RISC-V: The Free and Open RISC Instruction Set Architecture

Rodolfo Azevedo

MC404 – Organização Básica de Computadores e Linguagem de Montagem

http://www.ic.unicamp.br/~rodolfo/mc404

RISC X CISC

RISC (Reduced Instruction Set Computer)

1. **Instruções simples**
2. **Referencias a memória só com LOAD/STORE**
3. **Uso intensivo de Pipeline**
4. **Instruções de formato Fixo**
5. **Poucas instruções com poucos modos de endereçamento**
6. **Compilador complexo**
7. **Vários registradores**

CISC (Complex Instruction Set Computer)

1. **Instruções complexas**
2. **Qualquer instrução pode referenciar à memória**
3. **Pouco uso de Pipeline**
4. **Instruções com formato variável**
5. **Muitas instruções com muitos modos de endereçamento**
6. **A complexidade está no programa**
7. **Poucos registradores**

Como codificar as instruções?

Formatos das Instruções Tipo R

funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Campos da instrução:

- opcode: Código da instrução
- rd: número do reg. destino
- funct3: 3-bit Código da função (opcode adicional)
- rs1: nr. do registrador fonte (1. operando) – rs2: nr. do registrador fo
- funct7: 7-bit Código da função (opcode adicional)

Formatos das Instruções - Tipo R exemplo

funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

```
add x9,x20,x21
```

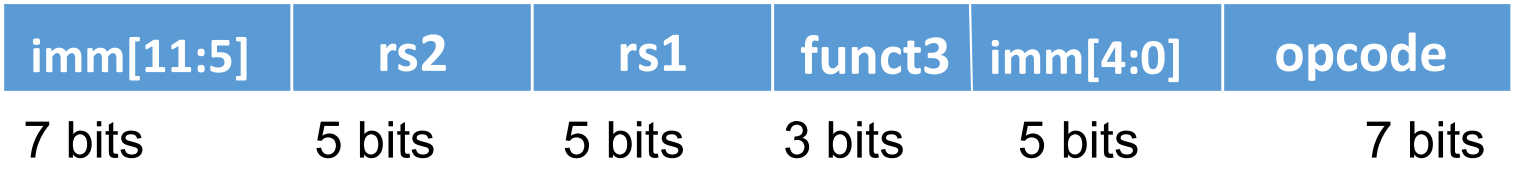
0	21	20	0	9	51
0000000	10101	10100	000	01001	0110011

Formatos das Instruções – Tipo I

immediate	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

- Instruções com constantes (Immediate) e instruções load
  - rs1: número do registrador fonte ou registrador base
  - immediate: operando constante ou offset adicionado ao end. base

Formatos das Instruções - Tipo S



- Formato imediato diferente para instruções store
  - rs1: número do registrador base
  - rs2: número do registrador fonte
  - immediate: offset adicionado ao end. base

Formatos das Instruções – RISC V resumo

Instruction	Format	funct7	rs2	rs1	funct3	rd	opcode
add (add)	R	0000000	reg	reg	000	reg	0110011
sub (sub)	R	0100000	reg	reg	000	reg	0110011

Instruction	Format	immediate	rs1	funct3	rd	opcode
addi (add immediate)	I	constant	reg	000	reg	0010011
ld (load doubleword)	I	address	reg	011	reg	0000011

Instruction	Format	immed -iate	rs2	rs1	funct3	immed -iate	opcode
sd (store doubleword)	S	address	reg	reg	011	address	0100011

Formatos das Instruções

Formato	Bits																												
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	func7							rs2					rs1					func3			rd					opcode						
I	imm[11:0]												rs1					func3			rd					opcode						
S	imm[11:5]							rs2					rs1					func3			imm[4:0]					opcode						
SB	[12]	imm[10:5]						rs2					rs1					func3			Imm[4:1]				[11]	opcode						
U	imm[31:12]																				rd					opcode						
UJ	[20]	imm[10:1]										[11]	imm[19:12]								rd					opcode						

7 bits

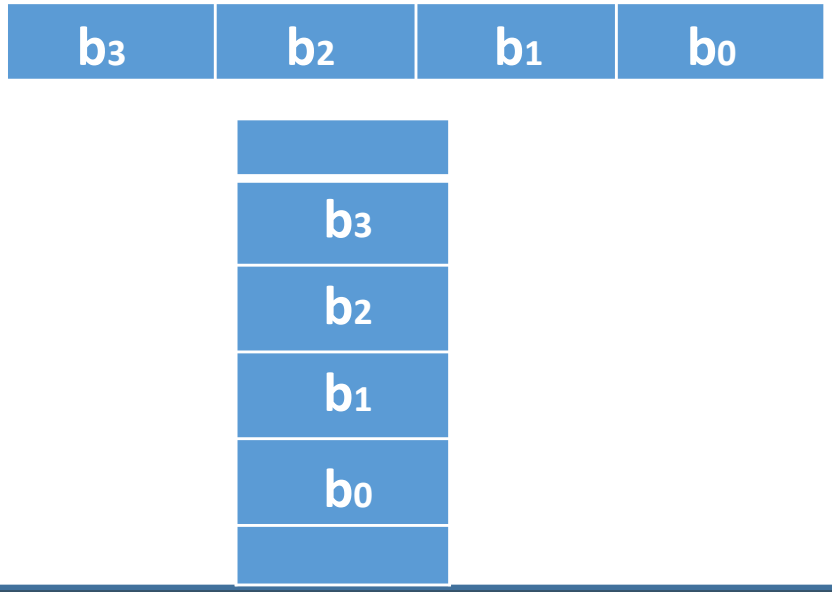
128 instruções po

Operações de Memória - Endianess

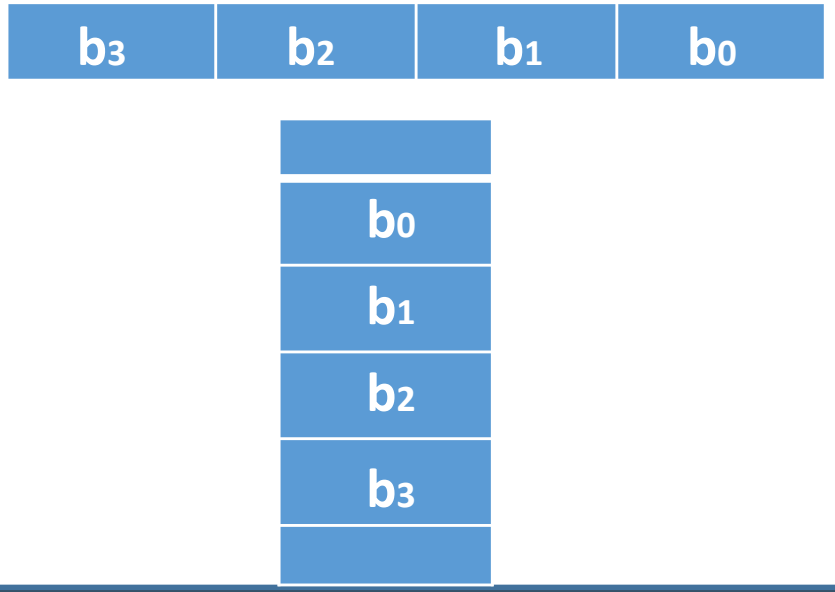
MIPS/ RISC V

- Inteiros com 32 bits
- Memória endereçada por byte

Big Endian (MIPS)

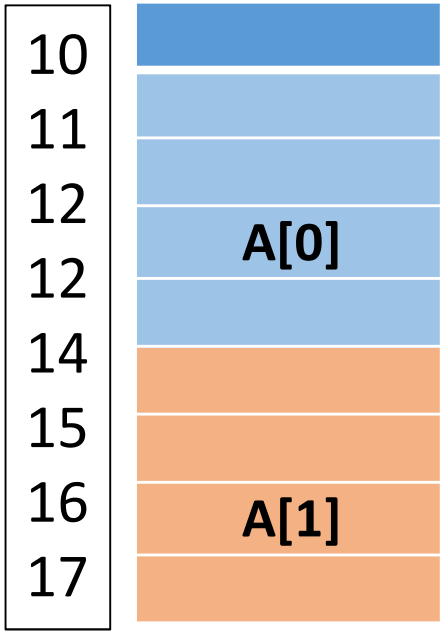


Little Endian (RISC V – x86)



Operandos de Memória

- Memória endereçada por byte



End (A[0]) = 10

End (A[1]) = 14

Formatos das Instruções

1- Sequencia de instruções para: A[2] = h + A[8]

2 - Sequencia de instruções para: A[i] = h + A[i]

Operandos de Memória - exemplos

1- Sequencia de instruções para: A[2] = h + A[8]

- Endereço de A em s1, h em t1

```
lw t0, 32(s1)
add t0, t1, t0
sw t0, 16(s1)
```

Formatos das Instruções

2 - sequencia de instruções para: A[i] = h + A[i]

- endereço inicial de A em s0 e h e i estão em t0 e t1, respectivamente

```
add t2, t1, t1
add t2, t2, t2          # 4*i
add t2, t2, s0          # 4i + A
lw t3, 0(t2)
add t3, t0, t3
sw t3,0(t2)
```

Funções

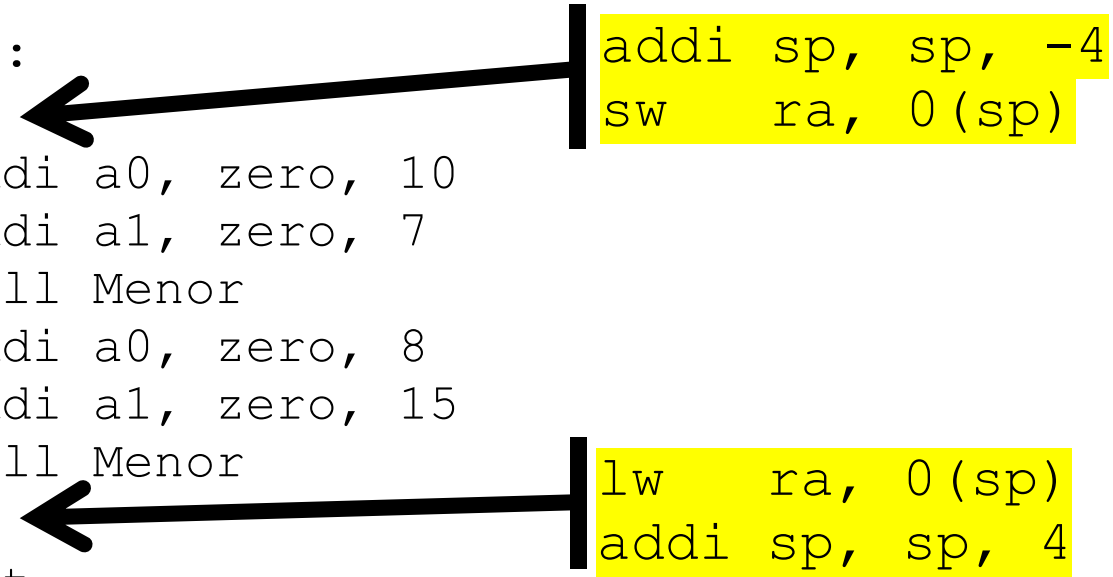
- Trechos de código que executam uma tarefa específica
- Organizadas separadamente para facilitar reuso e legibilidade
- Permite a divisão de tarefas entre desenvolvedores e uso de bibliotecas
- Precisam de convenções para operarem corretamente

Exemplo de função: Menor(int x, y)

A função Menor(int x, y)

```
Menor:
  blt a0, a1, fim
  add a0, zero, a1
fim:
  ret

Main:
  addi sp, sp, -4
  sw   ra, 0(sp)
  addi a0, zero, 10
  addi a1, zero, 7
  call Menor
  addi a0, zero, 8
  addi a1, zero, 15
  call Menor
  lw   ra, 0(sp)
  addi sp, sp, 4
  ret
```



Pilha

- Pilha cresce para baixo
- Registrador sp aponta para o último elemento da pilha
- Inclua sempre a sequência simétrica nas funções

Início

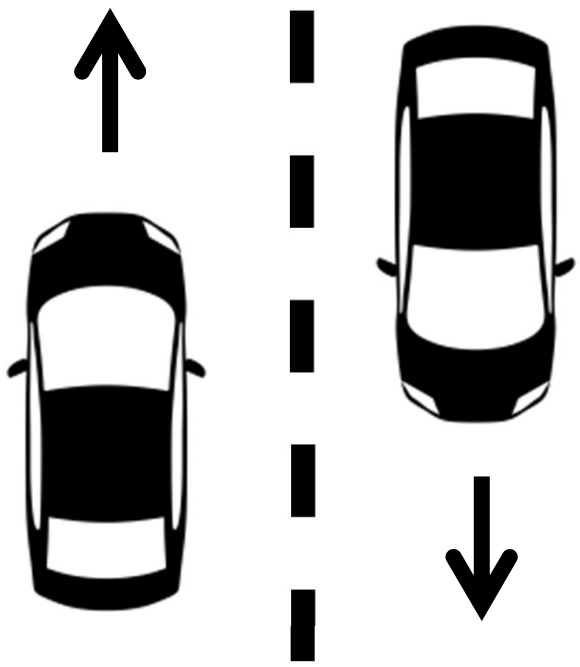
```
addi sp, sp, -8
sw   ra, 0(sp)
sw   s0, 4(sp)
```

Final

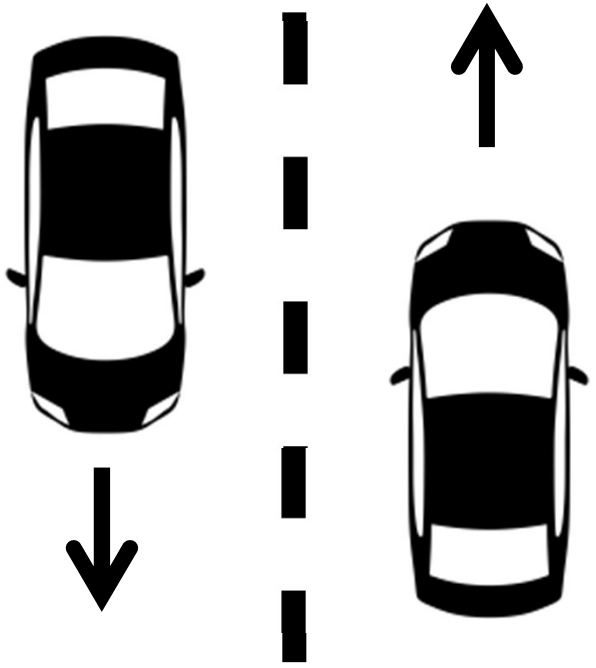
```
lw   s0, 4(sp)
lw   ra, 0(sp)
addi sp, sp, 8
```

Quem está dirigindo do lado certo?

Inglaterra



Brasil

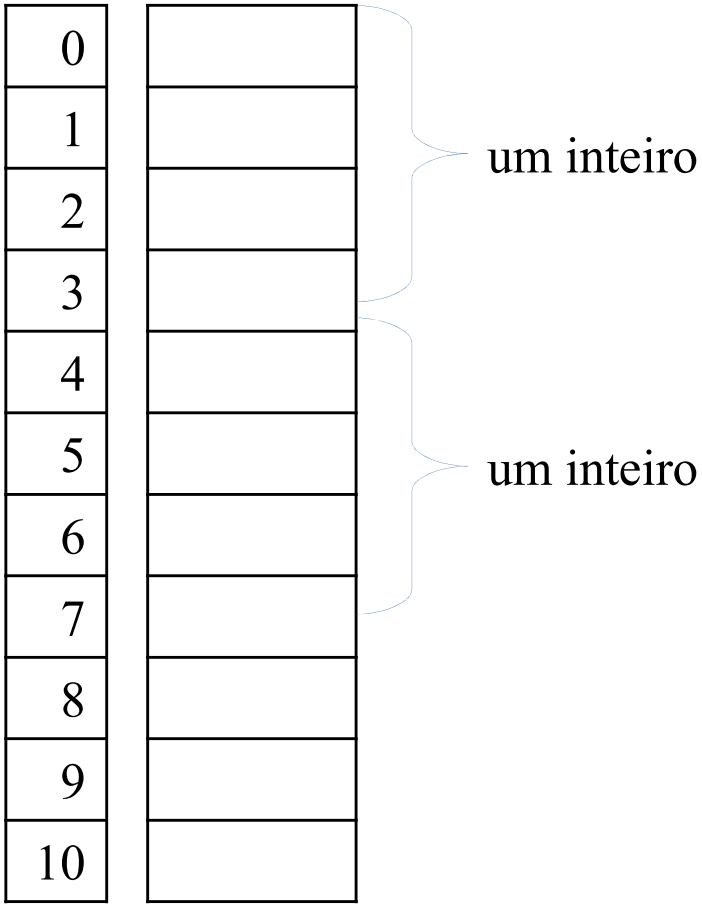


Convenções estão em todos os lugares

Registradores (novamente)

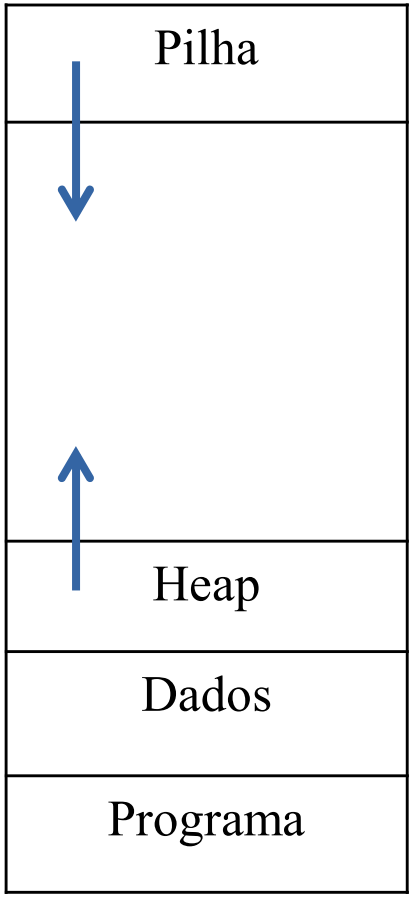
Register	ABI Name	Description	Saver
x0	Zero	Always zero	
x1	ra	Return addres	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	
x4	tp	Thread pointer	
x5	t0	Temporary / alternate return address	Caller
x6–7	t1–2	Temporary	Caller
x8	s0/fp	Saved register / frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function argument / return value	Caller
x12–17	a2–7	Function argument	Caller
x18–27	s2–11	Saved register	Callee
x28–31	t3–6	Temporary	Caller

Endereçamento da Memória



Organização da Memória

- Programas utilizam a memória para armazenar variáveis
- Normalmente, o compilador é responsável por organizar a memória e garantir o espaço para cada uma delas
- Em assembly, o programador é responsável por garantir o espaço das variáveis



Pilha

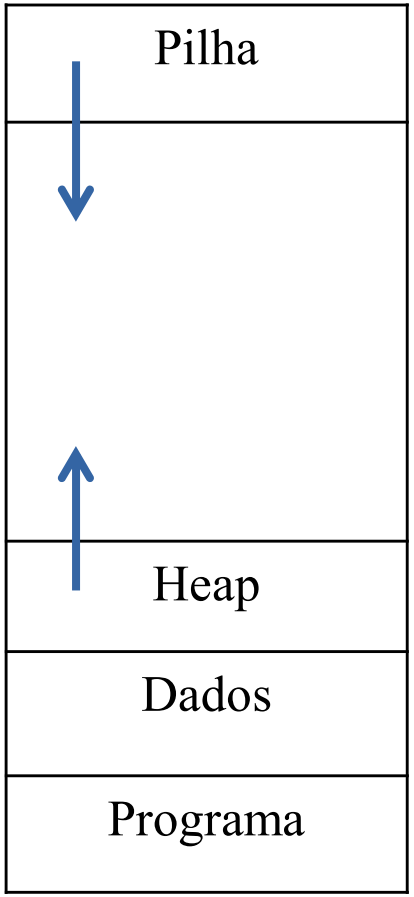
- Pilha cresce para baixo
- Registrador sp aponta para o último elemento da pilha
- Inclua sempre a sequência simétrica nas funções

Início

```
addi sp, sp, -8
sw   ra, 0(sp)
sw   s0, 4(sp)
```

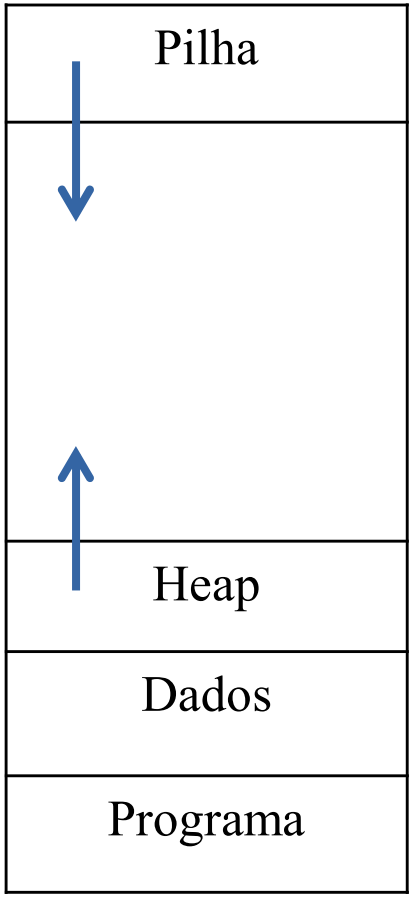
Final

```
lw   s0, 4(sp)
lw   ra, 0(sp)
addi sp, sp, 8
```



Heap

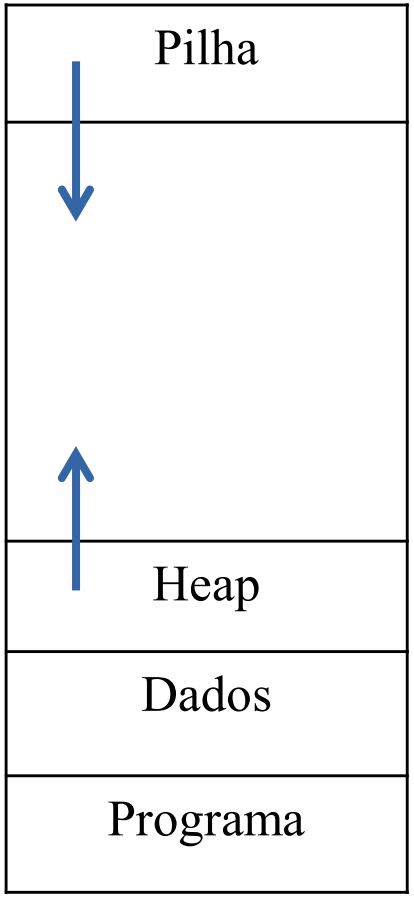
- Cresce para cima
- Armazena variáveis alocadas dinamicamente (ex.: malloc em C)
- Mais detalhes à frente



Dados

- Armazena constantes e variáveis globais dos programas

```
..section .data
.altura:
   .word 200
.largura:
   .word 17
..section .text
```



Ordem dos Bytes - Endian

- Temos duas formas de armazenar uma palavra (32 bits = 4 bytes) na memória
- Big Endian: Byte mais significativo primeiro
- Little Endian: Byte menos significativo primeiro

- Exemplo:
- 0x12345678

Endereço	Big Endian	Little Endian
0	0x12	0x78
1	0x34	0x56
2	0x56	0x34
3	0x78	0x12

PROGRAMA – MAIOR, NEMOR E SOMA DOS ELEMNTOS DE UM VETOR, COM PROCEDIMENTOS

```
.section .data
    vector:
      .word 10      #0x0000000a
      .word 5       #0x00000005
      .word 245     #0x000000f5
      .word 5461    #0x00001555
      .word 157     #0x0000009d
.section .text
   main:
      lui s0, %hi(vector)
      addi s0, s0, %lo(vector)
     addi sp,sp,-4
     sw ra,0(sp)
     call soma_vet
     call menor_vet
     call maior_vet
     lw ra,0(sp)
     addi sp,sp,4
  ret
```

```
soma_vet:
     addi sp,sp,-4
     sw s0,0(sp)
     addi t0,t0,5
    add s1,zero,zero
soma:
    lw t1,0(s0)
    add s1,s1,t1
    addi s0,s0,4
    lw t1, 0(s0)
    addi t0,t0,-1
    bne t0,zero, soma
    add a0, s1,zero
    addi t0,zero,1
    ecall
    lw s0,0(sp)
    addi sp,sp,4
ret
```

```
menor_vet:
    addi sp,sp,-4
    sw s0,0(sp)
    lw s1,0(s0)
    addi t0,zero,4
    teste_men:
    addi s0,s0,4
    lw s2, 0(s0)
   slt s3,s1,s2
   bne s3,zero,salta_men
   add s1,zero,s2
salta_men:
   addi t0,t0,-1
   bne t0,zero, teste_men
   add a0, s1,zero
   addi t0,zero,1
   ecall
   lw s0,0(sp)
   addi sp,sp,4
 ret
```

```
maior_vet:
   addi sp,sp,-4
   sw s0,0(sp)
   addi t0,t0,4
   lw s1, 0(s0)
teste_mai:
   addi s0,s0,4
   lw s2, 0(s0)
   slt s3,s1,s2
   beq s3,zero,salta_mai
   add s1,zero,s2
salta_mai:
   addi t0,t0,-1
   bne t0,zero, teste_mai
   add a0, s1,zero
   addi t0,zero,1
   ecall
   lw s0,0(sp)
   addi sp,sp,4
 ret
```

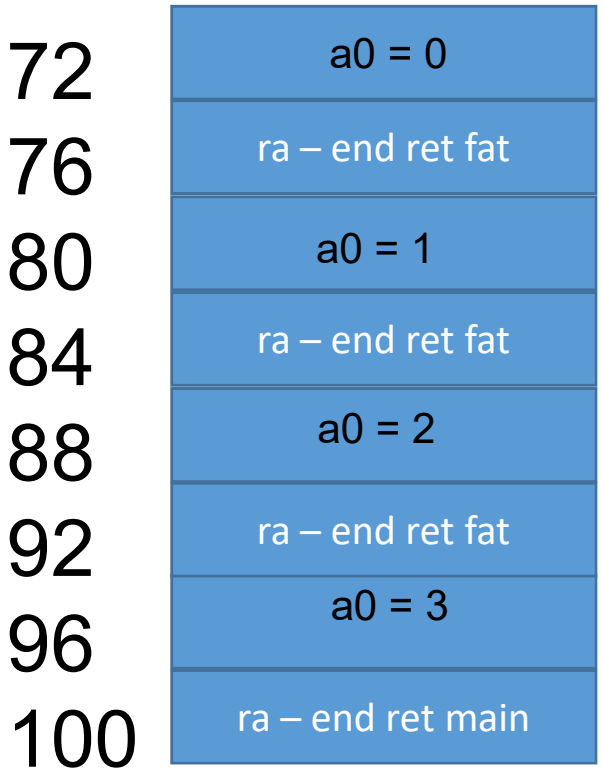
Exemplo - Procedimentos Aninhados - Fatorial

Código:

```
     int fact (int n)
{
  if (n < 1) return (1);
  else return n * fact(n - 1);
}
```

- Argumento n em a0
- Resultado em s0

Exemplo - Procedimentos Aninhados - Fatorial



<- sp – 8 novo sp

<- sp – 4

<- sp

```
addi  sp, sp,-8
sw  ra,4( sp)
sw  a0,0( sp)
```