Arquitetura de Computadores
ACH2055
Aula 04 – Assembly MIPS (Parte II)

Norton Trevisan Roman
(norton@usp.br)

4 de setembro de 2019

Programando em MIPS

Condicionais

- Existem 2 condicionais em MIPS
  - Ambos do tipo I (*immediate*)

Programando em MIPS

Condicionais

- Existem 2 condicionais em MIPS
  - Ambos do tipo I (*immediate*)
- `beq` $\texttt{reg}_1$`,` $\texttt{reg}_2$`, rótulo`
  - *branch on equal*
  - Vá ao rótulo se o valor em reg_1 for igual ao valor em reg_2

Programando em MIPS

Condicionais

- Existem 2 condicionais em MIPS
  - Ambos do tipo I (*immediate*)
- `beq` $\texttt{reg}_1$`,` $\texttt{reg}_2$`, rótulo`
  - *branch on equal*
  - Vá ao rótulo se o valor em reg_1 for igual ao valor em reg_2
- `bne` $\texttt{reg}_1$`,` $\texttt{reg}_2$`, rótulo`
  - *branch on not equal*
  - Vá ao rótulo se o valor em reg_1 for diferente do valor em reg_2

Programando em MIPS

Condicionais – Exemplo

- `if (i==j) f = g + h; else f = g - h;`

Programando em MIPS

Condicionais – Exemplo

- `if (i==j) f = g + h; else f = g - h;`

```
.text
li $s0, 4 # variável f
li $s1, 5 # variável g
li $s2, 2 # variável h
li $s3, 3 # variável i
li $s4, 3 # variável j
bne $s3, $s4, Else
add $s0, $s1, $s2
j Saida
Else: sub $s0, $s1, $s2
Saida:
```

Programando em MIPS

Condicionais – Exemplo

- `if (i==j) f = g + h; else f = g - h;`

```
.text
li $s0, 4 # variável f
li $s1, 5 # variável g
li $s2, 2 # variável h
li $s3, 3 # variável i
li $s4, 3 # variável j
bne $s3, $s4, Else
add $s0, $s1, $s2
j Saida
Else: sub $s0, $s1, $s2
Saida:
```

Definimos as variáveis

Programando em MIPS

Condicionais – Exemplo

- `if (i==j) f = g + h; else f = g - h;`

```
.text
li $s0, 4 # variável f
li $s1, 5 # variável g
li $s2, 2 # variável h
li $s3, 3 # variável i
li $s4, 3 # variável j
bne $s3, $s4, Else
add $s0, $s1, $s2
j Saida
Else: sub $s0, $s1, $s2
Saida:
```

Em geral, é mais fácil testar a condição oposta, para já desviar ao `else`

Programando em MIPS

Condicionais – Exemplo

- `if (i==j) f = g + h; else f = g - h;`

```
.text
li $s0, 4 # variável f
li $s1, 5 # variável g
li $s2, 2 # variável h
li $s3, 3 # variável i
li $s4, 3 # variável j
bne $s3, $s4, Else
add $s0, $s1, $s2
j Saida
Else: sub $s0, $s1, $s2
Saida:
```

Nesse caso, para o rótulo batizado como `Else`

Programando em MIPS

Condicionais – Exemplo

- `if (i==j) f = g + h; else f = g - h;`

```
.text
li $s0, 4 # variável f
li $s1, 5 # variável g
li $s2, 2 # variável h
li $s3, 3 # variável i
li $s4, 3 # variável j
bne $s3, $s4, Else
add $s0, $s1, $s2
j Saida
Else: sub $s0, $s1, $s2
Saida:
```

E então, se a condição oposta não for satisfeita, prosseguir com o código do `if`

Programando em MIPS

Condicionais – Exemplo

- `if (i==j) f = g + h; else f = g - h;`

```
.text
li $s0, 4 # variável f
li $s1, 5 # variável g
li $s2, 2 # variável h
li $s3, 3 # variável i
li $s4, 3 # variável j
bne $s3, $s4, Else
add $s0, $s1, $s2
j Saida
Else: sub $s0, $s1, $s2
Saida:
```

E desviar diretamente à saída

Programando em MIPS

Condicionais – Exemplo

- `if (i==j) f = g + h; else f = g - h;`

```
.text
li $s0, 4 # variável f
li $s1, 5 # variável g
li $s2, 2 # variável h
li $s3, 3 # variável i
li $s4, 3 # variável j
bne $s3, $s4, Else
add $s0, $s1, $s2
j Saida
Else: sub $s0, $s1, $s2
Saida:
```

`j rótulo` (*jump*) corresponde a um desvio incondicional à posição de memória definida pelo rótulo

Programando em MIPS

Condicionais – Exemplo

- E se não houvesse o `else`?
  - `if (i==j) f = g + h;`

Programando em MIPS

Condicionais – Exemplo

- E se não houvesse o `else`?
  - `if (i==j) f = g + h;`

```
.text
li $s0, 4 # variável f
li $s1, 5 # variável g
li $s2, 2 # variável h
li $s3, 3 # variável i
li $s4, 3 # variável j
bne $s3, $s4, Saida
add $s0, $s1, $s2
Saida:
```

Programando em MIPS

Condicionais – Exemplo

- E se não houvesse o `else`?
  - `if (i==j) f = g + h;`

```
.text
li $s0, 4 # variável f
li $s1, 5 # variável g
li $s2, 2 # variável h
li $s3, 3 # variável i
li $s4, 3 # variável j
bne $s3, $s4, Saida
add $s0, $s1, $s2
Saida:
```

Faríamos um desvio condicional ao restante do código, como se o restante fosse o `else`

Programando em MIPS

Condicionais – Exemplo

- E se não houvesse o `else`?
  - `if (i==j) f = g + h;`

```
.text
li $s0, 4 # variável f
li $s1, 5 # variável g
li $s2, 2 # variável h
li $s3, 3 # variável i
li $s4, 3 # variável j
bne $s3, $s4, Saida
add $s0, $s1, $s2
Saida:
```

Note que, mesmo executando o corpo do `if`, a instrução no rótulo `Saída` será rodada

Programando em MIPS

Condicionais – Outro Exemplo

```
if ( i < j )
  a = b + c;
else
  a = b - c;
```

Programando em MIPS

Condicionais – Outro Exemplo

```
if ( i < j )
  a = b + c;
else
  a = b - c;
```

```
.text
li $s0, 4 # variavel a
li $s1, 5 # variavel b
li $s2, 2 # variavel c
li $s3, 2 # variavel i
li $s4, 3 # variavel j
```

Programando em MIPS

Condicionais – Outro Exemplo

```
if ( i < j )
  a = b + c;
else
  a = b - c;
```

```
.text
li $s0, 4 # variavel a
li $s1, 5 # variavel b
li $s2, 2 # variavel c
li $s3, 2 # variavel i
li $s4, 3 # variavel j

slt $t0, $s3, $s4
beq $t0, $zero, Else
add $s0, $s1, $s2
j Saida
Else: sub $s0, $s1, $s2
Saida:
```

Programando em MIPS

Condicionais – Outro Exemplo

```
if ( i < j )
  a = b + c;
else
  a = b - c;
```

```
.text
li $s0, 4 # variavel a
li $s1, 5 # variavel b
li $s2, 2 # variavel c
li $s3, 2 # variavel i
li $s4, 3 # variavel j

slt $t0, $s3, $s4
beq $t0, $zero, Else
add $s0, $s1, $s2
j Saida
Else: sub $s0, $s1, $s2
Saida:
```

Programando em MIPS

Condicionais – Outro Exemplo

```
if ( i < j )
  a = b + c;
else
  a = b - c;
```

```
.text
li $s0, 4 # variavel a
li $s1, 5 # variavel b
li $s2, 2 # variavel c
li $s3, 2 # variavel i
li $s4, 3 # variavel j

slt $t0, $s3, $s4
beq $t0, $zero, Else
add $s0, $s1, $s2
j Saida
Else: sub $s0, $s1, $s2
Saida:
```

Programando em MIPS

Condicionais – Outro Exemplo

```
if ( i < j )
  a = b + c;
else
  a = b - c;
```

```
.text
li $s0, 4 # variavel a
li $s1, 5 # variavel b
li $s2, 2 # variavel c
li $s3, 2 # variavel i
li $s4, 3 # variavel j

slt $t0, $s3, $s4
beq $t0, $zero, Else
add $s0, $s1, $s2
j Saida
Else: sub $s0, $s1, $s2
Saida:
```

Programando em MIPS

Condicionais – Outro Exemplo

```
if ( i < j )
  a = b + c;
else
  a = b - c;
```

```
.text
li $s0, 4 # variavel a
li $s1, 5 # variavel b
li $s2, 2 # variavel c
li $s3, 2 # variavel i
li $s4, 3 # variavel j

slt $t0, $s3, $s4
beq $t0, $zero, Else
add $s0, $s1, $s2
j Saida
Else: sub $s0, $s1, $s2
Saida:
```

Programando em MIPS

Laços

- Laços nada mais são que desvios a partes já rodadas do código:

```
i = 0;
while (i < 5)
  i++;
```

Programando em MIPS

Laços

- Laços nada mais são que desvios a partes já rodadas do código:

```
i = 0;
while (i < 5)
  i++;
```

```
.text
li $t0, 0
Laco: slti $t1, $t0, 5
beq $t1, $zero, Saida
addi $t0, $t0, 1
j Laco
Saida:
```

Programando em MIPS

Laços

- Laços nada mais são que desvios a partes já rodadas do código:

```
i = 0;
while (i < 5)
  i++;
```

```
.text
li $t0, 0
Laco: slti $t1, $t0, 5
beq $t1, $zero, Saida
addi $t0, $t0, 1
j Laco
Saida:
```

Programando em MIPS

Laços

- Laços nada mais são que desvios a partes já rodadas do código:

```
i = 0;
while (i < 5)
  i++;
```

```
.text
li $t0, 0
Laco: slti $t1, $t0, 5
beq $t1, $zero, Saida
addi $t0, $t0, 1
j Laco
Saida:
```

Programando em MIPS

Laços

- Laços nada mais são que desvios a partes já rodadas do código:

```
i = 0;
while (i < 5)
  i++;
```

```
.text
li $t0, 0
Laco: slti $t1, $t0, 5
beq $t1, $zero, Saida
addi $t0, $t0, 1
j Laco
Saida:
```

Programando em MIPS

Laços

- Laços nada mais são que desvios a partes já rodadas do código:

```
i = 0;
while (i < 5)
  i++;
```

```
.text
li $t0, 0
Laco: slti $t1, $t0, 5
beq $t1, $zero, Saida
addi $t0, $t0, 1
j Laco
Saida:
```

Instruções Relacionais Revista

- MIPS usa `slt`, `slti`, `beq` e `bne`, além do valor 0, para criar todas as operações relacionais
  - $==$, $\neq$, $<$, $>$, $\leq$, $\geq$

Programando em MIPS

Instruções Relacionais Revista

- MIPS usa `slt`, `slti`, `beq` e `bne`, além do valor 0, para criar todas as operações relacionais
  - $==$, $\neq$, $<$, $>$, $\leq$, $\geq$
- Não há um *branch on less than* por ser complicado
  - Ou aumentaria o ciclo do clock, ou exigiria ciclos extras por instrução
  - 2 instruções mais rápidas são mais úteis

Programando em MIPS

Desvios Incondicionais

- Além de `j rótulo`, há outros 2 desvios incondicionais importantes

Programando em MIPS

Desvios Incondicionais

- Além de `j rótulo`, há outros 2 desvios incondicionais importantes
- `jr registrador`
  - *jump register*
  - Desvio incondicional ao endereço especificado em um registrador
  - `jr $s2`

Programando em MIPS

Desvios Incondicionais

- `jal endereço_do_procedimento`
  - *jump and link*
  - Carrega em `$ra` o valor de `PC + 4` (*Program Counter*) e então desvia para a instrução no endereço-alvo
    - `$ra` contém então o endereço da instrução seguinte ao `jal` no código
  - Usada em chamadas a sub-rotinas
  - `jal rotulo`

Programando em MIPS

Sub-rotinas

- Na execução de uma sub-rotina, precisamos:
  - Colocar os parâmetros onde ela possa acessar
  - Transferir o controle a ela
  - Alocar a memória necessária a ela
  - Executar a tarefa desejada
  - Colocar o resultado em um lugar onde quem chamou a sub-rotina possa encontrar
  - Devolver o controle ao ponto de origem

Programando em MIPS

Sub-rotinas

- Para auxiliar nessa tarefa, MIPS apresenta a seguinte convenção de uso de registradores:
  - `$a0` - `$a3`: parâmetros da sub-rotina
  - `$v0` e `$v1`: valores de retorno
  - `$ra`: endereço para retorno ao ponto de origem
  - `$fp`: ponteiro para a moldura de pilha (já veremos)
  - `$sp`: ponteiro para o topo da pilha

Programando em MIPS

Sub-rotinas

- O procedimento a ser seguido é, então:
  - A rotina que faz a chamada coloca os parâmetros em `$a0` - `$a3`
  - Ela então executa "`jal X`" para ir à sub-rotina X
  - A sub-rotina executa seus cálculos, colocando os resultados em `$v0` e `$v1`
  - Ela então devolve o controle à rotina original, executando "`jr $ra`"

Programando em MIPS

Sub-rotinas – Problemas

- Existe um número pequeno de registradores de uso geral $\rightarrow$ podem vir a ser sobrescritos
  - Como preservar a informação anterior?

Programando em MIPS

Sub-rotinas – Problemas

- Existe um número pequeno de registradores de uso geral $\rightarrow$ podem vir a ser sobrescritos
  - Como preservar a informação anterior?
- Com chamadas aninhadas, o próprio `$ra` é sobrescrito
  - Como saber para onde retornar, após o término de uma dessas chamadas?

Programando em MIPS

Sub-rotinas – Problemas

- Existe um número pequeno de registradores de uso geral $\rightarrow$ podem vir a ser sobrescritos
  - Como preservar a informação anterior?
- Com chamadas aninhadas, o próprio `$ra` é sobrescrito
  - Como saber para onde retornar, após o término de uma dessas chamadas?
- Podemos querer passar mais de 4 parâmetros
  - Como fazer essa passagem?

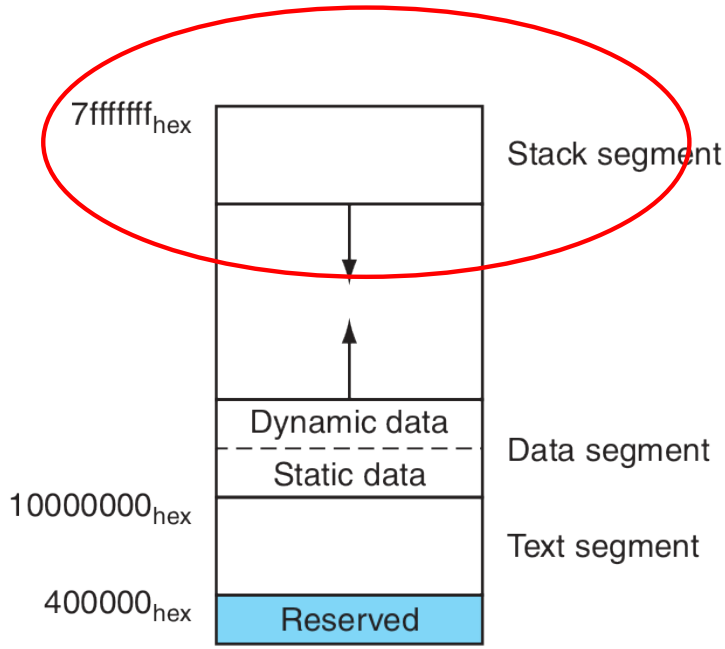
Sub-rotinas – Pilha de Execução

- A solução é armazenar em memória, em uma pilha

Programando em MIPS

Sub-rotinas – Pilha de Execução

- A solução é armazenar em memória, em uma pilha

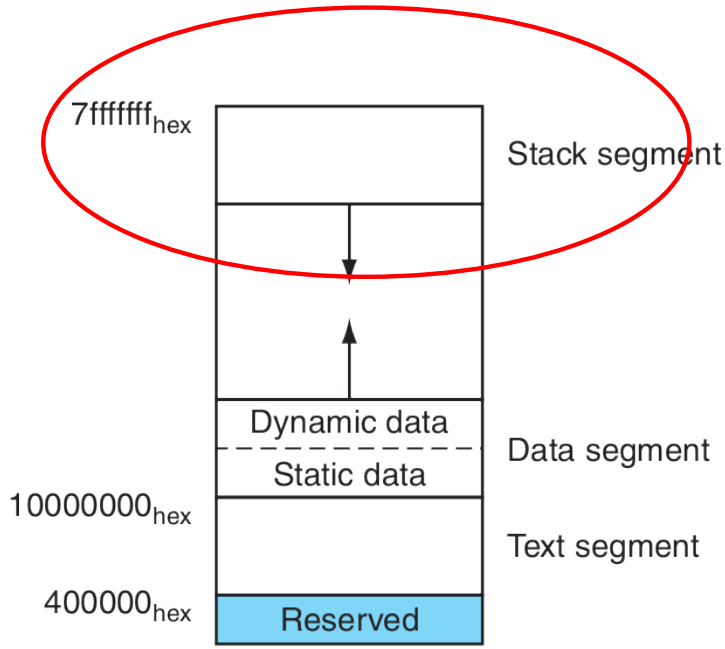


Fonte: Adaptado de [1]

Programando em MIPS

Sub-rotinas – Pilha de Execução

- A solução é armazenar em memória, em uma pilha
  - Iniciando no maior endereço possível
  - Iniciada manualmente ou pelo SO

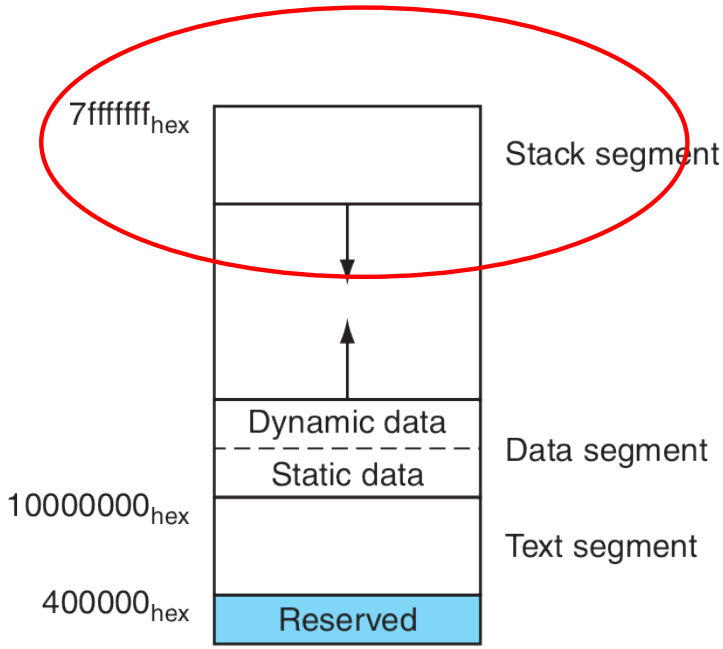


Fonte: Adaptado de [1]

Programando em MIPS

Sub-rotinas – Pilha de Execução

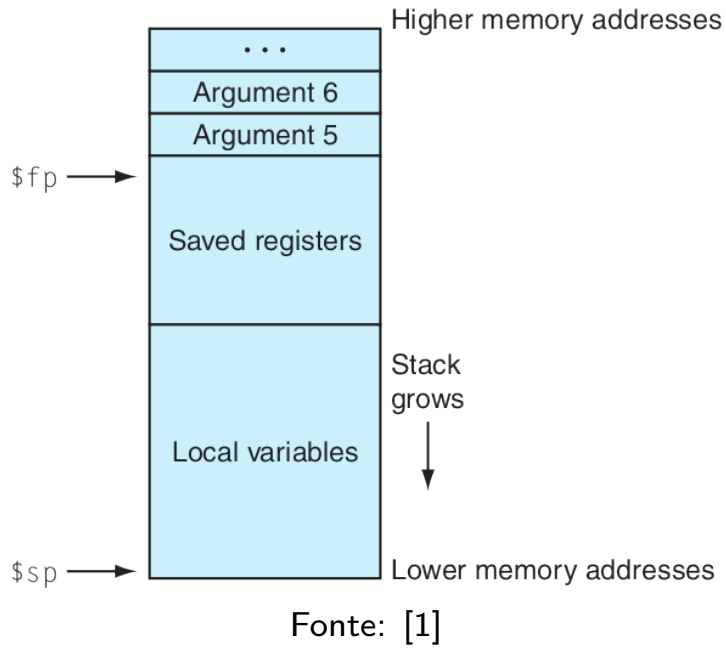
- A solução é armazenar em memória, em uma pilha
  - Iniciando no maior endereço possível
  - Iniciada manualmente ou pelo SO
- É de suma importância haver uma convenção para isso
  - Para que cada rotina e sub-rotina saiba onde encontrar o que busca



Fonte: Adaptado de [1]

Pilha de Execução – Moldura de Pilha

- A região da pilha destinada ao gerenciamento de uma determinada rotina é denominada **Moldura de Pilha**
  - Guardando então os registradores salvos, variáveis locais e argumentos passados à sub-rotina que será chamada

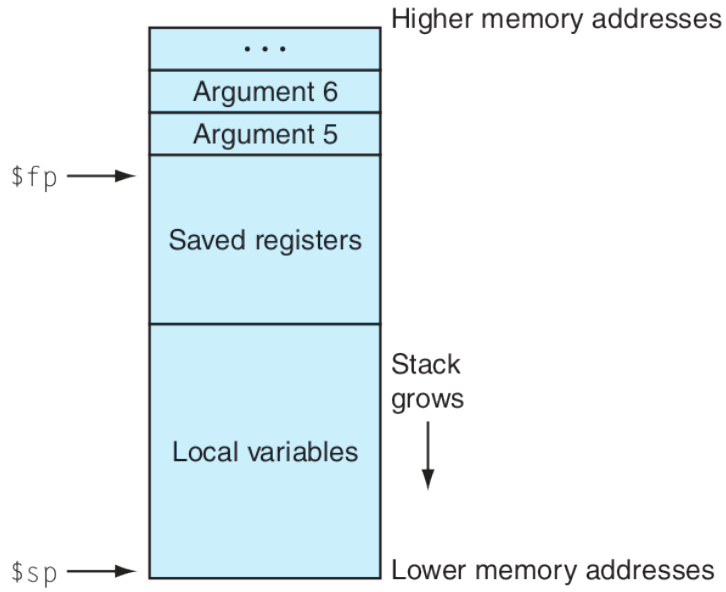


Fonte: [1]

Programando em MIPS

Pilha de Execução – Moldura de Pilha

- Em MIPS, gerenciada via 2 registradores

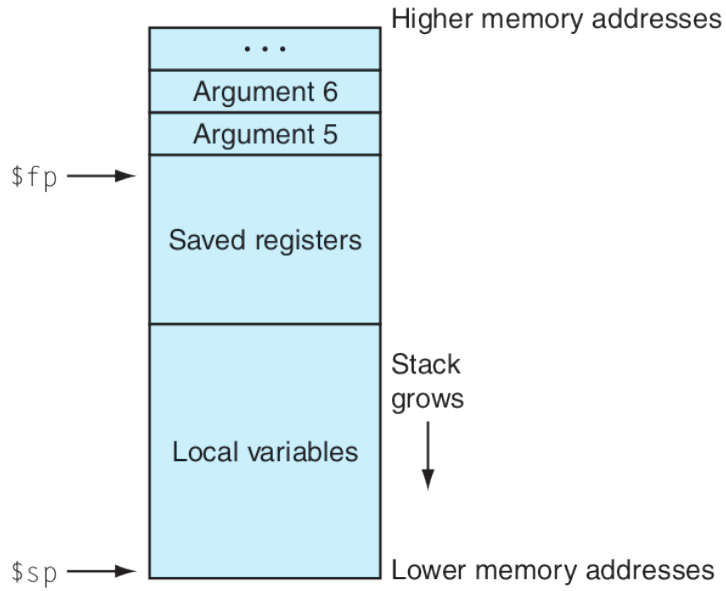


Fonte: [1]

Programando em MIPS

Pilha de Execução – Moldura de Pilha

- Em MIPS, gerenciada via 2 registradores
- $sp
  - *Stack pointer*
  - Contém o endereço da última palavra da pilha → seu topo
  - Necessário para que se saiba onde empilhar nova moldura

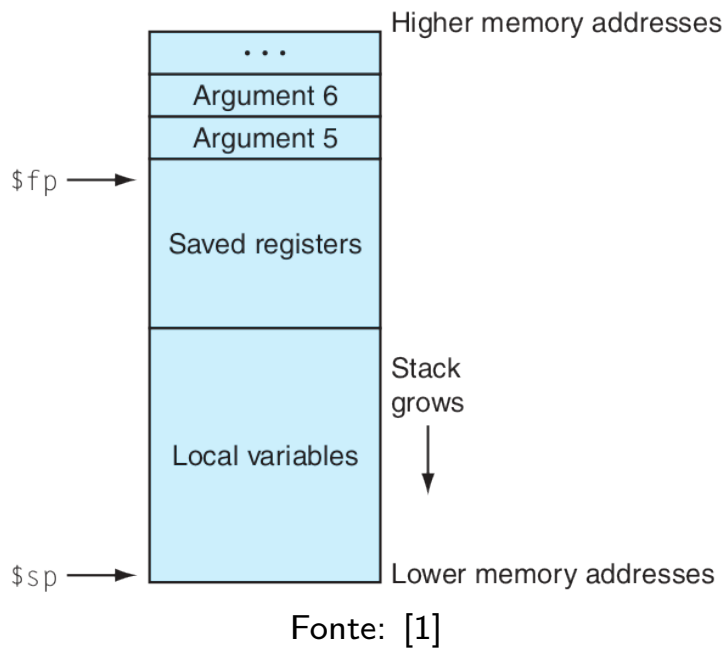


Fonte: [1]

Programando em MIPS

Pilha de Execução – Moldura de Pilha

- `$fp`
  - *Frame pointer*
  - Contém o endereço da primeira palavra da moldura

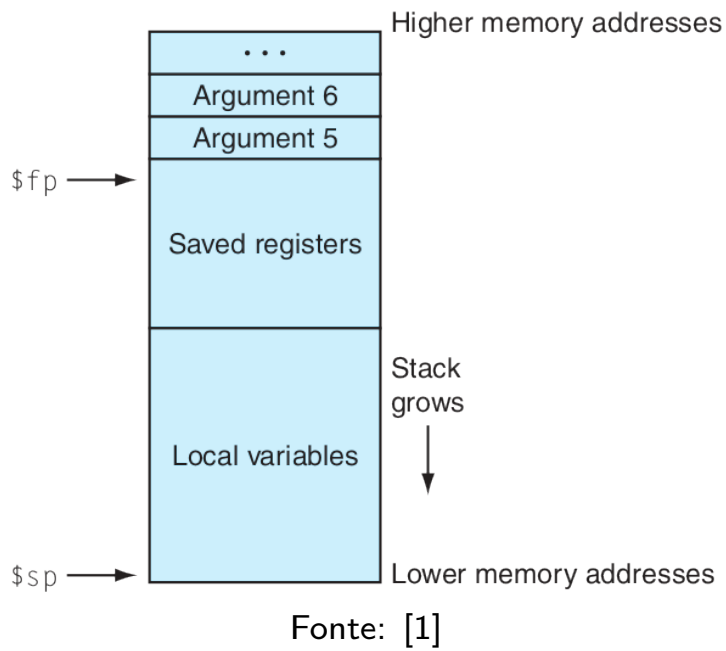


Fonte: [1]

Programando em MIPS

Pilha de Execução – Moldura de Pilha

- `$fp`
  - *Frame pointer*
  - Contém o endereço da primeira palavra da moldura
- A moldura do topo vai então do `$fp` ao `$sp`

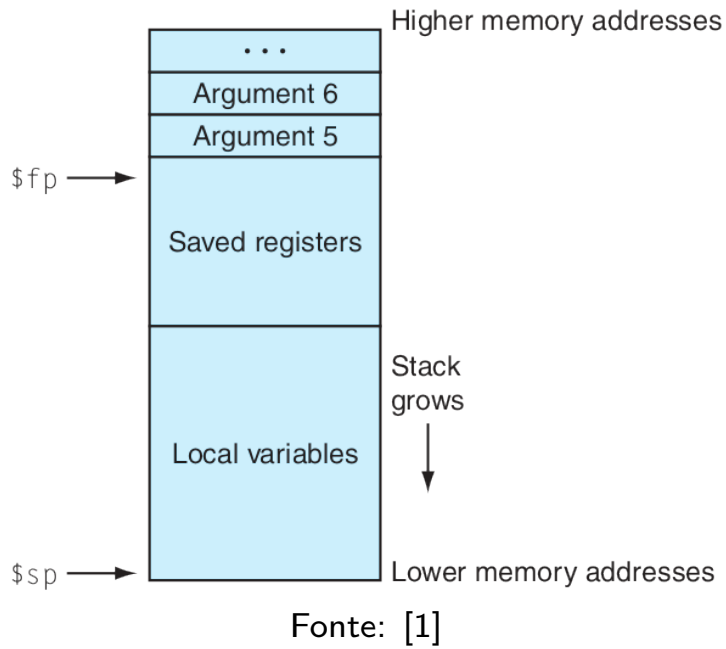


Fonte: [1]

Programando em MIPS

Pilha de Execução – Moldura de Pilha

- É necessário mesmo haver um `$fp`?

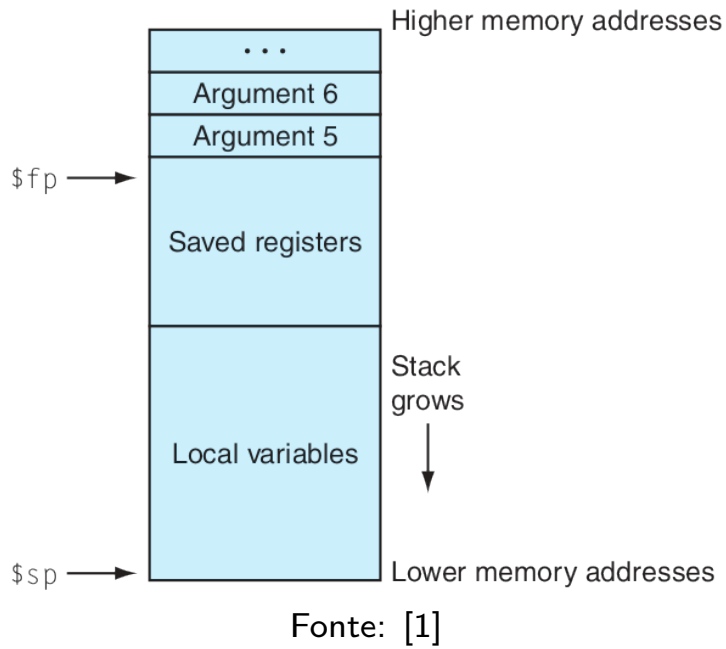


Fonte: [1]

Programando em MIPS

Pilha de Execução – Moldura de Pilha

- É necessário mesmo haver um `$fp`?
  - Não se pudermos acessar toda a moldura a partir do `$sp`

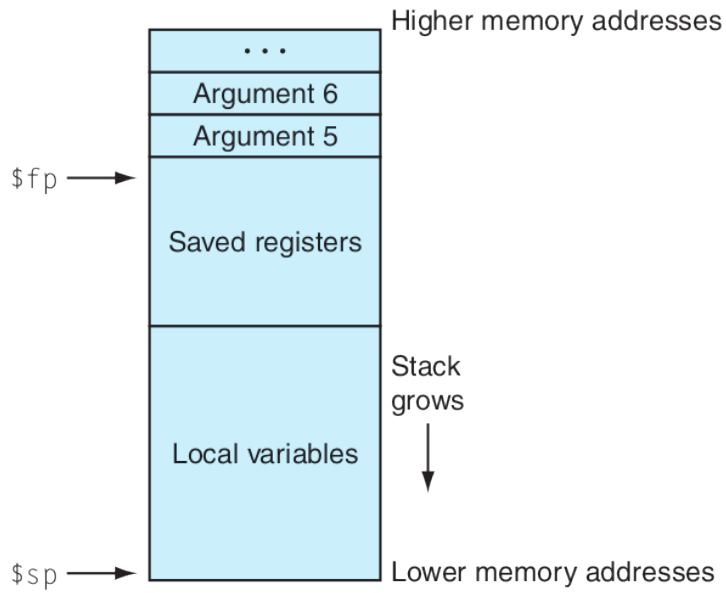


Fonte: [1]

Programando em MIPS

Pilha de Execução – Moldura de Pilha

- É necessário mesmo haver um `$fp`?
  - Não se pudermos acessar toda a moldura a partir do `$sp`
- Mas lembre que num `lw reg, desl(reg)` temos 16 bits para o deslocamento
  - Sendo 1 o de sinal

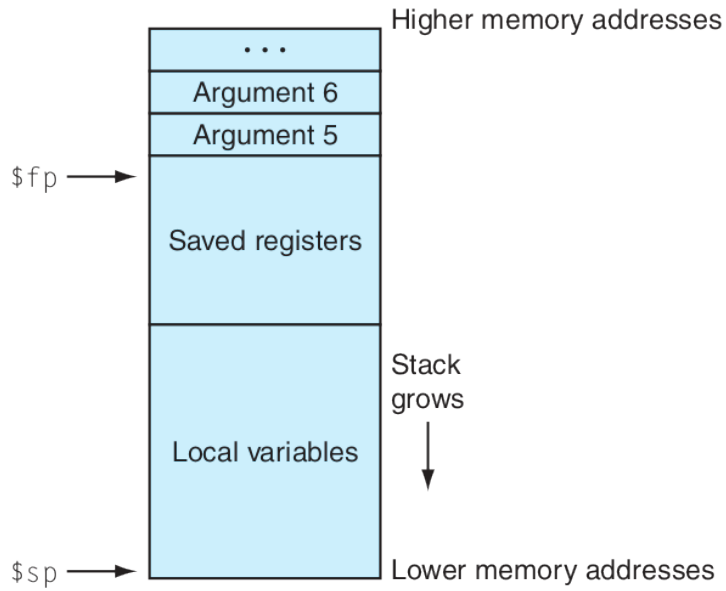


Fonte: [1]

Programando em MIPS

Pilha de Execução – Moldura de Pilha

- Isso nos diz que o máximo que podemos acessar é o endereço $\texttt{\$sp} + 32\text{KB}$ (2^{11})

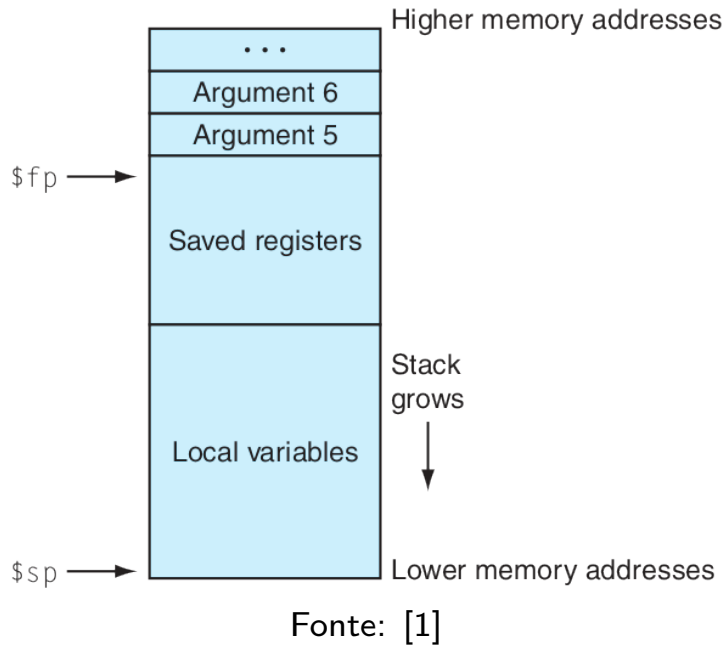


Fonte: [1]

Programando em MIPS

Pilha de Execução – Moldura de Pilha

- Isso nos diz que o máximo que podemos acessar é o endereço `$sp` + 32KB (2^{11})
- Se a moldura for maior, será necessário o uso do `$fp`
  - E assim permitimos molduras de até 64KB

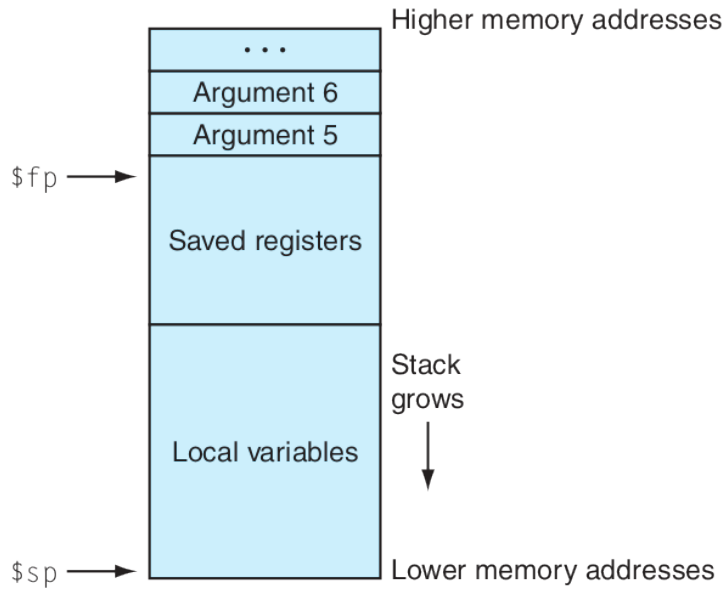


Fonte: [1]

Programando em MIPS

Pilha de Execução – Moldura de Pilha

- Isso nos diz que o máximo que podemos acessar é o endereço `$sp` + 32KB (2^{11})
- Se a moldura for maior, será necessário o uso do `$fp`
  - E assim permitimos molduras de até 64KB
- Por isso o uso de `$fp` é opcional

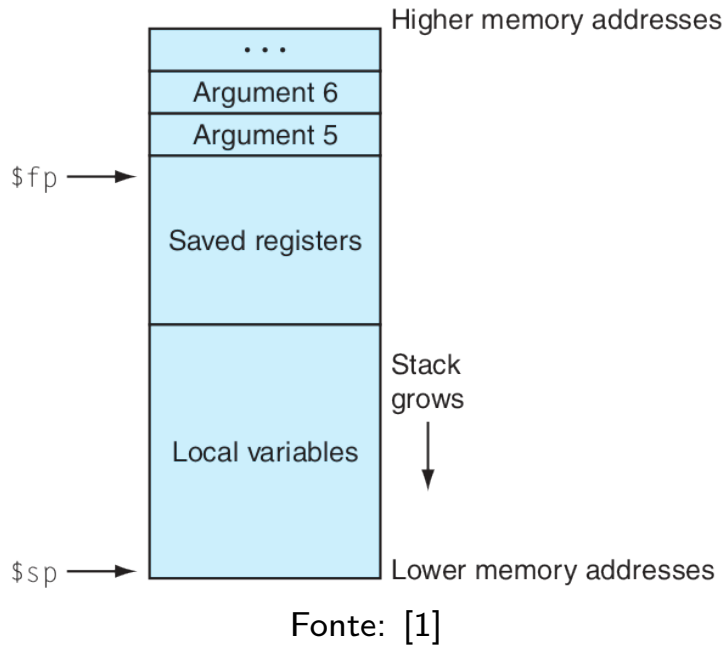


Fonte: [1]

Programando em MIPS

Pilha de Execução – Moldura de Pilha

- Mas mesmo que se acesse todo o espaço, há o problema do $sp ser móvel
  - Ele pode variar durante o procedimento, mudando sua distância aos elementos na pilha

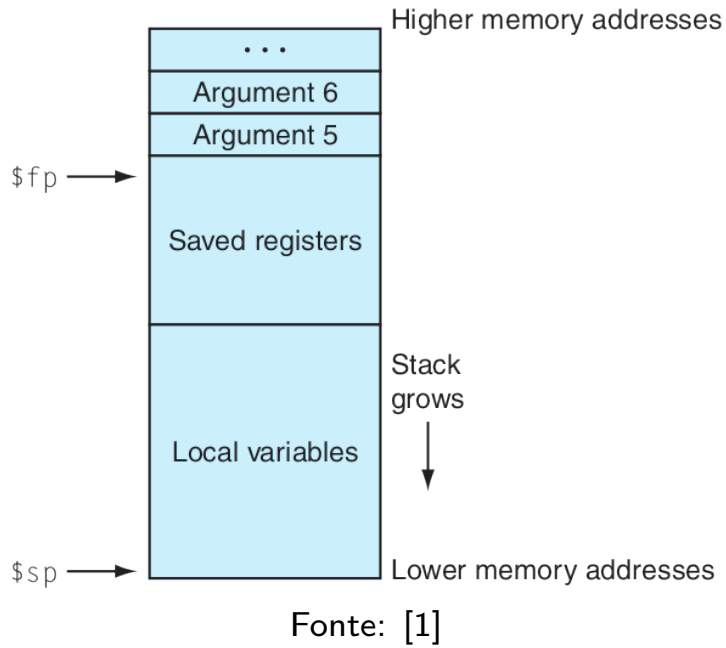


Fonte: [1]

Programando em MIPS

Pilha de Execução – Moldura de Pilha

- Mas mesmo que se acesse todo o espaço, há o problema do `$sp` ser móvel
  - Ele pode variar durante o procedimento, mudando sua distância aos elementos na pilha
- `$fp` fornece então uma base estável para referências locais

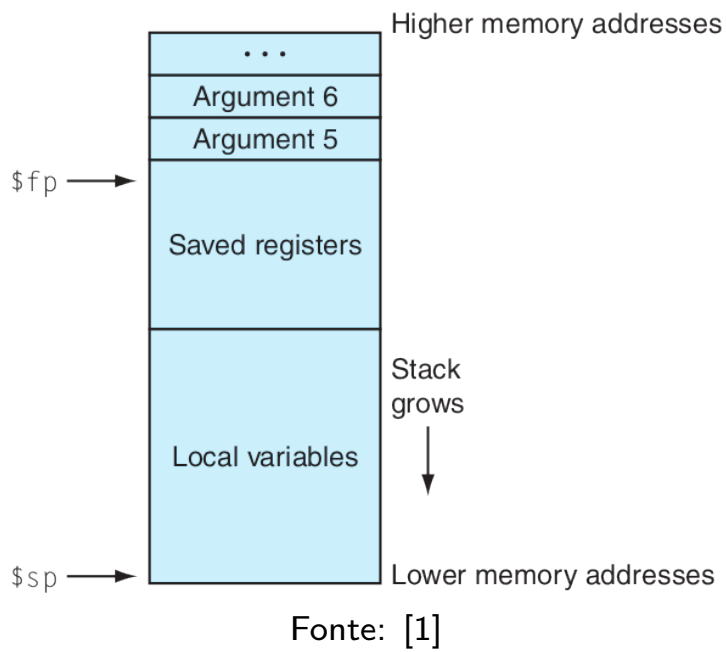


Fonte: [1]

Programando em MIPS

Pilha de Execução – Moldura de Pilha

- Podemos, no entanto, evitar usá-lo evitando mudanças no `$sp` dentro da rotina
  - Ajustando-o somente na entrada e saída da rotina

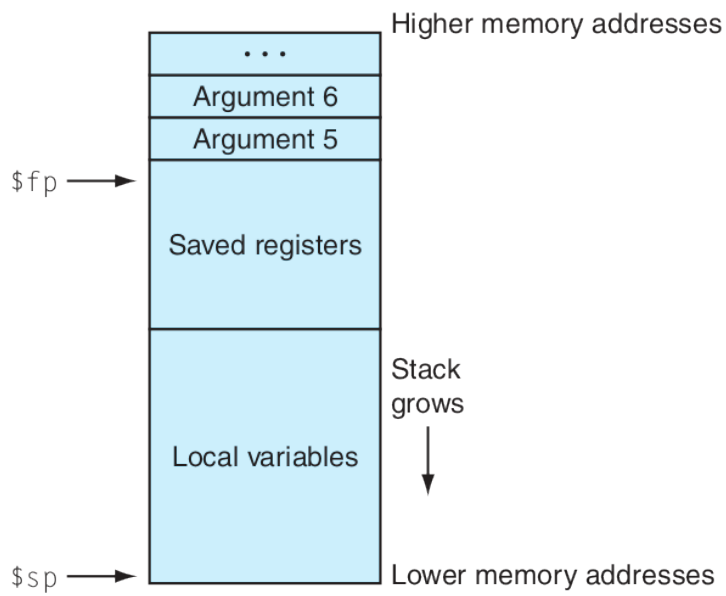


Fonte: [1]

Programando em MIPS

Pilha de Execução – Moldura de Pilha

- Podemos, no entanto, evitar usá-lo evitando mudanças no `$sp` dentro da rotina
  - Ajustando-o somente na entrada e saída da rotina
- Quando usarmos, devemos inicializá-lo com o `$sp` no início da chamada
  - Assim, ao fim da chamada, restauramos `$sp` usando o `$fp`

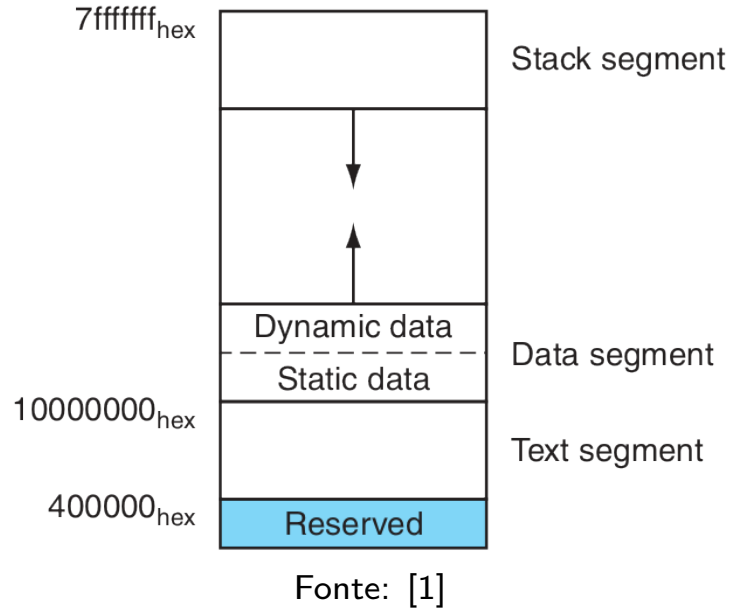


Fonte: [1]

Programando em MIPS

Sub-rotinas – Pilha de Execução

- O que armazenar na pilha?

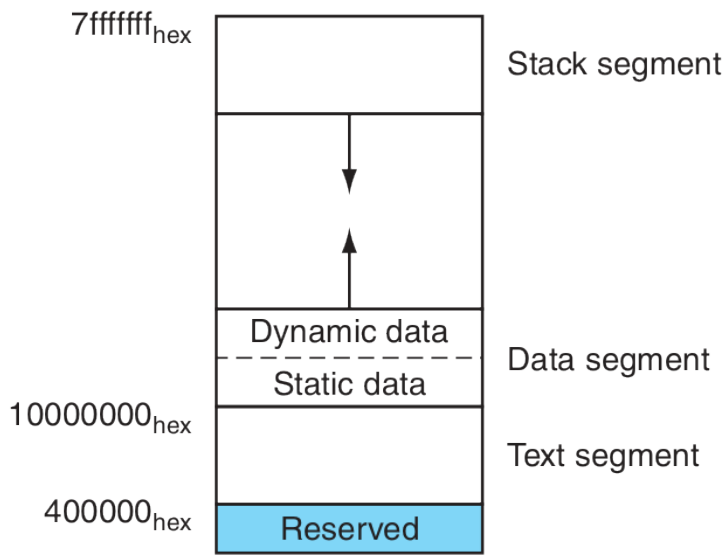


Fonte: [1]

Programando em MIPS

Sub-rotinas – Pilha de Execução

- O que armazenar na pilha?
- Imediatamente antes da chamada à sub-rotina:
  - Registradores `$a0` - `$a3` e `$t0` - `$t9` que a rotina possa precisar após a invocação da sub-rotina
  - Parâmetros-extra (para além dos 4 registradores)

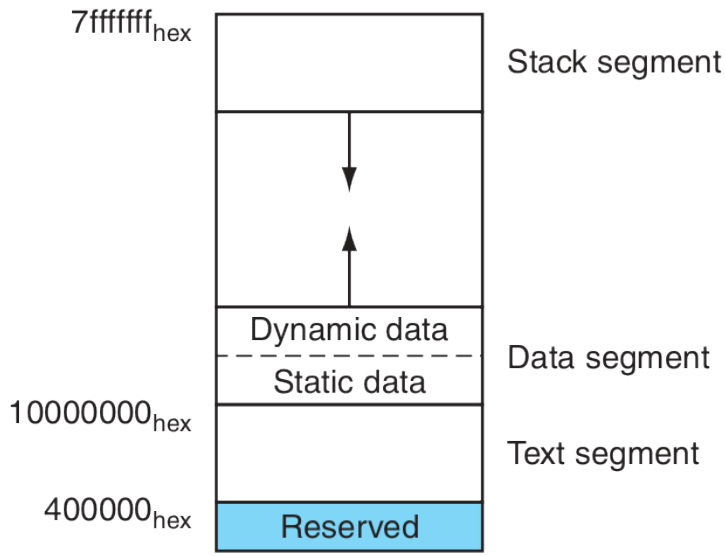


Fonte: [1]

Programando em MIPS

Sub-rotinas – Pilha de Execução

- Assim que a sub-rotina inicia:
  - `$fp` e `$ra` (da rotina que a chamou)
  - Registradores `$s0` – `$s7` antes de modificá-los, para restaurá-los depois
  - Variáveis locais



Fonte: [1]

Programando em MIPS

Pilha de Execução – Exemplo 1

- Considere a função

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

Programando em MIPS

Pilha de Execução – Exemplo 1

- Considere a função
- Ela não chama outra função nem possui variáveis locais
  - Não precisa salvar `$ra`, pois ninguém irá sobrescrever
  - Sequer precisa de moldura na pilha, pois não há o que salvar nela

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

Programando em MIPS

Pilha de Execução – Exemplo 1

- Considere a função
- Ela não chama outra função nem possui variáveis locais
  - Não precisa salvar `$ra`, pois ninguém irá sobrescrever

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

```
g: add $v0, $a0, $a1
jr $ra
```

- Sequer precisa de moldura na pilha, pois não há o que salvar nela

Programando em MIPS

Pilha de Execução

- Rotinas que não chamam outras rotinas são denominadas de **rotinas-folha**

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

```
g: add $v0, $a0, $a1
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 2

- Considere agora essa outra rotina-folha

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

Programando em MIPS

Pilha de Execução – Exemplo 2

- Considere agora essa outra rotina-folha
  - Dessa vez, com 1 variável local

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

Programando em MIPS

Pilha de Execução – Exemplo 2

- Considere agora essa outra rotina-folha
  - Dessa vez, com 1 variável local
  - Os 4 parâmetros foram passados em $a0 – $a3

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

Programando em MIPS

Pilha de Execução – Exemplo 2

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 2

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

Associaremos f a $s0

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 2

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

Por ser um registrador salvo, precisamos preservar seu valor anterior

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 2

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

Reservamos então
espaço para $s0 na pilha

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 2

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

Fazemos isso deslocando o topo da pilha

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 2

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

E armazenando $s0 lá

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 2

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

Prosseguimos nos cálculos

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 2

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

Prosseguimos nos cálculos

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 2

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

Prosseguimos nos cálculos

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 2

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

Note que não preservamos $t0 e $t1

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 2

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

Isso porque, pela convenção, estes não precisam ser preservados

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 2

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

Armazenamos a resposta no registrador de retorno

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 2

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

Restauramos o valor antigo de $s0

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 2

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

Restauramos o topo da pilha
(desempilhamos a moldura de X)

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 2

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

Voltamos à instrução posterior à chamada de X

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 2

```
int X(int g, int h, int i,
  int j) {
    int f;

    f = (g+h) - (i+j);
    return(f);
}
```

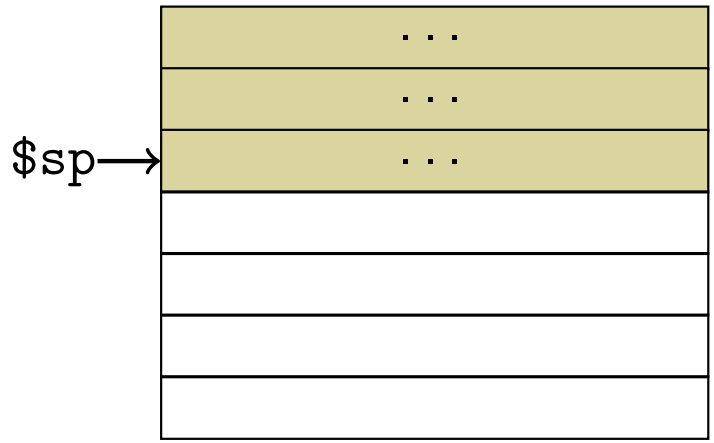
Uma vez que $sp permaneceu estático (após sua modificação inicial), não precisamos do $fp

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Programamando em MIPS

Pilha de Execução – Exemplo 2

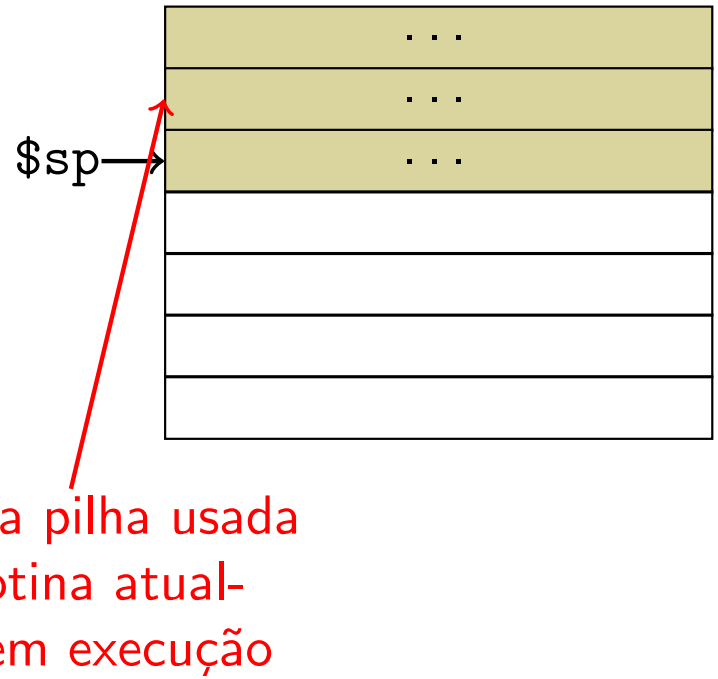
```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 2

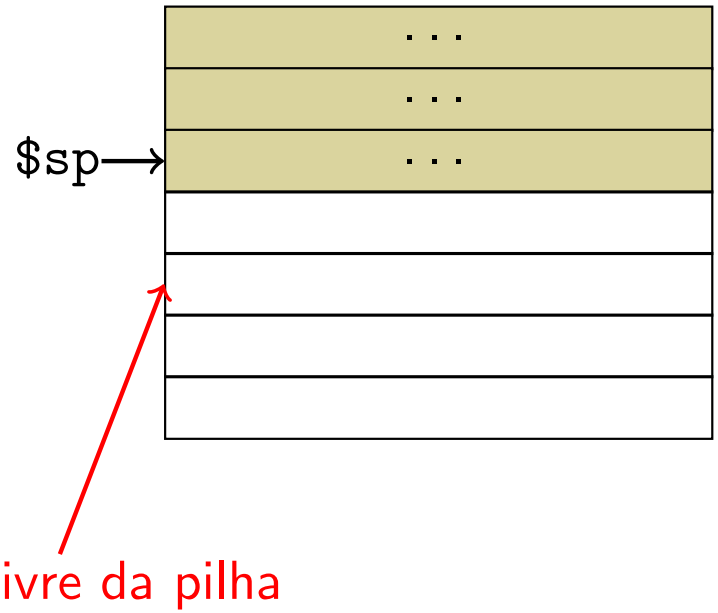
```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```



Programamando em MIPS

Pilha de Execução – Exemplo 2

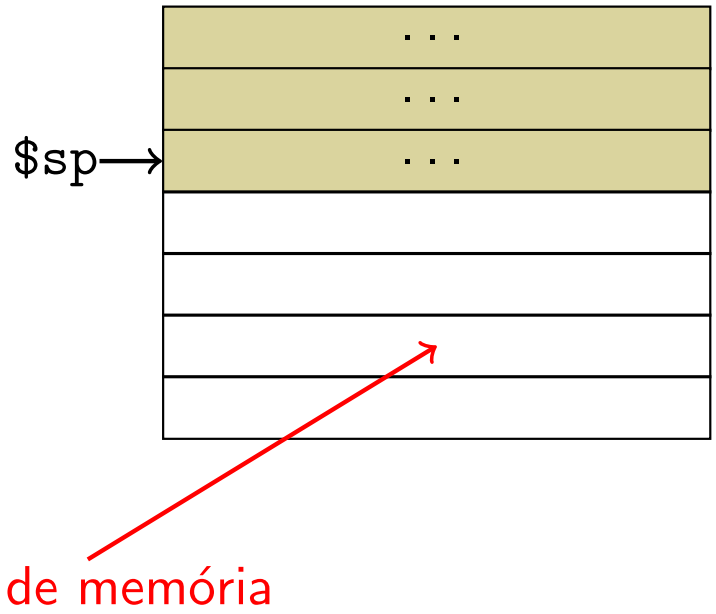
```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```



Programamando em MIPS

Pilha de Execução – Exemplo 2

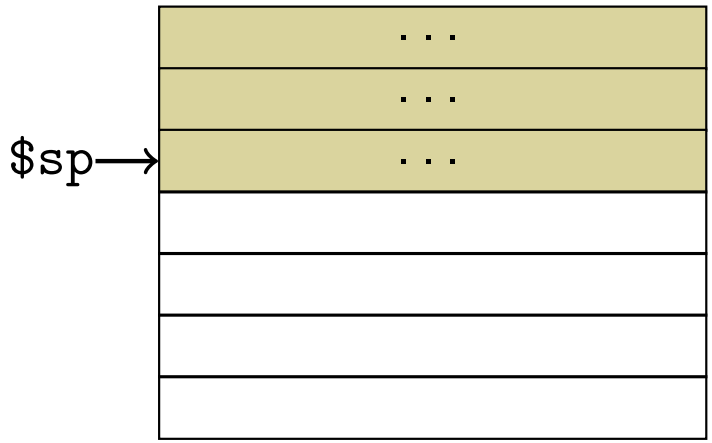
```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```



Programamando em MIPS

Pilha de Execução – Exemplo 2

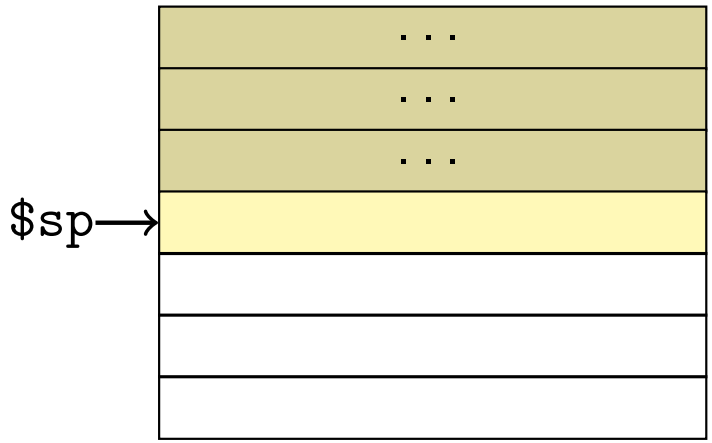
```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 2

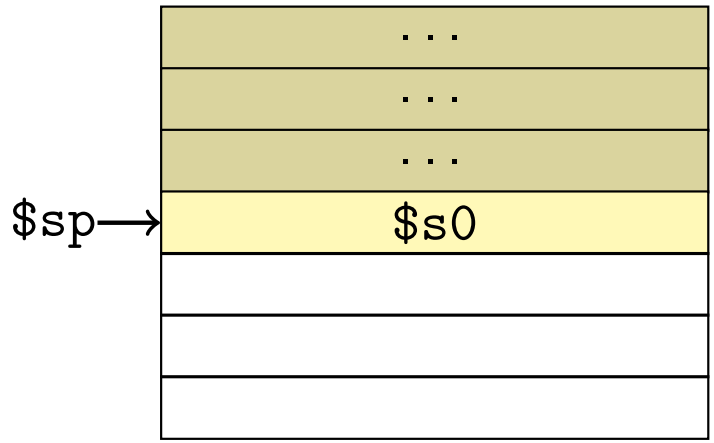
```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 2

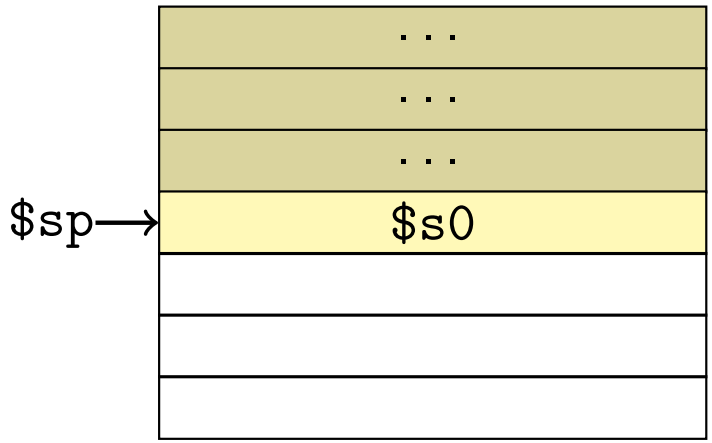
```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```



Programamando em MIPS

Pilha de Execução – Exemplo 2

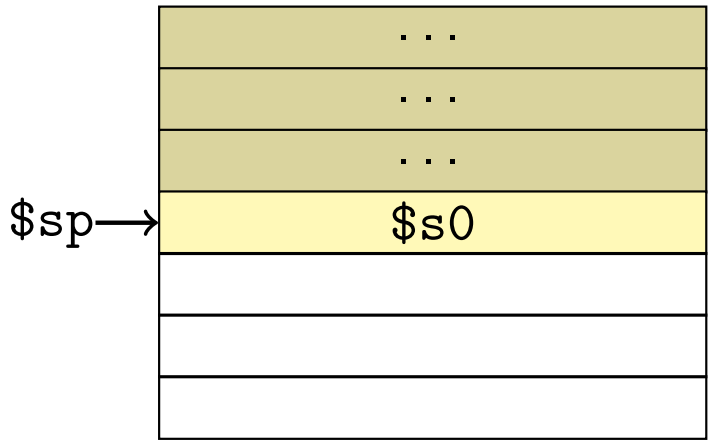
```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```



Programamando em MIPS

Pilha de Execução – Exemplo 2

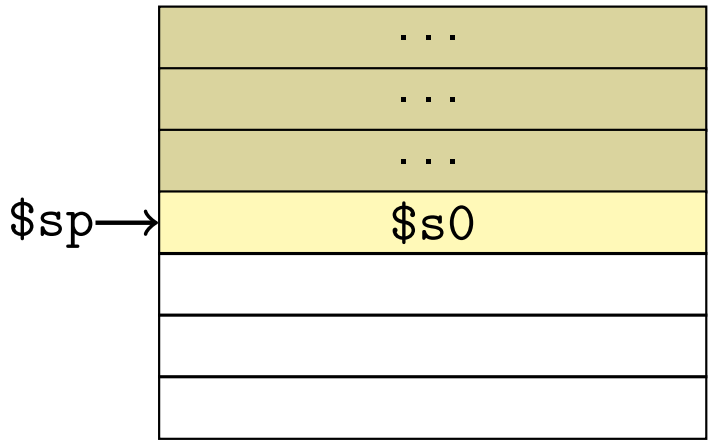
```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```



Programamando em MIPS

Pilha de Execução – Exemplo 2

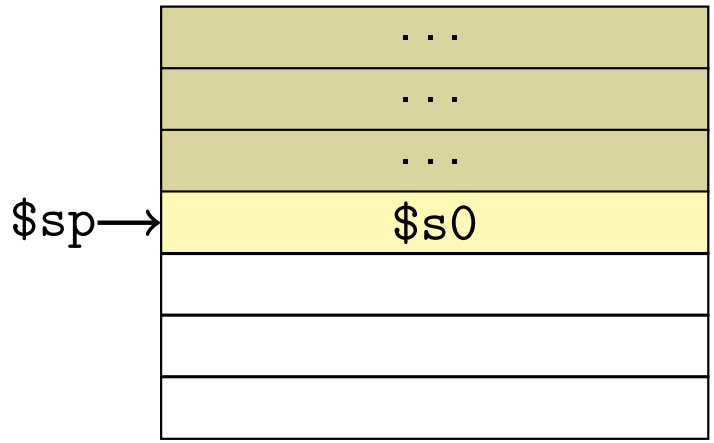
```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```



Programamando em MIPS

Pilha de Execução – Exemplo 2

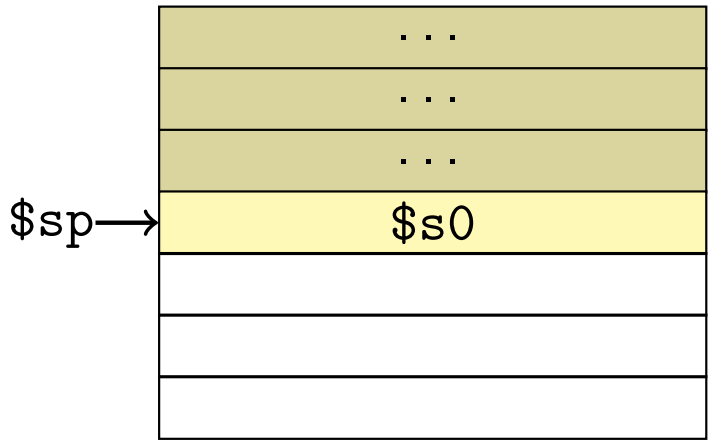
```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```



Programamando em MIPS

Pilha de Execução – Exemplo 2

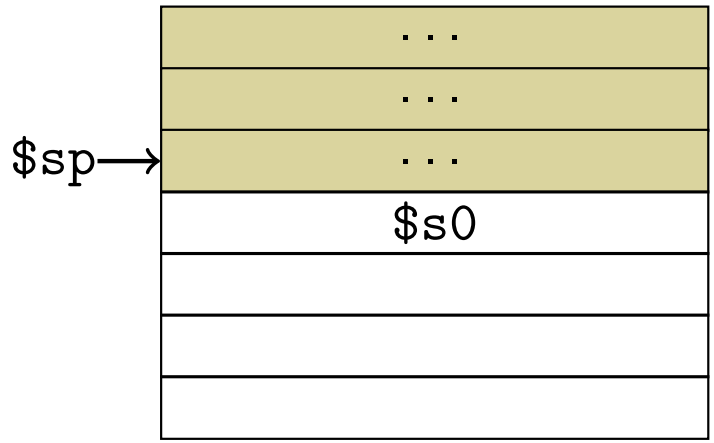
```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```



Programamando em MIPS

Pilha de Execução – Exemplo 2

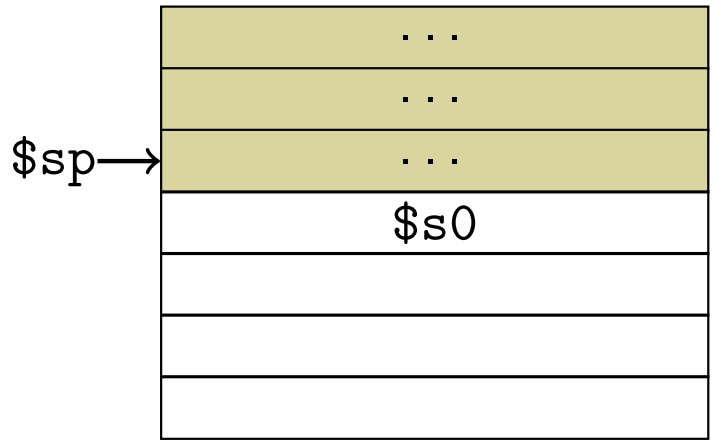
```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```



Programamando em MIPS

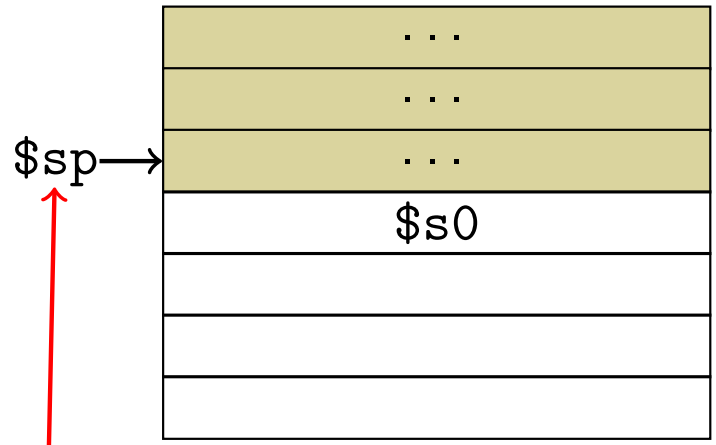
Pilha de Execução – Exemplo 2

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```



Pilha de Execução – Exemplo 2

```
X:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```



Ao final da sub-rotina, a pilha foi devolvida ao seu estado inicial

Programando em MIPS

Pilha de Execução – Exemplo 3

- Vejamos um programa completo

```
int main(void) {
  int x=3;

  x = x + m(2, x);
}

int m(int a, int b) {
  int y;

  y = a*b;
  return(y);
}
```

Programando em MIPS

Pilha de Execução – Exemplo 3

- Vejamos um programa completo
  - Com uma rotina-folha

```
int main(void) {
  int x=3;

  x = x + m(2, x);
}

int m(int a, int b) {
  int y;

  y = a*b;
  return(y);
}
```

Programando em MIPS

Pilha de Execução – Exemplo 3

- Vejamos um programa completo
  - Com uma rotina-folha
  - E outra não folha

```
int main(void) {
  int x=3;

  x = x + m(2, x);
}

int m(int a, int b) {
  int y;

  y = a*b;
  return(y);
}
```

Programando em MIPS

Pilha de Execução – Exemplo 3

```
int main(void) {
  int x=3;

  x = x + m(2, x);
}

int m(int a, int b) {
  int y;

  y = a*b;
  return(y);
}
```

Programando em MIPS

Pilha de Execução – Exemplo 3

```
int main(void) {
  int x=3;

  x = x + m(2, x);
}
```

```
.text
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
...
```

Programando em MIPS

Pilha de Execução – Exemplo 3

```
int main(void) {
  int x=3;

  x = x + m(2, x);
}
```

Associamos x a $s0

```
.text
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
...
```

Programando em MIPS

Pilha de Execução – Exemplo 3

```
int main(void) {
  int x=3;

  x = x + m(2, x);
}
```

Carregamos os parâmetros nos registradores $a0 e $a1

```
.text
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
...
```

Programando em MIPS

Pilha de Execução – Exemplo 3

```
int main(void) {
  int x=3;

  x = x + m(2, x);
}
```

Carregamos os parâmetros nos registradores $a0 e $a1

```
.text
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
...
```

Programando em MIPS

Pilha de Execução – Exemplo 3

```
int main(void) {
  int x=3;

  x = x + m(2, x);
}
```

Chamamos a sub-rotina. Nesse momento, `$ra = pc + 4` (o endereço de `add $s0...`)

```
.text
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
...
```

Programando em MIPS

Pilha de Execução – Exemplo 3

```
int main(void) {
  int x=3;

  x = x + m(2, x);
}
```

Somamos o retorno da sub-rotina (em $v0) a x, armazenando o resultado em x

```
.text
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
...
```

Programando em MIPS

Pilha de Execução – Exemplo 3

```
int main(void) {
  int x=3;

  x = x + m(2, x);
}
```

```
.text
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
...
```

Desviamos para o fim do programa

Programando em MIPS

Pilha de Execução – Exemplo 3

```
int m(int a, int b) {
  int y;

  y = a*b;
  return(y);
}
```

```
...
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```

Programando em MIPS

Pilha de Execução – Exemplo 3

Os parâmetros estão em $a0 e $a1

```
int m(int a, int b) {
  int y;

  y = a*b;
  return(y);
}
```

```
...
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```

Programando em MIPS

Pilha de Execução – Exemplo 3

Usaremos $s0 para y, então alocamos espaço na pilha

```
int m(int a, int b) {
  int y;

  y = a*b;
  return(y);
}
```

```
...
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```

Programando em MIPS

Pilha de Execução – Exemplo 3

E persistimos o valor de $s0 lá

```
int m(int a, int b) {
  int y;

  y = a*b;
  return(y);
}
```

```
...
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```

Programando em MIPS

Pilha de Execução – Exemplo 3

```
int m(int a, int b) {
  int y;

  y = a*b;
  return(y);
}
```

```
...
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```

Programando em MIPS

Pilha de Execução – Exemplo 3

Lembre que $v0 conterá o valor de retorno da sub-rotina

```
int m(int a, int b) {
  int y;

  y = a*b;
  return(y);
}
```

```
...
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```

Programando em MIPS

Pilha de Execução – Exemplo 3

Terminada a sub-rotina, restauramos $s0

```
int m(int a, int b) {
  int y;

  y = a*b;
  return(y);
}
```

```
...
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```

Programando em MIPS

Pilha de Execução – Exemplo 3

E liberamos o espaço na pilha

```
int m(int a, int b) {
  int y;

  y = a*b;
  return(y);
}
```

```
...
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```

Programando em MIPS

Pilha de Execução – Exemplo 3

Retornando à chamada da sub-rotina

```
int m(int a, int b) {
  int y;

  y = a*b;
  return(y);
}
```

```
...
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```

Programando em MIPS

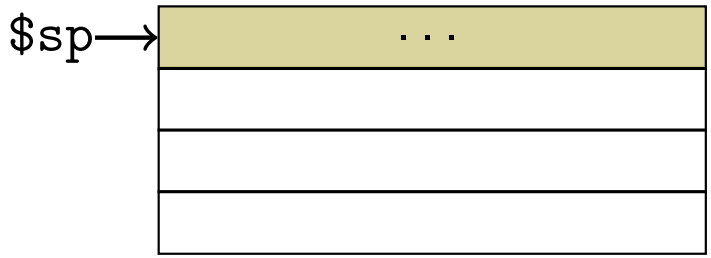
Pilha de Execução – Exemplo 3

```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```

Programando em MIPS

Pilha de Execução – Exemplo 3

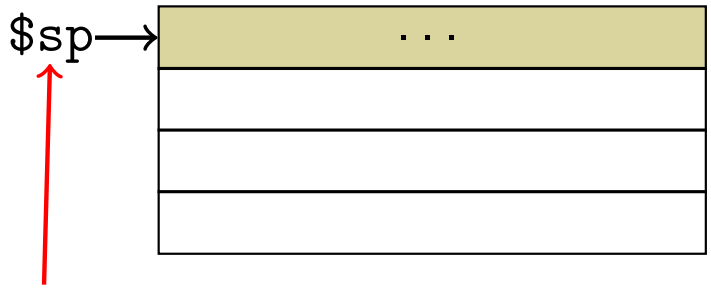
```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```



Programando em MIPS

Pilha de Execução – Exemplo 3

```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```

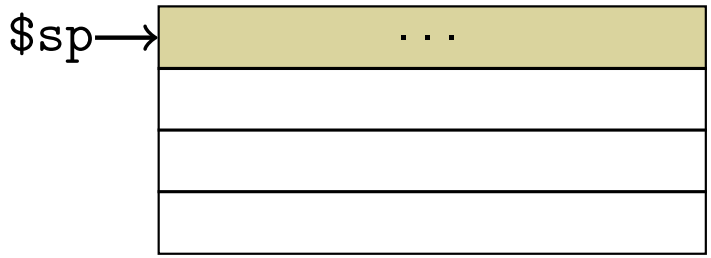


Inicialmente, $sp está no endereço mais alto possível ao programa

Programando em MIPS

Pilha de Execução – Exemplo 3

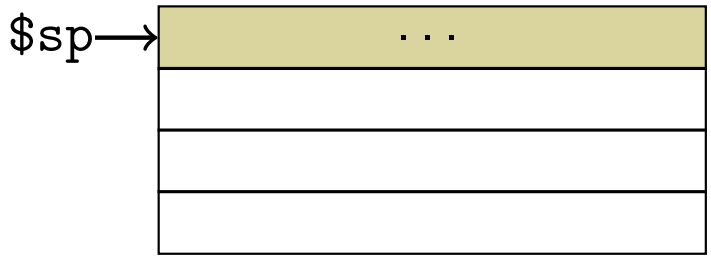
```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```



Programando em MIPS

Pilha de Execução – Exemplo 3

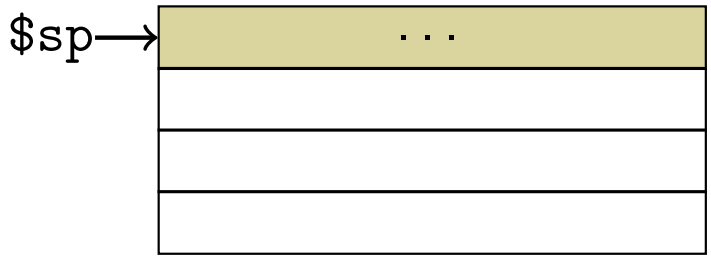
```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```



Programando em MIPS

Pilha de Execução – Exemplo 3

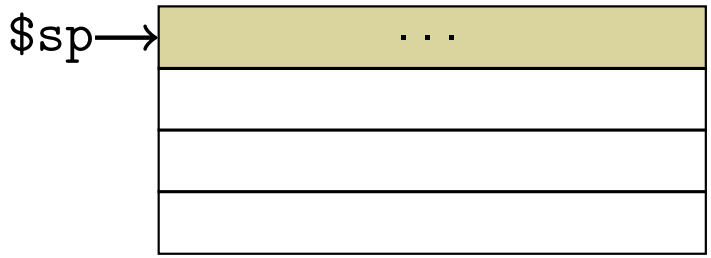
```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```



Programando em MIPS

Pilha de Execução – Exemplo 3

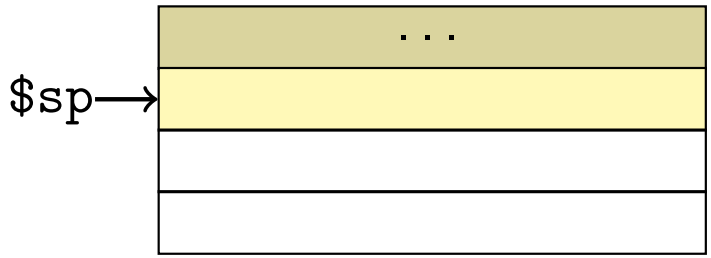
```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```



Programando em MIPS

Pilha de Execução – Exemplo 3

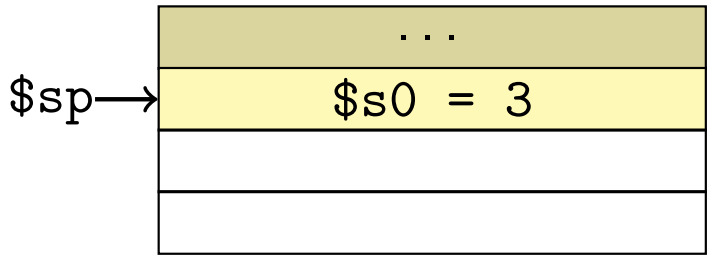
```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```



Programando em MIPS

Pilha de Execução – Exemplo 3

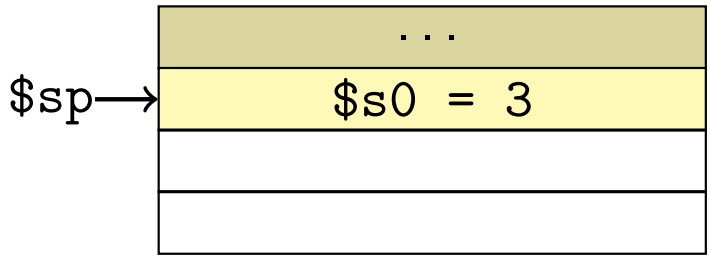
```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```



Programando em MIPS

Pilha de Execução – Exemplo 3

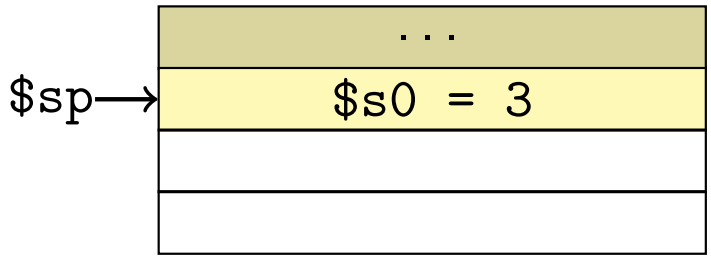
```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```



Programando em MIPS

Pilha de Execução – Exemplo 3

```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```

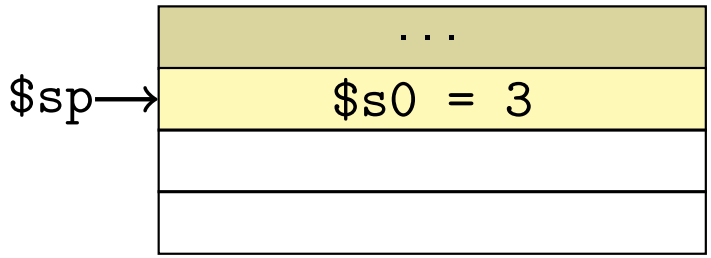


Programando em MIPS

Pilha de Execução – Exemplo 3

```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
```

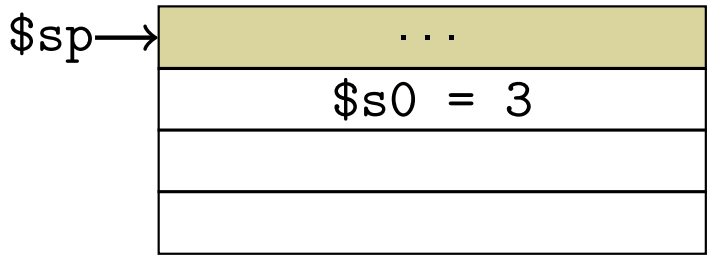
```
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```



Programando em MIPS

Pilha de Execução – Exemplo 3

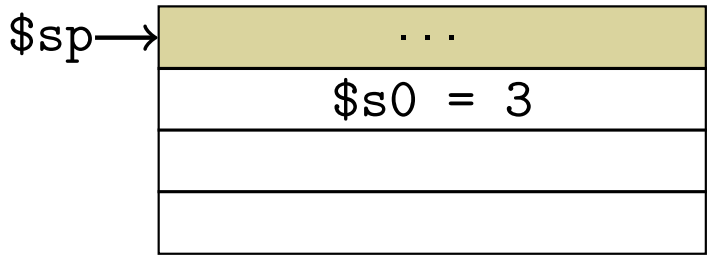
```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```



Programando em MIPS

Pilha de Execução – Exemplo 3

```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```

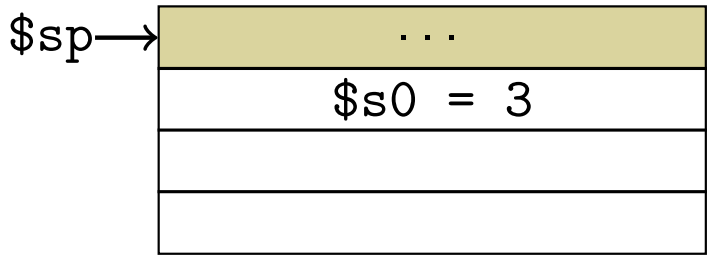


$ra contém o endereço da instrução seguinte ao jal

Programando em MIPS

Pilha de Execução – Exemplo 3

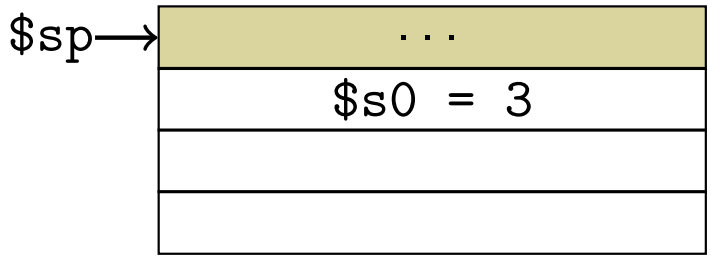
```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```



Programando em MIPS

Pilha de Execução – Exemplo 3

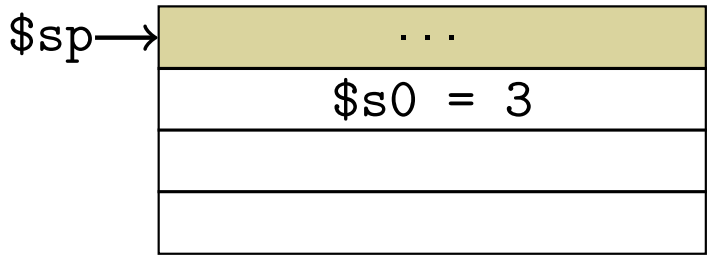
```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```



Programando em MIPS

Pilha de Execução – Exemplo 3

```
main:
li $s0, 3
li $a0, 2
move $a1, $s0
jal m
add $s0, $s0, $v0
j fim
m:
addi $sp, $sp -4
sw $s0, 0($sp)
mul $s0, $a0, $a1
move $v0, $s0
lw $s0, 0($sp)
addi $sp, $sp 4
jr $ra
fim:
```



Novamente o uso do
$fp não foi necessário

Programando em MIPS

Pilha de Execução – Exemplo 4

- E agora o terror do uso da pilha

Programando em MIPS

Pilha de Execução – Exemplo 4

- E agora o terror do uso da pilha
- A recursão

```
int fat(int n) {
  if (n<2) return(1);
  return (n * fat(n-1));
}
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}

fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

n é passado em $a0

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Cada chamada recursiva também usará $a0

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Além de $ra, para seu retorno

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Precisamos então preservar na pilha esses valores

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Precisamos então preservar na pilha esses valores

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Precisamos então preservar na pilha esses valores

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Precisamos então preservar na pilha esses valores

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}

fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}

fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Se n<2, o valor de retorno é 1

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Antes de retornar, restauramos a pilha

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Como nada foi mudado, não precisamos restaurar $a0 e $ra

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}

fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Se n≥2, prosseguimos em L1

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Preparamos o parâmetro da próxima chamada recursiva

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

E fazemos a chamada

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Ao retornar, precisamos fazer $v0 * $a0

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Mas $a0 foi modificado pelas chamadas

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Então temos que restaurá-lo

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Então temos que restaurá-lo

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Aproveitamos para restaurar o endereço de retorno também

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Além da pilha

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

Preparamos o resultado

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4

```
int fat(int n) {
  if (n<2) return(1);
  return(n * fat(n-1));
}
```

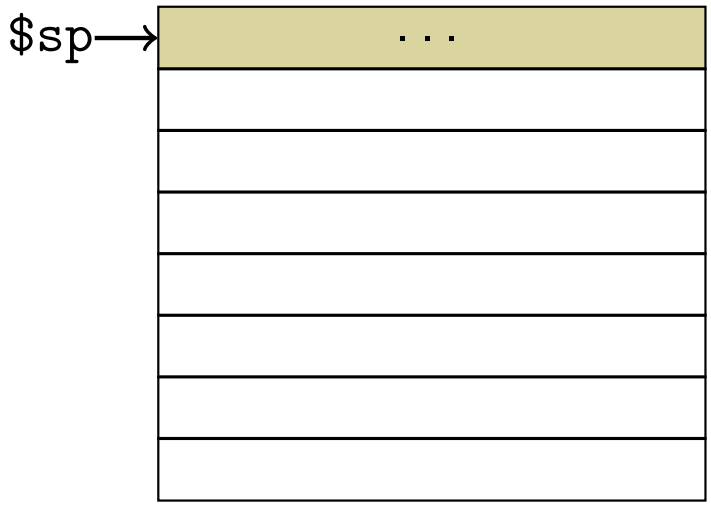
E retornamos

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

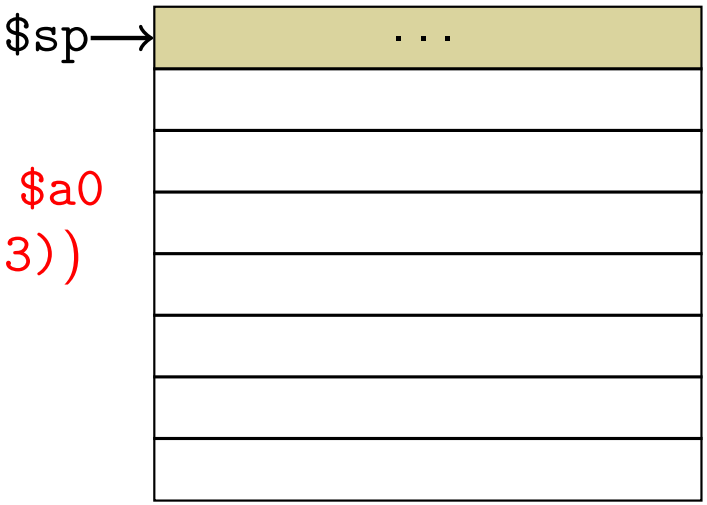


Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

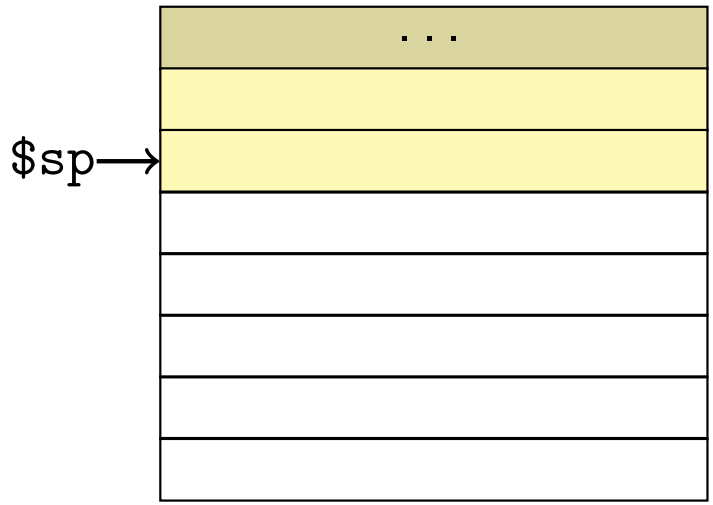
Temos que `$a0` = 3 (`fat(3)`)



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

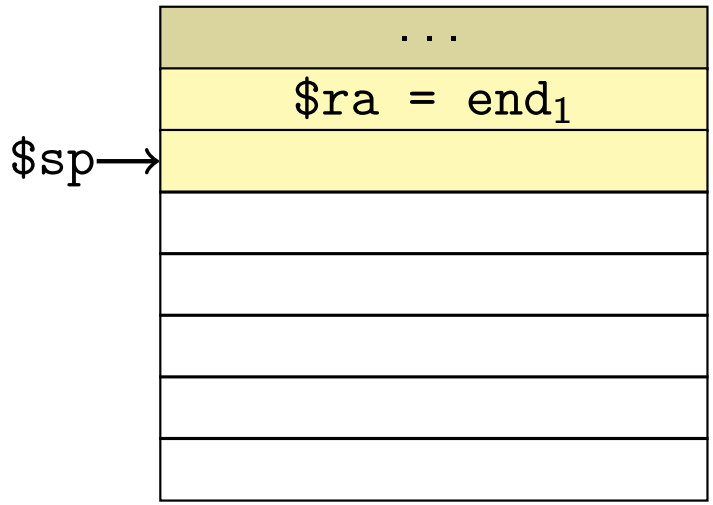
```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

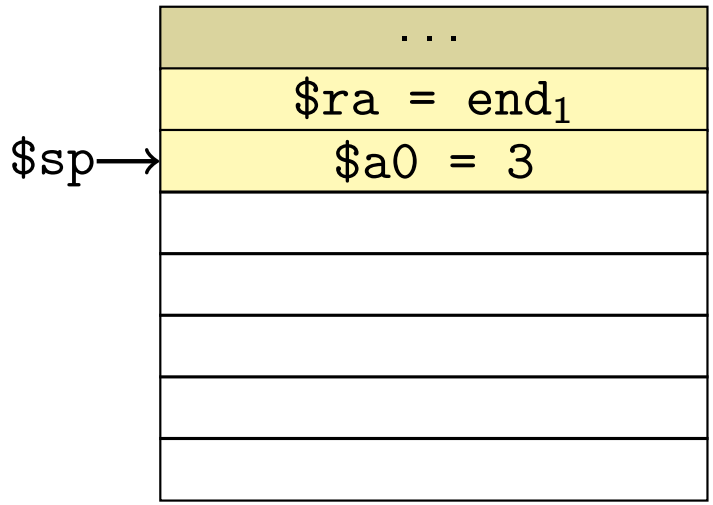
```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

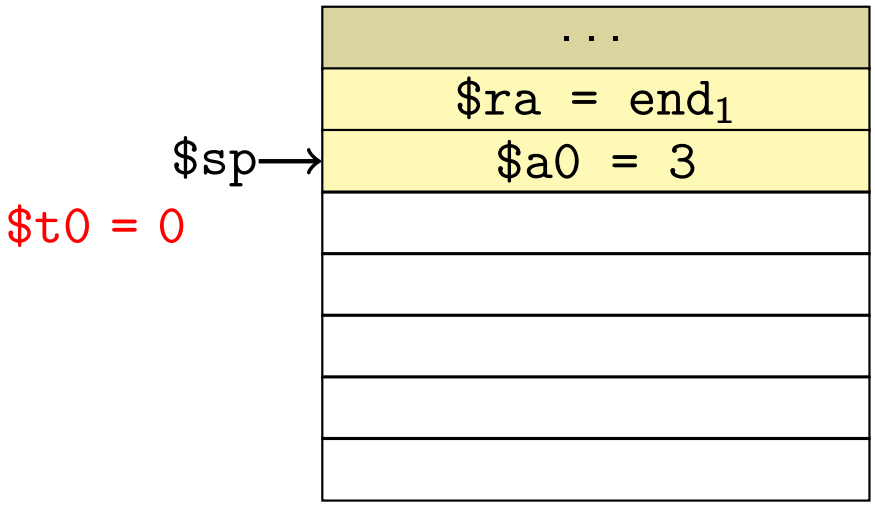
```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

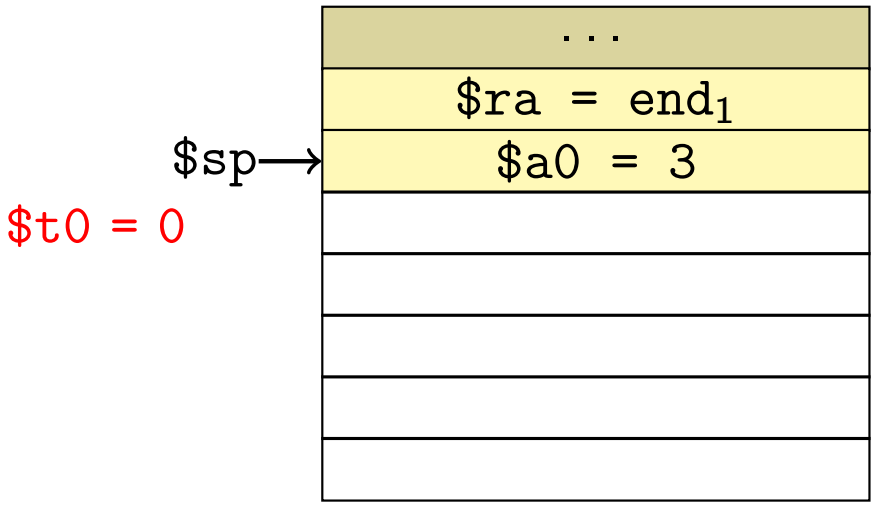
```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

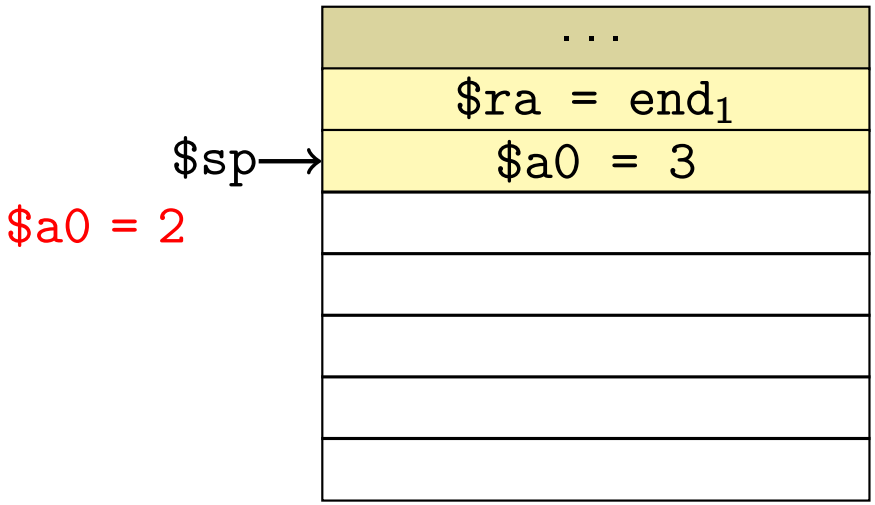
```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

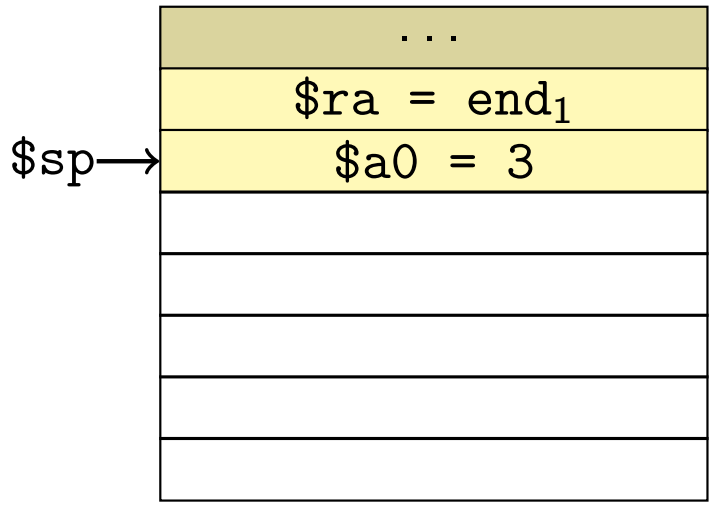
```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

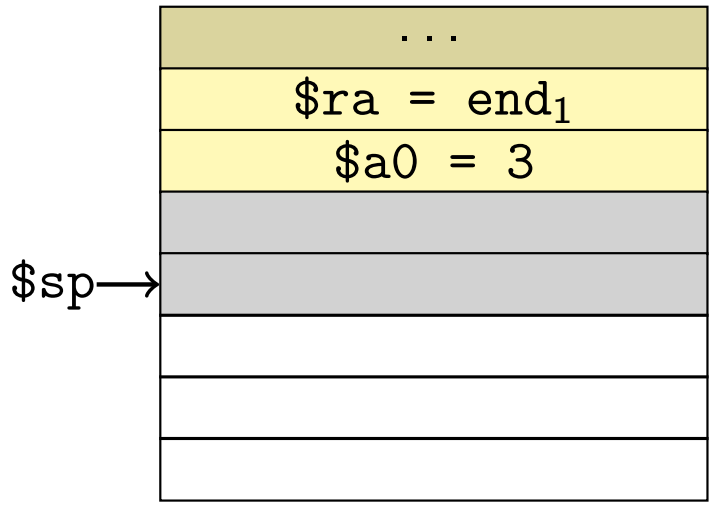
```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

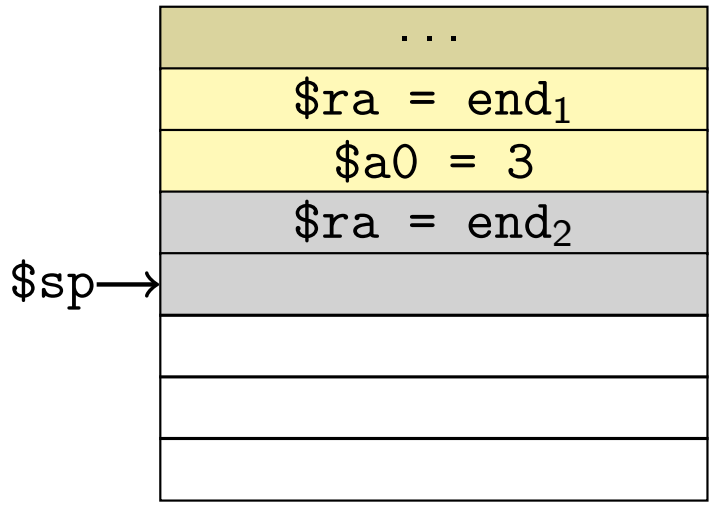
```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

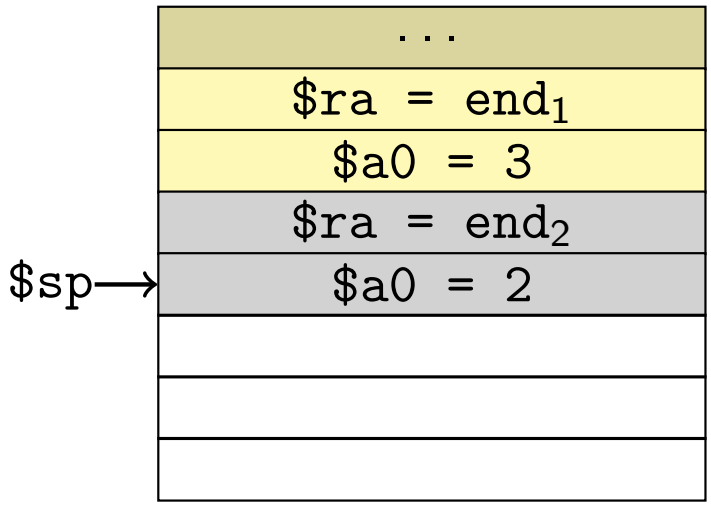
```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

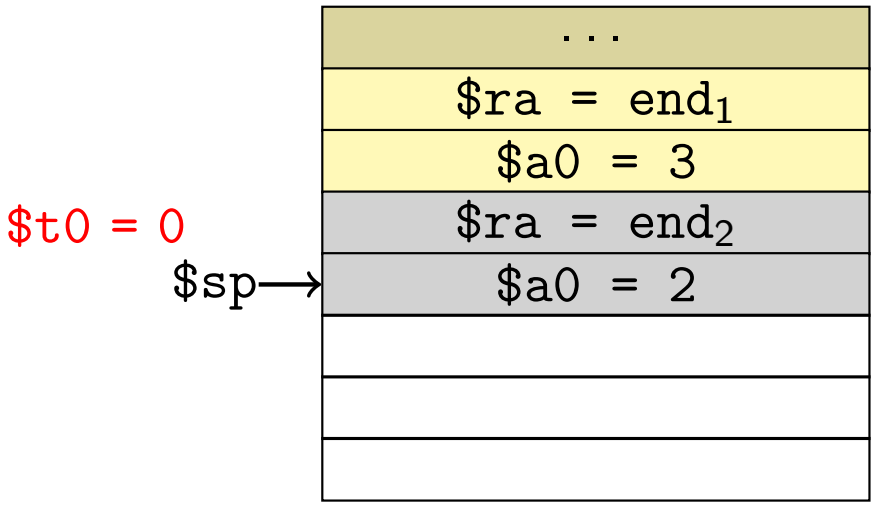
```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



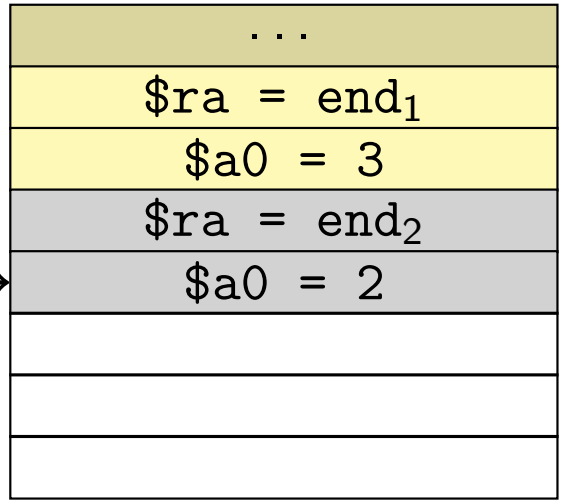
Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

$t0 = 0

$sp →

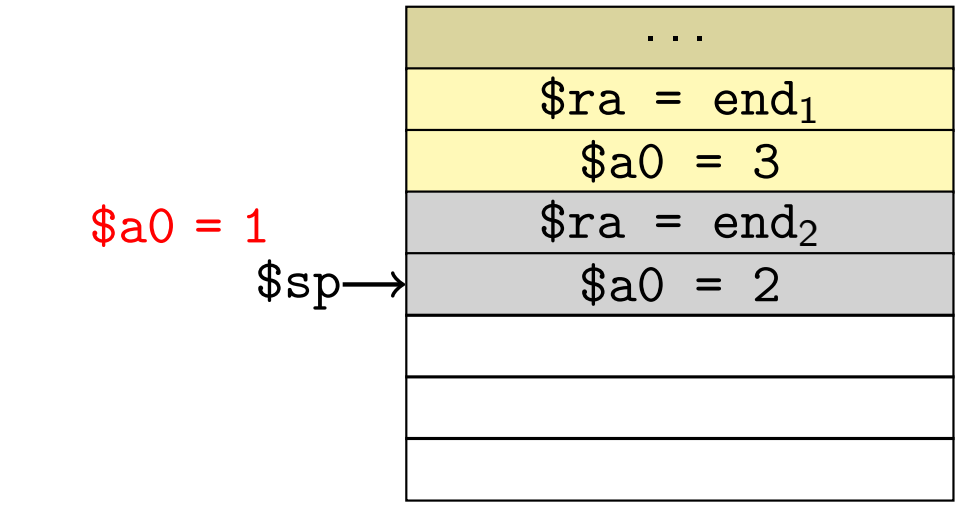


Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

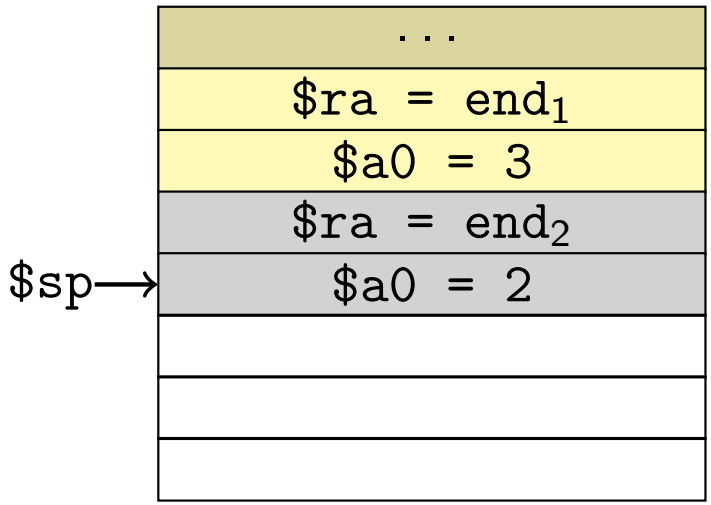
$a0 = 1



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

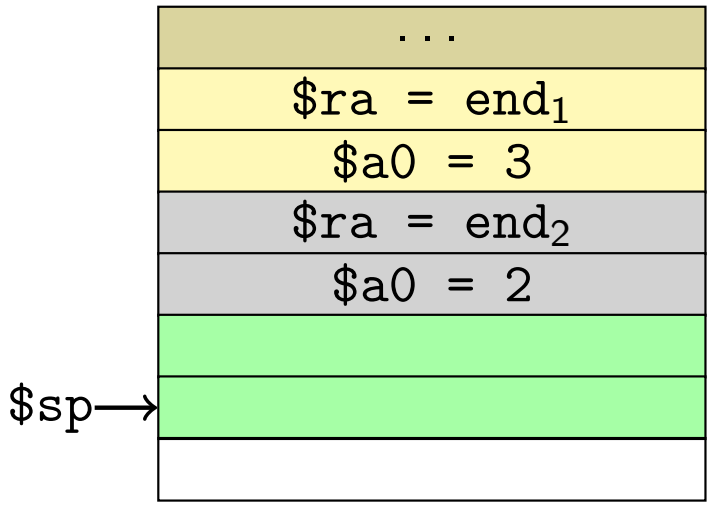
```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

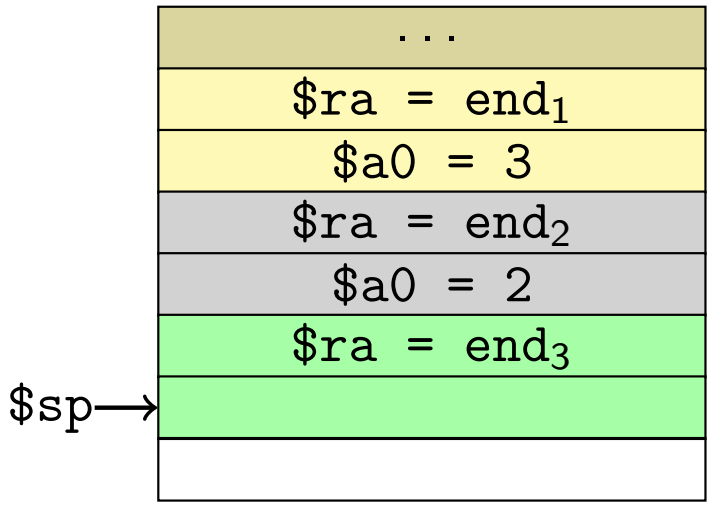
```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

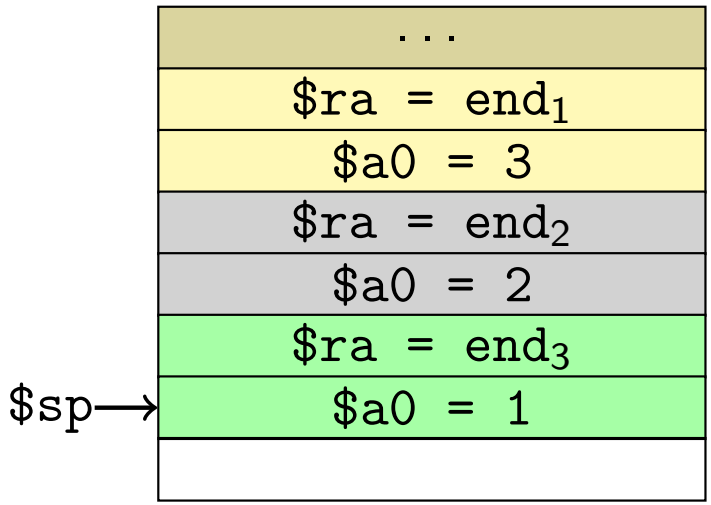
```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

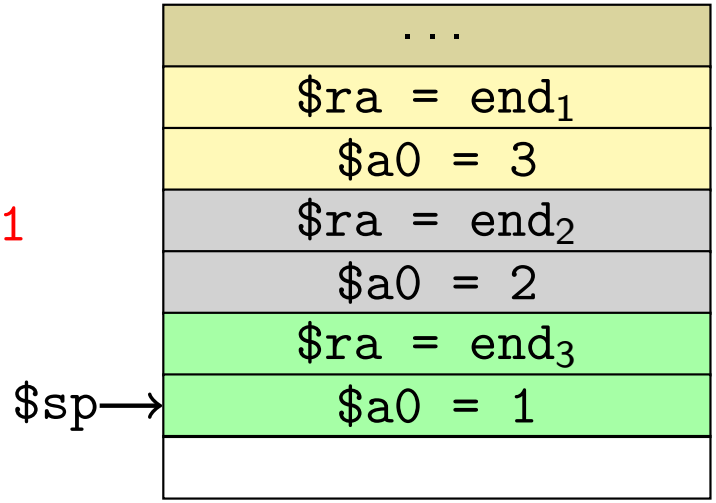


Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

$t0 = 1

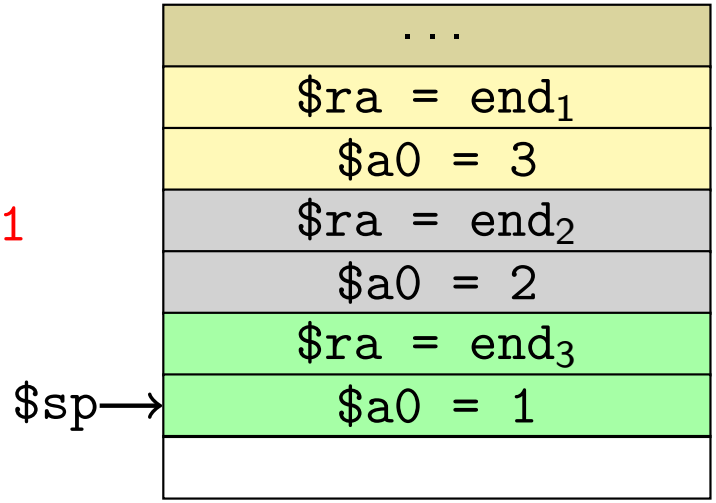


Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

$t0 = 1

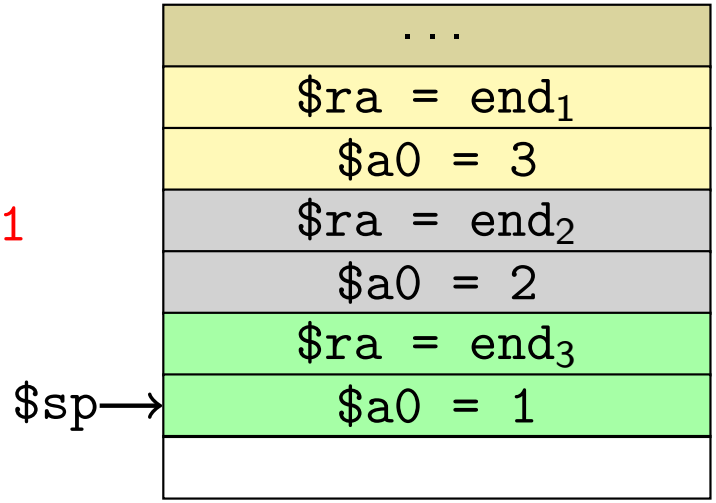


Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

$v0 = 1



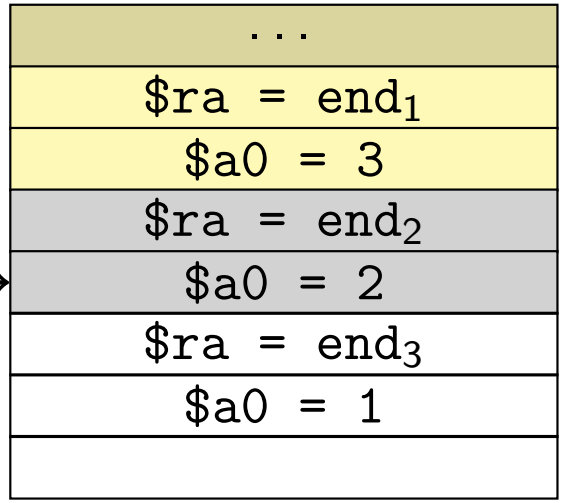
Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Restaura a pilha

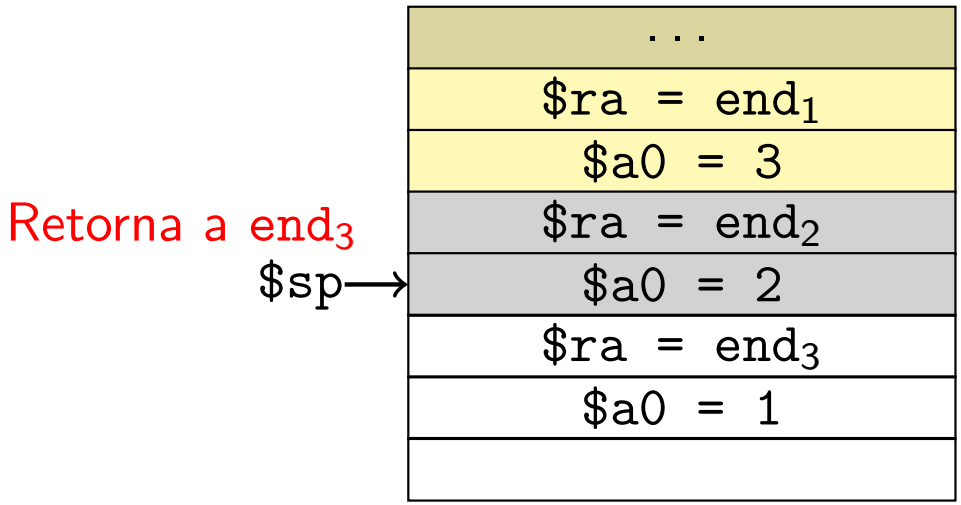
$sp⟶



Programando em MIPS

Pilha de Execução – Exemplo 4 (fat(3))

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



Programando em MIPS

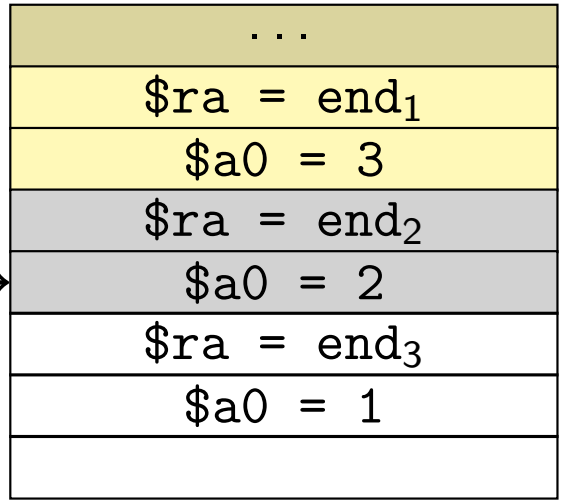
Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
```

```
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Restaura $a0

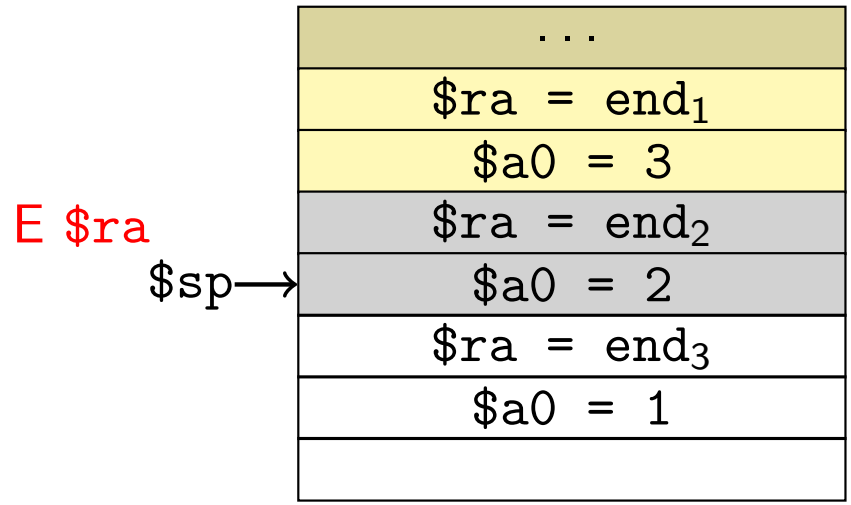
$sp →



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

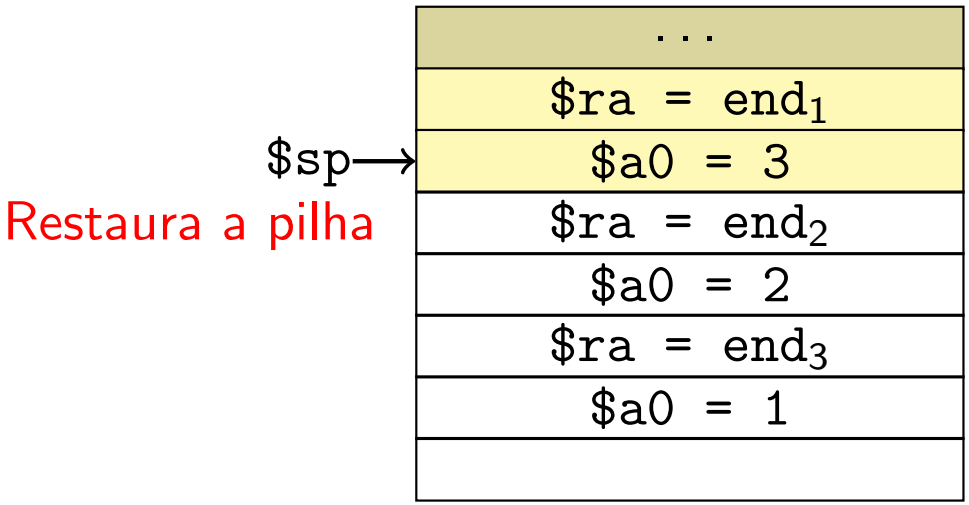


Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Restaura a pilha

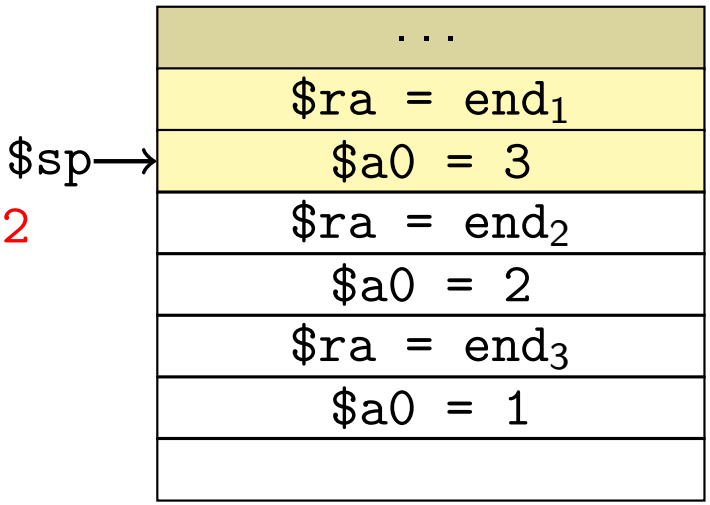


Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

$v0 = 2

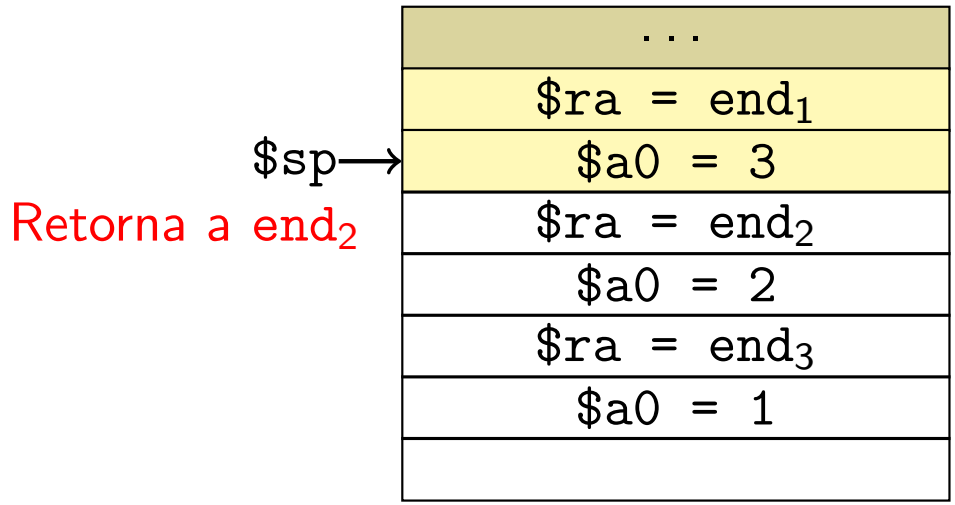


Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
```

```
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

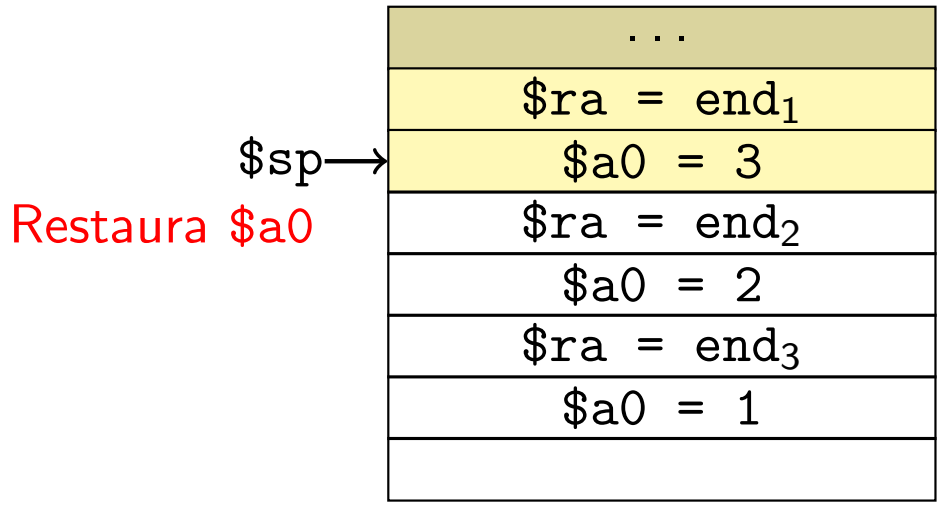


Programando em MIPS

Pilha de Execução – Exemplo 4 (fat(3))

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
```

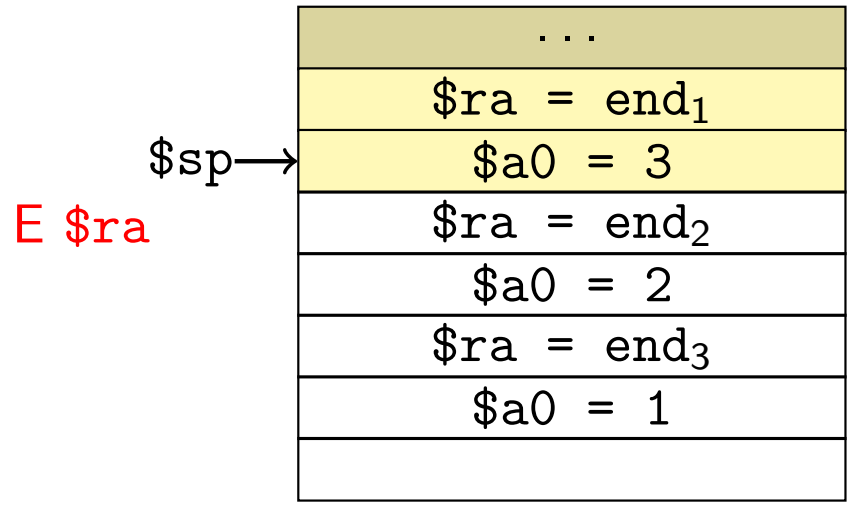
```
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```



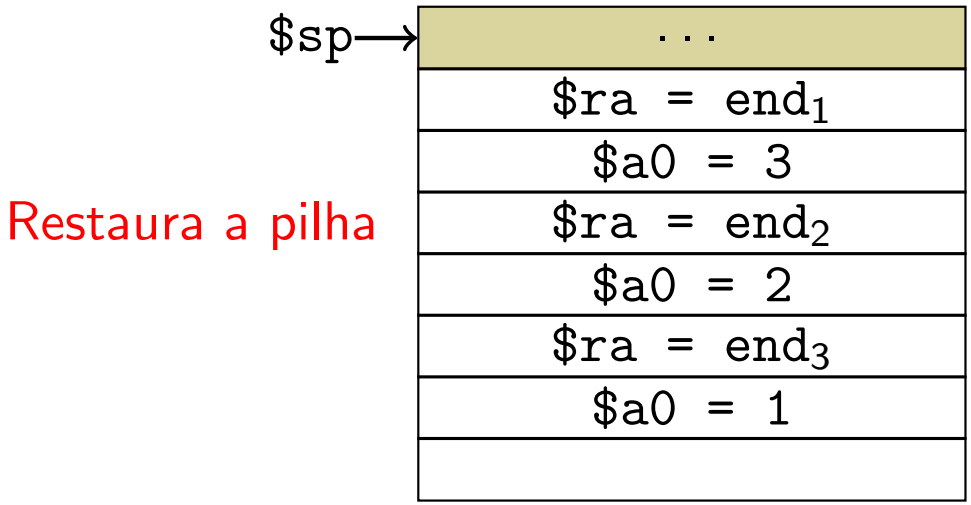
Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
```

```
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Restaura a pilha

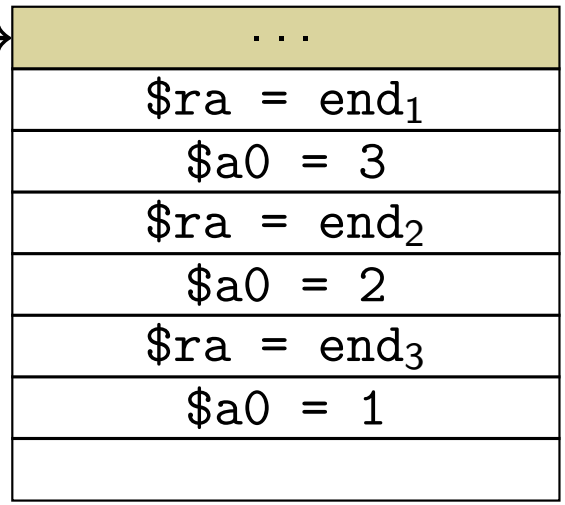


Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

$v0 = 6

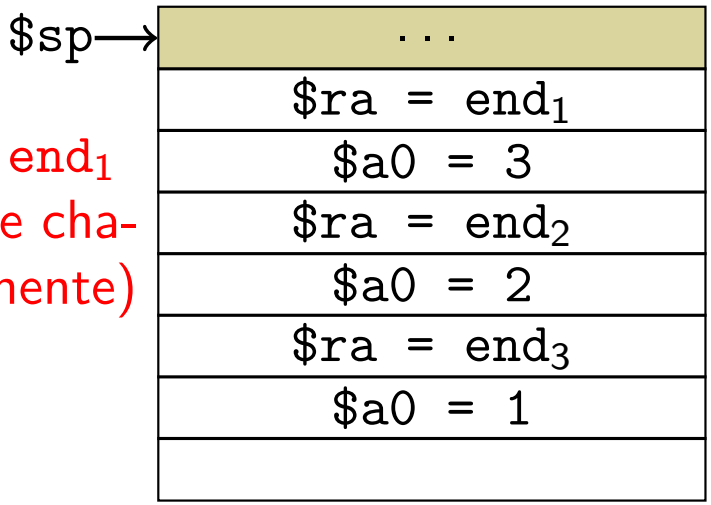


Programando em MIPS

Pilha de Execução – Exemplo 4 (`fat(3)`)

```
fat:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)
slti $t0, $a0, 2
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra
L1:
addi $a0, $a0, -1
jal fat
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $a0
jr $ra
```

Retorna a end_1 (a rotina que chamou inicialmente)



Programando em MIPS

Pilha de Execução – Mais Parâmetros

- Até 4 parâmetros usamos os registradores $a0-$a3
  - Precisando de mais, usamos a pilha

Programando em MIPS

Pilha de Execução – Mais Parâmetros

- Até 4 parâmetros usamos os registradores $a0-$a3
  - Precisando de mais, usamos a pilha
- Mas onde colocar na pilha?

Programando em MIPS

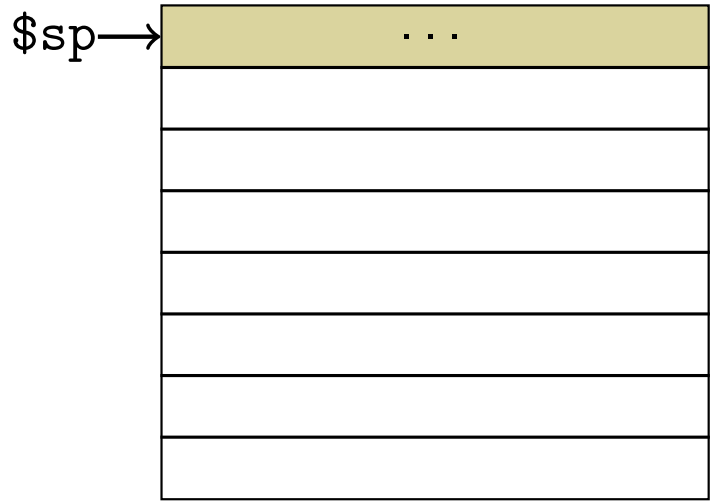
Pilha de Execução – Mais Parâmetros

- Até 4 parâmetros usamos os registradores $a0-$a3
  - Precisando de mais, usamos a pilha
- Mas onde colocar na pilha?
  - Imediatamente acima do `$fp` para a nova sub-rotina
  - Assim, ela poderá acessá-los via o `$fp`

Programando em MIPS

Pilha de Execução – Mais Parâmetros

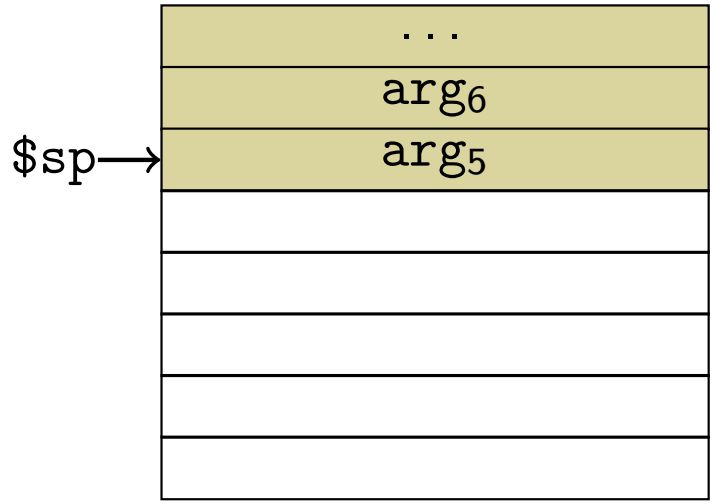
- Antes de chamar a sub-rotina a rotina empilha os argumentos



Programando em MIPS

Pilha de Execução – Mais Parâmetros

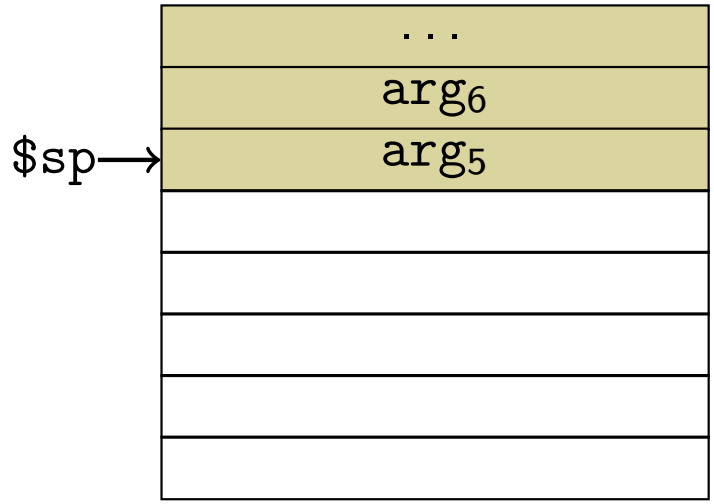
- Antes de chamar a sub-rotina a rotina empilha os argumentos



Programando em MIPS

Pilha de Execução – Mais Parâmetros

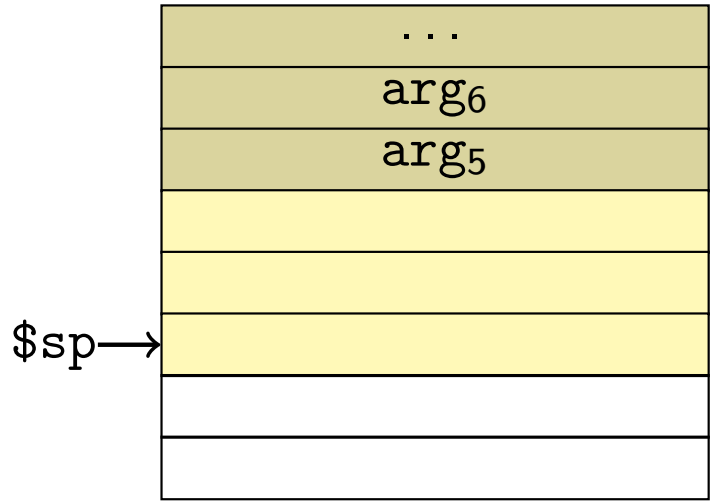
- Antes de chamar a sub-rotina a rotina empilha os argumentos
- E então chama a sub-rotina, que define seu `$sp`
  - Criando espaço para o que for modificar (`$fp`, `$ra` etc.)



Programando em MIPS

Pilha de Execução – Mais Parâmetros

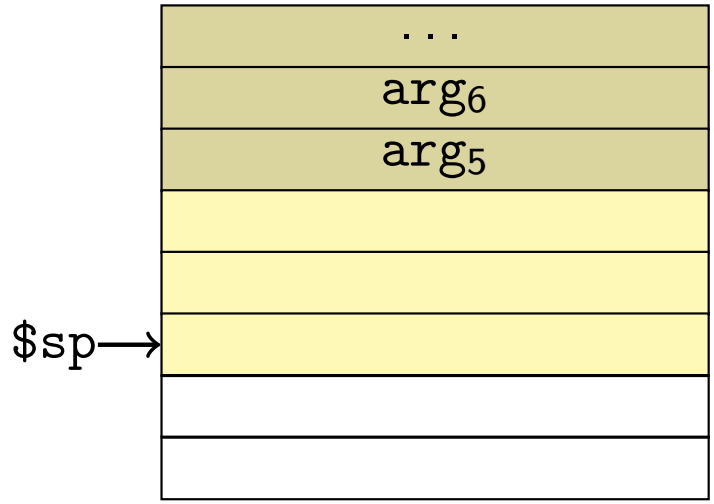
- Antes de chamar a sub-rotina a rotina empilha os argumentos
- E então chama a sub-rotina, que define seu `$sp`
  - Criando espaço para o que for modificar (`$fp`, `$ra` etc.)



Programando em MIPS

Pilha de Execução – Mais Parâmetros

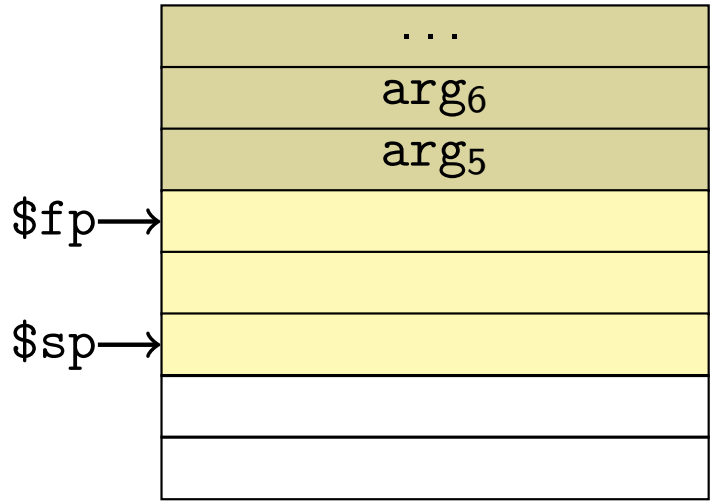
- Antes de chamar a sub-rotina a rotina empilha os argumentos
- E então chama a sub-rotina, que define seu `$sp`
  - Criando espaço para o que for modificar (`$fp`, `$ra` etc.)
- Definindo, por fim, seu `$fp`



Programando em MIPS

Pilha de Execução – Mais Parâmetros

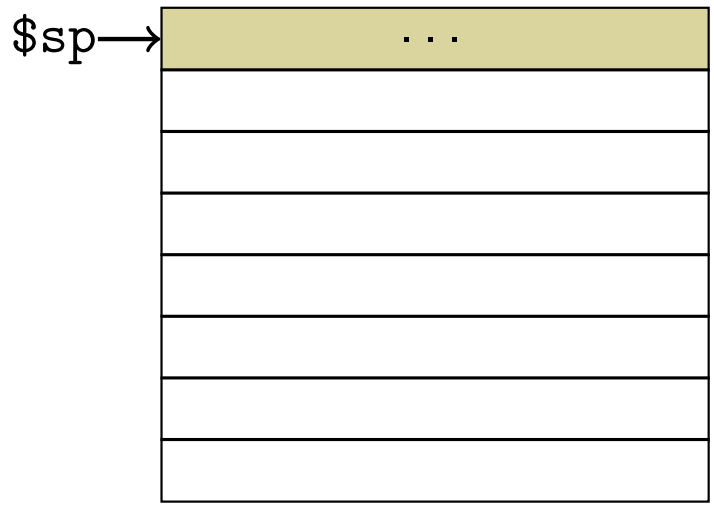
- Antes de chamar a sub-rotina a rotina empilha os argumentos
- E então chama a sub-rotina, que define seu `$sp`
  - Criando espaço para o que for modificar (`$fp`, `$ra` etc.)
- Definindo, por fim, seu `$fp`



Programando em MIPS

Pilha de Execução – Mais Parâmetros

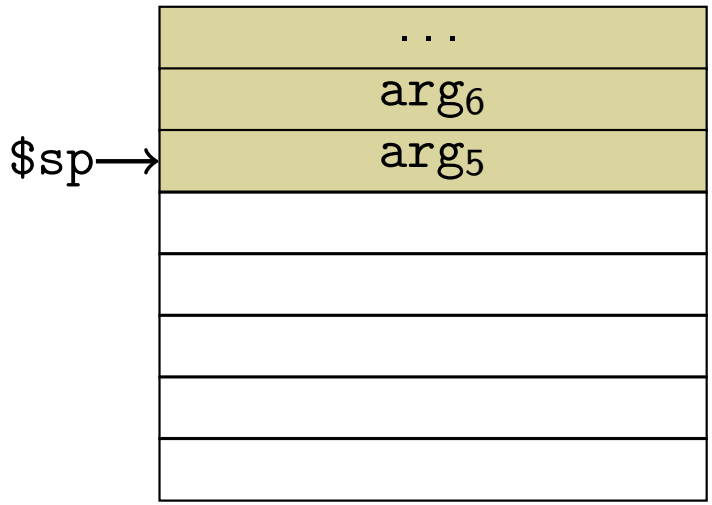
- Note que o $sp da rotina original mudou



Programando em MIPS

Pilha de Execução – Mais Parâmetros

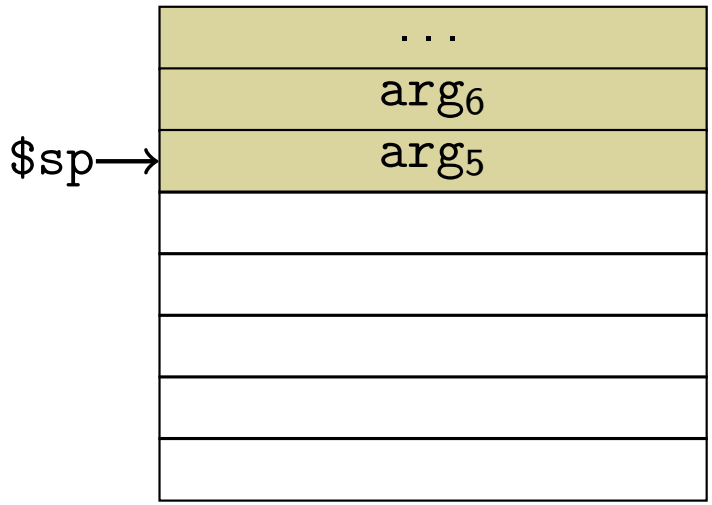
- Note que o $sp da rotina original mudou



Programando em MIPS

Pilha de Execução – Mais Parâmetros

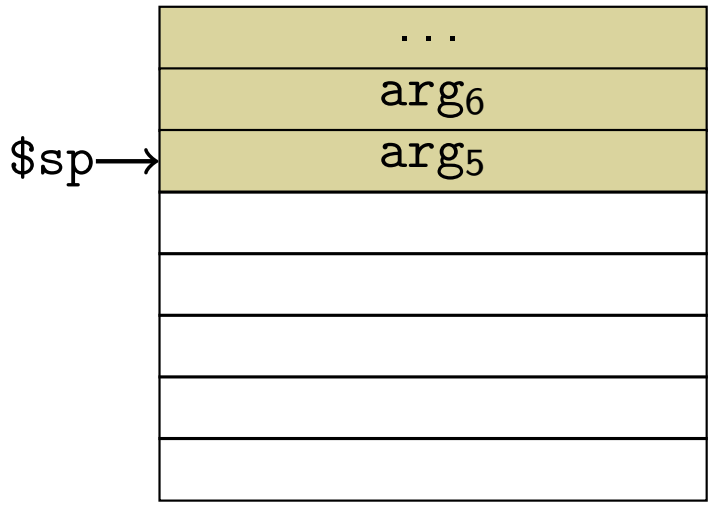
- Note que o `$sp` da rotina original mudou
  - Esta é uma das situações em que ajuda ter um `$fp`, para evitarmos ter que rever o endereço de cada elemento na pilha para a rotina



Programando em MIPS

Pilha de Execução – Mais Parâmetros

- Note que o `$sp` da rotina original mudou
  - Esta é uma das situações em que ajuda ter um `$fp`, para evitarmos ter que rever o endereço de cada elemento na pilha para a rotina



- E é por isso que definimos o `$fp` após o `$sp`
  - Para podermos guardar o `$fp` da rotina anterior na pilha

Referências

1. Patterson, D.A.; Hennessy, J.L. (2013): Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann. 5ª ed.
   - Para detalhes sobre MIPS consulte também o Apêndice A
2. `https://www.embarcados.com.br/arquitetura-de-conjunto-de-instrucoes-mips/`
   - Curso bastante completo, em português
3. `https://sites.cs.ucsb.edu/~franklin/64/lectures/mipsassemblytutorial.pdf`
   - Tutorial mais profundo
4. `https://youtu.be/y9Wv1RVbbNA`
   - Funcionamento geral da pilha de execução

Referências

1. https://courses.cs.washington.edu/courses/cse410/09sp/examples/MIPSCallingConventionsSummary.pdf
   - Exemplos de funcionamento da pilha de execução