UNIVERSIDADE
FEDERAL DO CARIRI

PROJETO DE ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

José Vitor Dias dos Santos e Antonio Nunes de Oliveira Filho

MIOPS

Juazeiro do Norte – CE

05/04/2024

INTRODUÇÃO:

Neste Relatório será apresentado todo o desenvolvimento do processador MIOPS que tem como funcionalidade soma, subtração, multiplicação, comparação, porém tem como objetivo principal a multiplicação de matrizes 4x4.

O relatório será dividido por tópicos de desenvolvimento, como o desenvolvimento do código em C, depois em Assembly, e todo o desenvolvimento da lógica do processador.

DESENVOLVIMENTO:

Código em C:

Inicialmente no desenvolvimento do processador para que possamos entender o funcionamento da multiplicação de matrizes 4x4 em assembly tivemos que fazer as funções em C, fizemos todo o código funcional em C, contendo a lógica da multiplicação de matrizes.

Código em Assembly:

Depois do código em C tivemos que fazer toda a “tradução” para o Mars Mips para que tivéssemos a ideia de como fazer nosso próprio código em assembly. Então, foi feita a “tradução” utilizando a mesma lógica do código em c.

miopes.asm

```
.data
    matriz1: .word 1, 2, 3, 4
             .word 5, 6, 7, 8
             .word 9, 10, 11, 12
             .word 13, 14, 15, 16

    matriz2: .word 1, 1, 1, 1
             .word 1, 1, 1, 1
             .word 1, 1, 1, 1
             .word 1, 1, 1, 1


.text
 miops:
        la $a1, matriz1
        la $a2, matriz2

        li $s0, 4 #so uma constante


        sub $sp, $sp, $t1 # liberando o espaço
        move $s3, $sp

        li $t3, 0 # i = 0

  for_i:
        beq $t3, $s0, end_i

        li $t4, 0 # j = 0
    for_j:
        beq $t4, $s0, end_j

        li $t9, 0
        li $t5, 0 # k = 0
      for_k:
        beq $t5, $s0, end_k

        sll $t7, $t3, 2
        add $t7, $t7, $t5
        sll $t7, $t7, 2
        add $t7, $t7, $a1

```

Imagem 1: Imagem do código em Assembly

Começamos com o .data e foi implementada duas matrizes 4x4. Em seguida foi iniciado o .text em que foi feito toda a alocação de matrizes para a matriz 1, 2 e resultado. Após isso, temos a implementação dos For's, após os 3 for's é feita a lógica para pegar os valores da matriz 1 e 2 e colocar a soma das multiplicações delas para colocar na memoria alocada da matriz resultado. Ademais, é feita a lógica de imprimir a matriz resultado.

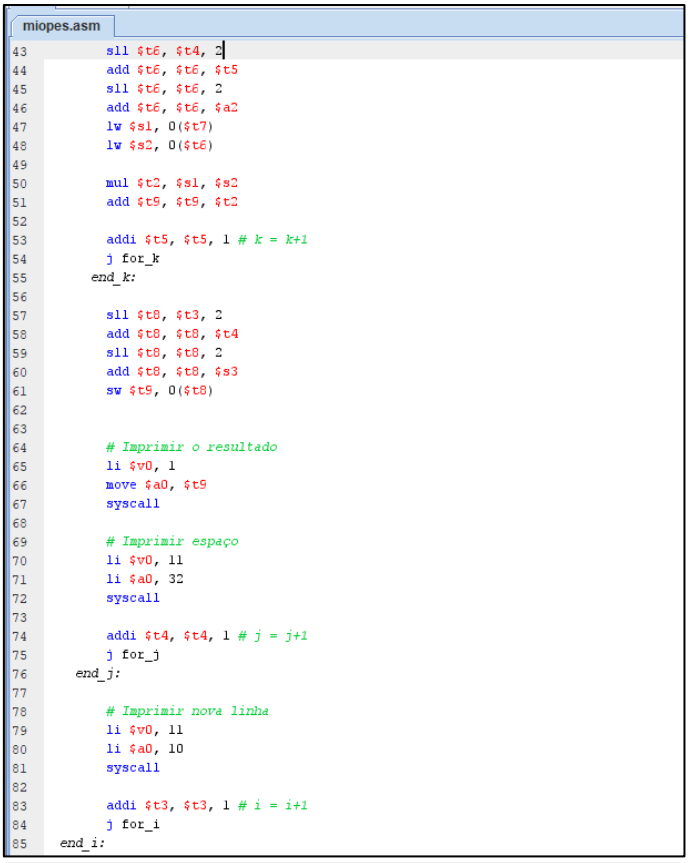

Imagem 2: Restante do código em Assembly

O restante do código em assembly é feita o restante da lógica do código, e assim é finalizado o desenvolvimento da lógica do código em assembly.

Desenvolvimento da lógica do MIOPS:

Para nosso processador foi feita a lógica de ter 16 registradores, então neste caso deve-se conter 4 bits para os registradores, pois 2^4=16. Então, como irão ser utilizados 3 registradores, sendo o registrador 1, registrador de destino e registrador 2, que no caso já totaliza 4x3=12 bits, como serão utilizados ao total 32 bits, temos que os 4 primeiros bits serão utilizados para o opcode, e depois 12 bits para os registradores, então finaliza com os 16 bits do imediato.

OUTCODE	REG1	REGDEST	REG2	IMEDIATO
0-3	4-7	8-11	12-15	16-31

Porém no Logisim fica da seguinte maneira;

OUTCODE	REG1	REGDEST	REG2	IMEDIATO
28-31	20-23	16-19	24-27	0-15

Essa é a configuração da distribuição dos bits do exemplo do ADDM, em que coloca no registrador de dest a soma dos reg1+reg2.

Após a distribuição, devemos codificar os registradores utilizados e as funções utilizadas em nosso código, em que deve ser feita a lógica do regdest, jump, memtoReg, Branch, MemRead, regWrite, Aluop, Alu01, Alu02.

$t0	0000
$t1	0001
$t2	0010
$t3	0011
$t4	0100
$t5	0101
$t6	0110
$t7	0111
$s0	1000
$s1	1001
$s2	1010
$s3	1011
$s4	1100
$s5	1101
$s6	1110
$v0	1111

Está é a codificação dos registradores utilizados, em que foram utilizados 16 espaços.

funct	opcode	regdest	jump	branch	memtoreg	aluop	memwrite	Alu01	Alu02	regwrite
ADDM	0000	0	0	0	0	0	0	0	0	1
ADDIM	0001	0	0	0	0	1	0	0	0	1
LWM	0010	0	0	0	0	1	1	0	0	0
SWM	0011	0	0	0	1	1	0	0	0	1

LIM	0100	1	0	0	0	1	1	0	0	1
MULTM	0101	0	0	0	0	0	0	1	0	1
BEQM	0110	0	0	1	0	0	0	1	1	0
SUBM	0111	0	0	0	0	0	0	0	1	1
JM	1000	0	1	0	0	1	0	0	0	0

Agora é feita a codificação das funções utilizadas, a partir desta tabela podemos criar a Unidade de Controle (UC).

Banco de Registradores:

O banco de registradores possui 16 registradores, e deve possuir as entradas de reg read 1, reg read 2, writeData, WriteReg, uc, clock e clear. E tem como saídas, reg1-data e reg2-data. Após isso, devemos fazer a lógica de escolha dos registradores que serão utilizadas na função/processador, tem-se que as entradas e saídas tem que possuir 32 bits.

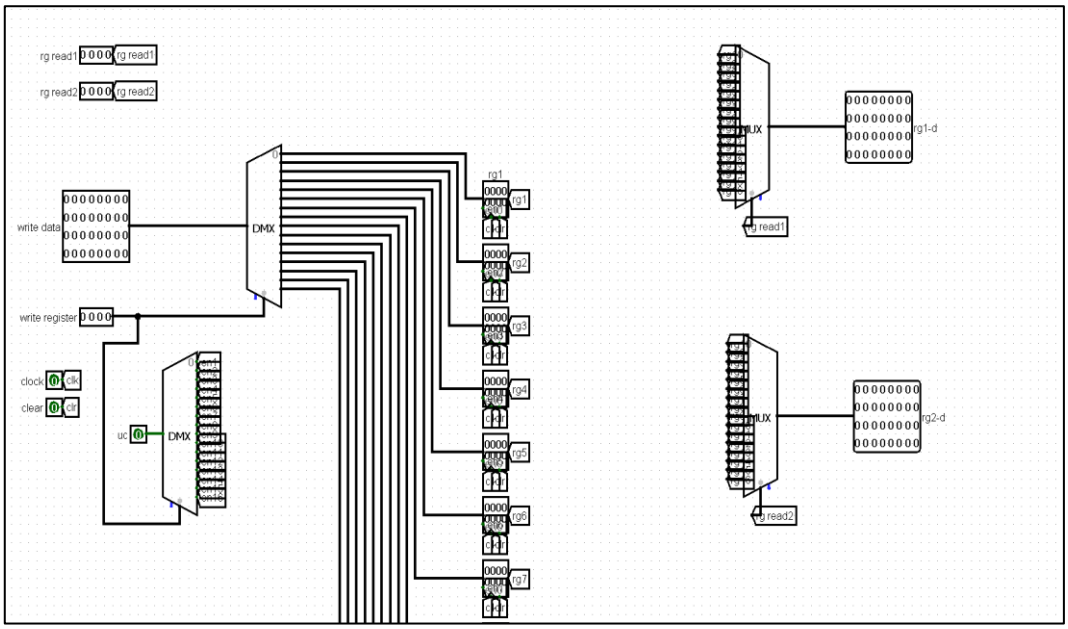

Imagem 3: Banco de Registradores

Foram utilizados alguns multiplexadores e demultiplexadores na seleção dos registradores.

ULA:

A ULA é a função responsável pelas operações lógicas e aritméticas do processador, na ULA utilizada terá como operações de soma, subtração, multiplicação e de comparação.

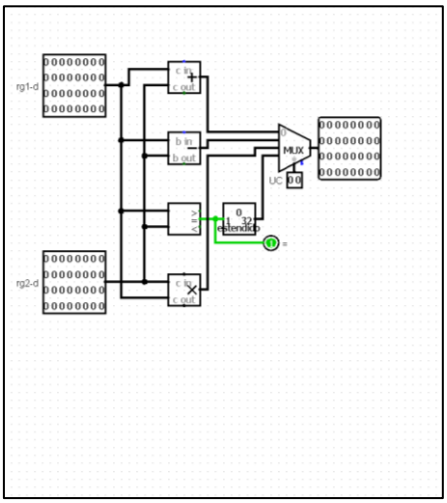

Imagem 5:ULA

A ULA tem duas entradas do reg1-data e do reg2-data, além da entrada de seleção da operação. Então, é feita as ligações das entradas dos reg nas operações lógicas/aritméticas, e as saídas das operações são conectadas no multiplexador, e possui como seletor um de 2 bits, além de possuir uma saída “igual” que será utilizada na parte de comparação lógica. E por fim, possui uma saída de 32 bits com os resultados.

Unidade de Controle:

A Unidade de controle é uma das partes mais importantes do processador, pois é por ele que ocorre toda a ligação e processamento do processador, resumindo, é a parte controladora do processador. Para fazer a tabela verdade da unidade de controle é utilizada a tabela utilizada na tabela 2. Para construir o circuito é utilizada uma função do próprio do Logisim.

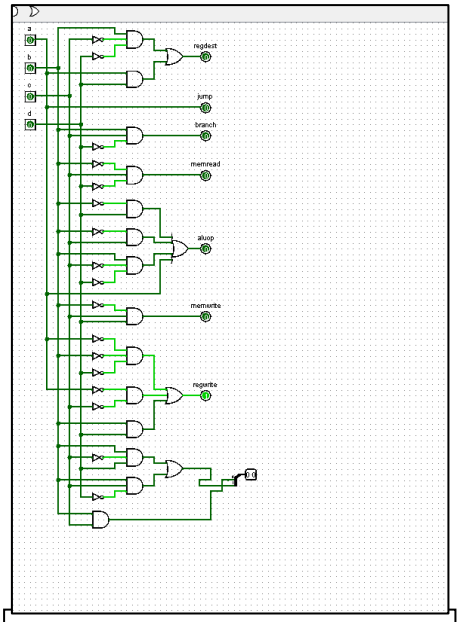

Imagem 6: UC

Neste caso foi feita toda uma lógica para que será feita corretamente seguindo a tabela feita.

Memória de instrução:

Para a memória de instrução teremos que ter um contador que pula as instruções conforme a borda de subida do clock, e é feita algumas extensões de bits de 16 para 32 quanto de 32 para 16 bits.

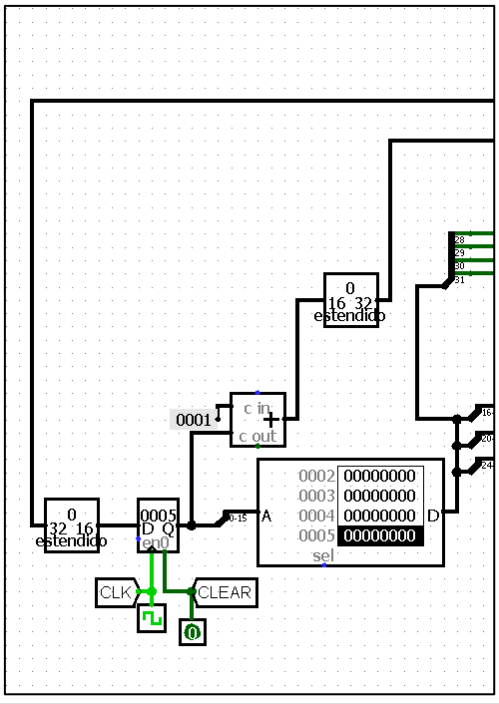

Imagem 7: Memória de instrução

No registrador utilizado também é conectado tanto o clock quanto o clear, e no somador é conectado o resultado do registrador com uma constante igual a 1. A memória de instrução é utilizada uma memória ROM que armazena as instruções que serão utilizadas.

Restante do processador:

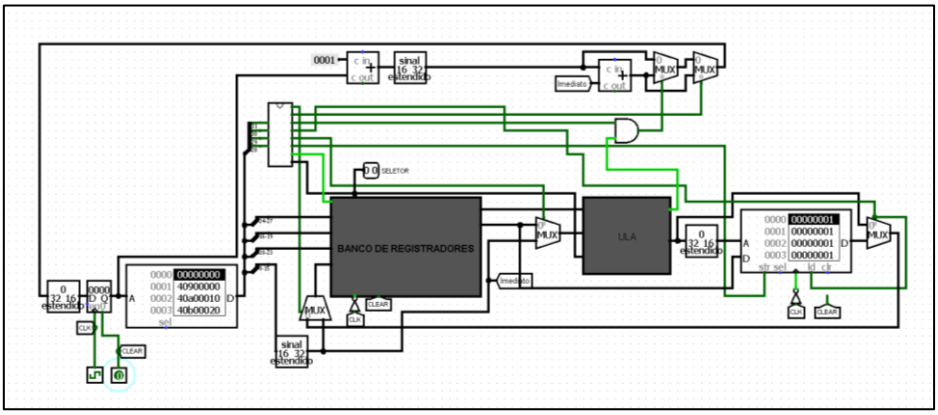

Imagem 8: Restante do processador

Temos que o processador deve ser composto pela ULA, unidade de Controle, Banco de registradores, memoria de instruções e memoria de dados. Então, temos que os fios da unidade de controle devem conectar todos os outros componentes para que funcione.

Na multiplicação de matrizes no logisim, devemos inicialmente colocar os dados das matrizes na memoria de dados, para que seja possível acessar esses valores com a função LW e assim multiplicar as matrizes.

Foram feitas as instruções de add, addi, lw, sw, sub, li, mul, beq e j. E a partir dessas instruções é possível fazer a multiplicação de matrizes.

CONCLUSÕES:

Temos que esse projeto de arquitetura foi muito importante para colocarmos em prática todos os nossos conhecimentos adquiridos na disciplina e também os conhecimentos de circuitos digitais.

Na elaboração do projeto enfrentamos diversos desafios, desde linhas de códigos errados até instruções mal formuladas, então através dos erros foi possível concluir o trabalho com sucesso.