PCS3225 - Sistemas Digitais II

por Bruno Albertini e Edson S. Gomi

20/06/2022

Este arquivo contém o enunciado dos projetos 2 a 5 de Sistemas Digitais 2, que consistem na construção escalonada do processador PoliLEGv8, uma implementação do LEGv8 do livro texto.

Introdução

Um processador pode ser visto como um projeto que utiliza o paradigma de unidade de controle e fluxo de dados. A unidade de controle é responsável pelo ciclo de busca, decodificação, execução e gravação dos dados, ciclo este que rege o funcionamento de um computador de uso geral pela sua característica programável. Já o fluxo de dados é composto por elementos de memória (e.g. banco de registradores), componentes de controle de fluxo (quase sempre combinatórios) e unidades funcionais.

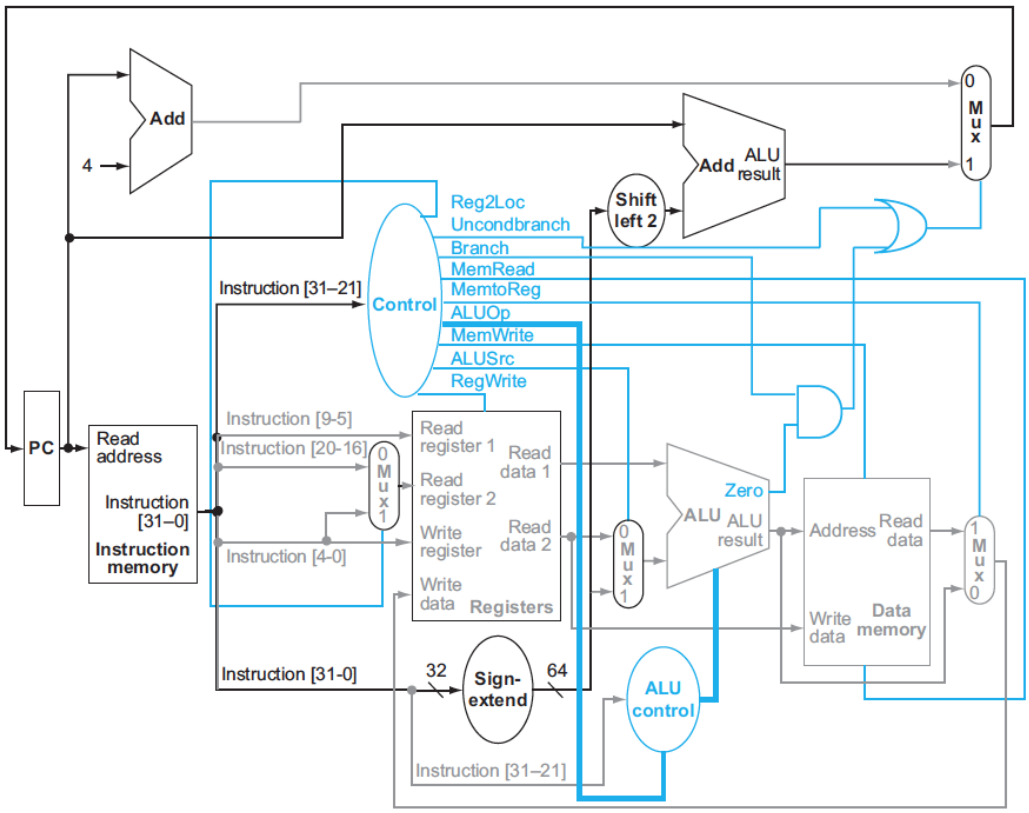

Figura 1: Arquitetura LEGv8, com o fluxo de dados (preto) e controle (azul).

O processador PoliLEG, que implementaremos e simularemos em VHDL nos próximos exercícios de simulação, é um processador monociclo. Isso significa que ele é projetado para executar uma única

instrução em um único ciclo de *clock*, seguindo essa regra para todas as instruções que suporta. Para conseguir este feito e manter sua implementação simples, é necessário usar uma arquitetura com memórias separadas para instrução (IM) e dados (DM). Isto se deve pois, em cada ciclo, sempre haverá uma leitura na IM, possivelmente concomitantemente com uma leitura ou escrita na DM. Como o programa não é alterável, usaremos uma **ROM** para a IM e uma **RAM** para a DM.

Este tipo de arquitetura é chamada de arquitetura Harvard.

Instrução	Formato	opcode	Sintaxe/Descrição
LDUR	D	11111000010	LDUR Rt, [Rn + address] Rt = Memory[Rn + address]
STUR	D	11111000000	STUR Rt, [Rn + address] Memory[Rn + address] = Rt
CBZ	CB	10110100	CBZ Rt, address if (Rt == 0) goto (PC + address)
B	B	000101	B address goto (PC + address))
ADD	R	10001011000	ADD Rd, Rn, Rm Rd = Rn + Rm
SUB	R	11001011000	SUB Rd, Rn, Rm Rd = Rn - Rm
AND	R	10001010000	AND Rd, Rn, Rm Rd = Rn AND Rm (bit a bit)
ORR	R	10101010000	ORR Rd, Rn, Rm Rd = Rn OR Rm (bit a bit)

Tabela 1: Instruções suportadas pelo PoliLEG monociclo.

O cálculo computacional é realizado nas unidades funcionais, portanto não é exagero afirmar que são os componentes principais de um processador. É possível construir máquinas com um ou até mesmo sem registradores, mas uma máquina sem uma unidade funcional simplesmente não realiza computação alguma. Como as unidades funcionais são muito comuns, é costume juntar as principais funções computacionais em uma única unidade, que chamamos de ULA (**Unidade Lógica e Aritmética**). Como o próprio nome diz, uma ULA reúne em um único componente operações lógicas (e.g. AND, OR, XOR, etc.) e aritméticas (adição, subtração, etc.), além de operações de comparação (menor, maior, igual, etc).

ALU, *Arithmetic and Logic Unit*.

Outro componente muito importante do fluxo de dados é registrador, pois é o elemento de memória mais próximo das unidades funcionais e também o mais rápido, por estar diretamente ligado às unidades funcionais através do fluxo de dados interno. Todas as operações importantes que acontecem no processador tem como origem ou destino um registrador, normalmente dentro de um **banco de registradores**.

No LEGv8 são 32 de 64 bits

Por último, a **Unidade de Controle** da implementação monociclo é completamente combinatória, pois todas as instruções devem ser executadas completamente em um único ciclo de *clock*.

Atividades

Este enunciado serve para os projetos 2 a 5 da disciplina. Atente-se pois cada Exercício Projeto (EP) tem uma data de abertura, conforme indicadas na Tabela 2.

Data Abertura	Data Encerramento	Projeto
29 de Maio (00:00)	2 de julho (23:59)	EP2
5 de junho (00:00)	2 de julho (23:59)	EP3
12 de junho (00:00)	2 de julho (23:59)	EP4
19 de junho (00:00)	2 de julho (23:59)	EP5

Tabela 2: Cronograma de Entregas dos Projetos

As submissões de cada projeto (de 2 a 5) serão liberadas no Juiz em semanas diferentes. Todos os projetos têm a mesma data e hora de encerramento. Após a data de encerramento o não será possível enviar sua solução. Recomendamos que faça os projetos de maneira incremental, distribuindo o trabalho ao longo das semanas indicadas. Não deixe para fazer todos os projetos em cima da hora.

Siga as recomendações padrões de envio para o juiz presente no enunciado do EP1. A nota de cada atividade de cada EP está especificada no enunciado.

Lembre-se que para tirar dúvidas é necessário anotar o número da sua submissão e também enviar o seu *testbench*. Sugerimos que faça o seu próprio *testbench* para cada unidade desenvolvida.

Exercício Projeto 2 - Memórias

Peso deste EP: 1

EP2A1 Projete uma ROM assíncrona, conforme a Listagem 1. O conteúdo da memória deve estar no próprio arquivo VHDL e deve ser o da Tabela 3.

2 pontos, 8 envios, maior nota.

Listagem 1: Entidade para a memória ROM.

```
entity rom_simples is
  port (
    -- 4 bits de endereco:
    addr: in bit_vector(3 downto 0);
    -- 8 bits de tamanho de palavra de dados:
    data: out bit_vector(7 downto 0)
  );
end rom_simples;
```

EP2A2 Projete uma ROM assíncrona, sem atraso, parametrizável no tamanho e na largura, conforme a Listagem 2. O conteúdo da memória é dado por arquivo `dat`, cujo nome é um dos parâmetros. O arquivo `dat` é um arquivo com uma palavra de memória por linha, em binário representado em ASCII. Exemplo: para uma memória de

3 pontos, 8 envios, maior nota.

Endereço (dec)	Conteúdo (bin)
0	`00000000`
1	`00000011`
2	`11000000`
3	`00001100`
4	`00110000`
5	`01010101`
6	`10101010`
7	`11111111`
8	`11100000`
9	`11100111`
10	`00000111`
11	`00011000`
12	`11000011`
13	`00111100`
14	`11110000`
15	`00001111`

Tabela 3: Conteúdo da ROM simples.

tamanho 2 e largura de 4, o arquivo teria duas linhas, contendo dois valores de 4b cada: `0101` e `1010`, por exemplo.

Listagem 2: Entidade para a memória ROM com carga externa.

```
entity rom is
  generic (
    addr_s : natural := 64; -- Size in bits
    word_s : natural := 32; -- Width in bits
    init_f : string  := "rom.dat"
  );
  port (
    addr : in  bit_vector(addr_s-1 downto 0);
    data : out bit_vector(word_s-1 downto 0)
  );
end rom;
```

EP2A3 Projete uma RAM síncrona, sem atraso, parametrizável no tamanho e na largura, conforme a Listagem 3. O conteúdo inicial da memória é dado pelo arquivo `dat`, que tem o mesmo formato que o da ROM. Esta memória faz uma leitura imediatamente quanto o sinal `rd` está habilitado (em alto) e faz uma escrita na próxima borda de subida do *clock* se e somente se o sinal `wr` estiver habilitado. O endereço para leitura e escrita é compartilhado, porém há uma porta de dados para entrada e outra para saída.

5 pontos, 8 envios, maior nota.

Listagem 3: Entidade para a memória RAM.

```
entity ram is
  generic (
    addr_s : natural := 64; -- Size in bits
    word_s : natural := 32; -- Width in bits
    init_f : string  := "ram.dat"
```

```
  );
  port (
    ck      : in  bit;
    rd, wr  : in  bit; -- enables (read and write)
    addr    : in  bit_vector(addr_s-1 downto 0);
    data_i  : in  bit_vector(word_s-1 downto 0);
    data_o  : out bit_vector(word_s-1 downto 0)
  );
end ram;
```

Exercício Projeto 3 - Fluxo de Dados

Peso deste EP: 2

EP3A1 Projete o deslocador de dois bits para a esquerda usado para o cálculo do endereço de salto, totalmente combinatório. A entidade está na Listagem 4.

1 ponto, 8 envios, maior nota.

Listagem 4: Entidade para o deslocador.

```
entity shiftleft2 is
  generic(
    ws: natural := 64); -- word size
  port(
    i: in  bit_vector(ws-1 downto 0); -- input
    o: out bit_vector(ws-1 downto 0)  -- output
  );
end shiftleft2;
```

EP3A2 Projete o extensor de sinal. O extensor de sinal é sensível à instrução. Sua entrada é sempre de 32b (uma instrução) e a saída 64b, que corresponde ao imediato contido na instrução já com sinal estendido. A entidade está na Listagem 5.

3 pontos, 8 envios, maior nota.

Listagem 5: Entidade para o extensor de sinal.

```
entity signExtend is
  port(
    i: in   bit_vector(31 downto 0); -- input
    o: out  bit_vector(63 downto 0)  -- output
  );
end signExtend;
```

EP3A3 Projete o Unidade Lógica e Aritmética (ULA) do PoliLEGv8 monociclo. A entidade a ser seguida está na Listagem 6. A ULA é totalmente combinatória e a arquitetura interna está a seu critério. As operações são sempre em complemento de base e as operações possíveis são as constantes na Tabela 4. A operação *Set on Less Than* (SLT) retorna 1 (decimal) na saída se e somente se A<B.

3 pontos, 8 envios, maior nota.

Sugerimos fortemente que leia os apêndices do livro texto, que contém um guia para a implementação escalonada desta ULA, partindo de uma ULA de 1 bit.

Listagem 6: Entidade para a ULA.

```
entity alu is
  generic (
    size : natural := 64
  );
  port (
    A, B : in  bit_vector(size-1 downto 0); -- inputs
    F    : out bit_vector(size-1 downto 0); -- output
    S    : in  bit_vector(3 downto 0); -- op selection
    Z    : out bit; -- zero flag
    Ov   : out bit; -- overflow flag
    Co   : out bit -- carry out
    );
end entity alu;
```

OP	Unidade	Descrição
0000	AND	$A\&B$, bit a bit
0001	OR	$A\|B$, bit a bit
0010	adição	$A + B$
0110	subtração	$A - B$
0111	SLT	$A < B$, Set on Less Than
1100	NOR	$\overline{A\|B}$, bit a bit

Tabela 4: Operações que podem ser realizadas pela ULA. O campo OP pode ser interpretado como $(Op_3Op_2Op_1Op_0)$, onde: Op_3 inverte A, Op_2 inverte B, e Op_1Op_0 escolhe a função entre: 00:AND, 01:OR, 10:+, 11:less.

EP3A4 Projete um banco de registradores parametrizável conforme a entidade da Listagem 7. O número de registradores presentes no banco é dado pelo parâmetro reg_n, sempre maior que 1, e o último registrador não aceita escrita e retorna sempre zero para leituras. O registrador a ser lido é indicado pelas portas rr1 e rr2, cuja saídas estarão imediatamente nas portas q1 e q1 respectivamente. O registrador a ser escrito é indicado pela porta wr e o dado a ser escrito é amostrado na porta d, porém a escrita só acontece na borda de subida do *clock*, se e somente se o sinal regWrite estiver ativo (alto). O *reset* é assíncrono para todos os registradores do banco.

3 pontos, 8 envios, maior nota.

Listagem 7: Entidade para o banco de registradores.

```
entity regfile is
  generic(
    reg_n:  natural := 10;
    word_s: natural := 64
  );
  port(
    clock:         in    bit;
    reset:         in    bit;
    regWrite:      in    bit;
    rr1, rr2, wr:  in    bit_vector(natural(ceil(log2(real(reg_n))))-1 downto 0);
    d:             in    bit_vector(word_s-1 downto 0);
    q1, q2:        out   bit_vector(word_s-1 downto 0)
  );
end regfile;
```

Exercício Projeto 4 - Fluxo de Dados e Unidade de Controle

Peso deste EP: 3
5 pontos, 8 envios, maior nota.

EP4A1 Junte os componentes do EP anterior para formar o fluxo de dados da Figura 1. O fluxo de dados não contém as memórias, que são externas ao processador. A entidade para o fluxo de dados está na Listagem 8. Note que o sinal `aluCtrl` está ligado diretamente na ULA.

Listagem 8: Entidade para o fluxo de dados.

```
entity datapath is
  port(
    -- Common
    clock : in bit;
    reset : in bit;
    -- From Control Unit
    reg2loc : in bit;
    pcsrc: in bit;
    memToReg: in bit;
    aluCtrl: in bit_vector(3 downto 0);
    aluSrc: in bit;
    regWrite: in bit;
    -- To Control Unit
    opcode: out bit_vector(10 downto 0);
    zero:   out bit;
    -- IM interface
    imAddr: out bit_vector(63 downto 0);
    imOut:  in  bit_vector(31 downto 0);
    -- DM interface
    dmAddr: out bit_vector(63 downto 0);
    dmIn:   out bit_vector(63 downto 0);
    dmOut:  in  bit_vector(63 downto 0)
  );
end entity datapath;
```

EP4A2 Projete a Unidade de Controle do processador, considerando que é monociclo e as instruções suportadas são as da Tabela 1. A entidade para a UC está na Listagem 9.

5 pontos, 8 envios, maior nota.

Listagem 9: Entidade para a unidade de controle.

```
entity controlunit is
  port (
    -- To Datapath
    reg2loc : out bit;
    uncondBranch : out bit;
    branch: out bit;
    memRead: out bit;
    memToReg: out bit;
    aluOp: out bit_vector(1 downto 0);
    memWrite: out bit;
    aluSrc: out bit;
    regWrite: out bit;
    -- From Datapath
    opcode: in bit_vector(10 downto 0)
  );
end entity;
```

Exercício Projeto 5 - PoliLEGv8

Peso deste EP: 2

EP5A1 Junte o fluxo de dados e a unidade de controle para formar o processador completo, cuja entidade está na Listagem 10.

6 pontos, 8 envios, maior nota.

Listagem 10: Entidade para o processador completo.

```
entity polilegsc is
  port (
    clock, reset: in bit;
    -- Data Memory
    dmem_addr: out bit_vector(63 downto 0);
    dmem_dati: out bit_vector(63 downto 0);
    dmem_dato: in  bit_vector(63 downto 0);
    dmem_we:   out bit;
    -- Instruction Memory
    imem_addr: out bit_vector(63 downto 0);
    imem_data: in  bit_vector(31 downto 0)
  );
end entity;
```

EP5A1 Adapte sua memória ROM para uma ROM pré carregada que respeite a entidade da Listagem 11. Considere que o processador começa, após o reset, a buscar instruções a partir do endereço zero desta memória, e o espaço de endereçamento do processador é de palavras de 8 bits. Sendo assim após o endereço `0x0`, o processador buscará a próxima instrução no endereço `0x4`, que corresponde à segunda palavra desta memória, depois para o endereço `0x8` correspondente à terceira palavra e assim sucessivamente. Esta memória então é capaz de suportar 2^8 instruções e preenche um espaço de $2^8 * 4$ bytes (no nosso processador este é o único espaço de endereçamento possível para instruções).

4 pontos, 8 envios, maior nota.

Listagem 11: ROM com o programa solução.

```
  entity rom is
  port (
    addr : in  bit_vector(7 downto 0);
    data : out bit_vector(31 downto 0)
  );
end rom;
```

Sua memória deve vir pré carregada com um programa para o PoliLEGv8, usando as instruções da Tabela 1. As instruções são ordenadas usando *big-endian*. O programa que você deve montar é uma solução para o MDC (Máximo Divisor Comum). Os parâmetros estarão na RAM, pré-carregados.

Assim como a ROM, a memória RAM que é fornecida com o exercício é idêntica à sua RAM do EP2A3. Os endereços são para um

espaço de endereçamento de palavras em bytes, porém a memória é alinhada em 64b (8B) pois o processador foi especificado desta forma. Portanto, o primeiro dado de 64b está no endereço `0x0`, o segundo no `0x8` e assim por diante.

No endereço `0x0` foi pré carregado um valor constante correspondente ao maior valor negativo em complemento de dois representável nesta arquitetura. No endereço `0x8` há o primeiro parâmetro para o algoritmo MDC, chamado de A. No endereço seguinte (próxima palavra de dado) está o segundo parâmetro, chamado de B. Ambos os parâmetros são de 64b e representam inteiros positivos em complemento de dois. O seu programa deve ler A e B, calcular o MDC entre ambos e gravar o resultado no endereço zero, onde está a constante. Não deve existir outra gravação na memória exceto para gravar o resultado, portanto use com cautela o registradores disponíveis.

O PoliLEGv8 que o juiz utiliza para esta atividade é o do professor, portanto mesmo que não tenha realizado os EPs anteriores na sua completude, ainda é possível realizar este EP.

Referências

[1] D.A. Patterson and J.L. Hennessy. *Computer Organization and Design ARM Edition: The Hardware Software Interface*. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 2016.