MICROPROCESSADORES II (EMA864315)

SUB-ROTINAS E PILHA

1º SEMESTRE / 2019

Alexandro Baldassin

MATERIAL DIDÁTICO

- **Harris & Harris**
  - 6.4.6 – Procedure Calls
- **Patterson & Hennessy (4a edição)**
  - 2.8 – Supporting Procedures in Computer Hardware
- **Usem as referências acima somente para entender os conceitos. As instruções para procedimentos e formato da pilha que usaremos difere dos apresentados acima**
  - Melhor referência: *Nios II Processor Reference Handbook, capítulo 7*

SUB-ROTINAS

- **Sub-rotinas são essenciais para o desenvolvimento de software modularizado e estruturado**
  - Programação procedural
- **Sub-rotinas permitem aos programadores se concentrarem em apenas uma parte da tarefa por vez**
- **Parâmetros funcionam como uma interface entre uma sub-rotina e o resto do programa**
  - Dados podem ser passados para a sub-rotina e retornados por essa

EXECUÇÃO DE SUB-ROTINAS

- **Na execução de uma sub-rotina, o programa deve seguir 6 passos básicos**

1. Colocar os parâmetros em algum lugar onde a sub-rotina possa acessá-los
2. Transferir controle para a sub-rotina
3. Reservar recursos (memória) necessários à sub-rotina
4. Realizar o trabalho necessário
5. Colocar o resultado em algum lugar acessível a quem chamou a sub-rotina
6. Retornar controle ao ponto de origem (quem chamou)

EXECUÇÃO DE SUB-ROTINAS

- **Na execução de uma sub-rotin… …uir 6 passos básicos**

Memória (pilha) ou registradores?

1. Colocar os parâmetros em algum lugar onde a sub-rotina possa acessá-los
2. Transferir controle para a sub-rotina
3. Reservar recursos (memória) necessários à sub-rotina
4. Realizar o trabalho necessário
5. Colocar o resultado em algum lugar acessível a quem chamou a sub-rotina
6. Retornar controle ao ponto de origem (quem chamou)

EXECUÇÃO DE SUB-ROTINAS

- **Na execução de uma sub-rotina, o programa deve seguir 6 passos básicos**

Instrução do processador

1. Colocar os parâmetros em algum lu… acessá-los
2. Transferir controle para a sub-rotina
3. Reservar recursos (memória) necessários à sub-rotina
4. Realizar o trabalho necessário
5. Colocar o resultado em algum lugar acessível a quem chamou a sub-rotina
6. Retornar controle ao ponto de origem (quem chamou)

EXECUÇÃO DE SUB-ROTINAS

- **Na execução de uma sub-rotina, o programa deve seguir 6 passos básicos**

1. Colocar os parâmetros em algum l
   acessá-los
2. Transferir controle para a sub-rotina
3. Reservar recursos (memória) necessários à sub-rotina
4. Realizar o trabalho necessário
5. Colocar o resultado em algum lugar acessível a quem chamou a sub-rotina
6. Retornar controle ao ponto de origem (quem chamou)

Configurar *stack frame*

EXECUÇÃO DE SUB-ROTINAS

- **Na execução de uma sub-rotina, o programa deve seguir 6 passos básicos**

1. Colocar os parâmetros em algum lugar onde a sub-rotina possa acessá-los
2. Transferir controle para a sub-rotina
3. Reservar recursos (memória) necessários à sub-rotina
4. Realizar o trabalho necessário
5. Colocar o resultado em algum lugar acessível a quem chamou a sub-rotina
6. Retornar controle ao ponto de origem (quem chamou)

EXECUÇÃO DE SUB-ROTINAS

- **Na execução de uma sub-rotina, o programa deve seguir 6 passos básicos**

1. Colocar os parâmetros em algum lugar onde a sub-rotina possa acessá-los
2. Transferir controle para a sub-rotin
3. Reservar recursos (memória) nece
4. Realizar o trabalho necessário
5. Colocar o resultado em algum lugar acessível a quem chamou a sub-rotina
6. Retornar controle ao ponto de origem (quem chamou)

Novamente... pilha ou registradores ?

EXECUÇÃO DE SUB-ROTINAS

- **Na execução de uma sub-rotina, o programa deve seguir 6 passos básicos**

1. Colocar os parâmetros em algum lugar onde a sub-rotina possa acessá-los
2. Transferir controle para a sub-rotin
3. Reservar recursos (memória) nece
4. Realizar o trabalho necessário
5. Colocar o resultado em algum luga ub-rotina
6. Retornar controle ao ponto de origem (quem chamou)

Instrução do processador

STACK FRAME

- **Também chamado de registro de ativação (*activation record*)**

- **Área de memória (pilha) que armazena o estado da sub-rotina, e contém geramente as seguintes informações**
  - Os argumentos passados para a sub-rotina
  - O endereço de retorno para quem chamou (caller)
  - Espaço para as variáveis locais à sub-rotina

ALGUMAS PERGUNTAS

- **Passar os parâmetros através da pilha ou registradores?**
  - Se registradores, quais ?
- **Qual o formato (conteúdo) do stack frame ?**
- **Quais registradores devem ser salvos pela sub-rotina chamada (*callee*) e quais pelo programa que a chamou (*caller*) ?**
- **Como retornar o(s) resultado(s) ?**
  - Se registradores, quais ?

APPLICATION BINARY INTERFACE (ABI)

- **As respostas para as perguntas anteriores são respondidas pela ABI**

- **ABI é uma *convenção* que determina a interface entre o processador e o sistema (sistema operacional + compilador)**
  - Quais os tamanhos para os tipos de dados básicos
  - Quais registradores precisam ser salvos pela sub-rotina chamada
  - Qual o layout do stack frame

ABI ALTERA NIOS II

- **Presente no capítulo 7 da referência "*Nios II Processor Reference Handbook*"**
  - Disponível no site da disciplina
- **Nos nossos programas em linguagem de montagem vamos seguir essa ABI**
  - Essencial quando estivermos mesclando código assembly e C

USO DE REGISTRADORES

- **A ABI define o seguinte uso**

Registrador	Nome	Uso regular
r0	zero	0x00000000
r1	at	Temporário usado pelo montador
r2		Valor de retorno (32 bits menos significativos)
r3		Valor de retorno (32 bits mais significativos)
r4 ... r7		Registradores de argumento (primeiro ao quarto, respectivamente)
r8 ... r15		Registradores de propósito geral, salvos pelo *caller*
r16 ... r23		Registradores de propósito geral, salvos pelo *callee*
r27	sp	Stack pointer
r28	fp	Frame pointer
r31	ra	Endereço de retorno

Os registradores omitidos serão vistos mais adiante na disciplina – procurar não usá-los no momento

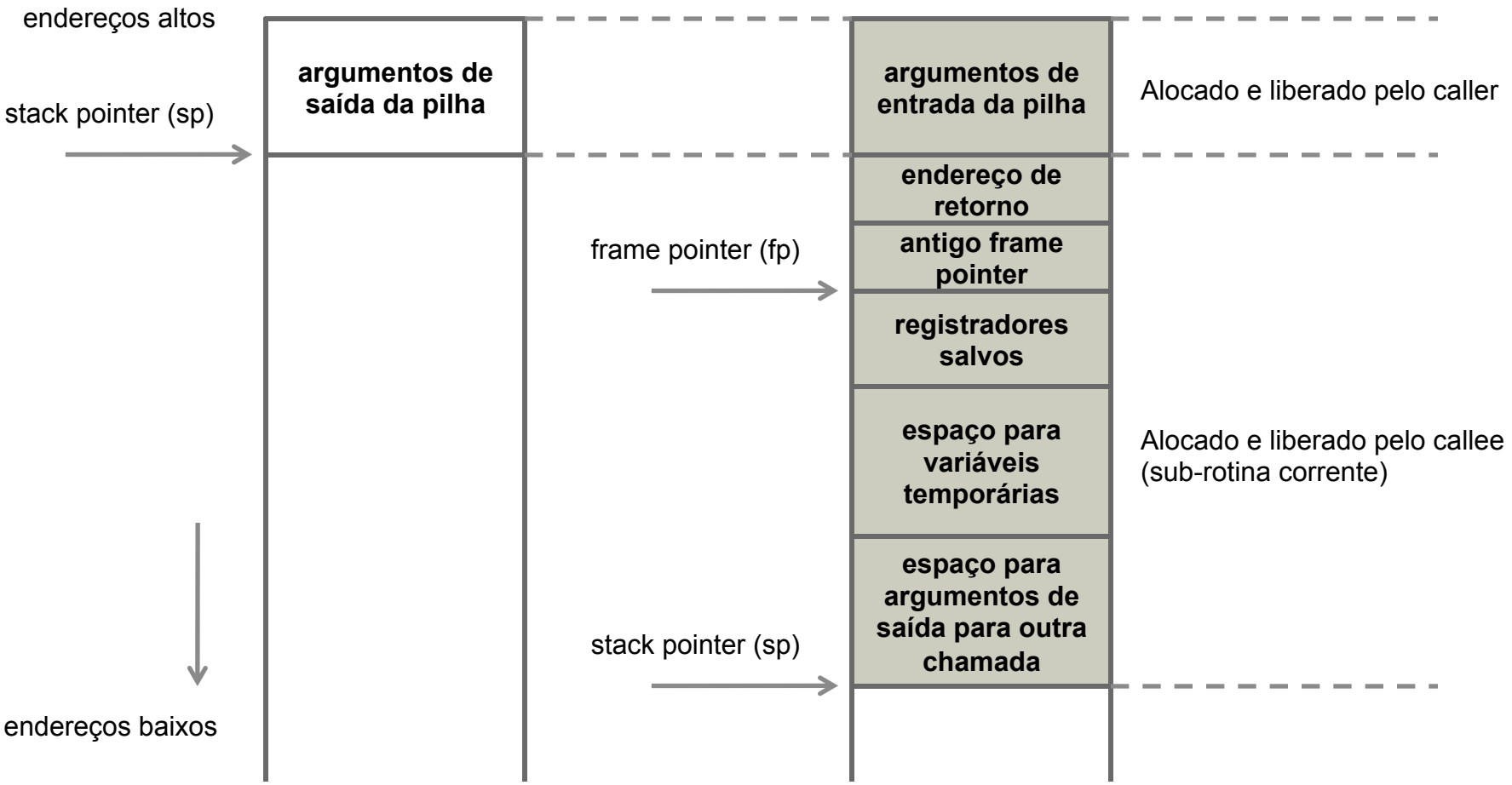
PILHA (1)

- **A pilha é a estrutura utilizada para comunicação entre o caller e o callee, e para variáveis automáticas (locais)**
  - Lembre-se: pilha –> last-in, first-out
- **Quando mais de quatro argumentos devem ser passados para uma sub-rotina, precisamos usar a pilha**
  - A ABI só fornece 4 registradores
- **Um registrador especial, *stack pointer* (sp), aponta para o último elemento inserido na pilha**

PILHA (2)

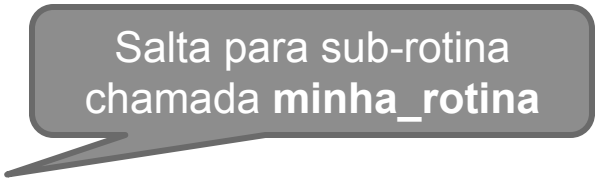
- A pilha “cresce para baixo”, ou seja, de endereços maiores para menores
- Para facilitar a manipulação do stack frame, um outro registrador, chamado de *frame pointer* (ou *base pointer*), é usado
- O frame pointer sempre aponta para o antigo frame pointer salvo na pilha (geralmente no topo da stack frame)

STACK FRAME – NIOS II



CHAMADA E RETORNO

- **A instrução do Nios II usada para saltar para sub-rotina é a call**




```
call    minha_rotina
```

- **A instrução call automaticamente já salva PC+4 (endereço de retorno) no registrador r31 (*ra*)**

- **Para retornar de uma sub-rotina, usamos a instrução ret**

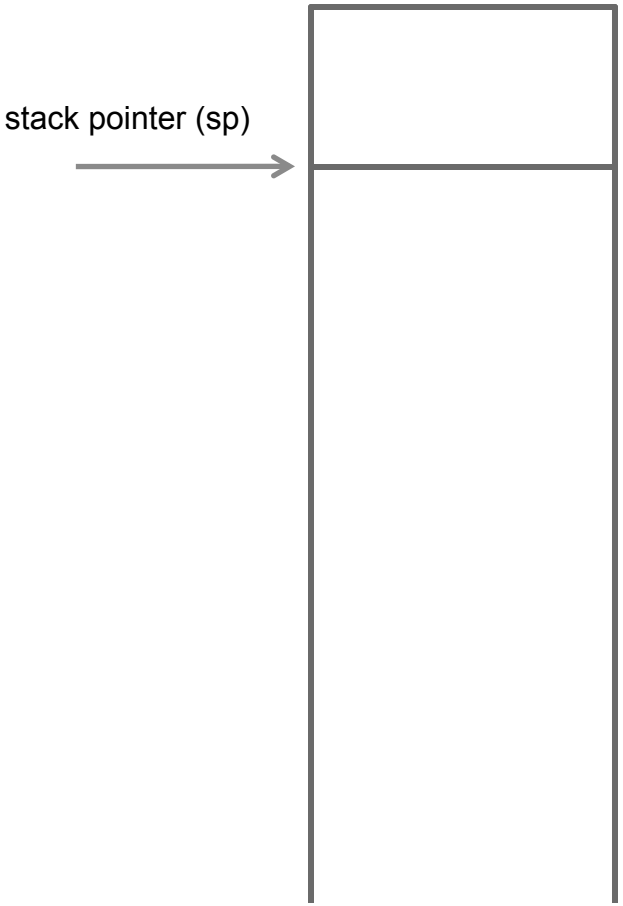
```
ret
```

PRÓLOGO E EPÍLOGO

- **Prólogo**
  - Trecho de código inicial em uma sub-rotina, responsável por configurar a stack frame
  - No Nios II, o prólogo deve:
    - Ajustar o stack pointer (alocar espaço para o frame)
    - Armazenar registradores no frame
    - Setar o frame pointer para a posição do frame pointer anterior salvo na pilha

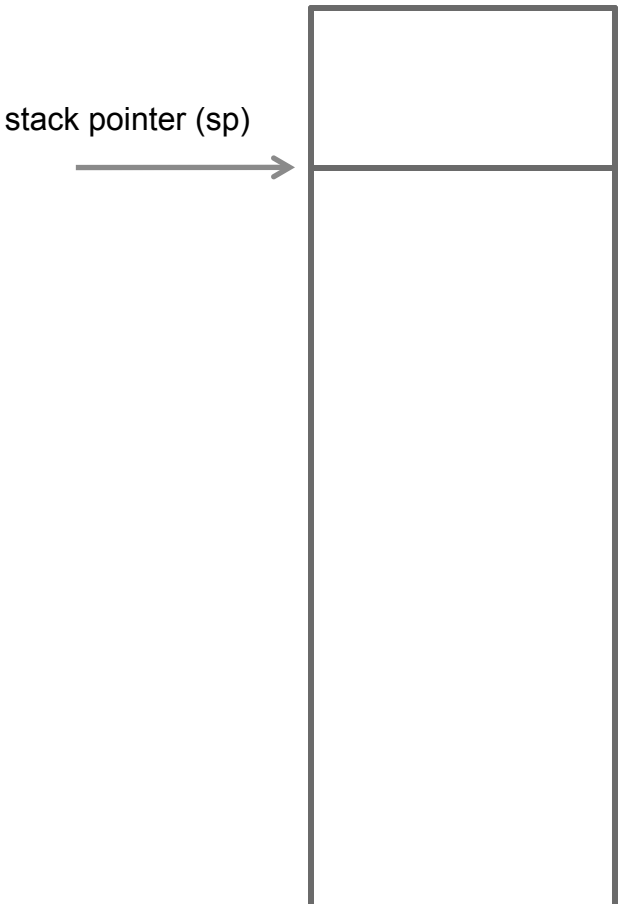
- **Epílogo**
  - Trecho de código no final da sub-rotina responsável por restaurar os registradores e o estado da pilha

EXEMPLO – PRÓLOGO



Estado atual da pilha antes da chamada

EXEMPLO – PRÓLOGO



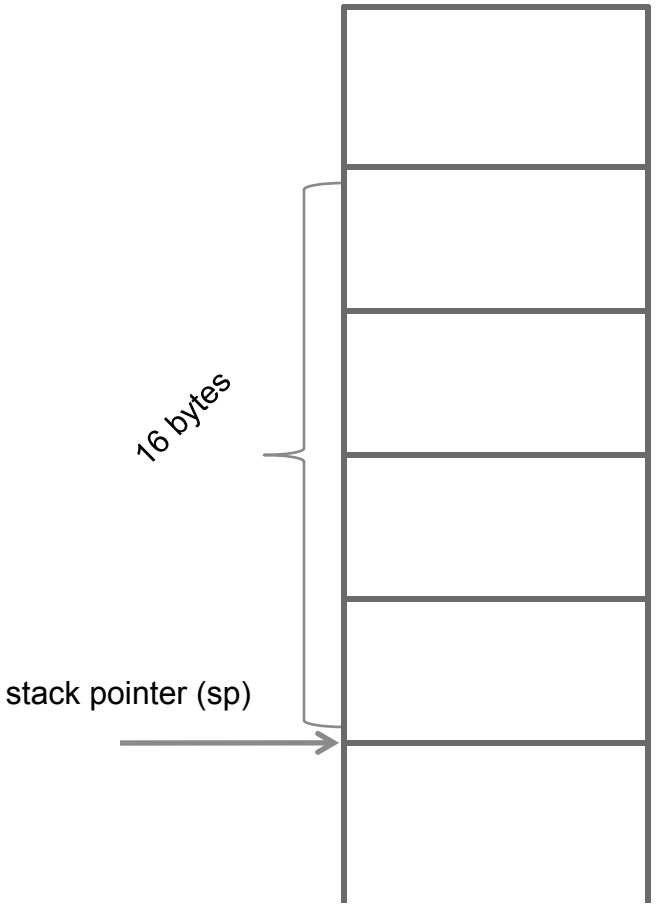
Após chamada ...

```
/* Adjust the stack pointer */
addi sp, sp, -16 /* make a 16-byte frame */

/* Store registers to the frame */
stw ra, 12(sp) /* store the return address */
stw fp, 8(sp)  /* store the frame pointer*/
stw r16, 4(sp) /* store callee-saved register */
stw r17, 0(sp) /* store callee-saved register */

/* Set the new frame pointer */
addi fp, sp, 8
```

EXEMPLO – PRÓLOGO



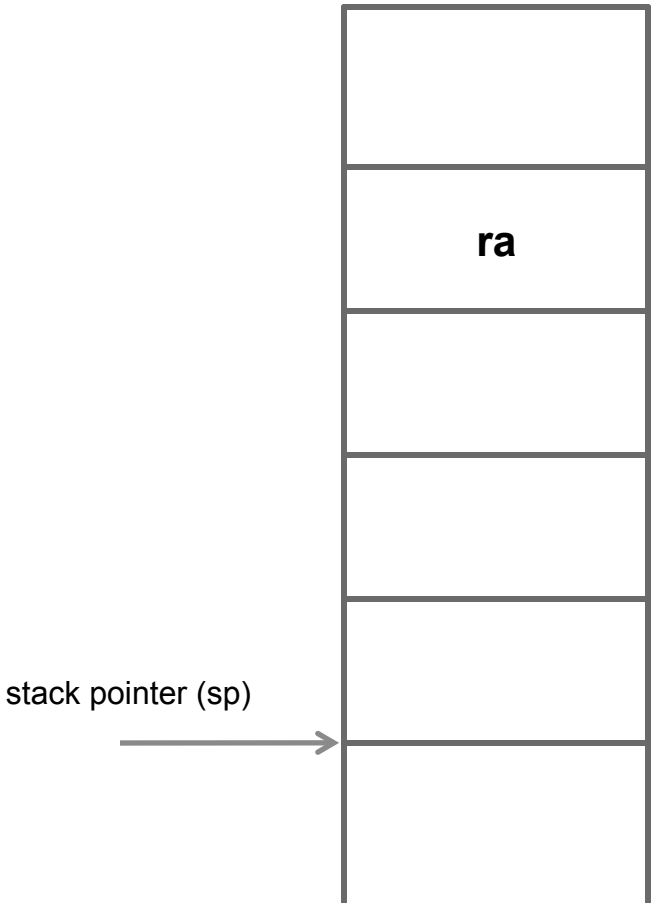
Após chamada ...

```
/* Adjust the stack pointer */
addi sp, sp, -16 /* make a 16-byte frame */

/* Store registers to the frame */
stw ra, 12(sp) /* store the return address */
stw fp, 8(sp)  /* store the frame pointer*/
stw r16, 4(sp) /* store callee-saved register */
stw r17, 0(sp) /* store callee-saved register */

/* Set the new frame pointer */
addi fp, sp, 8
```

EXEMPLO – PRÓLOGO



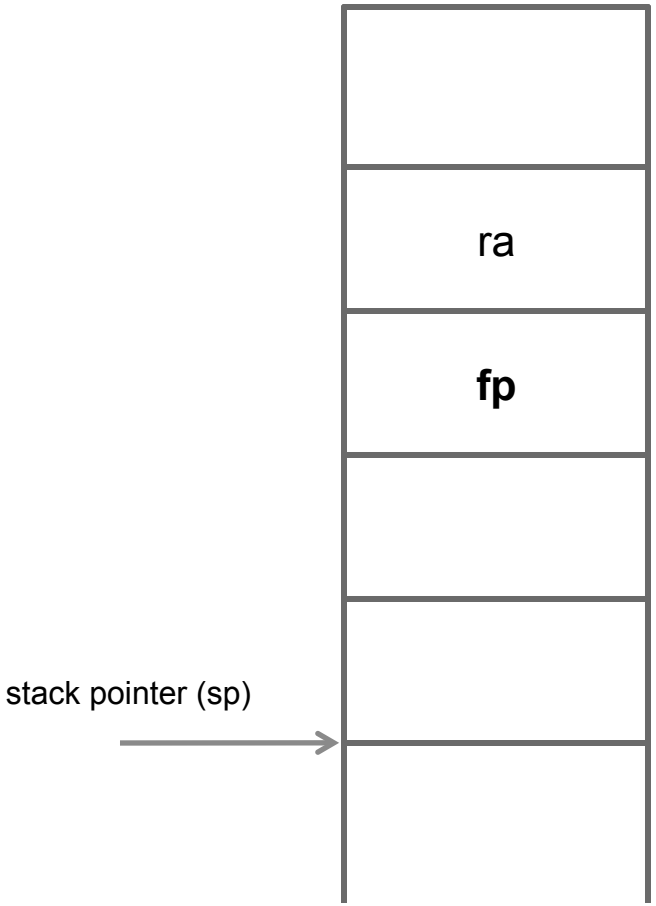
Após chamada ...

```
/* Adjust the stack pointer */
addi sp, sp, -16 /* make a 16-byte frame */

/* Store registers to the frame */
stw ra, 12(sp) /* store the return address */
stw fp, 8(sp)  /* store the frame pointer*/
stw r16, 4(sp) /* store callee-saved register */
stw r17, 0(sp) /* store callee-saved register */

/* Set the new frame pointer */
addi fp, sp, 8
```

EXEMPLO – PRÓLOGO



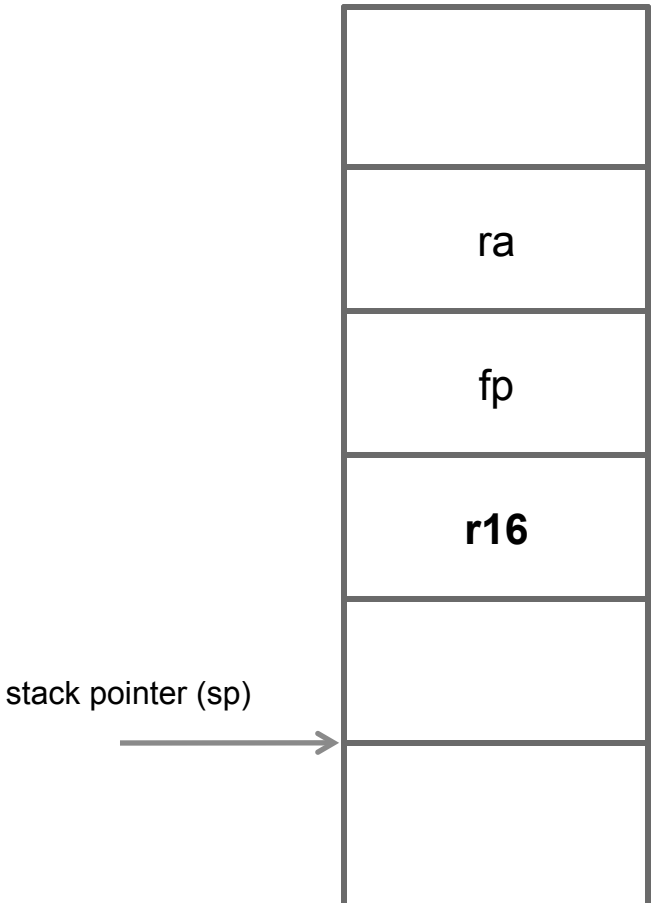
Após chamada ...

```
/* Adjust the stack pointer */
addi sp, sp, -16 /* make a 16-byte frame */

/* Store registers to the frame */
stw ra, 12(sp) /* store the return address */
stw fp, 8(sp)  /* store the frame pointer*/
stw r16, 4(sp) /* store callee-saved register */
stw r17, 0(sp) /* store callee-saved register */

/* Set the new frame pointer */
addi fp, sp, 8
```

EXEMPLO – PRÓLOGO



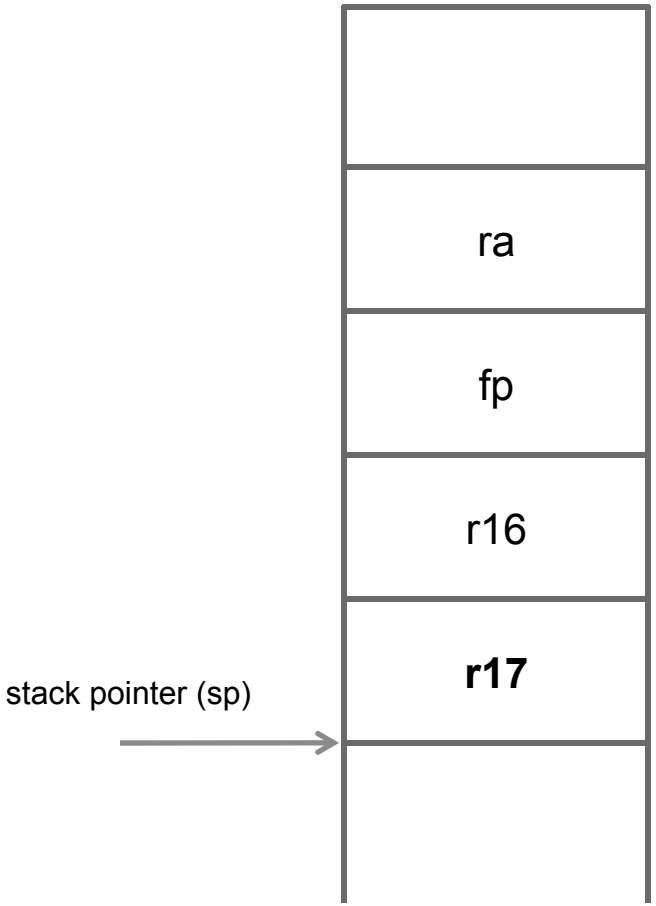
Após chamada ...

```
/* Adjust the stack pointer */
addi sp, sp, -16 /* make a 16-byte frame */

/* Store registers to the frame */
stw ra, 12(sp) /* store the return address */
stw fp, 8(sp)  /* store the frame pointer*/
stw r16, 4(sp) /* store callee-saved register */
stw r17, 0(sp) /* store callee-saved register */

/* Set the new frame pointer */
addi fp, sp, 8
```

EXEMPLO – PRÓLOGO



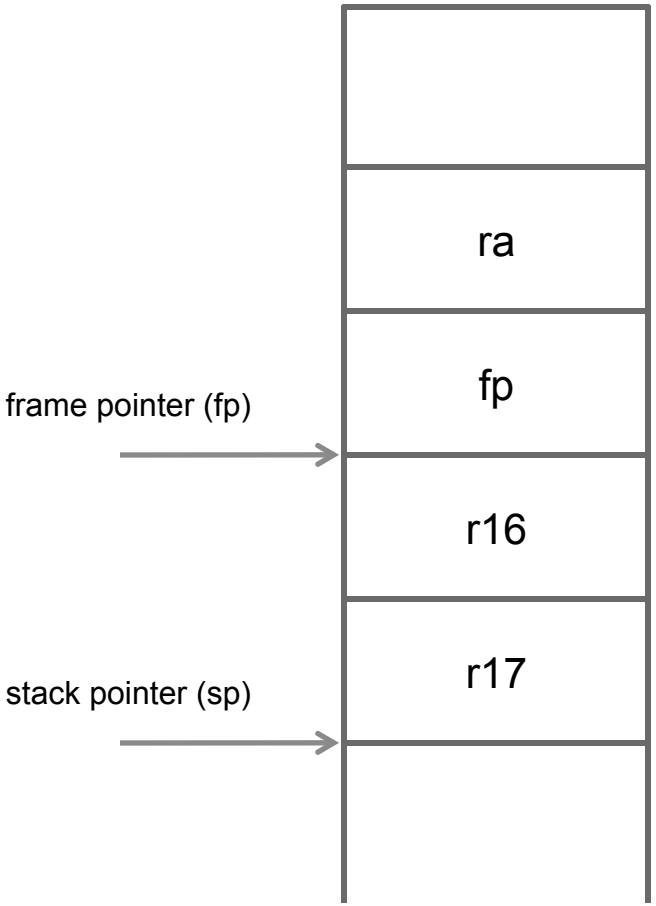
Após chamada ...

```
/* Adjust the stack pointer */
addi sp, sp, -16 /* make a 16-byte frame */

/* Store registers to the frame */
stw ra, 12(sp) /* store the return address */
stw fp, 8(sp)  /* store the frame pointer*/
stw r16, 4(sp) /* store callee-saved register */
stw r17, 0(sp) /* store callee-saved register */

/* Set the new frame pointer */
addi fp, sp, 8
```

EXEMPLO – PRÓLOGO



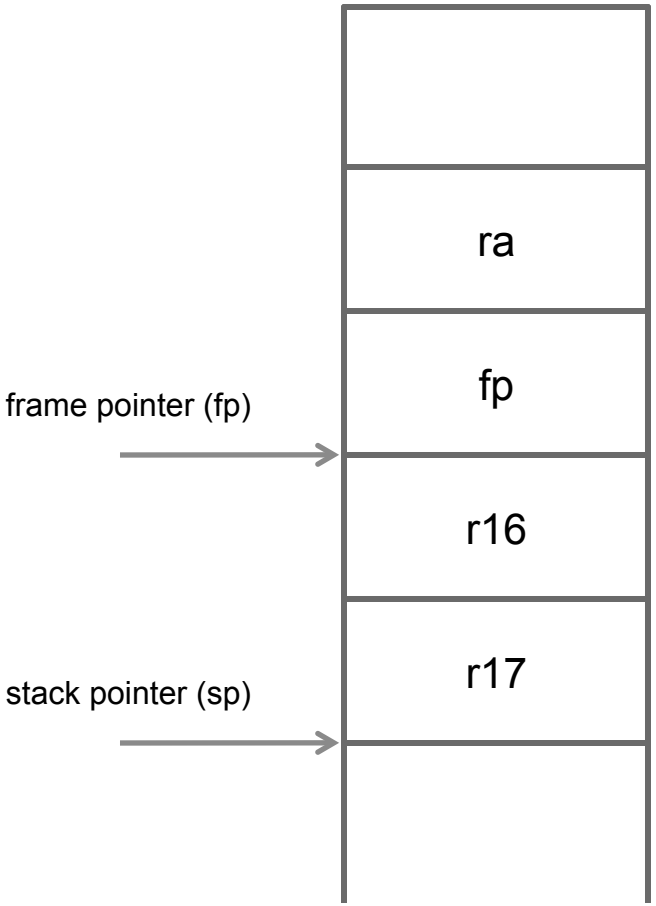
Após chamada ...

```
/* Adjust the stack pointer */
addi sp, sp, -16 /* make a 16-byte frame */

/* Store registers to the frame */
stw ra, 12(sp) /* store the return address */
stw fp, 8(sp)  /* store the frame pointer*/
stw r16, 4(sp) /* store callee-saved register */
stw r17, 0(sp) /* store callee-saved register */

/* Set the new frame pointer */
addi fp, sp, 8
```

EXEMPLO – EPÍLOGO

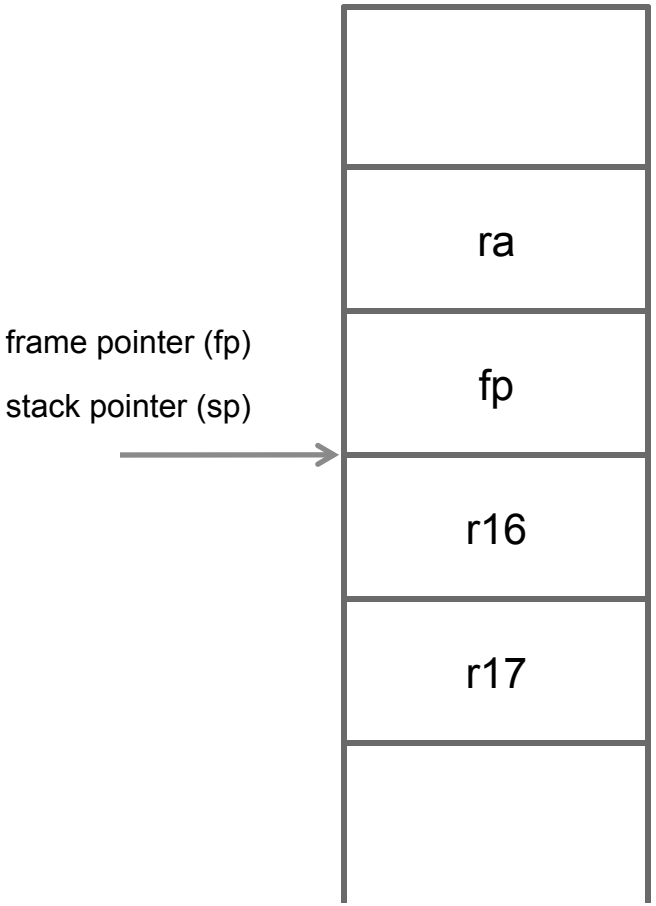


Antes do retorno ...

```
/* Restore ra, fp, sp, and registers */
mov  sp, fp
ldw  ra, 4(sp)
ldw  fp, 0(sp)
ldw  r16, -4(sp)
ldw  r17, -8(sp)
addi sp, sp, 8

/* Return from subroutine */
ret
```

EXEMPLO – EPÍLOGO

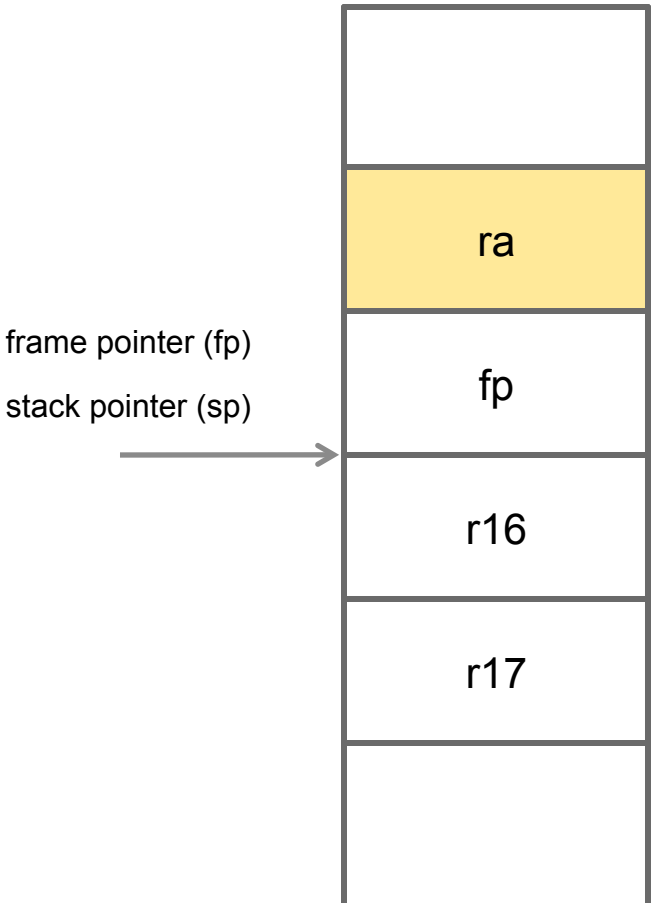


Antes do retorno ...

```
/* Restore ra, fp, sp, and registers */
mov  sp, fp
ldw  ra, 4(sp)
ldw  fp, 0(sp)
ldw  r16, -4(sp)
ldw  r17, -8(sp)
addi sp, sp, 8

/* Return from subroutine */
ret
```

EXEMPLO – EPÍLOGO

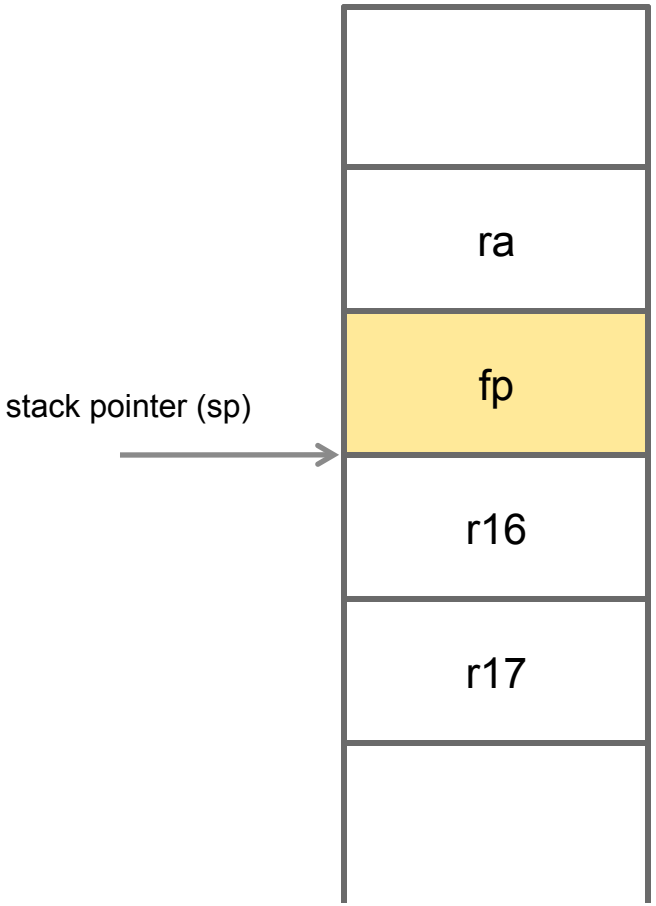


Antes do retorno ...

```
/* Restore ra, fp, sp, and registers */
mov  sp, fp
ldw  ra, 4(sp)
ldw  fp, 0(sp)
ldw  r16, -4(sp)
ldw  r17, -8(sp)
addi sp, sp, 8

/* Return from subroutine */
ret
```

EXEMPLO – EPÍLOGO

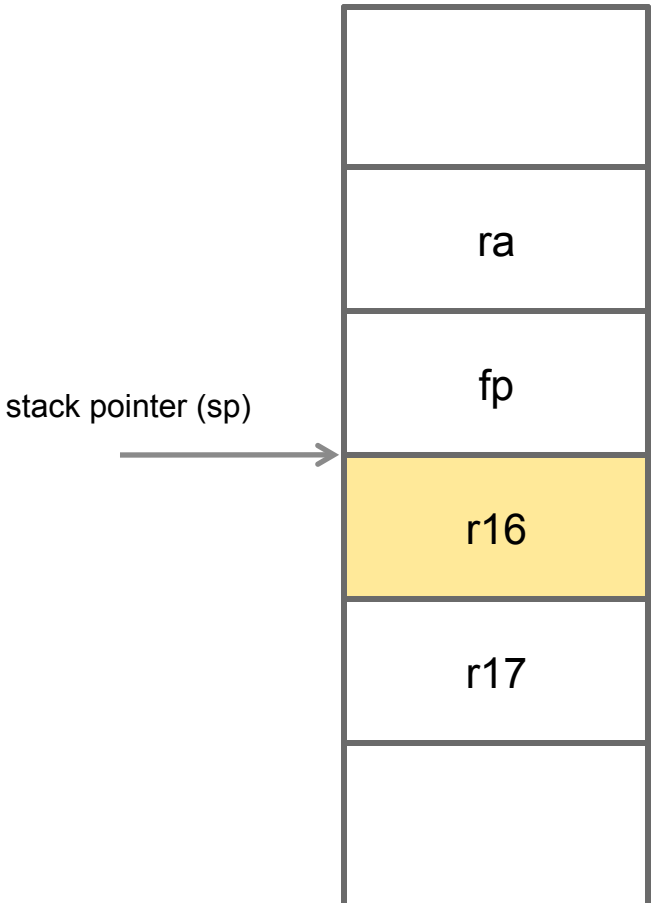


Antes do retorno ...

```
/* Restore ra, fp, sp, and registers */
mov  sp, fp
ldw  ra, 4(sp)
ldw  fp, 0(sp)
ldw  r16, -4(sp)
ldw  r17, -8(sp)
addi sp, sp, 8

/* Return from subroutine */
ret
```

EXEMPLO – EPÍLOGO

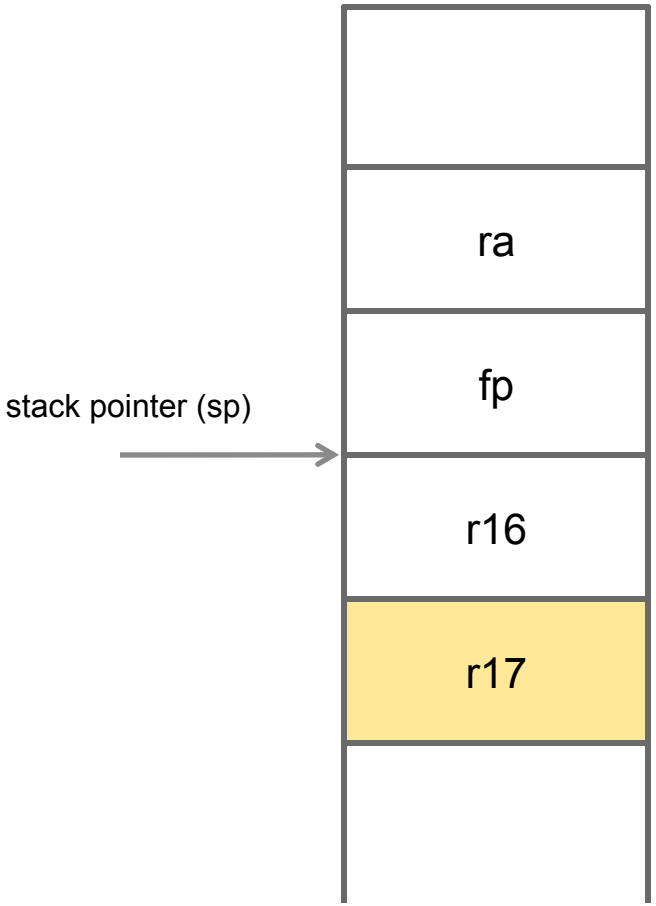


Antes do retorno ...

```
/* Restore ra, fp, sp, and registers */
mov  sp, fp
ldw  ra, 4(sp)
ldw  fp, 0(sp)
ldw  r16, -4(sp)
ldw  r17, -8(sp)
addi sp, sp, 8

/* Return from subroutine */
ret
```

EXEMPLO – EPÍLOGO

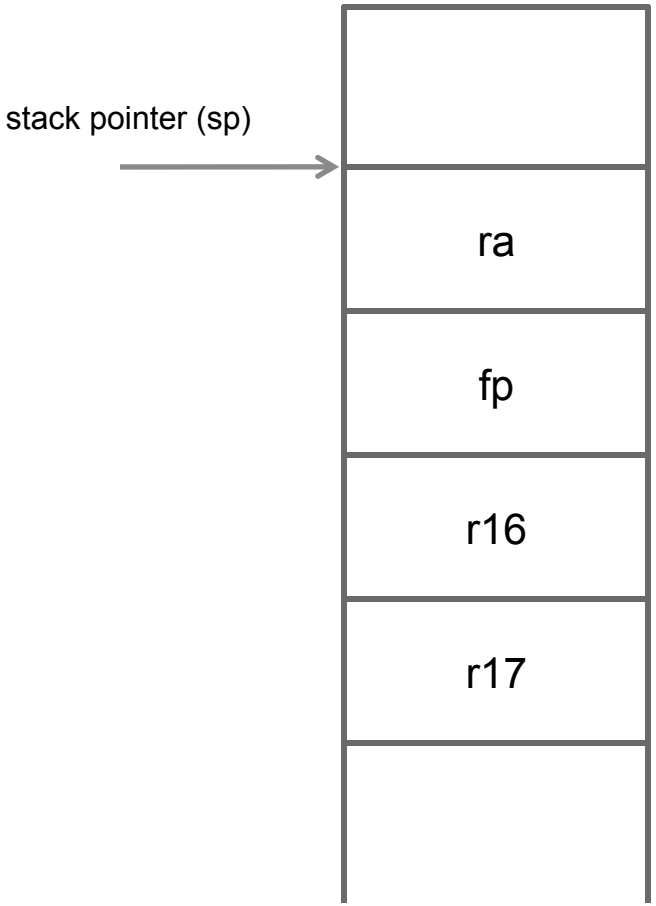


Antes do retorno ...

```
/* Restore ra, fp, sp, and registers */
mov   sp, fp
ldw   ra, 4(sp)
ldw   fp, 0(sp)
ldw   r16, -4(sp)
ldw   r17, -8(sp)
addi sp, sp, 8

/* Return from subroutine */
ret
```

EXEMPLO – EPÍLOGO

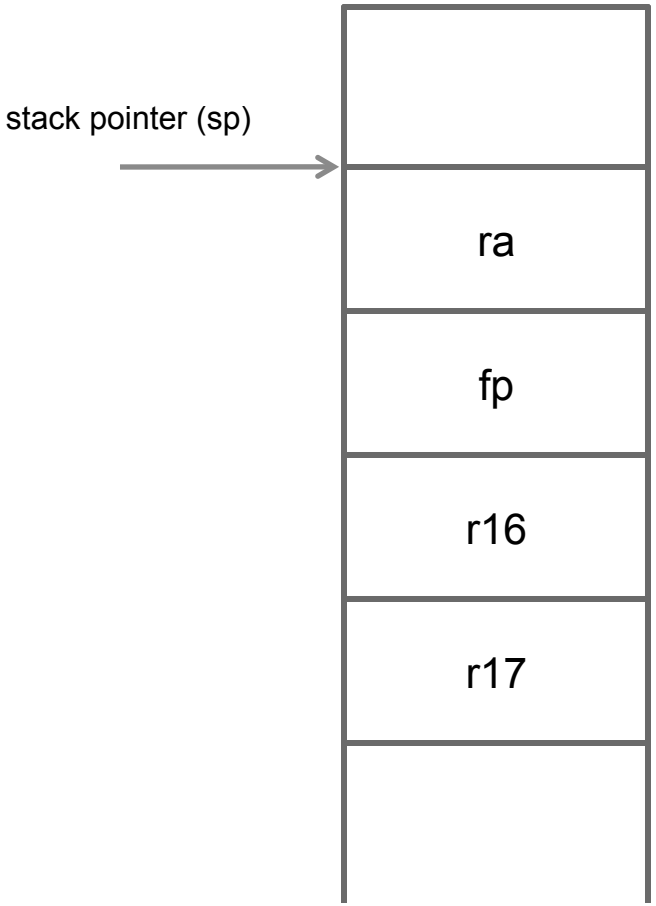


Antes do retorno ...

```
/* Restore ra, fp, sp, and registers */
mov   sp, fp
ldw   ra, 4(sp)
ldw   fp, 0(sp)
ldw   r16, -4(sp)
ldw   r17, -8(sp)
addi sp, sp, 8

/* Return from subroutine */
ret
```

EXEMPLO – EPÍLOGO



Antes do retorno ...

```
/* Restore ra, fp, sp, and registers */
mov  sp, fp
ldw  ra, 4(sp)
ldw  fp, 0(sp)
ldw  r16, -4(sp)
ldw  r17, -8(sp)
addi sp, sp, 8

/* Return from subroutine */
ret
```