Relatório do Trabalho 3 de Arquitetura de Computadores - CI1212

Andrieli Luci Gonçalves (GRR20244903)
Camila Yuki Shibata (GRR20245211)

11 de junho de 2025

1 Introdução

Este relatório descreve a implementação de dois microprocessadores monociclo, ambos de arquitetura Sagui (de 8-bits), nas versões Vetorial e VLIW, bem como o desenvolvimento de dois programas em Assembly que realizam uma soma vetorial.

O trabalho foi realizado a partir das seguintes etapas: desenvolvimento dos diagramas de caixa para o projeto dos processadores e Unidades Lógicas e Aritméticas (ULAs), definição das tabelas de sinais de controle, desenvolvimento dos circuitos na ferramenta *Logisim Evolution 3.9.0* e escrita dos programas de teste.

2 Arquitetura Vetorial

2.1 Organização Geral

O microprocessador é composto por três regiões principais: controle, elementos de processamento (PEs) vetoriais e elemento de processamento escalar.

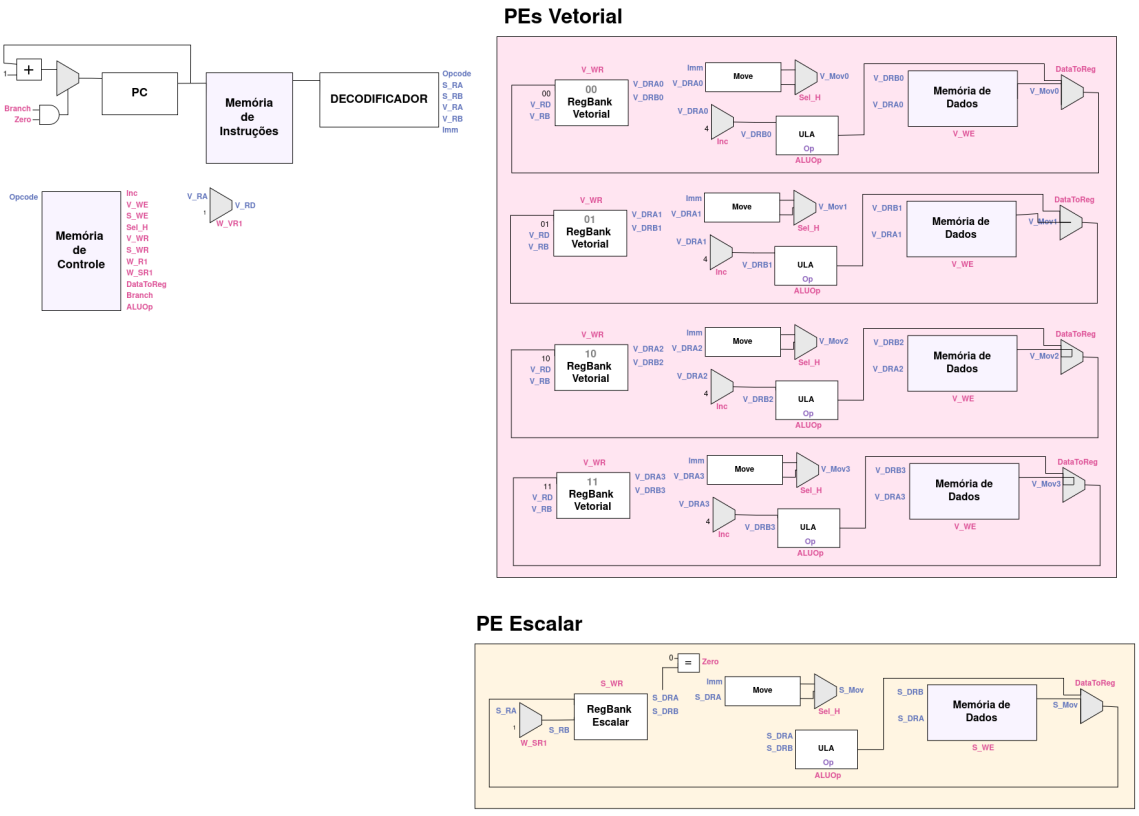

Figura 1: Diagrama do projeto do processador - Versão Vetorial

2.1.1 Região de controle

- **PC (Program Counter)**: registrador que guarda o ponteiro para a instrução a ser executada;
- **Memória de Instruções**: memória ROM onde estão armazenadas as instruções do programa;
- **Memória de Controle**: memória ROM reservada ao sinais de controle;
- **Decodificador**: ramifica os bits da instrução com base em sua categoria (vetorial ou escalar).

2.1.2 PEs vetoriais

A região é formada por quatro unidades de execução paralela, todas compostas por:

- **Banco de registradores**: contém quatro registradores, sendo o primeiro o identificador de cada PE (0, 1, 2 ou 3);
- **Move**: componente que manipula dados com base nas instruções `movh` e `movl`;
- **Memória de Dados**: memória RAM para a leitura e escrita de dados. No microprocessador, cada PE contém sua própria memória, representando, na realidade, apenas uma no total;
- **Incrementador**: estrutura externa à ULA com propósito de somar o valor 4 ao dado `dRB`;
- **Unidade Lógica e Aritmética**: realiza operações entre os operandos `A` e `B`, conforme indicado por `op`, e devolve o resultado em `S`. Operações: soma, subtração, `AND` e `NOT`.

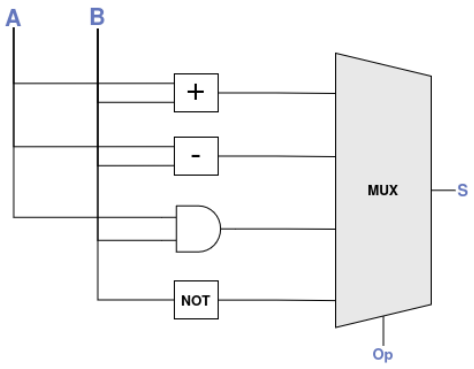

Figura 2: Diagrama da ULA - Versão Vetorial

2.1.3 PE escalar

É organizado do mesmo modo que os PEs vetoriais, contudo, possui apenas uma unidade de cada componente. No caso do banco de registradores, o primeiro (R0) é uma constante de valor 0. Além disso, há um comparador na saída `dRA` do banco de registradores, cujo objetivo é verificar se o valor é igual a zero.

2.2 Sinais de Controle

Os sinais de controle são utilizados por ambas regiões (vetorial e escalar). Operações que envolvem escrita em memória ou nos registradores são separadas.

Instr.	opcode	ALUOp	Branch	DataToReg	W_SR1	W_VR1	S_WR	V_WR	Sel_H	S_WE	V_WE	Inc
s.ld	0000	00	0	01	0	0	1	0	0	0	0	0
s.st	0001	00	0	00	0	0	0	0	0	1	0	0
s.movh	0010	00	0	10	1	0	1	0	1	0	0	0
s.movl	0011	00	0	10	1	0	1	0	0	0	0	0
s.add	0100	00	0	00	0	0	1	0	0	0	0	0
s.sub	0101	01	0	00	0	0	1	0	0	0	0	0
s.and	0110	10	0	00	0	0	1	0	0	0	0	0
s.brzr	0111	00	1	00	0	0	0	0	0	0	0	0
v.ld	1000	00	0	01	0	0	0	1	0	0	0	0
v.st	1001	00	0	00	0	0	0	0	0	0	1	0
v.movh	1010	00	0	10	0	1	0	1	1	0	0	0
v.movl	1011	00	0	10	0	1	0	1	0	0	0	0
v.add	1100	00	0	00	0	0	0	1	0	0	0	0
v.sub	1101	01	0	00	0	0	0	1	0	0	0	0
v.inc	1110	00	0	00	0	0	0	1	0	0	0	1
s.not	1111	11	0	00	0	0	1	0	0	0	0	0

Tabela 1: Tabela de sinais de controle para todas as instruções - Versão Vetorial

2.3 Novas instruções

2.3.1 `v.inc`: increment

Opcode: 1110
Descrição: incrementa em quatro unidades o registrador `RB`.
Motivação: facilita a realização de operações como `load` e `store`, que geralmente ocorrem de forma sequencial — sem a nova instrução, seria necessário utilizar, por exemplo, `v.movl` e `v.add` para percorrer um vetor.

2.3.2 `s.not`: NOT

Opcode: 1111
Descrição: realiza a operação `NOT` sobre `RB`. Se ele é igual a 0, retorna 1; caso contrário, 0.
Motivação: além de propósitos lógicos, simplifica a organização de um laço de repetição: uma vez que a Arquitetura Sagui em Bando não contém uma instrução de salto incondicional, a instrução `branch` combinada com `not` pode simplificar os desvios em um *loop*.

2.4 Programa de teste

```
v.sub r1, r1        ; zera VR[1]
v.sub r2, r2        ; zera VR[2]
v.sub r3, r3        ; zera VR[3]

s.sub r1, r1        ; zera SR[1]
s.sub r2, r2        ; zera SR[2]
s.sub r3, r3        ; zera SR[3]

v.add r2, r0        ; VR[2] = VR[0] (0, 1, 2, ou 3)
v.add r3, r0        ; VR[3] = VR[0] (0, 1, 2, ou 3)

s.movl 3            ; SR[1] = 0000 0011 (3)
s.add r2, r1        ; SR[2] = 0000 0011 (3)

loop_A:
    v.st r3, r2     ; guarda VR[3] no endereco VR[2]
    v.inc r2, r2    ; incrementa VR[2] em 4 unidades
    v.inc r3, r3    ; incrementa VR[3] em 4 unidades

    s.movl 1        ; SR[1] = 0000 0001 (1)
    s.sub r2, r1    ; decrementa SR[2] em 1
```

```
    s.not r3, r2    ; SR[2] == 0 ? 1 : 0
    s.movl 10       ; SR[1] = 0000 1010 (10)
    s.brzr r3, r1   ; se r3 == 0, volta para inst. 10

s.movl 3            ; SR[1] = 0000 0011 (3)
s.add r2, r1        ; SR[2] = 0000 0011 (3)

v.movl 8            ; SR[1] = 0000 1000 (8)
v.add r3, r1        ; SR[3] = SR[3] + 8

loop_B:
    v.st r3, r2     ; guarda VR[3] no endereco VR[2]
    v.inc r2, r2    ; incrementa VR[2] em 4 unidades
    v.inc r3, r3    ; incrementa VR[3] em 4 unidades

    s.movl 1        ; SR[1] = xxxx 0001 (low eh 1)
    s.movh 0        ; SR[1] = 0000 0001 (1)
    s.sub r2, r1    ; decrementa SR[2] em 1

    s.not r3, r2    ; SR[2] == 0 ? 1 : 0
    s.movl 6        ; SR[1] = 0000 0110 (6)
    s.movh 1        ; SR[1] = 0001 0110 (22)
    s.brzr r3, r1   ; se r3 == 0, volta para inst. 22

s.movl 3            ; SR[1] = 0001 0011 (19)
s.movh 0            ; SR[1] = 0000 0011 (3)
s.add r2, r1        ; SR[2] = 0000 0011 (3)

v.sub r2, r2        ; zera VR[2]
v.add r2, r0        ; VR[2] = VR[0] (0, 1, 2, ou 3)

loop_soma:
    v.ld r3, r2     ; guarda em VR[3] o valor no endereco VR[2]
    v.movl 12       ; VR[1] = 0000 1100 (12)
    v.add r2, r1    ; incrementa VR[2] em 12 unidades
    v.ld r1, r2     ; guarda em VR[1] o valor no endereco VR[2]

    v.add r3, r1    ; VR[3] = VR[3] + VR[1] (R = A + B)
    v.movl 12       ; VR[1] = xxxx 1100 (low eh 12)
    v.movh 0        ; VR[1] = 0000 1100 (12)
    v.add r2, r1    ; incrementa VR[2] em 12 unidades
    v.st r3, r2     ; guarda VR[3] no endereco VR[2]

    v.sub r2, r1    ; decrementa VR[2] em 12 unidades
    v.sub r2, r1    ; decrementa VR[2] em 12 unidades

    v.inc r2, r2    ; incrementa VR[2] em 4 unidades

    s.movl 1        ; SR[1] = xxxx 0001 (low eh 1)
    s.movh 0        ; SR[1] = 0000 0001 (1)
    s.sub r2, r1    ; decrementa SR[2] em 1

    s.not r3, r2    ; SR[2] == 0 ? 1 : 0
    s.movl 5        ; SR[1] = 0000 0101 (5)
    s.movh 2        ; SR[1] = 0010 0101 (37)
    s.brzr r3, r1   ; se r3 == 0, volta para inst. 37

s.movl 10           ; SR[1] = 0010 1010 (42)
s.movh 3            ; SR[1] = 0011 1010 (58)
s.brzr r0, r1       ; halt
```

3 Arquitetura VLIW

3.1 Organização Geral

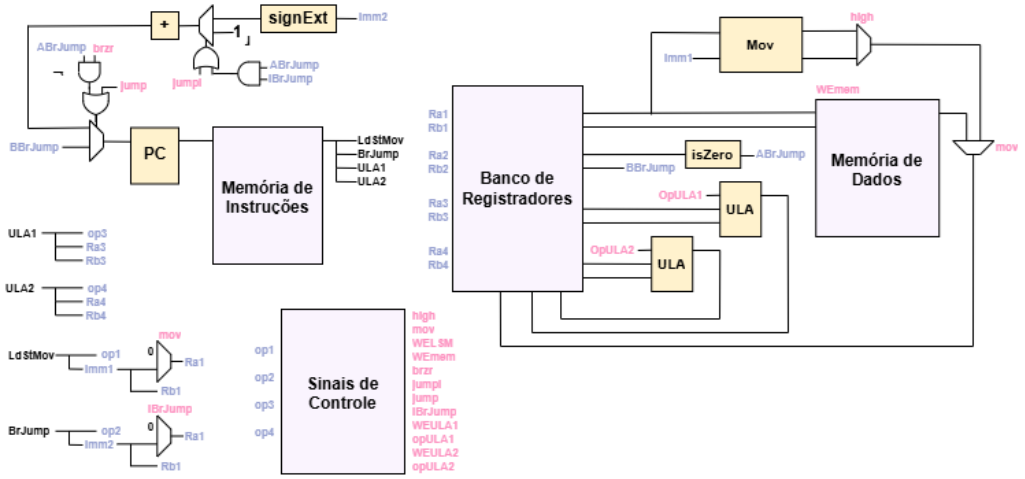

Figura 3: Diagrama do VLIW

- **PC (Program Counter)**: registrador que guarda o ponteiro para a instrução a ser executada;
- **Memória de Instruções**: memória ROM onde estão armazenadas as instruções do programa;
- **Sinais de Controle**: circuito combinacional com quatro entradas e com os sinais de controle necessários para a arquitetura como saída;
- **Banco de Registradores**: banco de registradores com quatro leituras (uma para cada *lane*) e três escritas (para as lanes `LD`/`ST`/`MOV` e ULAs);
- **Memória de Dados**: memória RAM para a leitura e escrita de dados;
- **Mov (Moves)**: realiza as manipulações dos dados necessárias para as instruções `movh` e `movl`;
- **isZero (Comparador)**: tem como saída 1 se a entrada for zero e 0 caso contrário;
- **SignExt (Extensor de Sinal)**: estende os bits do imediato de 4 para 8 bits, para as instruções `jump` e `jumpi`;
- **ULA**: tem duas entradas e como saída o resultado da operação indicada por `op`. Operações: soma, subtração, incremento, `OR`, `NOT` e `shift` (esquerda e direita).

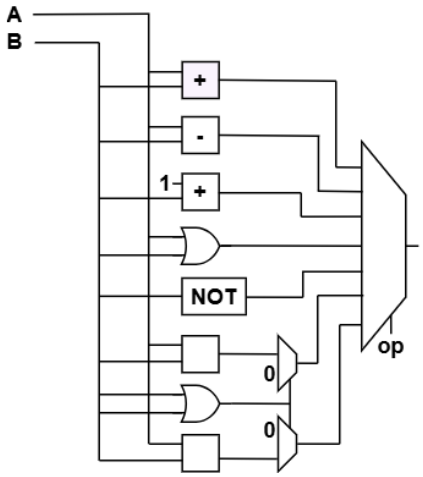

Figura 4: Diagrama da ULA - Versão VLIW

3.2 Sinais de Controle

Para organizar os sinais de controle, os dividimos por *lanes*. Nas tabelas a seguir, mostramos a elaboração desses sinais. Os sinais que não estão listados em uma *lane* podem ser entendidos como 0.

3.2.1 BR/JUMP

Instr.	brzr	jumpi	jump	IBrJump
0000	1	0	0	0
0001	0	0	0	1
0010	0	0	1	0
0011	0	1	0	0

Tabela 2: Tabela de sinais de controle das instruções `BR/JUMP` - Versão VLIW

3.2.2 `LD/ST/MOV`

Instr.	high	mov	WELdStMov	WEmem
0100	0	0	1	0
0101	0	0	0	1
0110	1	1	1	0
0111	0	1	1	0

Tabela 3: Tabela de sinais de controle das instruções `LD/ST/MOV` - Versão VLIW

3.2.3 ULA

Os sinais de controle são construídoss da mesma forma para as duas *lanes* de ULA.

Instr.	WEULA	opULA
1000	1	000
1001	1	001
1010	1	010
1011	1	011
1100	1	100
1101	1	101
1110	1	110

Tabela 4: Tabela de sinais de controle das instruções ULA - Versão VLIW

3.3 Novas Instruções

3.3.1 `jumpi`: jump immediate

Opcode: 0011
Descrição: atualiza o PC com PC + imediato, realizando um desvio relativo.
Motivação: facilita a implementação de pequenos laços ou desvios, eliminando a necessidade de carregar o novo valor de PC em um registrador auxiliar.

3.3.2 `inc`: increment

Opcode: 1010
Descrição: incrementa o conteúdo do registrador A em uma unidade.
Motivação: simplificar a construção de laços ou contadores, já que o incremento de um a um é comum em diversos programas.

3.4 Programa de Teste

3.4.1 Assembly

```
    movl 12
    add 2, 0     ; r2 = 12
    movl 9
    add 3, 0     ; r3 = 9
    movl 1       ; r0 = 1

cria_a:
    brzr 2, 3    ; if 2 = 0: l13
    sub 2, 0     ; r2--
    st 2, 2      ; M[r2] = r2
    ji -2        ; l8

    movl 12
    sub 3, 3
    add 3, 0     ; r3 = 12
    add 2, 0     ; r2 = 12
    movh 1
    movl 4
    add 1, 0     ; r1 = 20
    movh 0
    movl 1       ; r0 = 1

cria_b:
    st 1, 2      ; M[r2] = r1
    sub 3, 0     ; r3--
    inc 2        ; r2++
    inc 1        ; r1++
    movh 1
    movl 8
    brzr 3, 0    ; if r3 = 0: l35
    movh 0
    movl 1       ; r0 = 1
    ji -7        ; l24

    sub 2, 2
    movh 0
    movl 11
    add 2, 0     ; r2 = 11

loop_soma:
    ld 1, 2      ; r1 = M[r2]
    movh 0
    add 2, 0     ; r2 += 11
    ld 3, 2      ; r3 = M[r2]
    add 2, 0     ; r2 += 11
    add 3, 1     ; r3 (A[*]) + r1 (B[*])
    st 3, 2      ; M[r2] = r3
    movh 1
    movl 9
    sub 2, 0     ; r2 -= 25
    not 3, 1     ; 1 if 3 = 0, else 0
    movl 12
    brzr 3, 0    ; if 3 = 0: l41

    ji 0         ; halt
```

3.5 Escalonamento

O escalonamento a seguir foi feito respeitando as quatro lanes da arquitetura: `LdStMov`, `BrJump`, `ULA1` e `ULA2`. A instrução `movl` e outras operações de carga imediata estão na lane `LdStMov`; operações aritméticas como `add`, `sub`, `inc`, `not` ficam nas *lanes* `ULA1` e `ULA2`; desvios como `brzr` e `ji` são atribuídos à *lane* `BrJump`.

	LdStMov	**BrJump**	**ULA1**	**ULA2**
0	movl 12	-	-	-
1	-	-	add 2, 0	-
2	movl 8	-	-	-
3	-	-	add 3, 0	-
4	movl 1	-	-	-
5	-	brzr 2, 3	-	-
6	-	-	sub 2, 0	-
7	st 2, 2	ji -2	-	-
8	movl 12	-	sub 3, 3	-
9	-	-	add 3, 0	add 2, 0
10	movh 1	-	-	-
11	movl 4	-	-	-
12	-	-	add 1, 0	-
13	movh 0	-	-	-
14	movl 1	-	-	-
15	st 1, 2	-	sub 3, 0	-
16	-	-	inc 2	inc 1
17	movh 1	-	-	-
18	movl 7	-	-	-
19	-	brzr 3, 0	-	-
20	movh 0	-	-	-
21	movl 1	-	-	-
22	-	ji -7	-	-
23	movh 0	-	sub 2, 2	-
24	movl 11	-	-	-
25	-	-	add 2, 0	-
26	movl 12	-	-	-
27	ld 1, 2	-	-	-
28	movh 0	-	-	-
29	movl 12	-	-	-
30	-	-	add 2, 0	-
31	ld 3, 2	-	-	-
32	-	-	add 2, 0	add 3, 1
33	st 3, 2	-	-	-
34	movh 1	-	-	-
35	movl 9	-	-	-
36	-	-	sub 2, 0	not 3, 1
37	movl 11	-	-	-
38	-	brzr 3, 0	-	-
39	-	ji 0	-	-