PCS3225 - Sistemas Digitais II
Atividade Formativa 12 - Projeto do Processador PoliLEGv8 em VHDL
Parte 2 - Banco de Registradores e Unidade Lógica e Aritmética do PoliLEGv8

Antonio Vieira da Silva Neto
(Agradecimentos de Anos Anteriores: Edson Toshimi Midorikawa, Glauber de Bona e Sergio Roberto de Mello Canovas)

Data do Arquivo: 30/10/2025; 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.
Na primeira fase do projeto, os componentes elementares que integram o Fluxo de Dados do processador PoliLEGv8 foram construídos e verificados, constituindo uma biblioteca de componentes para a continuidade do projeto do processador. Nesta segunda fase, parte dos componentes dessa biblioteca serão integrados para construir dois blocos basais do processador PoliLEGv8, ilustrados na Figura 1: o banco de registradores e a Unidade Lógica e Aritmética (ULA) de 64 bits.

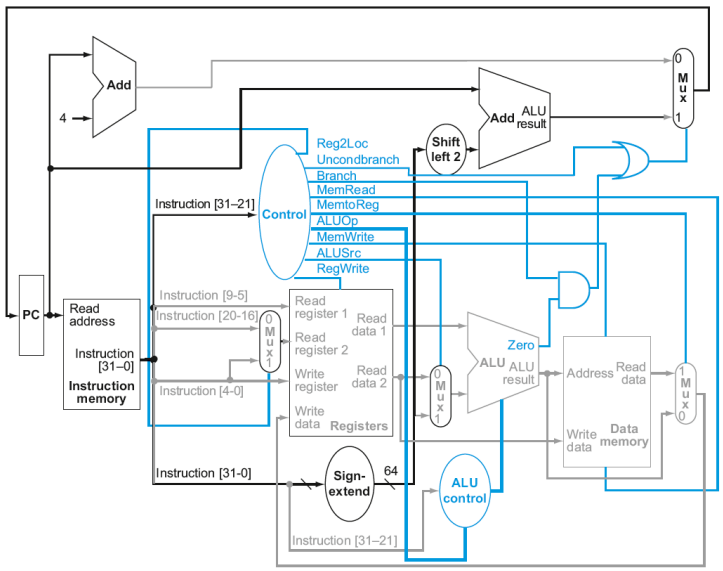

Figura 1: Processador PoliLEGv8 - Versão Monociclo

Para tanto, deve-se atender às seguintes determinações:

1. As interfaces de entrada e saída, as características funcionais e eventuais técnicas de projeto indicadas no enunciado devem ser

respeitadas;

2. Para cada componente, deve-se especificar um plano de testes que evidencie cobertura adequada de funcionamento e projetar uma bancada de testes que exercite os casos de teste previstos;

3. O projeto e a verificação funcional de cada componente devem ser documentados em um relato cumulativo a ser produzido no projeto. Nesse relato, recomenda-se seguir a estrutura subsequente: criar um capítulo para cada componente e dividi-lo em quatro partes: (i.) descrição funcional do projeto (interfaces e lógica interna), (ii.) descrição do plano de testes, (iii.) descrição da bancada de testes e (iv.) apresentação e análise de conformidade dos resultados obtidos. Cada capítulo não deve ocupar mais do que cinco páginas do relato.

Bloco 1 - Banco de Registradores

O registrador é um dos componentes básicos do processador. Ele é o elemento de memória mais próximo das unidades funcionais e também o mais rápido, pois está diretamente ligado a elas dentro do fluxo de dados. Todas as operações que ocorrem no processador PoliLEGv8, com exceção dos desvios incondicionais, envolvem ao menos um registrador.

Todos os registradores do PoliLEGv8 possuem 64 bits de largura - e, com exceção do registrador PC (*Program Counter*), os 32 outros registradores do PoliLEGv8 estão localizados em um **banco de registradores**. 31 deles, denominados de X0 até X30, comportam-se como instâncias do 'reg' da biblioteca de componentes, ao passo que o registrador X31, também denominado XZR, tem a seguinte particularidade: seu conteúdo é sempre zero e ele nunca pode ser sobrescrito.

Um **banco de registradores** é composto por um conjunto de registradores que podem ser lidos e escritos. Para isso, é necessário fornecer um número de registrador para o acesso ser realizado. Em um banco de registradores com `n` registradores, os números dos registradores variam de `0` a `n-1`. A Figura 2 a seguir ilustra um diagrama lógico geral de um banco de registradores que permite acessar simultaneamente até três registradores, sendo dois deles para leitura e um para escrita. Trata-se exatamente do caso do banco de registradores do processador PoliLEGv8.

A leitura de um registrador específico é feita de modo assíncrono e pode ser realizada somente com a definição do número de registrador. Por exemplo, no banco de registradores da figura anterior, fornece-se o código numérico de um registrador em qualquer das entradas `Read register number x`, com `x = 1, 2`, e obtém-se o dado correspondente na respectiva saída de dados `Read data x`.

A escrita em um registrador do banco, por sua vez, requer três entradas: (i.) o código numérico do registrador, (ii.) o dado a ser escrito e (iii.) um sinal de controle para definir a operação de escrita. O có-

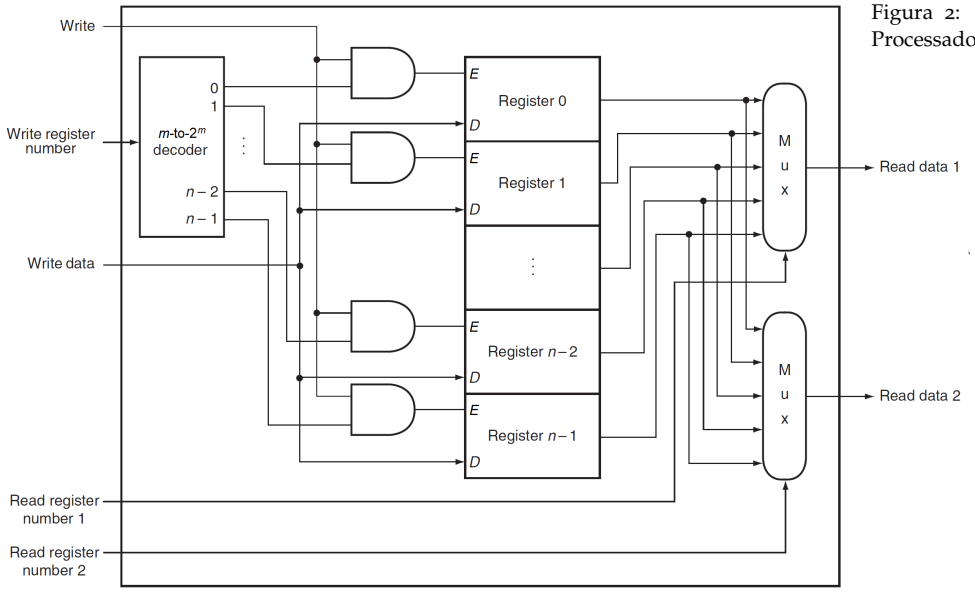

Figura 2: Banco de Registradores do Processador PoliLEGv8

digo numérico do registrador é definido na entrada `Write register number`, ao passo que o dado a ser escrito deve ser fornecido na entrada `Write data`. Quando ocorrer uma borda de subida do sinal de `Clock` e o sinal de controle de escrita `Write` for igual a `1`, a escrita do dado `Write data` é efetuada no registrador `Write register number`. Por outro lado, se o sinal de controle `Write` for igual a `0`, nenhuma operação de escrita em registrador é efetuada (independentemente dos valores de `Write register number` e `Write data`).
Dado o contexto prévio, implemente e verifique o funcionamento de um componente em VHDL correspondente ao banco de registradores do PoliLEGv8, que possui 32 registradores com 64 bits de largura cada. O banco de registradores deve respeitar a seguinte entidade:

```
entity regfile is
    port(
        clock    : in  bit;                      --! entrada de clock
        reset    : in  bit;                      --! entrada de reset
        regWrite : in  bit;                      --! entrada de carga do registrador wr
        rr1      : in  bit_vector(4 downto 0);   --! entrada define registrador 1
        rr2      : in  bit_vector(4 downto 0);   --! entrada define registrador 2
        wr       : in  bit_vector(4 downto 0);   --! entrada define registrador de escrita
        d        : in  bit_vector(63 downto 0);  --! entrada de dado para carga paralela
        q1       : out bit_vector(63 downto 0);  --! saida do registrador rr1
        q2       : out bit_vector(63 downto 0)   --! saida do registrador rr2
    );
end entity regfile;
```

Por padrão, os registradores são numerados de `0` até `31`. As entradas `rr1`, `rr2`, `wr` são de 5 bits e definem os registradores 1 e 2 a serem lidos e o registrador a ser escrito, respectivamente. Todos os registradores podem ser usados para leitura e escrita de dados. **Lembre-se de que o último registrador (`X31=XZR`) não deve aceitar escritas (não há efeito algum em escrever nele) e sempre devolve zero quando**

lido.
Todos os registradores do banco são sensíveis à borda de subida do *clock* e o *reset* é assíncrono.
A entrada de dados `d` possui largura de 64 bits e, se o sinal de controle `regWrite` for alto, na borda de subida do *clock* somente o registrador apontado pela entrada `wr` será escrito. Exemplo: se `wr=01010` o registrador `X10` (ou o décimo primeiro registrador) amostrará a entrada `d` para seus *flip-flops* internos. Caso o `regWrite` seja baixo na borda de subida do *clock*, a escrita não acontece.

Escrita no banco

A leitura é assíncrona, e o banco possui duas saídas de leitura. O registrador apontado pela entrada `rr1` coloca o seu valor na saída `q1` e o registrador apontado pela entrada `rr2` coloca seu valor na saída `q2`. Por exemplo: se `rr1=00011` e `rr2=10001`, o registrador `X3` (ou o quarto registrador) coloca o seu valor na saída `q1`, ao passo que o registrador `X17` (ou o 18º registrador) coloca o seu valor na saída `q2`.

Leituras no banco

DICA DE PROJETO 1: Construa o banco de registradores de forma estrutural, instanciando o registrador 'reg' criado na Parte 1 da AF12 ao menos 31 vezes (registradores X0 até X30).Consulte a postagem do Prof. Bruno Albertini em `https://balbertini.github.io/vhdl_generate-pt_BR.html` sobre descrição de circuitos em VHDL com a composição repetitiva de diversas instâncias do mesmo componente (usando o comando `for-generate`).

DICA DE PROJETO 2: Para o registrador X31 (XZR), você pode ou criar uma entidade alternativa para representá-lo, ou adicionar uma arquitetura alternativa para 'reg' que tenha o comportamento esperado para XZR e instanciar XZR usando essa arquitetura alternativa.

DICA DE PROJETO 3: Pelo diagrama da Figura 2, você precisará criar dois componentes não existentes na biblioteca da Parte 1: um decodificador 5 x 32, que habilita a escrita do registrador indicado pela entrada `wr`, e um multiplexador de 32 entradas de 64 bits cada. O multiplexador precisará ser instanciado duas vezes: uma recebendo como entrada de seleção o sinal `rr1`, para escolher o registrador que deve colocar seu conteúdo na saída `q1`, e outro recebendo como entrada de seleção o sinal `rr2`, para escolher o registrador que deve colocar seu conteúdo na saída `q2`.

Componente 2 - ULA de 64 Bits

A ULA do Processador PoliLEGv8 é capaz de realizar as operações aritméticas de adição e subtração, as operações lógicas AND, OR e NOR e uma operação de repasse da entrada B (*Pass B*). A Tabela 1 resume as seis operações passíveis de execução na ULA.
Nota: *Dado o conjunto de instruções do PoliLEGv8, apenas as operações lógicas AND e OR e as operações aritméticas de soma e subtração são diretamente utilizadas pelo programador. A operação "Pass B"é utilizada para detectar se um registrador possui valor zero em desvios condicionais do tipo*

OP	Operação	Descrição
`0000`	`AND`	$A\&B$, bit a bit
`0001`	`OR`	$A\|B$, bit a bit
`0010`	Soma	$A + B$
`0110`	Subtração	$A - B$
`0111`	Repasse da Entrada B ("Pass B")	B
`1100`	`NOR`	$\overline{A\|B}$, bit a bit

Tabela 1: Operações que podem ser realizadas pela ULA. O campo OP pode ser interpretado como ($Op_3Op_2Op_1Op_0$), sendo Op_3=inverte `A`, Op_2=inverte `B`, e Op_1Op_0=escolhe a unidade funcional entre: `00:AND`, `01:OR`, `10:ADD` (somador), `11:Pass B`.

"CBZ". A operação NOR está disponível para ser utilizada pelo programador em extensões futuras do conjunto de instruções do processador.

Sob o ponto de vista funcional, a ULA possui, na prática, três unidades funcionais - AND, OR e um somador - além da capacidade de inverter qualquer entrada independentemente. As operações `AND`, `OR` e adição são realizadas diretamente pelas unidades correspondentes. A subtração é obtida invertendo-se a entrada `B`, entrando-se com `1` no *carry-in*, e realizando a soma, com o complemento de base do número. A operação `NOR` é realizada pela unidade `AND`, invertendo-se as entradas. Já a operação `Pass B` (Repasse da Entrada B) consiste em replicar a entrada B na saída da ULA.

Assuma que *carry-in* deve ser 1 sempre que `B` for invertido.

Use o teorema de DeMorgan para entender o `NOR`.

Na prática, `Pass B` é utilizada para verificar alguma condição externa sobre a entrada B, - por exemplo, se ela é zero nos desvios condicionais do tipo "CBZ".

Dado o contexto prévio, implemente e verifique o funcionamento de um componente em VHDL correspondente à ULA de 64 bits do PoliLEGv8. Ela deve respeitar a entidade mostrada a seguir.

```
entity ula is
  port (
    A   : in  bit_vector(63 downto 0); -- entrada A
    B   : in  bit_vector(63 downto 0); -- entrada B
    S   : in  bit_vector(3 downto 0);
-- seleciona operacao
    F   : out bit_vector(63 downto 0); -- saida
    Z   : out bit;                     -- flag zero
    Ov  : out bit;                     -- flag overflow
    Co  : out bit                      -- flag carry out
  );
end entity alu;
```

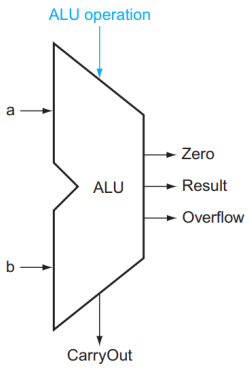

Figura 3: Diagrama da ULA.

A Figura 3 mostra o símbolo para a ULA de 64 bits. Note que ela ULA recebe como entradas dois dados com 64 bits de largura, denominados A e B, e que ela possui um sinal de controle de 4 bits denominado S, que serve para determinar a operação a ser executada. As saídas da ULA são o resultado da operação executada, denominado F e que também possui 64 bits de largura, e três *flags*: Z para indicar se o resultado da operação é zero, Ov para indicar se houver *overflow* e Co (*carry out*) para indicar se houve produção de vai-um ao final do processamento.

No material didático, o sinal de controle S é denominado *ALU control*.

Note que não há entrada de *carry*.

Os três *flags* são ativos em nível alto, tal como todo sinal do PoliLEGv8. Por exemplo, se o resultado de uma operação for zero, a saída Z da ULA será igual a 1. chamadas `ula1bit` e construídas na primeira parte do projeto.

Use os comandos `for-generate` e `if-generate` do VHDL; veja 'DICA DE PROJETO 1' do banco de registradores.

O circuito da ULA do PoliLEGv8 deve ser implementado como uma

composição de 64 ULAs elementares de 1 bit,
A Figura 4 mostra a estrutura interna da ULA, evidenciando como as unidades elementares formam a composição final. Repare também que a saída `Zero` é gerada por uma porta NOR. Estude essa figura e encontre o padrão repetitivo de conexão de um componente a outro.

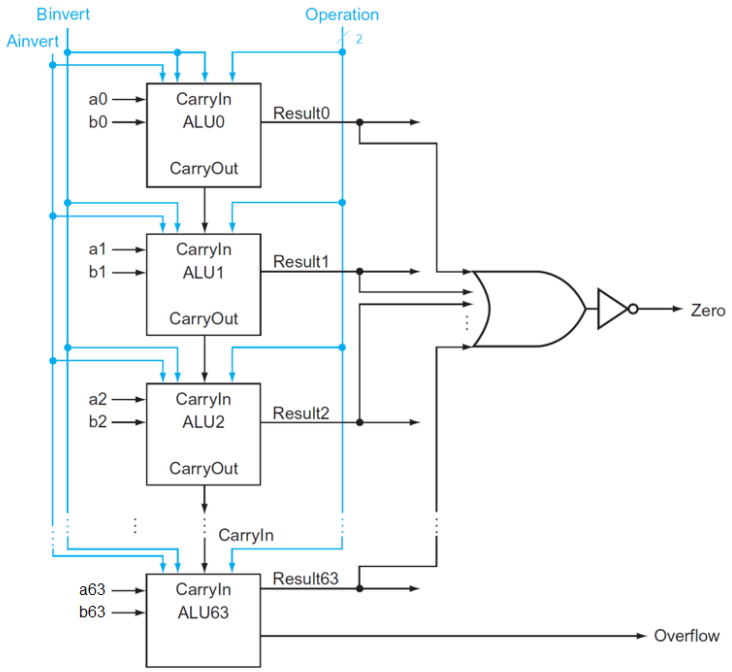

Figura 4: Diagrama da estrutura interna da ULA do PoliLEGv8.

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 de funcionamento dos componentes deve ser realizada com a implementação dos casos de teste, a criação dos *testbenches* e a simulação com GHDL/GTKWave, EDAPlayground ou Quartus/ModelSim.

Não é necessário realizar a entrega das produções intermediárias do projeto e do relato desta parte da Atividade Formativa: ela deve ser guardada e combinada com os desenvolvimentos das próximas partes para uma única entrega.

Reforça-se, porém, 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**.

Atenção: É vedado o uso da biblioteca `ieee.std_logic_1164`, ou qualquer outra biblioteca não padronizada, no projeto do PoliLEGv8.