



Aula 6

Assembly RISC-V

Procedimentos





Instruções de suporte a procedimentos

- ♦ Passos em um procedimento:
 1. Colocar os parâmetros em um lugar onde o procedimento possa acessá-los ($a_0 \sim a_7$);
 2. Transferir o controle para o procedimento (jal);
 3. Adquirir recursos de armazenamento necessários para o procedimento;
 4. Realizar a tarefa desejada;
 5. Colocar o valor de retorno em um lugar onde o programa que o chamou possa acessá-lo ($a_0 \sim a_1$);
 6. Retornar o controle para o ponto de origem (ret).



Exemplo de procedimento

C

```
main() {
    int a,b,c;
    c=soma(a,b); /* s0:=a; s1:=b; s2:=c */
    ...}
```

```
int soma(int x, int y) /* x:=a0; y:=a1 */
{ return x+y; }
```

RISC-V

endereço

```
0x00040000  add  a0,s0,zero    # x = a
0x00040004  add  a1,s1,zero    # y = b
0x00040008  jal  ra,soma      # prepara ra e j soma
0x0004000C  add  s2,a0,zero    # c=a+b
...
0x00040100  soma: add a0,a0,a1
0x00040104  jalr zero,ra,0    # retorna
```



Usando mais registradores

- ◆ Se precisar mais de 8 argumentos e/ou 2 valores de retorno?.
- ◆ Se o procedimento necessitar utilizar registradores salvos s_x ?
- ◆ Processo conhecido por: *Register Spilling*
 - Uso de uma pilha;
 - Temos um apontador para o topo da pilha;
 - Este apontador é ajustado em uma palavra para cada registrador que é colocado na pilha (*push*), ou retirado da pilha (*pop*).
 - Na arquitetura RISC-V, o registrador $x2$ é utilizado para indicar o topo da pilha: sp (*stack pointer*)



Usando a Pilha

- ◆ Por razões históricas, a pilha “cresce” do maior endereço para o menor endereço:
- ◆ Para colocar um valor na pilha (*push*), devemos decrementar sp em uma palavra e mover o valor desejado para a posição de memória apontada por sp ;
- ◆ Para retirar um valor da pilha (*pop*), devemos ler este valor da posição de memória apontado por sp , e então incrementar sp em uma palavra.



Exemplo:

- ♦ Suponha que tenhamos o seguinte código:

```
int exemplo_folha (int g, int h, int i, int j)
{
    int f;
    f = (g+h) - (i+j);
    return f;
}
```

Vamos gerar o código correspondente em assembly RV32I.



Exemplo :

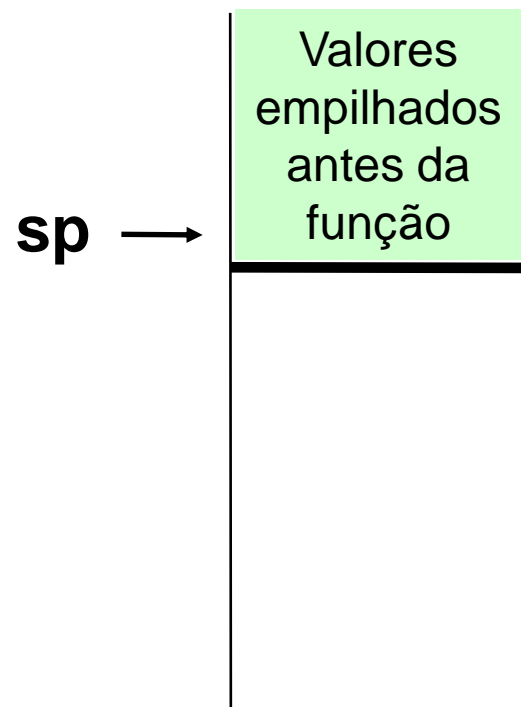
- ◆ Definição: Os argumentos g, h, i e j correspondem aos registradores $a0, a1, a2$ e $a3$, e f corresponde a $s0$.
- ◆ Definir o rótulo do procedimento:
exemplo_folha:
- ◆ Devemos então armazenar na pilha os registradores que serão utilizados pelo procedimento:

```
addi sp, sp, -12    # cria espaço para 3 words na pilha
sw  t1, 8(sp)       # empilha t1
sw  t0, 4(sp)       # empilha t0
sw  s0, 0(sp)       # empilha s0
```

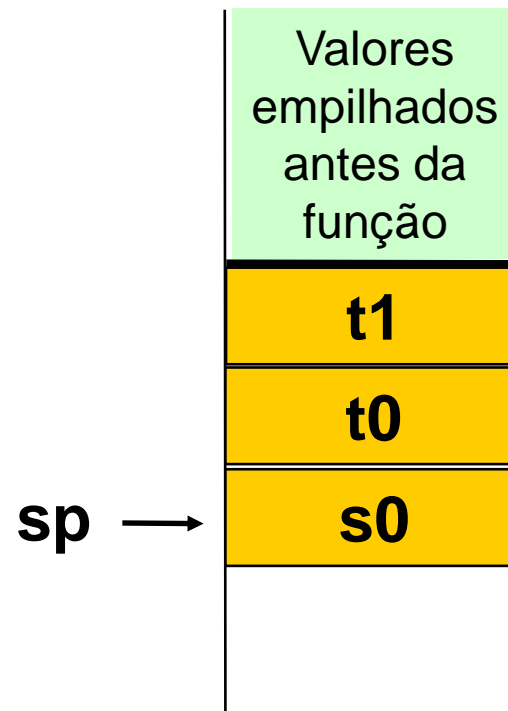


Exemplo : exemplo_folha

Como ficou a pilha?



**Pilha antes
da função**



**Pilha durante
execução da função**



Exemplo : exemplo_folha

- ◆ Corpo do procedimento:

```
add t0, a0, a1      # t0 = g + h
add t1, a2, a3      # t1 = i + j
sub s0, t0, t1      # f = s0 = (g+h) - (i+j)
```

- ◆ Resultado é colocado no registrador a0:

```
add a0, s0, zero    # retorna f em a0
```



Exemplo : exemplo_folha

- Antes de sair do procedimento, restaurar os valores dos registradores salvos na pilha:

```
lw s0, 0(sp)      # desempilha s0
lw t0, 4(sp)      # desempilha t0
lw t1, 8(sp)      # desempilha t1
addi sp, sp, 12    # remove 3 words da pilha
```

- Voltar o fluxo do programa para a instrução seguinte ao ponto em que a função exemplo_folha foi chamada:

```
jalr zero, ra, 0    # retorna
```



Versão Didática

exemplo_folha:

```
addi sp, sp, -12      # cria espaço para 3 words na pilha
sw t1, 8(sp)          # empilha t1
sw t0, 4(sp)          # empilha t2
sw s0, 0(sp)          # empilha s0
add t0, a0, a1         # t0 = g + h
add t1, a2, a3         # t1 = i + j
sub s0, t0, t1         # f = s0 = (g+h) - (i+j)
add a0, s0, zero       # retorna f em a0
lw s0, 0(sp)          # desempilha s0
lw t0, 4(sp)          # desempilha t0
lw t1, 8 (sp)         # desempilha t1
addi sp, sp, 12       # remove 3 words da pilha
jalr zero, ra, 0      # retorna
```

```
int exemplo_folha (int g, int j,
int i, int h)
{
    int f;
    f = (g+h) - (i+j);
    return f;
}
```



Versão otimizada

- Salvar somente o que realmente necessitar ser salvo
- Registradores t_x não precisam ser preservados.
- Utilizar registradores s_x quando realmente forem necessários.
- Ponderar uso de registradores com análise de desempenho.

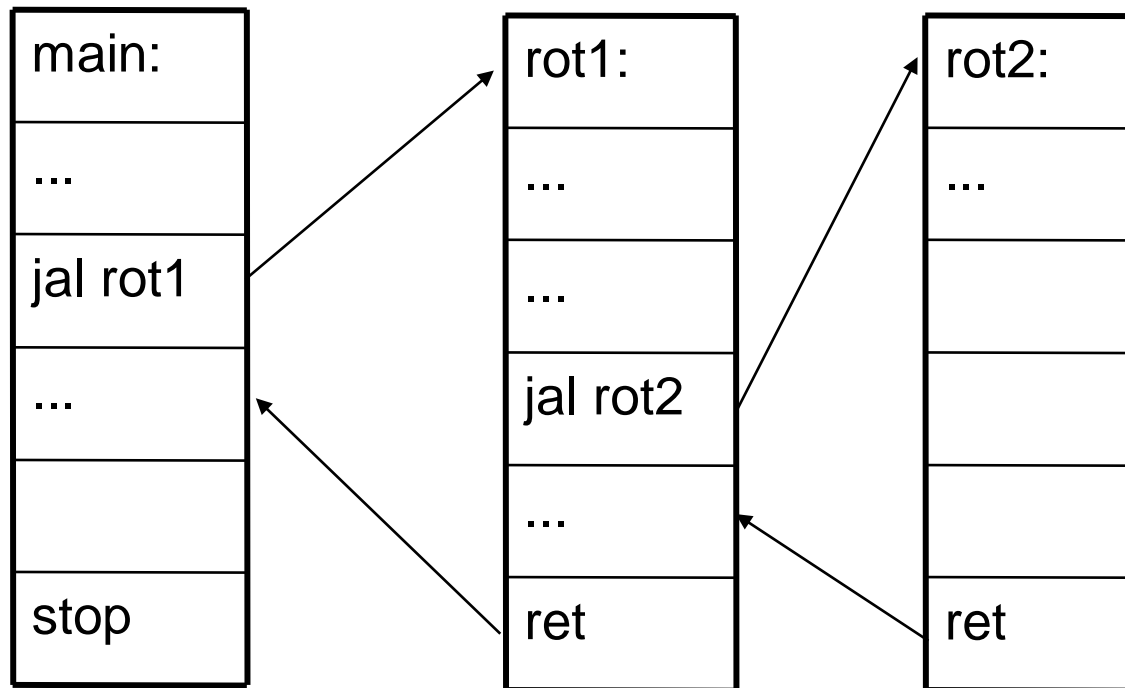
exemplo_folha:

```
add a0, a0, a1    # a0 = g + h
sub a0, a0, a2    # a0 = g+h-i
sub a0, a0, a3    # f = a0 = g+h-i-j
ret              # jalr zero, ra, 0 retorna
```

Procedimentos aninhados

- Suponha o seguinte procedimento aninhado:

MEMÓRIA



- Problema: conflito com registradores **ax** e **ra**!
- Como resolver?



Procedimentos aninhados: convenção sobre o uso dos registradores

- ♦ Uma solução é empilhar todos os registradores que precisam ser preservados.
- ♦ Estabelecer uma convenção entre sub-rotinas chamada e chamadora sobre a preservação dos registradores (uso eficiente da pilha).
- ♦ Definições
 - Chamadora: função que faz a chamada, utilizando jal/call;
 - Chamada: função sendo chamada.



Por que utilizar convenções para chamadas de procedimentos?

■ **Vantagens:**

- Programadores podem escrever funções que funcionam juntas;
- Funções que chamam outras funções – como as recursivas – funcionam corretamente.

■ **Desvantagem:**

- Muitas vezes precisaremos usar a memória (pilha) como local de armazenamento.