



Procedimentos

- Estruturação de programas.
- Mais fácil de entender.
- Reutilização.
- Dividir em partes menores.

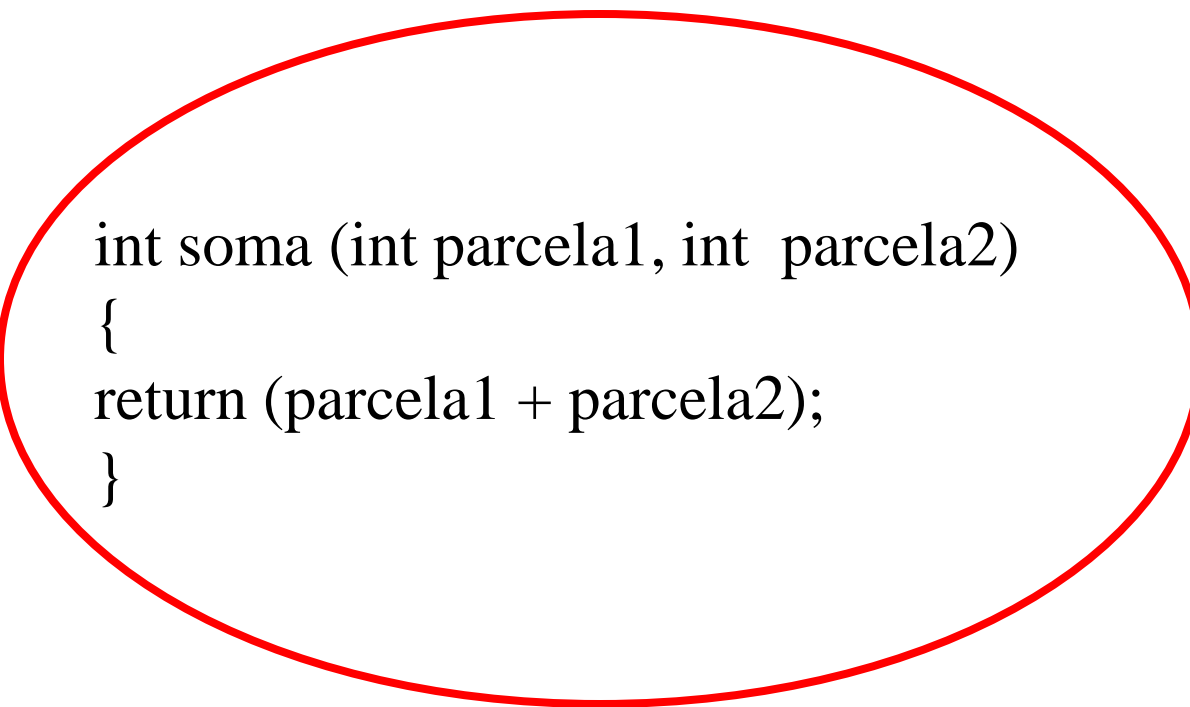


Panorama

- Funções em C
- Instruções MIPS para Funções
- A pilha (Stack)
- Convenções de Registradores
- Exemplo

```
main ( )  
{  
    int x, y, z;  
    x=1; y=2;  
    z = soma (x,y);  
  
}
```

**De quais informações o
compilador/programador
deve manter o registro?**



```
int soma (int parcela1, int parcela2)  
{  
    return (parcela1 + parcela2);  
}
```

**Como fazer essa
parte em MIPS?**

```
main ( )
{
    int x, y, z;
    x=1; y=2;
    z = soma (x,y);
}
```

```
int soma (int parc1, int parc2)
{
    return (parc1 + parc2);
}
```

MEMÓRIA

Endereço	instrução
1000	
1004	
1008	
1012	
1016	
1020	
1024	
1028	
1032	
1036	



Contabilidade de Chamada de Função

- Registradores tem papel fundamental para manter registro de informações nas chamadas de funções.
- **Convenção de Registradores:**
 - Endereço de retorno `$ra`
 - Argumentos `$a0, $a1, $a2, $a3`
 - Valor de Retorno `$v0, $v1`
 - Variáveis locais `$s0, $s1, • •, $s7`
- **Uso da instrução, jump register ou jr**

Endereço instrução

1000	addi \$s1, \$zero, 1	# x = 1
1004	addi \$s2, \$zero, 2	# y = 2

Definindo as variáveis do programa

Endereço instrução

1000	addi \$s1, \$zero, 1	# x = 1
1004	addi \$s2, \$zero, 2	# y = 2
1008	add \$a0, \$zero, \$s1	# \$a0 = x
1012	add \$a1, \$zero, \$s2	# \$a1 = y

Atribuindo valores aos argumentos da função:

x em a0

y em a1

Endereço	instrução	
1000	addi \$s1, \$zero, 1	# x = 1
1004	addi \$s2, \$zero, 2	# y = 2
1008	add \$a0, \$zero, \$s1	# \$a0 = x
1012	add \$a1, \$zero, \$s2	# \$a1 = y
1016	addi \$ra, \$zero, 1024	# \$ra = 1024, end. de retorno da função
1020	j soma	# salta para o endereço soma

Definindo e armazenando o endereço de retorno

e

Chamando a função

Endereço instrução

1000	addi \$s1, \$zero, 1	# i = 1
1004	addi \$s2, \$zero, 2	# j = 2
1008	add \$a0, \$zero, \$s1	# \$a0 = i
1012	add \$a1, \$zero, \$s2	# \$a1 = 2
1016	addi \$ra, \$zero, 1024	# \$ra = 1024, end. de retorno da função
1020	j soma	# salta para o endereço soma

....

soma: 5000	add \$vo, \$a0, \$a1	# \$vo = \$a0 + \$a1
5004	jr \$ra	# salta para o endereço contido em \$ra

Executando a função e Armazenando o resultado no registrador de retorno (v0)
Retornando para o valor armazenado anteriormente em ra

Endereço instrução

1000	addi \$s1, \$zero, 1	# i = 1
1004	addi \$s2, \$zero, 2	# j = 2
1008	add \$a0, \$zero, \$s1	# \$a0 = i
1012	add \$a1, \$zero, \$s2	# \$a1 = 2
1016	addi \$ra, \$zero, 1024	# \$ra = 1024, end. de retorno da função
1020	j soma	# salta para o endereço soma
1024	# k = soma(i,j)

....

soma: 5000	add \$vo, \$a0, \$a1	# \$vo = \$a0 + \$a1
5004	jr \$ra	# salta para o endereço contido em \$ra

Endereço instrução

1000	addi \$s1, \$zero, 1	# i = 1
1004	addi \$s2, \$zero, 2	# j = 2
1008	add \$a0, \$zero, \$s1	# \$a0 = i
1012	add \$a1, \$zero, \$s2	# \$a1 = 2
1016	addi \$ra, \$zero, 1024	# \$ra = 1024, end. de retorno da função
1020	j soma	# salta para o endereço soma
1024	# k = soma(i,j)

....

soma: 5000	add \$vo, \$a0, \$a1	# \$vo = \$a0 + \$a1
5004	jr \$ra	# salta para o endereço contido em \$ra



Instruções de Suporte para Funções

- Instrução única para **pular e salvar o endereço de retorno**: jump and link (`jal`)
- Antes:

```
1016  addi $ra, $zero, 1024    # $ra = 1024, end. de retorno da função
1020  j soma                  # salta para o endereço soma
```
- Depois:

```
1016  jal soma                # salta para o endereço soma
```
- Por que ter uma `jal`?
 - Torne o **caso comum rápido**: funções são muito comuns.



Instruções de Suporte para Funções

- Sintaxe de `jal` (jump and link) é a mesma de `j` (jump):
 - `jal label`
- `jal` deveria na verdade ser chamada `laj` de link and jump:
 - Passo 1 (link): **Salva o endereço da próxima instrução** em `$ra` (Por que a próxima instrução? Por que não a corrente?)
 - Passo 2 (jump): **Pule para o label** dado

Instruções de Suporte para Funções

- Sintaxe de **jr** (jump register):
 - `jr register`
- Ao invés de prover um label para pular para, a instrução `jr` provê um **registrador** que **contém** um **endereço** para onde pular.
- Útil somente se nós sabemos o **endereço exato** para onde pular: raramente aplicável.
- Muito útil para **chamadas de funções**:
 - `jal` **guarda** o **endereço de retorno** no registrador (\$ra) (**chamada de uma função**)
 - `jr` **pula** de volta **para** aquele **endereço (retorno da função)**

```
main ( )  
{  
...  
    x = 1;  
    y = 2;  
    z = soma ( x , y );  
  
}
```

```
int soma (int r, int s)  
{  
    return ( r + s )  
}
```

Mapeamento:

$x \rightarrow \$s1$

$y \rightarrow \$s2$

$z \rightarrow \$s3$

```
main ( )  
{  
...  
    x = 1;  
    y = 2;  
    z = soma ( x , y );  
}
```

```
int soma (int r, int s)  
{  
    return ( r + s )  
}
```

Mapeamento:

$x \rightarrow \$s1$

$y \rightarrow \$s2$

$z \rightarrow \$s3$

```
.text  
.globl main
```

```
main:  
addi $s1, $zero, 1  
addi $s2, $zero, 2
```

Definindo as variáveis do
programa

$x = 1$

$y = 2$


```
main ( )  
{  
...  
    x = 1;  
    y = 2;  
    z = soma ( x , y );  
}
```

```
int soma (int r, int s)  
{  
    return ( r + s )  
}
```

Mapeamento:

$x \rightarrow \$s1$

$y \rightarrow \$s2$

$z \rightarrow \$s3$

```
.text  
.globl main
```

```
main:  
addi $s1, $zero, 1  
addi $s2, $zero, 2
```

```
add $a0, $zero, $s1  
add $a1, $zero, $s2
```

Colocando os argumentos,
x em \$a0
y em \$a1

```

main ( )
{
...
    x = 1;
    y = 2;
    z = soma ( x , y );
}

```

```

int soma (int r, int s)
{
    return ( r + s )
}

```

Mapeamento:

$x \rightarrow \$s1$
 $y \rightarrow \$s2$
 $z \rightarrow \$s3$

```

.text
.globl main

```

```

main:
addi $s1, $zero, 1
addi $s2, $zero, 2

```

```

add $a0, $zero, $s1
add $a1, $zero, $s2
jal soma
nop
add $s3, $zero, $v0

```

Chamando a função e
 colocando o retorno em z,
 $v0 \rightarrow z$

```

main ( )
{
...
    x = 1;
    y = 2;
    z = soma ( x , y );
}

int soma (int r, int s)
{
    return ( r + s )
}

```

Mapeamento:

$x \rightarrow \$s1$

$y \rightarrow \$s2$

$z \rightarrow \$s3$

```

.text
.globl main

main:
    addi $s1, $zero, 1
    addi $s2, $zero, 2
    add $a0, $zero, $s1
    add $a1, $zero, $s2
    jal soma
    nop
    add $s3, $zero, $v0

```

soma:

add \$v0, \$a0, \$a1

jr \$ra

Definindo a função e retornando



Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20110001	addi \$17,\$0...	13: addi \$s1, \$zero, 1
<input type="checkbox"/>	0x00400004	0x20120002	addi \$18,\$0...	14: addi \$s2, \$zero, 2
<input type="checkbox"/>	0x00400008	0x00112020	add \$4,\$0,\$17	16: add \$a0, \$zero, \$s1
<input type="checkbox"/>	0x0040000c	0x00122820	add \$5,\$0,\$18	17: add \$a1, \$zero, \$s2
<input type="checkbox"/>	0x00400010	0x0c100008	jal 0x00400020	18: jal soma
<input type="checkbox"/>	0x00400014	0x00000000	nop	19: nop
<input type="checkbox"/>	0x00400018	0x00000000	nop	20: nop
<input type="checkbox"/>	0x0040001c	0x00029820	add \$19,\$0,\$2	21: add \$s3, \$zero, \$v0
<input type="checkbox"/>	0x00400020	0x00851020	add \$2,\$4,\$5	28: add \$v0, \$a0, \$a1
<input type="checkbox"/>	0x00400024	0x03e00008	jr \$31	29: jr \$ra

Labels

Label	Address
(global)	
main	0x00400000
funcao_01.asm	
soma	0x00400020

☒ Data ☒ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000



0x10010000 (.data)

☒ Hexadecimal Addresses☒ Hexadecimal Values☐ ASCII

Mars Messages

Run I/O

Assemble: assembling G:\aulas_puc\MIPS_pcspim_teorial\programas_mips\funcao_01.asm

Clear

Assemble: operation completed successfully.

Run speed at max (no interaction)

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000



Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20110001	addi \$17,\$0...	13: addi \$s1, \$zero, 1
<input type="checkbox"/>	0x00400004	0x20120002	addi \$18,\$0...	14: addi \$s2, \$zero, 2
<input type="checkbox"/>	0x00400008	0x00112020	add \$4,\$0,\$17	16: add \$a0, \$zero, \$s1
<input type="checkbox"/>	0x0040000c	0x00122820	add \$5,\$0,\$18	17: add \$a1, \$zero, \$s2
<input type="checkbox"/>	0x00400010	0x0c100008	jal 0x00400020	18: jal soma
<input type="checkbox"/>	0x00400014	0x00000000	nop	19: nop
<input type="checkbox"/>	0x00400018	0x00000000	nop	20: nop
<input type="checkbox"/>	0x0040001c	0x00029820	add \$19,\$0,\$2	21: add \$s3, \$zero, \$v0
<input type="checkbox"/>	0x00400020	0x00851020	add \$2,\$4,\$5	28: add \$v0, \$a0, \$a1
<input type="checkbox"/>	0x00400024	0x03e00008	jr \$31	29: jr \$ra

Labels

Label	Address
(global)	
main	0x00400000
funcao_01.asm	
soma	0x00400020

☒ Data ☒ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000



0x10010000 (.data)

☒ Hexadecimal Addresses☒ Hexadecimal Values☐ ASCII

Mars Messages

Run I/O

Clear

Run speed at max (no interaction)

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000001
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400004
hi		0x00000000
lo		0x00000000



Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20110001	addi \$17,\$0...	13: addi \$s1, \$zero, 1
<input type="checkbox"/>	0x00400004	0x20120002	addi \$18,\$0...	14: addi \$s2, \$zero, 2
<input type="checkbox"/>	0x00400008	0x00112020	add \$4,\$0,\$17	16: add \$a0, \$zero, \$s1
<input type="checkbox"/>	0x0040000c	0x00122820	add \$5,\$0,\$18	17: add \$a1, \$zero, \$s2
<input type="checkbox"/>	0x00400010	0x0c100008	jal 0x00400020	18: jal soma
<input type="checkbox"/>	0x00400014	0x00000000	nop	19: nop
<input type="checkbox"/>	0x00400018	0x00000000	nop	20: nop
<input type="checkbox"/>	0x0040001c	0x00029820	add \$19,\$0,\$2	21: add \$s3, \$zero, \$v0
<input type="checkbox"/>	0x00400020	0x00851020	add \$2,\$4,\$5	28: add \$v0, \$a0, \$a1
<input type="checkbox"/>	0x00400024	0x03e00008	jr \$31	29: jr \$ra

Labels

Label	Address
(global)	
main	0x00400000
funcao_01.asm	
soma	0x00400020

☒ Data ☒ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000



0x10010000 (.data)

☒ Hexadecimal Addresses☒ Hexadecimal Values☐ ASCII

Mars Messages

Run I/O

Clear

Run speed at max (no interaction)

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000001
\$s2	18	0x00000002
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400008
hi		0x00000000
lo		0x00000000



Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20110001	addi \$17,\$0...	13: addi \$s1, \$zero, 1
<input type="checkbox"/>	0x00400004	0x20120002	addi \$18,\$0...	14: addi \$s2, \$zero, 2
<input type="checkbox"/>	0x00400008	0x00112020	add \$4,\$0,\$17	16: add \$a0, \$zero, \$s1
<input type="checkbox"/>	0x0040000c	0x00122820	add \$5,\$0,\$18	17: add \$a1, \$zero, \$s2
<input type="checkbox"/>	0x00400010	0x0c100008	jal 0x00400020	18: jal soma
<input type="checkbox"/>	0x00400014	0x00000000	nop	19: nop
<input type="checkbox"/>	0x00400018	0x00000000	nop	20: nop
<input type="checkbox"/>	0x0040001c	0x00029820	add \$19,\$0,\$2	21: add \$s3, \$zero, \$v0
<input type="checkbox"/>	0x00400020	0x00851020	add \$2,\$4,\$5	28: add \$v0, \$a0, \$a1
<input type="checkbox"/>	0x00400024	0x03e00008	jr \$31	29: jr \$ra

Labels

Label	Address
(global)	
main	0x00400000
funcao_01.asm	
soma	0x00400020

☒ Data ☒ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000



0x10010000 (.data)

☒ Hexadecimal Addresses☒ Hexadecimal Values☐ ASCII

Mars Messages

Run I/O

Clear

Run speed at max (no interaction)

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000001
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000001
\$s2	18	0x00000002
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x0040000c
hi		0x00000000
lo		0x00000000



Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20110001	addi \$17,\$0...	13: addi \$s1, \$zero, 1
<input type="checkbox"/>	0x00400004	0x20120002	addi \$18,\$0...	14: addi \$s2, \$zero, 2
<input type="checkbox"/>	0x00400008	0x00112020	add \$4,\$0,\$17	16: add \$a0, \$zero, \$s1
<input type="checkbox"/>	0x0040000c	0x00122820	add \$5,\$0,\$18	17: add \$a1, \$zero, \$s2
<input type="checkbox"/>	0x00400010	0x0c100008	jal 0x00400020	18: jal soma
<input type="checkbox"/>	0x00400014	0x00000000	nop	19: nop
<input type="checkbox"/>	0x00400018	0x00000000	nop	20: nop
<input type="checkbox"/>	0x0040001c	0x00029820	add \$19,\$0,\$2	21: add \$s3, \$zero, \$v0
<input type="checkbox"/>	0x00400020	0x00851020	add \$2,\$4,\$5	28: add \$v0, \$a0, \$a1
<input type="checkbox"/>	0x00400024	0x03e00008	jr \$31	29: jr \$ra

Labels

Label	Address
(global)	
main	0x00400000
funcao_01.asm	
soma	0x00400020

☒ Data ☒ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000



0x10010000 (.data)

☒ Hexadecimal Addresses☒ Hexadecimal Values☐ ASCII

Mars Messages

Run I/O

Clear

Run speed at max (no interaction)

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000001
\$a1	5	0x00000002
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000001
\$s2	18	0x00000002
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400010
hi		0x00000000
lo		0x00000000



Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20110001	addi \$17,\$0...	13: addi \$s1, \$zero, 1
<input type="checkbox"/>	0x00400004	0x20120002	addi \$18,\$0...	14: addi \$s2, \$zero, 2
<input type="checkbox"/>	0x00400008	0x00112020	add \$4,\$0,\$17	16: add \$a0, \$zero, \$s1
<input type="checkbox"/>	0x0040000c	0x00122820	add \$5,\$0,\$18	17: add \$a1, \$zero, \$s2
<input type="checkbox"/>	0x00400010	0x0c100008	jal 0x00400020	18: jal soma
<input type="checkbox"/>	0x00400014	0x00000000	nop	19: nop
<input type="checkbox"/>	0x00400018	0x00000000	nop	20: nop
<input type="checkbox"/>	0x0040001c	0x00029820	add \$19,\$0,\$2	21: add \$s3, \$zero, \$v0
<input type="checkbox"/>	0x00400020	0x00851020	add \$2,\$4,\$5	28: add \$v0, \$a0, \$a1
<input type="checkbox"/>	0x00400024	0x03e00008	jr \$31	29: jr \$ra

Labels

Label	Address
(global)	
main	0x00400000
funcao_01.asm	
soma	0x00400020

☒ Data ☒ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000



0x10010000 (.data)

☒ Hexadecimal Addresses☒ Hexadecimal Values☐ ASCII

Mars Messages

Run I/O

Clear

Run speed at max (no interaction)

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000001
\$a1	5	0x00000002
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000001
\$s2	18	0x00000002
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00400014
pc		0x00400020
hi		0x00000000
lo		0x00000000



Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20110001	addi \$17,\$0...	13: addi \$s1, \$zero, 1
<input type="checkbox"/>	0x00400004	0x20120002	addi \$18,\$0...	14: addi \$s2, \$zero, 2
<input type="checkbox"/>	0x00400008	0x00112020	add \$4,\$0,\$17	16: add \$a0, \$zero, \$s1
<input type="checkbox"/>	0x0040000c	0x00122820	add \$5,\$0,\$18	17: add \$a1, \$zero, \$s2
<input type="checkbox"/>	0x00400010	0x0c100008	jal 0x00400020	18: jal soma
<input type="checkbox"/>	0x00400014	0x00000000	nop	19: nop
<input type="checkbox"/>	0x00400018	0x00000000	nop	20: nop
<input type="checkbox"/>	0x0040001c	0x00029820	add \$19,\$0,\$2	21: add \$s3, \$zero, \$v0
<input type="checkbox"/>	0x00400020	0x00851020	add \$2,\$4,\$5	28: add \$v0, \$a0, \$a1
<input type="checkbox"/>	0x00400024	0x03e00008	jr \$31	29: jr \$ra

Labels

Label	Address
(global)	
main	0x00400000
funcao_01.asm	
soma	0x00400020

☒ Data ☒ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000



0x10010000 (.data)

☒ Hexadecimal Addresses☒ Hexadecimal Values☐ ASCII

Mars Messages

Run I/O

Clear

Run speed at max (no interaction)

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000003
\$v1	3	0x00000000
\$a0	4	0x00000001
\$a1	5	0x00000002
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000001
\$s2	18	0x00000002
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffcfc
\$fp	30	0x00000000
\$ra	31	0x00400014
pc		0x00400024
hi		0x00000000
lo		0x00000000



Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20110001	addi \$17,\$0...	13: addi \$s1, \$zero, 1
<input type="checkbox"/>	0x00400004	0x20120002	addi \$18,\$0...	14: addi \$s2, \$zero, 2
<input type="checkbox"/>	0x00400008	0x00112020	add \$4,\$0,\$17	16: add \$a0, \$zero, \$s1
<input type="checkbox"/>	0x0040000c	0x00122820	add \$5,\$0,\$18	17: add \$a1, \$zero, \$s2
<input type="checkbox"/>	0x00400010	0x0c100008	jal 0x00400020	18: jal soma
<input type="checkbox"/>	0x00400014	0x00000000	nop	19: nop
<input type="checkbox"/>	0x00400018	0x00000000	nop	20: nop
<input type="checkbox"/>	0x0040001c	0x00029820	add \$19,\$0,\$2	21: add \$s3, \$zero, \$v0
<input type="checkbox"/>	0x00400020	0x00851020	add \$2,\$4,\$5	28: add \$v0, \$a0, \$a1
<input type="checkbox"/>	0x00400024	0x03e00008	jr \$31	29: jr \$ra

Labels

Label	Address
(global)	
main	0x00400000
funcao_01.asm	
soma	0x00400020

☒ Data ☒ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data)

☒ Hexadecimal Addresses☒ Hexadecimal Values☐ ASCII

Mars Messages

Run I/O

Clear

Reset: reset completed.

Run speed at max (no interaction)

Registers

Coprocc 1

Coprocc 0

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000003
\$v1	3	0x00000000
\$a0	4	0x00000001
\$a1	5	0x00000002
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000001
\$s2	18	0x00000002
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00400014
pc		0x00400014
hi		0x00000000
lo		0x00000000



Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20110001	addi \$17,\$0...	13: addi \$s1, \$zero, 1
<input type="checkbox"/>	0x00400004	0x20120002	addi \$18,\$0...	14: addi \$s2, \$zero, 2
<input type="checkbox"/>	0x00400008	0x00112020	add \$4,\$0,\$17	16: add \$a0, \$zero, \$s1
<input type="checkbox"/>	0x0040000c	0x00122820	add \$5,\$0,\$18	17: add \$a1, \$zero, \$s2
<input type="checkbox"/>	0x00400010	0x0c100008	jal 0x00400020	18: jal soma
<input type="checkbox"/>	0x00400014	0x00000000	nop	19: nop
<input type="checkbox"/>	0x00400018	0x00000000	nop	20: nop
<input type="checkbox"/>	0x0040001c	0x00029820	add \$19,\$0,\$2	21: add \$s3, \$zero, \$v0
<input type="checkbox"/>	0x00400020	0x00851020	add \$2,\$4,\$5	28: add \$v0, \$a0, \$a1
<input type="checkbox"/>	0x00400024	0x03e00008	jr \$31	29: jr \$ra

Labels

Label	Address
(global)	
main	0x00400000
funcao_01.asm	
soma	0x00400020

☒ Data ☒ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000



0x10010000 (.data)

☒ Hexadecimal Addresses☒ Hexadecimal Values☐ ASCII

Mars Messages

Run I/O

Clear

Reset: reset completed.

Run speed at max (no interaction)

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000003
\$v1	3	0x00000000
\$a0	4	0x00000001
\$a1	5	0x00000002
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000001
\$s2	18	0x00000002
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00400014
pc		0x0040001c
hi		0x00000000
lo		0x00000000



Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20110001	addi \$17,\$0...	13: addi \$s1, \$zero, 1
<input type="checkbox"/>	0x00400004	0x20120002	addi \$18,\$0...	14: addi \$s2, \$zero, 2
<input type="checkbox"/>	0x00400008	0x00112020	add \$4,\$0,\$17	16: add \$a0, \$zero, \$s1
<input type="checkbox"/>	0x0040000c	0x00122820	add \$5,\$0,\$18	17: add \$a1, \$zero, \$s2
<input type="checkbox"/>	0x00400010	0x0c100008	jal 0x00400020	18: jal soma
<input type="checkbox"/>	0x00400014	0x00000000	nop	19: nop
<input type="checkbox"/>	0x00400018	0x00000000	nop	20: nop
<input type="checkbox"/>	0x0040001c	0x00029820	add \$19,\$0,\$2	21: add \$s3, \$zero, \$v0
<input type="checkbox"/>	0x00400020	0x00851020	add \$2,\$4,\$5	28: add \$v0, \$a0, \$a1
<input type="checkbox"/>	0x00400024	0x03e00008	jr \$31	29: jr \$ra

Labels

Label	Address
(global)	
main	0x00400000
funcao_01.asm	
soma	0x00400020

☒ Data
 ☒ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000



0x10010000 (.data)

☒ Hexadecimal Addresses☒ Hexadecimal Values☐ ASCII

Mars Messages

Run I/O

Clear

Reset: reset completed.

Run speed at max (no interaction)

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000003
\$v1	3	0x00000000
\$a0	4	0x00000001
\$a1	5	0x00000002
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000001
\$s2	18	0x00000002
\$s3	19	0x00000003
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00400014
pc		0x00400020
hi		0x00000000
lo		0x00000000

```
main ( )  
{  
...  
  x = 1;  
  y = 2;  
  z = soma ( x , y );  
  k = soma (x , x );  
  m = soma (y , y );  
  x = soma ( k , m)  
}
```

Mapeamento:

$x \rightarrow \$s1$

$y \rightarrow \$s2$

$z \rightarrow \$s3$

$k \rightarrow \$s4$

$m \rightarrow \$s5$

```
int soma (int r, int s)  
{  
  return ( r + s )  
}
```

FAZER !

```

main ( )
{
...
  x = 1;
  y = 2;
  z = soma ( x , y );
  k = soma ( x , x );
  m = soma ( y , y );
  x = soma ( k , m )
}

int soma (int r, int s)
{
  return ( r + s )
}

```

Mapeamento:

```

x → $s1
y → $s2
z → $s3
k → $s4
m → $s5

```

```

main:

addi $s1, $zero, 1
addi $s2, $zero, 2

add $a0, $zero, $s1
add $a1, $zero, $s2
jal soma
nop

add $s3, $zero, $v0
add $a0, $zero, $s1
add $a1, $zero, $s1
jal soma
nop

```

```

add $s4, $zero, $v0
add $a0, $zero, $s3
add $a1, $zero, $s4
jal soma
nop

add $s5, $zero, $v0
add $a0, $zero, $s4
add $a1, $zero, $s5
jal soma
nop

add $s1, $zero, $v0

soma:
add $v0, $a0, $a1
jr $ra

```

```
int mult (int mando, int mdor)
{
    int m;
    m = mando * mdor;
    return m;
}
```

Fazer a função *mult*

- *só sabemos somar!*


```
main ( )
{
    int i, j, k;
    i = mult ( j , k );
...
}
```

```
int mult (int mando, int mdor)
{
    int produto;
    produto = 0;
    while ( mdor > 0 )
    {
        produto = produto + mando;
        mdor = mdor -1;
    }
    return produto;
}
```

Exercício:

Passar para o MIPS

Mapeamento:

i em \$s0

j em \$s1

k em \$s2

Restante você escolhe

Mapeamento: i em \$s0, j em \$s1 e k em \$s2

■ start:

Transfere argumentos e chama função

```
add $a0,$s1,$0      # arg0 = j
add $a1,$s2,$0      # arg1 = k
jal mult            # call mult
```

```
main ( )
{
    int i, j, k;
    i = mult ( j , k );
    ...
}
```

Mapeamento: i em \$s0, j em \$s1 e k em \$s2

main ()

```
{  
    int i, j, k;  
    i = mult ( j , k );  
    ...  
}
```

■ start:

Transfere argumentos e chama função

```
add $a0,$s1,$0      # arg0 = j  
add $a1,$s2,$0      # arg1 = k  
jal mult            # call mult
```

Salva valor de retorno na variável estática \$s0

```
add $s0,$v0,$0      # i = mult()
```

-
mult:

add \$t0,\$0,\$0 # prod (\$t0)=0

Loop:

slt \$t1,\$0,\$a1 # mlr(\$a1) > 0?

beq \$t1,\$0,Fim # não => vá para Fim

add \$t0,\$t0,\$a0 # sim: prod += mc(\$a0)

addi \$a1,\$a1,-1 # mlr -= 1

j Loop # goto Loop

Fim:

add \$v0,\$t0,\$0 # \$v0 = prod

jr \$ra # retorna

```
int mult (int mc, int mlr)
{
    int prod;
    prod = 0;
    while ( mlr > 0 )
    {
        prod = produto + mc;
        mdor = mdor -1;
    }
    return prod;
}
```



Funções Aninhadas (1/2)

```
int faz1 (int x , int y )  
{  
    return ( faz2 ( x , y ) )  
}
```

- Alguma coisa chamada **faz1**, que **chama** **faz2**
- Há um valor em **\$ra** que **faz1** quer pular de volta, mas **será sobrescrito** pela chamada a **faz2** .
- Precisa **salvar o endereço de retorno** de **faz1** antes de chamar **faz2** .



Funções Aninhadas (2/2)

- Em geral, pode ser necessário **salvar** algum outro **registrador** além de \$ra.
- Quando um programa C está rodando, existem 3 **importantes áreas de memória** que são alocadas:
 - **Static** (alocação estática): Variáveis declaradas uma vez por programa, deixam de existir somente quando a execução termina.
 - **Heap** (alocação dinâmica): Variáveis declaradas dinamicamente.
 - **Stack** (pilha): Espaço a ser utilizado pela função durante sua execução; é aqui que podemos **salvar** os valores dos **registradores**.

Alocação de Memória em C

Endereço
 ∞

Stack

Espaço para informações
salvas pela função

\$sp →
stack
pointer

Heap

Espaço explicitamente criado
e.g., malloc(); ponteiros C

Static

Variáveis declaradas
uma vez por programa

Code

Programa

0



Usando a Pilha (1/2)

- Nós temos um registrador **\$sp** que **sempre aponta** para o **último espaço utilizado na pilha**.
- Para utilizar a pilha:
 - 1º **decrementamos** o ponteiro **\$sp** de 4 bytes
 - 2º **preenchemos** os 4 bytes da pilha com a informação.
- Então, como compilamos isto?

Vamos usar a -> \$s1

main

```
{  
    int a;  
    a = 0;  
    a = um();  
    a = a + 1;  
}
```

main:

```
int um ( )  
{  
    return ( 1 );  
}
```

um:

Vamos usar a -> \$s1

main

```
{  
    int a;  
    a = 0;  
    a = um();  
    a = a + 1;  
}
```

```
int um ( )  
{  
    return ( 1 );  
}
```

main:

```
addi $s1, $zero, 0  
jal um  
nop  
addi $s1, $zero, $v0  
addi $s1, $s1, 1
```

um:

```
addi $v0, $zero, 1  
jr $ra  
nop
```

main

```
{  
    int a;  
    a = 0;  
    a = dois();  
    a = a + 1;  
}
```

```
int dois ( )  
{  
    return ( um( ) + 1 );  
}
```

```
int um ( )  
{  
    return ( 1 );  
}
```

Vamos usar a -> \$s1

main:

```
100    addi $s1, $zero, 0  
104    jal dois  
108    nop  
112    addi $s1, $v0, 0  
116    addi $s1, $s1, 1
```

dois:

```
200
```

um:

```
500    addi $v0, $zero, 1  
504    jr $ra  
508    nop
```

\$ra



main

```
{  
    int a;  
    a = 0;  
    a = dois();  
    a = a + 1;  
}
```

int dois ()

```
{  
    return ( um( ) + 1 );  
}
```

int um ()

```
{  
    return ( 1 );  
}
```

Vamos usar a -> \$s1

main:

```
100    addi $s1, $zero, 0  
104    jal dois  
108    nop  
112    addi $s1, $v0, 0  
116    addi $s1, $s1, 1
```

dois:

```
200    jal um  
204    addi $v0, $v0, 1  
208    jr $ra
```

um:

```
500    addi $v0, $zero, 1  
504    jr $ra  
508    nop
```

\$ra



Vamos usar a -> \$s1

main

```
{  
    int a;  
    a = 0;  
    a = dois();  
    a = a + 1;  
}
```

int dois ()

```
{  
    return ( um( ) + 1 );  
}
```

int um ()

```
{  
    return ( 1 );  
}
```

main:

```
100    addi $s1, $zero, 0  
104    jal dois  
108    nop  
112    addi $s1, $v0, 0  
116    addi $s1, $s1, 1
```

dois:

```
200    addi $sp, $sp, -4  
204    sw $ra, 4 ($sp)  
208    jal um  
212    addi $v0, $v0, 1  
216    lw $ra, 4 ($sp)  
220    addi $sp, $sp, 4  
224    jr $ra
```

um:

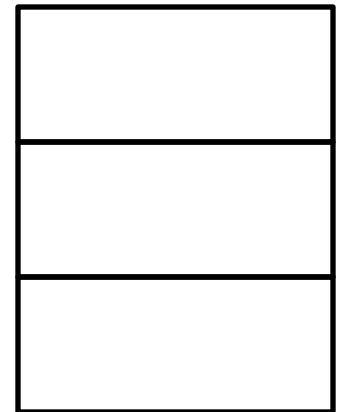
```
500    addi $v0, $zero, 1  
504    jr $ra  
508    nop
```

\$ra



stack

\$sp



Vamos usar a->\$s1, b->\$s2

main

```
{  
    int a,b;  
    a = 1;  
    b = soma1(a);  
    a = a + b;  
}
```

```
int soma1 ( int x )  
{  
    return ( x + 1 );  
}
```

main:

```
soma1:  
addi $v0, $a0, 1  
jr $ra  
nop
```

Vamos usar a->\$s1, b->\$s2

main

```
{  
  int a,b;  
  a = 1;  
  b = soma1(a);  
  a = a + b;  
}
```

```
int soma1 ( int x )  
{  
  return ( x + 1 );  
}
```

main:

```
addi $s1, $zero, 1  
addi $a0, $s1, 0  
jal soma1  
nop  
addi $s2, $v0, 0  
add $s1, $s1, $s2
```

soma1:

```
addi $v0, $a0, 1  
jr $ra  
nop
```

main

```
{  
    int a,b;  
    a = 1;  
    b = soma3(a);  
    a = a + b;  
}
```

```
int soma3 (int y )  
{  
    return ( soma1(y+1) + 1);  
}
```

```
int soma1 ( int x )  
{  
    return ( x + 1 );  
}
```

Vamos usar a->\$s1, b->\$s2

```
main:  
addi $s1, $zero, 1  
addi $a0, $s1, 0  
jal soma3  
nop  
addi $s2, $v0, 0  
add $s1, $s1, $s2
```

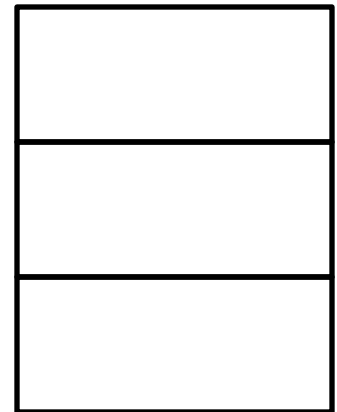
```
soma3:  
200
```

\$ra



stack

\$sp



```
soma1:  
500 addi $v0, $a0, 1  
504 jr $ra  
508 nop
```


main

```
{  
    int a,b;  
    a = 1;  
    b = soma3(a);  
    a = a + b;  
}
```

int soma3 (int y)

```
{  
    return ( soma1(y+1) + 1);  
}
```

int soma1 (int x)

```
{  
    return ( x + 1 );  
}
```

Vamos usar a->\$s1, b->\$s2

main:

```
addi $s1, $zero, 1  
addi $a0, $s1, 0  
jal soma3  
nop  
addi $s2, $v0, 0  
add $s1, $s1, $s2
```

soma3:

```
200 addi $sp, $sp, -4  
204 sw $ra, 4 ($sp)  
208 addi $a0, $a0, 1  
212 jal soma1  
216 addi $v0, $v0, 1  
220 lw $ra, 4 ($sp)  
220 addi $sp, $sp, 4  
224 jr $ra
```

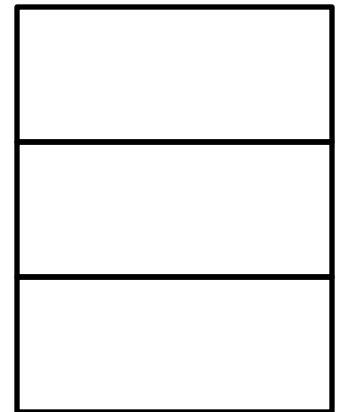
\$sp

soma1:

```
500 addi $v0, $a0, 1  
504 jr $ra  
508 nop
```

\$ra

stack



main

```
{  
    int a,b;  
    a = 1;  
    b = soma3(a);  
    a = a + b;  
}
```

```
int soma3 (int y )  
{  
    return ( soma1(y+1) + y);  
}
```

```
int soma1 ( int x )  
{  
    return ( x + 1 );  
}
```

Vamos usar a->\$s1, b->\$s2

```
main:  
addi $s1, $zero, 1  
addi $a0, $s1, 0  
jal soma3  
nop  
addi $s2, $v0, 0  
add $s1, $s1, $s2
```

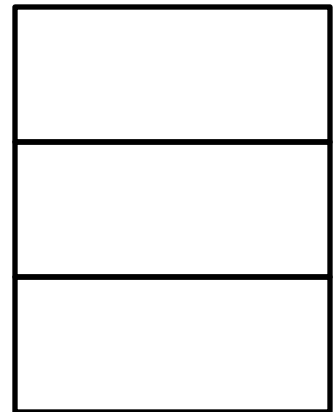
```
soma3:  
200
```

\$ra



stack

\$sp



```
soma1:  
500 addi $v0, $a0, 1  
504 jr $ra  
508 nop
```

main

```
{
    int a,b;
    a = 1;
    b = soma3(a);
    a = a + b;
}
```

int soma3 (int y)

```
{
    return ( soma1(y+1) + y);
}
```

int soma1 (int x)

```
{
    return ( x + 1 );
}
```

Vamos usar a->\$s1, b->\$s2

main:

```
addi $s1, $zero, 1
addi $a0, $s1, 0
jal soma3
nop
addi $s2, $v0, 0
add $s1, $s1, $s2
```

soma3:

```
200 addi $sp, $sp, -8
204 sw $ra, 4 ($sp)
208 sw $a0, 8 ($sp)
212 addi $a0, $a0, 1
216 jal soma1
220 lw $a0, 8 ($sp)
224 add $v0, $v0, $a0
228 lw $ra, 4 ($sp)
220 addi $sp, $sp, 8
224 jr $ra
```

\$sp

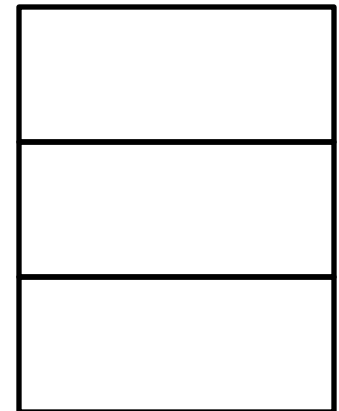
soma1:

```
500 addi $v0, $a0, 1
504 jr $ra
508 nop
```

\$ra



stack



```
main
```

```
{  
    int a=2, b=3, c;  
    c = SumSquare (a,b);  
}
```

```
int SumSquare(int x, int y)  
{  
    return ( mult (x , x ) + y );  
}
```

```
int mult (int m, int n )  
{  
    return ( m * n);  
}
```

Usando a Pilha (2/2)

- Compile manualmente

sumSquare:

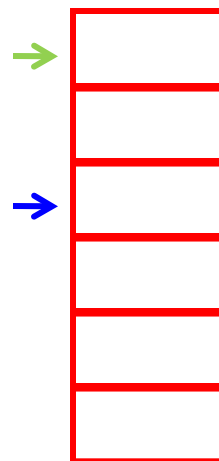
(salva end. retorno e argumento de SumSquare)

```
addi    $sp,$sp,-8
```

reserva espaço na pilha

Função SumSquare

```
int sumSquare(int x, int y)
{
    return mult(x,x)+ y;
}
```



Usando a Pilha (2/2)

- Compile manualmente

sumSquare:

(salva end. retorno e argumento de SumSquare)

addi \$sp, \$sp, -8

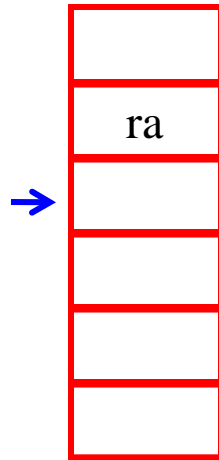
reserva espaço na pilha

sw \$ra, 4(\$sp)

*# **salva reg. end. retorno***

Função SumSquare

```
int sumSquare(int x, int y)
{
    return mult(x,x)+ y;
}
```



Usando a Pilha (2/2)

- Compile manualmente

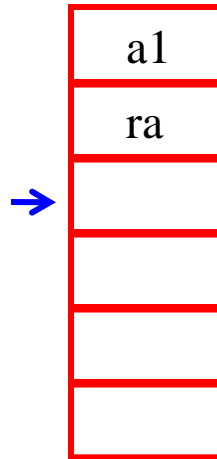
sumSquare:

(salva end. retorno e argumento de SumSquare)

```
addi    $sp,$sp,-8      # reserva espaço na pilha
sw       $ra, 4($sp)     # salva reg. end. retorno
sw       $a1, 8($sp)     # salvar argumento y
```

Função SumSquare

```
int sumSquare(int x, int y)
{
    return mult(x,x)+ y;
}
```



Usando a Pilha (2/2)

- Compile manualmente

sumSquare:

(salva end. retorno e argumento de SumSquare)

```
addi    $sp,$sp,-8           # reserva espaço na pilha
```

```
sw      $ra, 4($sp)          # salva reg. end. retorno
```

```
sw      $a1, 8($sp)          # salvar argumento y
```

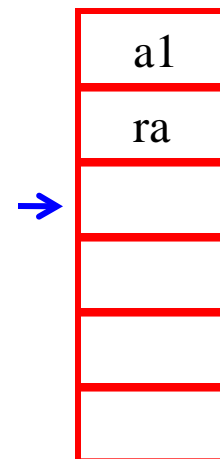
(Transfere arg. de mult e chama função mult)

```
add     $a1,$a0,$zero        # transfere arg.x de mult(x,x)
```

```
jal     mult                  # chama mult
```

Função SumSquare

```
int sumSquare(int x, int y)
{
    return mult(x,x)+ y;
}
```



Usando a Pilha (2/2)

- Compile manualmente

sumSquare:

(salva end. retorno e argumento de SumSquare)

```
addi    $sp,$sp,-8      # reserva espaço na pilha
sw      $ra, 4($sp)     # salva reg. end. retorno
sw      $a1, 8($sp)     # salvar argumento y
```

(Transfere arg. de mult e chama função mult)

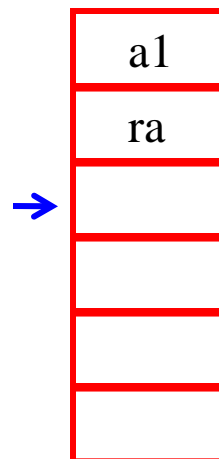
```
add     $a1,$a0,$zero   # transfere arg.x de mult(x,x)
jal     mult            # chama mult
```

(restaura arg. De SumSquare e executa operação)

```
lw      $a1, 8($sp)     # restaura arg. y
add     $v0,$v0,$a1     # mult()+y
```

Função SumSquare

```
int sumSquare(int x, int y)
{
    return mult(x,x)+ y;
}
```



Usando a Pilha (2/2)

■ Compile manualmente

sumSquare:

(salva end. retorno e argumento de SumSquare)

```
addi    $sp,$sp,-8      # reserva espaço na pilha
sw      $ra, 4($sp)     # salva reg. end. retorno
sw      $a1, 8($sp)     # salvar argumento y
```

(Transfere arg. de mult e chama função mult)

```
add     $a1,$a0,$zero   # transfere arg.x de mult(x,x)
jal     mult            # chama mult
```

(restaura arg. De SumSquare e executa operação)

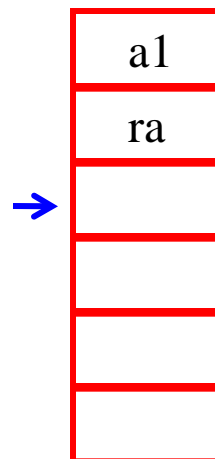
```
lw      $a1, 8($sp)     # restaura arg. y
add     $v0,$v0,$a1     # mult()+y
```

(restaura end. retorno de SumSquare e a pilha)

```
lw      $ra, 4($sp)     # restaura end. retorno
addi    $sp,$sp,8       # restaura pilha
jr      $ra             # retorna para prog. principal
```

Função SumSquare

```
int sumSquare(int x, int y)
{
    return mult(x,x)+ y;
}
```





Passos para fazer uma chamada de função

- 1) **Salvar** os valores necessários na pilha.
- 2) **Atribuir** argumento(s), se existir(em).
- 3) **Chamar** a função `jal`
- 4) **Restaurar** os valores da pilha.



Regras para Funções

- **Chamada** com uma instrução `jal` **retorna** com uma `jr $ra`
- Aceita **até 4 argumentos** em `$a0`, `$a1`, `$a2` e `$a3`
- **Valor** de **retorno** sempre está **em \$v0** (e se necessário em `$v1`)
- Deve seguir as **convenções de registradores** (mesmo em funções que somente você vai chamar)! Então, quais são elas?



Registradores MIPS

A constante 0	\$0	\$zero
Valores de Retorno	\$2-\$3	\$v0-\$v1
Argumentos	\$4-\$7	\$a0-\$a3
Temporários	\$8-\$15	\$t0-\$t7
Salvos	\$16-\$23	\$s0-\$s7
Mais Temporários	\$24-\$25	\$t8-\$t9
Stack Pointer	\$29	\$sp
Return Address	\$31	\$ra

- Em geral, você pode utilizar ou o nome ou o número. Os nomes deixam o código mais fácil de se ler.



Convenções de Registradores (1/5)

- Caller (*chamador*): a função que faz a chamada
- Callee (*função*): a função sendo chamada
- Quando a *função* retorna da execução, o *chamador* precisa saber quais **registradores podem ter mudado e quais não mudaram**.
- **Convenções de Registradores**: Um conjunto geralmente aceito de regras de quais registradores não mudam após uma chamada de função (já) e quais podem ter sido mudados.



Convenções de Registradores (2/5)

- \$0: Nunca muda. Sempre 0.
- \$v0-\$v1: Muda. Estes são esperados conter novos valores.
- \$a0-\$a3: Muda. Estes são registradores de argumentos voláteis.
- \$t0-\$t9: Muda. Por isso eles são chamados temporários: qualquer função pode mudá-los a qualquer momento.

Convenções de Registradores (3/5)

- `$s0-$s7`: **Sem mudança**. Muito importante, por isso eles são chamados registradores salvos. Se **a função chamada** (*callee*) muda estes registradores de algum modo, ela **deve restaurar os valores originais antes de retornar**.
- `$sp`: **Sem mudança**. O ponteiro da pilha deve apontar para o mesmo lugar antes e depois de uma chamada de `jal` ou então a função chamadora (*caller*) não será capaz de restaurar os valores da pilha. **A função chamada deve restaurar os valores originais antes de retornar**.
- `$ra`: Muda. A chamada a `jal` vai mudar este registrador por si mesma.



Convenções de Registradores (4/5)

- O que estas convenções significam?
 - Se a função A chama B
 - **a função A deve salvar qualquer registrador temporário** que esteja utilizando na pilha antes de fazer uma chamada `jal`.
 - **A função B deve salvar qualquer registrador S (salvos - `sp`, `$s0-$s7`)** que ela pretende utilizar antes de modificar seus valores.
 - Lembre-se: *Caller/callee* precisam salvar somente os registradores temporários que eles estejam utilizando, não todos os registradores.



Convenções de Registradores (5/5)

- Note que, se ***callee*** vai utilizar algum registrador S, ela **deve**:
 - **Salvar** aqueles **registradores S** na pilha.
 - Utilizar os registradores
 - Restaurar os registradores S da pilha.
 - `jr $ra`
- Com os **registradores temporários**, a *calle* **não precisa salvar** na pilha.
- Portanto, a ***caller*** deve **salvar** aqueles **registradores temporários que quer preservar** através da chamada.

Compile o seguinte problema:

```
{  
int VetA [ ... ];  
int i, j;  
  
...  
j = VetA [ i ];  
j = soma1 ( j );  
VetA [ i ] = j;  
}
```

```
int soma1 ( int x)  
{  
    int aux;  
    aux = x + 1;  
    return ( aux );  
}
```

Exemplo: Compile isto (1/2)

- Fatorial

Função recursiva

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

- Colocar na **pilha**, **todos** os **registradores** que **necessitam ser preservados**:
 - \$a0-\$a3, \$t0-\$t9, \$ra, \$s0, ajustar \$sp.
 - No retorno: restaurar regs, restaurar \$sp.

```

.text
.globl main
main:
addi $s0, $0, 3
add $a0, $s0, $0
jal fact
nop
add $s0, $0, $v0
nop
addi $v0, $zero, 10
syscall

```

```

fact:
addi $sp, $sp, -8 # reserva espaço na pilha (2 palavras)
sw $ra, 4($sp) # salva end. retorno na pilha
sw $a0, 0($sp) # salva argumento (n)
slti $t0, $a0, 1 # n<1?
beq $t0, $zero, L1 # não: (n>=1) vá para L1
addi $v0, $zero, 1 # n=0: $v0=1
addi $sp, $sp, 8 # restaura a pilha
jr $ra # retorna da função
nop

L1: addi $a0, $a0, -1 # n=n-1 => $a0=$a0-1
jal fact # chama fact(n-1)=> $v0=fact($a0)
nop
lw $a0, 0($sp) # restaura argumento n ($a0)
lw $ra, 4($sp) # restaura end. retorno
addi $sp, $sp, 8 # restaura pilha
mult $v0, $a0 # $v0=n*fact(n-1)
mflo $v0
jr $ra # retorna da função
nop

```



Coisas para Lembrar (1/2)

- Funções são chamadas com `jal`, e retornam com `jr $ra`.
- A **pilha** é sua amiga: use-a para **salvar** qualquer coisa que precise. Apenas assegure-se de **deixá-la como a achou**.
- **Convenções de Registradores**: Cada registrador tem um propósito e limites no seu uso. Aprenda-os e siga-os.



Coisas para Lembrar (2/2)

- Instruções que nós conhecemos até agora:

Aritmética: `add`, `addi`, `sub`

Memória: `lw`, `sw`

Decisão: `beq`, `bne`, `slt`

Desvios incondicionais (pulos): `j`, `jal`, `jr`

- Registradores que nós conhecemos até agora:

- `$zero`, `$v0-$v1`, `$a0-$a3`, `$t0-$t7`, `$s0-$s7`, `$t8-$t9`, `$sp`, `$ra`