Resumo do Código VHDL - ULA

Esse código implementa uma Unidade Lógica e Aritmética (ULA) em VHDL. A ULA é responsável por realizar operações matemáticas e lógicas básicas. Aqui está uma visão geral de como ela funciona:

Entradas e Saídas

- **Entradas:**
  - `A`, `B` (8 bits): Operandos para as operações.
  - `instrucao` (4 bits): Define a operação a ser realizada.
  - `Clk`: Clock para sincronização.
- **Saídas:**
  - `O` (8 bits): Resultado da operação.
  - `zero`: Indica se o resultado foi zero.
  - `sinal`: Indica se o resultado de uma subtração foi negativo.

Funcionalidade

O comportamento da ULA é controlado por um processo síncrono, ativado a cada borda de subida do clock. Dependendo do valor de `instrucao`, ela realiza as seguintes operações:

- **"0000" (ADD):** Soma `A` e `B`. Se o resultado for zero, ativa o sinal `zero`.
- **"0001" (SUB):** Subtrai `B` de `A`. Também verifica se `B > A` para ativar o sinal `sinal`.
- **"0010" (AND):** Faz a operação lógica AND entre os bits de `A` e `B`.
- **"0011" (OR):** Faz a operação lógica OR entre os bits de `A` e `B`.
- **"0100" (NOT A):** Inverte os bits de `A`.
- **Demais valores:** Retorna o resultado como zero.

Observações

- O sinal `zero` é usado para indicar resultados iguais a zero, facilitando condições em sistemas maiores.
- O sinal `sinal` é útil para identificar resultados negativos em operações de subtração.

Essa ULA é um componente essencial em processadores e pode ser facilmente expandida para suportar operações mais complexas.

Resumo do Código VHDL - Memória

Esse código implementa uma memória RAM de 256 posições, onde cada posição armazena 8 bits. Ela é usada para leitura e escrita de dados em sistemas digitais.

Entradas e Saídas

- **Entradas:**
  - `Address` (8 bits): Endereço da posição na memória.
  - `Clock`: Controla o momento das operações.
  - `DataIn` (8 bits): Dados a serem escritos na memória.

- `WriteMode`: Define se será feita uma operação de escrita (`'1'`).

- **Saídas:**
  - `DataOut` (8 bits): Retorna os dados lidos da posição especificada.

Funcionamento

A memória opera de forma síncrona:

1. **Escrita:** Quando `WriteMode` é `'1'` e ocorre uma borda de subida no `Clock`, o dado em `DataIn` é gravado no endereço especificado por `Address`.
2. **Leitura:** O dado armazenado no endereço indicado por `Address` é continuamente disponibilizado em `DataOut`.

Inicialização

A memória é pré-carregada com valores em algumas posições, enquanto o restante é preenchido com zeros. Isso simula uma memória inicializada com instruções ou dados para testes.

Integração com o Processador

No contexto do processador descrito no PDF, essa memória:

- Armazena instruções e dados temporários.
- Permite a CPU acessar e modificar informações via barramentos de endereço e dados.

Esse código demonstra uma memória simples e eficiente, essencial para sistemas digitais básicos e fácil de integrar em projetos maiores.

Relatório do Processador Simulado em VHDL

1. Visão Geral

O código representa um processador com uma arquitetura básica, capaz de executar operações aritméticas, lógicas, comparações, controle de fluxo (saltos) e manipulação de dados em memória. Ele utiliza um contador de programa (PC) para rastrear a execução das instruções e emprega componentes como uma unidade lógica e aritmética (ULA) e uma memória RAM.

---

2. Componentes do Processador

O processador é formado pelos seguintes componentes principais:

2.1. Memória (memoria256x8)

- **Descrição**: Simula uma memória RAM com capacidade de 256 palavras de 8 bits.
- **Entradas**:
  - `Address`: Endereço a ser lido ou escrito.
  - `Clock`: Sinal de sincronização.
  - `DataIn`: Dados para escrita.
  - `WriteMode`: Habilita o modo de escrita.
- **Saída**:
  - `DataOut`: Dados lidos da memória.

2.2. ULA (Unidade Lógica e Aritmética)

- **Descrição**: Executa operações aritméticas e lógicas entre dois operandos.
- **Entradas**:
  - `A`, `B`: Operandos.
  - `instrucao`: Código da operação (ADD, SUB, AND, OR, NOT).
  - `Clk`: Sinal de sincronização.
- **Saídas**:
  - `O`: Resultado da operação.
  - `zero`, `sinal`: Indicadores do estado do resultado.

---

3. Sinais

Os sinais internos controlam o fluxo e armazenam estados intermediários:

- `PC`: Contador de programa, indicando a próxima instrução a ser executada.
- `A`, `B`, `R`: Registradores de propósito geral para armazenar dados.
- `instrucao`: Representa a instrução atual a ser executada.
- `esperando` e `lerX`: Sinais de controle para coordenar acessos à memória e execução.
- `flag_igual`, `flag_maior`: Flags para indicar resultados de comparações.
- `pause`: Indica pausa na execução para aguardar um evento externo (como `RESET`).

3.1. Processo Principal

Um processo sensível à borda de subida do sinal `Clk`. Este processo é dividido em três fases principais:

Controle do Clock:
- Atualiza o estado de `luz_Clk`, indicando o funcionamento do clock.

Execução de Instruções:
- Decodifica a instrução armazenada em `instrucao`.
- Executa a ação correspondente, dependendo do código da instrução (`instrucao(7 downto 4)`).

Estados Intermediários:
- Controla o fluxo de leitura/escrita em memória e a execução da ULA por meio de sinais como `esperando`, `ler1`, e `ler2`.

3.2. Tipos de Instruções

As instruções são organizadas por tipo, com base nos 4 bits mais significativos (`instrucao(7 downto 4)`):

1. **Aritméticas/Lógicas (0000 a 0100)**:
   - `ADD`, `SUB`, `AND`, `OR`, `NOT`.
   - Operações realizadas na ULA.
2. **Comparação (0101)**:
   - `CMP`: Compara dois valores e atualiza `flag_igual` e `flag_maior`.
3. **Controle de Fluxo (0110 a 1000)**:
   - `JMP`, `JEQ`, `JGR`: Saltos condicionais com base nas flags.
4. **Memória (1001, 1010)**:

- `LOAD`: Carrega valor da memória para um registrador.
- `STORE`: Armazena o valor de um registrador na memória.

5. **Movimentação de Dados (1011)**:
   - `MOV`: Copia valores entre registradores.
6. **Entrada/Saída (1100, 1101)**:
   - `IN`: Lê valores de `INPUT_UNIT`.
   - `OUT`: Escreve valores em `OUTPUT_UNIT`.
7. **Controle (1110)**:
   - `WAIT`: Coloca o processador em estado de pausa.

---

4. Controle de Fluxo

O estado `esperando` é fundamental para coordenar operações que demandam mais de um ciclo, como acessos à memória. Durante a execução:

1. A instrução é decodificada.
2. Sinais como `ler1` e `ler2` são ativados para preparar os dados.
3. Após a execução, o processador retorna ao estado normal e avança o `PC`.

---

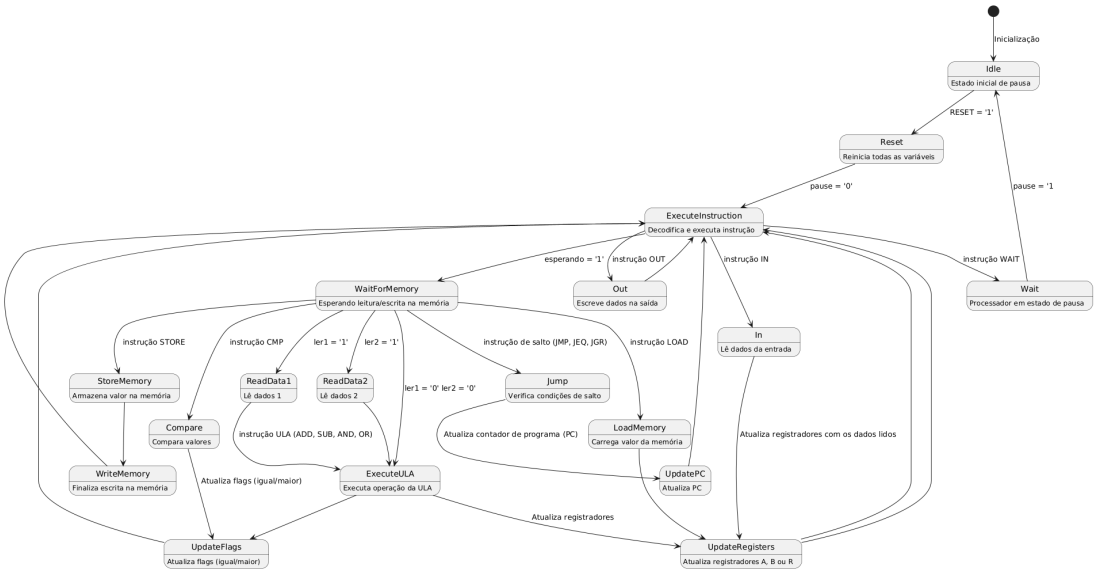
5. Estado de Pausa

O processador pode ser pausado (`pause = '1'`) e retomado apenas com a ativação de `RESET`. Isso é útil para sincronizar eventos externos ou controlar o fluxo de instruções.

---

6. Conclusão

O código implementa um processador básico com suporte a operações comuns de uma unidade de processamento. A estrutura modular, com a ULA e memória como componentes, facilita a expansão e adaptação para instruções mais complexas.



Casos de Entrada e Saída para o Processador

Caso 1: Soma Simples (ADD A, B)

Descrição

Executa a soma dos valores armazenados nos registradores `A` e `B`.

Entradas

- `INPUT_UNIT`: Não utilizado diretamente neste exemplo.
- Registrador `A`: `00000011` (3 em decimal).
- Registrador `B`: `00000101` (5 em decimal).
- Instrução (`instrucao`): `00000000` (ADD).

Saída

- Resultado no Registrador `R`: `00001000` (8 em decimal).
- `OUTPUT_UNIT`: Não alterado.

Caso 2: Subtração (SUB A, B)

Descrição

Realiza a subtração do valor em `B` a partir do valor em `A`.

Entradas

- Registrador `A`: `00001000` (8 em decimal).
- Registrador `B`: `00000100` (4 em decimal).
- Instrução (`instrucao`): `00010000` (SUB).

Saída

- Resultado no Registrador `R`: `00000100` (4 em decimal).
- Sinal `sinal`: `0` (resultado não negativo).

Caso 3: Operação Lógica AND (AND A, B)

Descrição

Faz uma operação lógica AND entre os bits de `A` e `B`.

Entradas

- Registrador `A`: `11001100`.
- Registrador `B`: `10101010`.
- Instrução (`instrucao`): `00100000` (AND).

Saída

- Resultado no Registrador `R`: `10001000`.
- Sinais auxiliares: Não alterados.

Caso 4: Carregar da Memória (LOAD A, endereço)

Descrição

Carrega o valor de uma posição de memória para o registrador `A`.

Entradas

- Endereço (`Address`): `00000010` (posição 2).
- Valor armazenado na posição: `11111111`.
- Instrução (`instrucao`): `10010000` (LOAD).

Saída

- Registrador `A`: `11111111`.

Caso 5: Escrever na Memória (STORE R, endereço)

Descrição

Escreve o valor do registrador `R` em uma posição específica da memória.

Entradas

- Registrador `R`: `10101010`.
- Endereço (`Address`): `00000100` (posição 4).
- Instrução (`instrucao`): `10100000` (STORE).

Saída

- Memória na posição `00000100`: `10101010`.

Caso 6: Comparação (CMP A, B)

Descrição

Compara os valores dos registradores `A` e `B`.

Entradas

- Registrador `A`: `00001010` (10 em decimal).
- Registrador `B`: `00000101` (5 em decimal).
- Instrução (`instrucao`): `01010000` (CMP).

Saída

- Flag `flag_igual`: `0` (valores diferentes).
- Flag `flag_maior`: `1` (A > B).

Caso 7: Pular se Igual (JEQ endereço)

Descrição

Pula para um endereço específico se a comparação anterior indicar igualdade.

Entradas

- Resultado da comparação: `flag_igual = 1`.
- Endereço (`Address`): `00000100` (posição 4).
- Instrução (`instrucao`): `01110000` (JEQ).

Saída

- Contador de Programa (`PC`): `00000100`.
