

Relatório de Atividade da Disciplina Design de Computadores

Projeto 2: Processador compatível com MIPS DLX

Entrega Intermediária

Lais Nascimento da Silva

William Augusto Reis da Silva

João Guilherme Cintra de Freitas Almeida

Professor Paulo Carlos Santos

São Paulo

Novembro/2021

**Descrição do funcionamento do MIPS**

1. **Arquitetura:**

O MIPS DLX é uma simplificação da arquitetura original do MIPS, onde este é um processador de 32 bits. Este apresenta uma arquitetura do tipo Load/Store, a qual expressa que o processador só pode fazer operações lógicas e aritméticas entre registradores ou entre registradores e constantes imediatas. Além disso, essas operações ou valores imediatos podem ser implantados (Load) em uma memória ou carregados (store) de uma.

O MIPS DLX possui 32 registradores de uso geral. A denominação desses registradores segue a seguinte tabela:

**Uma imagem contendo Aplicativo

Descrição gerada automaticamente**

Esses registradores são guardados em um banco o qual podem ser acessados no máximo por 3 ao mesmo tempo (Instruções do tipo R).

Além dos registradores do banco, é importante lembrar do PC, o qual é um contador de programa, que armazena o endereço da instrução em execução.

Para que o MIPS implemente todas as funcionalidades propostas ao final ele apresenta-se da seguinte forma:

**Diagram, schematic

Description automatically generated**

1. **Instruções** 
   1. **Tipo de Registro (R):**

Grupo de instruções que realiza exclusivamente operações aritméticas (soma e subtração) e trabalha apenas com 3 registradores. Nelas, o **OpCode é sempre zero** e o **funct** expressa qual operação será realizada.

**Tabela

Descrição gerada automaticamente**

R[rd] = R[rs] operação(funct) R[rt]

Do expresso acima, é possível entender que Rd, especificado pelo endereço deste registrador será o registrador que receberá o resultado da operação entre Rs e Rt.

* 1. **Tipo Imediato (I):**

Para o projeto, as operações do tipo I serviram apenas para escrever valores na memória (sw), ler valores da memória (lw) e fazer desvios condicionais (beq).

Tabela

Descrição gerada automaticamente

Nesse grupo de instruções, a operação a ser executada (sw, lw ou beq) é o Opcode. A operação beq executa o papel de que se os registradores Rs e Rt forem iguais, o programa faz um desvio para a linha especificada pelo Imediato. Nas operações de lw e sw o registrador Rt recebe o valor da memória, onde o endereço da memória é especificado pelo Rs.

* 1. **Tipo de Salto (J):**

Operações a qual realiza o desvio operacional. Nela, o campo imediato especifica o endereço que o PC vai ler para fazer o desvio

**Tabela

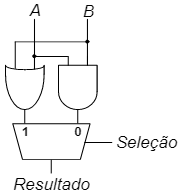
Descrição gerada automaticamente**

* **Opcode**: Contém o código da instrução a ser executada;
* **Rs**: O endereço (número em binário) do registrador com o primeiro operando da instrução definida em **opcode**;
* **Rt**: O endereço (número em binário) do registrador com o segundo operando da instrução definida em **opcode**;
* **Rd**: O endereço (número em binário) do registrador de destino para o resultado da instrução definida em **opcode**;
* **shamt**: Total de deslocamento (shift amount, número em binário), utilizado para as intruções de deslocamento e rotação.
* **funct**: Seleciona a operação que será aplicada aos operandos. O **opcode** só define que a instrução é do tipo R, deixando a definição da operação para o **funct**.

1. **Unidade Lógica e Aritmética para o MIPS DLX**

Para este projeto a Unidade Lógica e Aritmética (ULA) executa as seguintes instruções: Tipo R (ADD, SUB, AND, OR, SLT) e Tipo I (LW, SW, BEQ), no caso, tipo J não utilizada a ULA. Durante a execução do projeto, para entender melhor o funcionamento da ULA, ela foi implementada com 1 bit apenas e depois foi expandida para 32 bits. Além disso, é preciso analisar cada operação para aplicá-la na ULA.

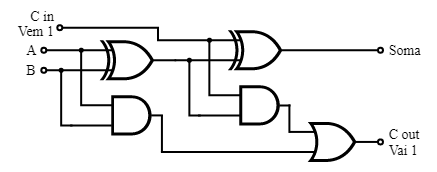
Iniciou-se o processo com as operações AND e OR:

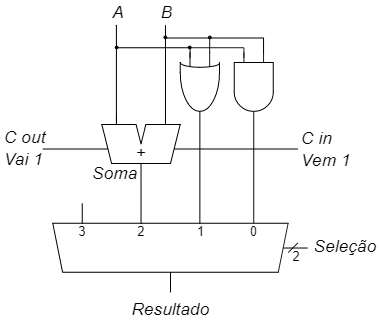


Para realizar estas 2 operações basta preencher as entradas A e B (1 bit cada), e selecionar qual das operações feitas pelas portas lógicas o MUX deve colocar na saída como resultado

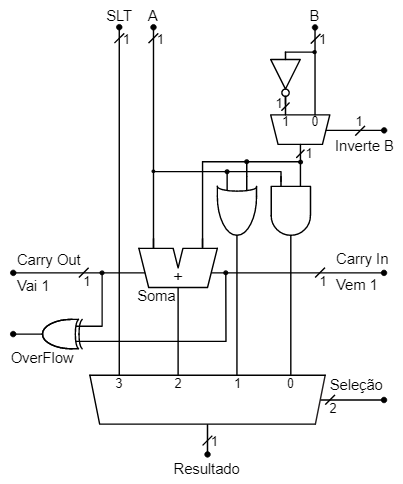
Tabela

Descrição gerada automaticamente com confiança baixa

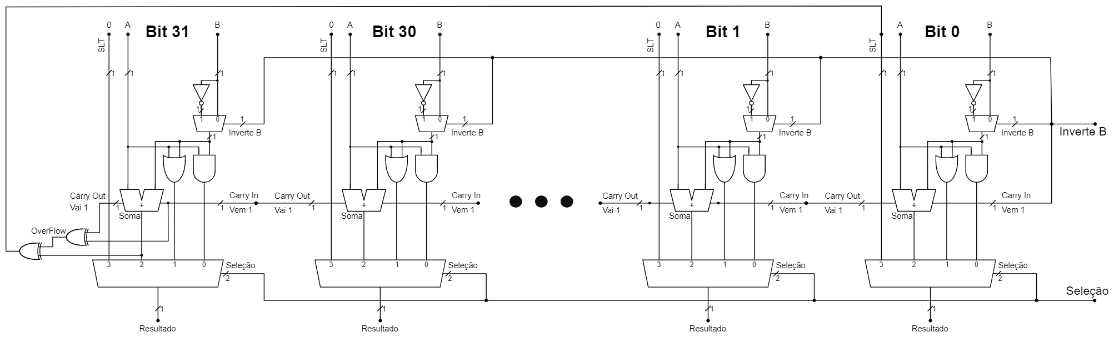
 Após este primeiro passo implementa-se o somador e insere mais está operação na ULA:



Para implementar a subtração tem-se que inverter a entrada B, ou seja, torná-la negativa e então realizar a operação de soma. Além disso, visto que a operação SLT compara se A<B, tem-se que com a subtração é possível implementá-la, pois basta subtrair e analisar o sinal do resultado, caso a subtração tenha sinal positivo A maior ou igual a B, então o resultado da ULA deve ser 0, se o sinal for negativo, A menor que B e então o resultado da ULA deve ser 1, para esta análise deve-se levar em conta apenas o ultimo bit da subtração e também o overflow.



Para finalizar, basta ligar a ter ao final os 32 bits:



Diagrama

Descrição gerada automaticamente E inserir a flag zero como saída:

1. **Unidade de Controle da ULA**

A unidade de controle da ULA deve analisar o funct, que vem da instrução quando é tipo R, e o ULAop, sinal este que sai da unidade de controle do fluxo de dados. O sinal que sai na Unidade de controle da ULA é o ULActrl.

O sinal ULAop tem 2 bits e segue as seguintes regras:

Texto

Descrição gerada automaticamente

O funct existe na instrução, quando é do tipo R, sendo os 6 primeiros bits da instrução, ou seja, os 6 menos significativos. Cada valor de funct representa e indica uma operação entre registradores

O ULActrl tem 3 bits, sendo que os 2 bits menos significativos formam o seletor dos MUX que formam a ULA, e o bit mais significativo indicar se inverte ou não a entrada B, a além entrar no “carry IN” do primeiro bit.

A unidade de controle da ULA pode ser resumida pela seguinte tabela:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instrução** **(opcode hex)** | **ULAop** **(N bits)** | **Funct** **(6 bits)** | **ULA** **Executa** | **ULActrl** **(4 bits)** |
| LW (0x23) (I) | 00 | 000000 | (add) | 010 |
| SW (0x2B) (I) | 00 | 000000 | (add) | 010 |
| BEQ (0x04) (I) | 01 | 000000 | (sub) | 110 |
| ADD (0x00) (R) | 10 | 100000 | (add) | 010 |
| SUB (0x00) (R) | 10 | 100010 | (sub) | 110 |
| AND (0x00) (R) | 10 | 100100 | (and) | 000 |
| OR (0x00) (R) | 10 | 100101 | (or) | 001 |
| SLT (0x00) (R) | 10 | 101010 | (slt) | 111 |

1. **Unidade de Controle**

A Unidade de Controle do Fluxo de Dados utiliza dos pontos de controle para realizar as suas funções com base no diagrama do tópico 2 desse relatório. Esses pontos são muito importantes para compreender as saídas dessa Unidade de Controle.

|  |  |
| --- | --- |
| Ponto de controle | utilização |
| Mux1 prox\_pc | Determina se irá continuar o fluxo de dados normalmente ou um desvio. |
| Mux2 rt/rd | Escolhe em qual registrador irá acessar – RT ou RD. |
| Hab escrita reg | Nome auto-explicativo, habilitando a escrita nos registradores (banco). |
| mux3 rt/imediato | Determina a entrada 1 da ULA, se é a saída do banco de registradores ou o valor imediato. |
| Mux4 ula/mem | Vai fazer a seleção entre o valor na memória ou a saída da ULA que irá para o banco de registradores. |
| BEQ | Como o próprio nome diz, se refere a instrução BEQ, sendo o que será habilidado somente quando a instrução for BEQ. Em outras, nunca é acionado. |
| Hab leitura mem | Ponto que habilita ou não a leitura da memória RAM. |
| Hab escrita mem | Ponto que habilita ou não a escrita na memória RAM. |
| UlaOP | Foi descrito no tópico anterior (5) quais são os possíveis valores e seus significados. Ele determina se irá ser soma, subtração ou análise da funct. Vai impactar no ULActrl. |

Com base nessas informações de utilização, partiu a se fazer a lógica da unidade de controle que foi implementada no código para fazer sentido o fluxo de dados com base nas instruções que se tem. A tabela utilizada foi a que estava no WORD de referência disponibilizado, ficando como abaixo.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Entrada** | **Saídas** | | | | | | | | |
| **Ins-trução** | **Opcode** | **Mux1**  **Prox\_PC** | **Mux2 Mux Rt/Rd** | **Hab** **Escrita** **Reg** | **Mux3 Rt/imediato** | **Mux4 ULA/mem** | **BEQ** | **Hab** **Le** **MEM** | **Hab** **Esc** **MEM** | **ULAop** |
| Tipo R | 000000 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 10 |
| LW | 100011 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 00 |
| SW | 101011 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 00 |
| BEQ | 000100 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 01 |
| J | 000010 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 |

1. **Instrução de uso:**

Para simular o projeto, basta compilá-lo para sua placa FPGA e se apropriar do KEY0 para passar para a próxima operação especificada no Projeto2.mif. Se o SW0 estiver em 0, os displays irão mostrar o valor de saída da ULA, caso contrário o valor do PC.