

# Introdução à Software Básico: Assembly em C / CISC vs. RISC

Departamento de Ciência da Computação  
Instituto de Ciências Exatas  
Universidade de Brasília

## Sumário

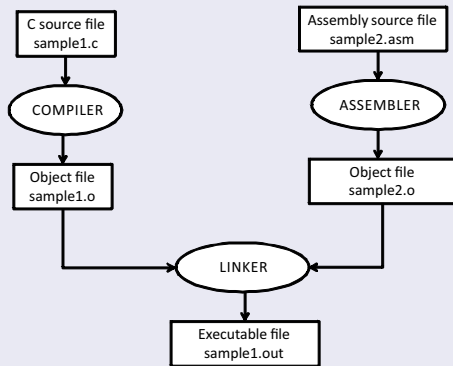
- 1 Assembly e Linguagens de Alto Nível
- 2 Chamando Funções Assembly em um programa C
- 3 Chamando Função C em um programa em Assembly
- 4 RISC vs. CISC

## Assembly e Linguagens de Alto Nível

- Parte de código escrito em Assembly pode ser utilizado por programas em Linguagens de Alto nível
- Algumas funções escritas em linguagens de Alto nível podem ser chamadas de um programa Assembly

## Processo de Compilação e Montagem

- `nasm -f elf sample2.asm -o sample2.o`
- `gcc -o sample1.out sample1.c sample2.o`



## Chamando Funções Assembly em um programa C

- Chamar funções significa passar parâmetros
- Em C:
  - `sum(a,b,c,d)`
  - Parâmetros empilhados da direita para esquerda: right-pusher

Right-Pusher



- O procedimento em Assembly deve fazer o setup da pilha
  - `push EBP`
  - `MOV EBP, ESP`
- O procedimento em Assembly deve preservar o conteúdo dos registradores ESI, EDI, EBP e todos os registradores de segmentos.

## Retornar valores para o C

- Para retornar valores para o programa em C, o procedimento em Assembly somente pode usar o registrador EAX
- Se o valor é menor a 4 bytes então EAX deve ter o valor a ser devolvido.
- Se o valor é maior a 4 bytes então EAX deve ter um ponteiro para o valor a ser devolvido.

## Exemplo


```
/* A simple program to illustrate how mixed-mode programs are
written in C and assembly languages. The main C program calls
the assembly language procedure testl
file name: chapter21/hll_ex1c.c
*/

#include <stdio.h>
int main(void)
{
    int X = 25, y = 70;
    int value;
    extern int testl (int, int, int);
    value = testl(x, y, 5);
    printf("Result = %d\n", value);
    return 0;
```

## Exemplo

```
/* A simple program to illustrate how mixed-mode programs are  
written in C and assembly languages. The main C program calls  
the assembly language procedure test!  
file name: chapter21/hll_ex1c.c  
*/
```

```
#include <stdio.h>  
int main(void)  
{  
    int X = 25, y = 70;  
    int value;  
    extern int testl (int, int, int);  
    value = testl(x, y, 5);  
    printf("Result = %d\n", value);  
    return 0;
```



```
push 5  
push 70  
push 25  
call test  
add ESP,12  
mov [EBP-12],EAX  
...
```



## Exemplo

```
;-----  
; This procedure receives three integers via the stack.  
; It adds the first two arguments and subtracts the  
; third one. It is called from the C program.  
;-----  
; filename: chapter21/hll_test.asm  
segment .text  
  
global test1  
  
test1:  
    enter 0,0  
    mov  EAX,[EBP+8]    ; get argument1 (x)  
    add  EAX,[EBP+12]   ; add argument 2 (y)  
    sub  EAX,[EBP+16]   ; subtract argument3 (5)  
    leave  
    ret
```

## Funções em Outros Módulos

- Em C, uma função em outro módulo deve ser declarada como `extern` para que possa ser utilizada:
  - `extern int test1 (int, int, int);`
- Em Assembly, para que uma função possa ser utilizada em outro módulo deve ser declarada como `global`
  - `global test1`
- Programa C que chama é responsável por limpar a pilha
  - `ret ;return simple in assembly program`

## Chamando Funções C em um programa Assembly

- Geralmente funções que dão muito trabalho em Assembly, quando eficiência não é um problema
  - `printf`, `scanf`, `math.h`, ...
- Ao ligar o programa em Assembly deve ser indicado a biblioteca contendo o função em C

## Exemplo

```
[SECTION .text]    ; Section containing code
extern puts        ; Simple "put string" routine from C library
global main        ; Required so linker can find entry point
main:
    push ebp       ; Set up stack frame for debugger
    mov ebp,esp
    push ebx       ; Program must preserve ebp, ebx, esi, & edi
    push esi
    push edi
    ;; Everything before this: use it for all ordinary apps!
    push dword msg ; Push a 32-bit ptr to the message on the stack
    call puts      ; Call the C library function for displaying strings
    add esp, 4     ; Clean stack by adjusting esp back 4 bytes
    ;; Everything after this: use it for all ordinary apps!
    pop edi        ; Restore saved registers
    pop esi
    pop ebx
    mov esp,ebp    ; Destroy stack frame before returning
    pop ebp
    ret           ; Return control to Linux
[SECTION .data]    ; Section containing initialised data
msg: db "Hello World... from Linux!!!",0
```

## Passando Parâmetros para a `main()` em C

- Como é sabido, a linguagem C permite que o usuário passe parâmetros para o programa C
  - `Myprog arq1.txt arq2.txt`
  - A quantidade de parâmetros que *main* recebe é variável
  - `int main(int argc, char **argv)`
  - `argc`: indica quantas strings o usuário digitou na linha de comando
  - `argv`: é um vetor de strings, que possui todas as strings que o usuário digitou na linha de comando

## Exemplo

```
/*Programa de Contagem regressiva */
/* Schildt, capitulo 6, pagina 149 */
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

int main (int argc, char *argv[]){

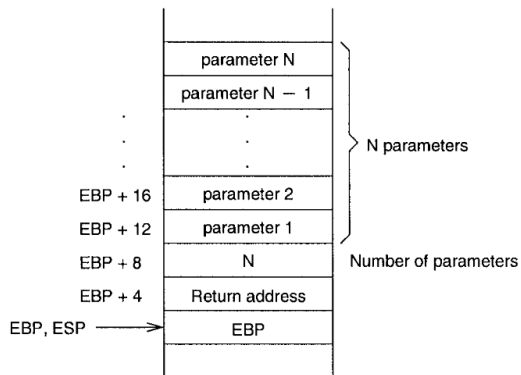
    int disp, count;

    if (argc < 2) {
        printf("Voce deve digitar o valor a contar \n");
        printf("na linha de comando. Tente novamente \n");
        exit(1);
    }

    if (argc == 3 && !strcmp(argv[2],"display")) disp = 1;
    else disp = 0;

    for (count = atoi(argv[1]); count; count -- )
        if (disp) printf("%d\n", count);
        putchar('\a'); /* isso ira tocar a campainha na maioria dos computadores */
        printf("Terminou\n");
        return(0);
    }
```

# Passando Parâmetros para a `main()` em C



- Se o argumento é um array (string), o sistema operacional armazena na pilha os endereços onde as strings dos argumentos dos programas são armazenados

## Projeto de um sistema computacional

- Características das arquiteturas computacionais são determinadas pelos domínios de aplicação:
  - Organização e hierarquia de memória;
  - Interface para o compilador e programador;
  - Consumo de energia.
- Segmentos mais populares: sistemas embarcados e de propósito geral



## Representantes do segmentos de embarcados

- Microcontroladores para serviços críticos (AVR, Texas, ARM);
- Infra-estrutura para redes de comunicação (MIPS, Power);
- Computação móvel com smartphones, tablets e netbooks (ARM).

## Representantes do segmento de propósito geral

- Computadores pessoais (IA-32, EM64T/AMD64);
- Servidores (SPARC, Power, Itanium)
- Processamento de alto desempenho (Mainframes, GPUs, clusters)

## Operação de Multiplicação CISC

- MUL [M1]
- Uma instrução de operação aritmética pode ser endereça diretamente para memória (no caso multipla o conteúdo de M1 com o acumulador A).

## Operação de Multiplicação RISC

- LOAD B, [M1]
- PROD A,B
- Operações aritméticas são feitas somente mediante registradores

## CISC

- Inclui instruções complexos de vários ciclos de relógio (LOAD-OPERATE)
- O programa é mais fácil e menor
- O hardware é mais complexo, usando uma maior quantidade de transistores.

## RISC

- Utiliza instruções simplificadas de preferência de um único ciclo de relógio (LOAD-STORE)
- O programa fica maior, porém mais controle sobre o desempenho
- Hardware mais simplificado, utiliza menor quantidade de transistores. Normalmente isso significa que pode ser investido mais espaço físico e transistores em registradores de uso geral.

## CISC

- Devido ao melhor controle do desempenho dos programas, a alta capacidade de realizar *pipeline*, a arquitetura RISC é mais usada nos sistemas embarcados.
- Devido a facilidade na programação de diversas tarefas a arquitetura CISC era mais utilizada nos PCs (uso geral)
- A partir da 6<sup>ta</sup> geração de processadores INTEL (1995) e da 5<sup>ta</sup> geração de processadores AMD (1996) não é mais utilizada a arquitetura CISC. É sim uma arquitetura híbrida CISC/RISC
  - Na arquitetura híbrida CISC/RISC, internamente o processador é RISC porém somente pode ser programado por um conjunto de instruções CISC.
  - O conjunto de instruções CISC é internamente traduzido para instruções RISC, as instruções RISC são conhecidas como **micro-instruções**.
  - Por essa razão atualmente tem se preferido a classificação entre arquiteturas como *load-store* e *load-operate* do que CISC e RISC.

## Próxima Aula

Assembly Floating point, e introdução a x64