

Arquitetura de Computadores

João Victor Briganti de Oliveira¹, Luiz Gustavo Takeda²

Universidade Tecnológica Federal do Paraná – UTFPR

COCIC – Coordenação do Curso de Bacharelado em Ciência da Computação

Campo Mourão, Paraná, Brasil

¹joliveira.2022@alunos.utfpr.edu.br

²luiztakeda@alunos.utfpr.edu.br

Resumo

Especificações e escolhas dos meios para a implementação da arquitetura MIPS(Microprocessor without Interlocked Pipeline Stages) utilizando o simulador Logisim.

1. Introdução

O presente trabalho serve como documentação para a arquitetura criada para a turma de Arquitetura e Organização de Computadores de 2023. A arquitetura sendo construída tem 32-bits, e segue o modelo da arquitetura MIPS(Microprocessor without Interlocked Pipeline Stages) [1].

Para a montagem da arquitetura foi utilizado o simulador Logisim [2], uma ferramenta usada para simulação de circuitos digitais.

A sessão 2 trás a implementação do banco de registradores e seus detalhes, a sessão 3 trás a implementação da ULA(Unidade Lógica Aritmética), a sessão 4 mostra detalhes da implementação da memória de instruções, a sessão 5 a implementação da memória de dados, a sessão 6 o decodificador de instruções, a sessão 7 o decodificador de instruções e por fim a sessão 8 o controle principal.

2. Banco de Registradores

Para os registradores o modelo do MIPS será usado, onde temos 32 registradores que são divididos entre salvos, temporários e especiais. A tabela [1] abaixo exemplifica a divisão e nomenclatura desses registradores:

Tabela 1: Lista e Descrição dos Registradores

Lista de Registradores		
Registradores	Nome	Descrição
\$0	\$zero	Sempre zero
\$1	\$at	Reservado para o montador
\$2-\$3	\$v0,\$v1	Valores de retorno
\$4-\$7	\$a0-\$a3	Argumento(s) da função
\$8-\$15 e \$24-\$25	\$t0-\$t7 e \$t8-\$t9	Registradores Temporários
\$16-\$23	\$s0-\$s7	Registradores Salvos
\$26-\$27	\$k0-\$k1	Reservado para o Kernel
\$28	\$gp	Ponteiro Global
\$29	\$sp	Ponteiro de Pilha

\$30	\$fp	Ponteiro de Frame
\$31	\$ra	Endereço de Retorno

O circuito do banco de registradores possui cinco entradas e duas saídas. Sendo elas:

- **LR1**: Load Register 1, entrada de 5-bits que especifica o primeiro registrador a ser lido.
- **LR2**: Load Register 2, entrada de 5-bits que especifica o segundo registrador a ser lido.
- **WR**: Write Register, entrada de 5-bits que especifica em qual registrador será escrito o WD
- **WD**: Write Data, entrada de 32-bits que traz os valores que serão escritos em um dos registradores.
- **EW**: Enable Write, bit que habilita/desabilita escrita de registrador. Em 0 desabilita e em 1 habilita.

Na implementação desse banco foram usados 31 registradores, o registrador \$zero por não permitir escrita e ser sempre o valor 0, foi “desligado” diretamente. Essa escolha de usar um registrador a menos se dá pela natureza do registrador \$zero, e pelo fato de que montando o circuito dessa maneira se torna possível diminuir a complexidade do mesmo economizando em espaço e transistores. Para as demais partes do circuito temos a utilização de dois multiplexadores que são usados para o envio de dados nas portas LD1 e LD2 e um demultiplexador para habilitar a escrita de dados que é enviado na porta WD. Abaixo a figura [1] que ilustra com suas entradas:

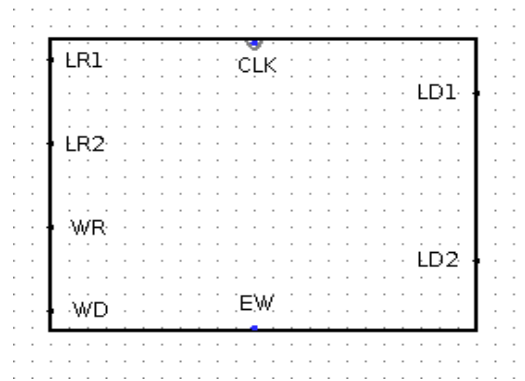


Figura 1: Banco de Registradores

Do circuito interno segue a figura[2]:

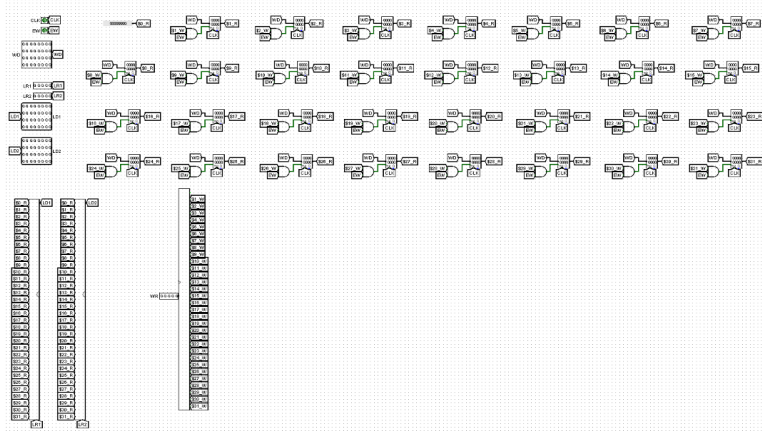


Figura 2: Circuito interno Banco de Registradores

3. Unidade Lógica Aritmética

3.1 Circuito

Para realizar as operações temos as seguintes entradas, **Operação** 4bits responsável por selecionar qual operação vai ser utilizada, **Shamt** 5bits utilizado para ditar quantas vezes será deslocado nos comandos **SLL** e **SRL**, **A** 32bits e **B** 32bits são as entradas para a operação.

E a saída, **Zero** 1bit, que indica quando o resultado é 0, **Result** 32bits é o resultado da operação e **Overflow** 1bit indica quando há um estouro de memória nas operações de **ADD** e **SUB**.

A figura[3] abaixo, ilustra a disposição das entradas e saídas:

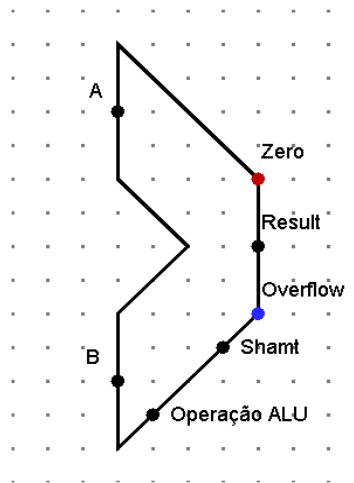


Figura 3: ULA

Do circuito interno segue a figura[4]:

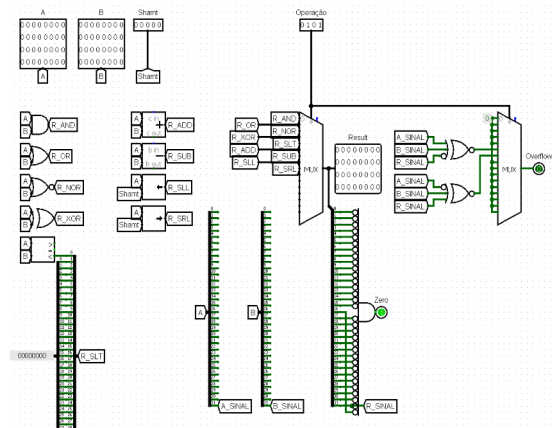


Figura 4: Circuito interno ULA

Funcionamento do circuito:

Sobre a entrada da seleção de operação é utilizado um MUX para multiplexar todos os resultados das operações em uma única saída, assim de acordo com o valor de **Operação**, o MUX direciona o valor de resultado de uma operação específica para o **Result**.

Para a saída **Overflow**, é utilizado um MUX, a fim de tratar o valor de overflow dependendo da operação que está sendo realizada, neste caso, apenas a operação de ADD e SUB, estão sendo tratadas, as demais operações o **Overflow**, é constante 0. Para o caso de ADD, quando os sinais são iguais entre A e B, e o sinal do valor resultando é o inverso, tem-se o overflow. Na operação SUB, quando os valores dos sinais A e B são distintos, e o sinal do resultado for igual ao sinal de B, temos overflow.

A saída **Zero**, é verificado se todos os valores de **Result**, são iguais a 0, utilizando uma porta AND com suas entradas negadas, quando caso algum valor seja 1, sua saída será 0.

3.2 Operações

As operações implementadas na ULA, são as seguintes:

AND

Utiliza a porta lógica AND: $R = A \& B$.

Operação: 0x0

Shamt: Indiferente

OR

Utiliza a porta lógica OR: $R = A | B$.

Operação: 0x1

Shamt: Indiferente

NOR

Utiliza a porta lógica NOR: $R = \sim(A \mid B)$.

Operação: 0x2

Shamt: Indiferente

XOR

Utiliza a porta lógica XOR: $R = A \wedge B$.

Operação: 0x3

Shamt: Indiferente

SLT

Realiza a comparação: $R = A < B$.

Operação: 0x4

Shamt: Indiferente

ADD

Realiza a soma: $R = A + B$.

Operação: 0x5

Shamt: Indiferente

SUB

Realiza a subtração: $R = A - B$.

Operação: 0x6

Shamt: Indiferente

SLL

Realiza o deslocamento para esquerda: $R = A \ll \text{Shamt}$.

Operação: 0x7

Shamt: Quantidade de deslocamento

SRL

Realiza o deslocamento para direita: $R = A \gg \text{Shamt}$.

Operação: 0x8

Shamt: Quantidade de deslocamento

4. Memória de Instruções

A memória de instruções do MIPS é uma unidade de memória que armazena as instruções do programa, e é responsável por fornecer a próxima instrução para ser executada pelo processador.

O acesso as instruções pode acontecer de maneira aleatória o que define esta memória como uma RAM (*Random Memory Access*) e ela é de somente leitura. O acesso aos endereços da memória é dado por meio do registrador PC (*Program Counter*), onde a cada clock de ciclo é adicionado 4-bits, fornecendo uma nova instrução de

32-bits. A figura [5], ilustra a entrada **ADD(Address)** que recebe o endereço que será acessado e a saída **LI(Load Instruction)** que é a instrução armazenada no endereço especificado.

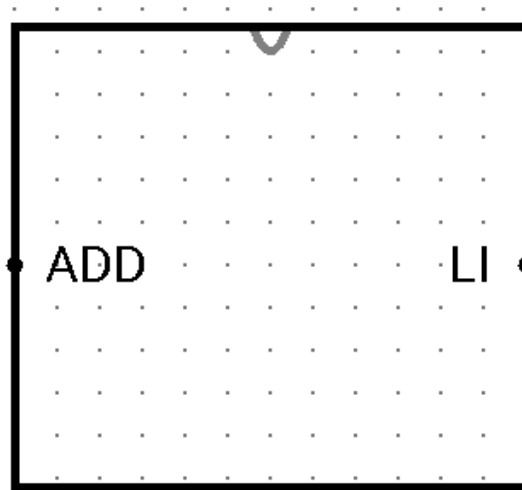


Figura 5: Memória de Instruções

4.1 Circuito

Para que esta memória possa ser implementada no Logisim, foi necessário realizar algumas mudanças de projeto pois o simulador não suporta memórias acima de 24-bits de endereçamento.

Sua implementação é simples e é utilizada dois distribuidores e uma memória ROM (*Read Only Memory*). O uso da memória ROM se dá pois esta é uma memória que pode ser acessada de maneira aleatória, assim como a RAM (*Random Access Memory*), porém não há como escrever nesta memória.

São dois os distribuidores utilizados o primeiro serve para separar os 32-bits de entrada, como as memórias no simulador possuem um endereçamento de 24-bits, o segundo distribuidor serve para eliminar os 8-bits restantes mais significativos e utilizar os 24-bits menos significativos. A figura [6] mostra a implementação dessa memória.

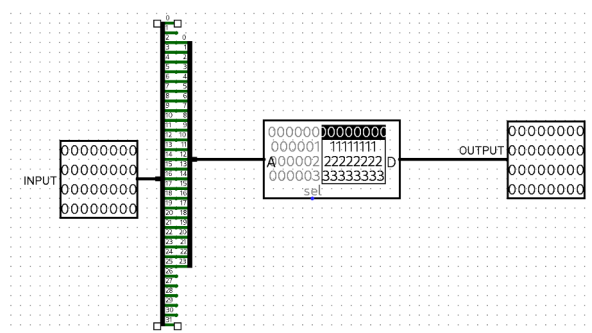


Figura 6: Circuito interno da Memória de Instruções

Na figura [6] o distribuidor está colocado de maneira a dividir a entrada por 4, isto é feito pois a entrada nas instruções de memória é dado por 4-bits, e para que o acesso possa ser feito na memória ROM sem “pularmos” instruções é necessário realizar esta divisão para termos um acesso contínuo.

5. Memória de Dados

A segunda unidade de memória que encontramos na arquitetura MIPS é o a memória de dados, que como seu nome já diz, armazena os dados que serão usados durante a computação.

Assim como a memória de instruções a memória de dados também pode ser acessada de maneira aleatória, com a diferença que a mesma pode ser escrita. Seu acesso é controlado pela unidade de controle, e este componente é utilizado pela ULA e o banco de registradores.

A figura [7] ilustra as entradas e saídas dessa unidade. As entradas são:

EL: *Effective Address Load*, entrada de 32-bits que especificam o endereço no qual os dados serão lidos ou escritos.

WD: *Write Data*, é uma entrada de 32-bits que será escrito no endereço de memória especificado na entrada **EL**.

MW(MemWrite): *Memory Write*, é um bit que ativa a escrita na memória.

MR(MemRead): *Memory Read*, é um bit que ativa a leitura da memória.

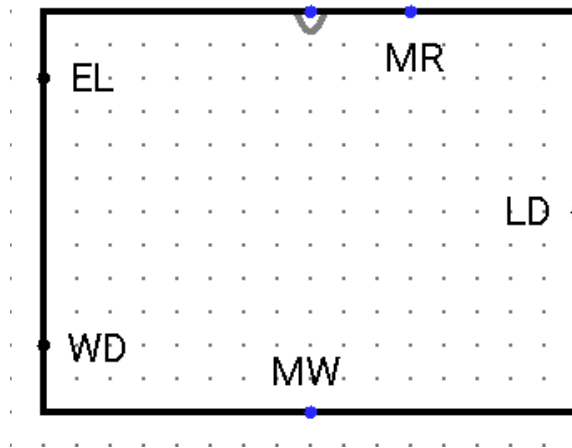


Figura 7: Circuito Memória de Dados

5.1 Circuito

Assim como na memória de instruções para a implementação desta memória foi usado dois distribuidores e uma memória RAM, esta memória é usada pois por ser uma memória de dados é possível ler e escrever.

Diferente da memória de instruções os distribuidores não foram colocados de maneira a dividir as entradas por 4, porém devido a memória RAM ter um endereçamento de 24-bits é necessário ignorar os bits mais significativos da memória.

Além dos distribuidores, a memória RAM também possui outras 2 entradas que são definidas por um bit, o bit de *load* que define que se o valor na memória será carregado e o bit de *store* que define se será escrito um valor na memória. A figura [8] as entradas e suas ligações na memória RAM usada no circuito.

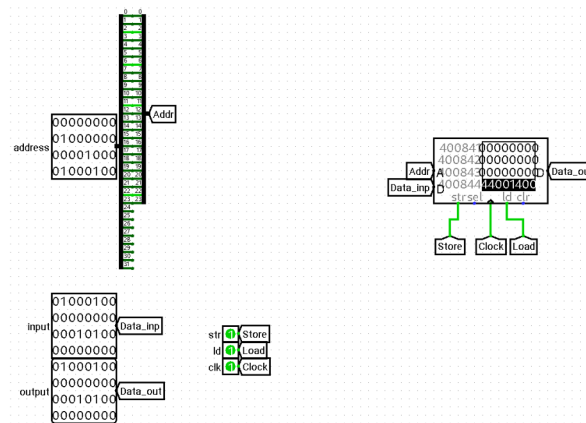


Figura 8: Circuito interno da Memória de Dados

6. Decodificador de Instrução

O módulo decodificador de instruções, consiste em um circuito que separa os campos de uma instrução de acordo com o tipo dela, assim segregando as informações contidas na instrução de 32 bits.

Na arquitetura escolhida temos 3 tipos de instruções, sendo elas R, I e J, cada uma delas sendo tratada diferentemente no decodificador, e para serem reconhecidas são lidos os 6 últimos bits, denominados **opcode**.

As instruções do tipo R todas tem um opcode igual a 0, já para instruções do tipo I é possível os seguintes opcodes, 8 (addi), 35 (lw), 43 (sw), 4 (beq) e 5 (bne), e para as instruções do tipo J, 2 (j) e 3 (jal), na figura [9] é possível identificar cada campo das instruções..

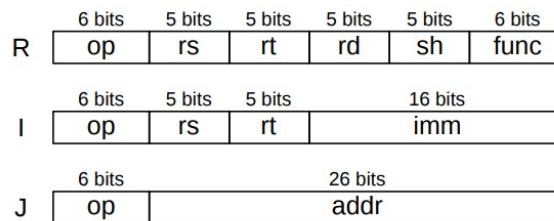


Figura 9: Repartições dos campos das instruções

Assim o decodificador tem a entrada da instrução 32bits, com as saídas, opcode 6 bits, rs 5 bits, rt 5 bits, rd 5 bits, shamt 5 bits, func 6 bits, imm 16 bits, addr 26 bits, na figura [10] é possível identificar a disposição da entrada e saídas.

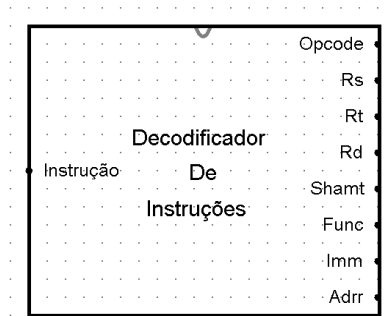


Figura 10: Módulo Decodificador de Instrução

6.1 Circuito

O circuito interno do decodificador de instruções figura [11], é composto por um decodificador feito de portas AND, que identificam o valor de opcode e sinaliza o tipo de instrução. O sinal do tipo da instrução é conectado aos DMXs, que controla para qual separador de campos vai ser enviado o restante da instrução sem o opcode, e nas saídas do separador são conectados os pinos de saída, com exceção do rs e rt, que são multiplexados, pois mais de um tipo de instrução utilizam eles, o tipo R e J.

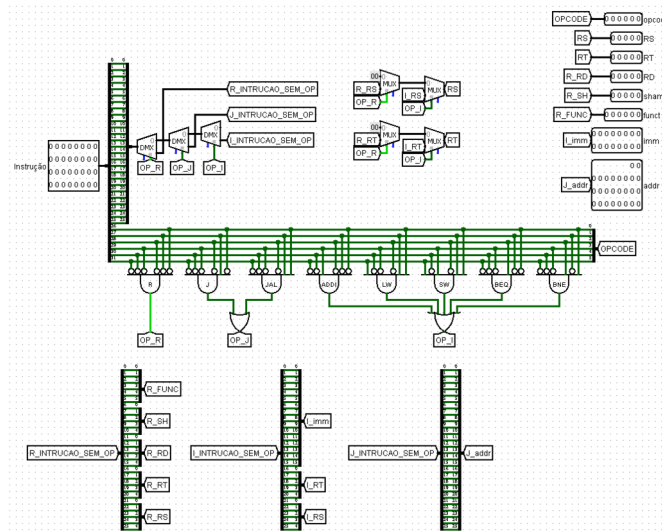


Figura 11: Circuito interno do Decodificador de Instruções

7. Unidade de Controle da ULA

Como descrito na sessão [3] a ULA executa diversas operações, cada uma definida por um código distinto. A geração desses códigos pode ser determinada uma unidade de controle. Esta unidade tem o nome de Unidade de Controle da ULA, e possui duas entradas: uma de 2 bits, chamada de ALUOp, que indica o tipo de operação a ser executada (por exemplo subtração para “beq”) e outra de 6 bits, chamada funct, que também influencia a operação de saída. A combinação dessas entradas é decodificada para então definir a operação desejada na ULA. Esse

esquema é utilizado para simplificar a unidade de controle, e em alguns casos pode até mesmo aumentar a velocidade das operações. A figura [12] ilustra as entradas e saídas de um Controle da ULA.

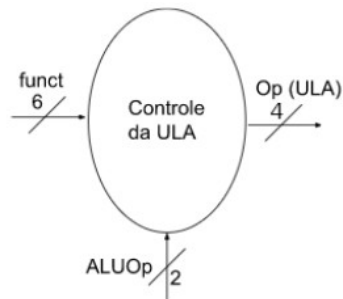


Figura 12: Controle da ULA

A tabela [2] ilustra os bits de entrada e saída desta unidade, perceba que no presente momento temos somente a definição das operações na qual a entrada da ALUOp seja 00. A decodificação de cada função prove uma operação da ULA assim como definido na sessão [3].

Tabela 2: Entradas e Saídas da Unidade de Controle da ULA

Entrada e Saída Unidade de Controle da ULA			
func	ALUOp	Op	Descrição
0x24	00	0000	AND
0x25	00	0001	OR
0x27	00	0010	NOR
0x26	00	0011	XOR
0x20	00	0100	SLT
0x22	00	0101	ADD
0x2A	00	0110	SUB
0x00	00	0111	SLL
0x02	00	1000	SRL

7.1 Circuito

O circuito de entradas é ilustrado na figura [13], já a figura [14] ilustra o comparador usado realizar as traduções das funções inseridas no funct. Sua saída usada em conjunto com ALUOp serve é o que vai definir o valor de saída de Op.

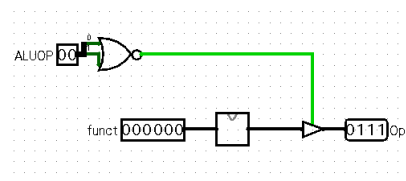


Figura 13: Circuito da unidade de controle da ULA.

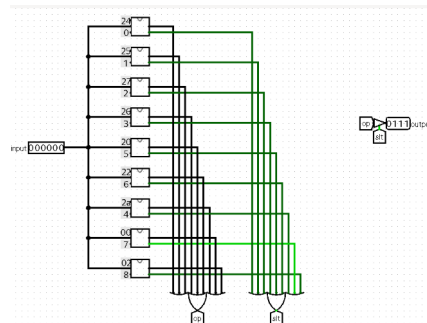


Figura 14: Circuito comparador da Unidade de Controle da ULA

8. Unidade de Controle Principal

A unidade de controle principal é responsável por controlar o fluxo dos dados de uma instrução, enviando para os módulos corretos dependendo do tipo da instrução. Na figura [15], pode-se identificar a entrada do opcode 6bits, que serve para identificar qual tipo de instrução está sendo executado, e as saídas RegDst 1bit, WriteReg 1bit, ExcOut 1bit, WriteRegEx 1bit, ALUOp 2bits.

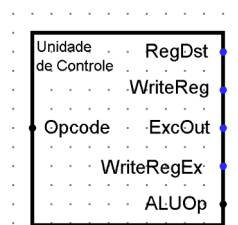


Figura 15: Unidade de Controle Principal

8.1 Circuito interno

O circuito interno da unidade de controle principal consiste em um decodificador de opcode[22], que identifica qual tipo de instrução é, e que uma sequência de MUX[21], direciona o valor correto para a saída, e sesse valores são definidos por constantes.

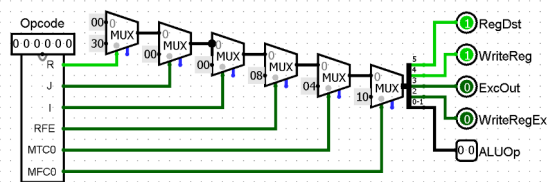


Figura 16: Circuito Interno da Unidade de Controle Principal

9. Apêndice

9.1 Registradores

Um registrador é um dos componentes básicos mais importantes de memória em circuitos digitais. Composto por um conjunto de flip-flops seu objetivo é o de armazenar um dado, como a arquitetura proposto é de 32-bits os registradores usados também seguem este padrão podendo armazenar 32-bits de dados.



Figura 17: Registrador do simulador Logisim

A figura[17] mostra algumas das entradas que temos no registrador dentro do simulador Logisim, a primeira delas no lado esquerdo é a entrada de dados onde são passados os 32-bits, abaixo dele temos o bit de select que irá indicar se a memória está selecionada(ou seja está ou não sendo usada), na parte de baixo do circuito temos mais duas entradas a primeira delas diz respeito ao ciclo, conforme selecionado no simulador durante o período da borda de subida ou descida que o dado será salvo, ao lado dele temos um bit que zera os dados que estão na memória e por fim do lado direito temos a saída dos dados que estão armazenados.

9.2 Memória RAM (Random Access Memory)

A memória RAM tem como objetivo em uma arquitetura prover uma maneira de armazenar valores temporários que podem ser acessados e modificados de forma aleatória. Sua implementação é complexa e exige diversos níveis de endereçamento para que esse acesso e a gravação possa ocorrer de forma aleatória. Infelizmente no simulador a memória RAM suporta um endereçamento de no máximo 24-bits o que na prática vai permitir que a mesma acesse até 2^{24} (ao longo do projeto mudanças foram realizadas para que o projeto pudesse funcionar).

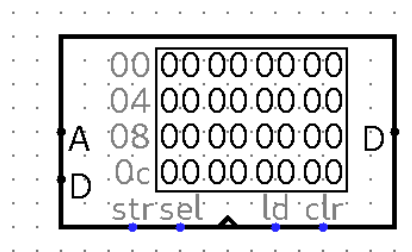


Figura 18: Memória RAM do simulador Logisim

Na figura[18] é possível ver as entradas e saídas do circuito de memória RAM do simulador, vale dizer que existem diferentes implementações do mesmo dentro do próprio simulador, por uma questão de conveniência foi escolhido usar a entrada de dados e endereçamento em entradas separadas, porém existem outras implementações que fazem uso dos dois em uma única porta. No circuito temos duas entradas a esquerda, sendo a primeira delas o endereço que se deseja acessar e abaixo o dado que se deseja inserir, na parte de baixo temos alguns bits de estado o primeiro deles é o *store* que se em 1 armazena o dado entrando e se em 0 ignora, o segundo deles é o *select* que funciona de maneira similar ao do registrador selecionando a memória RAM, logo depois temos o clock, depois o *load* que tem um funcionamento similar ao *store* porém seu uso se dá para ler o dado, após isso o clear que zera a memória e por fim do lado direito a saída de dados.

8.3 Memória ROM(Read Only Memory)

A memória ROM tem como objetivo similar a RAM no quesito de prover acesso a dados de maneira aleatória, porém seu uso é de leitura somente não podendo haver nenhum tipo de modificação nestes dados. Sua implementação também possui um certo nível de complexidade devido o acesso a endereçamentos. Assim como na RAM o simulador suporta um endereçamento de no máximo 24-bits.

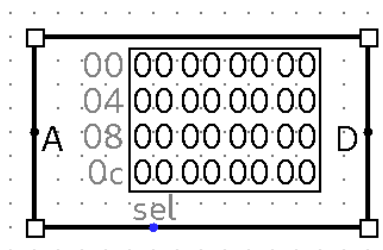


Figura 19: Memória ROM do simulador Logisim

Na figura[19] vemos que sua implementação é mais simples que a da RAM, pois como não há necessidade de gravar dados o nível de controle que o circuito precisa prover é menor. Neste circuito temos a esquerda a entrada do endereço que queremos ler, abaixo temos o bit *selection* que como já explicado serve para selecionar a memória e por fim temos a saída do lado direito que nós dá o dado que está armazenado.

8.4 DMX (Demultiplexador)

O demultiplexador é um circuito combinacional onde dado um valor de endereço, uma entrada única é direcionada a uma respectiva saída de acordo com o endereço, em outras palavras ele controla onde será conectada a entrada a uma saída específica.

Na figura [20], pode se analisar um demultiplexador, que tem como entrada de endereço um valor de 4 bits, que resulta em 2^4 saídas diferentes para uma única entrada.

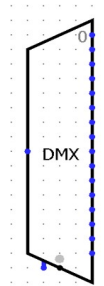


Figura 20: Demultiplexador

8.4 MUX (Multiplexador)

O multiplexador é um circuito combinacional que através de um endereço direciona qual entrada vai ser passada para uma única saída, assim podendo haver várias entradas possíveis para uma única saída. Em outras palavras ele controla qual entrada estará conectada a uma única saída.

Na figura[21], é possível identificar o multiplexador que tem a entrada do endereço com 4 bits, resultando em 2^4 possíveis saídas para uma única entrada.

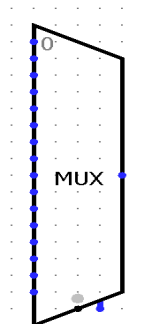


Figura 21:
Multiplexador

8.4 Decodificador de Opcode

O decodificador de opcode é utilizado para identificar o tipo de instrução a partir do valor de opcode, na figura [22], pode identificar e entrada do opcode 6bits, e as saídas R 1bit, J 1bit, 1bit, RFE 1bit, MTC0 1bit, MFC0 1bits.

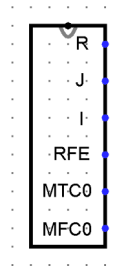


Figura 22: Decodificador de Opcode

8.4.1 Circuito interno

O circuito interno do decodificador de opcode, consiste em portas ANDs que representam o valor de cada instrução, o circuito está na figura[23].

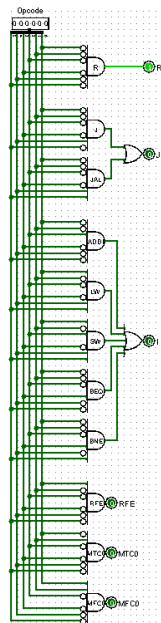


Figura 23: Circuito Interno do Decodificador de Instruções

10.Referências

[1] D. A. Patterson and J. L. Hennessy, *Computer organization and design: The hardware/software interface*. in Morgan Kaufmann series in computer architecture and design. Morgan Kaufmann, 2013.

[2] “Logisim,” www.cburch.com. <http://www.cburch.com/logisim/>

[3] J. L. Hennessy and D. A. Patterson, *Computer architecture: A quantitative approach*. in The morgan kaufmann series in computer architecture and design. Elsevier Science, 2017.