Universidade de Brasília

Instituto de Ciências Exatas – IE / Departamento de Ciência da Computação – CIC
Organização e Arquitetura de Computadores – Turma C – 2019/2
Prof. Marcelo Grandi Mandelli

PROJETO FINAL – IMPLEMENTAÇÃO RISC-V UNICICLO

OBJETIVOS

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

ESPECIFICAÇÃO DO TRABALHO

Este trabalho consiste na implementação de um processador RISC-V Uniciclo (unidade operativa e unidade de controle) em FPGA, utilizando uma linguagem de descrição de hardware. Para isso, serão utilizadas as ferramentas Quartus II e Modelsim, 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 RARS 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 RISC-V Uniciclo a ser implementado suportará palavras de 32 bits. Assim, a unidade 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 suportar 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. Como somente implementaremos as instruções lw e sw de acesso à memória, as memórias de instruções e dados nesse trabalho serão endereçadas a *word* (palavras de 32 bits), ou seja, cada endereço é uma word.

Instruções suportadas

O processador RISC-V Uniciclo a ser desenvolvido neste trabalho deve suportar as seguintes instruções da ISA RV32I:

- **tipo-R: ADD, SUB, SLL, SLT, SLTU, XOR, SRL, SRA, OR e AND**
- **tipo-I: ADDI, SLTI, SLTIU, XORI, ORI, ANDI, SLLI, SRLI, SRAI, LW, JALR**
- **tipo-S: SW**
- **tipo-B: BEQ, BNE, BLT, BGE, BLTU, BGEU**
- **tipo-J: JAL**
- **tipo-U: LUI, AUIPC**

Módulos

A implementação do processador RISC-V Uniciclo em linguagem de descrição de hardware 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). Como mencionado anteriormente, a MI será endereçada a *word*, ou seja, 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 de um endereço para ler a instrução, sem sinais adicionais de controle. A MI deve possuir uma entrada de endereço de leitura que acionará duas saídas de leitura: uma para leitura da instrução a ser utilizada no processador e outra para mostrar essa instrução nos displays de 7 segmentos da placa (como definido será definido a seguir neste trabalho). **Dica:** utilizando-se 7 bits de endereço, deve-se utilizar somente os bits 8 a 2 do PC como endereço de instrução (`PC(8 downto 2)` em VHDL).

- **Memória de Dados (MD):** armazena dados que podem ser lidos e/ou escritos na memória. A MD possui endereços de 7 bits e dados de 32 bits. Da mesma forma que a MI, essa memória será endereçada a *word*, ou seja, cada endereço da memória armazena um dado de tamanho 32 bits. A MD é escrita na borda de subida de clock, quando o sinal de controle EscreveMem estiver ativo (em 1). O sinal de leitura LeMem não é estritamente necessário, fazendo 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 a uma ROM. A MD deve possuir duas entradas de endereço de leitura: uma para a obtenção do dado a ser lido e utilizado na implementação do processador e outra para mostrar o dado de uma determinada posição de memória nos displays de 7 segmentos da placa (como definido será definido a seguir neste trabalho). **Dicas:** utilizando-se 7 bits de endereço, deve-se utilizar somente os bits 8 a 2 do endereço de entrada dessa memória. Ainda, como veremos mais adiante, o acesso à MD se dará partir do endereço 0x00002000 de memória. Desta forma, o endereço definido na MD deve ser subtraído de 0x00002000 para que a leitura seja correta.

- **Banco de Registradores (BREG):** é constituído por 32 registradores de 32 bits. O registrador de índice 0 não pode ser escrito, e quando lido sempre retorna 0. O BREG terá três entradas de endereços de leitura, permitindo a leitura de 3 registradores de forma simultânea. Dois desses registradores lidos são utilizados na implementação do processador. O terceiro registrador lido terá seu valor mostrado nos displays de 7 segmentos da placa (como definido será definido a seguir neste trabalho). Uma quarta 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. A ULA do Trabalho 2 pode ser utilizada.

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

- **Somadores:** são utilizados somadores de 32 bits deverão ser utilizados na implementação.

- Outros módulos não citados podem fazer parte da implementação!

A Figura 1 apresenta um exemplo de diagrama esquemático da implementação do RISC-V 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. Por exemplo, a Figura 1 não implementa as instruções JALR, LUI e AUIPC.

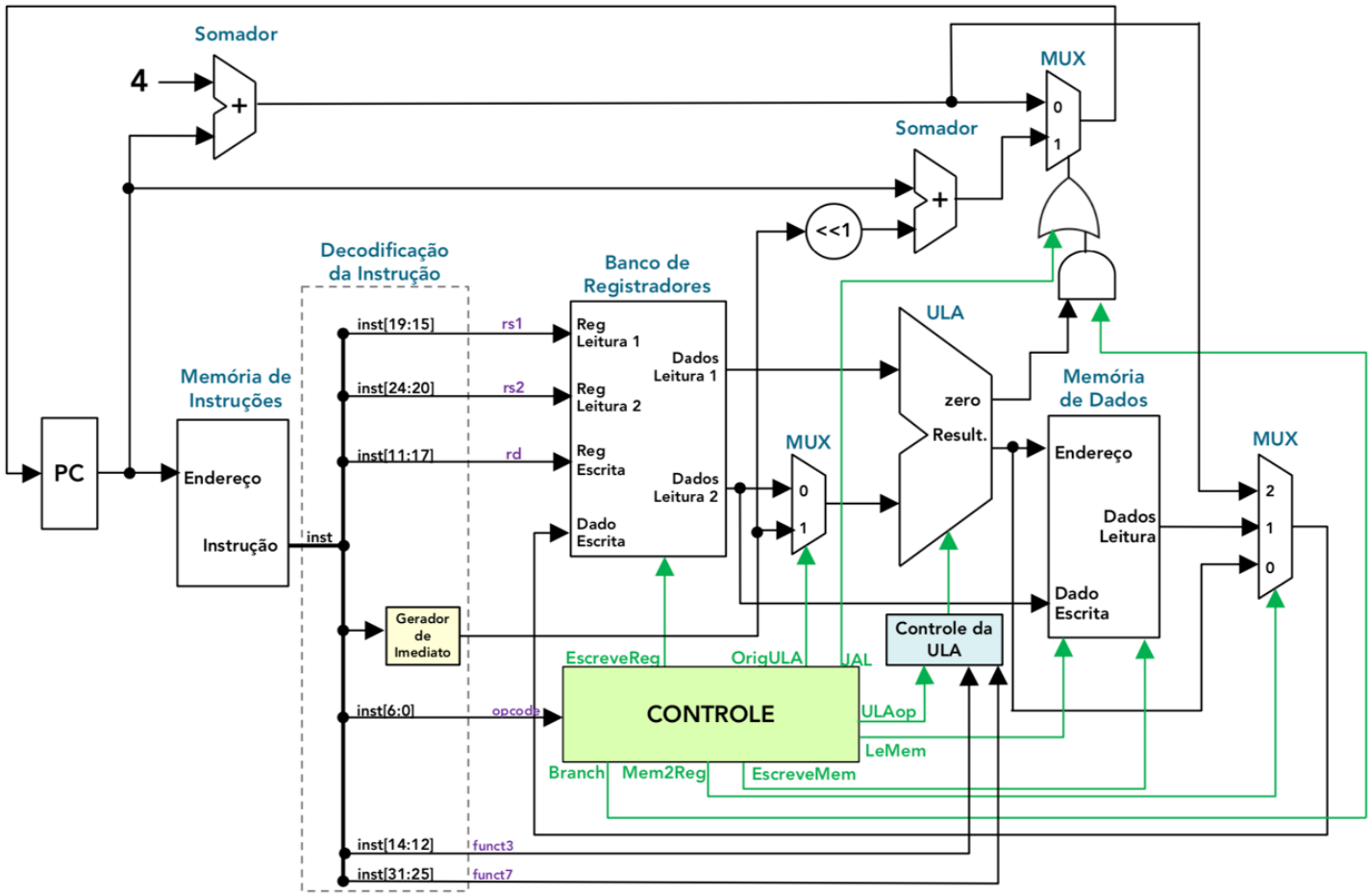

Figura 1 – Exemplo de RISC-V Uniciclo

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

O processador RISC-V 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 RARS 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 RARS 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 *<nome_do_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 *<nome_do_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 inicializá-la.

Um exemplo de implementação de memória de instruções será fornecido juntamente com o enunciado desse trabalho e pode ser visto no Código 1. Essa implementação apresenta, como descrito acima, uma entrada de endereço `address`, a qual aciona duas saídas de leitura: `inst`, a qual é usada para se ler a instrução usada na implementação do processador; e `inst_board`, o qual será utilizado para mostrar a instrução executada em determinado instante no processador nos displays de 7 segmentos do kit FPGA.

No Código 1, a memória é inicializada entre as linhas 18 e 32. Por exemplo, a linha 18 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 18 e 31). A linha 32 inicializa as demais posições de memória com ‘1’, ou seja, cada palavra a partir do endereço 14 de memória conterá o valor hexadecimal “ffffffff”.

```
1. library ieee;
2. use ieee.std_logic_1164.ALL;
3. use ieee.numeric_std.ALL;
4.
5. entity imem is
6.         generic(N: integer := 7; M: integer := 32);
7.       port(
8.                address          : in  std_logic_vector(N-1 downto 0);
9.                inst             : out std_logic_vector(M-1 downto 0);
10.               inst_board       : out std_logic_vector(M-1 downto 0)
11.      );
12. end;
13.
14. architecture imem_arch of imem is
15.
16.      type imem_array is array(0 to ((2**N)-1)) of STD_LOGIC_VECTOR(M-1 downto 0);
17.      signal i_mem: imem_array := (
18.                                          x"abababab",
19.                                          x"efefefef",
20.                                          x"02146545",
21.                                          x"85781546",
22.                                          x"69782314",
23.                                          x"25459789",
24.                                          x"245a65c5",
25.                                          x"ac5b4b5b",
26.                                          x"ebebebeb",
27.                                          x"cacacaca",
28.                                          x"ecececec",
29.                                          x"facfcafc",
30.                                          x"ecaecaaa",
31.                                          x"dadadeac",
32.                                          others => (others => '1')
33.                                    );
34. begin
35.
36.      inst <= i_mem(to_integer(unsigned(address))) when to_integer(unsigned(address))<((2**N)-1) else (others => '0');
37.
38.      inst_board <= i_mem(to_integer(unsigned(address))) when to_integer(unsigned(address))<((2**N)-1) else (others => '0');
39.
40. end;
```

Código 1 - Exemplo de implementação da memória de instruções em VHDL

Interface da FPGA

O processador RISC-V Uniciclo deverá ser prototipado em um kit 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) da plataforma FPGA deverá ser utilizado para geração do clock do processador;
- um conjunto de 2 chaves (sugere-se as chaves SW[1..0]) da plataforma FPGA definirão o valor apresentado nos 8 displays de 7 segmentos em hexadecimal. O valor apresentado deverá 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[6..2]), 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[13..7]).

ENTREGA E APRESENTAÇÃO

O processador RISC-V Uniciclo implementado na FPGA deverá ser apresentado ao professor, na sala do mesmo, no dia 05/12/2019 no horário de aula, 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 JALR, LUI e AUIPC foram implementadas.

Entregar um arquivo compactado em formato .zip no moodle (aprender.unb.br) até às 23h55 do dia 05/12/2019

GRUPOS

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

CRITÉRIOS DE AVALIAÇÃO

Funcionamento do processador RISC-V Uniciclo em uma plataforma FPGA, executando corretamente programas assembly definidos pelo professor – 70%

Códigos VHDL, testbench e relatório – 30% 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: 12/11/2019 às 11:40