Relatório Sagui em Bando

Andre Grassi de Jesus
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
agj23@inf.ufpr.br

I. Introdução

Esse relatório consiste na exposição da implementação da arquitetura Sagui em Bando (a qual consiste em uma *vector architecture*) bem como uma breve explicação à cerca dos seus componentes. Além disso serão apresentados dois programas de teste desenvolvidos para essa arquitetura.

O projeto foi criado utilizando o software de simulação de circuitos digitais *Logisim Evolution*.

A nomenclatura de entradas e a ideia geral do projeto foi baseada na arquitetura *RISC-V* apresentada no livro *Computer Organization and Design: the Hardware/Software Interface* [1].

II. Sinais de Controle

Os sinais de controle, gerados pelo circuito Ctrl, são os seguintes:

- **AluOp** (3 bits): indica qual operação deve ser realizada nas ULAs.
- **RegWrV** (1 bit): indica se a escrita no banco de registradores dos VPEs deve ser habilitada ou não.
- **RegWrS** (1 bit): indica se a escrita no banco de registradores do SPE deve ser habilitada ou não.
- **WbOp** (2 bits): indica o que deve ser escrito no banco de registradores na etapa de *write back* (0: saída da ULA, 1: saída da RAM)
- **Branch** (1 bit): indica se a instrução é uma Branch (1) ou não (0).
- **SelectR1** (1 bit): indica se deve ser selecionado obrigatoriamente o R[1] em uma das entradas do banco.
- **UseImm** (1 bit): indica o uso de imediato.
- **RamWrV** (1 bit): indica se habilita escrita na RAM dos VPEs.
- **RamWrS** (1 bit): indica se habilita escrita na RAM do SPE.

$Opcode[3..0]$				$AluOp[2..0]$			$RegWrV$	$RegWrS$	$WbOp$	$Branch$	$SelectR1$	$UseImm$	$RamWrV$	$RamWrS$
0	0	0	0	1	0	0	0	1	1	0	0	0	0	0
0	0	0	1	1	0	1	0	0	1	0	0	0	0	1
0	0	1	0	1	0	0	0	1	0	0	1	1	0	0
0	0	1	1	1	0	1	0	1	0	0	1	1	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	0	1	0	0	1	0	1	0	0	0	0	0	0
0	1	1	0	0	1	0	0	1	0	0	0	0	0	0
0	1	1	1	0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	1	0	0	1	0	1	0	0	0	0	0
1	0	0	1	1	0	1	0	0	1	0	0	0	1	0
1	0	1	0	1	0	0	1	0	0	0	1	1	0	0
1	0	1	1	1	0	1	1	0	0	0	1	1	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0
1	1	0	1	0	0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	1	0	1	0	0	0	0	0	0	0
1	1	1	1	0	1	1	1	0	0	0	0	0	0	0

III. ULA

A ULA recebe três entradas, que são os **operandos** (Operand1 e Operand2) e o sinal de controle **ALUOp** (ALU Operation) que escolhe qual operação será propagada à saída **Result**. As operações são as seguintes:

0) **Add**: soma operando 1 e 2.
1) **Sub**: subtrai o operando 2 do 1.
2) **And**: and entre operandos.
3) **Or**: or entre operandos.
4) **Movh**: concatena os 4 LSB (*Least significat bits*) do operando 2 com os 4 LSB do operando 1, colocando os bits do operando 2 na parte alta.
5) **Movl**: concatena, colocando os 4 LSB do operando 2 na parte baixa.

IV. Programas de Teste

Para desenvolvimento dos programas de teste foi utilizado um ”pseudo-assembly”, semelhante ao RISC-V, mas com as instruções do Sagui.

A. Teste das Instruções

Nesse programa foi realizado um teste de todas as 16 instruções da arquitetura. O assembly é o seguinte:

```
#       "Pseudo-assembly" da arquitetura Sagui em Bando           #
#      ##############################################             #
#                                                                 #
#          Organização dos comentários dos comandos:              #
#                                                                 #
# mnem. # sign. da operação --> # instr. em bin. = instr. em hex. #


######## Teste das Operações Escalarares ########

## MOV ##

# Inicializa registrador 1 com 00000000
# Como em uma cpu real, teríamos lixo de memória nos registradores,
# o que é representado por "?"
movh 0000   # R[1] = {Imm + R[1](3:0)} = {0000 + ????} = 0000???? = ? --> 00100000 = 20
movl 0000   # R[1] = {R[1](7:4) + Imm.} = {0000 + 0000} = 00000000 = 0 --> 00110000 = 30

# Coloca o valor 112 no R[1]
movh 0111   # R[1] = {Imm + R[1](3:0)} = {0111 + 0000} = 01110000 = 112 = 70 (hex) --> 00100111 = 27

# Reseta R[1]
movh 0000   # R[1] = {Imm + R[1](3:0)} = {0000 + 0000} = 00000000 = 0 --> 00100000 = 20

# Coloca o valor 7 no R[1]
movl 0111   # R[1] = {R[1](7:4) + Imm.} = {0000 + 0111} = 00000111 = 7 -->00110111 = 37


## STORE ##

# Guarda o valor do R[1] na memória na posição 0
st $1, $0 # M[R[0]] = R[1] <=> M[0] = 7 --> 00010100 = 14


## LOAD ##

# Carrega o valor da memória na posição 0 para o R[2]
ld $2, $0 # R[2] = M[R[0]] <=> R[2] = M[0] --> 00001000 = 08

# Carrega o valor da memória na posição 0 para o R[3]
ld $3, $0 # R[3] = M[R[0]] <=> R[3] = M[0] --> 00001100 = 0C


## ADD ##

# Soma R[2] com R[3]
add $2, $3 # R[2] = R[2] + R[3] = 7 + 7 = 14 --> 01001011 = 4B


## SUB ##

# Subtrai R[3] de R[2]
sub $2, $3 # R[2] = R[2] - R[3] = 14 - 7 = 7 --> 01011011 = 5B


## AND ##

# And de R[2] com R[3]
and $2, $3 # R[2] = R[2] & R[3] = 7 & 7 = 7 --> 01101011 = 6B


# BRZR #

# Coloca o valor 16 = 10 (hex) no R[1]
movl 0000  # R[1] = {R[1](7:4) + Imm.} = {0000 + 0000} = 00000000 = 0 --> 00110000 = 30
movh 0001  # R[1] = {Imm + R[1](3:0)} = {0001 + 0000} = 00010000 = 16 = 10 (hex) --> 00100001 = 21

# Pula para o endereço 10 (hex)
brzr $0, $1 # if (R[0] == 0) PC = R[1] = 160 = A0 (hex) --> 01110001 = 71


######## Teste das Operações Vetoriais ########

## MOV ##

# Inicializa registradores 1 com 00000000
movh 0000   # R[1] = {Imm + R[1](3:0)} = {0000 + ????} = 0000???? = ? --> 10100000 = A0
movl 0000   # R[1] = {R[1](7:4) + Imm.} = {0000 + 0000} = 00000000 = 0 --> 10110000 = B0

# Coloca o valor 112 nos R[1]
movh 0111   # R[1] = {Imm + R[1](3:0)} = {0111 + 0000} = 01110000 = 112 = 70 (hex) --> 10100111 = A7

# Reseta os R[1]
```

```
movh 0000   # R[1] = {Imm + R[1](3:0)} = {0000 + 0000} = 00000000 = 0 --> 10100000 = A0

# Coloca o valor 7 nos R[1]
movl 0111   # R[1] = {R[1](7:4) + Imm.} = {0000 + 0111} = 00000111 = 7 --> 10110111 = B7


## STORE ##

# Guarda os valores dos R[1] nas memórias na posição de 0 a 3
# pois o R[0] dos vetoriais varia de 0 a 3
st $1, $0 # M[R[0]] = R[1] <=> M[0] = 7 --> 10010100 = 94


## LOAD ##

# Carrega os valores das memórias na posição 0 para os R[2]
ld $2, $0 # R[2] = M[R[0]] <=> R[2] = M[0] --> 10001000 = 88

# Carrega os valores das memórias na posição 0 para os R[3]
ld $3, $0 # R[3] = M[R[0]] <=> R[3] = M[0] --> 10001100 = 8C


## ADD ##

# Soma os R[2] com os R[3]
add $2, $3 # R[2] = R[2] + R[3] = 7 + 7 = 14 --> 11001011 = CB


## SUB ##

# Subtrai os R[3] de R[2]
sub $2, $3 # R[2] = R[2] - R[3] = 14 - 7 = 7 --> 11011011 = DB


## AND ##

# And dos R[2] com os R[3]
and $2, $3 # R[2] = R[2] & R[3] = 7 & 7 = 7 --> 11101011 = EB

## OR ##

# Or dos R[2] com os R[3]
or $2, $3 # R[2] = R[2] | R[3] = 7 | 7 = 7 --> 11111011 = FB
```

B. Soma de Vetores

Nesse programa são inicializados três vetores de tamanho 10 (A, B e R), os vetores A e B são somados e o resultado é guardado no R.

A recebe [0, 2, 4, 6, 8, 10, 12, 14, 16, 18].

B recebe [1, 3, 5, 7, 9, 11, 13, 15, 17, 19].

R recebe a soma [1, 5, 9, 13, 17, 21, 25, 29, 33, 37].

Para melhor elaboração do assembly, foi desenvolvido um código simplificado em C:

```c
#include <stdio.h>

int main () {
    // Declaração dos vetores
    int vet_a[10];
    int vet_b[10];
    int vet_r[10];

    // Declaração das variáveis de controle
    int i;
    int j;

    // Loops que inicializam os vetores
    i = 0;
    j = 0;

inicializa_a:
    if (i < 10) {
        vet_a[i] = j;

        j = j + 2;
        i = i + 1;

        goto inicializa_a;
    }

    i = 0;
    j = 1;
```

```
inicializa_b:
    if (i < 10) {
        vet_b[i] = j;

        j = j + 2;
        i = i + 1;

        goto inicializa_b;
    }

    i = 0;

inicializa_r:
    if (i < 10) {
        vet_r[i] = 0;

        i = i + 1;

        goto inicializa_r;
    }

    // Loop que realiza a operação de soma
    i = 0;

soma:
    if (i < 10) {
        vet_r[i] = vet_a[i] + vet_b[i];
        i = i + 1;

        goto soma;
    }

    // Loop que imprime o vetor resultante
    i = 0;
    while (i < 10) {
        printf ("%d\n", vet_r[i]);
        i = i + 1;
    }
}
```

Já o código transformado no assembly é o seguinte:

```
## PROGRAMA DE SOMA ##

# Inicializa os VR com 0 para resetar seus valores

# Inicializa VR[1] com 0
movh 0000 # 10100000 = A0
movl 0000 # 10110000 = B0

# Inicializa VR[2] e VR[3] com 0
and $2, $1 # 11101001 = E9
and $3, $1 # 11101101 = ED

# Guarda no VR[1] o endereço em que será guardado o valor da
# variável índice, que indica qual o salto de endereço que deve ser
# realizado para guardar os dados no local certo do vetor na RAM

# Esse salto é de 4 em 4 posições, pois cada iteração guarda 4 valores
# sendo que cada VPE vai acessar um endereço diferente da RAM, de forma
# sequencial

# Por exemplo, na primeira iteração o índice vai valer 0
# logo, o VPE 0 vai guardar o valor na posição 0
# o VPE 1 vai guardar o valor na posição 1, etc.
# Já na segunda iteração o índice vai valer 4
# logo, o VPE 0 vai guardar o valor na posição 4
# o VPE 1 vai guardar o valor na posição 5, etc.

# Endereço 30 (hex) = 48 (dec)
movl 0000 # 10110000 = B0
movh 0011 # 10100011 = A3

# Guarda no endereço 0 o endereço 30
# Essa instrução serve apenas para transferir o
# endereço para o VR[2] na próxima instrução
st $1, $2 # 10010110 = 96

# Guarda no VR[2] o endereço do índice
ld $2, $2 # 10001010 = 8A

# Inicializa o valor do índice com 0 nos VPE
and $1, $3 # 11100111 = E7
```

```
# Guarda no endereço 30 de cada RAM o valor do índice
st $1, $2 # 10010110 = 96

# Inicializa SR[1], SR[2] e SR[3] com 0
and $1, $0 # 01100100 = 64
and $2, $0 # 01101000 = 68
and $3, $0 # 01101100 = 6C

# Coloca 1 no SR[1]
movl 0001 # 00110001 = 31

# Guarda no endereço 0 o valor 1
st $1, $0 # 00010100 = 14

# Coloca 1 no SR[2]
# Esse 1 vai servir para decrementar de 1 em 1 o contador do laço a cada
# iteração
ld $2, $0 # 00001000 = 08

# Aqui é definido o valor do contador, que controla quantas vezes o laço de
# inicialização do vetor será executado.
# Coloca o valor 3 (para controlar o laço) no SR[1] e depois passa para o SR[3]
movl 0011 # 00110011 = 33

# Guarda no endereço 0 o valor 3
st $1, $0 # 00010100 = 14

# Carrega 3 no SR[3]
ld $3, $0 # 00001100 = 0C

# Carrega o valor do índice nos VR
movl 0000 # 10110000 = B0
movh 0011 # 10100011 = A3
ld $3, $1 # 10001101 = 8D

# Pega posição em que será guardado o valor em cada VPE
# Como aqui o índice ainda vale 0, temos o seguinte:
# No VPE 0, o endereço vai ser 0
# VPE 1 vai ser 1
# VPE 2 vai ser 2...
add $3, $0 # 11001100 = CC

# Coloca 0 no VR[2]
movl 0000 # 10110000 = B0
movh 0000 # 10100000 = A0
and $2, $1 # 11101001 = E9

# Pega os 4 primeiros valores que vão ser armazenados no vetor
# Soma-se duas vezes com o VR[0] para obter os números pares
# Assim, temos:
# Significado das operações: $2 + $0 = $2 --> $2 + $0 = $2
# VPE 0: 0 + 0 = 0 --> 0 + 0 = 0
# VPE 1: 0 + 1 = 1 --> 1 + 1 = 2
# VPE 2: 0 + 2 = 2 --> 2 + 2 = 4
# VPE 3: 0 + 3 = 3 --> 3 + 3 = 6
add $2, $0 # 11001000 = C8
add $2, $0 # 11001000 = C8

# Início do laço de inicialização do vetor A

# Pega endereço do fim do laço
movl 1010 # 00111010 = 3a
movh 0100 # 00100100 = 24

# Se o valor da variável de controle (contador) for 0, pula para o fim do laço
brzr $3, $1 # 01111101 = 7D

# Subtrai 1 do SR[3] (variável de controle do laço)
sub $3, $2 # 01011110 = 5E

# Armazena o valor ($2) no vetor na posição indicada pelo índice ($3)
st $2, $3 # 10011011 = 9B

# Coloca o valor 4 no VR[1]
movl 0100 # 10110100 = B4
movh 0000 # 10100000 = A0

# Pega próxima posição do vetor
# Aqui soma-se 4 pois cada iteração armazena 4 valores simultaneamente
# Logo é necessário pular 4 posições para armazenar os próximos valores
add $3, $1 # 11001101 = CD
```

```
# Pega o próximo valor do vetor

# Há um padrão nos valores, a cada 4 posições, o valor é incrementado em 8
# Por exemplo na posição 0 temos 0, já na 4 temos 8
# Na 1 temos 2, já na 5 temos 10, e assim por diante

# Coloca 8 no VR[1]
movl 1000 # 10111000 = B8
movh 0000 # 10100000 = A0

# Soma 8 ao VR[2] (pega próximo valor)
add $2, $1 # 11001001 = C9

# Pega valor do jump = 3b (hex)
movh 0011 # 00100011 = 23
movl 1100 # 00111100 = 3C

brzr $0, $1 # 01110001 = 71

# Tudo certo até aqui

# Inicialização do B

# Coloca o valor 3 (para controlar o laço) no SR[1] e depois passa para o SR[3]
movh 0000 # 00100000 = 20
movl 0011 # 00110011 = 33

# Guarda no endereço 0 o valor 3
st $1, $0 # 00010100 = 14

# Coloca 3 no SR[3]
ld $3, $0 # 00001100 = 0C

# Carrega o valor do índice nos VR
movl 0000 # 10110000 = B0
movh 0011 # 10100011 = A3
ld $3, $1 # 10001101 = 8D

# Pega posição em que será guardado o valor
# Vetor inicia no endereço 0a
movl 1010 # 10111010 = BA
movh 0000 # 10100000 = A0
add $3, $1 # 11001101 = CD
add $3, $0 # 11001100 = CC

# Coloca 0 no VR[2]
movl 0000 # 10110000 = B0
movh 0000 # 10100000 = A0
and $2, $1 # 11101001 = E9

# Coloca 1 no VR[2]
movl 0001 # 10110001 = B1
movh 0000 # 10100000 = A0
add $2, $1 # 11001001 = C9

# Pega os 4 primeiros valores que vão ser armazenados no vetor
# Soma-se duas vezes com o VR[0] + 1 para obter os números ímpares
# Assim, temos:
# Significado das operações: $2 + $1 + $0 = $2 --> $2 + $0 = $2
# VPE 0: 0 + 1 + 0 = 1 --> 1 + 0 = 1
# VPE 1: 0 + 1 + 1 = 2 --> 2 + 1 = 3
# VPE 2: 0 + 1 + 2 = 3 --> 3 + 2 = 5
# VPE 3: 0 + 1 + 3 = 4 --> 4 + 3 = 7

add $2, $0 # 11001000 = C8
add $2, $0 # 11001000 = C8

# Início do laço de inicialização do vetor B

# Pega endereço do fim do laço = 7c (hex)
movl 1100 # 00111100 = 3C
movh 0111 # 00100111 = 27

# Se o valor da variável de controle for 0, pula para o fim do laço
brzr $3, $1 # 01111101 = 7D

# Subtrai 1 do SR[3] (variável de controle do laço)
sub $3, $2 # 01011110 = 5E

# Armazena o valor no vetor na posição indicada pelo índice
st $2, $3 # 10011011 = 9B

# Coloca o valor 4 no VR[1]
```

```
movl 0100 # 10110100 = B4
movh 0000 # 10100000 = A0

# Pega próxima posição em que será guardado o valor
add $3, $1 # 11001101 = CD

# Coloca 8 no VR[1]
movl 1000 # 10111000 = B8
movh 0000 # 10100000 = A0

# Soma 8 ao VR[2]
add $2, $1 # 11001001 = C9

# Pega valor do jump = 6e (hex)
movh 0110 # 00100110 = 26
movl 1110 # 00111110 = 3E

brzr $0, $1 # 01110001 = 71


# Inicialização do R com 0

# Coloca o valor 3 (para controlar o laço) no SR[1] e depois passa para o SR[3]
movh 0000 # 00100000 = 20
movl 0011 # 00110011 = 33

# Guarda no endereço 0 o valor 3
st $1, $0 # 00010100 = 14

# Coloca 3 no SR[3]
ld $3, $0 # 00001100 = 0C

# Carrega índice nos VPE
movl 0000 # 10110000 = B0
movh 0011 # 10100011 = A3
ld $3, $1 # 10001101 = 8D

# Pega posição em que será guardado o valor
# Vetor inicia no endereço 14
movl 0100 # 10110100 = B4
movh 0001 # 10100001 = A1
add $3, $1 # 11001101 = CD
add $3, $0 # 11001100 = CC

# Coloca 0 no VR[2]
movl 0000 # 10110000 = B0
movh 0000 # 10100000 = A0
and $2, $1 # 11101001 = E9

# Início do laço de inicialização do vetor R

# Pega endereço do fim do laço A2 (hex)
movl 0010 # 00110010 = 32
movh 1010 # 00101010 = 2A

# Se o valor do índice for 0, pula para o fim do laço
brzr $3, $1 # 01111101 = 7D

# Subtrai 1 do SR[3] (variável de controle do laço)
sub $3, $2 # 01011110 = 5E

# Armazena o valor no vetor na posição indicada pelo índice
st $2, $3 # 10011011 = 9B

# Coloca o valor 4 no VR[1]
movl 0100 # 10110100 = B4
movh 0000 # 10100000 = A0

# Pega próxima posição em que será guardado o valor
add $3, $1 # 11001101 = CD

# Pega valor do jump = 97 (hex)
movh 1001 # 00101001 = 29
movl 0111 # 00110111 = 37

brzr $0, $1 # 01110001 = 71


# Soma de A e B, guardando no vetor R

# Coloca o valor 3 (para controlar o laço) no SR[1] e depois passa para o SR[3]
movh 0000 # 00100000 = 20
movl 0011 # 00110011 = 33
```

```
# Guarda no endereço 0 o valor 3
st $1, $0 # 00010100 = 14

# Coloca 3 no SR[3]
ld $3, $0 # 00001100 = 0C

# Carrega o valor do índice nos VPE
movl 0000 # 10110000 = B0
movh 0011 # 10100011 = A3
ld $3, $1 # 10001101 = 8D

# Pega próxima posição em que será guardado o valor
# Vetor inicia no endereço 14
movl 0100 # 10110100 = B4
movh 0001 # 10100001 = A1
add $3, $1 # 11001101 = CD
add $3, $0 # 11001100 = CC

# Coloca 0 no VR[2] e VR[1]
movl 0000 # 10110000 = B0
movh 0000 # 10100000 = A0
and $2, $1 # 11101001 = E9

# Início do laço de soma

# Na soma, o registrador que vai ser o índice é o $2
# E na hora de calcular o resultado, $2 guarda o valor do vetor B
# e $3 guarda o valor do vetor A

# Pega endereço do fim do laço d7
movl 0111 # 00110111 = 37
movh 1101 # 00101101 = 2D

# Se o valor da variável de controle for 0, pula para o fim do laço
brzr $3, $1 # 01111101 = 7D

# Subtrai 1 do SR[3] (variável de controle do laço)
sub $3, $2 # 01011110 = 5E

# Pega endereço do vetor A
movl 0000 # 10110000 = B0
movh 0000 # 10100000 = A0

# Avança para posição atual
add $1, $2 # 11000110 = C6
# Avança o ponteiro de acordo com o VPE
add $1, $0 # 11000100 = C4

# Carrega o valor do A no VR[3]
ld $3, $1 # 10001101 = 8D

# Guarda índice acessado nos vetores no endereço 20 (hex)
movl 0000 # 10110000 = B0
movh 0010 # 10100010 = A2
st $2, $1 # 10011001 = 99

# Pega endereço do vetor B
movl 1010 # 10111010 = BA
movh 0000 # 10100000 = A0

# Avança para posição atual
add $1, $2 # 11000110 = C6
# Avança o ponteiro de acordo com o VPE
add $1, $0 # 11000100 = C4

# Carrega o valor do B no VR[2]
ld $2, $1 # 10001001 = 89

# Faz a soma
add $3, $2 # 11001110 = CE

# Carrega de volta ao VR[2] o índice, que está guaradado no endereço 20 (hex)
movl 0000 # 10110000 = B0
movh 0010 # 10100010 = A2
ld $2, $1 # 10001001 = 89

# Pega endereço do vetor R
movl 0100 # 10110100 = B4
movh 0001 # 10100001 = A1

# Avança para posição atual
add $1, $2 # 11000110 = C6
```

```
# Avança o ponteiro de acordo com o VPE
add $1, $0 # 11000100 = C4

# Guarda o valor da soma
st $3, $1 # 10011101 = 9D

# Coloca 4 no VR[1]
movl 0100 # 10110100 = B4
movh 0000 # 10100000 = A0

# Avança o índice
# Soma 4 com o VR[2]
add $2, $1 # 11001001 = C9

# Pega valor do jump = B7 (hex)
movh 1011 # 00101011 = 2B
movl 0111 # 00110111 = 37

brzr $0, $1 # 01110001 = 71
```

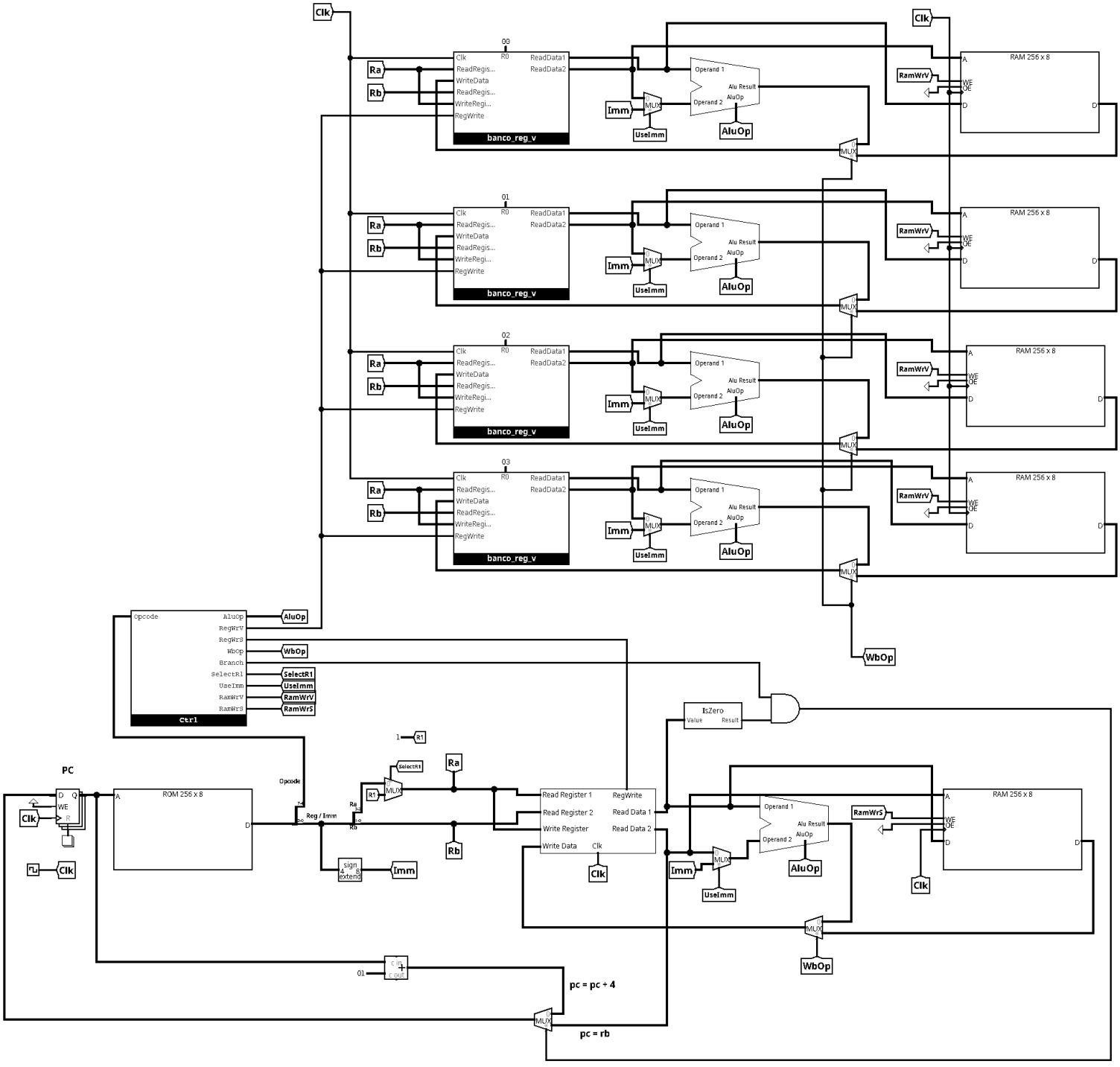

Figura 1. Imagem do processador.

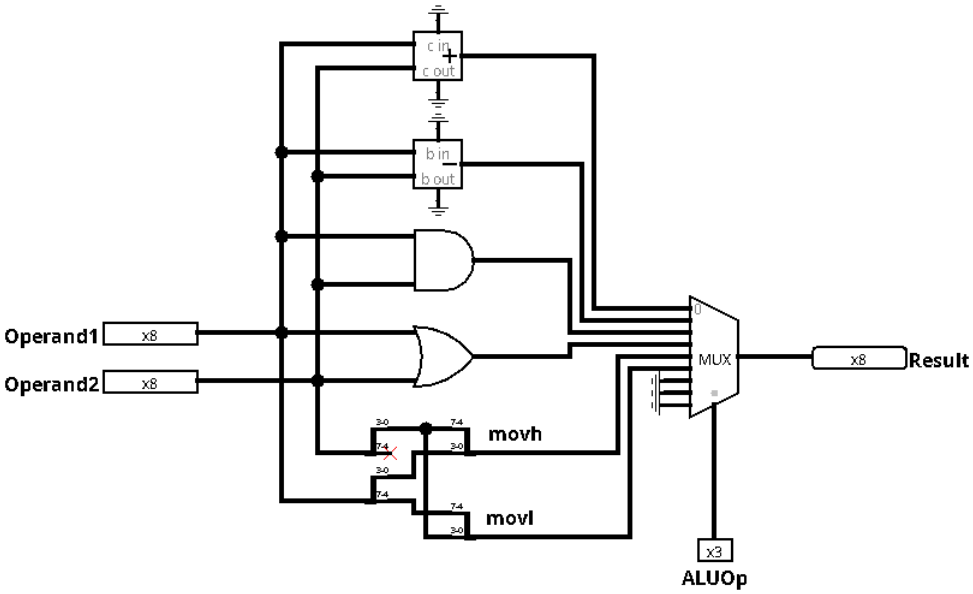

Figura 2. Imagem da ULA.

V. Conclusão

Foi desenvolvido o projeto do processador com êxito, bem como foram elaborados programas de teste que podem ser transformados em uma memória ROM e executados no *Logisim Evolution*.

Referências

[1] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 2004.