

# PCS3225 - Sistemas Digitais II

## Projeto 3 - Banco de Registradores e Instruções Aritméticas

Glauber De Bona

Prazo: 20/10/2021

O objetivo deste trabalho é exercitar a descrição de registradores em VHDL, que serão usados como componentes internos do PoliLEGv8. Além disso, este trabalho oferece um primeiro contato com a decodificação e implementação de instruções aritméticas.

### Introdução

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 através do fluxo de dados interno. Todas as operações importantes que acontecem no processador tem como origem ou destino um registrador, normalmente no **banco de registradores**.

Em sistemas digitais, um registrador nada mais é que um conjunto de *flip-flops*. Em VHDL, um *flip-flop* é descrito usando-se uma atribuição incompleta dentro de um bloco sequencial. Veja o exemplo abaixo de um bloco sequencial:

```
ffdr: process(clock, reset)
begin
    if reset='1' then
        q <= '0';
        q_n <= '1';
    elsif clock='1' and clock'event then
        if en = '1' then
            q <= d;
            q_n <= not d;
        end if;
    end if;
end process;
```

Neste bloco, descrevemos um *flip-flop* tipo D com *reset* assíncrono. Note que o processo tem em sua lista de sensibilidade o *clock* e o *reset*, mas o *if* divide as ações em assíncronas (se houver *reset*) ou síncronas (sensível à borda de subida do *clock*). Porém, se não houver um *reset* nem uma borda de subida do *clock*, o comportamento não é especificado. Chamamos isso de **atribuição incompleta**, e a ação padrão do sintetizador será de manter os sinais inalterados.

Para manter os sinais como estão, deve existir um elemento de memória. Nesse caso, o elemento de memória que forçamos o sintetizador a usar para manter o valor é exatamente um *flip-flop*. Caso desejássemos, poderíamos colocar um *else* contendo uma atribuição para o mesmo sinal, mas isto é desnecessário. Este tipo de atribuição incompleta também funciona para outros sinais. O sintetizador se encarregará de colocar o número de *flip-flops* adequado para manter

No LEGv8 são 32 de 64 bits

O *reset* é prioritário pois é avaliado antes.

q<=q;

o valor de um vetor ou sinal inalterado.

### Atividades

**P3A1** (3 pontos) Implemente um componente em VHDL correspondente a um registrador parametrizável, respeitando a seguinte entidade:

Projeto 3, Atividade 1

```
entity reg is
  generic(wordSize: natural :=4);
  port(
    clock: in bit; --! entrada de clock
    reset: in bit; --! clear assincrono
    load: in bit; --! write enable (carga paralela)
    d: in bit_vector(wordSize-1 downto 0); --! entrada
    q: out bit_vector(wordSize-1 downto 0) --! saída
  );
end reg;
```

O registrador é parametrizável através do parâmetro `wordSize`, que determina o seu tamanho em bits. Recebe um sinal de *clock* periódico e é sensível a borda de subida deste sinal. O *reset* é assíncrono e força todos os bits para zero. Quando o sinal *load* é alto, a escrita é considerada habilitada e o valor na entrada *d* é gravado nos *flip-flops* quando ocorrer uma borda de subida do *clock*, refletindo então na saída *q*. Contrariamente, quando o sinal *load* é baixo, nada acontece e a saída permanece inalterada. A saída é assíncrona e mostra o conteúdo atual do registrador.

Tanto a entrada *d* quando a saída *q* são paralelas e contém `wordSize` bits.

**P3A2** (3 pontos) Implemente um componente em VHDL correspondente a um banco de registradores (*register file*) que respeite a seguinte entidade:

Projeto 3, Atividade 2

```
entity regfile is
  generic(
    regn: natural := 32;
    wordSize: natural := 64
  );
  port(
    clock: in bit;
    reset: in bit;
    regWrite: in bit;
    rr1, rr2, wr: in bit_vector(natural(ceil(log2(real(regn))))-1 downto 0);
    d: in bit_vector(wordSize-1 downto 0);
    q1, q2: out bit_vector(wordSize-1 downto 0)
  );
end regfile;
```

Os parâmetros determinam o número de registradores (`regn`) e o tamanho da palavra de cada registrador (`wordSize`). Por padrão, os registradores são numerados de 0 até `regn-1`. O banco de registradores possui duas portas de leitura de dados, *q1* e *q2*, correspondentes às entradas de endereço *rr1* e *rr2*, respectivamente. Há apenas uma

porta para a escrita de dados, *d*, que corresponde ao endereço na entrada *wr*.

As entradas *rr1*, *rr2*, *wr* são declaradas em função de *regn*, fazendo  $\lceil \log_2 \text{regn} \rceil - 1$ . No caso de um banco com 32 registradores, estas entradas serão de 5 bits (`bit_vector(4 downto 0)`). Para fazer este cálculo dependente dos parâmetros, usamos a biblioteca matemática, então você obrigatoriamente deverá incluir as cláusulas de uso `use ieee.math_real.ceil`; e use `ieee.math_real.log2`; na sua descrição.

Assim como no caso dos registradores do PoliLEGv8, o último registrador não deve aceitar escritas (não há efeito algum em escrever nele) e sempre retorna zero quando lido.

É opcional, mas fortemente aconselhável, usar o registrador da atividade anterior como componente. O restante das características do registrador permanecem as mesmas: 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 *wordSize*, mas na borda de subida do *clock* somente o registrador apontado por *wr* será escrito, se o *regWrite* for alto. Exemplo: para 32 registradores, se *wr*=01010 o registrador 10 (ou o décimo primeiro) amostrará a entrada *d* para seus *flip-flops*. Caso o *regWrite* seja baixo na borda de subida do *clock*, a escrita não acontece.

A leitura é assíncrona. O registrador apontado pela entrada *rr1* coloca o seu valor na saída *q1*, e o apontado pela entrada *rr2* coloca na saída *q2*. Exemplo: para 32 registradores, se *rr1*=00011 registrador 3 (ou o quarto) coloca o seu valor armazenado na saída *q1*.

Se não usou a biblioteca *ieee*, terá que declará-la também.

No caso de 32 registradores, o de número 31 é o último.

**Dica:** Para instanciar *regn* componentes, use o comando *generate*, conforme <https://balbertini.github.io/vhdl-generate-pt-BR.html>

O *reset* vale para todos os registradores do banco.

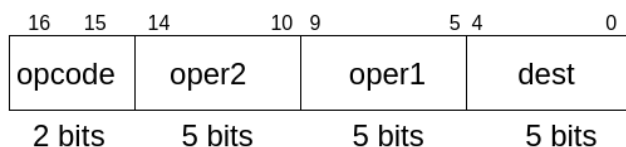
**P3A3** (4 pontos) Implemente um componente em VHDL, que executará instruções de adição e subtração, respeitando a seguinte entidade:

```
entity calc is
    port(
        clock:      in  bit;
        reset:      in  bit;
        instruction: in  bit_vector(16 downto 0);
        q1:         out bit_vector(15 downto 0)
    );
end calc;
```

Esta calculadora deve instanciar internamente um banco de registradores da atividade P3A2, contendo 32 registradores, cada um com 16 bits. As portas *reset* e *clock* de *calc* devem ser conectadas às portas de mesmo nome no banco de registradores. A calculadora deve implementar um conjunto de quatro instruções (ADD, ADDI, SUB e SUBI), similares àquelas do LEGv8, porém com uma codificação mais enxuta, de 17 bits, num único formato:

Projeto 3, Atividade 3

Como na atividade anterior, o último registrador (31) deve retornar sempre o.



- *opcode*: 00 para ADD, 01 para ADDI, 10 para SUB, 11 para SUBI;
- *oper2*: o segundo operando, que será um registrador para ADD e SUB, e será um imediato em complemento de 2 para ADDI e SUBI;
- *oper1*: o primeiro operando, que será sempre um registrador;
- *dest*: o registrador destino, onde se guardará o resultado da soma.

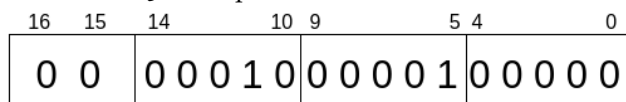
A cada borda de subida da entrada clock, a calculadora executa a instrução presente na entrada instruction: o registrador especificado no campo *dest* é atualizado com o resultado da soma/subtração do registrador especificado em *oper1* com o operando especificado em *oper2*, que pode ser um registrador (em ADD ou SUB) ou um imediato (em ADDI e SUBI), de acordo com o valor do *opcode*.

Para realizar as somas/subtrações, você pode descrever um componente somador/subtrator e instanciá-lo dentro da arquitetura da calculadora. Você pode usar os operadores correspondentes a soma/subtração associados aos tipos signed ou unsigned. Como o imediato é dado em 5 bits em complemento de 2, lembre-se de preservar seu sinal quando passá-lo para 16 bits.

No PoliLEGv8, vocês deverão somar e subtrair com sua própria ULA, tema do próximo projeto.

A saída q1 deve, a todo momento, corresponder à leitura do registrador especificado no campo *oper1* da entrada instruction.

Por exemplo, suponha que os registradores X1 e X2 contêm os números 200 e 300, respectivamente, e na entrada instruction lê-se:

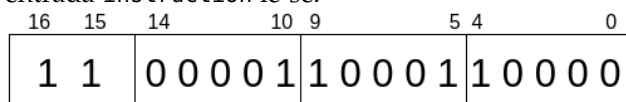


Em assembly:

```
ADD X0, X1, X2      // X0 = X1 + X2
```

A saída q1 deve conter os 16 bits do registrador especificado em *oper1*; ou seja, 200 em binário. Quando chegar uma borda de subida de clock, o registrador X0 deve ser atualizado com o resultado da soma, 500.

Agora suponha que o registrador X17 contém o número 200 e que na entrada instruction lê-se:



Em assembly:

```
SUBI X16, X17, #1    // X16 = X17 - 1
```

A saída q1 deve conter os 16 bits do registrador especificado em *oper1*; ou seja, 200 em binário. Quando chegar uma borda de subida de clock, o registrador X16 deve ser atualizado com o resultado da subtração, 199.

### *Instruções para Entrega*

Você deve acessar o link específico para cada tarefa (P3A1, P3A2 e P3A3) dentro da seção “Projetos” no E-Disciplinas, já logado com seu usuário e senha, que levará à página apropriada do juiz eletrônico. Você deve submeter apenas um arquivo, codificado em UTF-8, contendo inclusive a descrição dos subcomponentes utilizados que foram desenvolvidos por você (como `reg` e `regfile`). O prazo para a submissão das soluções no Juiz é 20 de outubro de 2021, quarta-feira, às 23:59. O Juiz aceitará até 5 submissões para cada atividade deste projeto. Sua submissão será corrigida imediatamente e sua nota será apresentada. A maior nota dentre as submissões será considerada. Neste trabalho, os problemas valem no máximo 10 pontos no juiz, porém a nota final deste trabalho será calculada com as ponderações indicadas em cada atividade neste enunciado, totalizando 10 para o trabalho todo. Como boa prática de engenharia, faça seus *test-benches* e utilize o GHDL para validar suas soluções antes de postá-las no juiz.

**Atenção:** Para as três atividades do projeto, está proibido o uso da biblioteca `ieee.std_logic_1164`. Se seu arquivo mencionar essa biblioteca, mesmo em um comentário, sua submissão nem será avaliada pelo juiz e ficará com nota 0 (zero).