PCS3225 - Sistemas Digitais II
Atividade Formativa 12 - Projeto do Processador PoliLEGv8 em VHDL
Parte 3 - Projeto Integrado do Processador PoliLEGv8 Monociclo

Antonio Vieira da Silva Neto
(Agradecimentos de Anos Anteriores: Edson Toshimi Midorikawa e Bruno Abrantes Basseto)

Data do Arquivo: 22/11/2025; Prazo da AF12: 13/12/2025

Introdução

A Atividade Formativa 12 (AF12) de PCS3225 baseia-se na implementação e na simulação do processador PoliLEGv8 em VHDL. Ao se utilizar um método estruturado de projeto de Engenharia, aderente a um ciclo de vida em V, parte-se de uma **especificação** sobre as características do processador e de sua **arquitetura** - ambas vistas em aula - para projetá-lo. Nas duas primeiras partes do projeto, foram projetados os componentes basais utilizados no Fluxo de Dados do processador PoliLEGv8. O objetivo da terceira parte é promover a integração desses componentes e construir um protótipo do processador PoliLEGv8 monociclo com base em três blocos: (i.) o Fluxo de Dados, (ii.) a Unidade de Controle e (iii.) a entidade topo (*top entity*) que os integra. Ao final, o processador terá a estrutura interna representada no diagrama da Figura 1 e poderá executar programas que contenham as instruções da Figura 2, que possuem os *opcodes* da Figura 3. Nas instruções do tipo R, **shamt = 000000**, e nas instruções do tipo D, **Op2 = 00**.

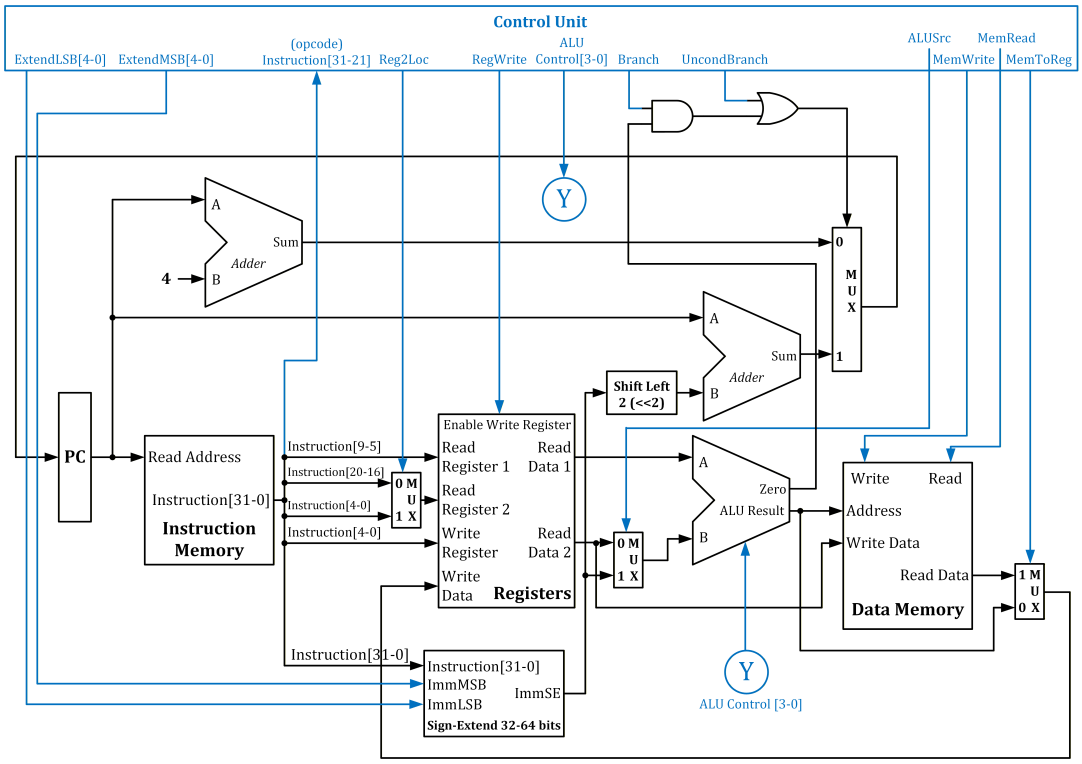

Figura 1: Processador PoliLEGv8 - Versão Monociclo

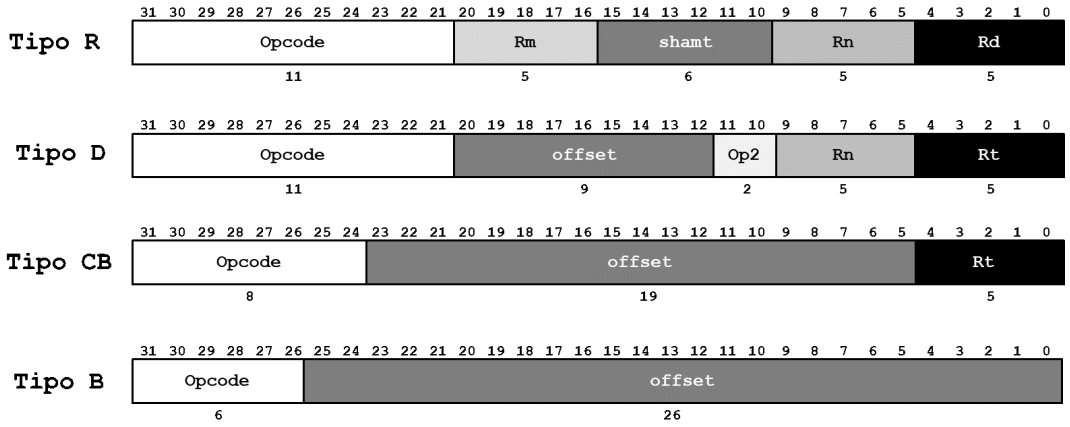

Figura 2: Processador PoliLEGv8 - Instruções

Instrução	*Opcode*
ADD	10001011000
SUB	11001011000
AND	10001010000
ORR	10101010000
LDUR	11111000010
STUR	11111000000
CBZ	10110100
B	000101

Figura 3: Processador PoliLEGv8 - *Opcodes* das Instruções

Para esta parte da Atividade Formativa 12 (AF12), as seguintes determinações devem ser seguidas:

1. As interfaces de entrada e saída do Fluxo de Dados, da Unidade de Controle e da entidade topo, bem como eventuais técnicas de projeto indicadas no enunciado, devem ser respeitadas;

2. Deve-se desenvolver uma bancada de testes, utilizando os conteúdos das memórias de instrução e dados fornecidas no e-Disciplinas junto com o enunciado, para realizar o teste do processador projetado;

3. O projeto e a verificação funcional devem ser documentados na última parte do relato cumulativo produzido ao longo do projeto. Nesse relato, recomenda-se criar um capítulo subdividido em cinco seções: (i.) descrição funcional do projeto do Fluxo de Dados, (ii.) descrição funcional da Unidade de Controle, (iii.) descrição funcional da entidade topo, (iv.) descrição da bancada de testes e (v.) apresentação e análise de conformidade dos resultados obtidos. Cada seção não deve ocupar mais do que três páginas do relato.

Bloco 1 - Fluxo de Dados

Implemente um componente em VHDL correspondente ao Fluxo de Dados do processador PoliLEGv8 monociclo. Ele deve respeitar a seguinte entidade:

```
entity fluxoDados is
    port(
        clock        : in bit;                      -- entrada de clock
        reset        : in bit;                      -- clear assincrono
        extendMSB    : in bit_vector (4 downto 0);  -- sinal de controle sign-extend
        extendLSB    : in bit_vector (4 downto 0);  -- sinal de controle sign-extend
        reg2Loc      : in bit;                      -- sinal de controle MUX Read Register 2
        regWrite     : in bit;                      -- sinal de controle Write Register
        aluSrc       : in bit;                      -- sinal de controle MUX entrada B ULA
        alu_control  : in bit_vector (3 downto 0);  -- sinal de controle da ULA
        branch       : in bit;                      -- sinal de controle desvio condicional
        uncondBranch : in bit;                      -- sinal de controle desvio incondicional
        memRead      : in bit;                      -- sinal de controle leitura RAM dados
        memWrite     : in bit;                      -- sinal de controle escrita RAM dados
        memToReg     : in bit;                      -- sinal de controle MUX Write Data
        opcode       : out bit_vector (10 downto 0) -- sinal de condição código da instrução
    );
end entity fluxoDados;
```

Note que, com exceção dos sinais de *reset* e *clock*, respectivamente necessários para iniciar os registradores do processador em zero e para cadenciar a operação do processador, todas as demais interfaces do Fluxo de Dados devem ser conectadas à Unidade de Controle.

O sinal *opcode* deve sempre corresponder aos 11 bits mais significativos de cada instrução - mesmo que a instrução tenha menos bits de *opcode*, como *CBZ* e *B* (veja a Figura 3).

Os sinais *extendMSB* e *extendLSB* devem indicar, nas instruções que manipulam imediatos (LDUR, STUR, CBZ e B), a posição do bit mais significativo e do bit menos significativo em que o imediato está posicionado na instrução. Por exemplo, pela Figura 2, verificam-se os seguintes valores - que devem ser gerados pela Unidade de Controle (ver Bloco 2 a seguir):

- LDUR e STUR: *extendMSB* = 20 e *extendLSB* = 12;
- CBZ: *extendMSB* = 23 e *extendLSB* = 5;
- B: *extendMSB* = 25 e *extendLSB* = 0.

O sinal *alu_control* deve controlar a operação da ULA de acordo com a lógica da Figura 4. Essa lógica deve ser implementada pela Unidade de Controle (ver Bloco 2 a seguir).

***Opcode*[10-0]**	***ALU control*[3-0]**	**Significado**
11111000010	0010	**Soma (operação LDUR)**
11111000000	0010	**Soma (operação STUR)**
10110100**XXX**	**XX**11	**Repassa entrada B "Pass B" (operação CBZ)**
10001011000	0010	**Soma (operação ADD)**
11001011000	0110	**Subtração (operação SUB)**
10001010000	0000	**"AND" bit a bit (operação "AND")**
10101010000	0001	**"OR" bit a bit (operação "ORR")**

Figura 4: Processador Poli-LEGv8 - *ALU Control*

Todos os demais sinais têm um único bit e são utilizados para controlar multiplexadores (reg2loc, aluSrc, branch, uncondBranch e memToReg), para controlar as operações de leitura e escrita da memória de dados (memRead e memWrite) e para controlar a escrita em um registrador do banco de registradores (regWrite). Eles têm lógica positiva e devem ser aderentes às conexões apresentadas na Figura 1.

Seu projeto deve prever a construção do Fluxo de Dados utilizando os componentes já projetados nas Partes 1 e 2 da AF12:

- O Contador de Programas (*Program Counter* - PC) deve ser uma instância do componente **reg** (Parte 1) parametrizada com n = 7. A largura de seu conteúdo é compatível com a quantidade de endereços da memória de instruções cujo conteúdo é fornecido para testes no e-Disciplinas;
- A Memória de Instruções (*Instruction Memory*) deve ser uma instância do componente **memoriaInstrucoes** (Parte 1) parametrizada com addressSize = 7, dataSize = 8 e datFileName = "memInstrPolilegv8.dat";
- O Banco de Registradores (*Registers*) deve ser uma instância do componente **regfile** (Parte 2);
- O Extensor de Sinal (*Sign-Extend 32-64 bits* deve ser uma instância do componente **sign_extend** (Parte 1) parametrizada com dataISize = 32, dataOSize = 64 e dataMaxPosition = 5;
- A Unidade Lógica e Aritmética (*Arithmetic and Logic Unit* - ALU/ULA) deve ser uma instância do componente **ula** (Parte 2);
- Cada um dos multiplexadores da Figura 1 deve ser uma instância do componente **mux_n** (Parte 1) parametrizada com dataSize = 64;
- A Memória de Dados (*Data Memory*) deve ser uma instância do componente **memoriaDados** (Parte 1) parametrizada com addressSize = 7, dataSize = 8 e datFileName = "memDadosInicialPolilegv8.dat";
- Cada um dos somadores da Figura 1 deve ser uma instância do componente **adder_n** parametrizada com dataSize = 64;
- O bloco de multiplicação por quatro ('*Shift Left* («2)') deve ser uma instância do componente **two_left_shifts** (Parte 1) parametrizada com dataSize = 64;
- Por simplicidade, a lógica combinatória que associa a saída *zero* da ULA com os sinais de controle *branch* e *uncondBranch* deve ser implementada no Fluxo de Dados por meio de uma expressão lógica de atribuição;
- Todas as conexões mostradas na Figura 1 devem ser realizadas com o auxílio de sinais (*signals*) que interligam os terminais dos componentes envolvidos.

Bloco 2 - Unidade de Controle

Implemente um componente em VHDL correspondente à Unidade de Controle do processador PoliLEGv8 monociclo. Ele deve respeitar a seguinte entidade:

```
entity unidadeControle is
    port(
        opcode       : in bit_vector (10 downto 0); -- sinal de condição código da instrução
        extendMSB    : out bit_vector (4 downto 0); -- sinal de controle sign-extend
        extendLSB    : out bit_vector (4 downto 0); -- sinal de controle sign-extend
        reg2Loc      : out bit;                     -- sinal de controle MUX Read Register 2
        regWrite     : out bit;                     -- sinal de controle Write Register
        aluSrc       : out bit;                     -- sinal de controle MUX entrada B ULA
        alu_control  : out bit_vector (3 downto 0); -- sinal de controle da ULA
        branch       : out bit;                     -- sinal de controle desvio condicional
        uncondBranch : out bit;                     -- sinal de controle desvio incondicional
        memRead      : out bit;                     -- sinal de controle leitura RAM dados
        memWrite     : out bit;                     -- sinal de controle escrita RAM dados
        memToReg     : out bit                      -- sinal de controle MUX Write Data
    );
end entity unidadeControle;
```

A Unidade de Controle deve implementar a 'tabela' da Figura 5, que estabelece os valores de todos os sinais de controle em função dos *opcodes* da Figura 3. Indicações de valores 'X' na Figura 5 podem ser indiscriminadamente substituídas por '0' ou '1' em seu projeto, de forma a viabilizar o uso dos tipos *bit* e *bit_vector*.

		ADD	SUB	AND	ORR	LDUR	STUR	CBZ	B
Sinais de Condição (Entradas) *"opcode"*	opcode[10]	1	1	1	1	1	1	1	0
	opcode [9]	0	1	0	0	1	1	0	0
	opcode [8]	0	0	0	1	1	1	1	0
	opcode [7]	0	0	0	0	1	1	1	1
	opcode [6]	1	1	1	1	1	1	0	0
	opcode [5]	0	0	0	0	0	0	1	1
	opcode [4]	1	1	1	1	0	0	0	X
	opcode [3]	1	1	0	0	0	0	0	X
	opcode [2]	0	0	0	0	0	0	X	X
	opcode [1]	0	0	0	0	1	0	X	X
	opcode [0]	0	0	0	0	0	0	X	X
Sinais de Controle (Saídas)	Reg2Loc	0	0	0	0	X	1	1	X
	ALUSrc	0	0	0	0	1	1	0	X
	MemtoReg	0	0	0	0	1	X	X	X
	RegWrite	1	1	1	1	1	0	0	0
	MemRead	0	0	0	0	1	0	0	0
	MemWrite	0	0	0	0	0	1	0	0
	Branch	0	0	0	0	0	0	1	0
	Uncondbranch	0	0	0	0	0	0	0	1
	ALU Control [3-0]	0010	0110	0000	0001	0010	0010	XX11	XXXX
	ExtendMSB[4-0]	XXXXX	XXXXX	XXXXX	XXXXX	10100	10100	10111	11001
	ExtendLSB[4-0]	XXXXX	XXXXX	XXXXX	XXXXX	01100	01100	00101	00000

Figura 5: Processador PoliLEGv8 - *Unidade de Controle*

Note que a Unidade de Controle do processador PoliLEGv8 monociclo é um circuito estritamente **combinatório** - o que é reforçado pela ausência de entradas de *clock* ou *reset*. Em seu projeto, você deve definir o valor de todos os sinais de controle (saídas da Unidade

de Controle) em função da entrada *opcode*, que define o tipo da instrução processada pelo Fluxo de Dados em dado instante de tempo.

Bloco 3 - Entidade Topo

Implemente e verifique o funcionamento do processador PoliLEGv8. Ele deve ser aderente à entidade a seguir:

```
entity polilegv8 is
  port (
    clock  : in  bit;
    reset  : in  bit
  );
end entity polilegv8;
```

O processador possui apenas duas entradas: *clock* e *reset*. Ele deve ser construído instanciando-se os componentes **fluxoDados** e **unidadeControle**, desenvolvidos nesta parte da AF12, e realizando as conexões mostradas na Figura 1 com o auxílio de sinais (*signals*).

Bloco 4 - Bancada de Testes

Para testar seu projeto, construa uma bancada de testes que gere os sinais de *clock* e *reset* e monitore, via formas de onda do GTKWave ou do ModelSim, os sinais intermediários do Fluxo de Dados e da Unidade de Controle para verificar se cada uma das instruções é executada corretamente e se os conteúdos do Banco de Registradores e da Memória de Dados são atualizados de forma adequada. **Se julgar relevante, você pode modificar a entidade polilegv8 e criar saídas extras para monitorar quaisquer sinais internos de interesse para fins de depuração**.
O arquivo e "memDadosInicialPolilegv8.dat", fornecido no e-Disciplinas, está estruturado em formato *big endian* e possui os seguintes valores iniciais para as posições de memória de dados:

```
Posições [0-7]      0x0000000000000008
Posições [8-15]     0x0000000000000005
Posições [16-23]    0xFEDCBA9876543210
Posições [24-31]    0x0123456789ABCDEF
Posições [32-39]    0x0000000000000000
Posições [40-47]    0x0000000000000000
Posições [48-55]    0x0000000000000000
Posições [56-63]    0x0000000000000000
Posições [64-71]    0x0000000000000000
```

O arquivo "memInstrPolilegv8.dat", fornecido no e-Disciplinas, está estruturado em formato *big endian* e implementa o programa a seguir. Nos comentários, já estão anotados os resultados esperados para cada instrução.

```
LDUR X0, [XZR, #0]    -- Faz X0 = 0x0000000000000008
LDUR X1, [XZR, #8]    -- Faz X1 = 0x0000000000000005
LDUR X12, [XZR, #16]  -- Faz X12 = 0xFEDCBA9876543210
LDUR X13, [XZR, #24]  -- Faz X13 = 0x0123456789ABCDEF
ADD X4, X0, X1        -- Faz X4 = X0 + X1 = 0x000000000000000D (13)
SUB X25, X0, X1       -- Faz X25 = X0 - X1 = 0x0000000000000003 (3)
SUB X16, X1, X0       -- Faz X16 = X1 - X0 = 0xFFFFFFFFFFFFFFFD (-3)
ORR X30, X12, X13     -- Faz X30 = X12 OR X13 = 0xFFFFFFFFFFFFFFFF (todos os bits em 1)
AND X8, X12, X13      -- Faz X8 = X12 and X13 = 0x0000000000000000 (todos os bits em 0)
ORR XZR, X0, X1       -- Tenta fazer XZR = X0 OR X1 = 0x000000000000000E (15),
                      -- mas XZR deve permanecer com 0
CBZ XZR, #3           -- Se não sobrescreveu indevidamente XZR na linha anterior,
                      -- pula três linhas para a frente
STUR XZR, [X9, #32]   -- Se sobrescreveu XZR indevidamente na linha anterior, escreve
                      -- nas posições [32-39] o valor errado de XZR.
                      -- Como X9 não foi escrito, é zero.
B #0                  -- Fica em loop infinito
STUR X4, [XZR, #32]   -- Escritas na memória se não sobrescreveu XZR indevidamente.
                      -- Posições [32-39] recebem X4.
STUR X25, [XZR, #40]  -- Posições [40-47] recebem X25.
STUR X16, [XZR, #48]  -- Posições [48-55] recebem X16.
STUR X30, [XZR, #56]  -- Posições [56-63] recebem X30.
STUR X8, [XZR, #64]   -- Posições [64-71] recebem X8.
B #0                  -- Fica em loop infinito
```

Instruções para os Grupos

A Atividade Formativa deve ser realizada no mesmo **grupo de 4 a 6 alunos** das Atividades Formativas anteriores. A verificação funcional do processador PoliLEGv8 deve ser realizada de acordo com as instruções e recomendações do Bloco 4 e mediante simulações com GHDL/GTKWave, EDAPlayground ou Quartus/ModelSim.

O grupo deve realizar a entrega das produções das três partes do projeto (Parte 1, Parte 2 e Parte 3) em um arquivo ZIP denominado **PCS3225_AF12_<Código do Grupo>.zip**, em que **<Código do Grupo>** é o **identificador atribuído à equipe** divulgado no e-Disciplinas (exemplo: **TnGxx**). Dentro desse arquivo ZIP, o material deve ser organizado da seguinte forma:

- Pasta P1-C1: Arquivos VHDL do Componente reg (Parte 1 - implementação e bancada de testes);
- Pasta P1-C2: Arquivos VHDL do Componente mux_n (Parte 1 - implementação e bancada de testes);
- Pasta P1-C3: Arquivos VHDL do Componente memoriaInstrucoes (Parte 1 - implementação, DAT e bancada de testes);

- Pasta P1-C4: Arquivos VHDL do Componente memoriaDados (Parte 1 - implementação, DAT e bancada de testes);
- Pasta P1-C5: Arquivos VHDL do Componente adder_n (Parte 1 - implementação e bancada de testes);
- Pasta P1-C6: Arquivos VHDL do Componente ula1bit (Parte 1 - implementação e bancada de testes);
- Pasta P1-C7: Arquivos VHDL do Componente sign_extend (Parte 1 - implementação e bancada de testes);
- Pasta P1-C8: Arquivos VHDL do Componente two_left_shifts (Parte 1 - implementação e bancada de testes);
- Pasta P2-B1: Arquivos VHDL do Componente regfile (Parte 2 - implementação e bancada de testes);
- Pasta P2-B2: Arquivos VHDL do Componente ula (Parte 2 - implementação e bancada de testes);
- Pasta P3: Arquivos VHDL dos Componentes fluxoDados, unidadeControle e polilegv8 (implementação, DAT e bancada de testes);
- Arquivo PCS3225_AF12_<Código do Grupo>.pdf, com o relato do projeto completo (documentação incremental das três partes).

Não é necessário entregar os arquivos com as formas de onda simuladas no arquivo ZIP, pois esses arquivos podem ter tamanho grande e são redundantes com os resultados já incluídos no relato. Se, ainda assim, seu arquivo ZIP superar 100 MB, submeta no e-Disciplinas um arquivo TXT com um *link* do Google Drive devidamente liberado para acesso dos professores. O *link* deve obrigatoriamente apontar para o arquivo **PCS3225_AF12_<Código do Grupo>.zip** construído de acordo com as instruções anteriores.

Todo o material produzido pelo grupo (**relato E arquivos de código**) deve conter **explicitamente** a identificação de **todos** os membros do grupo por meio de **Número USP, Nome Completo e Turma**.