***Relatório referente ao trabalho final de sistemas digitais***

Nome: Bruna Izabel da Silva Pereira - 15635441

Camila Piscioneri Magalhães - 15697249

# Introdução

O objetivo é desenvolver em VHDL um processador digital simples segundo as especificações do processador, sendo elas baseadas em uma cpu monocíclica, como a *figura 1*. Para realizá-lo foram usadas as seguintes unidades em formato vhdl:

* alu (arithmetic logic unit);
* Registrador (unidade de registrador padrão);
* banco\_de\_reg(unidade para a instanciação e escolha de qual registrador usar);
* mux\_eight\_two (multiplexador 8x2);
* mux\_eight\_four (multiplexador 8x4);
* demux (desmultiplexador);
* contador (Process counter(Pc);
* unid\_controle (unidade de controle);
* cpu\_offic(montagem do processador);
* —------

# Arithmetic Logic Unit (Alu)

## Funcionalidades:

A ALU possui as seguintes funcionalidades:

* **Operações Aritméticas**: Adição e Subtração;
* **Operações Lógicas**: AND, OR, e NOT;
* **Geração de Flags**: Zero, Sinal, Carry e Overflow.

## Entradas:

* A (8 bits): Primeiro operando;
* B (8 bits): Segundo operando;
* Op (4 bits): Código da operação a ser realizada.

## Saídas:

* Result (8 bits): Resultado da operação;
* Zero (1 bit): Indica se o resultado é zero;
* Sinal (1 bit): Indica o sinal do resultado (bit mais significativo);
* Carry (1 bit): Indica um transporte na adição ou subtração;
* Overflow (1 bit): Indica se ocorreu overflow durante a operação.

## Operações:

| Código (op) | Operação | Descrição |
| --- | --- | --- |
| 0000 | ADD | Soma dos operandos A e B. Gera as flags Carry e Overflow. |
| 0001 | SUB | Subtração de A por B. Gera as flags Carry, Sinal e Overflow. |
| 0010 | AND | Operação lógica **AND** bit a bit entre A e B. |
| 0011 | OR | Operação lógica **OR** bit a bit entre A e B. |
| 0100 | NOT | Operação lógica **NOT** bit a bit sobre o operando A. |
| others | DEFAULT | Saída zerada para valores de Op não especificados. |

## Detalhes de implementação:

### Aritmética com Carry:

* A soma e subtração são realizadas com extensão de sinal, adicionando um bit extra ('0' & A, '0' & B).
* O bit mais significativo do resultado estendido (temp\_result(8)) é usado para definir o **Carry**.

### **Cálculo de Overflow**:

* O **Overflow** é calculado comparando os sinais dos operandos e o sinal do resultado:
  + Soma: Ocorre overflow se os sinais de A e B forem iguais, mas diferentes do resultado.
  + Subtração: Ocorre overflow se o sinal de A for diferente de B, mas o resultado for inconsistente.

### **Operações Lógicas**:

* São realizadas diretamente sobre os operandos sem necessidade de cálculos adicionais.

### **Flags**:

* **Zero**: Ativado ('1') se o resultado for 00000000.
* **Sinal**: Indica o bit mais significativo do resultado (signo em complemento de dois).
* **Carry**: Indica transporte em operações aritméticas.
* **Overflow**: Indica inconsistência de sinal em operações aritméticas.

## **Resultado Padrão**:

* Para operações não definidas (when others), o resultado e todas as flags são zerados.

# Registrador

## **Entradas**

* + clk: Sinal de clock para sincronização.
  + reset: Sinal de reset (ativo alto) para zerar o registrador.
  + enable: Sinal de controle para habilitar a carga de dados.
  + d\_in: Vetor de 8 bits representando os dados de entrada.

## **Saída**s

* + d\_out: Vetor de 8 bits contendo o valor armazenado no registrador.

## Funcionamento

### **Reset**:

* + Quando reset = '1', o registrador é zerado (d\_out <= (others => '0')).

### Sincronismo:

* + Em uma borda de subida do clock (rising\_edge(clk)), verifica-se o valor de enable:
    - Se enable = '1', os dados de entrada (d\_in) são carregados no registrador e disponibilizados na saída (d\_out).
    - Caso contrário, o valor armazenado permanece inalterado.

Banco\_de\_reg (Banco de registradores)

**Resumo**

Este código implementa um **banco de registradores de 8 bits** com 4 registradores e um mecanismo de seleção para leitura e escrita. O design permite armazenar, ler e selecionar dados em múltiplos registradores controlados por sinais de entrada.

### Descrição Geral

**Componentes Principais**:

* + **Registradores**: Armazenam valores de 8 bits. São instâncias do componente Registrador.
  + **Demultiplexador (Demux)**: Direcciona o sinal de escrita (reg\_write) ao registrador correspondente, baseado no endereço de escrita (WR).
  + **Multiplexadores (Mux)**: Selecionam os dados de saída dos registradores com base nos endereços de leitura (end\_RA e end\_RB).

**Funcionamento Geral**:

* + O sinal de escrita (reg\_write) e o endereço (WR) habilitam o registrador apropriado para armazenar os dados de entrada (WD).
  + Os dados podem ser lidos de dois registradores ao mesmo tempo, controlados pelos endereços de leitura (end\_RA e end\_RB).
  + Sinais de reset (reset) são usados para limpar os valores armazenados.

#### **Entradas:**

* **clk**: Sinal de clock para sincronização.
* **reset**: Sinal de reset (ativo alto) para limpar todos os registradores.
* **reg\_write**: Sinal de habilitação para escrita.
* **WR**: Endereço (2 bits) para seleção do registrador de escrita.
* **WD**: Dados de entrada (8 bits) a serem escritos.
* **end\_RA e end\_RB**: Endereços (2 bits cada) para seleção dos registradores de leitura.

#### **Saídas:**

* **RA e RB**: Dados de saída (8 bits) dos registradores selecionados para leitura.

### Funcionamento Interno

**Escrita**

* + O Demux direciona o sinal de escrita (reg\_write) ao registrador correspondente com base no endereço (WR).
  + Apenas o registrador selecionado é habilitado para armazenar o valor de entrada (WD).

**Leitura**

* + Dois multiplexadores (Mux1 e Mux2) são usados para selecionar os dados de saída dos registradores, com base nos endereços de leitura (end\_RA e end\_RB).

**Reset**

* + Quando reset = '1', todos os registradores são zerados.

Mux\_eight\_two

### Componentes do Código

#### **Entidade (entity)**

Define a interface do módulo com as entradas e saídas:

* **A**: Entrada de 8 bits.
* **B**: Outra entrada de 8 bits.
* **Sel**: Sinal de seleção ('0' ou '1') para escolher entre as entradas.
* **Y**: Saída de 8 bits que recebe o valor de A ou B.

#### **Arquitetura (architecture)**

Descreve o comportamento do multiplexador:

* Um **processo sensível às mudanças em A, B e Sel** implementa a lógica de seleção:
  + Se Sel = '0': A saída Y recebe o valor de A.
  + Se Sel = '1': A saída Y recebe o valor de B.

### Funcionamento do Multiplexador

* **Entrada**: Duas palavras de 8 bits (A e B) são disponibilizadas para o módulo.
* **Seleção**: O sinal Sel determina qual das entradas será passada para a saída:
* **Sel = '0'**: A entrada A é direcionada para a saída Y.
* **Sel = '1'**: A entrada B é direcionada para a saída Y.
* **Saída**: A saída Y é atualizada dinamicamente com base nas entradas e no sinal de seleção.

Mux\_eight\_four

Este código implementa um multiplexador (MUX) **4:1** com entradas de 8 bits (A, B, C e D) e uma saída (Y). O multiplexador seleciona uma das entradas com base no valor do sinal de seleção de 2 bits (Sel).

### Componentes do Código

#### **Entidade (entity)**

Define a interface do módulo:

* **Entradas**:
  + **A, B, C, D**: Entradas de 8 bits.
  + **Sel**: Sinal de seleção de 2 bits para determinar qual entrada será passada para a saída.
* **Saída**:
  + **Y**: Vetor de 8 bits que contém o valor da entrada selecionada.

#### **Arquitetura (architecture)**

Implementa o comportamento funcional do MUX:

* Um **processo sensível às entradas (A, B, C, D) e ao sinal de seleção (Sel)** decide qual entrada será atribuída à saída:
  + **Sel = "00"**: Y recebe o valor de A.
  + **Sel = "01"**: Y recebe o valor de B.
    - * + **Sel = "10"**: Y recebe o valor de C.
        + **Sel = "11"**: Y recebe o valor de D.
        + **others**: Por segurança, a saída Y é zerada.

**Funcionamento do Multiplexador**

**Entrada**:

* + O multiplexador recebe quatro palavras de 8 bits (A, B, C, D) como entradas.
  + Um sinal de seleção de 2 bits (Sel) é usado para determinar qual entrada será direcionada à saída.

**Seleção**:

* + O valor do sinal de seleção (Sel) é analisado em um bloco **case**:
    - "00" seleciona a entrada A.
    - "01" seleciona a entrada B.
    - "10" seleciona a entrada C.
    - "11" seleciona a entrada D.

**Saída**:

* + A saída Y é atribuída dinamicamente com base no valor do sinal de seleção.