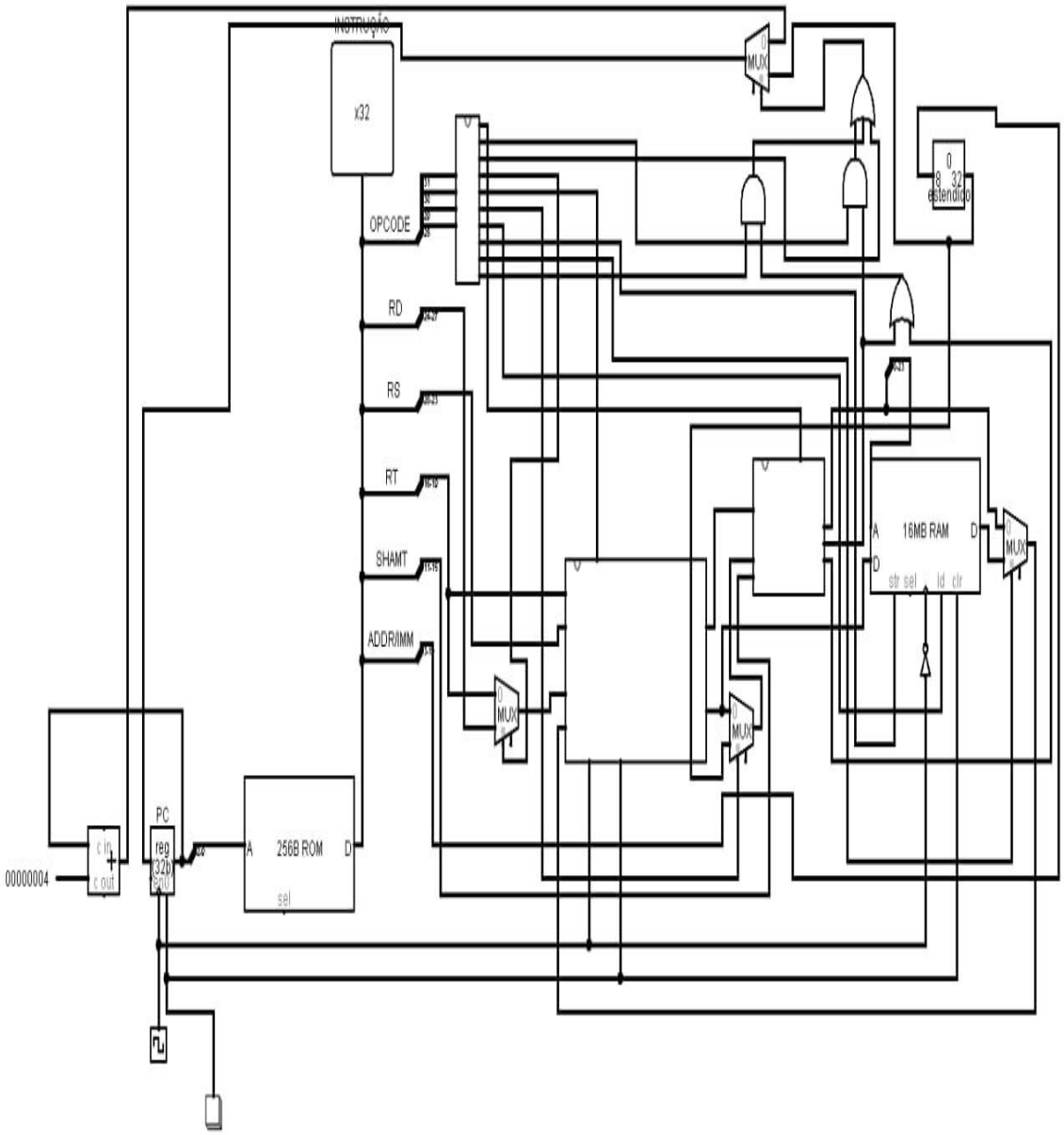
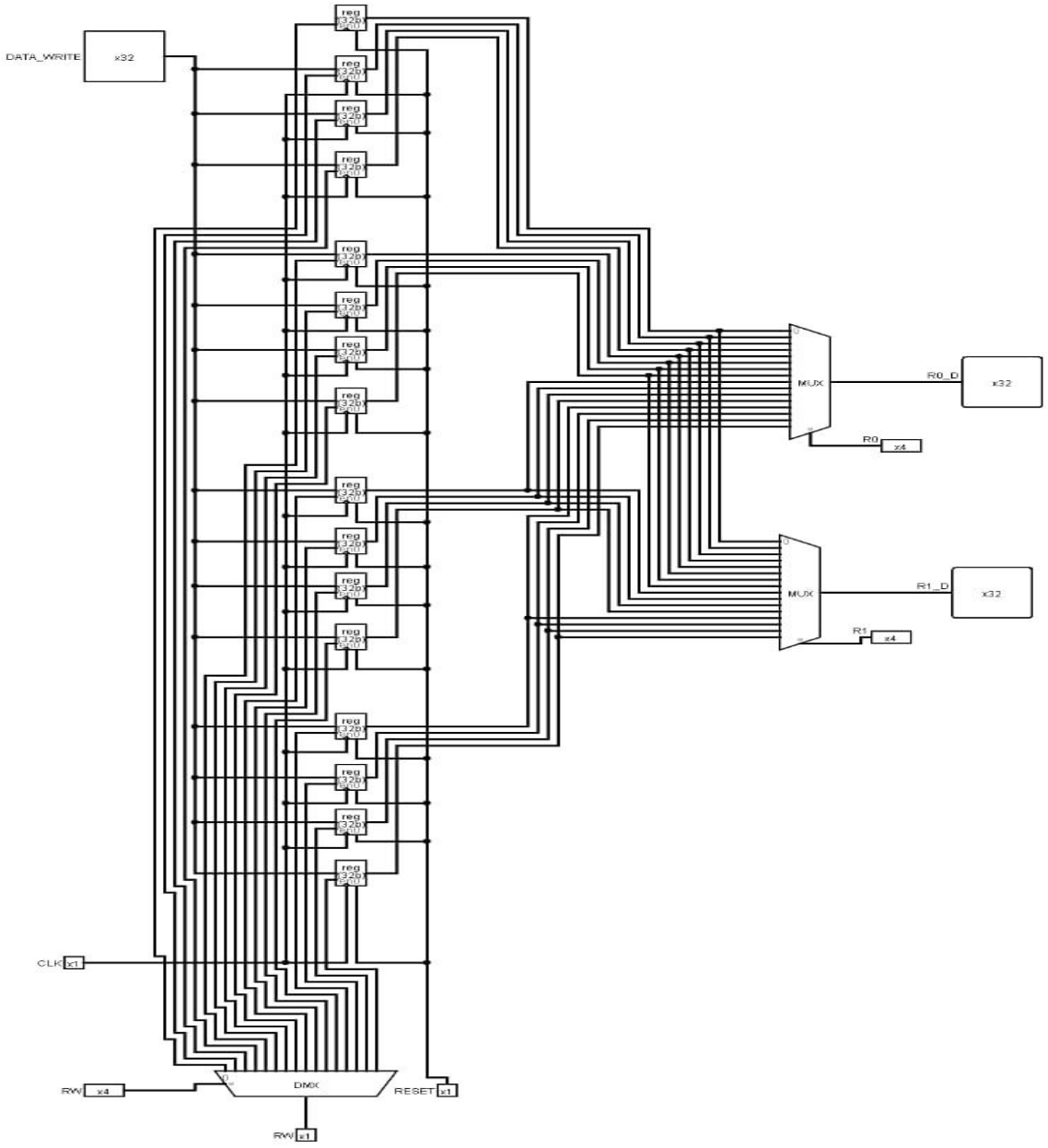
Projeto: Processador em monociclo

Dupla: Manuela Menezes Alves e Stéfani Rodrigues Neves; Prof: Ramon Napuceno; Disciplina: Arquitetura e organização de computadores.

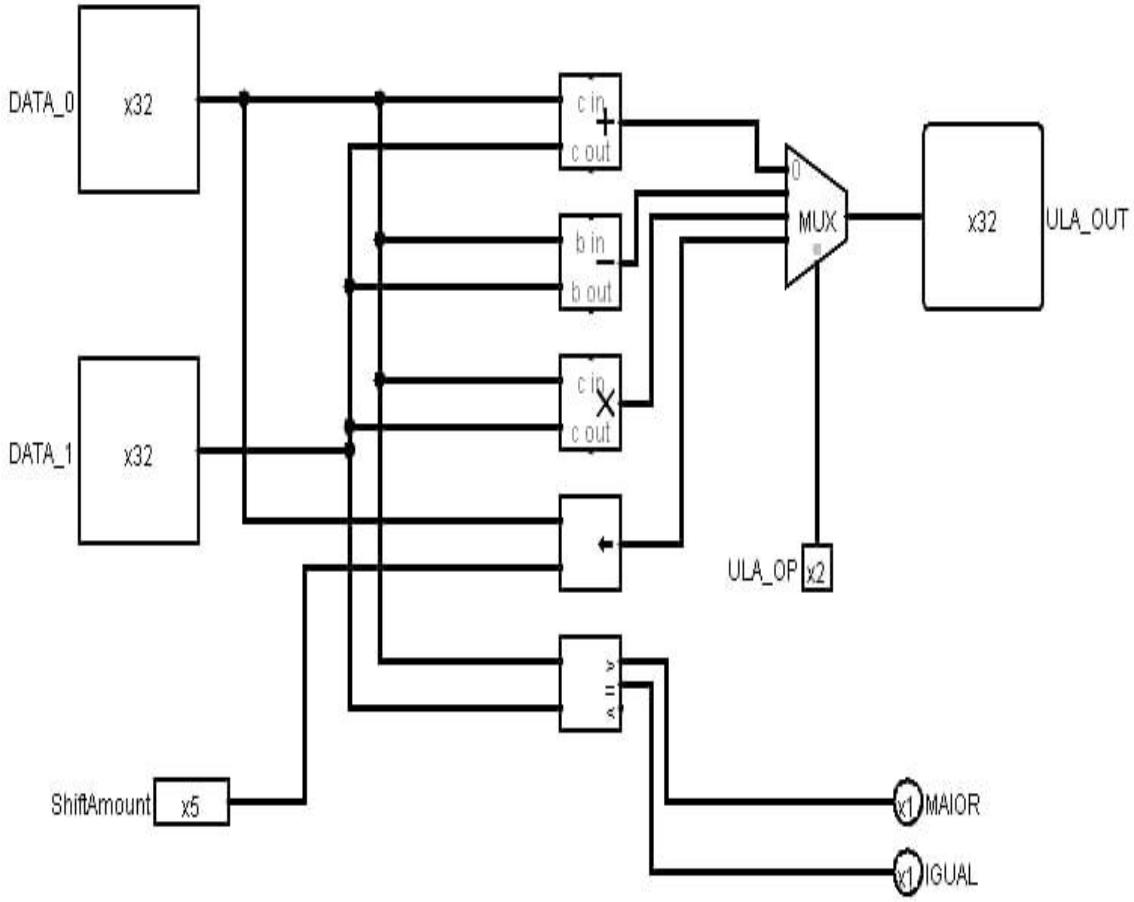
Processador(visão geral)



Inicialmente temos aqui a visão geral do nosso processador. O PC é o responsável por ler as instruções na memória de instruções, os quais o somador pega o valor atual do PC e soma mais 4 para ler a próxima instrução. O sinal de clock é responsável por controlar o ciclo, pois a cada pulso de clock uma instrução é executada, nesse sentido, temos também o reset, que volta a M.I para zero. Cada instrução tem 32 bits, onde os 4 primeiros bits é o opcode(define qual instrução). Em seguida temos o RD(registrador de destino no caso de add), RS e RT(registradores que vão entrar para realizar uma operação e RT registrador destino no caso de addi,lw e sw), o Shamt, que vai dizer quantos bits vão ser deslocados, e o endereço do imediato. A entrada regdst controla qual registrador destino vai passar(RD para instruções tipo R e RT para tipo I). Temos o banco de registradores em seguida:

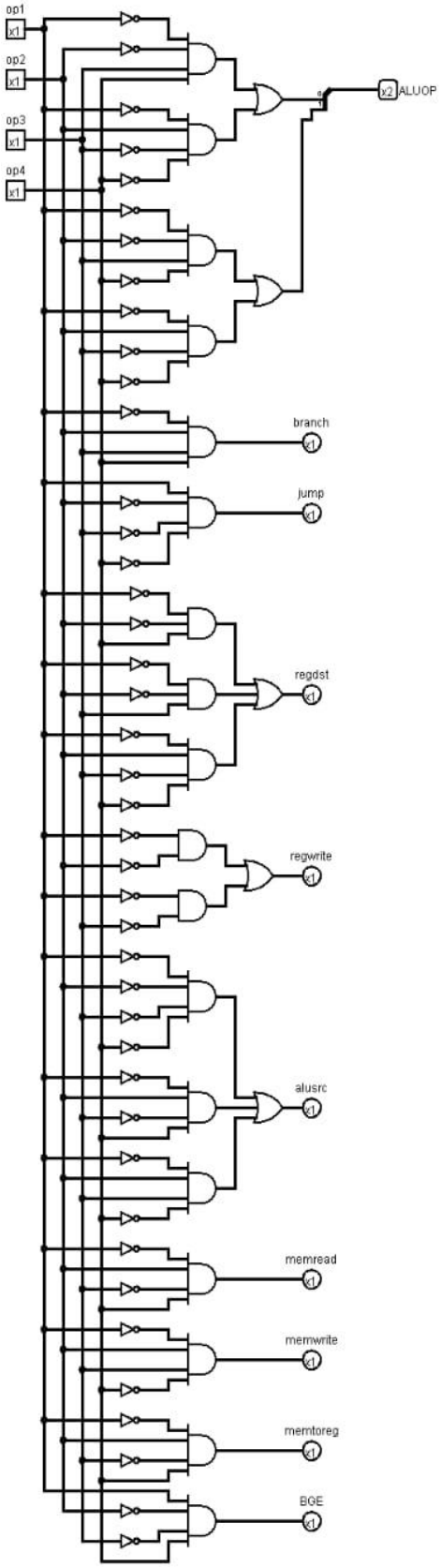


Nosso banco de registradores contém 16 registradores de 32 bits. As entradas R0 e R1 vão dizer quais deles vão passar pelo multiplexador para realizar a operação, os quais vão para as entradas da ula. O resultado dessa operação vai está no Data Write, assim, a entrada RW de 4 bits vai dizer em qual registrador esse resultado vai ser escrito(passando pelo DMX). A entrada RW de 1 bit vai habilitar essa escrita. Temos também o clock e o reset conectados a cada registrador. A entrada alusrc controla se passa o imediato ou conteúdo do registrador para a ula, instruções como Add ou Addi, Lw ou Sw. Em seguida temos nossa Ula...



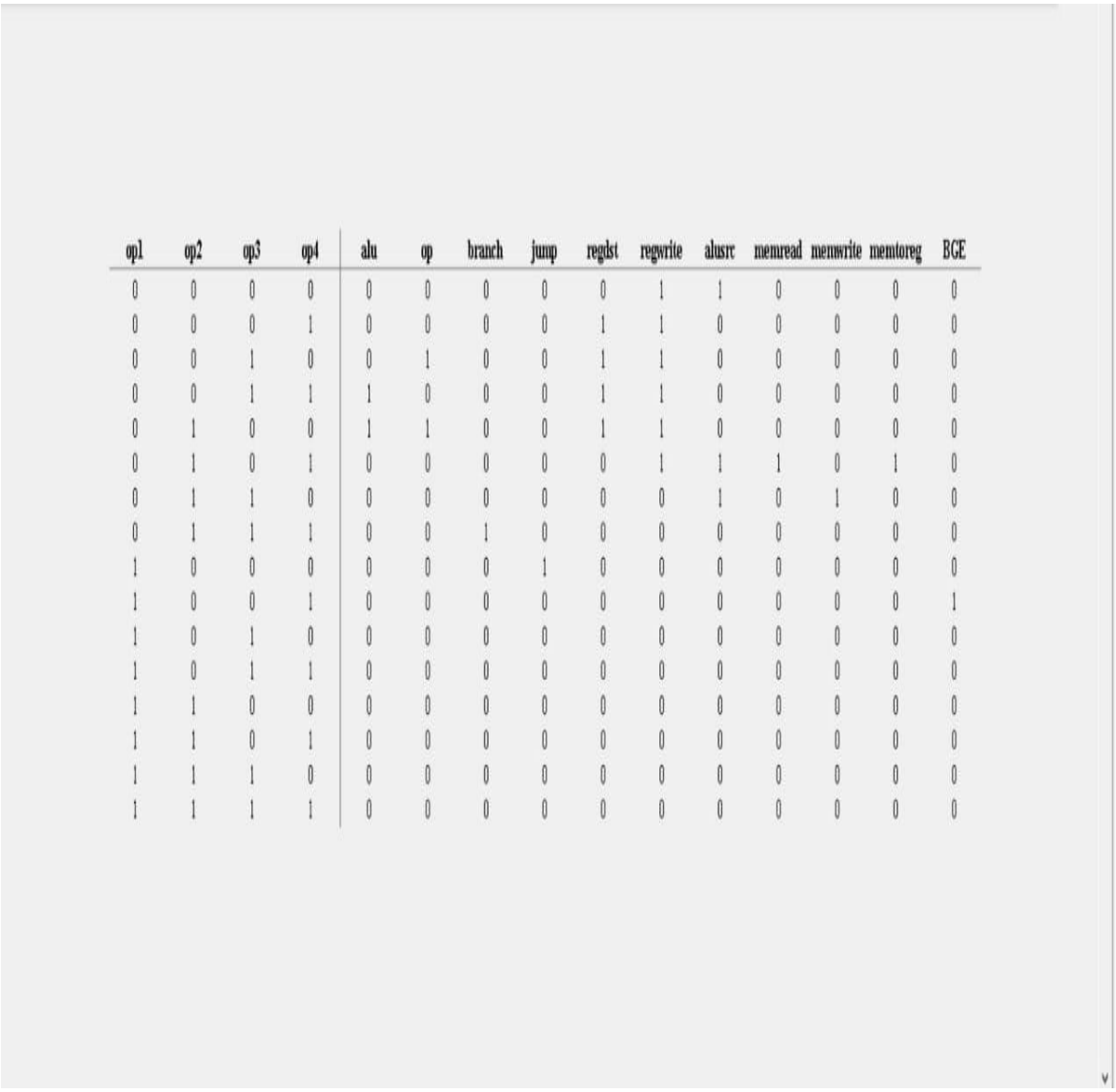
Nossa Ula executa 5 operações: soma, subtração, multiplicação, deslocamento para esquerda(Sll) e o comparador(instruções beq e bge). As entradas Data 0 e Data 1 contém o conteúdo dos registradores que foram escolhidos para ser realizada uma operação como já vimos no banco de registradores. O sinal de controle Ula op de 2 bits indica qual operação vai ser realizada. As entradas maior e igual serão utilizadas para as instruções beq e bge. O ShiftAmount desloca os bits, no qual a quantidade máxima que pode deslocar é 32, isso significa que a cada casa deslocada multiplicamos por 2. Para a instrução beq se for igual e se for um branch adicionamos uma porta and e conecatmos em uma porta or. Se for um bge, se for maior ou igual conectamos á uma porta or e depois a uma porta and, pois o sinal bge estaria habilitado na unidade de controle. Estes estarão conectados a uma porta or com três entradas, uma para o bge, beq e jump. Se for alguma dessas três instruções, o endereço vai passar diretamente para o PC, onde o multiplexador vai

escolher se passa o imediato ou o valor que vem do somador. O sinal de controle da ula é o Aluop. Em seguida temos nossa mémoria de dados, os quais executamos instruções como lw e sw. A entrada A é onde vai entrar o cálculo do endereço de mémoria (RT + Imediato). A entrada D é onde vai passar o conteúdo do registrador RT para a instrução de sw , então essa instrução vai escrever na mémoria o conteúdo de RT, que o sinal de controle memwrite habilita, nesse caso, o clock é invertido para ter esse tempo entre o cálculo do endereço onde quer escrever o conteúdo de RT e a escrita de fato. Para a instrução lw, vai ser calculado um endereço de memória onde irá pegar esse conteúdo e escrever em RT, os quais o sinal de controle memread habilita essa leitura do valor. O sinal memtoreg controla se passa um valor da memória ou conteúdo do registrador para ser escrito em um registrador destino, no caso de lw, vai passar esse valor da mémoria e ser escrito em RT. Por fim temos a unidade de controle...



Nossa unidade de controle é a responsável por controlar o processador, pois cada sinal da uc(que já explicamos ao longo do relatório) tem sua função dentro dele em prol de

conseguir realizar as instruções nele. È necessário uma tabela verdade que vai definir as saídas(sinais de controle) para as entradas do opcode, dessa forma, ao definir a tabela o próprio simulador logisim constrói o circuito da uc. Em seguida temos a tabela verdade da uc...



op1	op2	op3	op4	alu	op	branch	jump	regdst	regwrite	alusrc	memread	memwrite	memtoreg	BGE
0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	1	0	0	0	0	1	1	0	0	0	0	0
0	0	1	0	0	1	0	0	1	1	0	0	0	0	0
0	0	1	1	1	0	0	0	1	1	0	0	0	0	0
0	1	0	0	1	1	0	0	1	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	1	1	0	1	0
0	1	1	0	0	0	0	0	0	0	1	0	1	0	0
0	1	1	1	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	1
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

Você deve está se perguntando, afinal o quê esse processador está fazendo?

Bom, a mémoria de instruções é para ler um algoritmo de multiplicação de matrizes, nos quais nele precisa das instruções addi, add, sub, mul, sll, lw, sw, beq, jump e bge, por isso nosso processador está organizado dessa maneira. Foi necessário primeiro a gente passar o código em C para assembly, depois de assembly para hexadecimal, pois a mémoria de instruções ler elas em hexadecimal. Em seguida está os códigos nessas duas linguagens...

Em Assembly:

```
.globl main
.text

main:
addi $sp, $sp, -64
move $a1, $sp
li $s1, 1
li $s0, 0
for_i:
li $t1, 4
bge $s0, $t1, fim_for_i
li $s4, 0
for_k:
li $t5, 4
bge $s4, $t5, fim_for_k
sll $t6, $s0, 2
add $t7, $t6, $s4
add $s5, $t7, $a1
sw $s1, 0($s5)
addi $s4, $s4, 1
j for_k
fim_for_k:
addi $s0, $s0, 1
j for_i
fim_for_i:
addi $sp, $sp, -64
move $a2, $sp
li $s0, 0
fork:
li $t1, 4
bge $s0, $t1, fim_fork
li $s4, 0
for_j:
```

```
li $t5, 4
bge $s4, $t5, fim_for_j
sll $t6, $s0, 2
add $t7, $t6, $s4
add $s6, $t7, $a2
sw $s1, 0($s6)
addi $s4, $s4, 1
j for_j
fim_for_j:
addi $s0, $s0, 1
j fork
fim_fork:
addi $sp, $sp, -64
move $a0, $sp
li $s0, 0

for_i_2:
li $t1, 4
bge $s0, $t1, fim_fori2
li $s4, 0
for_j_2:
li $t5, 4
bge $s4, $t5, fim_forj2
sll $t6, $s0, 2
add $t7, $t6, $s4
add $s3, $t7, $a0
sw $zero, 0($s3)
li $t2, 0
for_k_2:
li $t5, 4
bge $t2, $t5, fim_fork2
sll $t6, $s0, 2
add $t7, $t6, $s4
```

Em hexadecimal

```
v2.0 raw
000E0010 000B0C00 000C0C00 000D0C00 00010000 00020040 90120250 00030000 90320210 44105000 4F302000 144F0000 144C0000
60E40000 00330010 80000110 00110010 800000b0 00010000 00020040 90120410 00010000 903203d0 44105000 4F302000 144F0000
144D0000 60E40040 00330010 800002d0 00110010 80000270 00010000 00020040 909407d0 00030000 90320790 44105000 4F302000
144F0000 144B0000 60040080 50A40080 00050000 90520750 46105000 4F502000 166F0000 166C0000 50760000 48505000 4F302000
148F0000 188D0000 50980040 39970000 19A90000 60940080 00550010 80000550 00330010 80000450 00110010 800003f0
```

Bom, chegamos ao fim. Gratidão professor Ramon!!...

“È superando desafios que ultrapassamos nossos limites”

-Pensador.