

# Projeto de Processador REDUX-V

Pietro Comin - GRR20241955

5 de maio de 2025

## Resumo

Este relatório documenta o projeto do processador REDUX-V, abordando sua arquitetura, conjunto de instruções, componentes principais como a ULA e a memória de controle, além da justificativa para instruções estendidas adicionadas.

## 1 Introdução

Este relatório apresenta o desenvolvimento completo da arquitetura REDUX-V, uma arquitetura de 8 bits do tipo load-store, com organização monociclo e endereçamento por byte. O projeto foi implementado no Logisim Evolution, utilizando memória RAM do tipo `dual_port_ram`, e com clock de dois ticks por ciclo.

Por praticidade e facilidade nos testes, foram adicionadas duas memórias ROM dentro do componente `dual_port_ram`, que contêm o hexadecimal das instruções de ambos os trabalhos desenvolvidos até então na disciplina. Na implementação do código, o fato de haver uma ou duas unidades de memória não é importante, pois somente são armazenados dados na RAM a partir do endereço final das instruções contidas na ROM.

Além das instruções da arquitetura base, foram implementadas três novas instruções utilizando opcodes reservados: `jien`, `muli` e `brzrue`. Estas instruções, que serão melhor detalhadas mais à frente, visam ampliar a flexibilidade do programa de teste, com destaque para saltos condicionais estendidos e multiplicação com imediato.

## 2 Conjunto de Instruções Base

A arquitetura REDUX-V suporta um conjunto enxuto de instruções, como:

- Operações lógicas: `and`, `or`, `not`, `xor`
- Operações aritméticas: `add`, `sub`, `addi`, `slr`, `srr`
- Controle de fluxo: `ji` (jump immediate), `brzr` (branch if zero)
- Manipulação de registradores e memória: `load`, `store`

Estas que podem ser de dois tipos:

Opcode				Imediato			
7	6	5	4	3	2	1	0

Opcode				r[a]		r[b]	
7	6	5	4	3	2	1	0

Figura 1: Formatos de instrução Tipo I e Tipo R

### 3 Datapath do Processador

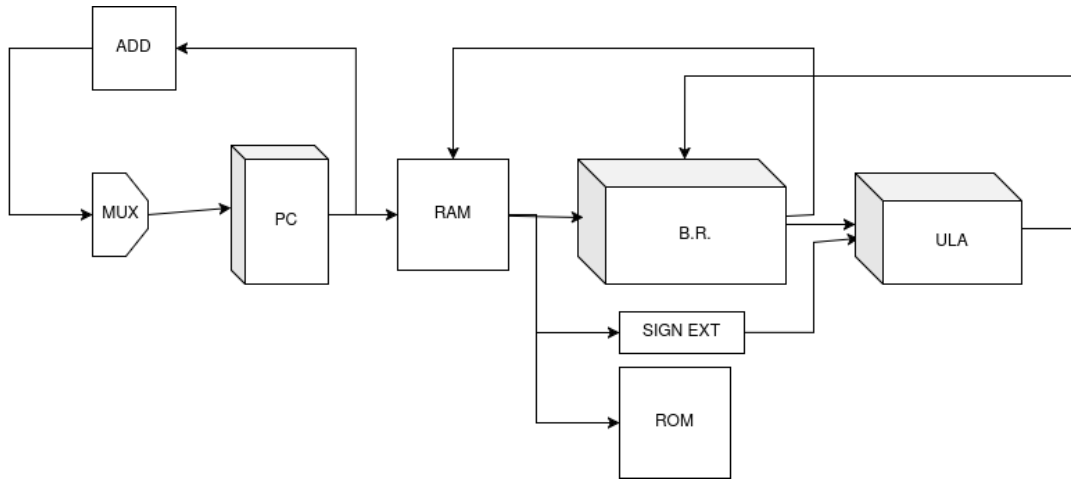


Figura 2: Datapath do processador REDUX-V

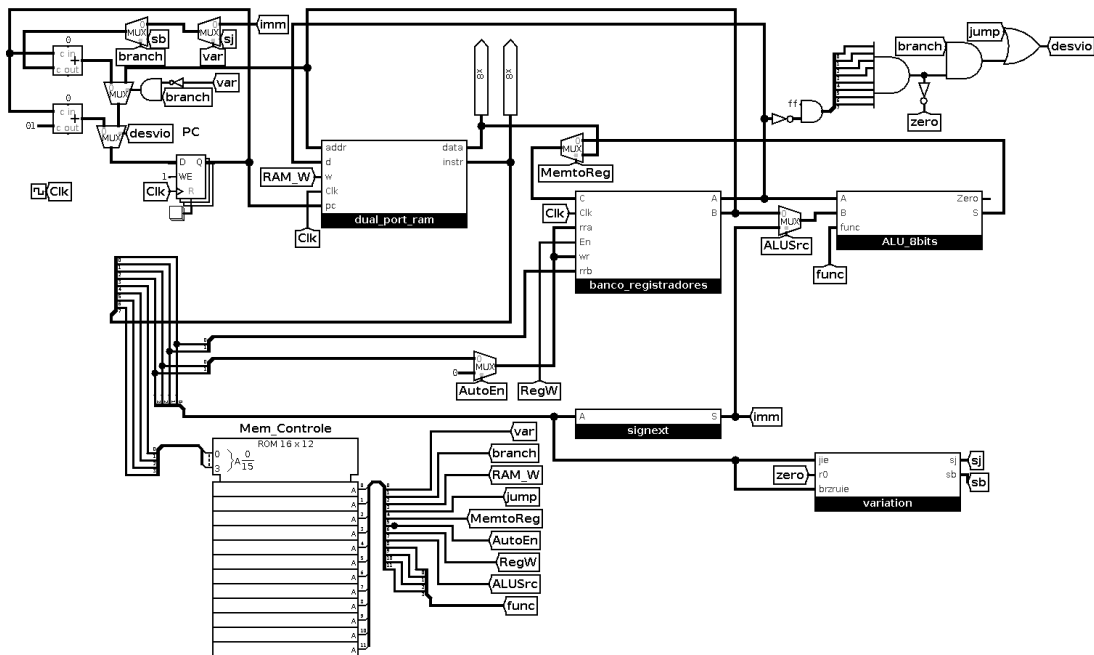


Figura 3: Figura do processador REDUX-V em circuito

O datapath define o caminho de dados percorrido durante a execução de uma instrução. Ele inclui componentes como o banco de registradores, a ULA, multiplexadores, unidade de controle, extensor de sinal, componente de variação nos saltos e memória de dados.



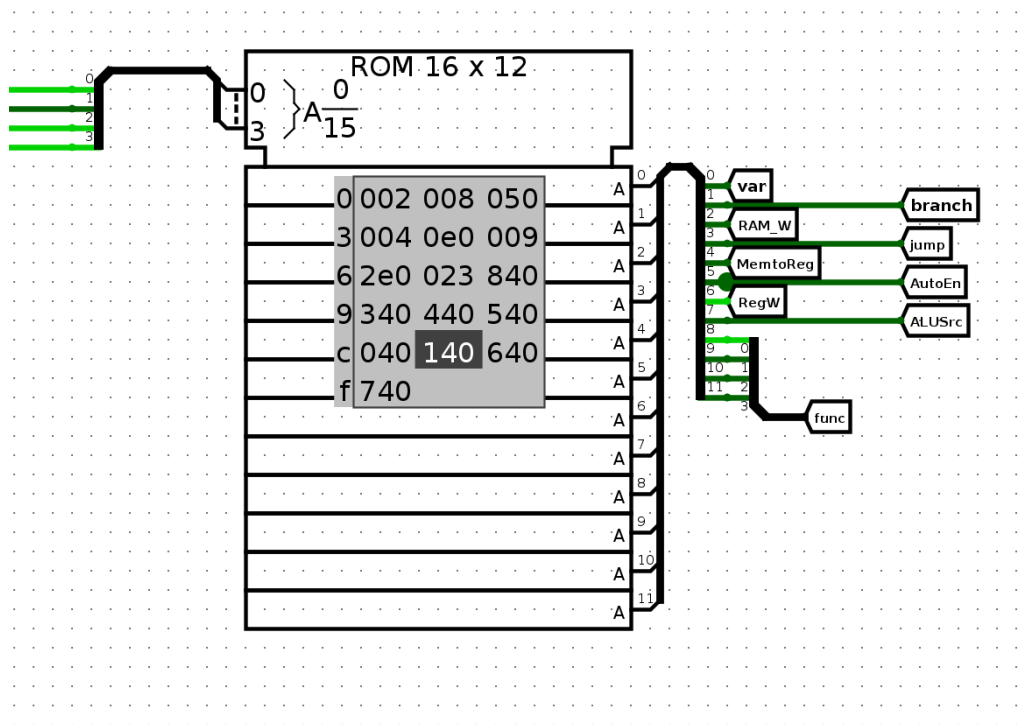


Figura 5: Diagrama da memória de controle do processador REDUX-V

## 6 Justificativa das Instruções Adicionais

As três instruções estendidas adicionadas à arquitetura (**jien**, **muli** e **brzrue**) visam ampliar a capacidade da linguagem de máquina e tornar os programas mais enxutos e expressivos.

- **jien** (Jump Immediate Extended Negative): permite saltos para trás mais longos do que o **ji**. Ideal para loops e estruturas de repetição. É uma instrução do tipo I que recebe um imediato, multiplica-o por dois, soma 9 e retorna o resultado negativo em B+1. A soma com 9 acontece pois o range de um imediato de 4 bits é [-8, 7], então a instrução é otimizada.

*Adendo: A instrução **jien** foi primeiramente implementada com a ideia de permitir um salto positivo ou negativo, dependendo do conteúdo de  $r[0]$ . Se seu conteúdo fosse igual a 0, o salto seria positivo; caso contrário, negativo. Contudo, para melhor usabilidade no código de teste, a instrução foi modificada para sempre realizar um salto negativo. O circuito anterior foi mantido, mas encontra-se inoperante.*

- **muli** (Multiplication Immediate): substitui múltiplas instruções **addi** por uma única operação de multiplicação com imediato. Como são reservados somente 4 bits de imediato, é difícil atingir valores grandes usando poucas instruções e com mais precisão que usar um shift. Por isso **muli** foi implementado.
- **brzrue** (Branch on Zero Unsigned Extended): permite saltos condicionais longos para frente, o que facilita implementações mais complexas com bifurcações. Também uma instrução do tipo I, recebe um imediato, multiplica-o por dois e soma 8.

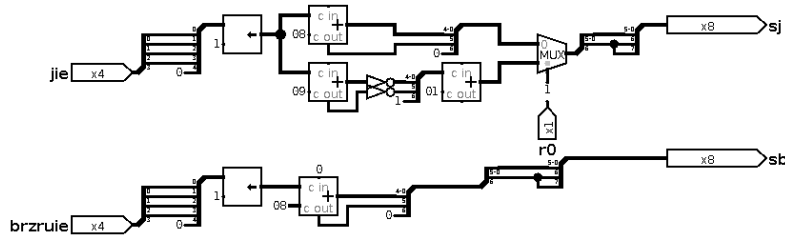


Figura 6: Componente de variação utilizado no cálculo de endereço das instruções jien e brzrue

## 6.1 Exemplo de Código com Instruções Estendidas

```

1 ; Programa que calcula um valor, aplica salto positivo e negativo
2 loop:
3 brzrue 2      ; PC = PC + 12 (se r[0] == 0, salta para fim_loop)
4 ... (11 instrucoes)
5 jien 1        ; PC = PC - 11 (salto para loop)
6
7 fim_loop:
8 ...

```

Essas instruções contribuem para a legibilidade e eficiência do código Assembly por aumentar o range dos saltos e tirar a necessidade de cálculo manual do endereço de destino do PC.

# 7 Assembly

## 7.1 Trabalho 1

```

1 sub r0, r0
2 sub r1, r1
3 sub r2, r2
4 sub r3, r3
5
6 ; os vetores comecarao no endereco de memoria 110
7
8 ; inicia valores
9 addi 1
10 sub r3, r0      ; r3 = -1
11 addi 7
12 addi 2          ; r0 = 10
13 sub r1, r0      ; r1 = -10
14 add r0, r0      ; r0 = 20
15 add r0, r0      ; r0 = 40
16 addi 5          ; r0 = 45
17 addi 5          ; r0 = 50
18 add r2, r0      ; r2 = 50
19 sub r0, r1      ; r0 = 60
20 st r0, r3       ; 60 armazenado em 0xff
21 add r3, r3      ; r3 = -2
22 add r2, r2      ; r2 = 100
23 sub r2, r1      ; r2 = 110
24 st r2, r3       ; 110 armazenado em 0xfe

```

```

25 sub r0, r0
26 sub r2, r2
27 sub r3, r3
28
29
30 ; calcula o fim_loop (PC = 23)
31 addi 1
32 sub r3, r3
33 sub r3, r0      ; end = -1
34 ld r3, r3       ; end = fim_loop
35 sub r0, r0
36
37 loop:
38 brzr r1, r3
39 ; ajusta i
40 addi 7          ; r0 = 7
41 addi 3          ; r0 = 10
42 add r1, r0      ; i += 10
43 ; guarda 1
44 sub r0, r0      ; r0 = 0
45 addi 2          ; r0 = 2
46 sub r3, r3
47 sub r3, r0      ; end = -2
48 ld r3, r3       ; end = 110
49 add r3, r1      ; end = 110 + deslocamento (i)
50 sub r0, r0      ; r0 = 0
51 st r2, r3       ; M[end] = x
52 ; guarda 2
53 addi 1          ; r0 = 1
54 add r2, r0      ; x += 1
55 addi 7          ; r0 = 8
56 addi 2          ; r0 = 10
57 add r3, r0      ; end = 110 + deslocamento + 10
58 st r2, r3       ; M[end] = x
59 ; ajusta parametros para reiniciar loop
60 sub r0, r0
61 addi 7
62 addi 2          ; r0 = 9
63 sub r1, r0      ; i -= 9
64 addi 1          ; r0 = 10
65 add r0, r0      ; r0 = 20
66 addi 3          ; r0 = 23
67 sub r3, r3      ; end = 0
68 add r3, r0      ; end = 23
69 sub r0, r0      ; r0 = 0
70 addi 1          ; r0 = 1
71 add r2, r0      ; x += 1
72 sub r0, r0      ; r0 = 0
73 brzr r0, r3     ; forca o salto para PC = 23
74
75 fim_loop:
76
77
78
79 ; soma dos vetores
80
81 sub r2, r2      ; PC = 60
82 sub r3, r3

```

```

83 addi 7
84 addi 3          ; r0 = 10
85 sub r1, r0      ; i = -10
86 sub r0, r0
87
88 ; calcula r3 (PC = 66)
89 sub r3, r3
90 addi 2          ; r0 = 2
91 sub r3, r0      ; r3 = -2
92 ld r3, r3       ; r3 = 110
93 addi 2          ; r0 = 4
94 sub r3, r0      ; r3 = 106
95 sub r0, r0
96
97 loop_2:
98 brzr r1, r3
99 addi 7
100 addi 3          ; r0 = 10
101 add r1, r0       ; i += 10
102 ;
103 sub r0, r0
104 sub r3, r3
105 addi 2          ; r0 = 2
106 sub r3, r0      ; r3 = -2
107 ld r3, r3       ; r3 = 110
108 add r3, r1       ; r3 += deslocamento (i)
109 ld r2, r3        ; r2 = M[end]
110 addi 7          ; r0 = 9
111 addi 1          ; r0 = 10
112 add r3, r0       ; r3 += 10
113 ld r0, r3        ; r0 = M[r3]
114 add r2, r0       ; soma dos vetores
115 sub r0, r0
116 addi 7
117 addi 3          ; r0 = 10
118 add r3, r0       ; r3 += 10
119 st r2, r3        ; M[end] = r2
120 sub r0, r0       ; r0 = 0
121 ;
122 addi 7
123 addi 2          ; r0 = 9
124 sub r1, r0       ; i -= 9
125 addi 7          ; r0 = 16
126 add r0, r0       ; r0 = 32
127 add r0, r0       ; r0 = 64
128 addi 2          ; r0 = 66
129 sub r3, r3
130 add r3, r0       ; r3 = 66
131 sub r0, r0
132 brzr r0, r3
133
134
135 fim_loop_2:
136 addi 3          ; PC = 106
137 add r3, r0       ; r3 = 109
138 sub r0, r0
139 ji 0            ; halt (PC = PC + 0)

```

## 7.2 Trabalho 2

```
1
2 sub r0, r0
3 sub r1, r1
4 sub r2, r2
5 sub r3, r3
6
7 ; os vetores comecarao no endereco de memoria 72
8
9 ; inicia valores
10 addi 7
11 addi 3          ; r0 = 10
12 sub r1, r0      ; r1 = -10
13 muli 7          ; r0 = 70
14 addi 2          ; r0 = 72
15 add r3, r0      ; r3 = 72
16 sub r2, r2
17 sub r0, r0
18 add r0, r1      ; r0 = i
19
20 loop:
21 brzruie 8       ; 8 * 2 + 8
22 ; ajusta i
23 sub r0, r0
24 addi 7          ; r0 = 7
25 addi 3          ; r0 = 10
26 add r1, r0      ; i += 10
27 ; guarda 1
28 sub r0, r0      ; r0 = 0
29 add r3, r1      ; end = base + deslocamento (i)
30 st r2, r3       ; M[end] = x
31 ; guarda 2
32 addi 1          ; r0 = 1
33 add r2, r0      ; x += 1
34 addi 7          ; r0 = 8
35 addi 2          ; r0 = 10
36 add r3, r0      ; end = base + deslocamento + 10
37 st r2, r3       ; M[end] = x
38 sub r3, r1      ; r3 = base + 10
39 sub r3, r0      ; r3 = base
40 ; ajusta parametros para reiniciar loop
41 addi -1         ; r0 = 9
42 sub r1, r0      ; i -= 9
43 sub r0, r0      ; r0 = 0
44 addi 1          ; r0 = 1
45 add r2, r0      ; x += 1
46 sub r0, r0      ; r0 = 0
47 add r0, r1      ; se i = 0 saira do loop
48 jie 8          ; -(8 * 2 + 7)
49
50 fim_loop:
51
52
53
54 ; soma dos vetores
55
56 sub r2, r2
```



```

57 addi 7
58 addi 3          ; r0 = 10
59 sub r1, r0      ; i = -10
60 sub r0, r0
61 add r0, r1      ; r0 = i
62 or r0, r0       ; jump pula para ca
63
64 loop_2:
65 brzruie 9       ; PC = 44
66 sub r0, r0      ; r0 = 0
67 addi 7
68 addi 3          ; r0 = 10
69 add r1, r0      ; i += 10
70 ;
71 sub r0, r0
72 add r3, r1      ; r3 += deslocamento (i)
73 ld r2, r3       ; r2 = M[end]
74 addi 7          ; r0 = 7
75 addi 3          ; r0 = 10
76 add r3, r0      ; r3 = base + 10
77 ld r0, r3       ; r0 = M[end]
78 add r2, r0      ; soma dos vetores
79 sub r0, r0
80 addi 7
81 addi 3          ; r0 = 10
82 add r3, r0      ; r3 = base + 10
83 st r2, r3       ; M[end] = r2
84 sub r3, r1
85 sub r3, r0
86 sub r3, r0      ; r3 = base
87 ;
88 addi -1         ; r0 = 9
89 sub r1, r0      ; i -= 9
90 sub r0, r0
91 add r0, r1      ; r0 = i
92 jien 8          ; PC = 68
93
94
95 fim_loop_2:
96 or r0, r0
97 ji 0           ; halt (PC = PC + 0)

```

## 8 Conclusão

O projeto do processador REDUX-V evidenciou os princípios fundamentais da arquitetura de computadores, incluindo o funcionamento de uma ULA, memória de controle e fluxo de dados via datapath. As instruções adicionais propostas aumentaram a flexibilidade da linguagem e a expressividade dos programas escritos para essa arquitetura.