

[MAC0211] Laboratório de Programação I

Aula 7

Linguagem de Montagem (Depuradores, Recursão e Segmentação da Memória)

Kelly Rosa Braghetto

DCC-IME-USP

21 de março de 2013

Acessando os argumentos passados via linha de comando [32 bits] – argc e argv são passados na pilha

```
.global main
.text
main:
    movl    4(%esp),%ecx        # ecx <- argc
    movl    8(%esp),%ebx        # ebx <- argv

mostra:
    pushl   %ecx                # salva os registradores
    pushl   %ebx
    pushl   (%ebx)              # a string argumento a ser mostrada
    call    puts                # imprime-a
    addl    $4,%esp             # libera o espaço dos args na pilha
    popl    %ebx                # recupera os registradores
    popl    %ecx

    addl    $4, %ebx            # aponta para o proximo argumento
    decl    %ecx                # atualiza a qtde de argumentos a mostrar
    jnz     mostra             # se ainda houver argumentos, continua

    ret
```

Acessando os argumentos passados via linha de comando [64 bits] – argc em RDI e argv em RSI

```
.global main
.text
main:
    push    %rdi           # salva os registradores que
    push    %rsi           # o puts estraga
    mov     (%rsi), %rdi   # o argumento a ser mostrado
    call    puts          # imprime-o
    pop     %rsi           # restaura os registradores
    pop     %rdi

    add     $8, %rsi       # aponta para o prox argumento
    dec     %rdi           # atualiza a qtde de argumentos
    jnz    main           # se ainda ha argumentos, continua
    ret
```

Depuração de programas em linguagem de montagem

Passo 1:

- ▶ Inserir a opção `-g` na linha de comando do GCC, do NASM ou do GAS, para instruir o programa a incluir informações de depuração no executável final

Passo 2:

- ▶ Rastrear a execução do programa com o *Data Display Debugger* (ddd) ou o *GNU Project debugger* (gdb) do Linux.

Exemplo

```
$ gcc -g -o prog prog.c soma.s  
$ ddd prog
```

Depuração de programas em linguagem de montagem

DDD

Exemplo de uso: `$ ddd prog`

- ▶ Para visualizar os registradores, ative a opção por meio do menu `Status > Registers`

GDB

Exemplo de uso: `$ gdb prog`

Comandos básicos:

- ▶ `l` – lista o código
- ▶ `break N` – coloca um breakpoint na linha N
- ▶ `run` – executa o programa
- ▶ `s` – executa passo a passo
- ▶ `info registers` – mostra os registradores (informações mostradas: nome, valor em hexadecimal e valor em decimal)

Recursão

Fatorial – algoritmo

```
função Fatorial(N)
  se N = 0 então
    devolva 1; // base da recursão!
  senão
    devolva N * Fatorial(N-1);
  fim-se
fim
```

Recursão

Fatorial – em linguagem de montagem

```
fatorial:
    push    ebp                ; define o stack frame
    mov     ebp, esp

    mov     eax, [ebp+8]      ; obtem n
    cmp     eax, 0            ; n < 0?
    ja     continua          ; sim: continua
    mov     eax, 1            ; nao: retorna 1
    jmp     fim

continua:                      ; calcula o fatorial de n-1
    dec    eax
    push   eax                ; fatorial(n-1)
    call  fatorial
    add   esp,4                ; libera espaco do parametro

    mov    ebx, [ebp+8]      ; obtem n
    mul   ebx                ; edx:eax = eax * ebx

fim: pop    ebp
    ret
```

Recursão: Problema da Torre de Hanoi – 1 disco(s)

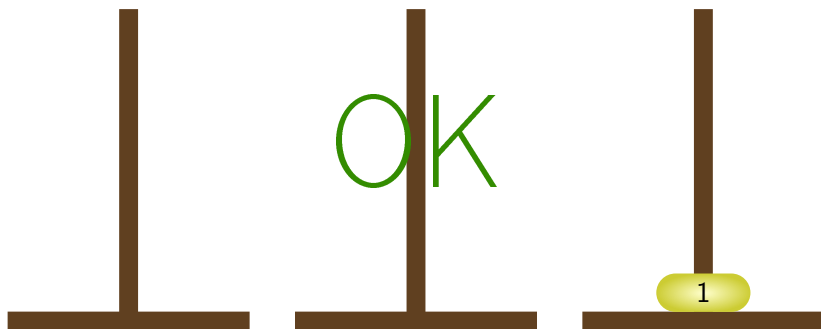


Recursão: Problema da Torre de Hanoi – 1 disco(s)



Moveu disco do pino 1 para o pino 3.

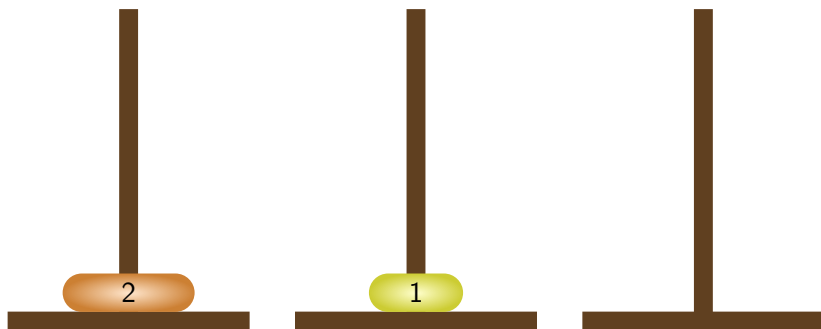
Recursão: Problema da Torre de Hanoi – 1 disco(s)



Recursão: Problema da Torre de Hanoi – 2 disco(s)

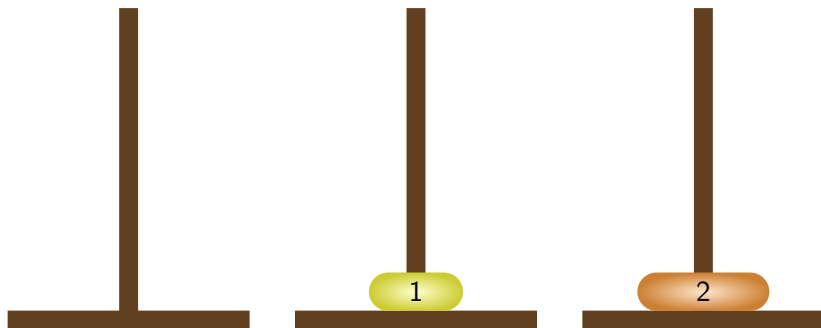


Recursão: Problema da Torre de Hanoi – 2 disco(s)



Moveu disco do pino 1 para o pino 2.

Recursão: Problema da Torre de Hanoi – 2 disco(s)



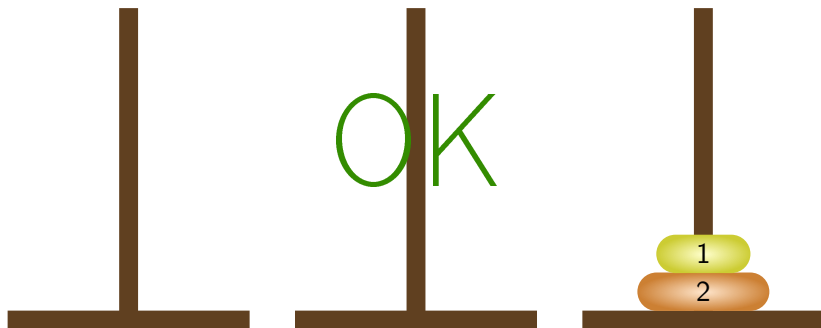
Moveu disco do pino 1 para o pino 3.

Recursão: Problema da Torre de Hanoi – 2 disco(s)



Moveu disco do pino 2 para o pino 3.

Recursão: Problema da Torre de Hanoi – 2 disco(s)



Recursão: Problema da Torre de Hanoi – 3 disco(s)

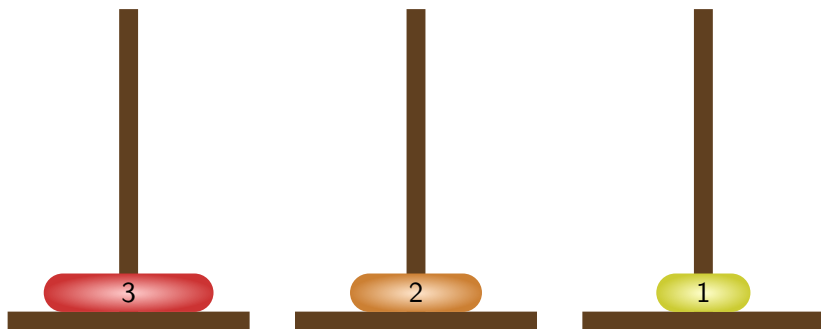


Recursão: Problema da Torre de Hanoi – 3 disco(s)



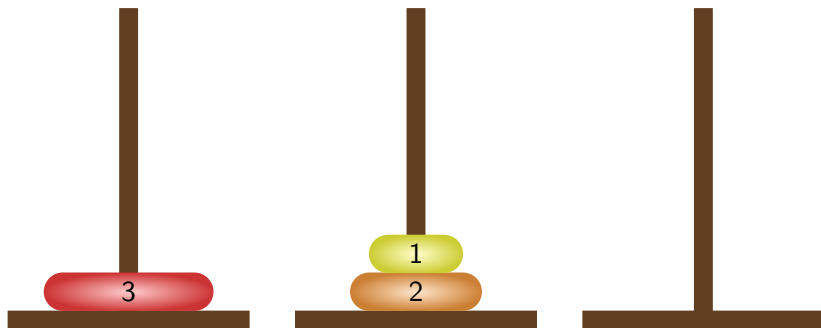
Moveu disco do pino 1 para o pino 3.

Recursão: Problema da Torre de Hanoi – 3 disco(s)



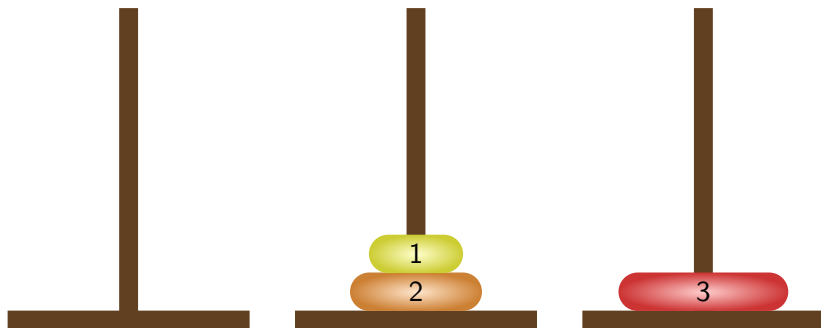
Moveu disco do pino 1 para o pino 2.

Recursão: Problema da Torre de Hanoi – 3 disco(s)



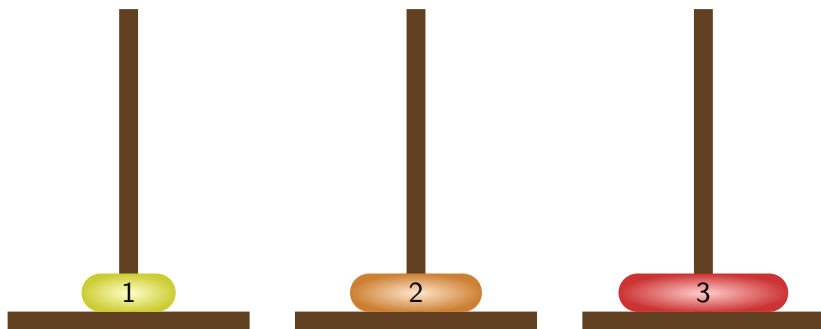
Moveu disco do pino 3 para o pino 2.

Recursão: Problema da Torre de Hanoi – 3 disco(s)



Moveu disco do pino 1 para o pino 3.

Recursão: Problema da Torre de Hanoi – 3 disco(s)



Moveu disco do pino 2 para o pino 1.

Recursão: Problema da Torre de Hanoi – 3 disco(s)



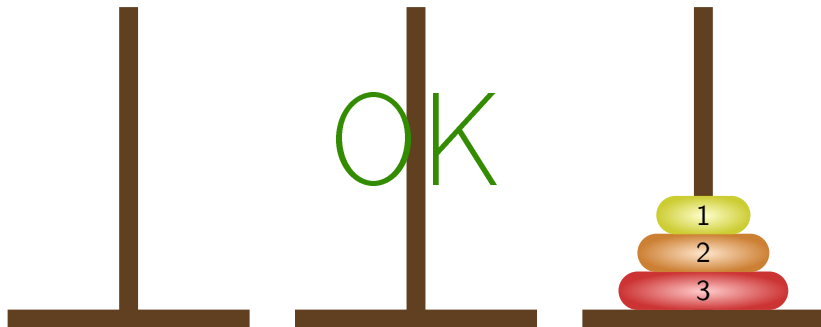
Moveu disco do pino 2 para o pino 3.

Recursão: Problema da Torre de Hanoi – 3 disco(s)

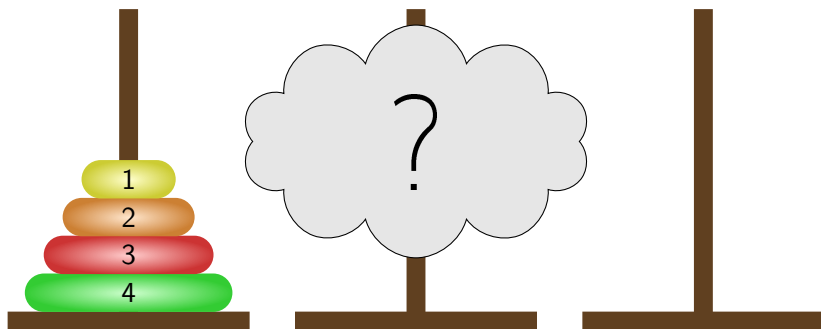


Moveu disco do pino 1 para o pino 3.

Recursão: Problema da Torre de Hanoi – 3 disco(s)



Torre de Hanoi – 4 disco(s)



Recursão

Torre de Hanoi – algoritmo

```
procedimento Hanoi(N, Orig, Dest, Temp)
  se N = 0 então
    não faça nada; // base da recursão!
  senão
    Hanoi(N-1, Orig, Temp, Dest);
    mover o N-ésimo menor disco de Orig para Dest;
    Hanoi(N-1, Temp, Dest, Orig);
  fim-se
fim
```

Recursão

Torre de Hanoi – em linguagem de montagem

- ▶ Ver arquivos `hanoi32.asm` e `hanoi32.s` no Paca.

Diferentes “visões” da memória

Endereçamento linear (ou plano)

- ▶ A memória é vista como um vetor de bytes
- ▶ Um único índice (endereço) seleciona algum byte específico do vetor.

Endereçamento segmentado

- ▶ A memória é vista como um vetor de bytes bidimensional
- ▶ Dois componentes são necessários para especificar um byte na memória: um *valor de segmento* e um *deslocamento (offset)* dentro do segmento.

Memória segmentada (família Intel 80x86)

- ▶ Vantagem da segmentação: aumento da capacidade de endereçamento do processador
- ▶ Nos processadores do 8086 até o 80286: cada segmento contém no máximo 64K (2^{16}) bytes, ou seja, os deslocamentos são número de até 16 bits
- ▶ Nos processadores a partir do 80386: cada segmento contém no máximo 4GB (2^{32}) bytes, ou seja, os deslocamentos são número de até 32 bits
- ▶ Em toda a família 80x86: o endereço de um segmento ocupa 16 bits, ou seja, temos no máximo 65536 segmentos diferentes

Memória segmentada (família Intel 80x86)

- ▶ Embora os processadores da família 80x86 usem memória segmentada, a memória real (física) conectada a UCP continua sendo um “vetor de bytes”
- ▶ O processador converte o valor de um segmento em um endereço físico de memória (por meio de uma função ou consultando uma tabela). Depois, o processador adiciona a esse endereço o valor do deslocamento para obter o endereço real de um dado na memória

Registradores de segmento (família Intel 80x86)

Todos os registradores de segmento na arquitetura de 32 bits têm 16 bits:

- ▶ CS – aponta para o segmento de código
- ▶ DS – aponta para o segmento de dados
- ▶ SS – aponta para o segmento da pilha
- ▶ ES, FS, GS – registradores de segmento extras, para manipular registradores de segmento ou apontar para outro lugar de interesse do programador

Registradores de segmento (na arquitetura de 32 bits)

- ▶ Pode-se usar o formato completo [seletor:deslocamento] em vários comandos, por exemplo:

```
MOV AX, [ES:DI]
```

- ▶ Se o seletor não é especificado, um segmento padrão é utilizado (por exemplo, DS para dados, CS para código e SS para pilha)
- ▶ A partir do 80386, cada segmento passou a ter até 4GB (2^{32}). Com segmentos tão grandes, a necessidade de usar os registradores de segmentos explicitamente passou a ser menor.

Bibliografia e materiais recomendados

- ▶ *The GNU Project Debbuger* – GDB
<http://www.gnu.org/software/gdb/>
- ▶ *Data Display Debbuger* – DDD
<http://www.gnu.org/software/ddd/>
- ▶ Capítulo 4 (*Memory Layout and Access*) do livro *The Art of Assembly Language Programming*, de R. Hyde
<http://cs.smith.edu/~thiebaut/ArtOfAssembly/artofasm.html>
- ▶ Notas das aulas de MAC0211 de 2010, feitas pelo Prof. Kon
<http://www.ime.usp.br/~kon/MAC211>

Cenas dos próximos capítulos...

- ▶ Noções básicas de sistemas operacionais
- ▶ Bibliotecas estáticas × bibliotecas dinâmicas