOS

- Compreender o funcionamento e o projeto de um processador;
- Compreender o processo de desenvolvimento e simulação de projetos de hardware utilizando a linguagem VHDL;
- Compreender o funcionamento e prototipação em FPGAs.

ESPECIFICAÇÃO DO TRABALHO

Este trabalho consiste na implementação de um processador MIPS Uniciclo (parte operativa e parte de controle) em FPGA, utilizando a linguagem VHDL. Para isso, serão utilizadas as ferramentas Quartus II e Modelsim-Altera, além de um kit de desenvolvimento FPGA da Altera. Para esse trabalho podem ser utilizados tanto o kit DE2-35 (EP2C35F672C6), assim como o kit DE2-70 (EP2C70F896C6). O simulador MARS deverá ser utilizado, necessariamente, para a obtenção do segmento de código e do segmento de dados de programas assembly a serem testados no processador implementado.

Tamanho de palavra

O processador MIPS Uniciclo a ser implementado suportará palavras de 32 bits. Assim, a parte operativa desse processador será de 32 bits, ou seja, os dados armazenados em memória, os registradores, a ULA, os somadores, as instruções e as conexões utilizam 32 bits.

Tamanho de Memória

Devido às restrições de memória das placas FPGA, as memórias de instruções e dados devem ser dimensionadas para suportas pequenos programas de teste. As memórias de instruções e dados a serem implementadas nesse trabalho utilizarão apenas 7 bits de endereço, de forma a prover um espaço de endereçamento de 128 palavras. Ou seja, as memórias de instruções e dados armazenarão 128 palavras de 32 bits cada

Instruções suportadas

Como mencionado anteriormente O simulador MARS deverá ser utilizado, necessariamente, para a obtenção do segmento de código e do segmento de programas assembly a serem testados no processador implementado. Dessa forma, todas as instruções a serem implementadas devem seguir o mesmo formato gerado pelo simulador MARS, ou seja, o formato tradicional do processador MIPS.
O processador MIPS Uniciclo a ser desenvolvido neste trabalho deve suportar as seguintes instruções: **LW, SW, ADD, ADDI, SUB, AND, ANDI, OR, ORI, NOR, XOR, XORI, SLT, SLTI, SLL, SRL, SRA, J, JR, JAL, BEQ, BNE**

Módulos

A implementação do processador MIPS Uniciclo em VHDL consiste na implementação de cada módulo presente no processador e sua interligação através de sinais. Os principais módulos que deverão ser implementados incluem:

- **PC:** contador de programa. É um registrador de 32 bits. Entretanto, pelas restrições de memória adotadas, apenas o número de bits necessário (7 bits) deve ser utilizado como endereço da memória de instruções, sendo o restante ignorado.

- **Memória de Instruções (MI):** memória ROM que armazena o código a ser executado pelo processador. As instruções são de 32 bits. O espaço de endereçamento é reduzido (7 bits). Cada endereço da memória armazena uma instrução de 32 bits. Idealmente, a memória deve funcionar como um bloco combinacional para leitura, ou seja, necessita-se apenas do endereço para ler a instrução/dado, sem sinais adicionais de controle. Essa memória não permite o endereçamento a byte. Desta forma, se forem utilizados 7 bits de endereço, deve-se utilizar os bits 2 a 8 do PC como endereço de instrução.

- **Memória de Dados (MD):** armazena dados que podem ser lidos e/ou escritos na memória. Endereços de 7 bits e dados de 32 bits. A memória é escrita na subida do relógio, quando o sinal de controle EscreveMem estiver acionado. O sinal de leitura LeMem não é estritamente necessário, faz com que o conteúdo da posição de memória endereçada seja colocado na saída de dados. Se não houver sinal LeMem, basta fornecer o endereço que o dado é lido, de forma similar à ROM.

- **Banco de Registradores (BREG):** é constituído por 32 registradores de 32 bits. O registrador índice zero, BREG[0], é uma constante. Sua leitura retorna sempre zero, e não pode ser escrito. O BREG tem duas entradas de endereços, permitindo a leitura de 2 registradores de forma simultânea. Uma terceira entrada de endereço é utilizada para selecionar um registrador para escrita de dados. A escrita de um dado em registrador ocorre na transição de subida do relógio.

- **Unidade Lógico-Aritmética (ULA):** opera sobre dados de 32 bits. Provê o resultado em 32 bits, juntamente com o sinal ZERO, que indica que o resultado da operação realizada é zero. Operações implementadas na ULA: ADD, SUB, AND, OR, XOR, SLT, NOR, SLL, SRL, SRA (ou seja, a ULA do Trabalho 3 pode ser utilizada sem modificações)

- **Multiplexadores:** multiplexadores de entradas de 32 bits e saída de 32 bits deverão ser utilizados na implementação.

- **Somadores:** são utilizados 2 somadores de 32 bits para operar com endereços.

- **<u>Outros módulos não citados podem fazer parte da implementação!</u>**

A Figura 1 apresenta um exemplo de diagrama esquemático da arquitetura do MIPS Uniciclo. **O diagrama apresentado na Figura 1 não contém a implementação de todas as instruções pedidas nesse trabalho. Assim, essa Figura não apresenta necessariamente todos os módulos necessários para a implementação pedida nesse trabalho e só serve de exemplo de implementação.**

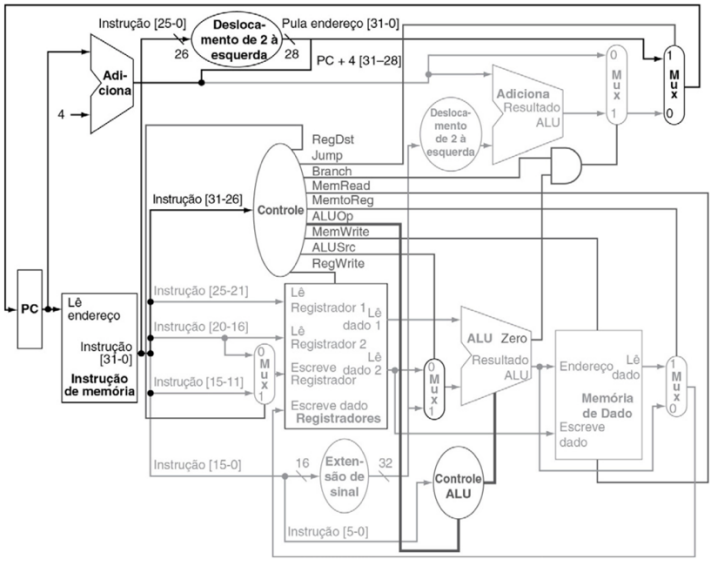

Figura 1 – MIPS Uniciclo

Geração do segmento de código e de dados de um programa assembly

O processador MIPS Uniciclo a ser desenvolvido nesse trabalho contará com uma memória de instruções e outra de dados. Essas memórias devem ser inicializadas com o segmento de código (memória de instruções) e segmento de dados (memória de dados) de um programa assembly para que a execução desse programa seja possível no processador.

Para isso, o simulador MARS deverá ser utilizado, necessariamente, para a obtenção do segmento de código e do segmento de dados do programa assembly. Estes segmentos são obtidos primeiramente configurando o MARS através da opção:

Settings→Memory Configuration, opção Compact, Text at Address 0

Utilizando essa opção o segmento de código (*Text Segment*) desse programa começa no endereço 0x00000000, e o segmento de dados (*Data Segment*) começa na posição 0x00002000.

Depois disso monta-se o programa através da opção *Run→Assemble* ou F3.

A obtenção do segmento de código e do segmento de dados desse programa é efetuada através da opção:

File -> Dump Memory...

Para o salvamento do segmento de código selecione:

Memory segment: *.text (0x00000000 - <endereço_final_da_memorória>)*

Dump Format: *Hexadecimal Text*

Clique em *Dump To File...* e salve o arquivo com extensão **.txt**. Sugestão, salve como <programa>*_text.txt*.

Para o salvamento do segmento de dados selecione:

Memory segment: *.data (0x00002000 - <endereço_final_da_memorória>)*

Dump Format: *Hexadecimal Text*

Clique em *Dump To File...* e salve o arquivo com extensão **.txt.** Sugestão, salve como <programa>*_data.txt*.

Os arquivos gerados apresentarão as instruções e os dados de um programa em formato hexadecimal. O conteúdo de cada arquivo deve ser copiado para a respectiva memória para inicializa-la.

Um exemplo de implementação de memória será fornecido juntamente com o enunciado desse trabalho. Essa memória pode ser na Figura 2**.**

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3. use IEEE.NUMERIC_STD.ALL;
4.
5. entity memory is
6.   generic(N: integer := 7; M: integer := 32);
7.   port(
8.                   clk:  in  STD_LOGIC := '0';
9.                   we:   in  STD_LOGIC := '0';
10.                  adr:  in  STD_LOGIC_VECTOR(N-1 downto 0) := (others => '0');
11.                  din:  in  STD_LOGIC_VECTOR(M-1 downto 0) := (others => '0');
12.                  dout: out STD_LOGIC_VECTOR(M-1 downto 0)
13.           );
14. end;
15.
16. architecture memory_arch of memory is
17.
18.    type mem_array is array(0 to (2**N-1)) of STD_LOGIC_VECTOR(M-1 downto 0);
19.    signal mem: mem_array := (
20.                                          x"abababab",
21.                                          x"efefefef",
22.                                          x"02146545",
23.                                          x"85781546",
24.                                          x"69782314",
25.                                          x"25459789",
26.                                          x"245a65c5",
27.                                          x"ac5b4b5b",
28.                                          x"ebebebeb",
29.                                          x"cacacaca",
30.                                          x"ecececec",
31.                                          x"facfcafc",
32.                                          x"ecaecaaa",
33.                                          x"dadadeac",
34.                                          others => (others => '1')
35.                                    );
36.
37. begin
38.
39.    process(clk) begin
40.
41.           if clk'event and clk='1' then
42.                  if we='1' then
43.                         mem(to_integer(unsigned(adr))) <= din;
44.                  end if;
45.           end if;
46.
47.    end process;
48.
49.    dout <= mem(to_integer(unsigned(adr)));
50.
51. end;
```

Figura 2 - Exemplo de implementação de uma memória em VHDL

No código da Figura 2 a memória é inicializada entre as linhas 20 e 34. Por exemplo, a linha 20 inicializa o valor do endereço 0 da memória com o valor hexadecimal “abababab”. Os 14 primeiros endereços de memória (de 0 a 13) estão sendo inicializados com valores específicos (entre as linhas 20 e 33). A linha 34 inicializa as demais posições de memória com ‘1’, ou seja, cada palavra a partir do endereço 14 de memória conterá o valor hexadecimal “ffffffff”.

Interface da FPGA

O processador MIPS Uniciclo deverá ser prototipado em um FPGA da família Altera. Como mencionado anteriormente, podem ser utilizados tanto o kit DE2-35 (EP2C35F672C6), assim como o kit DE2-70 (EP2C70F896C6). As chaves, botões e os displays de 7 segmentos das plataformas FPGA deverão ser utilizados como interface de operação e debug do processador implementado.

A interface criada na FPGA deverá ser a seguinte:

- um botão (sugere-se a KEY3 para DE2-35 ou iKEY3 para DE2-70) da plataforma FPGA deverá ser utilizado para geração do clock do processador;
- um conjunto de 2 chaves (sugere-se as chaves SW[0..1] para DE2-35 ou iSW[0..1] para DE2-70) da plataforma FPGA definirão o valor apresentado nos 8 displays de 7 segmentos em hexadecimal. O valor apresentado poderá ser:
    - o valor de PC, quando as chaves apresentarem o valor 00;
    - a instrução definida pelo valor de PC, quando as chaves apresentarem o valor 01;
    - um registrador, definido por um conjunto de 5 chaves (sugere-se as chaves SW[2..6] para DE2-35 ou iSW[2..6] para DE2-70), quando as chaves apresentarem o valor 10;
    - o conteúdo de um endereço da memória de dados, quando as chaves apresentarem o valor 11. O endereço da memória de dados será definido por um conjunto de 7 chaves da plataforma FPGA (sugere-se as chaves SW[7..13] para DE2-35 ou iSW[7..13] para DE2-70).

GRUPOS

O trabalho deverá ser realizado em grupos de até 5 alunos.

ENTREGA E APRESENTAÇÃO

O processador MIPS Uniciclo implementado na FPGA deverá ser apresentado ao professor até o dia 06/07/2017, impreterivelmente! Trabalhos não apresentados até essa data receberão nota 0!

Também deverão ser entregues:

- **código VHDL comentado** (só diga o que cada módulo faz, não precisa comentar todas linhas)
- **testbench comentado** – o testbench será utilizado para simular o programa assembly a ser testado no trabalho
- **relatório contendo:**
    - a) resultados da simulação (imagens) mostrando o funcionamento de um programa assembly de teste de todas as instruções.
    - b) explicação de como as instruções JAL e JR foram implementadas.

Entregar um arquivo compactado em formato .zip no moodle (aprender.unb.br) até às 23h55 do dia 05/07/2017

O nome do arquivo deve conter as matrículas dos integrantes do grupo:
matriculaaluno1_matriculaaluno2_matriculaaluno3.zip

CRITÉRIOS DE AVALIAÇÃO

Funcionamento do processador MIPS Uniciclo em uma plataforma FPGA, executando corretamente programas assembly definidos pelo professor – 60%

Códigos VHDL, testbench e relatório – 40% da nota

Esta especificação pode ser atualizada para se efetuar correções de texto ou alterações para se deixar mais claro o que está sendo pedido.

Caso a especificação sofrer uma atualização, os alunos serão informados via moodle (aprender.unb.br).

Última atualização: 08/06/2017 às 12:35