



**UNIVERSIDADE FEDERAL DO CARIRI
CENTRO DE CIÊNCIAS E TECNOLOGIA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**

**Av. Tenente Raimundo Rocha Nº 1639
Bairro Cidade Universitária
Juazeiro do Norte - Ceará
CEP 63048-080**

JOSÉ FELIPE BARBOSA DA SILVA – 2022003521

**ATIVIDADE AVALIATIVA - TESTE 2
PROJETO 2 - MIPS-like**

INTRODUÇÃO.....	3
ESTRUTURA DO PROGRAMA.....	3
BANCO DE REGISTRADORES.....	4
UNIDADE LÓGICA E ARITMÉTICA.....	5
UNIDADE DE CONTROLE.....	6
MEMÓRIA DE DADOS.....	7
ESTRUTURAS ADICIONAIS.....	8
CÓDIGO DE MULTIPLICAÇÃO DE MATRIZES.....	9

INTRODUÇÃO

A atividade avaliativa em questão trata da construção de um processador através do simulador Logisim. O item construído deverá ser capaz de efetuar as operações necessárias para uma multiplicação de matrizes 4x4, indo de somas simples até os *loads* e *saves* na memória. Como base para a construção do circuito e orientador principal para a construção, a seguinte estrutura foi fornecida como “esqueleto” do projeto.

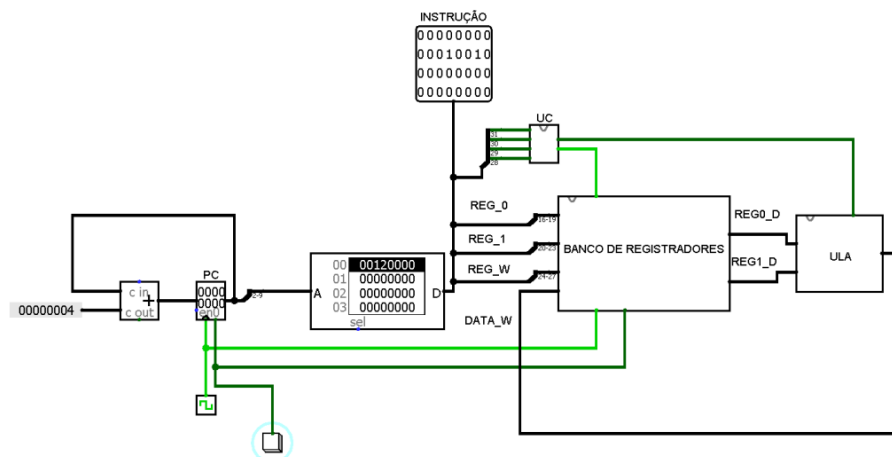


Figura 1: Estrutura de base fornecida.

ESTRUTURA DO PROGRAMA

Com o intuito de executar o trabalho de forma mais simples, a ideia inicial foi a construção do algoritmo de multiplicação de matrizes em uma linguagem de programação de programação com maior familiaridade e de alto nível. No caso, a escolha em questão para esse trabalho foi a linguagem de programação C para estruturação inicial. Posteriormente, já com o programa em C completo e funcional, foi feita a adaptação direta para o Assembly MIPS. Tal tradução foi feita com o intuito de definir as funções essenciais para o algoritmo que se desejava concluir e até mesmo encontrar uma quantidade adequada de registradores para a realização das operações.

```

Algoritmo-MultiMatrix EM C - Bloco de Notas
Arquivo  Editar  Formatar  Exibir  Ajuda
#include <stdio.h>
#include <stdlib.h>

void print_mat(int mat[][4])
{
    for (int i = 0; i < 4; i++)
    {
        printf("\n");
        for(int j = 0; j < 4; j++)
            printf("%d ", mat[i][j]);
    }
}

int main ()
{
    int mat_1[4][4] = {
        1, 2, 3, 4,
        5, 6, 7, 8,
        9, 10, 11, 12,
        13, 14, 15, 16};

    int mat_2[4][4] =
    {1, 0, 0, 0,
    0, 1, 0, 0,
    0, 0, 1, 0,
    0, 0, 0, 1
    };

    /*{
        1, 2, 3, 4,
        5, 6, 7, 8,
        9, 10, 11, 12,
        13, 14, 15, 16};*/

    int mat_3[4][4];

    for (int i = 0; i < 4; i++)

```

Figura 2: Trecho do código de multiplicação de matrizes em C.

```

Mult_DEFINITIVA - Bloco de Notas
Arquivo  Editar  Formatar  Exibir  Ajuda
addi $s2, $s2, 1
j for_de_j

fim_de_j:
addi $s1, $s1, 1
j for_de_i

fim_de_i:
li $s1, 0 #Índice I
li $t0, 1 #Valor a ser incrementado em cada elemento da matriz
li $t1, 4 #Aqui representa os limites pra I e J
for_p_i:
beq $s1, $t1, fim_p_i
li $s2, 0 #Índice J
for_p_j:
beq $s2, $t1, fim_p_j
li $s3, 0 #Índice K
for_p_k:
beq $s3, $t1, fim_p_k
sll $t3, $s1, 4
sll $t4, $s2, 2
add $t3, $t3, $t4 # T3 e T4 para preencher a matriz resultado
sll $t5, $s1, 4
sll $t6, $s2, 2
add $t5, $t5, $t6 #T5 e T6 para a matriz 1
sll $t7, $s3, 4
sll $s4, $s2, 2
add $t7, $t7, $s4 # T7 e S4 para a matriz 2
lw $s5, meuArray1($t5)
lw $s6, meuArray2($t7)
mul $s6, $s6, $s5
lw $s7, resultado($t3)
add $s7, $s7, $s6
sw $s7, resultado($t3)
addi $s3, $s3, 1
j for_p_k

fim_p_k:
addi $s2, $s2, 1
j for_p_j

fim_p_j:
addi $s1, $s1, 1
j for_p_i

fim_p_i:

```

Figura 3: Trecho da adaptação do código da figura 2.

BANCO DE REGISTRADORES

Com o código já estruturado e a quantidade de registradores necessária em mãos, o primeiro passo tomado foi a reestruturação do banco de registradores. Que se deu como necessária em decorrência da quantidade maior de espaço no banco para trabalho com as operações requeridas pelas operações. Com isso, o banco de registradores recebeu um incremento, indo de 4 para 16 registradores no total, com o número 0 reservado para não

ser utilizado por motivos posteriores. Com o uso de pontes para simplificar a fiação, o banco de registradores foi a parte mais rápida de se trabalhar.

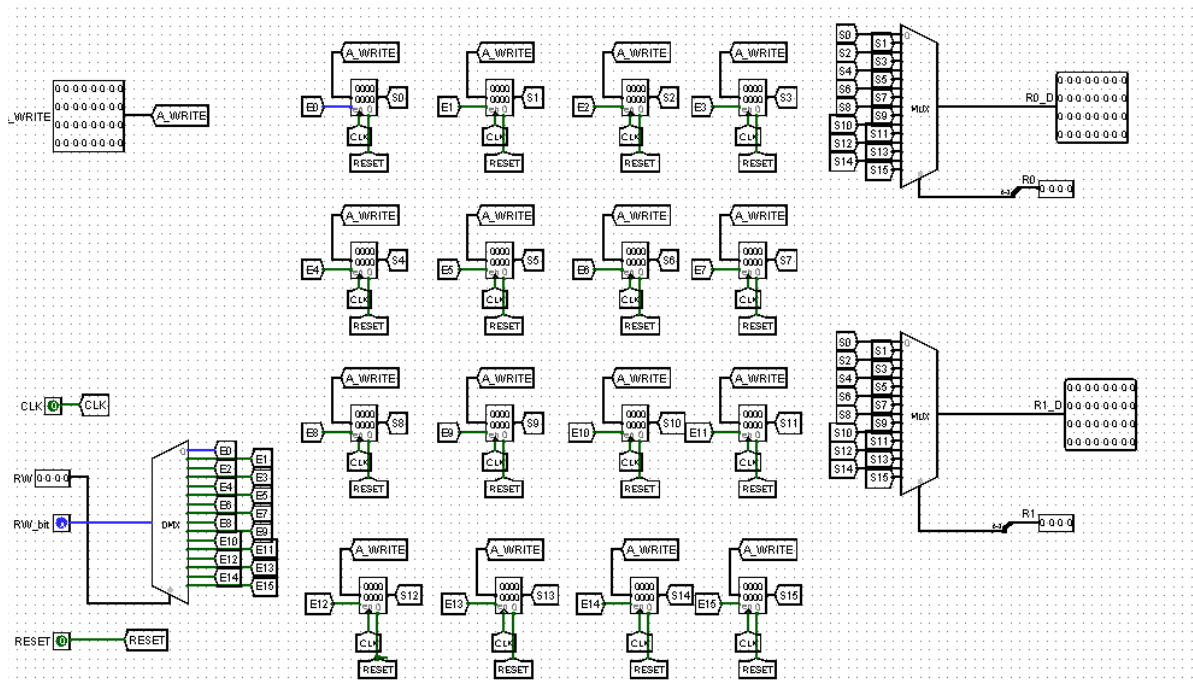


Figura 4: Estrutura final do banco de registradores após a alteração

UNIDADE LÓGICA E ARITMÉTICA

O próximo passo foi a construção da Unidade Lógica e Aritmética como definidora dos códigos para as operações. Usando o código em Assembly MIPS, ficou definido que as operações que poderiam ser escritas na ULA e que seriam necessárias poderiam ser expressas em apenas 5, sendo elas: soma (com soma com imediato), subtração, multiplicação, comparação e o Shift Left Lógico. Nesse momento, foram definidos os códigos de operação pela sequência de entradas dos demultiplexadores utilizados. Dessa maneira, os códigos ficaram definidos como na tabela logo abaixo:

Operação	OPcode
Adição	0000
Subtração	0001
Multiplicação	0010
Comparação	0011
SLL	0100
Soma imediata	0101

Uma observação cabível é que a operação de soma imediata retorna o mesmo código de adição da soma comum, porém também traz consigo da um bit de uso imediato da Unidade de Controle. Esse bit é responsável por selecionar o valor do imediato no multiplexador no momento de efetuar a soma. E vale também dizer que, como a soma é a operação 0000 e existe um operador 0, muitas vezes a operação 0000 de quando o PC terminou todas as funções era lida acabava como sendo interpretada como a operação “reg 0 = some reg0 + reg0” e esse é o motivo da decisão de deixá-lo inutilizado.

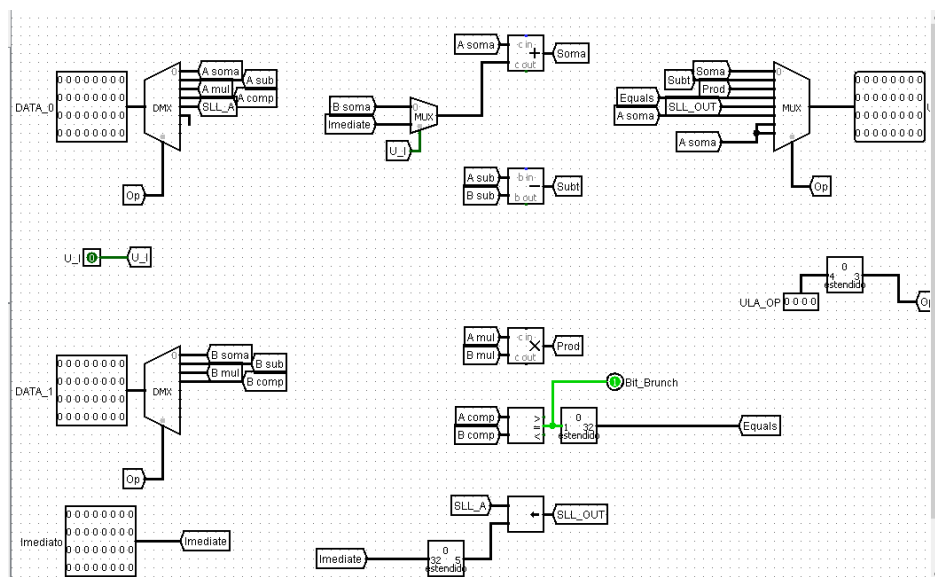


Figura 5: Estrutura final da ULA.

UNIDADE DE CONTROLE

A unidade de controle do circuito do processador foi construída com o auxílio de uma ferramenta do Logisim (“Analisar Circuito”), retornando as saídas de OPCODE e bits especiais que serão utilizados pelas demais unidades e multiplexers. Dentro dela, 4 novas operações são determinadas com suas respectivas saídas, sendo elas:

Função	OPCode	W_reg	Imd	Jump	Branch	Mem_R	Mem_W	LI
Load IM	0110	1	0	0	0	0	0	1
Load W	0111	1	1	0	0	1	0	0
Save W	1000	0	1	0	0	0	1	0
Jump	1001	0	0	1	0	0	0	0

Vale lembrar que o OPCODE nesse caso é o código de entrada, a saída pode ser diferente, como no caso do LW e SW que retornam o código 0000 para a ULA com o intuito de efetuar a soma entre o registrador e o imediato para calcular o endereço específico na memória.

Os códigos restantes das 5 operações serão expandidos ao final para demonstrar a sua saída na Unidade de Controle, apenas para não deixar aberturas sem explicação.

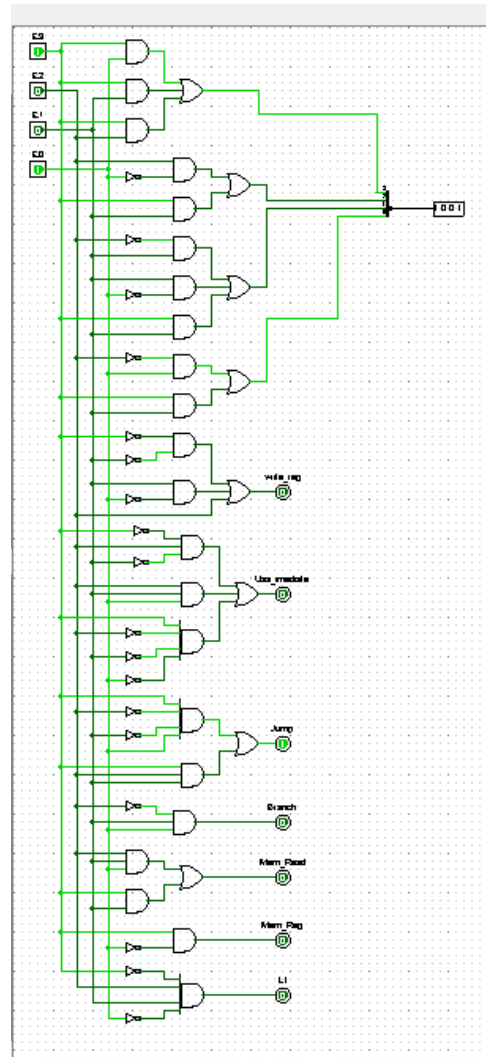


Figura 6: Estrutura final da Unidade de Controle

MEMÓRIA DE DADOS

A memória de dados foi o mais simples da execução do trabalho, já que só foi necessário o uso de uma memória RAM do próprio MIPS para guardar os dados. Que, por sua vez, recebe as entradas dos bits para leitura e gravação de dados na memória, uma saída direta da ULA para o acesso aos endereços e uma diretamente do banco de registradores para os dados vindos diretamente do mesmo. Por sua vez, o valor vindo em direção da ULA acaba por precisar ser reduzido para 8 bits por um motivo de praticidade e ser o máximo necessário para o trabalho. Convenientemente, ao permitir uma entrada de dados de até 8 bits, cada linha da aba “Editar conteúdo” da memória acaba por servir como o armazenamento completo de uma matriz 4x4.

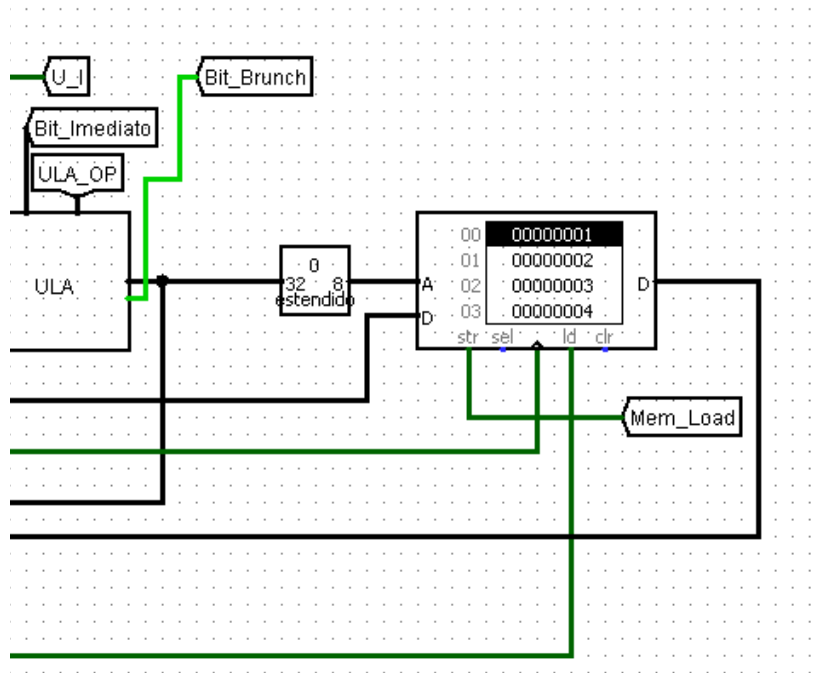


Figura 7: Memória de dados posicionada, já apresentando as matrizes carregadas.

ESTRUTURAS ADICIONAIS

Como principais estruturas adicionais existem as estruturas específicas de *Branch* e *Jump*, que usam entradas especiais dentro do segmento de instruções para determinar endereços de salto e retorno. No caso do processador implementado, ambas recebem um *shift* lógico para a esquerda de 2 bits com o intuito de adequar os endereços de instrução repassados para o padrão requerido na estrutura fornecida. Ambas funcionam essencialmente com a passagem dos endereços de instrução através de imediatos, já que não foi implementado um sistema que traduza diretamente os endereços.

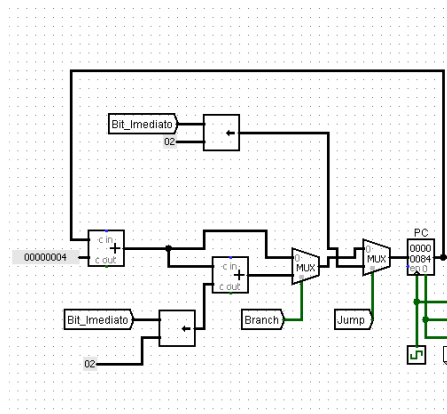


Figura 8: Estruturas adicionais de *Jump* e *Branch*.

Além disso, como estruturas adicionais para o padrão externo, podem ser citados os multiplexadores utilizados, tanto para definir a instrução de load de um valor imediato, quanto para aceitar a passagem de um valor da memória para que seja escrito em um registrador.

CÓDIGO DE MULTIPLICAÇÃO DE MATRIZES

Sem muito mais o que explicar sobre a estrutura em si, visto que é mais simples visualizar em funcionamento do que descrever, é melhor partir para o código projetado para o processador. Em primeiro lugar, é válido dizer que a estrutura de instruções descritas na documentação da avaliação foi alterada, seguindo com a mesma ordem para todas. Dito isso, a implementação do código foi, sem dúvidas, uma das partes mais complexas. Em especial pela tradução ser em hexadecimal, o ponto mais complicado de se compreender foi a movimentação através dos diferentes saltos implementados.

Esse ponto foi tão complicado que acabou sendo responsável por muitos erros, mas partindo para a implementação de vez. A ideia inicial era a construção de um programa que efetuasse a tradução diretamente, simplificando o trabalho e evitando o aparecimento de tantos erros. Ocorreu, porém, que a implementação desse modelo seria mais complexa do que simplesmente partir diretamente para escrever diretamente o código para o processador, sendo um trabalho extra. Portanto, é feita a implementação através da escrita direta. Nessa etapa, foi sugerido pelo professor tratar como se as matrizes em questão já estivessem na memória. Considerando tal, foi iniciada a estruturação do programa tendo como base os casos escritos em C e Assembly MIPS.

Inicialmente, as partes de inscrição de valores em registradores para uso posterior foram seguindo o mesmo formato. Dividindo o código em “funções” ou trechos de salto, para posteriormente montar a estrutura. A primeira parte foi definir o uso de um campo de registrador para o limite superior da memória de matrizes, sendo o valor 4. Posteriormente, a declaração das funções responsáveis por mimetizar os loops de *FOR* em C, com os seus inícios definidos de modo linear e o fim de interações através de saltos. Inicialmente foram definidos os inícios das instruções, sendo o início de cada loop definido pela instrução de comparação entre o registrador que armazena a variável de controle e o limite superior.

Após a construção dos loops, foi a vez de estruturar a multiplicação efetivamente. Ela, por sua vez, foi construída inteiramente dentro do loop de “for_de_k”. Dentro dele, dois registradores novos foram utilizados para os *Shifts* lógicos de I e K para possibilitar trabalhar com as matrizes, visto que a sua organização na memória é linear, logo as operações são necessárias para adquirir as variáveis certas pelos conceitos, seguindo a mesma lógica do que foi efetuado no trabalho anterior com o manuseio da memória no MIPS.

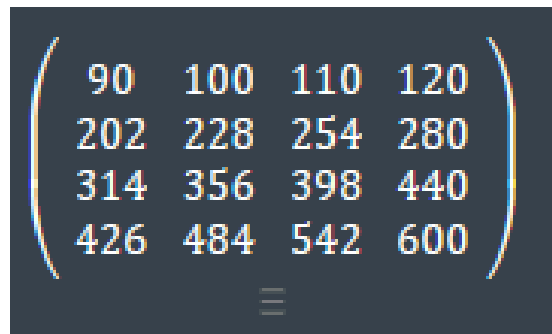
Posteriormente, os valores que passaram pelo shift são somados em registradores que carregam: $(i*4 + j)$, $(i*4 + k)$ e $(k*4 + j)$. Esses registradores serão passados na soma como resultado do cálculo das posições dos valores para escrita e coleta de dados na matriz de resultado, e apenas coleta nos últimos dois casos, respectivamente. Após isso, é definido um registrador com o valor 0, que será necessário para que seja feita a passagem do valor da memória das matrizes, e então são feitos os processos lógicos. Três instruções do tipo Load word (no meu processador) são usadas para carregar nos registradores 9, a e b, os respectivos valores localizados nas matrizes Mat0, Mat1 e MatR (contando os incrementos).

Calculam então o valor da multiplicação do termo da linha da primeira matriz pela coluna da segunda, guardando-o no registrador 9, somando com o termo que estava posteriormente no endereço que se está calculando da matriz resultante, e só então devolvendo-o para dentro da matriz resultado.

Da forma que aqui foi explicada, fica um pouco abstrato. Porém, de modo resumido, o algoritmo executa a seguinte operação para cada termo da matriz resultante:

$$c_{i,j} = \sum_{k=1}^n a_{i,k} + b_{k,j}$$

Lendo os valores na primeira matriz em $a_{i,k}$, na segunda em $b_{k,j}$ e efetua o somatório. Posteriormente adicionando o valor na posição $c_{i,j}$ da matriz resultante. As matrizes escolhidas foram idênticas para exemplificar o funcionamento. Usando os valores preenchidos de 1 até 16 em ordem crescente pelas matrizes, encontramos o seguinte resultado



$$\begin{pmatrix} 90 & 100 & 110 & 120 \\ 202 & 228 & 254 & 280 \\ 314 & 356 & 398 & 440 \\ 426 & 484 & 542 & 600 \end{pmatrix}$$

Figura 9: Resultado esperado para a multiplicação de matrizes.

```
00 00000001 00000002 00000003 00000004 00000005 00000006 00000007 00000008 00000009 0000000a 0000000b 0000000c 0000000d 0000000e 0000000f 00000010
10 00000001 00000002 00000003 00000004 00000005 00000006 00000007 00000008 00000009 0000000a 0000000b 0000000c 0000000d 0000000e 0000000f 00000010
20 0000005a 00000064 0000006e 00000078 000000ca 000000e4 000000fe 00000118 0000013a 00000164 0000018e 000001b8 000001aa 000001e4 0000021e 00000258
30 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
40 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Figura 10: Resultado da multiplicação de matrizes em hexadecimal.