Universidade de Brasília
Departamento de Ciência da Computação

Aula 7
Assembly MIPS
Recursividade e I/O

Exemplo: soma_recursiva

- Suponha que tenhamos o seguinte código, que calcula a soma *n + (n-1) + … + 2 + 1* de forma recursiva:

```
int soma_recursiva (int n)
{
if (n < 1)
        return 0;
else
        return n + soma_recursiva(n-1)
}
```

- Vamos gerar o código correspondente em assembly MIPS.

Exemplo: soma_recursiva

- O parâmetro *n* corresponde ao registrador *$a0*.
- Devemos inicialmente colocar um rótulo para a função, e salvar o endereço de retorno *$ra* e o parâmetro $a0:

```
Soma_recursiva:
addi $sp, $sp, -8       # prepara a pilha para receber 2 itens
sw $ra, 4($sp)          # empilha $ra  (End. Retorno)
sw $a0, 0($sp)          # empilha $a0 (n)
```

- Na primeira vez que soma_recursiva é chamada, o valor de $ra que é armazenado corresponde ao endereço que está na rotina chamadora.

Exemplo: soma_recursiva

- Vamos agora compilar o corpo da função. Inicialmente, testamos se *n < 1*:

```
slti $t0, $a0, 1        # testa se n < 1
beq $t0, $zero, L1      # se n>=1, vá para L1
```

- Se *n <1*, a função deve retornar o valor 0. Não podemos nos esquecer de restaurar a pilha.

```
add $v0, $zero, $zero   # valor de retorno é 0
addi $sp, $sp, 8        # remove 2 itens da pilha
jr $ra                  # retorne para depois de jal
```

- ***Por que não carregamos os valores de $a0 e $ra antes de ajustar $sp??***

Exemplo: soma_recursiva

- Se *n* >=*1*, decrementamos *n* e chamamos novamente a função soma_recursiva com o novo valor de *n*.

```
L1:     addi $a0, $a0, -1       # argumento passa a ser (n-1)
        jal soma_recursiva      # calcula a soma para (n-1)
```

- Quando a soma para *(n-1)* é calculada, o programa volta a executar na próxima instrução. Restauramos o endereço de retorno e o argumento anteriores, e incrementamos o apontador de topo de pilha:

```
lw $a0, 0($sp)          # restaura o valor de n
lw $ra, 4($sp)          # restaura o endereço de retorno
addi $sp, $sp, 8        # retira 2 itens da pilha.
```

Exemplo: soma_recursiva

- Agora o registrador $v0 recebe a soma do argumento antigo $a0 com o valor atual em $v0 (soma_recursiva para *n-1*):

```
add $v0, $a0, $v0      # retorne n + soma_recursiva(n-1)
```

- Por último, voltamos para a instrução seguinte à que chamou o procedimento:

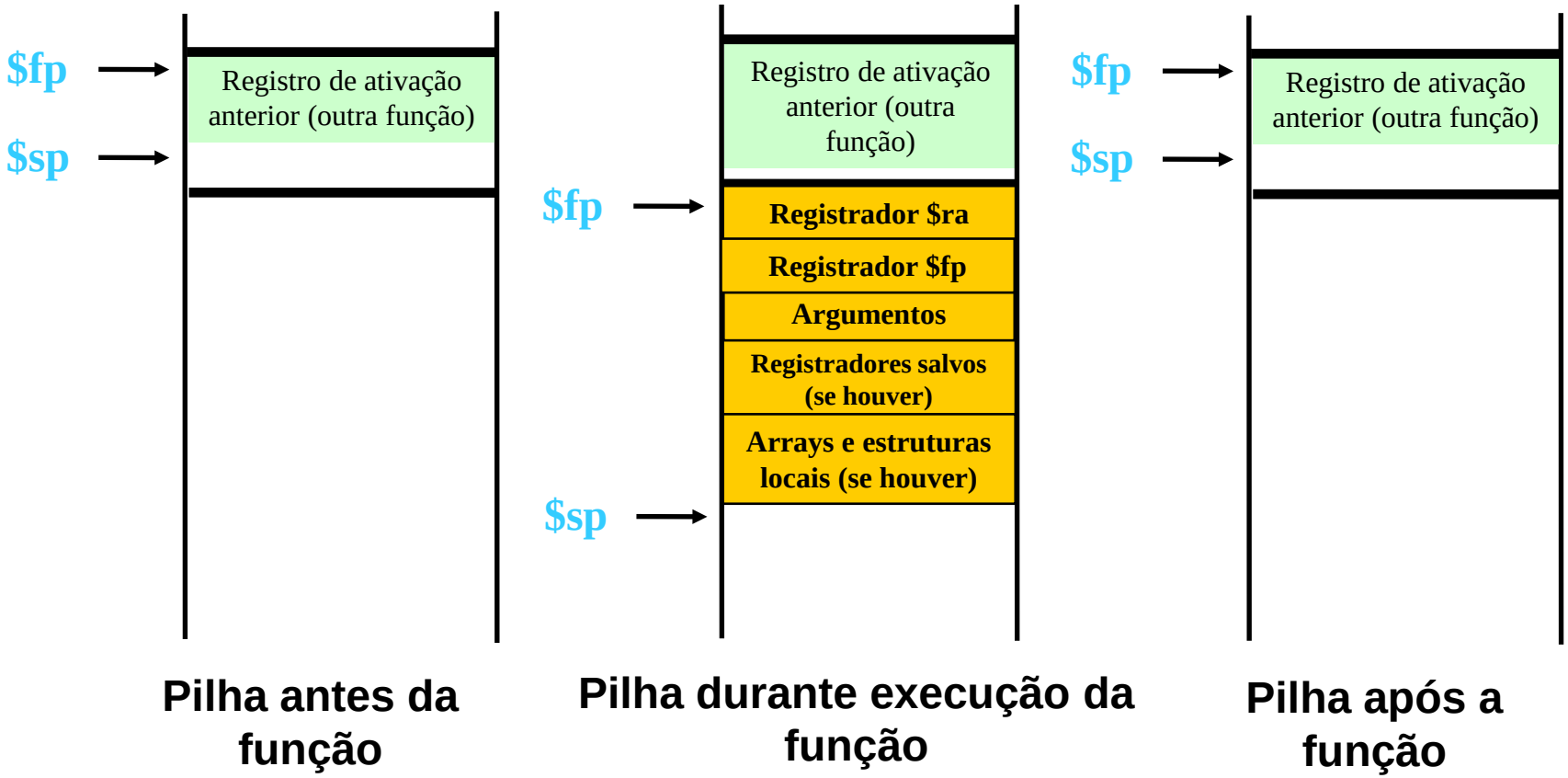
```
jr $ra                 # retorne para a chamadora
```

Listagem

```
Soma_recursiva:
    addi $sp, $sp, -8          # prepara a pilha para receber 2 itens
    sw $ra, 4($sp)             # empilha $ra  (End. Retorno)
    sw $a0, 0($sp)             # empilha $a0 (n)
    slti $t0, $a0, 1           # testa se n < 1
    beq $t0, $zero, L1         # se n>=1, vá para L1
    add $v0, $zero, $zero      # valor de retorno é 0
    addi $sp, $sp, 8           # remove 2 itens da pilha
    jr $ra                     # retorne para depois de jal
L1:
    addi $a0, $a0, -1          # argumento passa a ser (n-1)
    jal soma_recursiva         # calcula a soma para (n-1)
    lw $a0, 0($sp)             # restaura o valor de n
    lw $ra, 4($sp)             # restaura o endereço de retorno
    addi $sp, $sp, 8           # retira 2 itens da pilha.
    add $v0, $a0, $v0          # retorna n + soma_recursiva(n-1)
    jr $ra                     # retorna para a chamadora
```

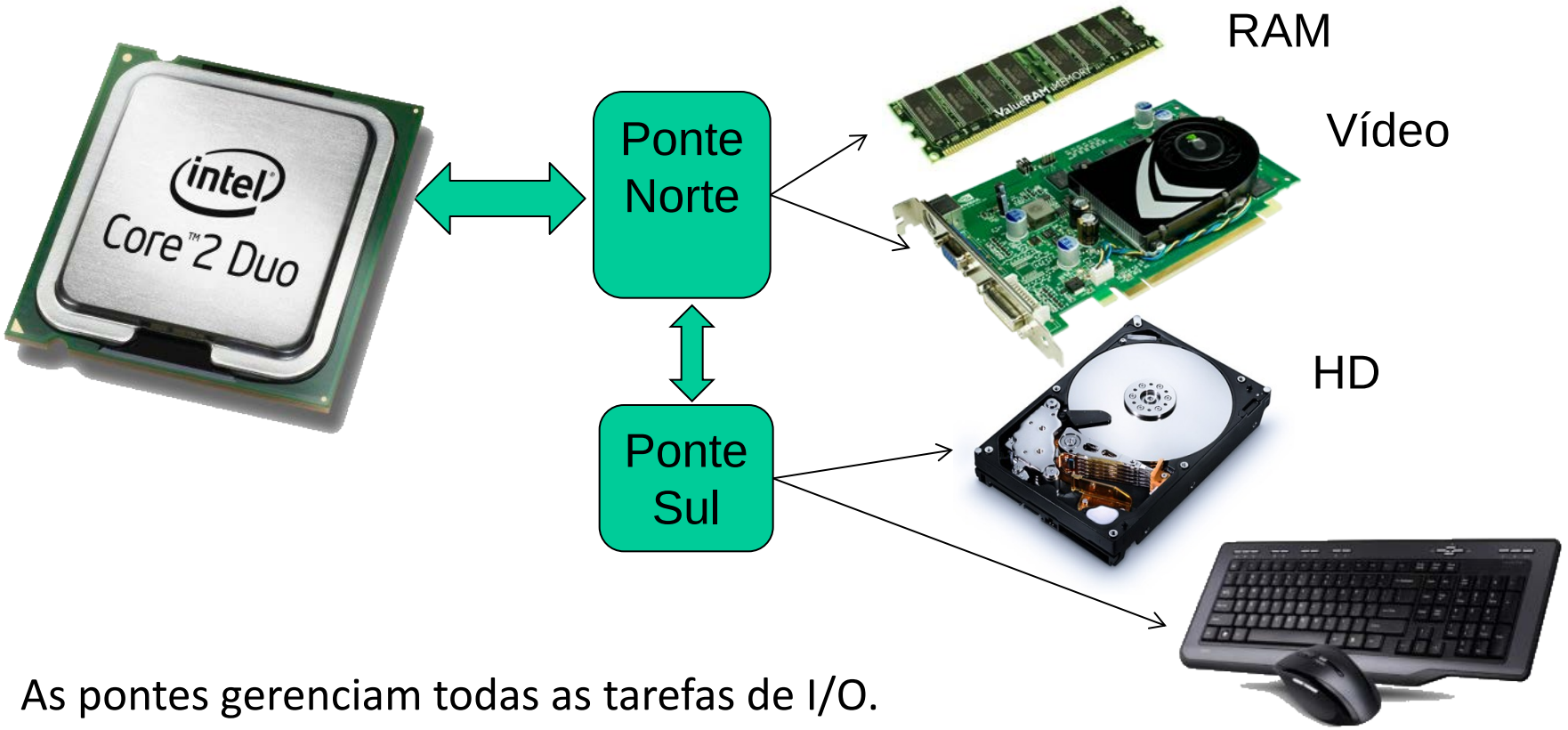
Alocando espaço para novos dados locais na pilha

- Frame de Procedimento (Registro de Ativação)
  - Armazenar variáveis locais a um procedimento
  - Facilita o acesso a essas variáveis locais ter um apontador estável $fp



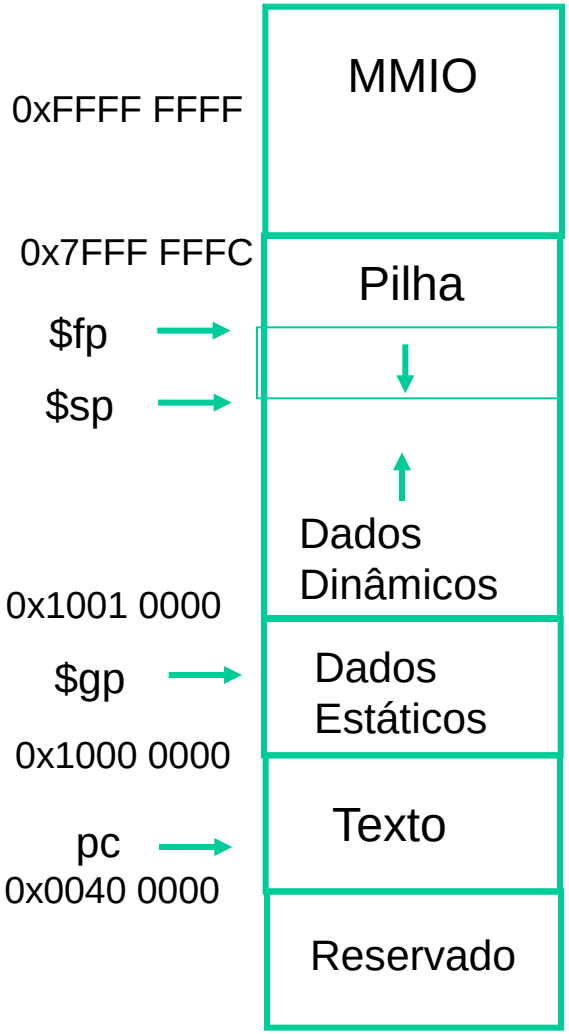
Conexão com dispositivos de I/O

Na arquitetura x86 o processador se comunica com os dispositivos externos rápidos (incluindo a memória (antes do i7)) através da North Bridge e os dispositivos lentos são ligados à South Bridge.



As pontes gerenciam todas as tarefas de I/O.
i3,i5,i7: Controladores da Memória e GPU integrados ao Processador
Em outros sistemas mais simples é comum o uso de 1(ou mais) barramentos

Mapeamento da Memória e Conexão com dispositivos de I/O



Memory Mapped I/O

Todos os dispositivos de I/O utilizam um endereço de memória (são mapeados).

Espaço para procedimentos armazenarem informações

Frame de Procedimento

Heap: Espaço explicitamente criado
malloc: apontadores em C

Variáveis estáticas declaradas uma vez para todo programa

Programa Código de Máquina

Rotinas do sistema

Instrução Syscall

É na realidade uma forma elegante de gerar uma interrupção! (estudado adiante)

Realiza a chamada à rotina de tratamento de interrupção do sistema. Coloca o processador em modo Kernel.

PC=endereço de tratamento de exceção
EPC=PC+4

- Recebe parâmetros nos registradores específicos
- Retorna valores nos registradores específicos

Simulador MARS – chamadas do sistema

Implementa um montador com várias pseudo-instruções e
Simula um sistema operacional com funções mínimas de Entrada/Saída em console próprio.

```
1   .data
2   STR: .asciiz "Numero = " # Define uma string na memória de dados
3
4   .text
5           li $v0,5        # read int
6           syscall         # resultado em $v0
7           move $t0,$v0
8
9           li $v0,4        # print string
10          la $a0,STR
11          syscall
12
13          li $v0,1        # print int
14          move $a0,$t0
15          syscall
16
17          li $v0,10       # exit
18          syscall
```

Simulador MARS – chamadas do sistema

Serviço	Código de chamada do sistema	Argumentos	Resultado
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (em $v0)
read_float	6		float (em $f0)
read_double	7		double (em $f0)
read_string	8	$a0 = buffer, $a1 = tamanho	
sbrk	9	$a0 = valor	endereço (em $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (em $a0)
open	13	$a0 = nome de arquivo (string), $a1 = flags, $a2 = modo	descritor de arquivo (em $a0)
read	14	$a0 = descritor de arquivo, $a1 = buffer, $a2 = tamanho	número de caracteres lidos (em $a0)
write	15	$a0 = descritor de arquivo, $a1 = buffer, $a2 = tamanho	número de caracteres escritos (em $a0)
close	16	$a0 = descritor de arquivo	
exit2	17	$a0 = resultado	

FIGURA A.9.1 Serviços do sistema.

Exemplo 1 : Clear (ponteiro x array)

Objetivo: Zerar os componentes do array de tamanho size

```
void clear1(int array[], int size)
{
   int i;
   for(i=0;i<size;i++)
       array[i]=0;
}
```

```
clear1:    move $t0,$zero
Loop1:   sll $t1,$t0,2
       add $t2,$a0,$t1
       sw $zero,0($t2)
       addi $t0,$t0,1
       slt $t3,$t0,$a1
       bne $t3,$zero,Loop1
       jr $ra
```

```
void clear2(int *array, int size)
{
   int *p;
   for(p=&array[0];p<&array[size];p++)
       *p=0;
}
```

```
clear2:   move $t0,$a0
       sll $t1,$a1,2
       add $t2,$a0,$t1
Loop2:   sw $zero,0($t0)
       addi $t0,$t0,4
       slt $t3,$t0,$t2
       bne $t3,$zero,Loop2
       jr $ra
```

Qual o mais eficiente?

Exemplo 2 : Soma

Objetivo: Ler do teclado um valor positivo n
Calcular recursivamente o valor da soma 1+2+3+4+...+(n-1)+n
Escrever na tela o valor da soma

```
void main (void)
{
      int n, s;
      printf("Digite n:");
      scanf("%d",&n);
      s=soma(n);
      printf(“Soma(%d)=%d\n",n,s);
}
```

```
int soma(int n)
{
      if(n<1)
            return 0;
      else
            return n+soma(n-1);
}
```

Qual o tempo de execução do seu procedimento ‘soma’ para n=200 caso seja executado em um processador MIPS com frequência de 1GHz e CPI=1 ?

Exemplo : SORT

- Compile para Assembly MIPS o seguinte programa C

```
#include <stdio.h>

void show(int v[], int n)
{
   int i;
   for(i=0;i<n;i++)
        printf("%d\t",v[i]);
   printf("\n");
}

void swap(int v[], int k)
{
   int temp;
   temp=v[k];
   v[k]=v[k+1];
   v[k+1]=temp;
}

void sort(int v[], int n)
{
  int i,j;
   for(i=0;i<n;i++)
      for(j=i-1;j>=0 && v[j]>v[j+1];j--)
         swap(v,j);
}
```

```
int vetor[10]={9,2,5,1,8,2,4,3,6,7};

void main()
{
   show(vetor,10);
   sort(vetor,10);
   show(vetor,10);
}
```

```
        .data
vetor:      .word 9,2,5,1,8,2,4,3,6,7
newl: .asciiz "\n"
tab:   .asciiz "\t"

        .text
        ....
        li $v0,10
        syscall
show:       ....
swap:       ....
sort:  ....
```

```
        .data
vetor:  .word 9,2,5,1,8,2,4,3,6,7
newl:   .asciiz "\n"
tab:    .asciiz "\t"

        .text

        la $a0,vetor
        li $a1,10
        jal show

        la $a0,vetor
        li $a1,10
        jal sort

        la $a0,vetor
        li $a1,10
        jal show

        li $v0,10
        syscall
```

```
swap:   sll $t1,$a1,2
        add $t1,$a0,$t1
        lw $t0,0($t1)
        lw $t2,4($t1)
        sw $t2,0($t1)
        sw $t0,4($t1)
        jr $ra


show:   move $t0,$a0
        move $t1,$a1
        move $t2,$zero
loop1:  beq $t2,$t1,fim1
        li $v0,1
        lw $a0,0($t0)
        syscall
        li $v0,4
        la $a0,tab
        syscall
        addi $t0,$t0,4
        addi $t2,$t2,1
        j loop1
fim1:   li $v0,4
        la $a0,newl
        syscall
        jr $ra
```

```
sort:   addi $sp,$sp,-20
        sw $ra,16($sp)
        sw $s3,12($sp)
        sw $s2,8($sp)
        sw $s1,4($sp)
        sw $s0,0($sp)
        move $s2,$a0
        move $s3,$a1
        move $s0,$zero
for1:   slt $t0,$s0,$s3
        beq $t0,$zero,exit1
        addi $s1,$s0,-1
for2:   slti $t0,$s1,0
        bne $t0,$zero,exit2
        sll $t1,$s1,2
        add $t2,$s2,$t1
        lw $t3,0($t2)
        lw $t4,4($t2)
        slt $t0,$t4,$t3
        beq $t0,$zero,exit2
        move $a0,$s2
        move $a1,$s1
        jal swap
        addi $s1,$s1,-1
        j for2
exit2:  addi $s0,$s0,1
        j for1
exit1:  lw $s0,0($sp)
        lw $s1,4($sp)
        lw $s2,8($sp)
        lw $s3,12($sp)
        lw $ra,16($sp)
        addi $sp,$sp,20
        jr $ra
```