

Existem pelo menos duas convenções para chamadas de procedimentos:

1. MIPS Calling Convention
2. GCC Calling Convention

(não usa \$fp) \$fp → \$30 → \$58  
Se não usa \$fp ele pode ser usado como um saved register.

## Regras:

① A primeira instrução na função manipula \$sp para alocar F words na pilha.

addi \$sp, \$sp, -F\*4 ou subu \$sp, \$sp, F\*4

Onde:

$$F = A + L + P + 1 \text{ (para } \$ra, \text{ e necessário)} + 1 \text{ (para } \$fp \text{ se está em uso)} + 2 \text{ (para } \$gp) + \text{padding}$$

F é o tamanho do frame em words.

Corolário 1: \$sp não é modificado em qualquer outro ponto a não ser na primeira e na última instruções na função.

(Para alocação do frame e para a dealocação do mesmo).

② Se você quer usar algum dos registradores que devem ser preservados em qualquer lugar durante a execução da função, você deve armazenar eles na pilha quando entrar na função (não quando for usá-los).

Assumindo que os registradores \$s0, \$s1 e \$s2 são utilizados na função. Assim, eles precisam ser salvos na pilha como os registradores \$ra e \$sp.

sw \$ra, (F-1)\*4(\$sp)

# Não precisa guardar \$ra para funções que não fazem chamadas a outras procedimentos

sw \$fp, (F-2)\*4(\$sp)

# Salva os registradores.

sw \$s2, (F-3)\*4(\$sp)

sw \$s1, (F-4)\*4(\$sp)

sw \$s0, (F-5)\*4(\$sp)

③ Após sair os registradores que devem ser preservados, copie \$sp para \$fp. De agora em diante até o retorno você pode usar deslocamentos a partir de \$fp. Para acessar os locais na pilha.

Salve e restaure variáveis locais se for preciso após esse ponto usando \$fp e os ponteiros alocados para elas na pilha.

④ Argumentos podem ser passados para as funções usando-se somente \$a0-\$a3 e argumentos extras são colocados na pilha. Não pode usar o valor mantido em outros registradores exceto \$sp, \$fp, \$gp e \$ra (não use \$v0, \$s0). Para passar argumentos para a função.

O argumento #4 (quinto argumento) para a função corrente está disponível em  $(F+4)*4($fp)$ .

O #5 estará em  $(F+5)*4($fp)$  e assim por diante.

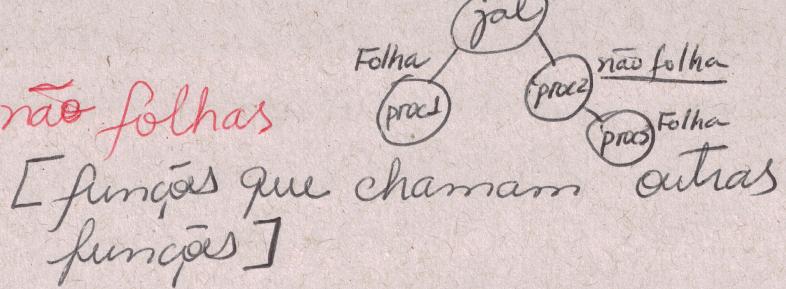
Você pode armazenar o argumento 0 (#0) de \$a0 nos locais reservados na pilha na posição/endereço  $\cancel{(F+0)}*4($fp)$ . Argumento #1 em  $(F+1)*4($fp)$  e assim por diante, caso você queira. (uma vez que os registradores \$a0-\$a3 já armazenam esses valores).

⑤ Informações retornadas pela função (retorno de valores) podem ser somente feitos utilizando-se \$v0 - \$v1. Não utilize \$a0 - \$a3, os registradores de argumentos.

⑥ Você somente pode entrar na função em seu início. Não faça um jal para o meio do código da função. (Go To).

⑦ Você pode ter somente um jr \$ra em cada função. E isso deve ser a última instrução na função, após as

## Regras extras para funções não folhas



[funções que chamam outras funções]

⑧ Olhe para a declaração de todos os procedimentos que a função possa chamar. Encontre a que tem o maior número de argumentos, A. Você deve alocar  $A * 4$  bytes após entrar na função. A deve ser no mínimo 4.

⑨ Passando somente /ou até 4 argumentos: Coloque os argumentos em \$a0 - \$a3.

Passando mais que 4 argumentos: Coloque os 4 primeiros em \$a0 - \$a3. Os demais argumentos devem ser colocados na pilha.

Supondo que a função utilize 6 argumentos que estão em \$\$s0 - \$s5:

move \$a0, \$s0 # e 0(\$fp) está reservado para este argumento na pilha.

move \$a1, \$s1 # e 4(\$fp) está reservado na pilha.

move \$a2, \$s2 # e 8(\$fp)

move \$a3, \$s3 # e 12(\$fp)

sw \$s4, 16(\$fp) # armazena o 5º argumento na pilha.  
# sempre neste localização.

sw \$s5, 20(\$fp) # 6º argumento na pilha.

⑩ Você deve preservar o registrador \$ra porque a função chamada já destruirá o valor e ele é um dos registradores que deve ser preservado. Veja regra ②.

```

.file      1 "teste-001.c"
.section .mdebug.abi32
.previous
.gnu_attribute 4, 1
.abicalls
.text
.align    2
.globl    magica
.set nomips16
.ent magica
.type     magica, @function

magica:
.frame   $fp,24,$ra          # vars= 8, regs= 1/0, args= 0, gp= 8
.mask    0x40000000, -4
.fmask   0x00000000, 0
.set noreorder
.set nomacro
addiu   $sp,$sp,-24
sw      $fp,20($sp)
move   $fp,$sp
sw      $a0,24($fp) } coloca os args na pilha.
sw      $a1,28($fp) } na pilha.
sw      $a2,32($fp) } args
sw      $a3,36($fp)
sw      $0,8($fp) <res>
lw      $3,24($fp) a0->x
lw      $2,28($fp) a1->y
nop
addu   $3,$3,$2 $3<-x+y
lw      $a0,32($fp) z.
lw      $2,36($fp) w
nop
addu   $2,$a0,$2 $2<-z+w
subu   $3,$3,$2 $0<-(x+y)-(z+w)
lw      $2,40($fp) t
nop
addu   $2,$3,$2 $((x+y)-(z+w))+t
sw      $2,8($fp) res <resultado>
lw      $2,8($fp) $V0 <res>
move   $sp,$fp sp <- fp
lw      $fp,20($sp) restaura fp antigo
addiu  $sp,$sp,24 restaura a pilha. (elimina o frame de magica)
jr      $ra retorna .
nop

```

*(x+y)-(z+w)*

*res <resultado>*

*\$V0 <res>*

*sp <- fp*

*restaura fp antigo*

*restaura a pilha. (elimina o frame de magica)*

*retorna .*

```

.set macro
.set reorder
.end magicala
.size magicala, .-magicala
.align 2
.globl main
.set nomips16
.ent main
.type main, @function

```

### main:

```

.frame $fp,64,$ra      # vars= 24, regs= 2/0, args= 24, gp= 8
.mask 0xc00000000,-4
.fmask 0x00000000,0
.set noreorder
.set nomacro
addiu   $sp,$sp,-64
sw      $ra,60($sp)
sw      $fp,56($sp)
move   $fp,$sp
li     $2,2          # 0x2
sw      $2,32($fp)    a=2
li     $2,3          # 0x3
sw      $2,36($fp)    b=3
li     $2,4          # 0x4
sw      $2,40($fp)    c=4 $fp→$sp →
li     $2,5          # 0x5
sw      $2,44($fp)    d=5
li     $2,6          # 0x6
sw      $2,48($fp)    e=6
sw      $0,52($fp)    f=0
lw      $2,48($fp)    $2<[e]
nop
sw      $2,16($sp)
lw      $a0,32($fp)
lw      $a1,36($fp)
lw      $a2,40($fp)
lw      $a3,44($fp)
.option pic0
jal   magicala
nop
.option pic2
sw      $2,52($fp)    f←$r0 (res de mágica)
move   $2,$0          r0←0
move   $sp,$fp          sp←fp
lw      $ra,60($sp)    restaura fp
lw      $fp,56($sp)    restaura fp
addiu  $sp,$sp,64      elimina o frame
jr      $ra              retorna $0
nop
.set macro
.set reorder
.end main
.size main, .-main
.ident "GCC: (GNU) 4.8.2"

```

a,b,c,d,e,f  
 $6 \times 4 = 24$  bytes  
 $\text{ra e fp} : 2 \text{regs.}$   
 $5 \text{args pl a função + 1 conforme} = 6 \times 4 = 24$   
 $\frac{24}{50} \quad \frac{2}{50} \quad \frac{50}{58}$   
 $\frac{24}{50} \quad \frac{8}{58}$   
 $\frac{64}{16}$   
 $F = A + L + P + L(\text{ra}) + L(\text{fp}) + 2(\text{gp}) + \text{padding}$   
 $5 + 6 + 0 + 1 + 1 + 2 + 0$

$F = A + L + P + L(\text{ra}) + L(\text{fp}) + 2(\text{gp}) + \text{padding}$   
 $5 + 1 + 0 + 1 + 1 + 2 + 0$

00	frame magicala
44	
88	
1212	
1616	
2020	
x [a0]	0 24
y [a1]	4 28
z [a2]	8 32
w [a3]	12 36
t rarge	16 40
	20
Egp	24
Egp	28
a:2	32
b:3	36
c:4	40
d:5	44
e:6	48
f:0	52
[fp]	56
[ra]	60

pilha →  
 a0 a1 a2 a3 →  
 (a,b,c,d,e)

f ← \$r0 (res de mágica)  
 r0 ← 0  
 sp ← fp  
 restaura fp  
 restaura fp  
 elimina o frame  
 retorna \$0  
 Fim!  
 \$sp →

padding: 0  
 preserved registers: 0

# Factorial

# + Pilha

\$fp → \$sp →

v0: 1

[fp]  
[ra]

a0: 1

\$fp → \$sp →

v1 ← v0  
v1: 1  
v2 ≤ 32(fp)

[fp]  
[ra]

a0: 2

\$fp → \$sp →

v1: 2 ≤ v0: 2  
v0: 3 ≤ 32(fp)  
v0: 6

[fp]  
[ra]

a0: 3

\$fp → \$sp →

v1: 6 <= v0: 6  
v0: 4 ≤ 32(fp)  
v0: 24

[fp]  
[ra]

a0: 4

\$fp → \$sp →

v1: 24 <= v0: 24  
v0: 5  
v0: 120

[fp]  
[ra]

a0: 5

\$fp → \$sp →

v0: 120

[fp]  
[ra]

res 0 120

padding

[fp]  
[ra]

\$sp →

0  
4  
8  
12  
16  
20  
24  
28  
32

0  
4  
8  
12  
16  
20  
24  
28  
32

0  
4  
8  
12  
16  
20  
24  
28  
32

0  
4  
8  
12  
16  
20  
24  
28  
32

0  
4  
8  
12  
16  
20  
24  
28  
32

0  
4  
8  
12  
16  
20  
24  
28  
32

0  
4  
8  
12  
16  
20  
24  
28  
32

0  
4  
8  
12  
16  
20  
24  
28  
32

frame  
factorial (1)  
a0: 1

frame  
factorial (2)  
a0: 2 → 1

frame  
factorial (3)  
a0: 3 → 2

frame  
factorial (4)  
a0: 4 → 3

frame  
factorial (5)  
a0: 5 → 4

frame do  
main  
a0: 5

# Fatorial (livro) (B-648)

①

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

```
int main (){  
    printf ("The factorial of 10 is %d\n", fact(10));  
}
```

• data \$LC: ascii "The factorial of 10 is %d\n\000"

• text

• globl main

main:  
 subu \$sp,\$sp,32 # cria frame main : 32  
 sw \$ra, 20(\$sp) # salva \$ra SO  
 sw \$fp, 16(\$sp) # Salva frame pointer  
 addiu \$fp,\$sp,28 # Prepara o frame pointer.  
 li \$a0, 10 # Argumento em \$a0.  
 jal fact  
 la \$a0, \$LC # Carrega o endereço da string.  
 move \$a1, \$v0 # Move resultado de fact para \$a1.  
 jal printf  
 lw \$ra, 20(\$sp) # Restaura endereço de retorno.  
 lw \$fp, 16(\$sp) # Restaura frame pointer.  
 addiu \$sp,\$sp,32 # Remove frame julta.  
 jr \$ra # Retorna p/ o chamador do programa (SO).

fact :

subu \$sp, \$sp, 32 # Cria frame na pilha.  
 sw \$ra, 20(\$sp) # Salva endereço de retorno.  
 sw \$fp, 16(\$sp) # Salva frame pointer.  
 addiu \$fp, \$sp, 28 # Prepara frame pointer.  
 sw \$a0, 0(\$fp) # Salva argumento (n).

lw \$v0, 0(\$fp) # Carrega n  
 bgtz \$v0, \$L2 # Desvia se n > 0;  
 li \$v0, 1 # Retorna 1.  
 jr \$L1 # Desvia o código de retorno.

\$L2:

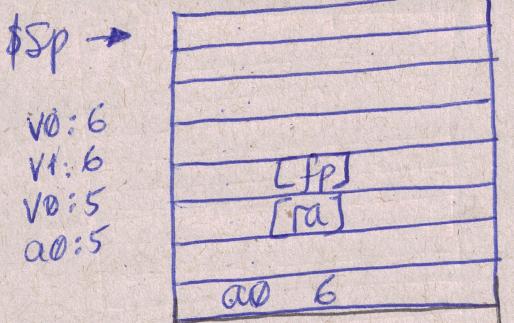
lw \$v1, 0(\$fp) # Carrega n.  
 subu \$v0, \$v1, 1 # Calcula n-1.  
 move \$a0, \$v0 # Move valor para \$a0.  
 jal fact # Chama função de fatorial.  
 lw \$v1, 0(\$fp) # Carrega n.  
 mul \$v0, \$v0, \$v1 # Calcula fact() \* n.

\$L1:

lw \$ra, 20(\$sp) # Restaura \$ra.  
 lw \$fp, 16(\$sp) # Restaura \$fp.  
 addiu \$sp, \$sp, 32 # Remove o frame de pilha.  
 jr \$ra

(3)

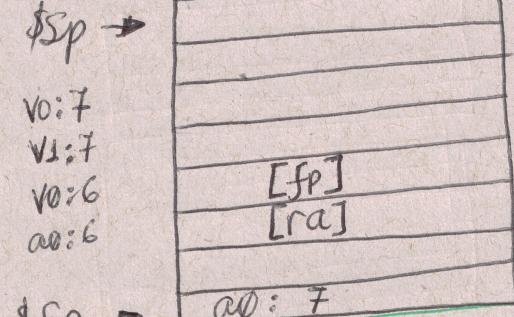
$V1: 6$   
 $V0: 6 * 120$   
 $V0: 720$   
 $ra \leftarrow$   
 $fp \leftarrow$



0  
4  
8  
12  
16  
20  
24  
28

frame  
fact(6)

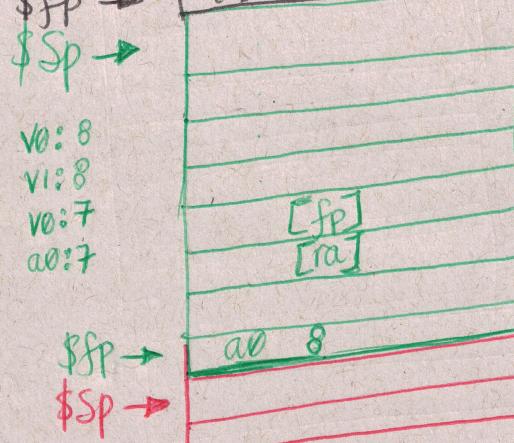
$V1: 7$   
 $V0: 7 * 720$   
 $V0: 5040$   
 $ra \leftarrow$   
 $fp \leftarrow$



0  
4  
8  
12  
16  
20  
24  
28

frame  
fact(7)

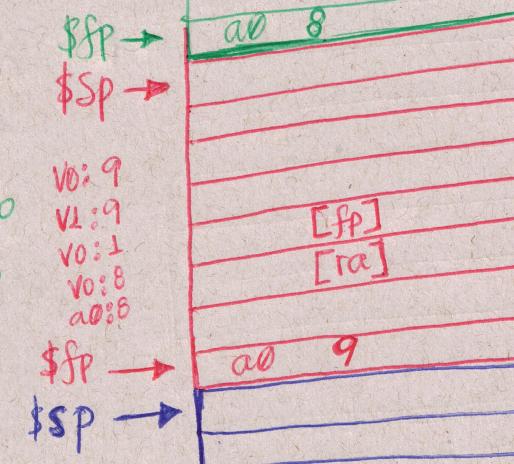
$V1: 8$   
 $V0: 8 * 5040$   
 $V0: 40320$   
 $ra \leftarrow$   
 $fp \leftarrow$



0  
4  
8  
12  
16  
20  
24  
28

frame  
fact(8)

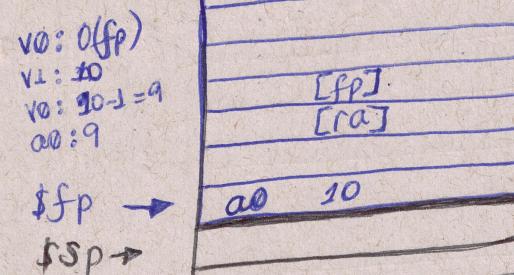
$V1: 9$   
 $V0: 9 * 40320$   
 $V0: 362880$   
 $ra \leftarrow$   
 $fp \leftarrow$



0  
4  
8  
12  
16  
20  
24  
28

frame  
fact(9)

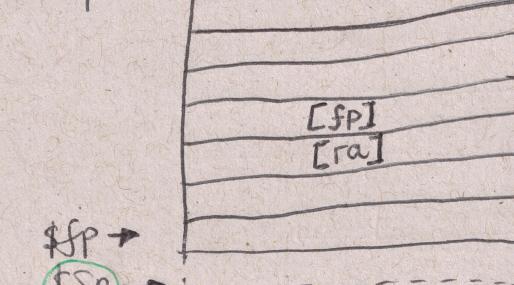
$V1: 10$   
 $V0: 10 * 362880$   
 $V0: 3628800$   
 $ra \leftarrow$   
 $fp \leftarrow$



0  
4  
8  
12  
16  
20  
24  
28

frame  
fact(10)

$a1 \leftarrow V0$   
 $printf.$   
 $ra \leftarrow$   
 $fp \leftarrow$



0  
4  
8  
12  
16  
20  
24  
28

frame  
main  
ad: 10

↓ Tim

início ↑

\$sp →		4 8 12 16 20 24 28
V0: 0	[fp]	
V0: 1	[ra]	
ra: [ra]		
fp: [fp]		
\$fp →	a0 0	
\$sp →		0 4 8 12 16 20 24 28
V1: 1		
V0: 1 * 1		
ao: 0		
ra: [ra]	[fp]	
fp: [fp]	[ra]	
\$fp →	a0 1	
\$sp →		0 4 8 12 16 20 24 28
V1: 2		
V0: 2 * 1		
ra ← [ra]	V0: 2	
fp ← [fp]	V1: 2	
ao: 1	V0: 1	
\$fp →	a0: 1	
\$sp →		0 4 8 12 16 20 24 28
V1: 3		
V0: 3 * 2		
V0: 6		
ra ← [ra]	V0: 3	
fp ← [fp]	V1: 3	
ao: 2	V0: 2	
\$fp →	a0: 2	
\$sp →		0 4 8 12 16 20 24 28
V1: 4		
V0: 4 * 6		
V0: 24		
ra ← [ra]	V0: 4	
fp ← [fp]	V1: 4	
ao: 3	V0: 3	
\$fp →	a0: 3	
\$sp →		0 4 8 12 16 20 24 28
V1: 5		
V0: 5 * 24		
V0: 120	V0: 5	
ra ←	V1: 5	
fp ←	V0: 4	
\$fp →	ao 4	
\$sp →		0 4 8 12 16 20 24 28
V1: 5		
V0: 5 * 24		
V0: 120	V0: 5	
ra ←	V1: 5	
fp ←	V0: 4	
\$fp →	ao 5	

frame fact(1)

frame fact(2)

frame fact(3)

frame fact(4)

frame fact(5)