Universidade Tecnológica Federal do Paraná – UTFPR
Bacharelado em Ciência da Computação

BCC33B – Arquitetura e Organização de Computadores

Prof. Rogério A. Gonçalves

rogerioag@utfpr.edu.br

Aula 009

Aula de Hoje:

Conjunto de Instruções

- **Chamadas de procedimentos**

Suporte a Procedimentos

- **Para a execução de um procedimento deve-se:**
  - **Colocar os parâmetros em um local onde o procedimento possa acessá-los**
  - **Transferir o controle ao procedimento**
  - **Adquirir os recursos necessários ao procedimento**
  - **Executar a tarefa**
  - **Colocar o resultado em um local onde o programa possa acessá-lo**
  - **Retornar o controle ao ponto onde o procedimento foi chamado**

Suporte a Procedimentos

- **Para este mecanismo, o MIPS aloca seus registradores, para chamada de procedimentos, da seguinte maneira:**
  - **$a0 .. $ a3 ➔ 4 registradores para passagem de argumentos**
  - **$v0 .. $v1 ➔ para retornar valores**
  - **$ra ➔ para guardar o endereço de retorno**

- **Instrução para chamada de procedimento**
  - **`jal end_procedimento` - *(jump-and-link)***
    - desvia para o procedimento
    - salva o endereço de retorno (PC+4) em ***$ra*** (*return address* - $31)
- **Instrução para retorno de chamada de procedimento**
  - **`jr $ra`**
    - desvia para o ponto de onde foi chamado o procedimento
    - PC ← $ra

Chamada de Procedimento

Chamada de Procedimento - convenções:

- **Chamada:**
  - **Passa argumentos para o procedimento.**
- **Procedimento:**
  - **Não deve sobrescrever os registradores nem a memória usados por quem chama**
  - **Retorna ao ponto de chamada**
  - **Retorna o resultado para quem chama**

Convenções MIPS:

- **Chamada de procedimento: jump e link (`jal`)**
- **Retorno de procedimento: jump register (`jr`)**
- **Argumentos: $a0 - $a3**
- **Retorno do valor calculado: $v0**

Chamada de Procedimento

High-level code

```
int main() {
  simple();
  a = b + c;
}

void simple() {
  return;
}
```

MIPS assembly code

```
0x00400200 main: jal  simple
0x00400204       add  $s0, $s1, $s2
...

0x00401020 simple: jr $ra
```

Chamada de Procedimento

High-level code

```
int main() {
  simple();
  a = b + c;
}

void simple() {
  return;
}
```

MIPS assembly code

```
0x00400200 main: jal  simple
0x00400204       add  $s0, $s1, $s2
...

0x00401020 simple: jr $ra
```

`jal:` salta para `simple` e salva `PC+4` no registrador de endereço de retorno (`$ra`), neste caso, `$ra` = 0x00400204 após `jal` ser executado.

`jr $ra:` salta para o endereço em `$ra`, neste caso volta para o endereço de retorno: PC = 0x00400204.

Suporte a Procedimentos

- **Exemplo:**
  - **Os parâmetros g, h, i e j correspondem a $a0 .. $a3, respectivamente e f a $s0. Antes precisaremos salvar $s0, $t0 e $t1 na pilha, pois serão usados no procedimento**
  - **Seja o procedimento:**

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

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

Suporte a Procedimentos

exemplo-proc-1.asm

```
13
14  p_exe:  addi $sp,$sp,-12        # ajuste do sp para empilhar 3 palavras
15          sw  $t1,8($sp)          # salva $t1 na pilha para usar depois
16          sw  $t0,4($sp)          # salva $t0 na pilha para usar depois
17          sw  $s0,0($sp)          # salva $s0 na pilha para usar depois
18
19          # No procedimento
20
21          add $t0,$a0,$a1         # reg. $t0 contém g + h
22          add $t1,$a2,$a3         # reg. $t1 contémi i + j
23          sub $s0,$t0,$t1         # f = $t0 - $t1, que é (g + h) – ( i + j)
24
25          # Para retornar o valor f
26
27          add $v1,$s0,$zero       # retorna f ($v1 = $s0 + 0)
28
29          # Antes do retorno é necessário restaurar os valores dos registradores salvos na pilha
30
31          lw $s0, 0($sp)          # restaura reg. $s0 para o chamador
32          lw $t0, 4($sp)          # restaura reg. $t0 para o chamador
33          lw $t1, 8($sp)          # restaura reg. $t1 para o chamador
34          addi $sp,$sp,12         # ajusta pilha par excluir 3 itens
35
36          # Retornar
37
38          jr  $ra                 # desvia de volta à rotina que chamou
39
40  main:   nop                     # parametros para a função.
41
42          # colocar valores em t0, t1 e s0, imprimi-los antes e depois da chamada.
43
```

Line: 42 Column: 3 ☑ Show Line Numbers

Suporte a Procedimentos

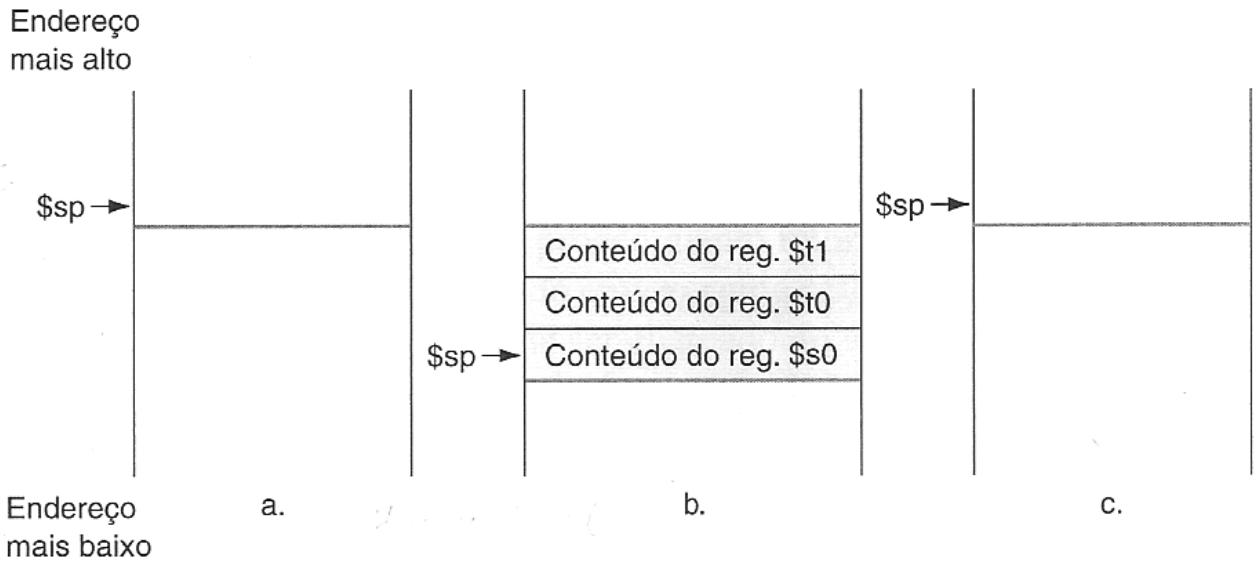


Figura 2.14 **Os valores do stack pointer e a pilha (a) antes, (b) durante e (c) após a chamada do procedimento.** O stack pointer sempre aponta para o "topo" da pilha, ou para a última word na pilha, neste desenho.

Exercício

High-level code

```
void main(){
    int y;
    y = sum(42, 7);
    ...
}

int sum(int a, int b){
    return (a + b);
}
```

Argumentos e Retorno de Valores

High-level code

```
int main() {
  int y;
  ...
  y = diffofsums(2, 3, 4, 5);  // 4 arguments
  ...
}

int diffofsums(int f, int g, int h, int i){
  int result;
  result = (f + g) - (h + i);
  return result;               // return value
}
```

Argumentos e Retorno de Valores

Código MIPS (assembly)

```
# $s0 = y
main:  ...
       addi $a0, $0, 2          # argument 0 = 2
       addi $a1, $0, 3          # argument 1 = 3
       addi $a2, $0, 4          # argument 2 = 4
       addi $a3, $0, 5          # argument 3 = 5
       jal  diffofsums          # call procedure
       add  $s0, $v0, $0        # y = returned value
       ...                     # $s0 = result
diffofsums: add $t0, $a0, $a1   # $t0 = f + g
        add $t1, $a2, $a3       # $t1 = h + i
        sub $s0, $t0, $t1       # result = (f + g) - (h + i)
        add $v0, $s0, $0        # put return value in $v0
        jr  $ra                 # return to caller
```

Argumentos e Retorno de Valores

Código MIPS (assembly)

```
# $s0 = result
diffofsums: add $t0, $a0, $a1    # $t0 = f + g
            add $t1, $a2, $a3    # $t1 = h + i
            sub $s0, $t0, $t1    # result =(f+g)-(h+i)
            add $v0, $s0, $0     # put return value in $v0
            jr  $ra              # return to caller
```

- ***diffofsums*** sobrescreve 3 registradores: $t0, $t1, e $s0
  - ***diffofsums*** pode usar a *pilha* para armazenar temporariamente os registradores

Chamada de Procedimentos Usando a Pilha

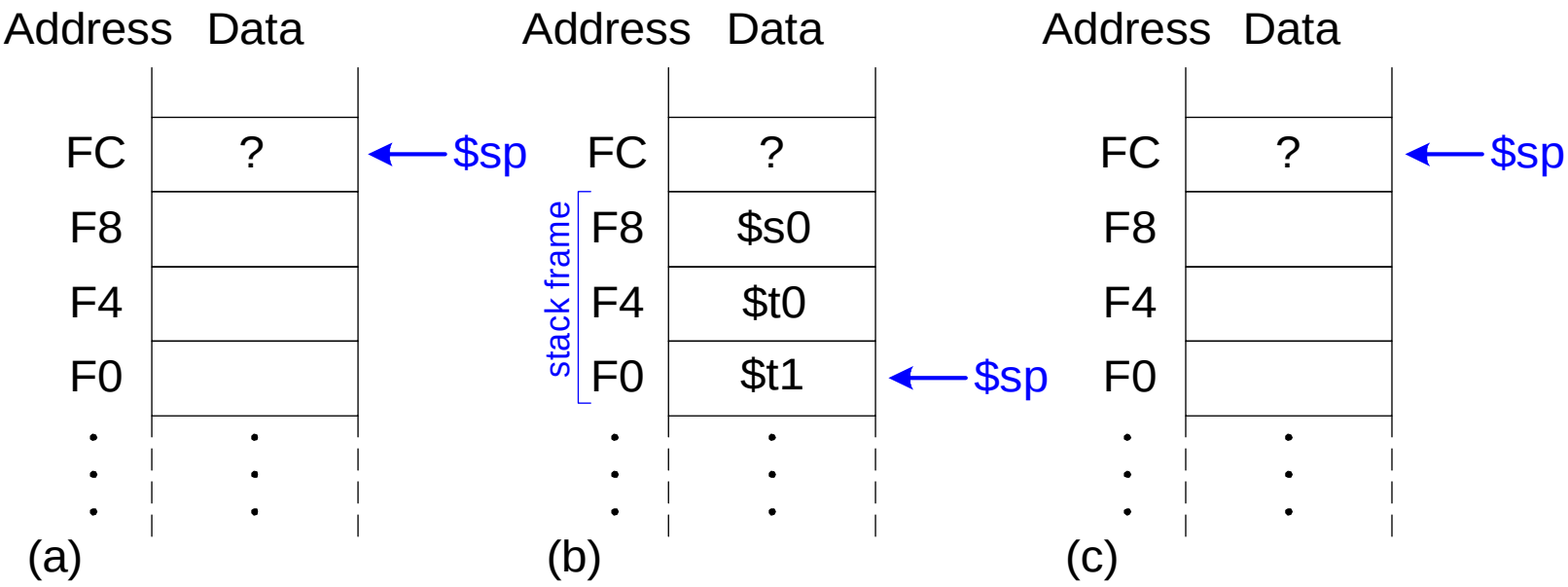
- O procedimento chamado não deve provocar nenhum efeito colateral.
- Mas `diffofsums` sobrescreve 3 registradores: `$t0`, `$t1`, `$s0`

```
# MIPS assembly
 # $s0 = result
 diffofsums:
       add $t0, $a0, $a1  # $t0 = f + g
       add $t1, $a2, $a3  # $t1 = h + i
       sub $s0, $t0, $t1  # result = (f + g) - (h + i)
       add $v0, $s0, $0   # put return value in $v0
       jr  $ra            # return to caller
```

Chamada de Procedimentos Usando a Pilha

```
# $s0 = result
diffofsums:    addi $sp, $sp, -12  # make space on stack
                                   # to store 3 registers
               sw   $s0, 8($sp)    # save $s0 on stack
               sw   $t0, 4($sp)    # save $t0 on stack
               sw   $t1, 0($sp)    # save $t1 on stack
               add  $t0, $a0, $a1  # $t0 = f + g
               add  $t1, $a2, $a3  # $t1 = h + i
               sub  $s0, $t0, $t1  # result = (f + g) - (h + i)
               add  $v0, $s0, $0   # put return value in $v0
               lw   $t1, 0($sp)    # restore $t1 from stack
               lw   $t0, 4($sp)    # restore $t0 from stack
               lw   $s0, 8($sp)    # restore $s0 from stack
               addi $sp, $sp, 12   # deallocate stack space
               jr   $ra            # return to caller
```

A Pilha durante a Chamada de `diffofsums`



Registradores

Preserved ***Callee-Saved***	**Nonpreserved** ***Caller-Saved***
`$s0 - $s7`	`$t0 - $t9`
`$ra`	`$a0 - $a3`
`$sp`	`$v0 - $v1`
stack above `$sp`	**stack below** `$sp`

Chamadas Múltiplas de Procedimentos

```
proc1:
  addi $sp, $sp, -4    # make space on stack
  sw   $ra, 0($sp)     # save $ra on stack
  jal  proc2
  ...
  lw   $ra, 0($sp)     # restore $r0 from stack
  addi $sp, $sp, 4    # deallocate stack space
  jr  $ra              # return to caller
```

Armazenando Registradores na Pilha

```
# $s0 = result
diffofsums:
  addi $sp, $sp, -4  # make space on stack to
                     # store one register
  sw  $s0, 0($sp)    # save $s0 on stack
  add $t0, $a0, $a1  # $t0 = f + g
  add $t1, $a2, $a3  # $t1 = h + i
  sub $s0, $t0, $t1  # result = (f + g) - (h + i)
  add $v0, $s0, $0   # put return value in $v0
  lw  $s0, 0($sp)    # restore $s0 from stack
  addi $sp, $sp, 4   # deallocate stack space
  jr  $ra            # return to caller
```

Suporte a Procedimentos

- **Observações**
  - **$t0 .. $t9: 10 registradores temporários que não são preservados em uma chamada de procedimento**
  - **$s0 .. $s7: 8 registradores que devem ser preservados em uma chamada de procedimento**

Exemplo – procedimento recursivo

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

Chamada Recursiva de Procedimentos

exemplo-proc-rec.asm

```
4
5           .text
6           j main                  # salta para o prog. principal.
7
8           # função fatorial.
9   fat:    addi $sp, $sp, -8       # cria espaço para empilhar.
10          sw   $a0, 4($sp)        # store $a0
11          sw   $ra, 0($sp)        # store $ra
12          addi $t0, $0, 2
13          slt  $t0, $a0, $t0      # a <= 1 ?
14          beq  $t0, $0, else      # não: go to else
15          addi $v1, $0, 1         # sim: retorna 1
16          addi $sp, $sp, 8        # restaura $sp
17          jr   $ra                # retorna
18  else:   addi $a0, $a0, -1       # n = n - 1
19          jal  fat                # chamada recursiva de fat.
20          lw   $ra, 0($sp)        # restaura $ra
21          lw   $a0, 4($sp)        # restaura $a0
22          addi $sp, $sp, 8        # restaura $sp
23          mul  $v1, $a0, $v1      # n * fat(n-1), multiplica o valor atual pelo retorno da função.
24          jr   $ra                # retorno.
25
26  main:   nop                     # parametros para a função.
27          addi $a0, $zero, 5      # arg1
28
29          jal fat
30                                  # impressão.
31          li $v0, 1               # carrega o valor da syscall para imprimir um inteiro (1).
32          add $a0, $v1, $zero     # valor a ser impresso deve estar em $a0.
33          syscall
34
```

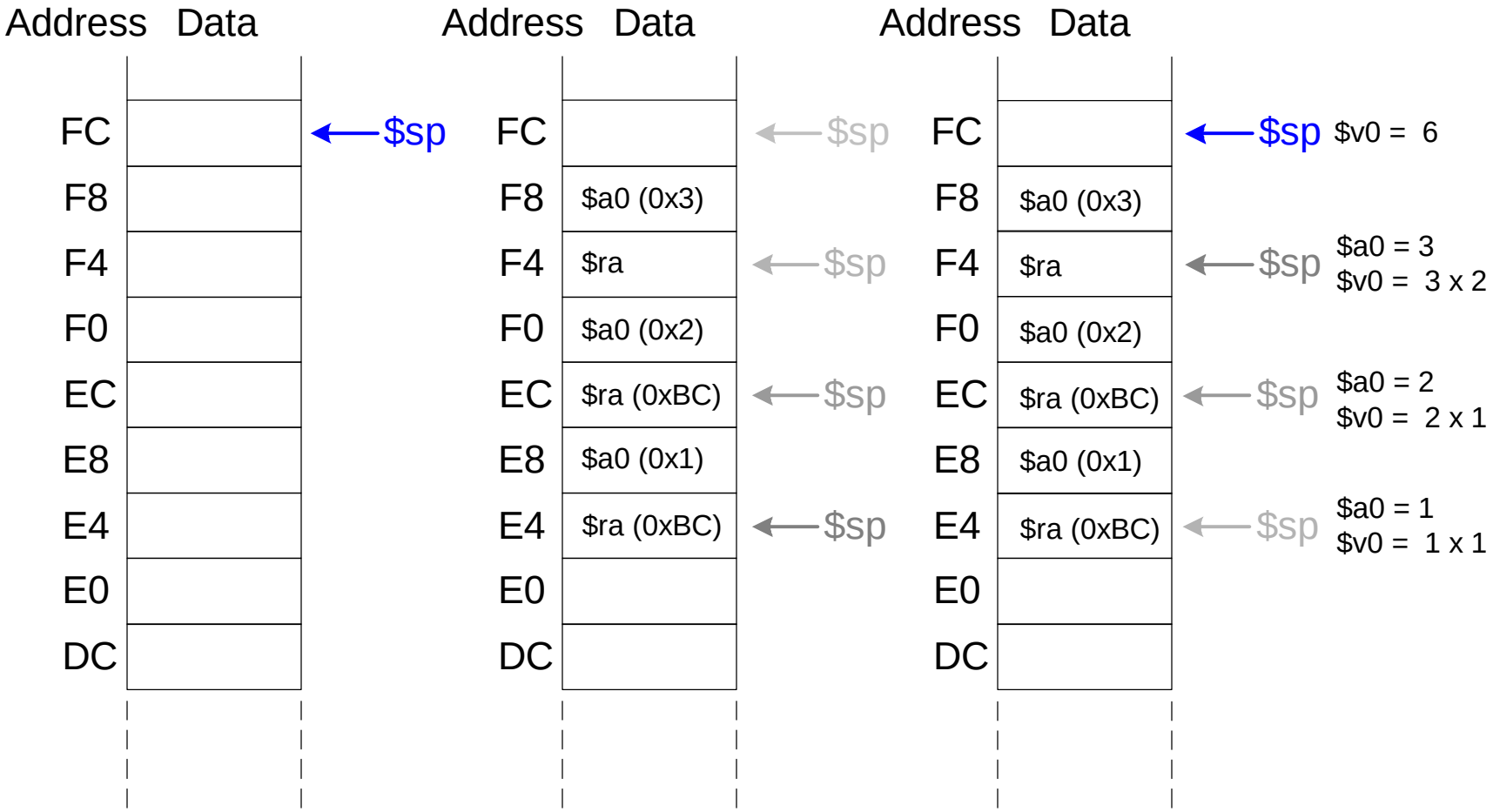
Line: 2 Column: 14 ☑ Show Line Numbers

Chamada Recursiva de Procedimentos

MIPS assembly code

```
0x90 factorial:  addi $sp, $sp, -8       # make room
0x94             sw   $a0, 4($sp)        # store $a0
0x98             sw   $ra, 0($sp)        # store $ra
0x9C             addi $t0, $0, 2
0xA0             slt  $t0, $a0, $t0      # a <= 1 ?
0xA4             beq  $t0, $0, else      # no: go to else
0xA8             addi $v0, $0, 1         # yes: return 1
0xAC             addi $sp, $sp, 8        # restore $sp
0xB0             jr   $ra                # return
0xB4      else:  addi $a0, $a0, -1       # n = n - 1
0xB8             jal  factorial          # recursive call
0xBC             lw   $ra, 0($sp)        # restore $ra
0xC0             lw   $a0, 4($sp)        # restore $a0
0xC4             addi $sp, $sp, 8        # restore $sp
0xC8             mul  $v0, $a0, $v0      # n * factorial(n-1)
0xCC             jr   $ra                # return
```

A Pilha Durante a Chamada Recursiva



Exemplo: Programa em C

```
int f, g, y;  // global variables

int main(void){
  f = 2;
  g = 3;
  y = sum(f, g);

  return y;
}

int sum(int a, int b) {
  return (a + b);
}
```

Exemplo: Programa em *Assembly*

```c
int f, g, y;  // global


int main(void)
{


  f = 2;
  g = 3;

  y = sum(f, g);
  return y;
}

int sum(int a, int b) {
  return (a + b);
}
```

```
        .data
f:    .word 0x00000000
g:    .word 0x00000000
y:    .word 0x00000000
        .text
main: addi $sp, $sp, -4   # stack frame
      sw   $ra, 0($sp)    # store $ra
      addi $a0, $0, 2     # $a0 = 2
      sw   $a0, f         # f = 2
      addi $a1, $0, 3     # $a1 = 3
      sw   $a1, g         # g = 3
      jal  sum            # call sum
      sw   $v0, y         # y = sum()
      sw   $ra, 0($sp)    # restore $ra
      addi $sp, $sp, 4    # restore $sp
      jr   $ra            # return to OS
sum:  add  $v0, $a0, $a1  # $v0 = a + b
      jr   $ra            # return
```

Leitura Recomendada

Capítulo 2 e Anexo B.6

PATTERSON, David A.; HENNESSY, John L. ***Organização e projeto de computadores: a interface hardware/software.*** Rio de Janeiro, RJ: Elsevier, 2005. 484 p. ISBN 9788535215212.

Resumo da Aula de Hoje

<u>Tópicos mais importantes:</u>

Linguagem *Assembly*

Microprocessador MIPS

Chamada a procedimentos