

**Disciplina: 116432**  
**SOFTWARE BÁSICO**

**Aula #15**

**Carla Koike**

Universidade de Brasília – UnB  
Instituto de Ciências Exatas – IE  
Departamento de Ciência da Computação – CIC



# Última Aula...

- Procedimentos e Funções em Assembly

# Roteiro

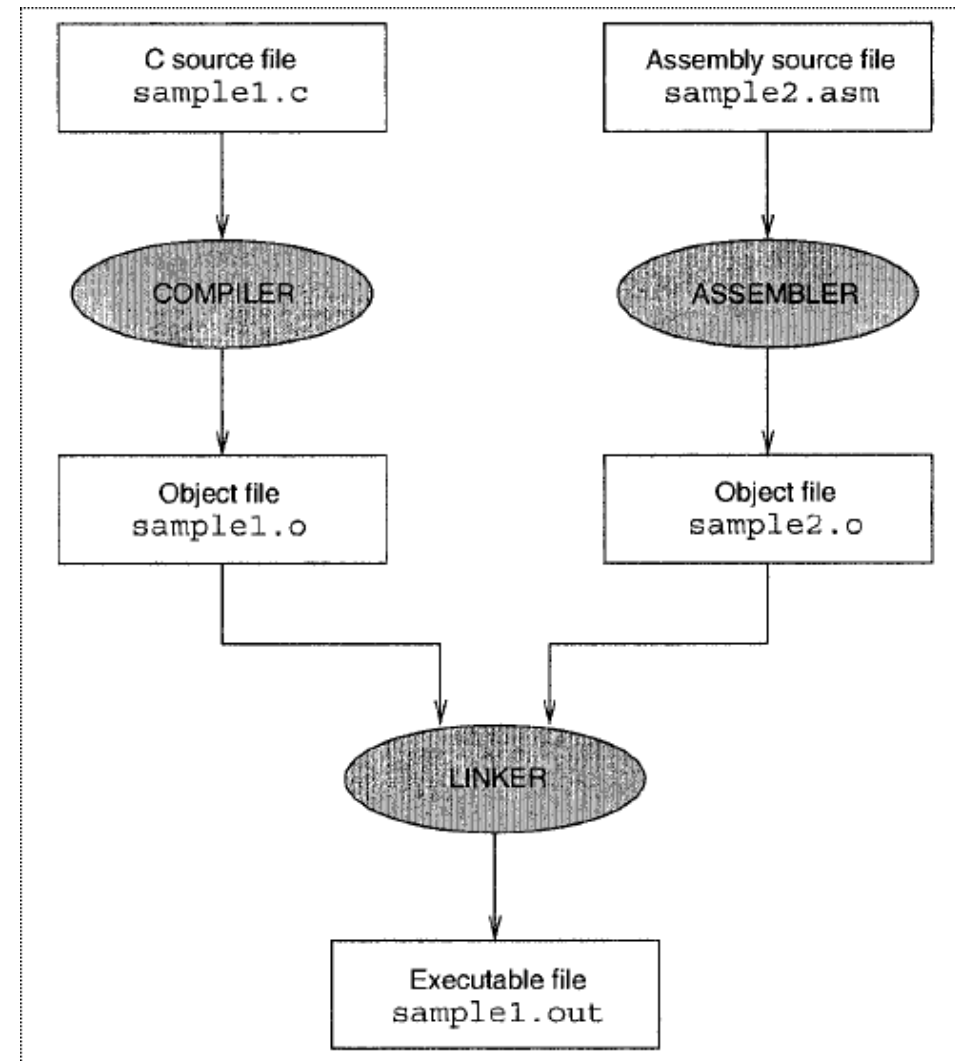
- Interface entre linguagem C e Assembly
- Macros em C e Assembly
- C (Schildt), capítulos 6 e 10
- Assembly (Dandamudi), capítulos 10 e 21
- Duntemann, capítulos 11 e 12

# 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

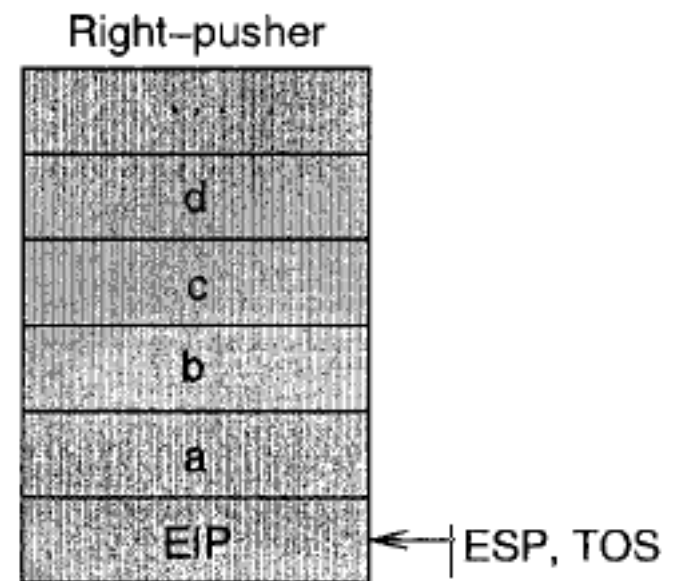
# Processo de Compilação e Montagem

```
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 são empilhados da direita para esquerda: *right-pusher*



# Chamando Funções Assembly em um programa C

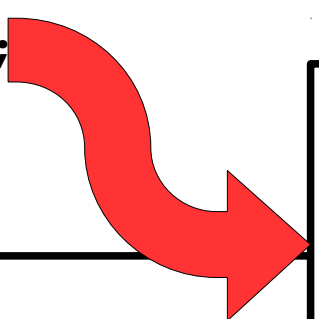
```
/* A simple program to illustrate how mixed-mode
programs are written in C and assembly
languages. The main C program calls the assembly
language procedure test1
file name: chapter21/h11_ex1c.c
*/

#include <stdio.h>
int main(void)
{
    int X = 25, y = 70;
    int value;
    extern int test1 (int, int, int);
    value = test1(x, y, 5);
    printf("Result = %d\n", value);
    return 0;
```

# Chamando Funções Assembly em um programa C

```
/* 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 test1
file name: chapter21/h11_ex1c.c
*/

#include <stdio.h>
int main(void)
{
    int X = 25, y = 70;
    int value;
    extern int test1 (int, int, int);
    value = test1(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
...
```



# Chamando Funções Assembly em um programa C

```
;-----  
; 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/h11_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`

# Exemplos:

- hll\_minmaxc.c e hll\_minmaxa.asm
- hll\_arraysumc.c e hll\_arraysuma.asm

# Chamando Funções Assembly em um programa C

- Programa principal em Assembly e função chamada em C
- Geralmente funções que dão muito trabalho em Assembly, quando eficiência não é um problema
  - printf, scanf, math.h, ...

# asm\_c.asm

```
[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
```

# Exercícios

- Altere o programa `asm_c.asm` para que a mensagem que o usuário imprimiu seja mostrada  $n$  vezes, onde  $n$  também é um dado fornecido pelo usuário. Tente fazer isso de duas formas:
  - Somente em assembly usando `io.o`
  - Usando `printf()`, `gets()` e `atoi()`, ou seja, usando funções em C.

# Passando Parâmetros para a main()

- 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 */
```

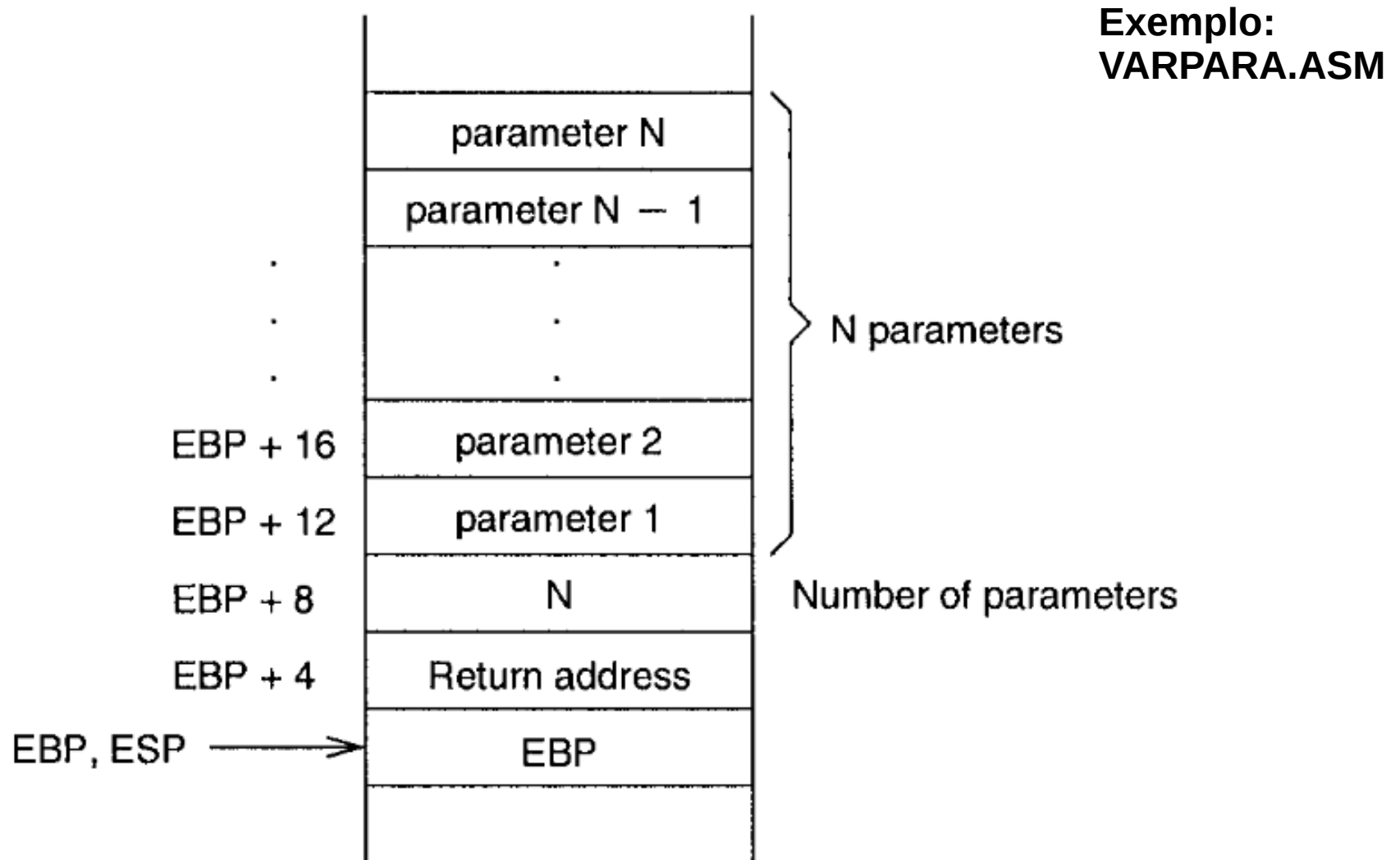
```
    printf("Terminou\n");
```

```
    return(0);
```

```
}
```



# Funções com Número Variável de Parâmetros



# Parâmetros argc e argv

- O sistema operacional armazena na pilha os endereços onde as strings dos argumentos dos programas são armazenados
- Programa showargs2.asm

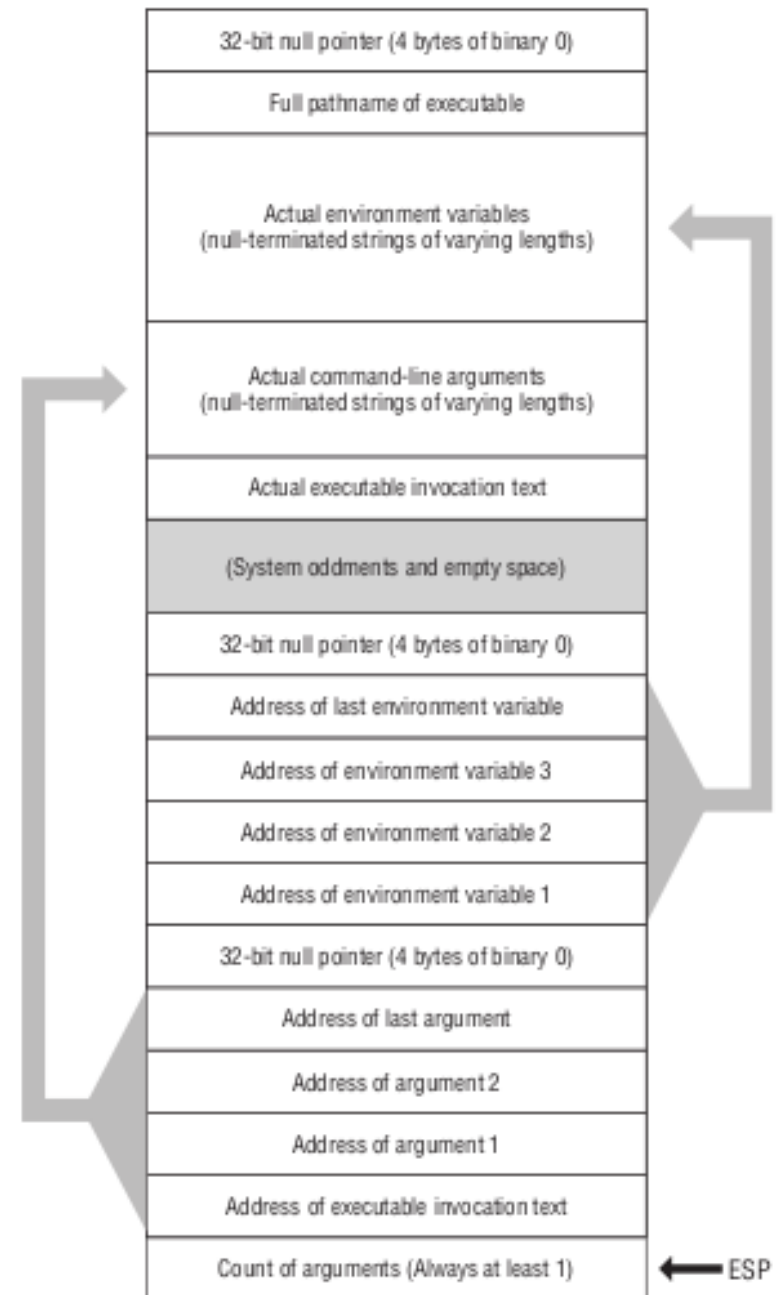


Figure 11-4: Linux stack at program startup

# Instrução xchg

- Troca os conteúdos dos operandos com 8, 16, ou 32 bits
- Permite realizar a troca sem usar uma terceira posição de memória
  - `xchg EAX,EDX`
  - `xchg [response],CL`
  - `xchg [total],DX`