



Aula 7

Assembly RISC-V

Recursividade e I/O





Exemplo: soma_recursiva

- ◆ Dado o seguinte código, que calcula a soma $soma(n) = n + (n-1) + \dots + 2 + 1$ 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.



Exemplo: soma_recursiva

- ◆ O parâmetro n é associado 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 words  
sw ra, 4(sp)         # empilha ra (End. Retorno)  
sw a0, 0(sp)         # empilha a0 (argumento 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.
Devemos lembrar de restaurar a pilha!

```
add a0, zero, zero      # valor de retorno é 0
addi sp, sp, 8           # remove 2 itens da pilha
ret                     # retorne
```

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



Exemplo: soma_recursiva

- Se $n \geq 1$, 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 (n-1)
    mv t0, a0           # salva a soma (n-1) em t0
```

- Quando a soma de $(n-1)$ 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 do argumento n
lw ra, 4(sp)            # restaura o endereço de retorno
addi sp, sp, 8          # retira 2 words da pilha.
```



Exemplo: soma_recursiva

- ♦ O registrador *a0* recebe a soma do argumento antigo *a0* (n) com o valor atual da soma(n-1) salvo em *t0*:

```
add a0, a0, t0    # calcula n + soma_recursiva(n-1)
```

- ♦ Retorna para a instrução seguinte à que chamou o procedimento:

```
ret              # retorna
```



Exemplo: soma_recursiva

soma_recursiva:

```

addi sp, sp, -8      # prepara a pilha para receber 2 words
sw ra, 4(sp)         # empilha ra (End. Retorno)
sw a0, 0(sp)         # empilha a0 (argumento n)
slti t0, a0, 1       # testa se n < 1
beq t0, zero, L1     # se n >= 1, vá para L1
add a0, zero, zero   # valor de retorno é 0
addi sp, sp, 8       # remove 2 words da pilha
ret                 # retorna para a chamadora

```

L1:

```

addi a0, a0, -1      # argumento passa a ser (n-1)
jal soma_recursiva   # calcula a soma_recursiva(n-1)
mv t0, a0            # salva a soma em t0
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 a0, a0, t0       # calcula n + soma_recursiva(n-1)
ret                 # retorna para a chamadora

```

Memória

2^N-1

...

48

44

40

36

32

28

24

20

16

12

8

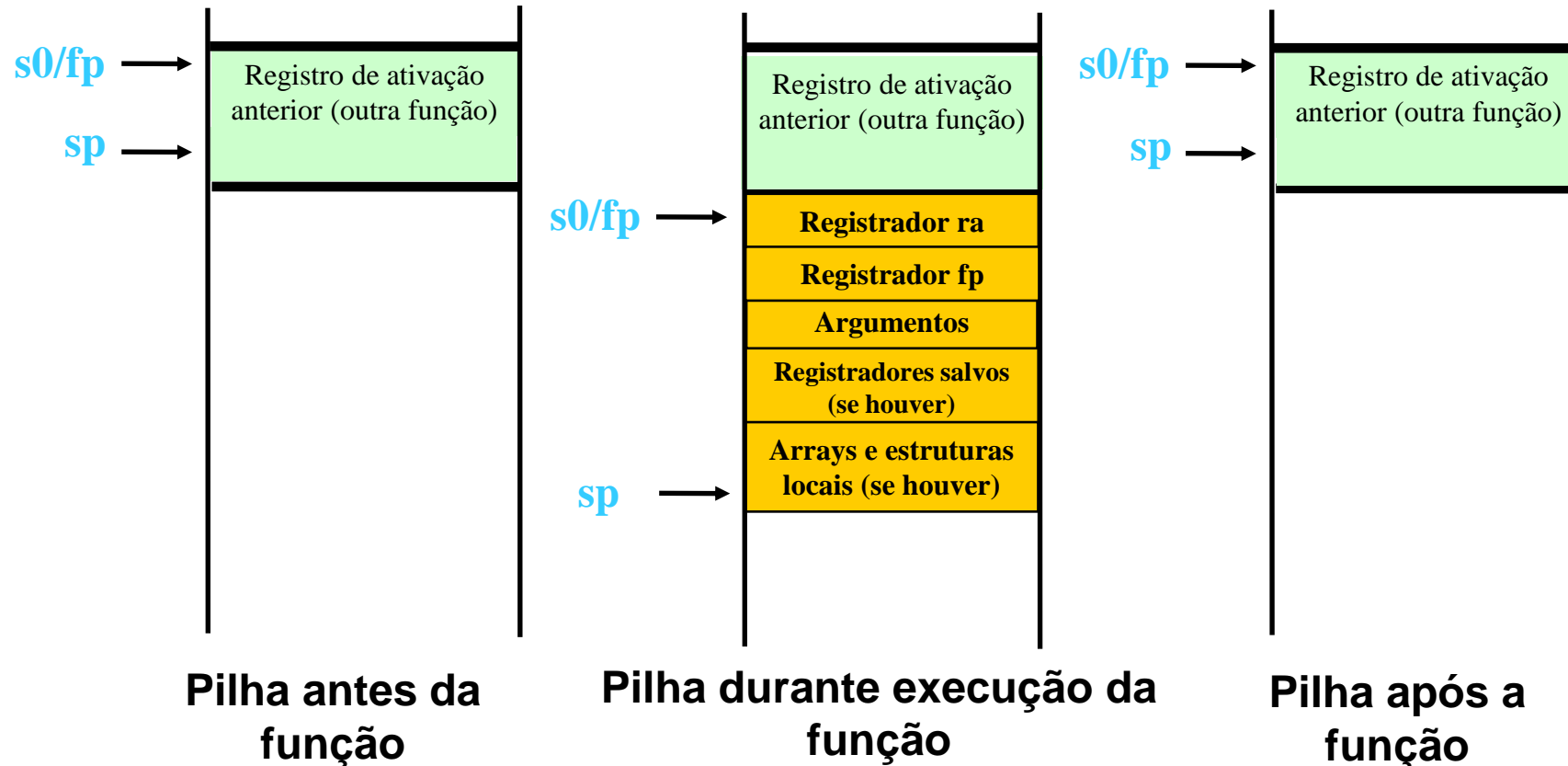
4

0



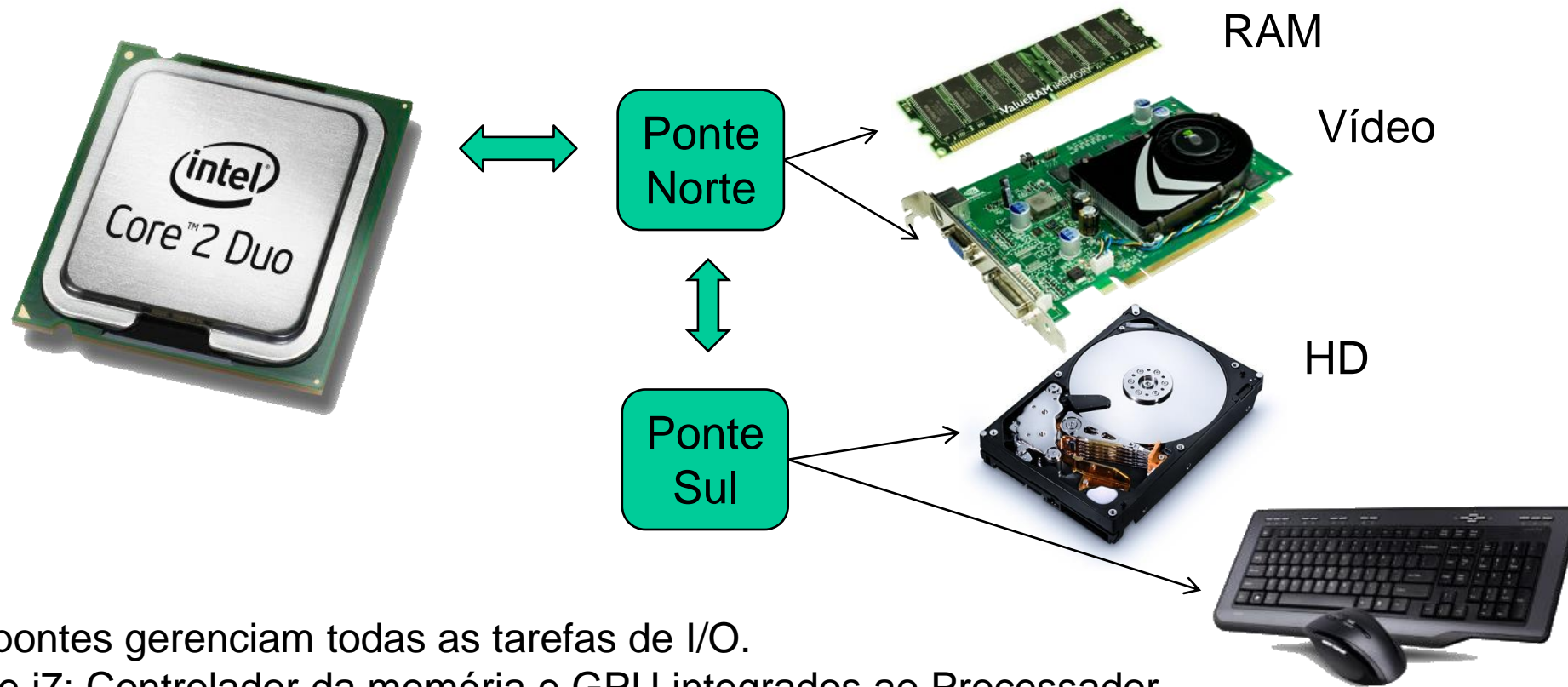
Alocando espaço para novos dados locais na pilha

- Frame de Procedimento (Registro de Ativação)
 - Armazenar variáveis locais a um procedimento
 - Facilita o acesso a essas variáveis locais ter um ponteiro estável ($s0/fp$)



Conexão com dispositivos de I/O

Na arquitetura x86 o processador se comunica com os dispositivos externos rápidos (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.

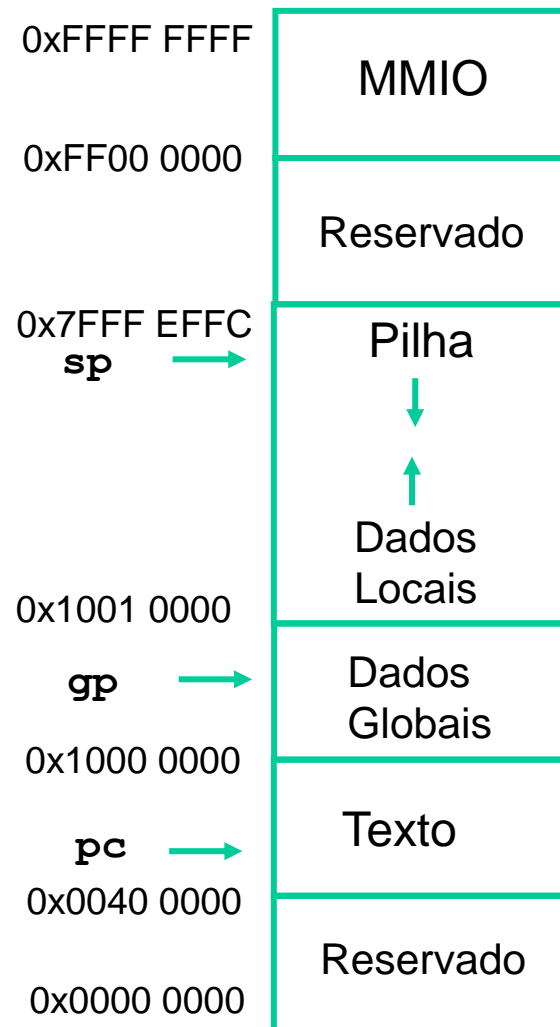
Core i7: Controlador da memória e GPU integrados ao Processador

No entanto, em sistemas RISCs é comum o uso de apenas 1 barramento (endereçamento).



Mapeamento da Memória e dispositivos no Rars

Memory Mapped I/O (MMIO)



Os dispositivos de I/O utilizam endereços de memória.

Área de dados: pilha e heap
Segmento: .data

Variáveis estáticas declaradas para todos procedimentos:
Segmento: .extern

Área de Programa
Segmento: .text



Estrutura básica de um programa em Assembly

```
Edit Execute
riscv5.asm
1  .data
2  D_WORD:  .word 0xF0CAFOFA
3  D_HALF:   .half 237, 0xBOB0
4  D_BYTE:   .byte 1, 2, 3, 0xFE
5  D_FLOAT:  .float 3.1415927
6  D_CHAR:   .ascii "A"
7  D_STRING: .string "Teste de string"
8
9  .text
10 INICIO: la t0, D_WORD
11         lw t1, 0(t0)
12         lhu t2, 4(t0)
13         lh t3, 6(t0)
14         lbu t4, 8(t0)
15         lb t5, 9(t0)
16         lb t6, 10(t0)
17         la t0, D_FLOAT
18         flw ft0, 0(t0)
19         lb a0, 4(t0)
20         la t0, INICIO
21         lw a1, 0(t0)
22
23
```

Line: 1 Column: 1 ☒ Show Line Numbers

Segmento de dados

Segmento de texto



Instrução `ecall` - Environment Call

Realiza chamadas aos serviços fornecidos pelo sistema operacional (se houver).

Ex.:

```
li a7,1          # serviço de print int
li a0,137        # dado a ser impresso na tela
ecall            # chamada ao sistema

li a7,10         # serviço exit
ecall            # chamada ao sistema

# Retorna ao Sistema Operacional
```



RARS – RISC-V Assembler and Runtime Simulator

Implementa um emulador da ISA RV32IMFD, montador com várias pseudo-instruções e um sistema operacional com funções mínimas de Entrada/Saída em console próprio.

Table of Available Services

Serviço selecionado pelo registrador a7

Name	Number	Description	Inputs	Ouputs
PrintInt	1	Prints an integer	a0 = integer to print	N/A
PrintFloat	2	Prints an floating point number	fa0 = float to print	N/A
PrintString	4	Prints a null-terminated string to the console	a0 = the address of the string	N/A
ReadInt	5	Read an int from input console	N/A	a0 = the int
ReadFloat	6	Read a float from input console	N/A	fa0 = the float
ReadString	8	N/A	N/A	N/A
Sbrk	9	Allocate heap memory	a0 = amount of memory in bytes	a0 = address to the allocated block
Exit	10	Exits the program with code 0	N/A	N/A
PrintChar	11	Prints an ascii character	a0 = character to print (only lowest byte is considered)	N/A
ReadChar	12	Read a character from input console	N/A	a0 = the character



Exemplo 1 : Clear (array × ponteiro)

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:mv t0, zero  
Loop1: slli t1, t0, 2  
        add t2, a0, t1  
        sw zero, 0(t2)  
        addi t0, t0, 1  
        blt t0, a1, Loop1  
        ret
```

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

```
clear2:mv t0, a0  
        slli t1, a1, 2  
        add t2, a0, t1  
Loop2: sw zero, 0(t0)  
        addi t0, t0, 4  
        bltu t0, t2, Loop2  
        ret
```

Qual o mais eficiente?



Exemplo 2 : Soma

Objetivo: Ler do teclado um valor inteiro positivo n
Calcular recursivamente o valor da soma $1+2+3+4+\dots+(n-1)+n$
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 RISC-V com frequência de 1GHz e CPI=1 ?
- Qual a quantidade de memória de dados (em bytes) serão necessários na pilha?



Exemplo : SORT

- Compile para Assembly RISC-V 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);
}
```

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

    show(vetor,10);
    sort(vetor,10);
    show(vetor,10);
}
```

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

.text
main:        ....
             li a7,10
             ecall

show:        ....
swap:        ....
sort:        ....
```