Datapath Monociclo[1]

Beatriz Caroline Moggio[1], Isabela Camille Barotto Janguas[2]

Universidade Tecnológica Federal do Paraná – UTFPR

COCIC – Coordenação do Curso de Bacharelado em Ciência da Computação
Campo Mourão, Paraná, Brasil

[1]beatrizcarolinemoggio@alunos.utfpr.edu.br
[2]isabelacamille@alunos.utfpr.edu.br

[1] Trabalho desenvolvido para a disciplina de BCC2003 – Arquitetura e Organização de Computadores

Resumo

Este projeto tem como objetivo apresentar o desenvolvimento de um Datapath Monociclo, implementado por meio do software de simulação de circuitos digitais Logisim Generic 2.7.1. A proposta consiste na criação de um caminho de dados capaz de executar instruções do tipo R, I e J, conforme a arquitetura MIPS. Para isso, são apresentados a implementação e funcionamento de diversos componentes essenciais, como: Decodificador, Banco de Registradores, Unidade Lógica Aritmética (ULA), Unidade de Controle, Unidade Controle da ULA, Memória de Instruções e Memória de Dados. O objetivo é demonstrar o funcionamento integrado das partes, por meio da execução de instruções em linguagem de máquina obtidas a partir da tradução do código escrito em linguagem de alto nível.

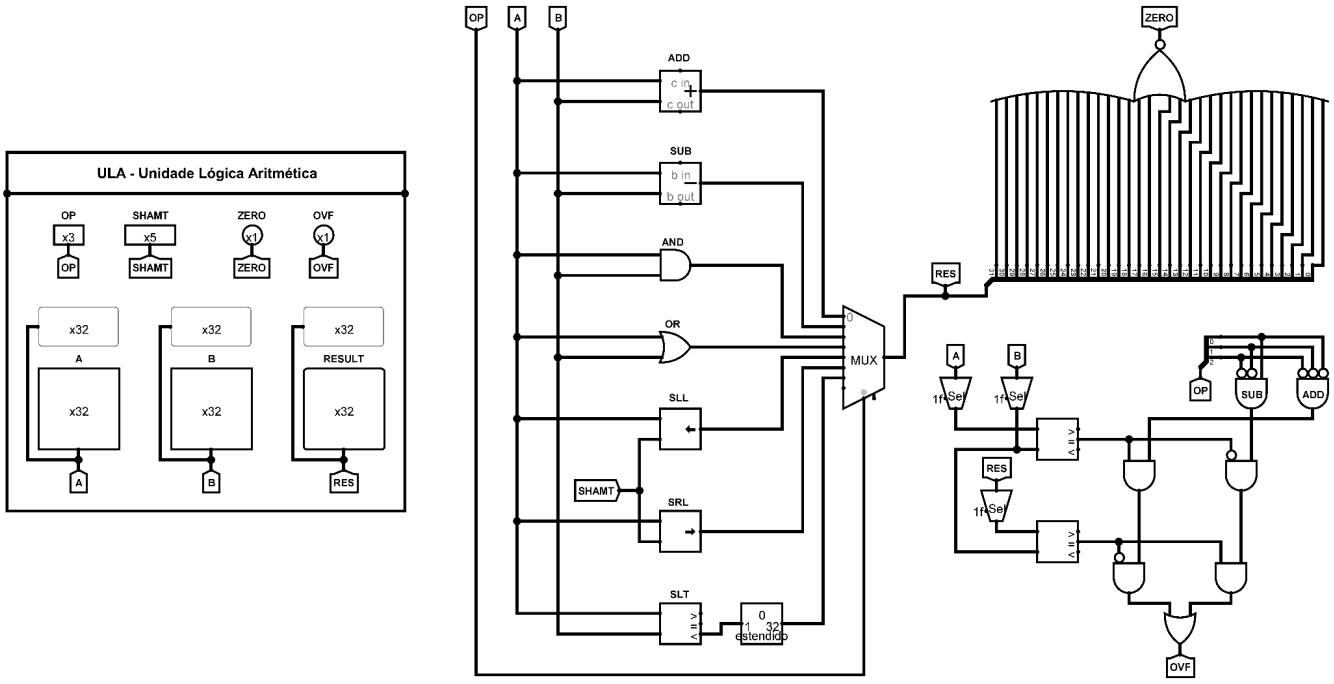
1. Introdução

A arquitetura MIPS é amplamente utilizada no ensino de Arquitetura de Computadores por ser uma representação clara e simplificada do funcionamento de um processador. Este trabalho tem como objetivo desenvolver um Datapath do tipo monociclo, onde cada instrução é executada em apenas um ciclo de clock. A ideia é demonstrar, de forma prática, como as instruções são interpretadas e executadas dentro do processador. Para isso, foram desenvolvidos componentes como a Unidade de Controle, que coordena as ações internas do processador com base na instrução que está sendo lida, e a Unidade de Controle da ULA, responsável por definir as operações lógicas e aritméticas que devem ser realizadas. O projeto foi desenvolvido no simulador Logisim, com base em instruções do conjunto MIPS, como add, sub, lw, sw, beq e j. Para garantir o funcionamento correto, foi desenvolvido um algoritmo simples em linguagem C, traduzido para Assembly, convertido para linguagem de máquina (hexadecimal), e testado no circuito implementado. Este trabalho busca contribuir para uma melhor compreensão dos princípios básicos de um processador.

2. ULA (Unidade Lógica Aritmética)

A ULA implementada é capaz de realizar operações aritméticas e lógicas de até 32 bits. Ela possui duas entradas principais, denotadas por A e B, que representam os operandos, um sinal de controle da operação (OP), de 3 bits, e um campo SHAMT, de 5 bits, para as operações de deslocamento. A saída principal é o resultado da operação, com 32 bits, além das flags ZERO e OVF (overflow). O resultado final é determinado por um multiplexador (MUX), que seleciona qual operação será executada com base no valor do sinal OP (operação).

Figura 1: *Componentes da ULA*



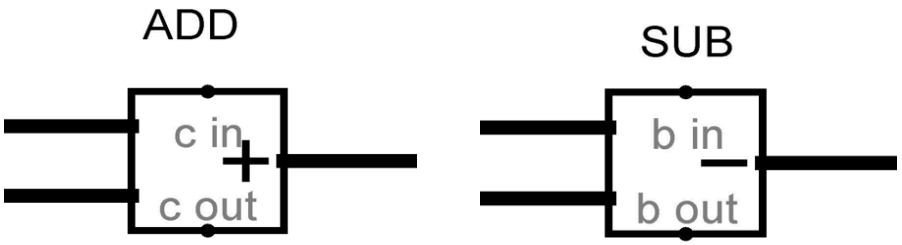
Fonte: Autoria Própria, 2025.

2.1 Componentes utilizados para a implementação da ULA

Os componentes utilizados do Logisim para a implementação da ULA foram:

2.1.1 Operações Aritméticas: Para realizar as operações aritméticas de adição (ADD) e subtração (SUB), foram utilizados os blocos prontos de somador e subtrator do Logisim, facilitando a construção de circuitos eficientes para essas instruções.

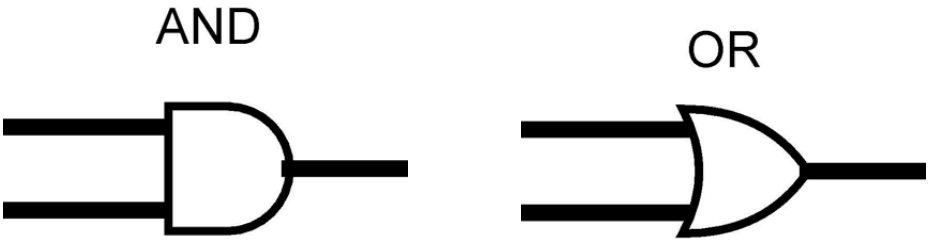
Figura 2: *Operações Aritméticas*



Fonte: Autoria Própria, 2025.

2.1.2 Portas Lógicas: As operações lógicas AND e OR foram implementadas utilizando portas lógicas disponíveis no próprio simulador Logisim.

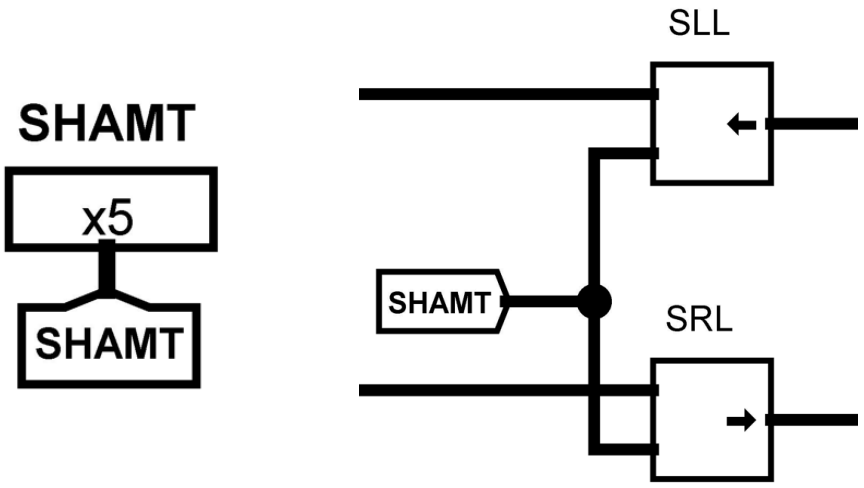
Figura 3: *Portas Lógicas*



Fonte: Autoria Própria, 2025.

2.1.3 Operações Deslocamento: Para realizar as operações lógicas de SLL (Shift Left Logical) e SRL (Shift Right Logical), foram utilizadas ferramentas de deslocamento do Logisim. Esses componentes permitem deslocar os bits do operando A com base no valor definido pelo campo SHAMT.

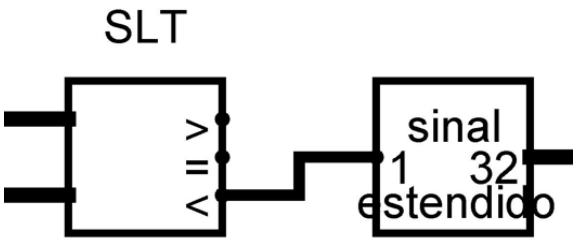
Figura 4: *Operações Lógicas*



Fonte: Autoria Própria, 2025.

2.1.4 Operação Condicional: Para realizar a operação condicional SLT (Set if Less Than), foi utilizado um comparador que verifica se o operando A é menor que o operando B. A saída booleana, de 1 bit, do comparador é conectada a um extensor de bits, que converte o valor para 32 bits, mantendo a padronização da saída da ULA.

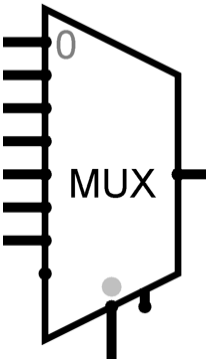
Figura 5: *Operação Condicional*



Fonte: Autoria Própria, 2025.

2.1.5 Multiplexador: Um multiplexador (MUX) foi utilizado para selecionar, entre os resultados das operações implementadas, qual será encaminhado à saída. Essa seleção ocorre com base no valor do sinal de controle da operação (OP).

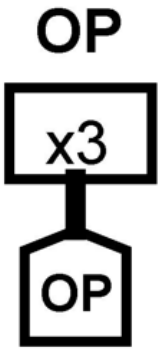
Figura 6: *Multiplexador*



Fonte: Autoria Própria, 2025.

2.1.6 Sinal de Controle da Operação: O sinal de controle da operação é um valor binário, de 3 bits, responsável por definir qual operação a ULA deve executar. Esse valor é interpretado pelo MUX para selecionar o resultado adequado.

Figura 7: *Sinal de Controle da Operação*

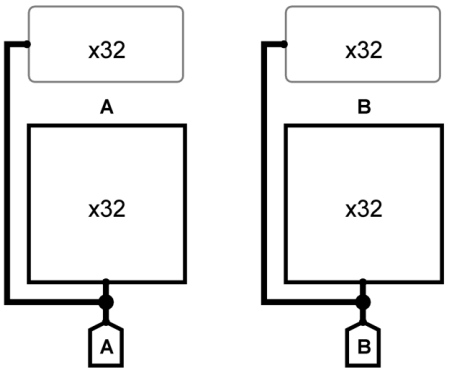


Fonte: Autoria Própria, 2025.

2.1.7 Shamt: O campo SHAMT (Shift Amount) foi implementado com um pino de 5 bits, utilizado exclusivamente nas operações de deslocamento SLL (Shift Left Logical) e SRL (Shift Right Logical) para definir quantas posições os bits devem ser deslocados. *Referência visual: Figura 4*.

2.1.8 Operandos: Dois pinos de entrada de 32 bits, identificados como A e B, foram utilizados para fornecer os valores binários a serem processados pela ULA.

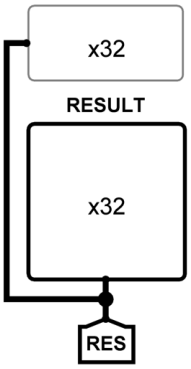
Figura 8: *Operandos*



Fonte: Autoria Própria, 2025.

2.1.9 Resultado da Operação: A saída da ULA consiste em um pino de 32 bits, que representa o resultado final da operação realizada com base nos operandos e no valor do sinal de controle da operação (OP).

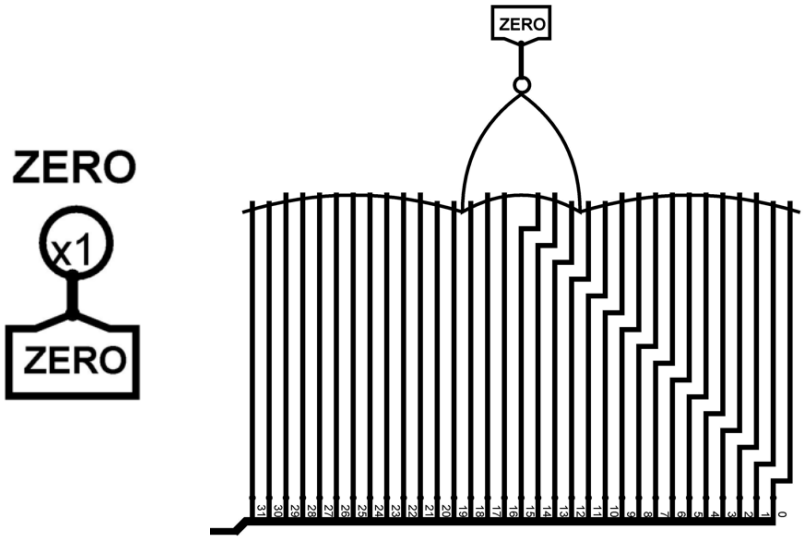
Figura 9: *Resultado da Operação*



Fonte: Autoria Própria, 2025.

2.1.10 Zero: A flag ZERO foi implementada utilizando uma porta NOR com 32 entradas, conectadas à saída da ULA. Quando o resultado da operação é igual a zero, a saída será 1; caso contrário, será 0.

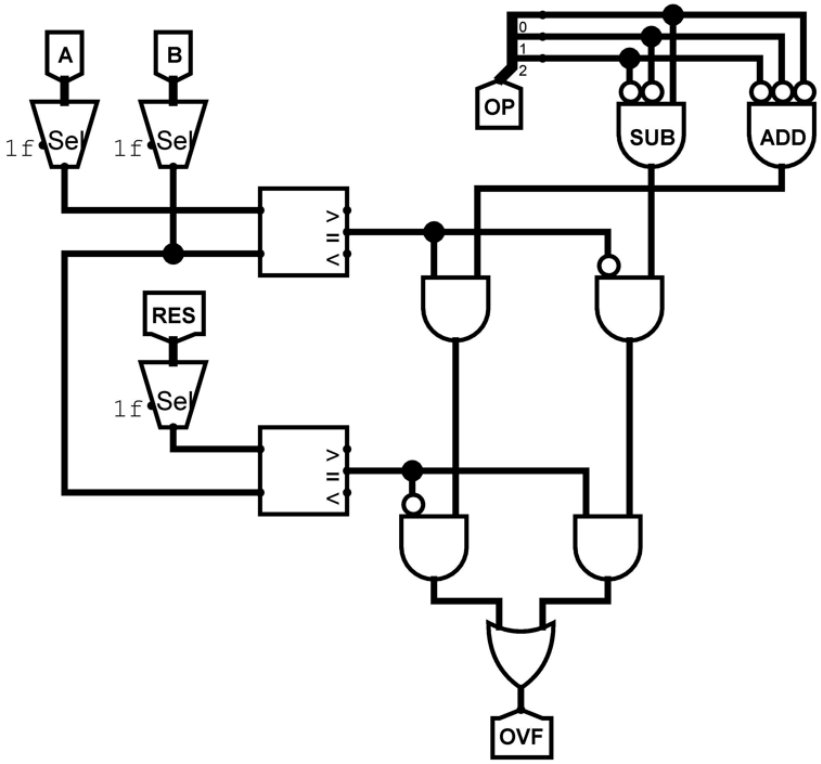
Figura 10: *Zero*



Fonte: Autoria Própria, 2025.

2.1.11 Overflow: O circuito de overflow foi implementado para identificar situações em que o resultado de uma soma ou subtração ultrapassa o limite que pode ser representado com 32 bits (em complemento de dois). Na soma, ocorre o overflow quando os bits mais significativos (bit 31) de A e B são iguais, mas o resultado é diferente. Na subtração, a lógica é parecida: ocorre overflow se o bit mais significativo de B for igual ao do resultado, mas o de A for diferente. Se uma dessas comparações forem atendidas, a flag OVF (overflow) é ativada.

Figura 11: *Overflow*



Fonte: Autoria Própria, 2025.

2.2 Funcionamento da ULA

A ULA recebe como entrada dois operandos de 32 bits, identificados como A e B, um sinal de controle da operação (OP), de 3 bits, responsável por definir a operação a ser executada, e o campo SHAMT (Shift Amount), de 5 bits, utilizado nas operações de deslocamento. As saídas são compostas pelo resultado da operação escolhida, de 32 bits, além das flags ZERO, de 1 bit, que indica se o resultado é igual a zero, e OVF (overflow), também de 1 bit, que indica quando a operação aritmética excede os limites em complemento de dois.

As operações implementadas na ULA são:

2.2.1 Somador: realiza operações de adição entre os operandos A e B.

2.2.2 Subtrator: realiza operações de subtração entre os operandos A e B.

2.2.3 AND: Realiza uma comparação bit a bit entre os operandos A e B. O resultado será 1 somente quando ambos os bits nas mesmas posições forem iguais a 1; caso contrário, o resultado será 0.

2.2.4 OR: Realiza uma comparação bit a bit entre os operandos A e B. O resultado será 1 se pelo menos um dos bits for 1.

2.2.5 SLL: Realiza operação de deslocamento para a esquerda dos bits do operando A.

2.2.6 SRL: Realiza a operação de deslocamento para a direita dos bits do operando A.

2.2.7 SLT: Compara se o operando A é menor que o operando B. O resultado é 1 se a condição for verdadeira, e 0 caso contrário.

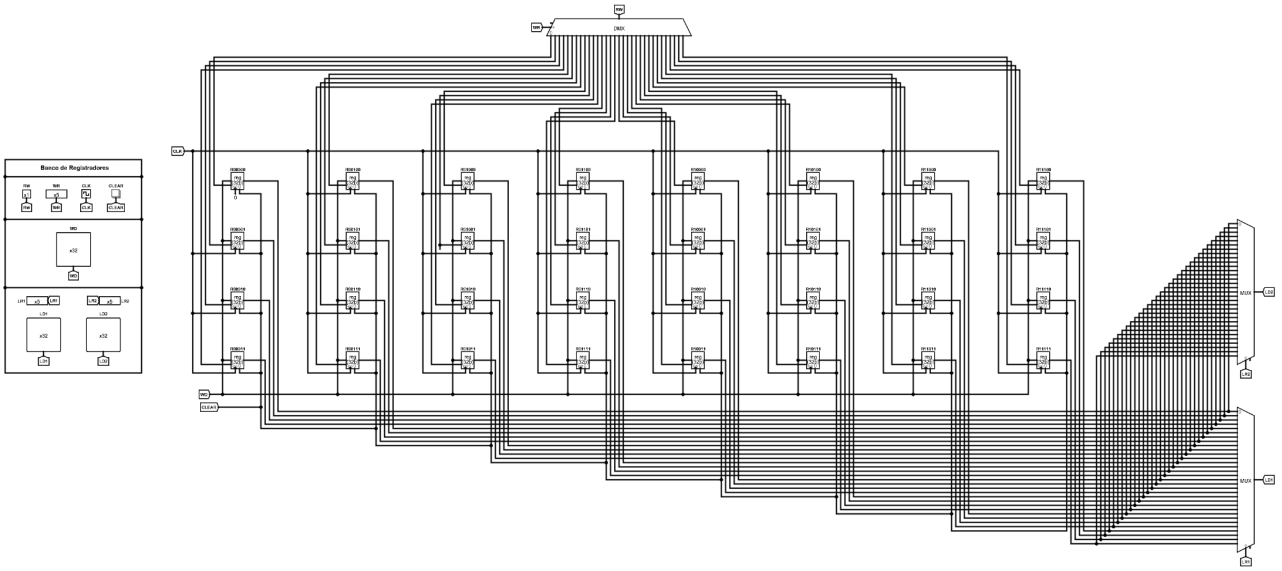
Tabela 1: *Sinais das Operações*

Operação	Sinal da Operação
ADD	000
SUB	001
AND	010
OR	011
SLL	100
SRL	101
SLT	110

3. Banco de Registradores

O Banco de Registradores implementado é uma estrutura capaz de realizar a leitura e o armazenamento de valores de até 32 bits em 32 registradores. Possui três entradas de 5 bits para seleção dos registradores (WR, LR1 e LR2), uma entrada de dados WD de 32 bits, informando o valor que se deseja armazenar, e uma entrada de controle RegWrite de 1 bit que habilita a escrita nos registradores. As duas saídas do circuito apresentam os valores armazenados nos registradores selecionados.

Figura 12: *Banco de Registradores*



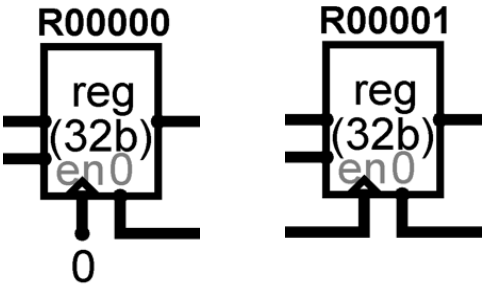
Fonte: Autoria Própria, 2025.

3.1 Componentes utilizados para a implementação do Banco de Registradores

Os componentes utilizados do Logisim para a implementação do Banco de Registradores são:

3.1.1 Registradores: Foram utilizados 32 registradores de 32 bits disponíveis no simulador Logisim, responsáveis pelo armazenamento dos dados. O registrador 00000 é configurado para sempre armazenar o valor 0. Para isso, sua entrada de clock está conectada a constante 0, impedindo a operação de escrita.

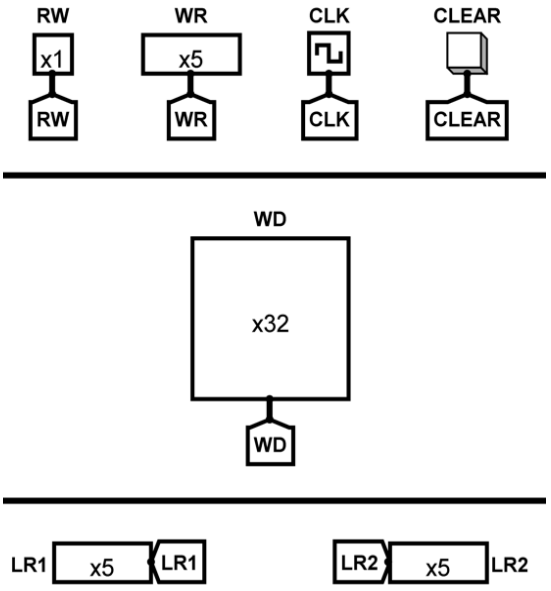
Figura 13: *Registradores 00000 e 00001*



Fonte: Autoria Própria, 2025.

3.1.2 Entradas de seleção e de dados: O Banco de Registradores possui três entradas de 5 bits (WR, LR1 e LR2), responsáveis respectivamente, pela seleção do registrador de escrita e dos dois registradores de leitura. A entrada WD, com 32 bits, fornece o dado a ser escrito. O controle de escrita é feito pela entrada de 1 bit RegWrite, que habilita ou não a escrita nos registradores. Há também um botão Clear, que limpa todos os registradores quando pressionado, independente dos demais sinais de controle. Por fim, há uma entrada de clock, que garante o funcionamento síncrono: a escrita só ocorre na borda de subida do clock e com RegWrite ativado.

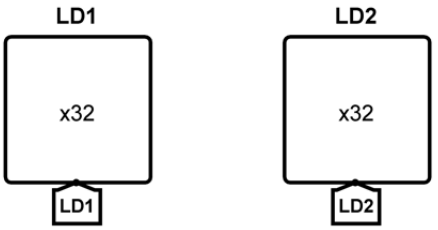
Figura 14: *Entradas de bits*



Fonte: Autoria Própria, 2025.

3.1.3 Saída de dados: O Banco de Registradores possui duas saídas de 32 bits (LD1 e LD2) que apresentam os valores armazenados nos registradores selecionados nas entradas LR1 e LR2.

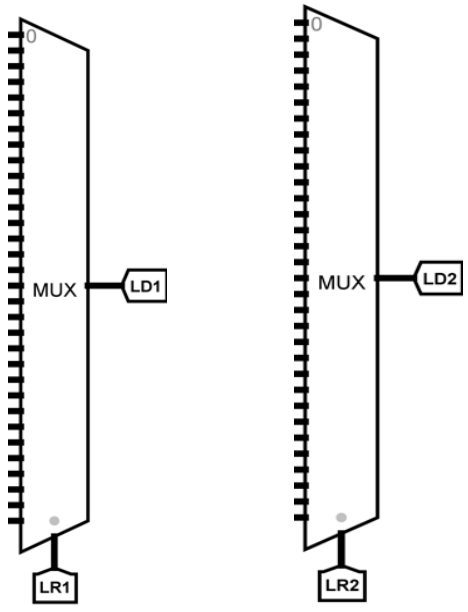
Figura 15: *Saída de dados*



Fonte: Autoria Própria, 2025.

3.1.4 Multiplexadores: Foram utilizados dois Multiplexadores (MUX) de 32 bits de dados, disponíveis no simulador Logisim, que recebem as saídas dos valores armazenados em cada um dos 32 registradores, e apresentam nas saídas LD1 e LD2 o valor salvo no registrador selecionado em LR1 e LR2, respectivamente.

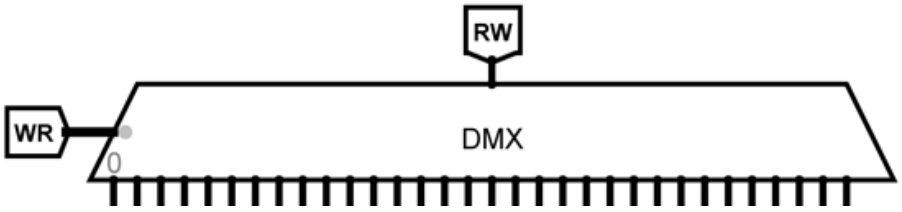
Figura 16: *Multiplexadores*



Fonte: Autoria Própria, 2025.

3.1.5 Demultiplexador: Foi utilizado um Demultiplexador (DEMUX) disponível no simulador Logisim, responsável pelo controle da função de escrita, selecionando o registrador que receberá o dado que deseja ser armazenado.

Figura 17: *Demultiplexador*



Fonte: Autoria Própria, 2025.

3.1.6 Clock: O clock, disponível no simulador Logisim, foi usado para a leitura e escrita nos registradores, que estão configurados para a borda de subida do clock.

3.1.7 Clear: Foi utilizado um botão disponível no simulador que foi ligado a todas as entradas 0 dos registradores, permitindo que todos eles possam receber o valor 0 caso o botão seja acionado.

Figura 18: *Botão Clear*

Fonte: Autoria Própria, 2025.

3.2 Funcionamento do Banco de Registradores

O banco de registradores possui a função de armazenar valores e disponibilizá-los ao processador para a realização de operações. Dessa forma, seu funcionamento pode ser dividido em duas partes, a de escrita e a de leitura dos registradores.

3.2.1 Escrita nos registradores: As entradas utilizadas para realizar o armazenamento de um valor em um registrador são:

- RegWrite (1 bit): Quando possui o valor 1 habilita a escrita nos registradores. É conectado na entrada do DEMUX.
- WR (5 bits): Realiza a seleção do registrador de destino do dado que deseja ser armazenado. Ele é conectado na porta de seleção do DEMUX.
- WD (32 bits): É a entrada responsável por informar o valor que será armazenado. Ela é enviada para a entrada de todos os registradores, porém somente será armazenada no que estiver sendo selecionado pelo DEMUX.

  Dessa forma, quando RegWrite possuir valor 1, o DEMUX é habilitado, ativando o fio correspondente ao registrador selecionado por WR. O dado presente em WD, distribuído para todos os registradores, será armazenado somente no registrador selecionado, no momento da borda de subida do clock.

  Para redefinir todos os registradores ao valor zero, o botão Clear é pressionado, resetando seus valores quando o Clear é 1. Para essa ação, não há a necessidade de modificação do clock e/ou do RegWrite estar habilitado.

3.2.2 Leitura dos registradores: As entradas utilizadas e as saídas apresentadas para realizar a leitura do valor de um registrador são:

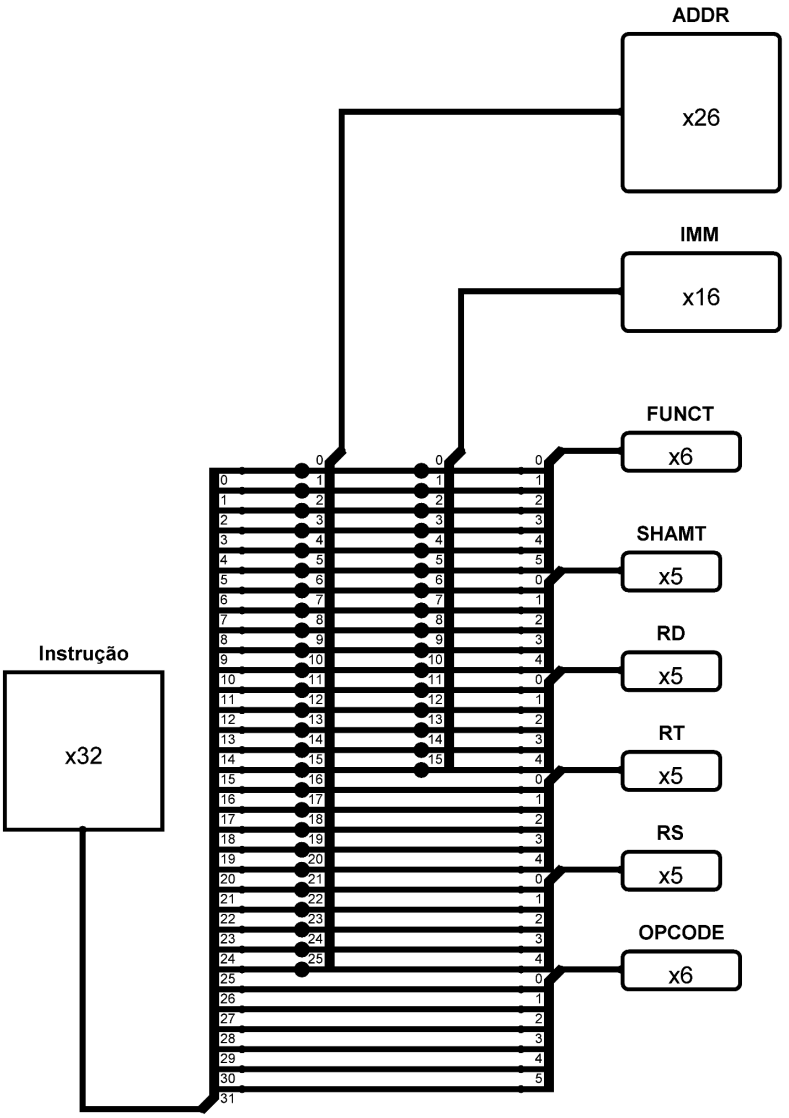
- LR1 e LR2 (5 bits): Realizam a seleção dos registradores cujos valores deseja-se realizar a leitura. Cada uma está ligada à porta de seleção de um MUX, selecionando o registrador desejado.
- LD1 e LD2 (32 bits): Saídas que apresentam o valor que está no registrador selecionado pelas entradas LR1 e LR2. Essas saídas estão ligadas na porta de saída de cada um dos MUXs.

  Portanto, ao selecionar os registradores nas entradas LR1 e LR2, seus respectivos valores serão apresentados em LD1 e LD2 por meio do MUX.

4. Decodificador de Instruções

O Decodificador de Instruções implementado é responsável por extrair os campos individuais de uma instrução no formato MIPS de 32 bits. Ele não interpreta os campos nem determina o tipo da instrução, apenas separa e disponibiliza cada parte de forma simultânea. Assim, cabe às demais unidades do processador utilizar os campos relevantes para cada tipo de operação. A divisão é feita conforme a estrutura no formato MIPS, que pode representar instruções do tipo R, I, ou J.

Figura 19: *Decodificador de Instruções*



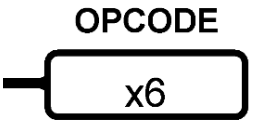
Fonte: Autoria Própria, 2025.

4.1 Campos extraídos pelo Decodificador de Instruções

A seguir, são descritos os campos extraídos de uma instrução MIPS de 32 bits pelo Decodificador de Instruções, com suas posições e funções específicas:

4.1.1 Operação: O campo opcode (bits de 31 a 26) indica o código da operação. É utilizado para identificar o tipo da instrução (R, I ou J) e determinar qual operação será realizada. Sua interpretação é feita pela Unidade de Controle, que usa esse campo para gerar os sinais de controle adequados ao tipo da instrução.

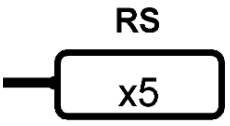
Figura 20: *Opcode*



Fonte: Autoria Própria, 2025.

4.1.2 RS: O campo rs (bits de 25 a 21) representa o registrador de origem, geralmente utilizado como primeiro operando em operações aritméticas/lógicas, ou como base em endereçamentos com deslocamento. Sua interpretação é feita pelo Banco de Registradores, que realiza a leitura do conteúdo do registrador indicado.

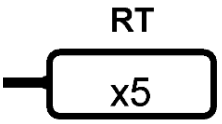
Figura 21: *RS*



Fonte: Autoria Própria, 2025.

4.1.3 RT: O campo rt (bits de 20 a 16) pode representar o segundo operando (em instruções do tipo R) ou o registrador de destino (em instruções do tipo I), dependendo do formato da instrução. Sua interpretação é feita pelo Banco de Registradores, sob controle da Unidade de Controle, que define o papel do campo rt conforme a operação.

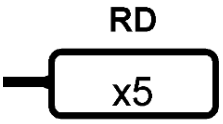
Figura 22: *RT*



Fonte: Autoria Própria, 2025.

4.1.4 RD: O campo rd (bits de 15 a 11) indica o registrador onde será armazenado o resultado da operação (somente em instruções do tipo R). Sua interpretação é feita pelo Banco de Registradores, que realiza a escrita do resultado no registrador especificado, quando instruído pela Unidade de Controle.

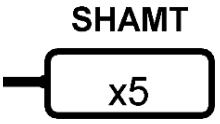
Figura 23: *RD*



Fonte: Autoria Própria, 2025.

4.1.5 Shamt: O campo shamt (bits de 10 a 6) especifica a quantidade de bits a ser deslocada em operações de shift (SLL e SRL). Sua interpretação é feita pela ULA, que recebe esse valor da Unidade de Controle apenas quando a instrução for de deslocamento.

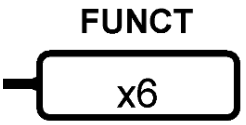
Figura 24: *Shamt*



Fonte: Autoria Própria, 2025.

4.1.6 Função: O campo funct (bits de 5 a 0) especifica a operação exata a ser executada em instruções do tipo R, funcionando como complemento ao campo opcode. Sua interpretação é feita pelo Controle da ULA, que interpreta esse campo para definir a operação lógica ou aritmética a ser realizada.

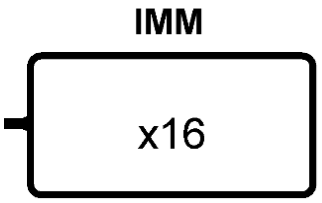
Figura 25: *Funct*



Fonte: Autoria Própria, 2025.

4.1.7 Imediato: O campo imm (bits de 15 a 0) contém um valor constante imediato, utilizado em instruções do tipo I. Sua interpretação é feita pelo Extensor de sinal, que converte o valor para 32 bits (com sinal), e a Unidade de Controle, que determina se esse campo será usado e de que forma.

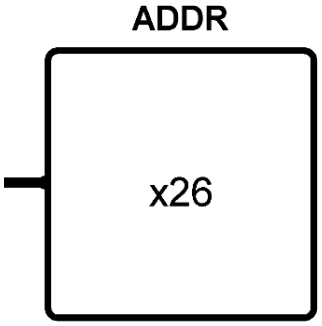
Figura 26: *Imm*



Fonte: Autoria Própria, 2025.

4.1.8 Endereço: O campo addr (bits de 25 a 0) representa o endereço de destino em instruções do tipo J (jump). Esse valor é ajustado para formar o novo endereço com o auxílio de <<2conc. Sua interpretação é feita pela Unidade de Controle.

Figura 27: *Addr*

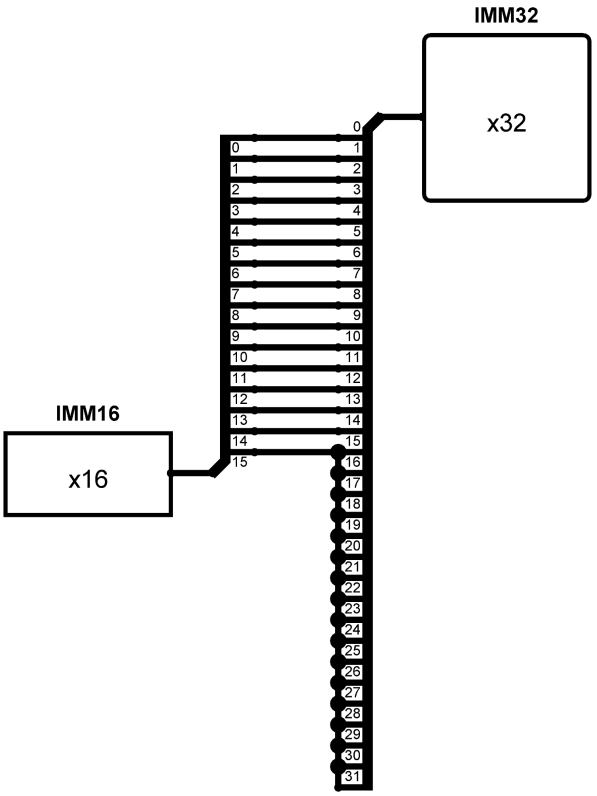


Fonte: Autoria Própria, 2025.

5. Extensor de Sinal

O Extensor de Sinal tem a função de converter um valor imediato de 16 bits para 32 bits. Essa conversão é necessária para que o número possa ser utilizado corretamente em operações realizadas pela ULA. O circuito realiza isso repetindo o bit mais significativo (bit de sinal) da entrada nos 16 bits mais à esquerda da saída, preservando o sinal original do valor. Essa lógica é implementada por meio de distribuidores conectados de forma adequada.

Figura 28: *Extensor de Sinal*

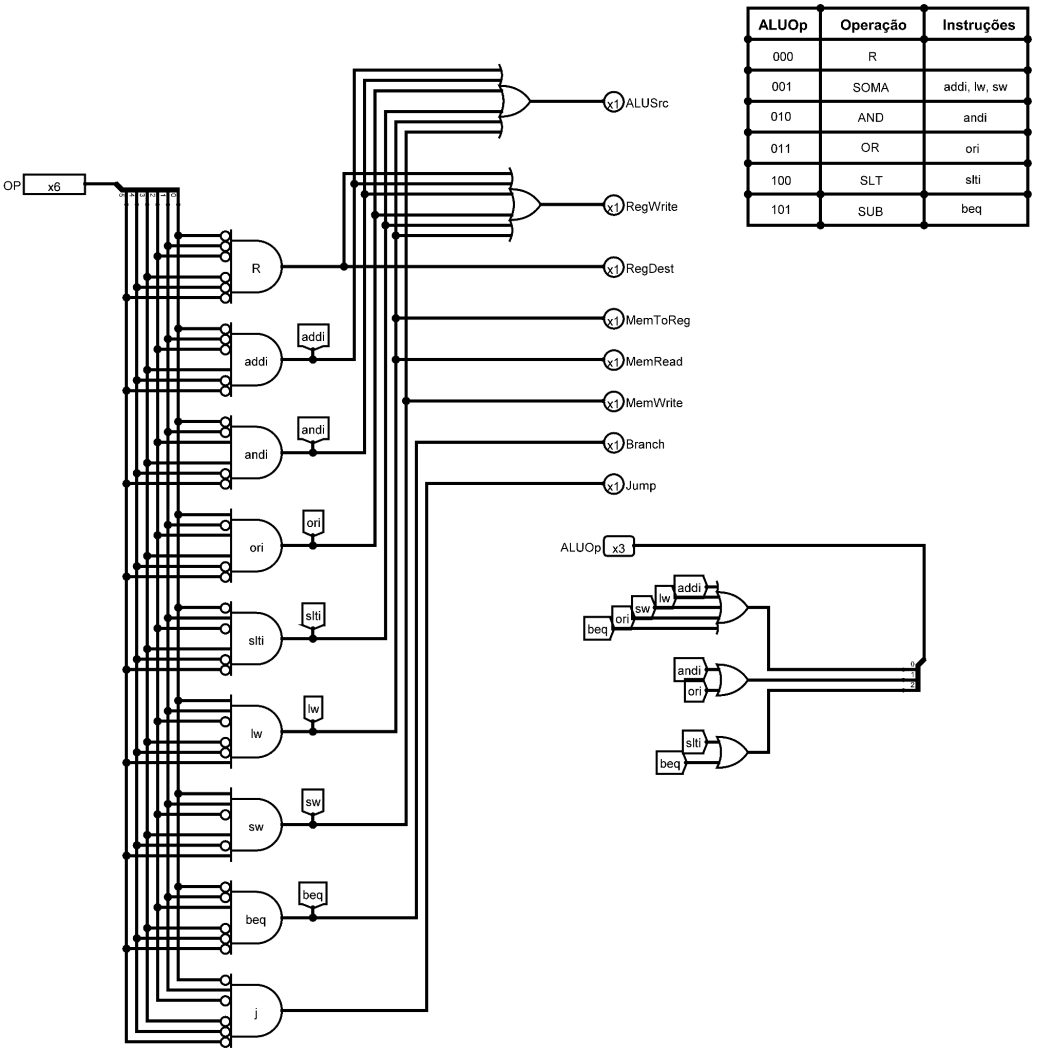


Fonte: Autoria Própria, 2025.

6. Unidade de Controle

A Unidade de Controle do Datapath Monociclo é responsável pela seleção das flags presentes ao longo de todo o circuito, podendo ativá-las ou desativá-las dependendo da operação que deseja ser realizada. Assim, definindo o caminho que os dados devem seguir ao longo do Datapath.

Figura 29: *Unidade de Controle*



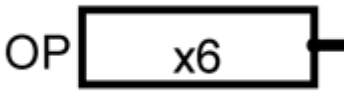
ALUOp	Operação	Instruções
000	R	
001	SOMA	addi, lw, sw
010	AND	andi
011	OR	ori
100	SLT	slti
101	SUB	beq

Fonte: Autoria Própria, 2025.

6.1 Componentes utilizados para a implementação da Unidade de Controle

6.1.1. Opcode: A Unidade de Controle possui como entrada única o Opcode, que será responsável por informar se a instrução é do tipo R, ou qual exatamente é a instrução, caso ela seja do tipo I ou J.

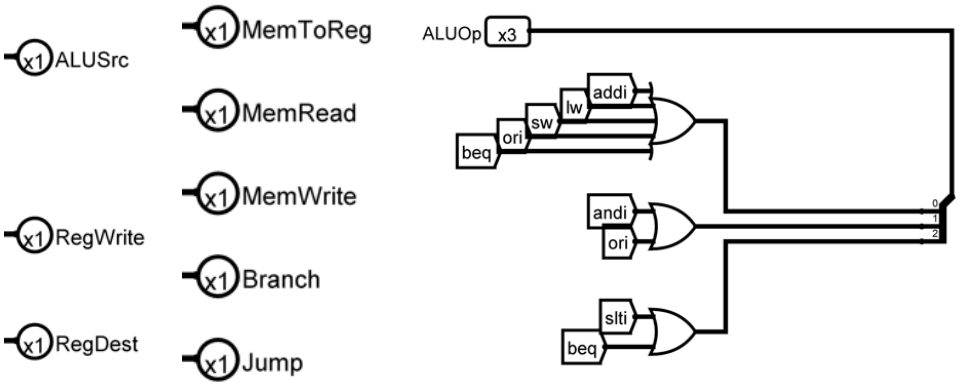
Figura 30: *Entrada Opcode*



Fonte: Autoria Própria, 2025.

6.1.2 Flags de controle: Há 9 saídas presentes na Unidade de Controle. Cada uma delas possui uma finalidade diferente e pode ser ativada, ou não, de acordo com o opcode informado.

Figura 31: *Flags de controle*



Fonte: Autoria Própria, 2025.

- **ALUSrc (1 bit):** Responsável por selecionar qual será a entrada B da ULA, podendo ser o LD2 (0) ou o imediato (1);

- **RegWrite (1 bit):** Responsável por habilitar (1) ou desabilitar (0) a escrita no Banco de Registradores;
- **RegDest (1 bit):** Possui a função de selecionar qual será o registrador de destino, ou seja, onde será salvo o resultado da operação realizada, podendo ser o rt (0) ou o rd (1);
- **MemToReg (1 bit):** Seleciona qual das saídas serão salvas no registrador de destino, podendo ser a saída da ULA (0) ou a saída da Memória de Dados (1);
- **MemRead (1 bit):** Responsável por habilitar (1) ou desabilitar (0) a leitura na Memória de dados;
- **MemWrite (1 bit):** Responsável por habilitar (1) ou desabilitar (0) a escrita na Memória de dados;
- **Branch (1 bit):** É habilitado quando a instrução é beq;
- **Jump (1 bit):** É habilitado quando a instrução é j;
- **ALUOp (3 bits):** É a saída da Unidade de Controle responsável por informar ao Controle da ULA qual operação deve ser realizada quando uma das instruções da Unidade de Controle é selecionada, como mostra a tabela a seguir:

Tabela 2: *Sinais de ALUOp*

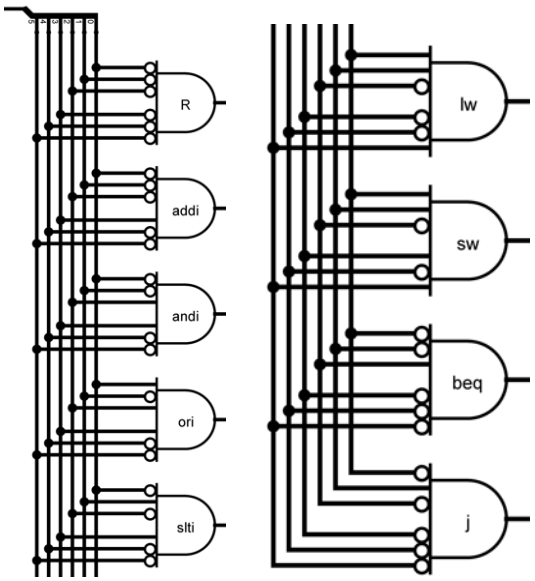
ALUOp	Operação	Instrução
000	R	tipo R
001	SOMA	addi, lw, sw
010	AND	andi
011	OR	ori
100	SLT	slti
101	SUB	beq

6.1.3. Identificadores de operações: Foram utilizadas 9 portas AND de 6 entradas, que recebem os bits do opcode por meio de um distribuidor e verifica qual operação a seguir está sendo selecionada:

Tabela 3: *Opcode e instruções*

Opcode	Instrução
000000	tipo R
001000	addi
001100	andi
001101	ori
001010	slti
100011	lw
101011	sw
000100	beq
000010	j

Figura 32: *Portas ANDs de identificação*



Fonte: Autoria Própria, 2025.

6.2 Funcionamento da Unidade de Controle

A Unidade de Controle possui a função de definir o caminho que os dados seguirão ao longo da sua execução de acordo com a instrução informada pelo opcode. Podendo habilitar ou desabilitar cada uma das flages citadas acima, além de atribuir um valor para ALUOp de acordo com a operação necessária. Dessa forma, temos:

- **R:** Serão ativados os sinais de RegWrite e RegDest, e ALUOp tem valor 000;
- **addi:** Serão ativados os sinais ALUSrc e RegWrite, e ALUOp assume valor 001;
- **andi:** Serão ativados os sinais ALUSrc e RegWrite, e ALUOp assume valor 010;
- **ori:** Serão ativados os sinais ALUSrc e RegWrite, e ALUOp assume valor 011;
- **slti:** Serão ativados os sinais ALUSrc e RegWrite, e ALUOp assume valor 100;
- **lw:** Serão ativados os sinais ALUSrc, RegWrite, MemToReg e MemRead, e ALUOp assume valor 001;
- **sw:** Serão ativados os sinais ALUSrc e MemWrite, e ALUOp assume valor 001;

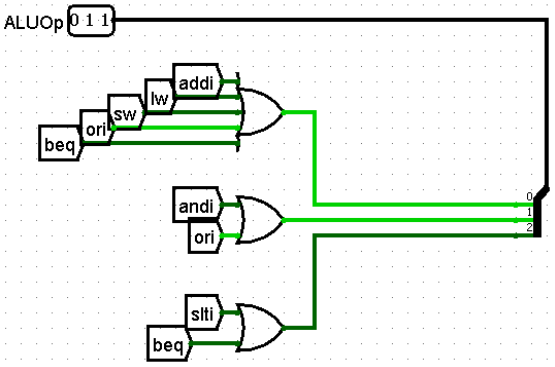
- **beq:** Será ativado o sinal Branch e ALUOp assume valor 101;
- **j:** Será ativado o sinal Jump.

Para realizar a ativação de cada uma das flags foi feita a ligação da saída de cada uma das ands somente nas flags que precisam ser habilitadas para aquela instrução. Como há 6 instruções que devem ativar as saídas ALUSrc e RegWrite, foram utilizadas duas portas OR de 6 entradas que foram ligadas a essas saídas.

Já no ALUOp foram utilizadas três portas OR, que se ligam em um distribuidor de 3 bits onde cada um representa um bit da saída. Dessa forma, de acordo com a instrução escolhida no opcode ele irá ativar os bits específicos que representam a operação em ALUOp.

Por exemplo, quando a operação ori está selecionada, a saída da AND que verifica se o opcode é um ori será 1 e o ALUOp deve assumir o valor 011. Para isso, a saída com valor 1 será ligada no distribuidor de ALUOp na porta OR que representa o bit 0 e também na que representa o bit 1, assim toda vez que a ori for selecionada os dois bits menos significativos de ALUOp serão ativados.

Figura 33: *Exemplo ori em ALUOp*

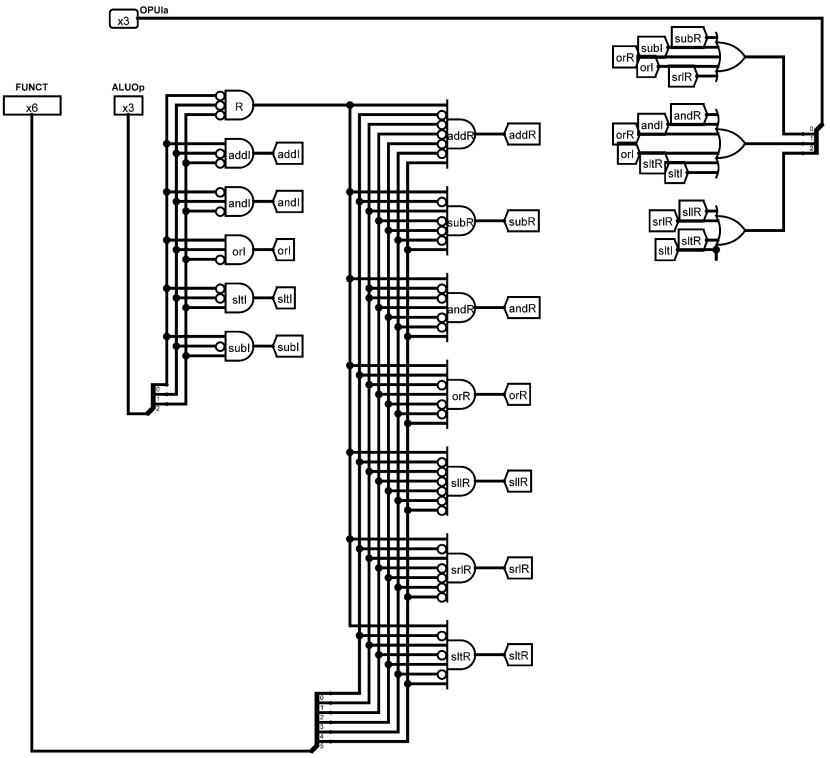


Fonte: Autoria Própria, 2025.

7. Unidade de Controle da ULA

A Unidade de Controle da ULA é responsável por enviar para a entrada Op da ULA qual operação deseja ser realizada, diferenciando as instruções do tipo R ou I/J por meio da entrada ALUOp e utilizando a entrada Funct quando necessário.

Figura 34: *Unidade de Controle da ULA*



Fonte: Autoria Própria, 2025.

7.1 Componentes utilizados para a implementação da Unidade de Controle da ULA

7.1.1. Entradas FUNCT e ALUOp: A Unidade de Controle da ULA recebe duas entradas:

- **ALUOp (3 bits):**Vem diretamente da Unidade de Controle principal e informa se a instrução que deve ser executada é do tipo R (ALUOp = 000), ou de outro tipo, informando qual a operação a ULA deve realizar, seguindo a ***Tabela 2***.

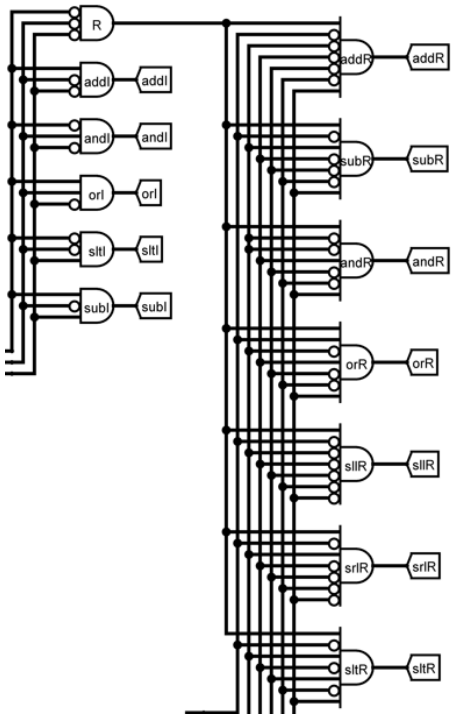
- **FUNCT (6 bits):** Vem diretamente do Decodificador e é utilizada pela Unidade de Controle da ULA caso a instrução que está sendo executada seja do tipo R (Opcode = 0 e ALUOp = 000). Dessa forma, por conta do opcode receber o valor 0 para todas as instruções do tipo R é necessário usar o Funct para definir qual operação a ULA deverá realizar.

Figura 35: *Entradas Funct e ALUOp*

Fonte: Autoria Própria, 2025.

7.1.2. Portas ANDs para seleção de operação: Há 13 portas AND na Unidade de Controle da ULA, onde 6 delas possuem 3 entradas, que recebem os 3 bits de ALUOp e verificam se a instrução é do tipo R, com ALUOp 000, ou se é alguma das outras operações possíveis de ALUOp (add, and, or, slt ou sub). As outras 7 portas ANDs possuem 7 entradas cada, onde recebem os 6 bits da entrada Funct, por meio de um distribuidor, e também a saída da AND R, responsável por identificar se a instrução é do tipo R. Dessa maneira, essas ANDs possuem a responsabilidade de identificar qual das 7 operações do tipo R presentes na ULA deve ser selecionada (add, sub, and, or, sll, srl, slt).

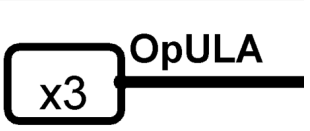
Figura 36: *Portas AND para seleção*



Fonte: Autoria Própria, 2025.

7.1.3. Saída OpULA: A Unidade de Controle da ULA possui uma saída de 3 bits que será conectada diretamente a entrada Op da ULA, portanto ela possui a funcionalidade de selecionar a operação que a ULA irá realizar.

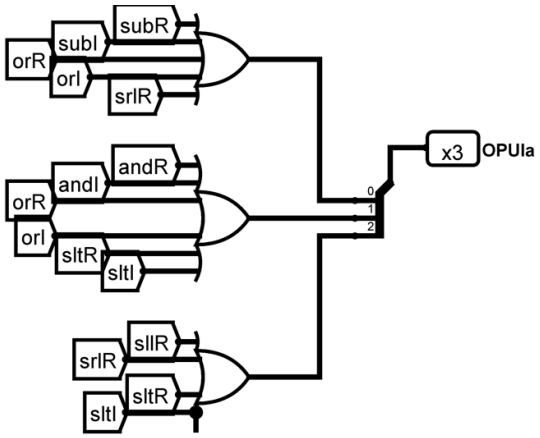
Figura 37: *Saída OpULA*



Fonte: Autoria Própria, 2025.

7.1.4. Portas OR para saída: Foram utilizadas 3 portas OR, que se ligam às entradas de um distribuidor que representa os três bits da saída OpULA. Cada uma das portas recebe o valor de saída das ANDs de seleção, explicadas no tópico 7.1.2, de acordo com os bits que precisam ser ativados para a operação desejada na ULA.

Figura 38: *Portas OR para gerenciamento da saída*



Fonte: Autoria Própria, 2025.

7.2 Funcionamento da Unidade de Controle da ULA

A Unidade de Controle da ULA possui a função de definir qual será a operação realizada pela ULA, utilizando as entradas ALUOp e Funct como parâmetros para essa definição.

Dessa forma, é necessário primeiramente, verificar qual o valor da entrada ALUOp. Caso o valor represente as operações add, and, or, slt ou sub a saída da porta AND responsável por identificar a operação será ativada, e irá, por meio das portas OR, ativar os bits da saída OpULA necessários para a realização da operação na ULA, seguindo a ***Tabela 1***.

Portanto, caso ALUOp tenha o valor 011, que segundo a ***Tabela 2*** representa a operação or, a AND responsável por identificar se ALUOp possui valor 011 assume saída 1 e é ligada às portas OR dos bits 0 e 1 da saída OpULA, gerando como saída 011, o valor da operação or na ***Tabela 1***.

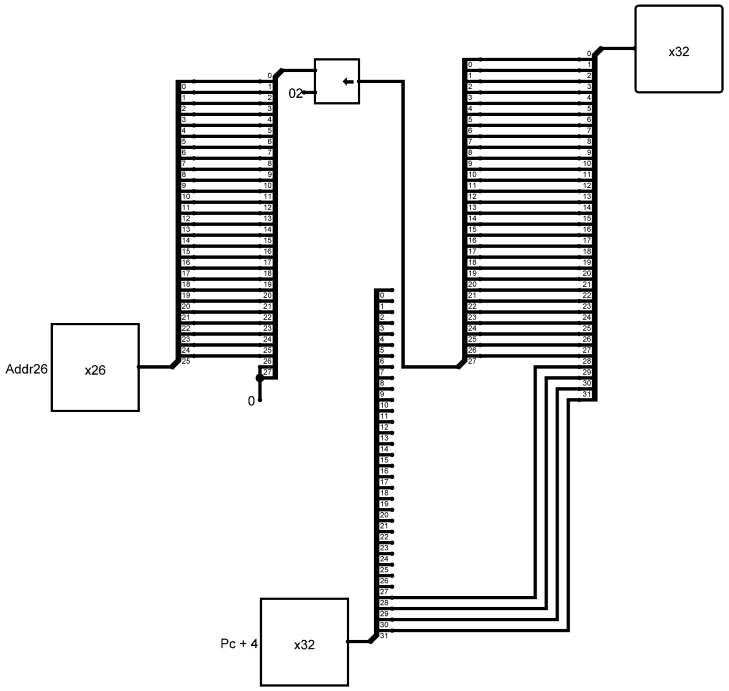
Porém, caso o valor de ALUOp for 000, a AND responsável por identificar as instruções do tipo R recebe a saída 1, e é conectada nas entradas de todas as 7 ANDs do tipo R. Portanto, essas ANDs somente terão saída 1 quando ALUOp for 000, e o valor do Funct for correspondente às suas respectivas operações, segundo o cartão de referência do Mips.

Dessa forma, as saídas das portas ANDs são conectadas às portas OR de OpULA, nos respectivos bits que necessitam ser ativados.

8. <<2 conc

O circuito <<2conc é responsável por executar a instrução Jump, do tipo J. Ele tem a função de realizar dois deslocamentos para a esquerda da entrada addr26 e acrescentar os 4 bits mais significativos de Pc + 4 aos bits mais significativos do endereço de saída.

Figura 39: *Circuito de <<2 conc*



Fonte: Autoria Própria, 2025.

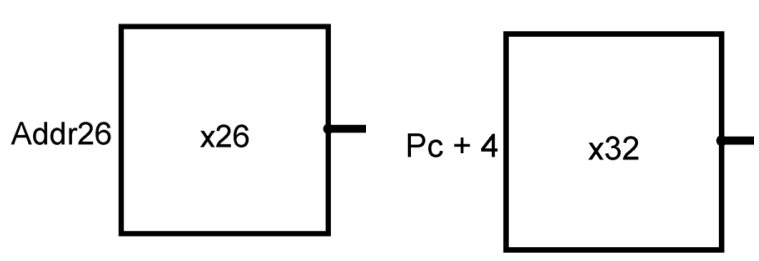
8.1 Componentes utilizados na implementação de <<2 conc

8.1.1. Entradas de endereços: O circuito recebe duas entradas:

- **Addr26 (26 bits):** Vem diretamente do Decodificador, que representa o endereço que PC irá receber, após a execução do Jump;

- **Pc + 4 (32 bits):** Representa o endereço da próxima instrução que será buscada na memória.

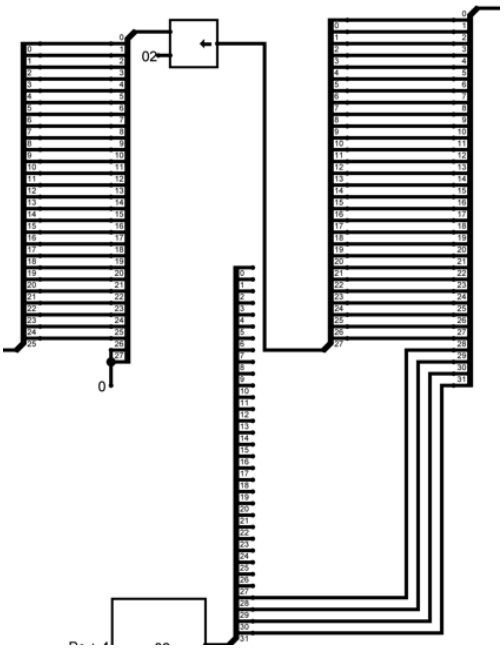
Figura 40: *Entradas addr26 e Pc + 4*



Fonte: Autoria Própria, 2025.

8.1.2. Distribuidores: Foram utilizados 5 distribuidores prontos do Logisim, com quantidades de entradas diferentes, sendo elas: 1 distribuidor de 26 bits, 2 distribuidores de 28 bits e outros dois de 32 bits.

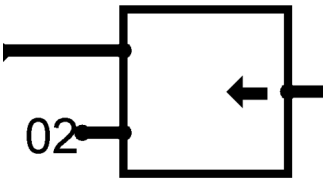
Figura 41: *Distribuidores*



Fonte: Autoria Própria, 2025.

8.1.3. Deslocador de bits: Foi utilizado um deslocador de bits, configurado para realizar deslocamentos para a esquerda. Ele possui duas entradas: uma de 28 bits, que recebe o valor a ser deslocado, e uma de 5 bits para indicar a quantidade de deslocamentos que serão feitos.

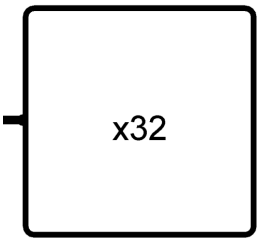
Figura 42: *Deslocador*



Fonte: Autoria Própria, 2025.

8.1.4. Saída de endereço: O circuito possui uma saída de 32 bits que representa o endereço da próxima instrução que o pc receberá após a execução do Jump.

Figura 43: *Saída de endereço*



Fonte: Autoria Própria, 2025.

8.2 Funcionamento do <<2 conc

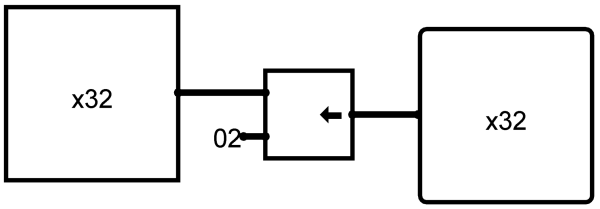
A entrada addr26 é ligada ao distribuidor de 26 bits, que se liga diretamente ao de 28, com os dois bits mais significativos recebendo a constante 0. Após isso, é realizado dois deslocamentos de 1 bit para a esquerda utilizando o deslocador pronto do simulador Logisim, que recebe o fio de 28 bits e a constante 2, referente a quantidade de deslocamentos.

Ao final, a saída do deslocador de bits é ligada a um distribuidor de 28 bits que se liga diretamente a outro 32 bits, sendo os 28 bits os menos significativos, e os outros vem do distribuidor de 32 bits ligado à entrada PC + 4, pegando somente os 4 bits mais significativos.

9. <<2

O circuito <<2 possui a função de realizar dois deslocamentos para a esquerda. Ele possui uma entrada de 32 bits, que se liga a uma entrada de 32 bits de um deslocador pronto do simulador Logisim, que por sua vez, também possui uma entrada de 5 bits que recebe a constante de valor 2, indicando a quantidade de bits deslocados. A saída do deslocador se liga a um pino de saída de 32 bits que recebe o valor de entrada já deslocado.

Figura 44: *Circuito de <<2*



Fonte: Autoria Própria, 2025.

10. Datapath Monociclo

O Datapath (ou Caminho de Dados) monociclo é uma arquitetura onde cada instrução é executada somente em um ciclo (pulso) de clock.

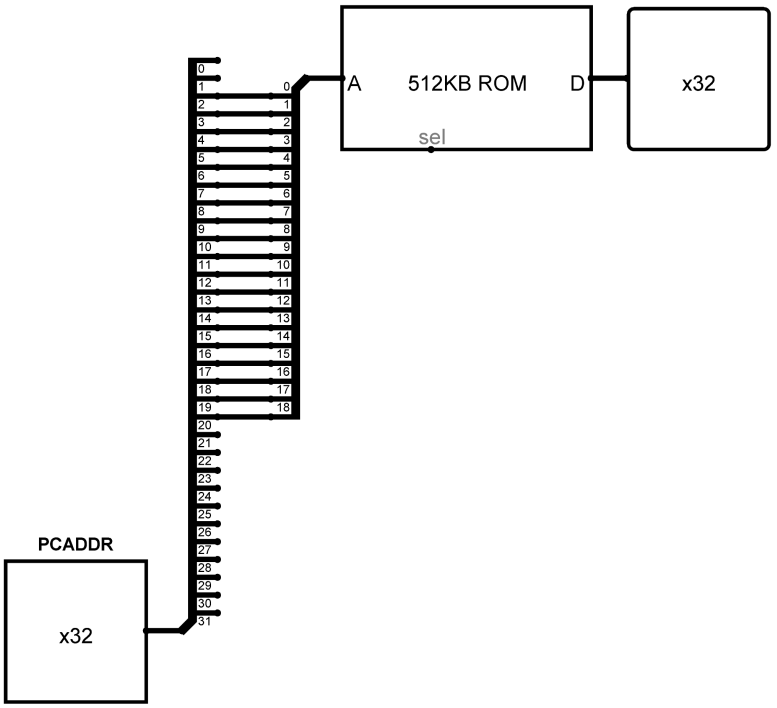
10.1 Componentes utilizados para a implementação do Datapath monociclo

10.1.1. Memória de Instruções: Foi utilizado um circuito de Memória de Instruções disponibilizado no material de apoio. A memória de instruções é o local onde são armazenados os conjuntos de instruções a serem executados pelo datapath. Dessa maneira, a medida que o registrador PC avança nos endereços a memória de instruções fornece a instrução correspondente.

Figura 45: *Memória de Instruções no Datapath*

Fonte: Autoria Própria, 2025.

Figura 46: *Circuito da Memória de Instruções*



Fonte: Autoria Própria, 2025.

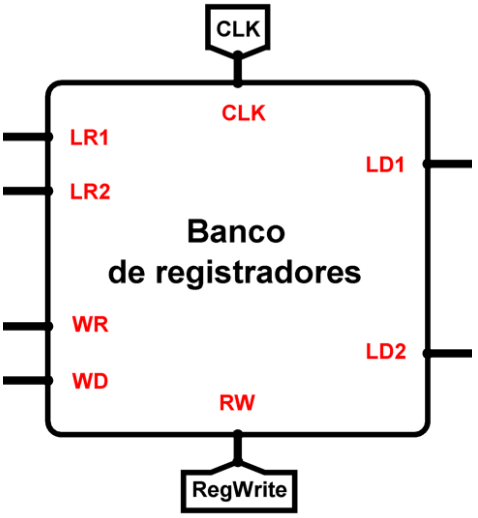
10.1.2. Decodificador: Responsável por dividir as partes da instrução vinda da Memória de Instruções e distribuí-la para todas as entradas necessárias ao longo do Datapath.

Figura 47: *Decodificador no Datapath*

Fonte: Autoria Própria, 2025.

10.1.3. Banco de Registradores: Circuito que contém os 32 registradores da arquitetura MIPS com a funcionalidade de leitura e escrita de dados.

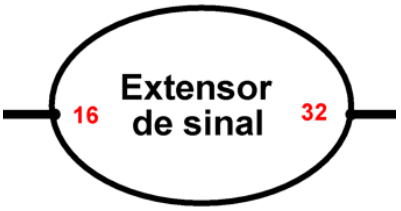
Figura 48: *Banco de Registradores no Datapath*



Fonte: Autoria Própria, 2025.

10.1.4. Extensor de sinais: Possui a função de converter o imediato de 16 bits em 32 bits.

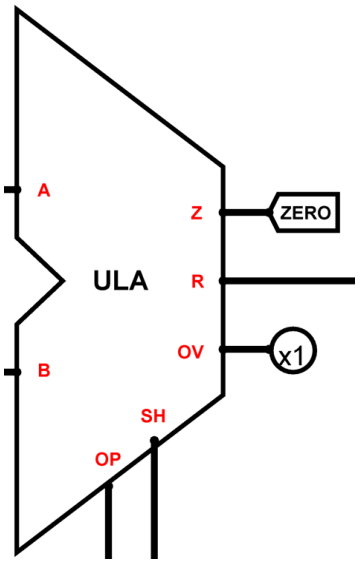
Figura 49: *Extensor de sinais no Datapath*



Fonte: Autoria Própria, 2025.

10.1.5. Unidade Lógica e Aritmética (ULA): O circuito possui 7 operações lógicas e aritméticas. Recebe como entrada as saídas do banco de registradores ou o imediato vindo do Extensor de Sinais para realizar os cálculos, e apresenta três saídas: Overflow, resultado e zero.

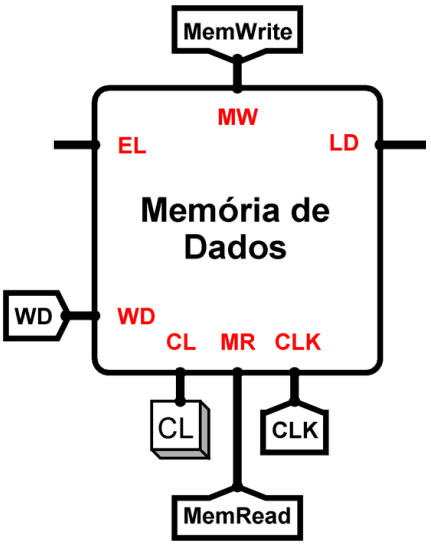
Figura 50: *ULA no Datapath*



Fonte: Autoria Própria, 2025.

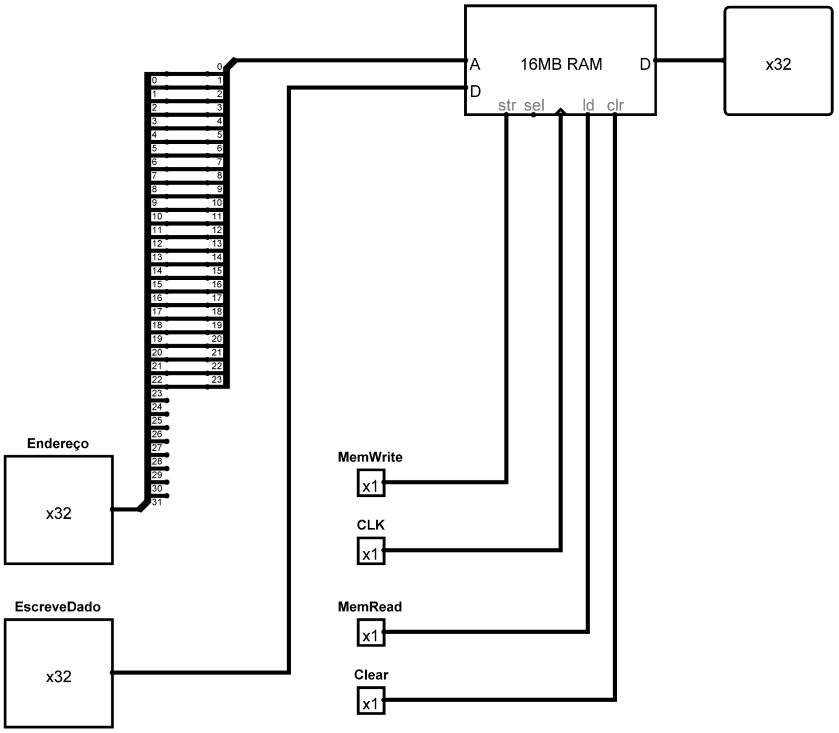
10.1.6. Memória de dados: Foi utilizado um circuito de Memória de Dados disponibilizado no material de apoio. Ela possui a função de realizar a leitura e escrita dos dados, recebendo como entrada o resultado da ULA e a saída LD2 do Banco de registradores.

Figura 51: *Memória de dados no Datapath*



Fonte: Autoria Própria, 2025.

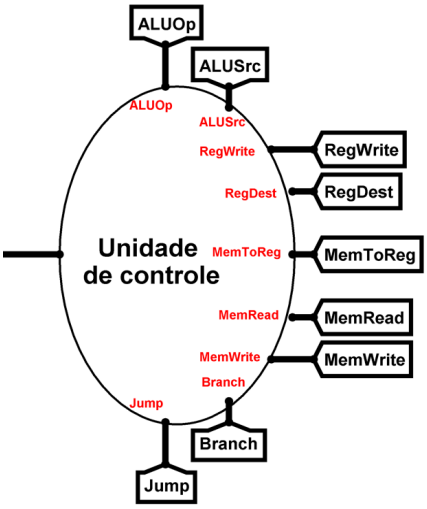
Figura 52: *Circuito da memória de dados*



Fonte: Autoria Própria, 2025.

10.1.7. Unidade de controle: Esse circuito possui a função de habilitar e desabilitar as flags presentes ao longo de todo o circuito, além de gerar o valor de ALUOp. Ele recebe como entrada o opcode do Decodificador e gera 9 saídas.

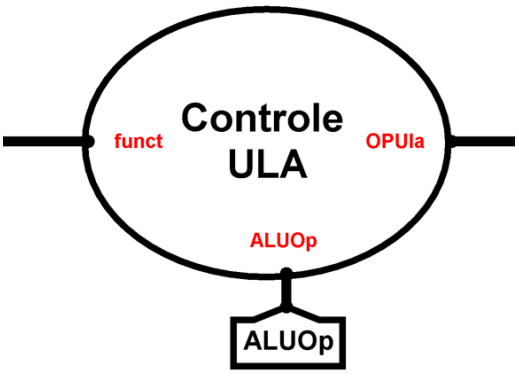
Figura 53: *Unidade de controle no Datapath*



Fonte: Autoria Própria, 2025.

10.1.8. Unidade de controle da ULA: O circuito possui a função de definir qual a operação que a ULA deve realizar, de acordo com a instrução a ser executada. Recebendo como entrada o Funct, vindo do Decodificador e o ALUOp vindo da Unidade de Controle, gerando como saída a operação que deverá ser feita na ULA (OpULA).

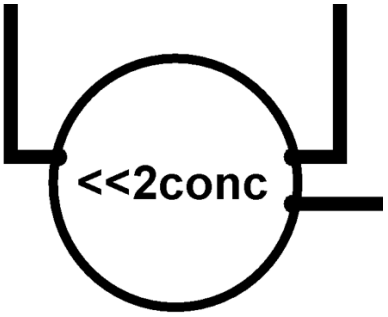
Figura 54: *Unidade de controle da ULA no Datapath*



Fonte: Autoria Própria, 2025.

10.1.9. Concatenador: O circuito que recebe o nome “<<2conc” possui a função calcular os endereços para a instrução j (jump). Ele recebe como entrada o endereço (addr26) vindo do Decodificador, realiza os dois deslocamentos para a esquerda e concatena a resposta com os 4 bits mais significativos de Pc + 4.

Figura 55: *Concatenador no Datapath*

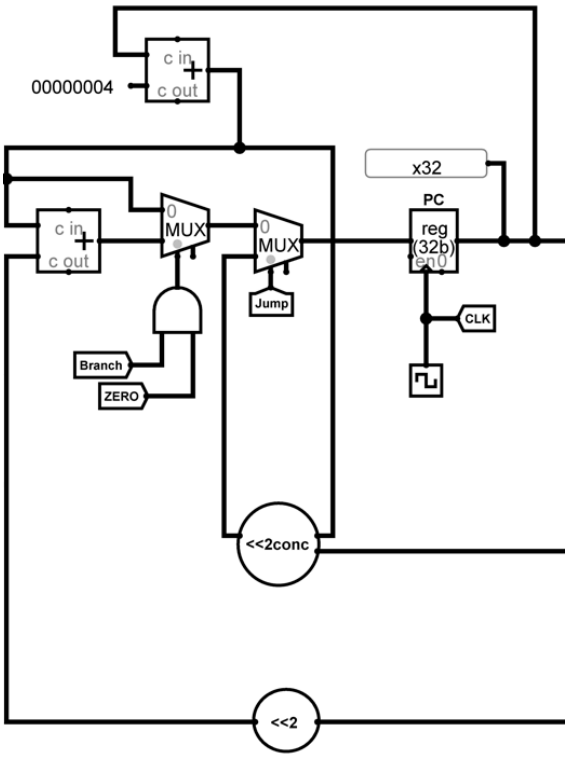


Fonte: Autoria Própria, 2025.

10.1.10. Multiplexadores (MUX): Foram utilizados 5 MUX de duas entradas e 1 bit de seleção, que recebem as flags de saída da Unidade de Controle e são responsáveis por definir o caminho que os dados irão seguir durante a execução de cada instrução.

10.1.11. Somadores e deslocador: O Datapath Monociclo possui um circuito de deslocamento chamado “<<2” que recebe o imediato do Decodificador e o desloca duas vezes para a esquerda. E dois somadores: um realiza a soma do endereço atual + 4, atribuindo o valor do próximo endereço a Pc. Já o outro somador recebe o valor de de Pc + 4 e o imediato deslocado (a saída do circuito “<<2”).

Figura 56: *Somadores e deslocador no Datapath*



Fonte: Autoria Própria, 2025.

11. Execução

O algoritmo de alto nível correspondente ao Datapath Monociclo foi implementado em linguagem C. O código é dividido em blocos que simulam instruções comuns no conjunto MIPS. Inicialmente são declaradas e inicializadas as variáveis `a` e `b` com os valores 2 e 7, respectivamente. Também são declaradas as variáveis auxiliares `result`, `soma` e `i`, sendo `i` iniciado com `a + 1`. O primeiro bloco representa uma estrutura condicional utilizando a instrução `beq`. Nele o código verifica se `a` é igual a `b`. Caso verdadeiro, o resultado (`result`) é a soma das duas variáveis. Caso contrário, `result` armazena a subtração das mesmas. O segundo bloco também representa uma estrutura condicional, mas simula a instrução `slt`, que avalia se `a` é menor que `b`. Se a condição for verdadeira, `result` recebe o dobro de `a` (`a + a`). Caso contrário, result recebe o dobro de `b` (`b + b`). No terceiro bloco, temos dois laços de repetição não alinhados, que simulam um controle de fluxo

baseado em `slt`. O `while` acumula em soma todos os valores entre `a` e `b`, enquanto `i` for menor que `b`. Em seguida, um `for` percorre o intervalo inverso, subtraindo de `soma` os mesmos valores, enquanto `i` for maior que `a`. No último bloco, são utilizadas operações de memória que simulam as instruções `sw` e `lw`. Um vetor `mem` com 5 posições é criado, e o valor de `a` é armazenado na posição 2. Em seguida, `b` recebe novamente o valor salvo na memória (`mem[2]`).

Código em Linguagem de Alto Nível (C)

```c
C/C++
#include <stdio.h>

int main() {
    int a = 2;
    int b = 7;
    int result = 0;
    int soma = 0;
    int i = a + 1;

    // Estrutura Condicional (beq) = 10 pontos
    if (a == b) {
        result = a + b;

    } else {
        result = a - b;
    }

    // Estrutura Condicional (slt) = 10 pontos
    if (a < b) {
        result = a + a;
    } else {
        result = b + b;
    }

    // Laço de Repetição Não-Aninhado (while e for com slt) = 24 pontos
    while (i < b) {
        soma = soma + i;
        i = i + 1;
    }

    for (i; i > a; i--) {
        soma = soma - i;
    }

    int mem[5] = {0};
    mem[2] = a;
    b = mem[2];

    return 0;
}
```

Tradução em Assembly

Unset

```
# R = add, sub, slt (12 pontos)
# Aritméticas com Imediato = addi (5 pontos)
# Desvios Condicionais = beq (15 pontos)
# Desvios Incondicionais = J (10 pontos)
# Operações de Memória = lw, sw (30 pontos)
# TOTAL = 12 + 5 + 15 + 10 + 30 = 72 pontos

Main:
        # Inicializa as variaveis a, b, result, soma, i
        addi $s0, $0, 2               # $s0 = a = 2
        addi $s1, $0, 7               # $s1 = b = 7
        add $s2, $0, $0               # $s2 = result = 0
        add $s3, $0, $0               # $s3 = soma = 0
        addi $s4, $s0, 1              # $s4 = i = a + 1

        # Desvios Condicionais (beq) + Instrucoes R (add, sub, slt)
        IF1:
                beq $s0, $s1, ELSE1   # se a == b ? ELSE1 : PROX_LINHA
                sub $s2, $s0, $s1     # result = a - b
                j IF2
        ELSE1:
                add $s2, $s0, $s1     # result = a + b
                j MEMORIA

        IF2:
                slt $t0, $s0, $s1     # $t0 = (a < b) ? 1 : 0
                beq $t0, $0, ELSE2    # se $t0 == 0 ? IF3 : PROX_LINHA
                add $s2, $s0, $s0     # result = a + a
                j WHILE
        ELSE2:
                add $s2, $s1, $s1     # result = b + b
                j MEMORIA

        # Desvios Incondicionais (j) + Aritméticas com Imediato (addi)
        WHILE:
                slt $t0, $s4, $s1     # $t0 = i < b
                beq $t0, $0, FOR      # se i > b : FOR
                add $s3, $s3, $s4     # soma = soma + i
                addi $s4, $s4, 1      # i = i + 1
                j WHILE               # volta para WHILE

        FOR:
                slt $t0, $s0, $s4
                beq $t0, $0, MEMORIA
                sub $s3, $s3, $s4
                addi $s4, $s4, -1
                j FOR
        # Operações de Memória (lw, sw)
        MEMORIA:
                addi $s5, $0, 20      # $s5 = -4
                sw $s0, 0($s5)        # mem[2] = a
                lw $s1, 0($s5)        # b = mem[2]
```

Código Hexadecimal Correspondente na Linguagem de Máquina

```
Unset
20100002
20110007
00009020
00009820
22140001
12110002
02119022
0810000a
02119020
0810001a
0211402a
11000002
02109020
08100010
02319020
0810001a
0291402a
11000003
02749820
22940001
08100010
0214402a
11000003
02749822
2294ffff
08100015
20150014
aeb00000
8eb10000
```

11. Pontuação Datapath de acordo com o código implementado

Tabela 4: *Pontuação*

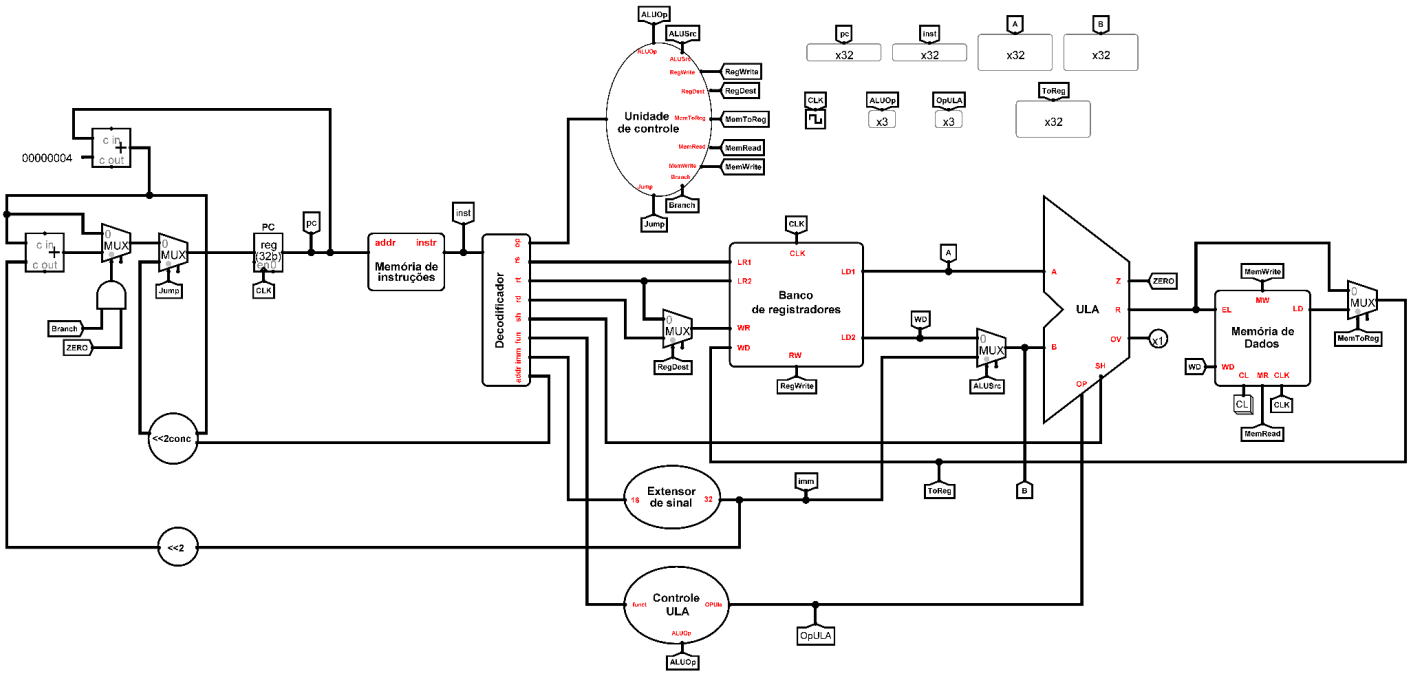
Tipo		Pts.	Máx.	Máx.	Quant.	Pts. Totais	Soma Final
Instruções	Instruções R	4	40	70	3	12	72
	Instruções aritméticas com imediato	5	25		1	5	
	Desvios Condicionais	15	30		1	15	
	Desvios Incondicionais	10	10		1	10	
	Operações de Memória	15	45		2	30	
Código	Estrutura Condicional	10	20	30	2	20	44
	Laços de repetição não aninhados	12	24		2	24	
	Laços de repetição aninhados	15	15		0	0	

12. Conclusão

A implementação do Datapath Monociclo evidencia a complexidade envolvida na construção de um caminho de dados funcional, onde cada componente desempenha um papel essencial para a execução das instruções. Durante o desenvolvimento do projeto, foi fundamental compreender e abstrair corretamente cada conceito presente no Datapath, visto que a integração organizada e padronizada dos componentes é essencial para um trabalho coletivo e eficiente. A documentação e padronização permitiram uma construção clara, funcional e objetiva do sistema, viabilizando a execução de códigos traduzidos de linguagens de alto nível, como C, em linguagem de máquina.

Dessa forma, o trabalho reforça que o datapath pode ser dividido em seções independentes, que, combinadas, formam uma arquitetura completa e funcional, como demonstrado na organização clara dos componentes na ***Figura 57***.

Figura 57: *Datapath Monociclo*



Fonte: Autoria Própria, 2025.

13. Referências

[1] PATTERSON, David A. Hennessy, John L. Organização e Projeto de Computadores. Disponível em: Minha Biblioteca, (5a. edição). Grupo GEN, 2017