Grupo: Clara Dias Guimarães, Gabriela Alves Fernandes Vieira, Letícia Santos Xavier, Roni Victor Nicolau Oliveira.

1a Entrega: Aritmética, lógica e deslocamentos

Nesta etapa do projeto, implementamos a alu (Unidade Lógica e Aritmética) e o ALUController. O objetivo foi permitir que o processador executasse operações matemáticas, lógicas, deslocamentos de bits e avaliações de condições de desvio (branches).

O ALUController recebe o ALUOp do Controller, além dos campos Funct3 (3 bits) e Funct7 (7 bits) da instrução atual para ser possível fazer a diferenciação entre as instruções. Então ele decodifica as instruções e gera um sinal de controle de 4 bits (Operation) para a ALU.

A ALU foi projetada como um bloco combinacional que recebe dois operandos (SrcA e SrcB) e executa a operação selecionada pelo sinal Operation. As principais características da implementação incluem:

1. **Aritmética e Lógica:** Suporte para adição, subtração, AND, OR e XOR.
2. **Deslocamentos (Shifts):** Implementação de deslocamentos à esquerda e à direita.
3. **Comparação e Branches:** A ALU foi configurada para ajudar na lógica de desvios. Para operações como BEQ (Branch if Equal) ou BLT (Branch if Less Than), a ALU compara os operandos e retorna 1 (verdadeiro) ou 0 (falso) no ALUResult. Isso simplifica a lógica de decisão do *Branch* no estágio de memória ou *writeback*.

2a Entrega: Acesso à memória

Nesta etapa, foi implementado o módulo datamemory, responsável por gerenciar as operações de leitura e escrita (Load/Store). Ele atua como uma interface de adaptação entre o pipeline do processador e o bloco de memória física (Memoria32Data). O principal desafio superado nesta implementação foi o tratamento de dados de diferentes tamanhos (Byte, Half-word, Word).

Gerenciamento de Escrita (Store Instructions)

Para suportar instruções de escrita parcial, como SB (Store Byte) e SH (Store Half), sem destruir os dados vizinhos na mesma palavra, usamos a técnica de *Byte Enables* (sinal Wr de 4 bits).

Essa lógica de escrita utiliza o campo Funct3 pra definir o tamanho do dado (Word, Half, Byte). Para tamanhos menores que uma palavra, os bits menos significativos do endereço (a[1:0]) são decodificados para ativar apenas os bits correspondentes no sinal Wr. Para simplificar, o dado a ser escrito (wd) é replicado no barramento de entrada Datain. No caso de um SB, o byte menos significativo é replicado 4 vezes ({4{wd[7:0]}}). Combinado com a

máscara Wr, garante que o byte correto seja escrito na posição correta da memória, independentemente do alinhamento.

Gerenciamento de Leitura (Load Instructions)

Nas operações de leitura, o módulo recebe a palavra de 32 bits da memória e concatena de acordo com a instrução (Funct3):

- Extensão de Sinal: Para LB e LH, o módulo extrai o byte ou half-word relevante e realiza a extensão de sinal para 32 bits, preservando a representação numérica em complemento de dois.
- Extensão de Zero: Para LBU, o byte é extraído e os bits superiores são preenchidos com zero, tratando o dado como um número sem sinal.

Mapeamento de Endereços

O módulo também realiza o tratamento dos endereços. Enquanto o RISC-V utiliza endereçamento a byte, a memória física é acessada por palavras. O módulo vai ajustar o waddress (endereço de escrita) ignorando os dois bits menos significativos (a[8:2]).

3a Entrega: Aritmética, lógica e deslocamentos com valores imediatos

Nesta etapa, foi implementado o módulo imm_Gen, um circuito combinacional responsável por extrair e estender os imediatos embutidos nas instruções de 32 bits. Como a arquitetura RISC-V utiliza formatos de instrução fixos, os bits que representam valores numéricos variam de posição dependendo do tipo da instrução (I, S, B, ou J). O módulo padroniza esses valores para uma saída de 32 bits (Imm_out).

Decodificação por Tipo de Instrução

O módulo opera verificando o Opcode (inst_code[6:0]) e aplicando a lógica de reconstrução adequada:

1. Tipo I (Immediate - Loads, JALR, Aritmética):
    - O módulo realiza a Extensão de Sinal replicando o bit mais significativo para os 20 bits superiores da saída, permitindo operações com números negativos (complemento de dois).
    - Para instruções de deslocamento (SLLI, SRLI, SRAI), identificadas pelo Funct3, o campo de imediato é de apenas 5 bits. Neste caso específico, o código aplica Extensão de Zero, pois a quantidade de deslocamento é sempre um valor positivo não assinado.
2. Tipo S (Store):
    - Neste formato, o imediato de 12 bits é dividido em duas partes na instrução para manter os registradores nos mesmos locais. O módulo concatena os bits [31:25] e [11:7] e aplica a extensão de sinal.
3. Tipo B (Branch) e Tipo J (Jump - JAL):

- Estas instruções possuem o imediato embaralhado para otimizar o datapath físico. O módulo reconstrói o valor original reordenando os bits.
- Bit Implícito: O módulo insere explicitamente um 1'b0 na posição 0 da saída, aumentando o alcance efetivo do salto (13 bits para Branch e 21 bits para JAL).

4a Entrega: Desvios e jumps

Nesta etapa, o processador ganhou a capacidade de executar instruções de quebra de fluxo, essenciais para a implementação de laços de repetição, condicionais e chamadas de função. Para isso, foi desenvolvido o módulo BranchUnit e alterada a lógica do Datapath para lidar com Hazards de Controle através de técnicas de flushing (limpeza de pipeline).

Unidade de Desvio (BranchUnit)

A BranchUnit opera no estágio de Execução e tem duas funções primárias: calcular o endereço de destino do salto e decidir se o salto deve ser tomado ou não.

1. Cálculo de Endereço:
   - Endereçamento PC-Relativo: Para instruções de Branch e Salto Incondicional, o endereço alvo é calculado somando o Program Counter atual ao imediato (PC + Imm).
   - Endereçamento Indireto: Pro JALR, o endereço é calculado somando o valor do registrador base ao imediato (Reg1 + Imm). A distinção entre JAL e JALR é feita verificando o bit 3 da instrução, uma otimização de decodificação.
2. Lógica de Decisão: O sinal de seleção do PC (PcSel) é ativado se:
   - Ocorrer um salto incondicional (JAL/JALR); ou
   - Ocorrer uma instrução de Branch (Branch == 1) E a condição avaliada pela ALU for verdadeira (AluResult[0] == 1).

Integração no Pipeline e Resolução de Hazards

Como a decisão de desvio ocorre no estágio de Execução, o processador já buscou a instrução sequencial (PC+4) e a armazenou no registrador de pipeline IF/ID. Quando um desvio é tomado (PcSel = 1), essa instrução buscada torna-se inválida. Para resolver este Hazard de Controle, foi implementada uma lógica de Flush síncrono no Datapath. Quando o sinal PcSel é ativado o Multiplexador do PC (pcmux) seleciona o BrPC (endereço calculado pela BranchUnit) em vez de PC+4. O registrador (A) é resetado (preenchido com zeros/NOP), descartando efetivamente a instrução incorreta antes que ela avance para o estágio de decodificação.

Forwarding para JALR

Um ponto crítico foi a conexão da entrada Reg1 da BranchUnit à saída do multiplexador de Forwarding (FAmux_Result). Isso garante que, se uma instrução JALR depender de um valor calculado pela instrução imediatamente anterior, o endereço de salto seja calculado

com o dado correto e atualizado, evitando erros de dependência de dados no cálculo de endereço.

5a Entrega: Halt

Para permitir o encerramento controlado da execução de programas e a finalização automática das simulações, foi implementado o Halt. Diferente de um reset, que reinicia o processador, o Halt tem a função de congelar o estado da máquina, preservando os valores finais nos registradores e na memória de dados para verificação posterior.

Lógica de Controle e Propagação

Durante a implementação da funcionalidade, modificamos o Controlador e o Datapath:

1. Decodificação: O Controlador foi expandido para reconhecer um opcode específico reservado para o encerramento do programa. Ao identificar esta instrução, o sinal de controle Halt é ativado.
2. Congelamento do Program Counter (PC): No Datapath, o registrador do PC recebeu uma lógica de bloqueio adicional. O sinal de habilitação de escrita do PC passa a depender da condição Reg_Stall | Halt. Ou seja, se o sinal Halt estiver ativo, o PC ignora o valor de Next_PC e mantém seu valor atual indefinidamente, interrompendo a busca de novos endereços.
3. Bloqueio do Pipeline (Estágio IF/ID): Para evitar que instruções espúrias entrem no pipeline após a parada, a lógica de escrita do registrador de barreira IF/ID foi alterada para !Reg_Stall && !Halt. Isso garante que o estágio de busca seja efetivamente desativado.