RELATÓRIO - IMPLEMENTAÇÃO DO FLUXO DE DADOS LOAD-STORE.

Grupo 03 - Integrantes:

• Celso Tadaki Sinoka – 13682851
• Felipe Luis Korbes – 13682893
• Lucas Suzin Bertan – 13682548

Introdução:

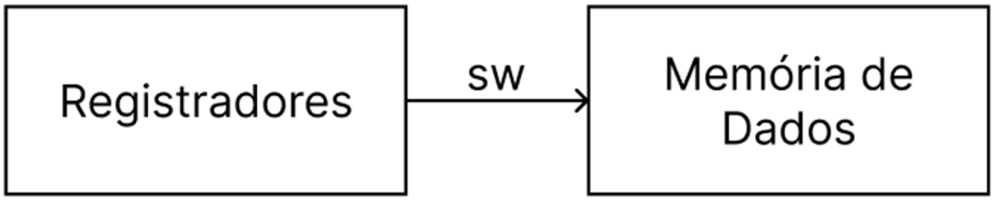
A proposta da aula 8 da disciplina foi implementar o fluxo de dados Load-Store, ou seja, o fluxo de dados da Memória de Dados para os Registradores do processador RISC-V 64-bits.

Os recursos propostos para o projeto foram:

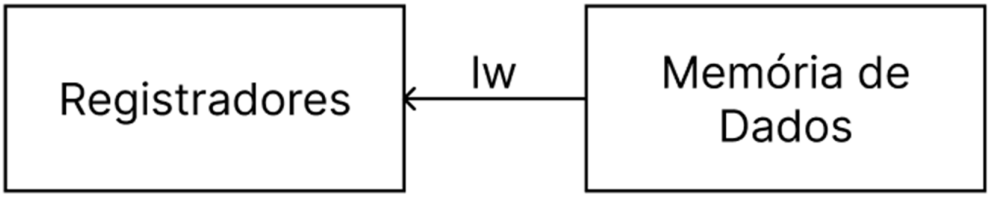
- Memória de 64 bits e 32 posições;
- Banco de Registradores seguindo a arquitetura RISC-V 64-bits:
  - 32 registradores de 64 bits endereçados de x0 a x31;
  - Registrador x0 *hard-wired* para ter valor 0.
- Somador-subtrator de 64 bits.

O grupo também definiu os recursos funcionais e não-funcionais do projeto:

- Requisitos funcionais:
  - Ser capaz de emular o comando de *load double word* (*lw*);



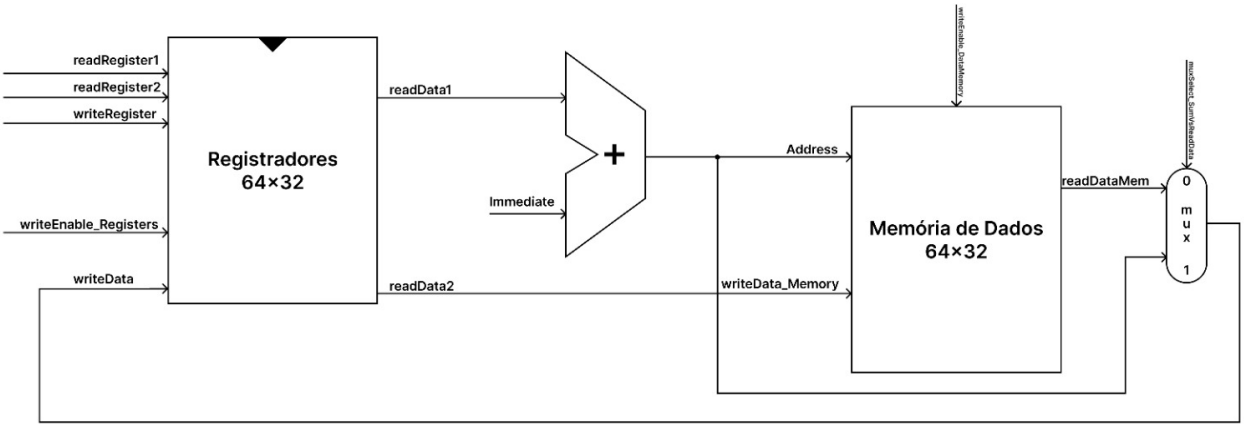
  - Ser capaz de emular o comando de *storage double word* (*sw*);



- Requisitos não-funcionais:
  - Ter a estrutura proposta pela arquitetura do RISC-V para que, por exemplo, o somador-subtrator só precise ser substituído pela ALU e o fluxo de dados continue funcionando normalmente.

Implementação:

Como todos os integrantes fizeram parte de algum grupo que implementou o RISC-V em Verilog na disciplina de SDI, utilizamos nossos conhecimentos para facilitar a implementação do *datapath*. Em uma reunião onde analisamos o fluxo de dados do RISC-V do livro de Hennessy&Patterson, chegamos na seguinte implementação:



Uma curiosidade é que o multiplexador só foi implementado para estar de acordo com o fluxo de dados do próprio RISC-V e para que o código atual seja 100% reutilizável na implementação final do processador deste semestre.

A Data Memory 64x32 foi implementada no formato de *bytes*, com 265 posições de 8 bits. Assim, o *immediate* deve ser múltiplo de 8.

Veja os exemplos do código Assembly e seu significado no circuito:

Exemplo 1 - lw

(1) lw x1, 10(x2)

- Lê o registrador x2:
  - readRegister1: x2;
  - readData1 = valor do registrador x2
- *Immediate* valendo 8.

(2) lw x1, 8(x2)

- Calcula valor do address (readRegister1 + *Immediate*).

(3) lw x1, 8(x2)

- writeRegister vale x1:
  - A palavra (writeData_Memory) da posição address é guardada no registrador de posição x1.

Observações: entrada do mux é sempre 0, writeEnable_Registers deve ser 1'b1 e writeEnable_Memory deve ser 1'b0. readRegister2 é *don't care*.

Exemplo 2 - sw

(1) sw x1, 8(x2)

- Lê o registrador x2:
  - readRegister1: x2;
  - readData1 = valor do registrador x2
- *Immediate* valendo 8.

(2) sw x1, 8(x2)

- Calcula valor do address (readRegister1 + *Immediate*).

(3) sw x1, 8(x2)

- readRegister2 vale x1:
  - A palavra do registrador de posição x1 (writeData) é guardada na posição da memória de endereço address.

Observações: entrada do mux é *don't care*, writeEnable_Registers deve ser 1'b0 e writeEnable_Memory deve ser 1'b1. writeRegister é *don't care*.

Simulação:

Como o circuito não possui *outputs*, optamos por analisar através do GTKWave. Aqui está uma descrição passo a passo da tomada de testes feitos.

LOAD:

x1 = (0 + x0); -> salva em x1 o valor 0 da Data Memory. (read data lerá o valor salvo na memória que é formada unindo as palavras de 8bits que estão de memory[0] até memory[7]);

STORE:

(0 + x0) = x1; -> salva na posição x0 + 0 da memória o valor do registrador x1 do banco de registradores. (Read Data terá o mesmo valor salvo nessa posição e o Write Data valerá o valor previamente salvo nessa posição);

INICIALIZAÇÃO: (0 a 10):

Inicialização dos sinais.

LOAD WORD 1: (10 a 20) -> lw x1, 0(x0):

Nessa operação, o valor guardado na posição X0 + 0 é salvo no registrador 1 do banco de registradores.

Esse valor, salvo de memory[0] ao memory[7] será mandado para o registrador x1 do banco de memória, esse valor é unido em um read data de 64 bits:

- O endereço[7:0] da Data Memory deverá ser 0, para colocar tal endereço em observação;
- O valor esperado para read_data será o valor salvo nesse endereço, no nosso caso, esse valor é 8.
- O endereço 1 deve ser mandado para o banco de registrador ligar o load respectivo; O que implica Write Register[4:0] = 5'b1;
- O enable Reg Write valerá 1, possibilitando salvar nesse respectivo registrador;
- Isso finaliza o ciclo do primeiro load, tendo o valor 8 salvo no registrador 1;

STORAGE WORD 1: (20 a 30) -> sw x1, 0(x0):

Em storage word, pegamos o valor previamente salvo em x1, (no caso 8) e salvaremos na posição x0 + 0 (a posição 0) do data memory. Em outras palavras:

- Endereço readRegister1 = 0 (pois estamos lendo o valor da posição x0); ReadRegister2 = 1 (mandaremos o valor do registrador 1 para o datamemory);
- ReadData1 = 0 (valor salvo na posição 0 do banco de registradores);
- ReadData2 = 8 (valor salvo na posição 1 do banco de registradores);
- memwrite = 1; (permitindo salvar um valor na memória);
- endereco[7:0] (da Data Memory) deverá receber o valor salvo x0 + 0, ou seja, recebido do somador. O valor esperado é 0;
- Isso finaliza o ciclo;

LOAD WORD 2: (30 a 40) -> lw x2, 0(x0):

Nesse ciclo, salvaremos o valor salvo na posição x0 + 0 da data memory no registrador x2.

- O endereço[7:0] da Data Memory deverá ser 0, para colocar tal endereço em observação;
- Nesse momento, o valor esperado será 8, e esse valor será mostrado no read_data.
- O endereço 2 deve ser mandado para o banco de registrador ligar o load respectivo; O que implica Write Register[4:0] = 5'd2;
- O enable Reg Write valerá 1, possibilitando salvar nesse respectivo registrador;
- Isso finaliza o ciclo do primeiro load, tendo o valor 8 salvo no registrador 2;

STORAGE WORD 2: (40 a 50) -> sw x2, 0(x0):

Salvamos o valor contido no registrador 2 na posição definida por x0 + 0.

- Endereço ReadRegister1 = 0 (pois estamos lendo o valor da posição x0); ReadRegister2 = 2 (mandaremos o valor do registrador 2 para o datamemory);
- ReadData1 = 0 (valor salvo na posição 0 do banco de registradores);
- ReadData2 = 8; (valor salvo na posição 2 do banco de registradores);
- memwrite = 1; (permitindo salvar um valor na memória);
- endereco[7:0] (da Data Memory) deverá receber o valor salvo x0 + 0, ou seja, recebido do somador. O valor esperado é 0;
- Isso finaliza o ciclo;

LOAD WORD 3: (50 a 60ns) -> x3, 16(x0):

Nesse ciclo, salvaremos o valor salvo na posição x0 + 16 da data memory no registrador x3.

- O endereço[7:0] da Data Memory deverá ser 16, para colocar tal endereço em observação;
- ReadRegister1 deverá valer 0, indicando a posição do registrador do qual se tira o x0.
- Immediate deverá ser 16;
- Nesse momento, o valor esperado será 16, e esse valor será mostrado no read_data.
- O endereço 3 deve ser mandado para o banco de registrador ligar o load respectivo; O que implica Write Register[4:0] = 5'd3;
- O enable Reg Write valerá 1, possibilitando salvar nesse respectivo registrador;
- Isso finaliza o ciclo do terceiro load, tendo o valor 16 salvo no registrador 3;

STORAGE WORD 3: (70 a 80ns) -> sw x2, 16(x2):

Guarda o conteúdo o conteúdo do registrador x2 em um endereço que é definido pelo conteúdo desse registrador acrescido de 16. -> O valor salvo é 8, logo o endereço deve ser 24, e o valor salvo na respectiva posição será 8 também.

- O endereço[7:0] da Data Memory deverá ser 24, para colocar tal endereço em observação;
- Immediate deverá ser 16;
- O valor esperado para o write data é 8, que é o valor salvo no registrador de posição 2 do banco de dados.
- O endereço 2 deve ser mandado para o banco de registrador ligar o load respectivo; O que implica Read Register1[4:0] = 5'd2;
- O enable Mem Write valerá 1, possibilitando salvar palavras na data memory;
- Isso finaliza o ciclo do terceiro storage, tendo o valor 8 salvo no endereço 24 da Data Memory;

LOAD WORD 4: (80 a 90 ns) -> lw x4, 16(x1):

- Obtemos o valor do registrador x1, acrescemos de 16, o valor total esperado será 24.
- O endereco[7:0] para o Data Memory valerá 24.
- O valor desse registrador deve ser 8, que é o valor salvo na operação de storage anterior.
- Esse valor será salvo no registrador x4, portanto, o endereço write register deve ser 4, Write Register[4:0] = 5'd4
- O write data do registrador deve ser o valor esperado, no caso, 8.

Sinais gerados pelo GTKwave comprovam o funcionamento e podem ser vistos abaixo.

1 - Tutorial de como usar o GTKwave:

Se você é um usuário do icarus verilog, certamente deve ter o GTKwave instalado por tabela. Caso contrário, vá ao site oficial e realize o download para o seu sistema operacional.

- Com o icarus verilog, abra o terminal na pasta dos arquivos com os módulos e execute o arquivo principal, que chamamos de index.v (uma referência ao arquivo principal de um site em html, que une os demais arquivos em algo apresentável e funcional).

- No terminal digite, digite o comando: iverilog index.v index_tb.v

- Isso fornecerá um arquivo de saída: a.out

- No Linux, faremos: ./a.out

- No Windows: vvp a.out

- Isso fornecerá um arquivo da extensão .vcd. teremos o arquivo: wave.vcd

- Agora, insira o comando: gtkwave wave.vcd para executar o arquivo mencionado no gtkwave.

- Adicione os sinais mostrados na parte 2: "Como entender os sinais pelo gtkwave" e a figura observada deve ser a mesma que foi obtida e anexada nesse documento.

2 - Como entender os sinais pelo gtkwave:

- **Immediate:** Valor constante que é fornecido pelo usuário do teste e que será sempre acrescido no dado ciclo no somador para fornecer o endereço.

- **Clk:** O sinal de clock do sistema.

- **Endereço:** O endereço a se observar no Data Memory.

- **read_data:** O valor que é lido e que está salvo em dada posição do Data Memory, essa posição citada é justamente o endereço dado pelo sinal anterior.

- **write_data:** O valor que será fornecido ao Data Memory para ser salvo, a posição no qual esse valor é salvo é indicado pelo sinal de endereço também.

- **mem_read:** É o enable de leitura do DM.

- **mem_write:** Habilita o salvamento de valores no DM.

- **readRegister1:** Indica a posição do primeiro registrador que será lido, esse valor fornece a posição do registrador cujo valor será acrescido com o immediate no somador para fornecer o valor de endereço para o Data Memory.

- **readRegister2:** Indica o endereço do segundo registrador lido, esse registrador fornece o valor que será enviado para o Data Memory salvar.

- **readData1:** O próprio valor que está na posição determinada pelo readRegister1.

- **readData2:** O valor que está na posição determinada pelo readRegister2.

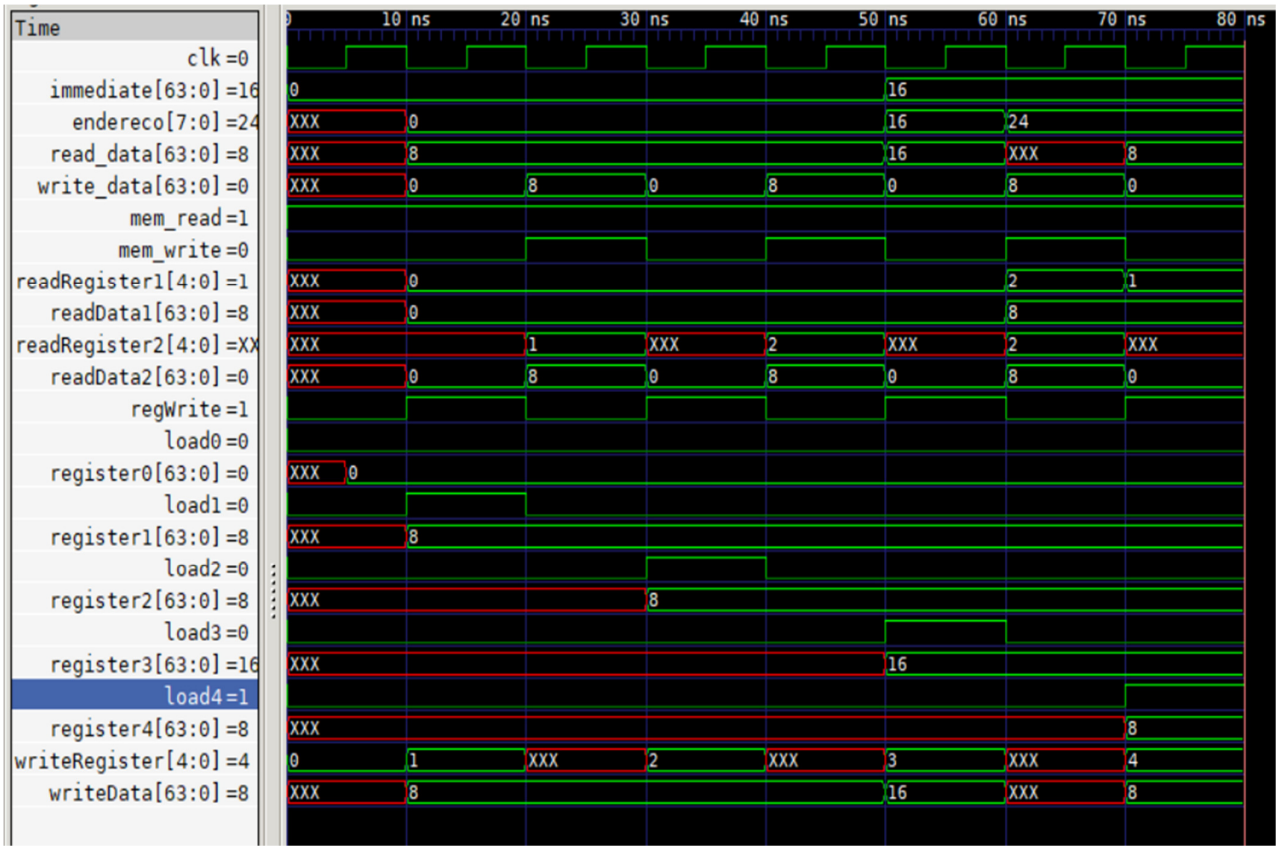
- **regWrite:** Habilita o salvamento de valores no registrador.

- **load"n":** Permite salvar em um registrador n especifico.

- **Register"n":** Indica o valor salvo no registrador n.

- **Write Register:** Indica o endereço do registrador que será usado para o salvamento do valor de Write Data.

- **Write Data:** O valor salvo no registrador, a posição a ser usada para o salvamento é indicada pelo endereço fornecido pelo sinal anterior (write Register).

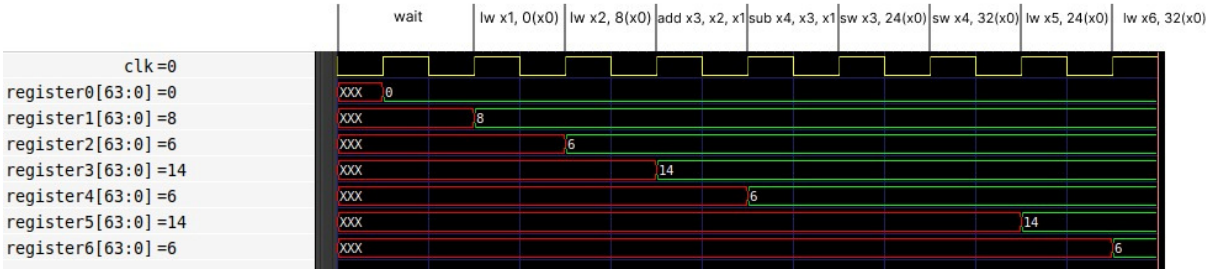


Tarefa da aula 10 (26/04):

Nesse aula foi pedido para produzir um conjunto de instruções RISC-V no módulo de load-store implementado acima. As instruções a serem executadas são as seguintes:

1. $lw\ x1, 8(x0)$
2. $lw\ x2, 16(x0)$
3. $add\ x3, x2, x1$
4. $sub\ x4, x3, x1$
5. $sw\ x3, 24(x0)$
6. $sw\ x4, 32(x0)$

Abaixo está a wave form do nosso circuito em funcionamento com a adição de instruções de load nos registradores 5 e 6 dos valores dos registradores 3 e 4.



Uma imagem mais detalhada com os sinais das operações acima pode ser vista abaixo. Os altos valores em read_data são devidos ao fato da memória estar sempre lendo os valores vindos do somador como endereço mesmo quando não necessário, o que resulta em leituras de posições da memória fora de lugar.