Execução do datapath monociclo

Eduardo Knabben Tiyo, Felipe Kenzo Suguimoto, Kaike Carvalho

Universidade Tecnológica Federal do Paraná – UTFPR

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

tiyo@alunos.utfpr.edu.br
felipekenzo@alunos.utfpr.edu.br
kaikecarvalho@alunos.utfpr.edu.br

Resumo

Este relatório procura compreender o funcionamento de um datapath monociclo utilizando um subconjunto de instruções da arquitetura MIPS, por meio da execução de códigos. A construção e análise do datapath foram realizadas no simulador Logisim. O objetivo final é obter uma compreensão do funcionamento básico de uma arquitetura RISC e entender como um código de alto nível é executado em uma máquina.

1. Introdução

A implementação de um datapath monociclo com o subconjunto de instruções MIPS é crucial para a compreensão da arquitetura RISC. Portanto, o projeto foi dividido em partes de diferentes processos de implementação, estes estão organizados e explicados nas cinco seções subsequentes do relatório. Na seção Conclusão será discutido o resultado final da implementação do datapath monociclo.

2. ULA & Banco de Registradores

A Unidade Lógica Aritmética (ULA) é um componente presente em processadores de 32 bits, realizando operações matemáticas e lógicas em números de 32 bits. Ela executa adições, subtrações, multiplicações, divisões e operações lógicas.

Juntamente com a ULA, o Banco de Registradores de 32 Bits armazena dados temporários. Cada registrador no banco, com uma largura de 32 bits, mantém dados para operações da ULA e armazena resultados temporários mantendo o acesso rápido a esses dados, algo que é essencial para o funcionamento do processador.

2.1 As operações da ULA

A ULA possui diversas operações lógicas e aritméticas, sendo elas AND, OR, NOR, XOR, SLT, ADD, SUB, SLL e SRL. Suas entradas consistem em dois números de 32 bits, identificados como A e B, juntamente com um código de 4 bits para selecionar a operação desejada. Além disso, há uma entrada de 5 bits chamada SHAMT, que determina a quantidade de bits a serem deslocados em operações de shift.

As saídas da ULA incluem:

- RESULT: Um valor de 32 bits que representa o resultado da operação escolhida entre A e B.
- ZERO: Um bit que é 1 se e somente se todos os bits em RESULT são zero, indicando um resultado nulo.
- OVERFLOW: Um bit que é 1 se e somente se ocorrer um overflow durante a operação, indicando que o resultado não pode ser representado corretamente em 32 bits.

Para utilizar efetivamente a ULA de 32 bits, é necessário compreender os códigos dos operadores, que determinam as operações a serem realizadas. Os códigos para as operações são:

0000 → AND: Realiza a operação lógica “AND” entre as entradas A e B.
0001 → OR: Realiza a operação lógica "OR" entre as entradas A e B.
0010 → NOR: Realiza a operação lógica "NOT OR" entre as entradas A e B.
0011 → XNOR: Realiza a operação lógica "EXCLUSIVE NOR" entre as entradas A e B.
0100 → ADIÇÃO: Realiza a operação de adição entre as entradas A e B.
0101 → SUBTRAÇÃO: Realiza a operação de subtração entre as entradas A e B.
0110 → SLL: Realiza a operação de Shift Left Logical em A, deslocando os bits especificados no SHAMT.
0111 → SRL: Realiza a operação de Shift Right Logical em A, deslocando os bits especificados no SHAMT.

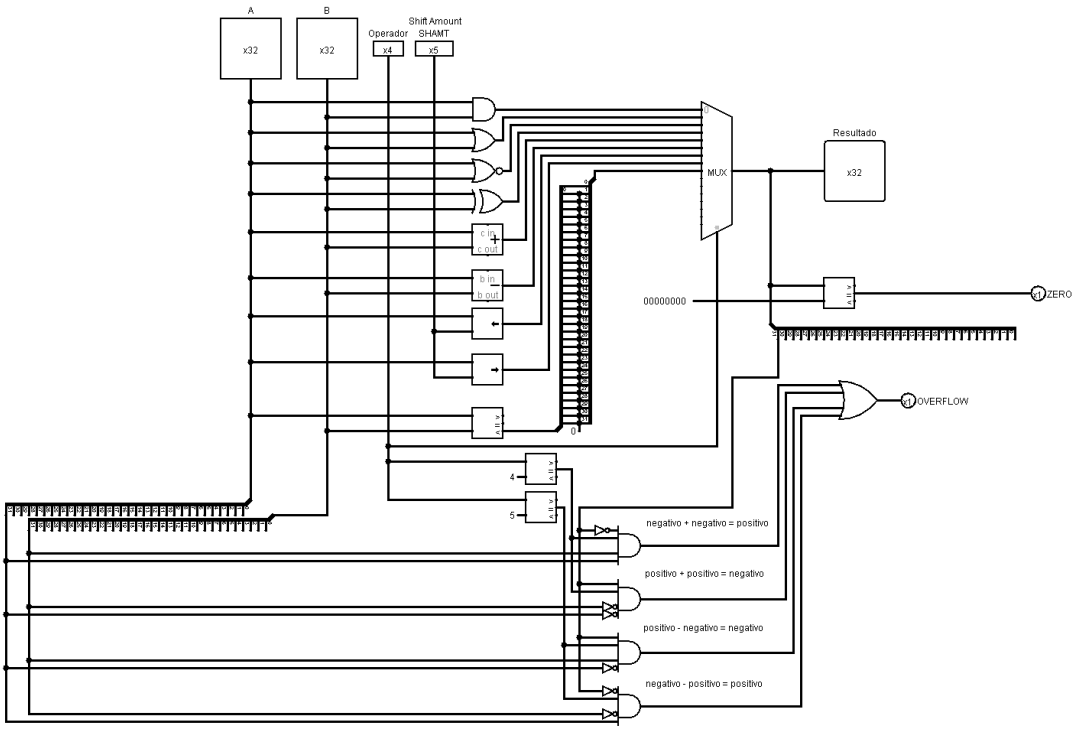

Figura 1 – Unidade Lógica Aritmética.

2.2 Comportamento do Banco de Registradores

O Banco de Registradores de 32 Bits é uma parte fundamental do processador que consiste em um conjunto de registradores de dados capaz de armazenar um valor de 32 bits. Cada registrador no banco tem um endereço único, que é usado para acessar ou modificar o conteúdo do registrador.

Os registradores são identificados por entradas de 5 bits, incluindo LR1, LR2 e WR. Além disso, existem entradas adicionais, como:

- WD (Write Data): Uma entrada de 32 bits que contém os dados a serem escritos no registrador especificado por WR.
- LD1 e LD2: São dados armazenados nos registradores especificados por LR1 e LR2, respectivamente.
- RegWrite: Um bit de controle que habilita (quando 1) ou desabilita (quando 0) a escrita dos dados contidos em WD no registrador identificado em WR.

As entradas LD1 e LD2 permitem o acesso aos dados nos registradores especificados, enquanto o WD contém dados para escrita. O RegWrite controla se a operação de escrita deve ser realizada.

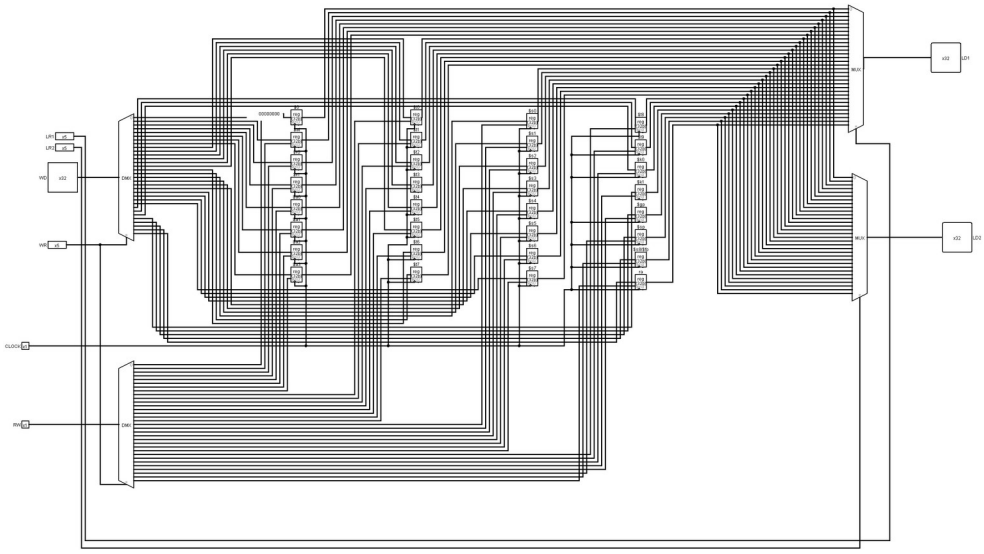

Figura 1.1 – Banco de Registradores.

3. Decodificador & Memórias

Nesta seção, serão apresentadas quatro partes fundamentais de um Datapath Monociclo, sendo elas: o Decodificador de Instruções, a Unidade de Extensão de Sinal, a Memória de Instruções e a Memória de Dados. Cada componente será acompanhado por suas respectivas imagens e especificações, oferecendo uma análise detalhada desses elementos do datapath.

3.1 Decodificador de Instruções

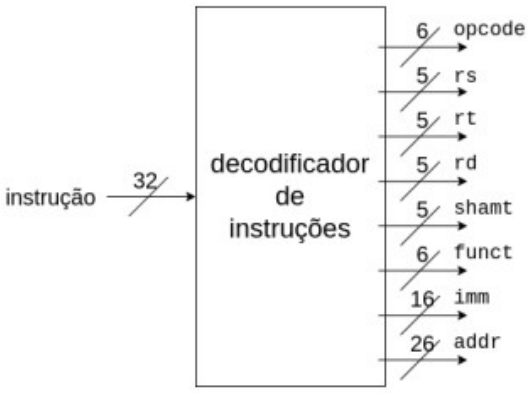

Figura 2 – Entradas e saídas do decodificador de instruções.

O decodificador de instruções é um componente que recebe os bits da instrução e decodifica em (opcode,rs, rt, rd, shamt, imm, addr), sendo o opcode comum para as 3 instruções existentes no MIPS (R, I e J).

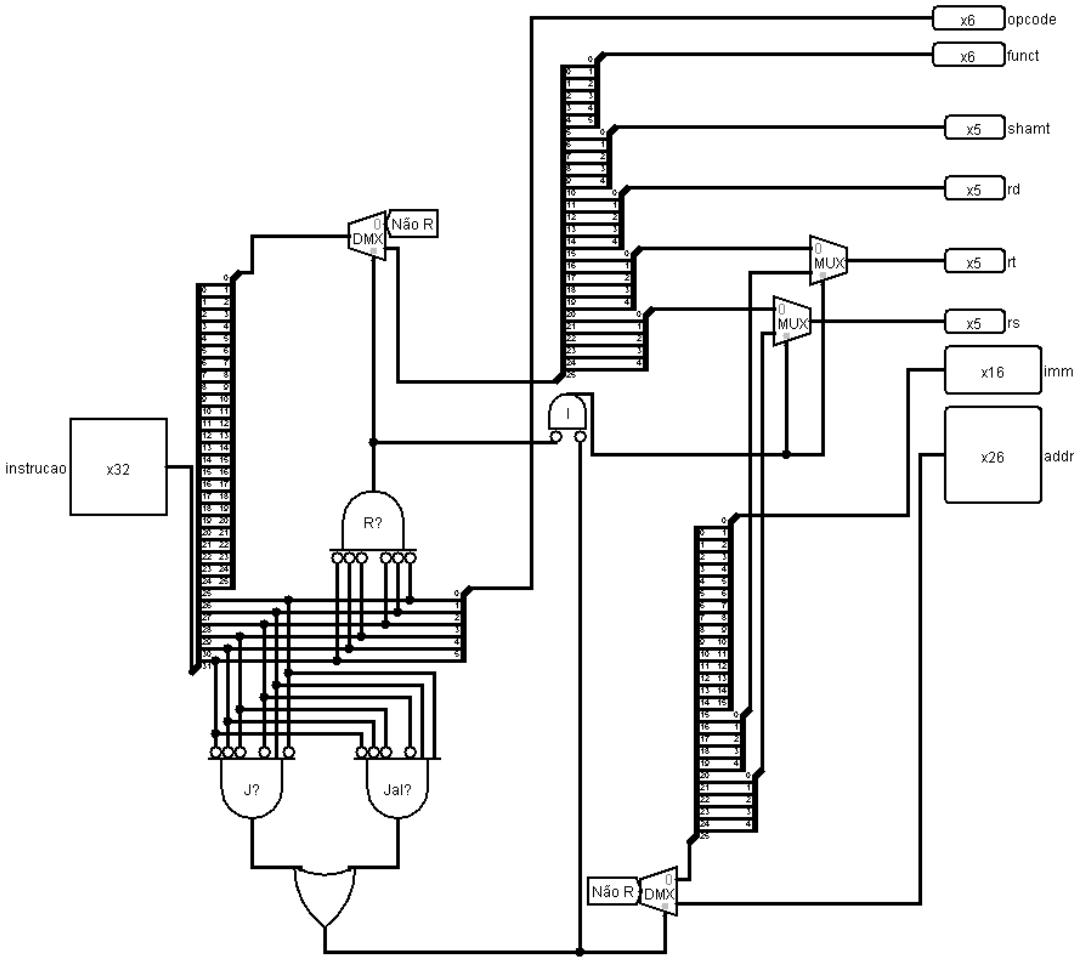

Figura 2.1 – Decodificador de Instruções.

Conforme a Figura 2.1, os 32 bits da instrução passados do PC são distribuídos em uma estrutura de dados. Os seis bits mais significativos correspondem ao opcode, que é comum para três tipos de instruções. Esses bits são diretamente conectados à entrada do opcode. No conjunto dos seis bits mais significativos, uma porta AND inteira negada é empregada para detectar a instrução do tipo R (opcode 000000). Se o opcode for 000000 (indicando uma instrução do tipo R), o sinal na saída da porta AND "R?" será ativado. Esse sinal é então direcionado para um DEMUX, que seleciona entre instruções não-R ou R.

Além disso, um bloco lógico OR é utilizado, composto por duas portas AND, para identificar os opcodes 2 e 3 (J e JAL). A saída deste bloco OR é conectada a outro DEMUX. Esse DEMUX, quando ativado, diferencia entre operações não-R, identificando se a instrução é do tipo I ou J com base na saída do bloco OR. Se a saída do bloco OR for 1, a instrução é do tipo J; caso contrário, é do tipo I.

É importante notar que as entradas "rs" e "rt" possuem multiplexadores (MUX) para determinar se os bits devem ser utilizados para uma instrução do tipo I ou R. Para essa seleção, uma porta AND de duas entradas, ambas

negadas, é empregada. Se a instrução não for do tipo R (indicado pelo sinal na saída da AND "R?") nem do tipo J (indicado pelo sinal na saída do bloco OR "J?" e "Jal?"), ela será uma instrução do tipo I. Por fim, as instruções do tipo J utilizam os bits restantes para formar o endereço de destino (addr). A tabela abaixo resume a distribuição dos bits para cada tipo de instrução:

Tipos de Instrução	Bits do OPCODE	Bits da Instrução
R	000000	[0-5] funct, [6-10] shamt, [11-15] rd, [16-20] rt, [21-25] rs
I	Diferentes de 000000 e 110000	[0-15] imm, [16-20] rt, [21-25] rs
J	000010(J) ou 000011(JAL)	Restantes para o endereço(addr)

3.2 Extensor de Sinal

No MIPS, um extensor de sinal é usado para estender números imediatos de 16 bits para 32 bits, mantendo o valor original. Se o bit mais significativo do número de 16 bits for 1 , os bits extras na extensão (bits 16 a 31) também serão 1. Se o bit for 0, os bits extras na extensão serão 0.

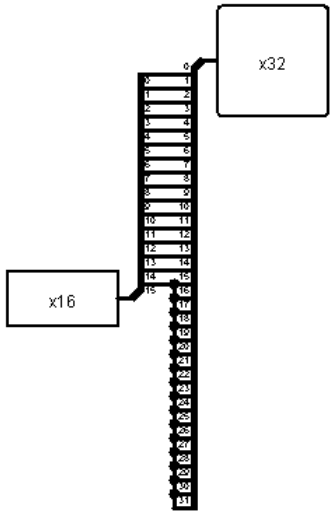

Figura 2.2 – Extensor de Sinal.

3.3 Memória de Intruções

A memória de instruções (Figura 2.3) é estruturada em três partes distintas: o campo de endereço, a memória ROM e a instrução. O campo de endereço possui 32 bits, enquanto a ROM suporta até 24 bits. Devido a essa diferença de tamanho, é necessário empregar dois distribuidores: um de 32 bits, que recebe o endereço completo, e outro de 24 bits, responsável por enviar dados para a entrada da ROM.

É importante notar que o distribuidor conectado à ROM é vinculado do bit 2 ao 24 do distribuidor principal. Esse arranjo é implementado para permitir que o ROM avance de 4 em 4, garantindo uma leitura

sequencial e correta das instruções. Essa organização da memória de instruções é essencial para o funcionamento do sistema.

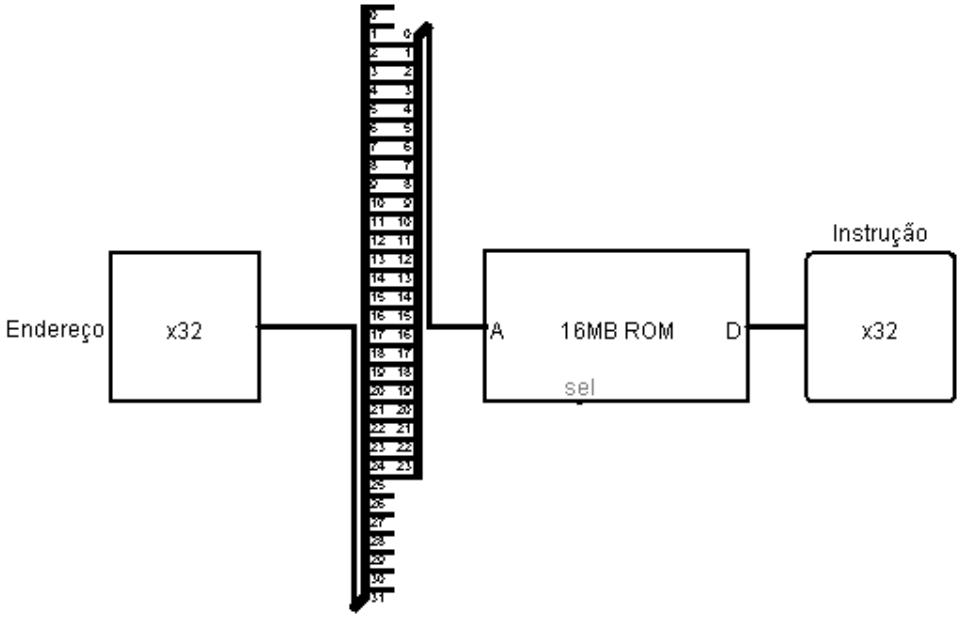

Figura 2.3 – Memória de Instruções.

3.4 Memória de Dados

A memória de dados (Figura 2.4) é um componente projetado para armazenar e manipular dados, permitindo a leitura e modificação de informações em diferentes endereços de memória. É composta por três elementos principais: o registrador EL, que contém 32 bits e atua como índice para os endereços de memória; o registrador WD, que também possui 32 bits e representa os dados a serem escritos na memória no endereço especificado durante operações de escrita; e a memória RAM, que possui duas entradas, uma para o endereço (com 24 bits) e outra para os dados (com 32 bits).

Para garantir compatibilidade com a entrada de endereço da memória RAM, é necessário utilizar dois distribuidores para ajustar o formato do registrador EL. Além disso, a memória RAM possui três entradas adicionais: "memory write" para escrever os dados no endereço especificado, "memory read" para ativar ou desativar a saída e um sinal de clock para sincronização. Essa configuração permite operações eficientes de leitura e escrita, garantindo que o sistema possa acessar e manipular dados de maneira precisa.

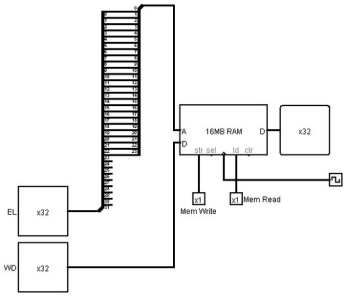

Figura 2.4 – Memória de dados.

4. Unidades de Controle

As unidades de controle são componentes fundamentais para o funcionamento do datapath. Elas desempenham o papel de coordenar e executar todas as operações que ocorrem no datapath, que é a parte responsável por realizar as operações aritméticas, lógicas e de transferência de dados. Além disso, elas desempenham um papel crucial para selecionar o caminho que os dados vão percorrer para realizar as instruções desejadas.

4.1 Unidade de Controle Principal

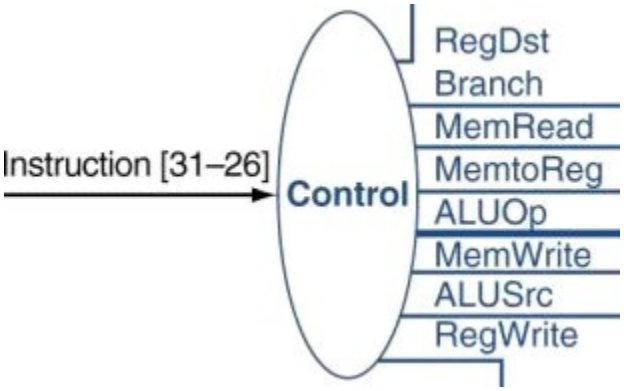

Figura 2.5 – Entradas e saídas da Unidade de Controle Principal.

A unidade de controle principal tem como objetivo determinar o caminho dos dados dentro do processador, ativando ou desativando flags de seleção de dados no datapath. Essa seleção é feita através dos seis bits mais significativos da instrução atual, ou seja, o opcode. Existem três tipos principais de instruções na unidade de controle: R, I e J.

Para as instruções do tipo R, cujo opcode é 000000, são ativadas as flags RegDST, WRITEREG e OPCODE. As instruções do tipo I são acionadas quando o opcode é 000101, 000100, 101011 ou 100011, representando as funções BNE, BEQ, SW e LW, respectivamente. Cada uma dessas funções ativa flags específicas: BNE ativa BRANCH, BNE e OPCODE; BEQ ativa BRANCH e OPCODE; LW ativa WRITEREG, ALUSCR, MemToReg e MemRead; e SW ativa ALUSRC e MemWrite.

As instruções do tipo J são ativadas quando o opcode é 000010 ou 000011, representando as funções J e JAL, respectivamente. A função JAL ativa as flags WRITEREG, JUMP e LINK, enquanto a função J ativa apenas a flag JUMP.

Além disso, a unidade de controle possui três funções para exceção quando o opcode é 100001, 100100 e 100010, representando as funções MTC0, MFC0 e RFE, respectivamente. A função MFC0 ativa as flags WRITEREGEX e EXCOUT; a função MTC0 ativa MTC0 e EXCOUT; e a RFE ativa WRITEREG, EXCOUT e WRITEREGEX. O propósito dessas funções é direcionar instruções que contenham erros de exceção para a unidade de controle de exceção, onde MTC0 leva a instrução, MFC0 retorna a instrução e RFE envia a instrução corrigida para o datapath.

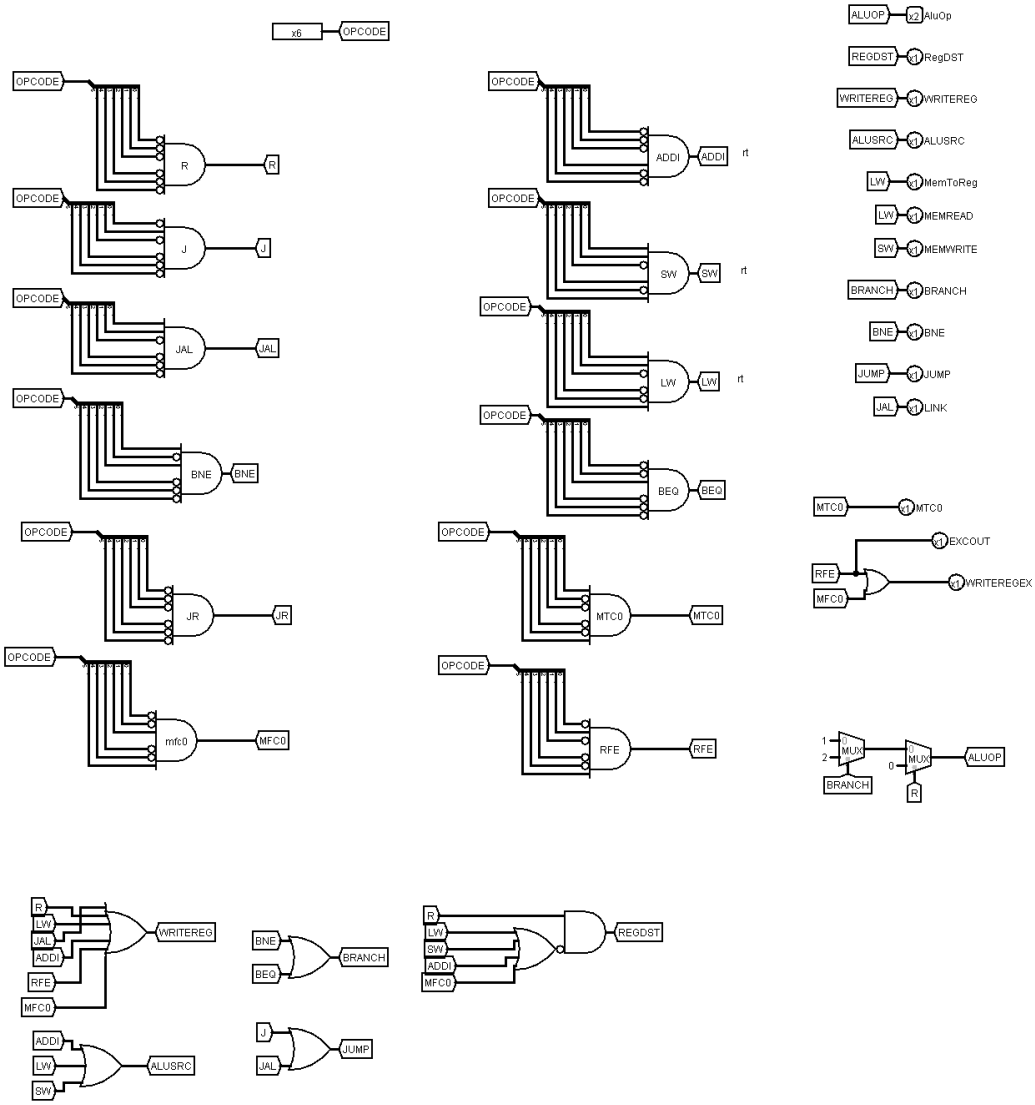

Figura 2.6 – Unidade de Controle Principal.

4.2 Unidade de Controle da ULA

A entrada FUNCT na Unidade de Controle da ULA determina a operação a ser realizada quando o OPCODE for 000000 (instrução R). A decisão é baseada apenas nos 4 bits mais significativos da FUNCT. Na Figura 2.7, existem dois distribuidores que selecionam esses bits e os direcionam para um MUX entre a FUNCT da instrução, o OPCODE 0100 (soma para LW e SW), ou o OPCODE 0101 (subtração para BRANCH).

A saída do MUX é determinada pelo aluOp, sendo 0 para instrução R, 01 para instrução I sem BRANCH e 10 para instrução I com BRANCH. Para a instrução JR (FUNCT 110000), embora teoricamente o OPCODE seria 1100, essa operação não está presente. Portanto, para implementar a instrução JR, foi utilizada a operação de soma, adicionando ao registrador PC o endereço guardado mais o valor 0.

O circuito verifica se FUNCT é 110000 e se aluOp é 00; nesse caso, a instrução é JR. Em um segundo MUX, se a instrução for JR, ele determina soma (0100) como o operador da ULA. Caso contrário, o circuito segue normalmente com o resultado do primeiro MUX.

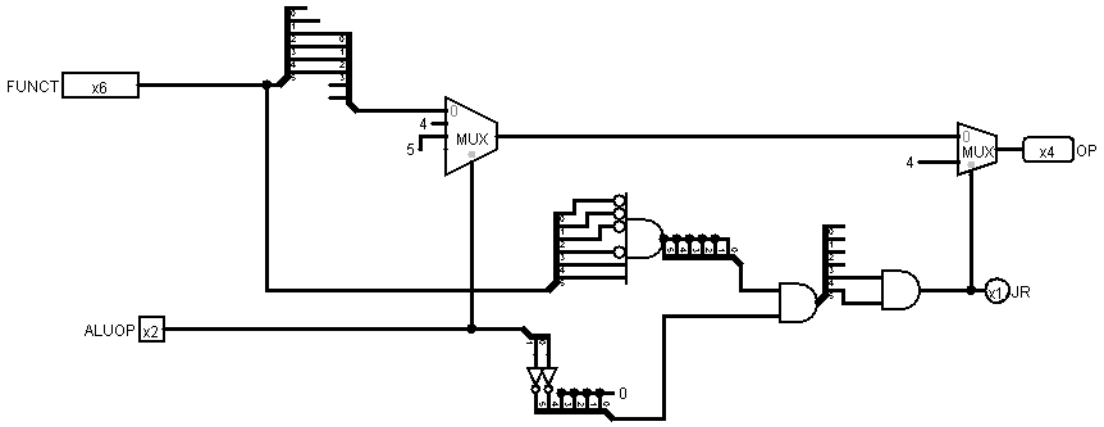

Figura 2.7 – Unidade de Controle da ULA.

FUNCT	OPERAÇÃO
000000	AND
000100	OR
001000	NOR
001100	XOR
010000	ADD
010100	SUB
011000	SLL
011100	SLR
100000	SLT
110000	JR

4.3 Unidade de Controle de Exceção

A exceção é uma mudança inesperada no fluxo de controle independentemente da origem, e pode ser tanto síncrona quanto assíncrona. Quando uma exceção ocorre, o controle é enviado para um programa chamado "exception handler", que resolverá o problema e posteriormente retornará no ponto onde o programa parou. O co-processador 0 (Unidade de tratamento de exceções) possui registradores internos que armazenam informações sobre o tratamento de exceções. O co-processador criado para este trabalho trata apenas os overflows aritméticos. Esse circuito possui dois registradores internos de 32 bits o EPC e RES, e um de 5 bits chamado WR.

A unidade de controle de exceções possui 5 entradas e 3 saídas:

Entradas	Função
Except	Indica se a instrução atual causou uma exceção.
EPC_in	Salva o valor que deve ser gravado no registrador EPC,que contém o endereço da instrução causadora da exceção.
WR_in	Identificador do registrador que receberia o resultado incorreto gerado que será gravado no registrador interno WR.
RES_data	Valor que vai ser gravado no registrador interno “RES”, que pode ter origem no resultado incorreto gerado na ULA ou no registrador lido no caso de instrução MTC0.
WriteRes	Habilita a escrita do valor res_data no registrador interno RES.

Saídas	Função
EPC_out	Último valor gravado no registrador EPC.
WR_out	Último valor gravado no registrador WR.
MFC0_out	Último valor gravado no registrador RES.

A Figura 2.8 retrata o circuito da unidade de controle de exceções, e seu funcionamento é descrito da seguinte maneira: O bit "Except" é uma flag que permite a escrita nos registradores EPC, WR e RES, ativando-se em caso de ocorrência de overflow. O bit "WriteRes" é uma flag que possibilita a escrita no registrador RES, ativando-se dependendo do valor de MTC0. O clock é aplicado a todos os registradores internos, tendo a função de atualizar os valores recebidos.

O EPC_in recebe o valor do registrador PC + 4 e o encaminha para o registrador interno EPC, que por sua vez o envia para a saída EPC_out. Essa operação visa armazenar o endereço que contém a ocorrência da exceção. O WR_in recebe o valor de WR, que é então enviado para o registrador interno WR, seguindo para a saída WR_out. Essa operação tem o propósito de identificar o registrador que receberia o resultado incorreto.

O RES_data recebe o valor a ser gravado no registrador interno RES e será enviado para a saída MFC0_OUT. Se o registrador RES for ativado pela flag "Except", indica que ocorreu algum erro de exceção; se ativado pela flag "WriteRes", significa que nenhum erro ocorreu.

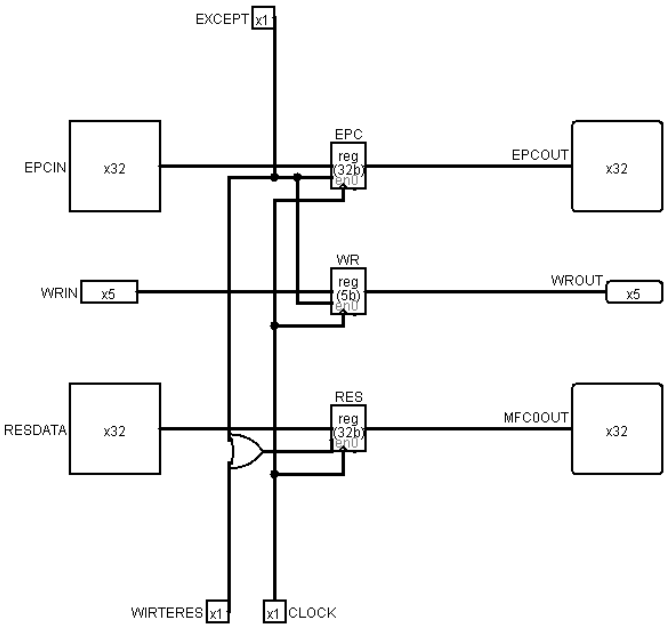

Figura 2.8 – Unidade de Controle de Exceção.

5. O Datapath

Por fim, a junção de todos esses componentes resulta em um datapath monociclo capaz de executar códigos de todas as instruções implementadas.

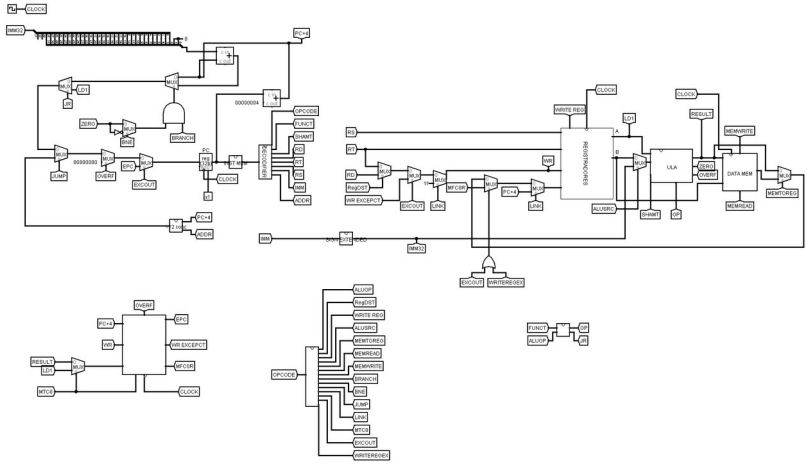

Figura 2.9 – Datapath

O funcionamento do datapath é coordenado pela unidade de controle, que sequencia as operações com base nas instruções do programa. Durante a execução, o datapath busca instruções na memória, as decodifica, executa as operações correspondentes e atualiza o estado do processador conforme necessário.

6. Implementação do código de alto nível

Esta seção, possuí uma representação em linguagem de alto nível, sua tradução assembly e o correspondente em código de máquina, juntamente com o tratamento de exceções.

Esse código em C implementa uma função chamada findMax para encontrar o valor máximo em uma matriz bidimensional.

```
int findMax(int rows,int cols,int m[rows][cols]){
    int max = m[0][0];
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            if(m[i][j] > max) max = m[i][j];
        }
    }
    return max;
}
int main(){
    int m[2][2] = {
        {-6, 8},
        {1,0}
    };
    int max = findMax(2,2,m);
    return 0;
}
```

Código traduzido para assembly MIPS:

```
    addi $a0, $0, 2
    addi $a1, $0, 2
    addi $a2, $0, 128
    jal FINDMAX
    sw $v0, 0($s3)
    j EXIT2
    FINDMAX:
            lw $t0, 0($a2)
    LOOP1:
            beq $s0, $a0, EXIT1
            add $t1, $0, $0
            addi $s0, $s0, 1
    LOOP2:
            beq $t1, $a1, LOOP1
            lw $t2, 0($a2)
            slt $t3, $t2, $t0
            addi $a2, $a2, 1
            addi $a1, $a1, 1
            bne $t3, $0, LOOP2
            add $t0, $0, $t2
            j LOOP2
    EXIT1:
            add $v0, $t0, $0
            jr
    EXIT2:
```

Código traduzido para linguagem de máquina:

```
0x20040002 // $a0 = 2
0x20050002 // $a1 = 2
0x20060080 // $a2 = 128
0x0c000050 // jal FINDMAX
0xae620000 // max = $v0
0x8CC80000 // max = m[0][0]
0x1204003b // beq (if (i==rows) exit)
0x00004810 // j = 0
0x22100001 // i++
0x1125fffc beq (if (j==cols) loop1)
0x8cca0000 // $t2 = m[i][j]
0x01485820 // slt
0x20c60001 // m++
0x21290001 // j++
0x1560fffa // bne loop2
0x000a4010 // max = m[i][j]
0x08000054 // j loop2
exit:
0x01001010 // v0 = max
0x03e00030 // jr
0xae620000 // sw
0x08000100 // j
```

O código para o tratamento de exceções:

```
0x028c6810 // add
0x901a0000 // mfc0
0x0000d010 // add
0x84000000 // mtc0
0x88000000 // rfe
```

7. Conclusão

A implementação do datapath monociclo, baseado nas instruções MIPS, ofereceu um bom entendimento da arquitetura RISC. Explorando desde a ULA até o tratamento de exceções e a implementação de códigos de alto nível. Ao traduzir a função findMax para assembly MIPS e código de máquina, observamos diretamente a transformação de instruções de alto nível em operações de baixo nível, para que assim, as intruções fossem utilizadas no caminho de dados.

8. Referências

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