Integração C e Assembly (Linux 32bits)

Prof. Eduardo Tavares

eagt@cin.ufpe.br

Assembly

Desvantagens na criação de programas completos em assembly

- Baixa produtividade
- Alto custo de manutenção
- Baixa portabilidade

Alternativa: uso conjunto de linguagens de baixo nível com linguagens de alto nível

C e Assembly

Duas formas usuais de integração:

- Código assembly *inline*
  - Inserção de código assembly em um código C
- Módulos em assembly separados
  - Integração durante o *link*

Inline Assembly

Uso da construção asm (ou __asm__)

Ex: asm(“incl %eax”)

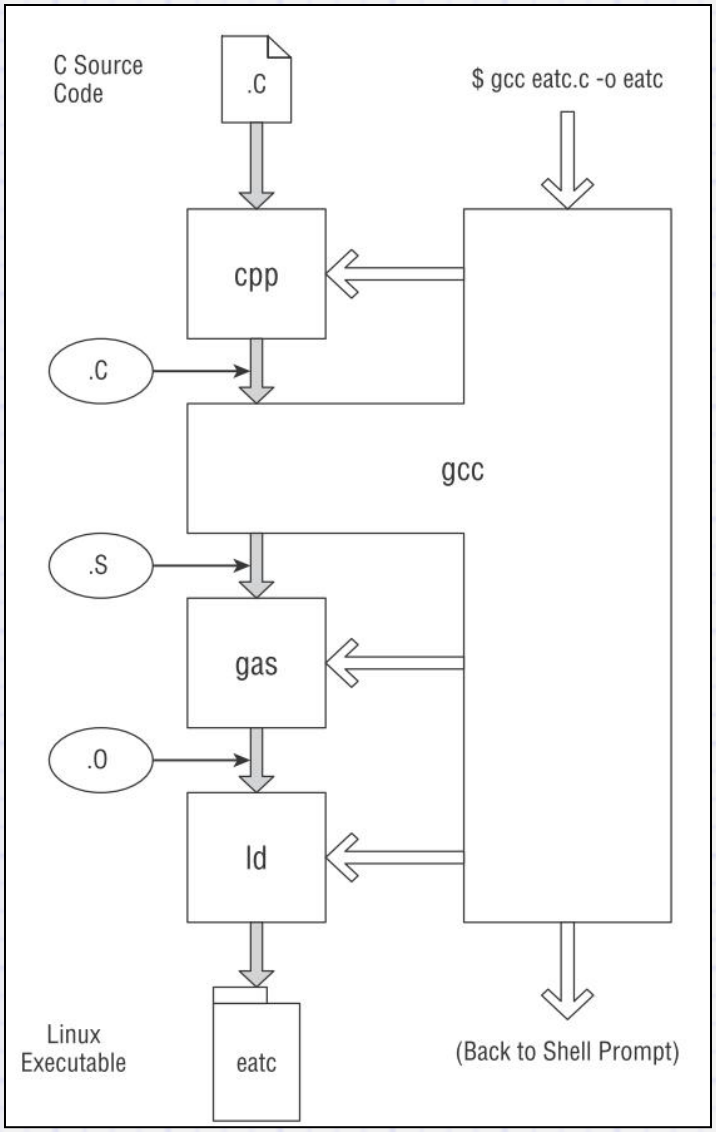
Não será adotado, pois usa a sintaxe da AT&T

GCC e arquivos executáveis (Linux)

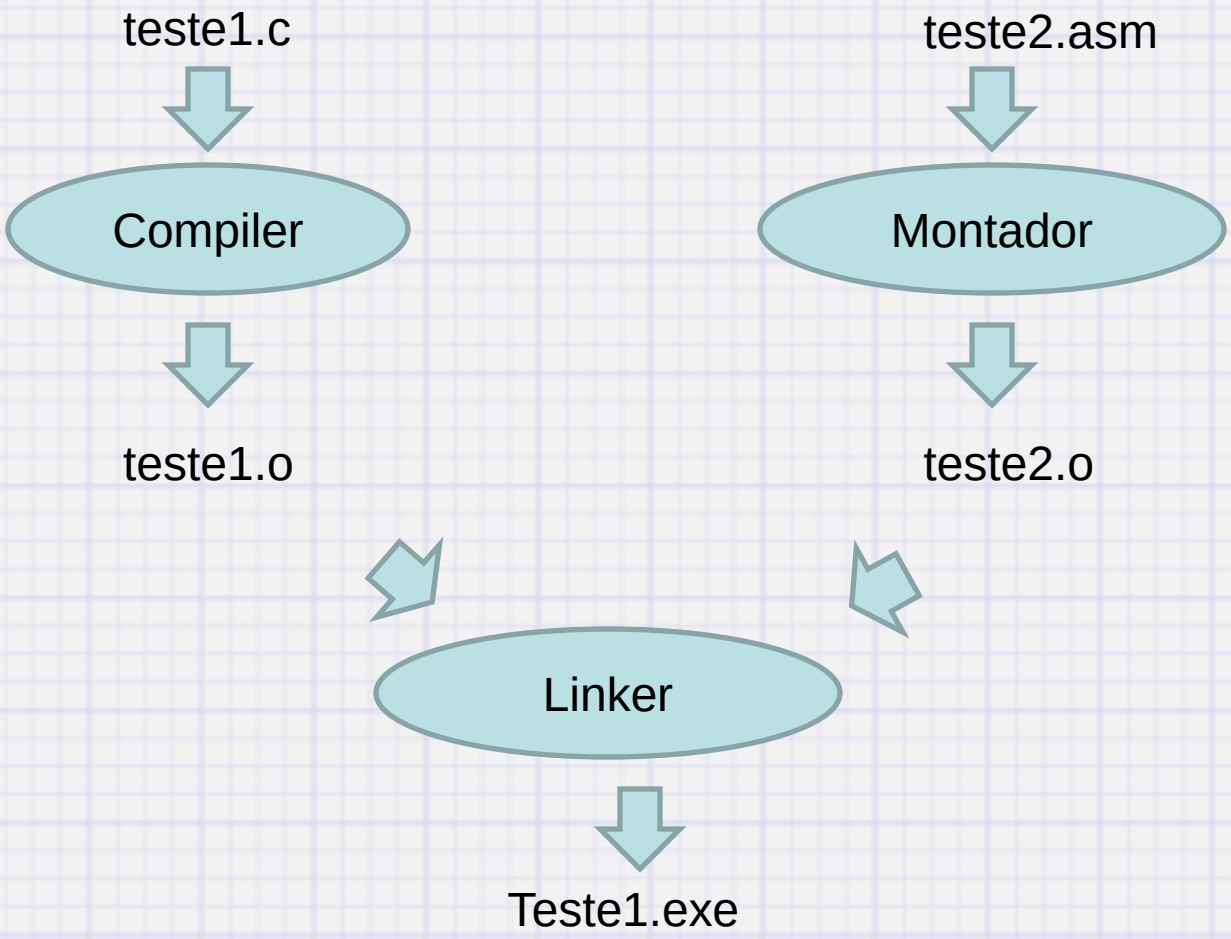
gcc eatc.c -o eatc

gcc -S eatc.c
(Assembly no formato da AT&T)

gcc -S -masm=intel eatc.c
(Assembly no formato Intel – somente em algumas versões do gcc)



C e Assembly


teste1.c
teste2.asm
Compiler
Montador
teste1.o
teste2.o
Linker
Teste1.exe

Stack frames

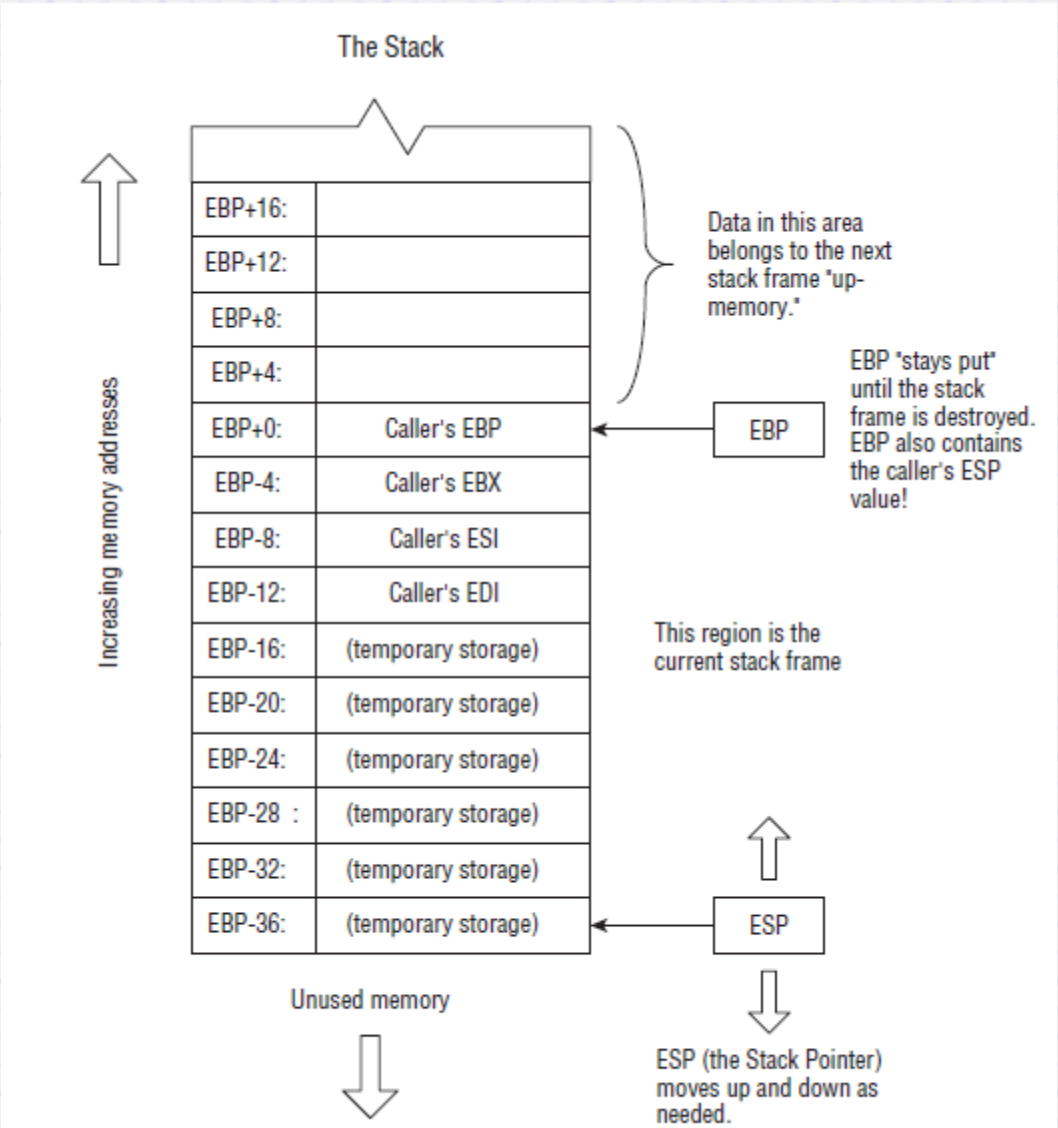
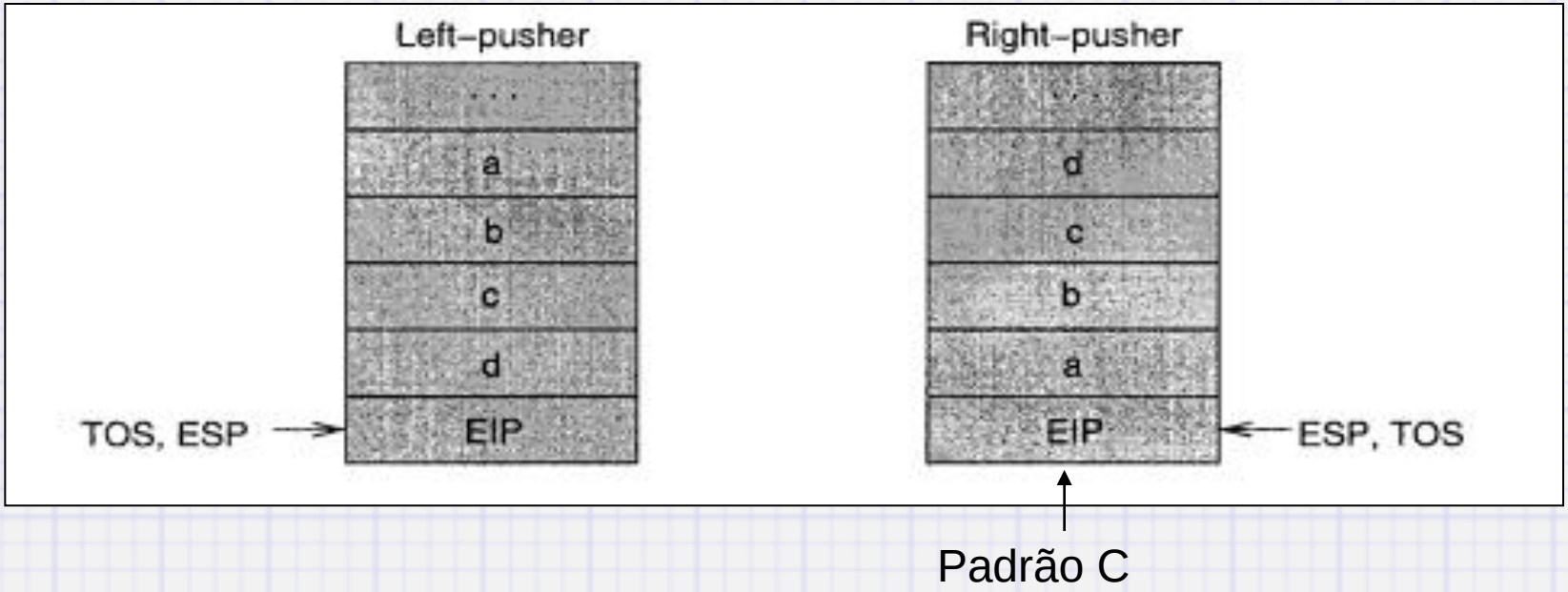


Figure 12-3: A stack frame

C e Assembly

- Passagem de Parâmetros em Funções C

sum(a,b,c,d)



C e Assembly

- Passagem de Parâmetros em Funções C

```
int main(void)
{
    int     x=25, y=70;
    int     value;
    extern  int test(int, int, int);

    value = test (x, y, 5);
    . . .
}
```

```
push    5
push    [EBP-8]
push    [EBP-4]
call    test
add     ESP,12
mov     [EBP-12],EAX
```

C e Assembly

- Passagem de Parâmetros em Funções C
  - Em geral os registradores **EBP, EBX, ESI e EDI** são preservados (ou seja, não são usados nas funções), mas isto não é uma regra.

C e Assembly

- Retorno de Valores de Funções C
  - Valores de 8, 16 e 32 bits são devolvidos em **EAX**,
  - Valores de 64 bits são devolvidos em **EDX:EAX**,
  - Valores de ponto flutuante são devolvidos em **ST0**,
  - Endereços (ponteiros) são devolvidos em **EAX**.

C e Assembly

- Passagem de Parâmetros em Funções C
  - Instrução enter
  - enter bytes,level
    - bytes – número de bytes relativos as variáveis locais da função (procedimento).
    - level – nível de aninhamento do procedimento.

C e Assembly

- Passagem de Parâmetros em Funções C
  - Instrução enter

Exemplo:

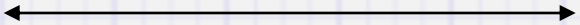
enter xx,0

```
push EBP
mov EBP,ESP
sub ESP,XX
```

C e Assembly

- Passagem de Parâmetros em Funções C
  - Instrução leave

Exemplo:

leave  mov ESP, EBP
pop EBP

C e Assembly

```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;
}
```

```
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
```

C e Assembly

```
SECTION .data
CDE db 'O valor eh %d',0AH,0

SECTION .text
                global main
                extern printf
                main:
                        enter 0,0
                        push ebx
                        push esi
                        push edi

                        push 5
                        push dword CDE
                        call printf
                        add esp,8

                        pop edi
                        pop esi
                        pop ebx

                        leave
                ret
```

```
nasm –f elf program.asm
gcc program.o – o program
```

C e Assembly

- Usando Funções de Tempo do C
  - Está estabelecido que a “Era” Unix iniciou-se em 1 de Janeiro de 1970 as 00:00:00 hs
  - A cada segundo que se passa adiciona-se 1 a este valor.
  - Quando você ler o tempo ou uma data através de bibliotecas do C, você obtêm o número atual desta associado a uma variável.
  - Esta variável é denominada *time_t*.
  - Para obter o valor de *time_t*, chama-se a função *time*.

C e Assembly

- Usando Funções de Tempo do C
  - A função *time* pode retornar valores de duas maneiras:
    - em EAX
    - em um *buffer* que você tenha definido
      - Para ter o tempo armazenado no *buffer* você tem que passar o ponteiro do endereço inicial do *buffer* como parâmentro (via pilha).
      - Se você não quer armazenar o tempo no *buffer*, você tem que passar um ponteiro nulo (0) como parâmetro.

C e Assembly

- Usando Funções de Tempo do C

```
SECTION .bss
        oldtime    resb        4

SECTION .text
        global main
        extern time
main :
        push epb       ;poderia ser substituído
        mov ebp,esp    ;por enter 0,0

        push ebx
        push esi
        push edi

        push dword 0
        call time
        add esp, 4
        mov [oldtime],eax
```

```
pop edi
pop esi
pop ebx

mov esp,ebp ;leave
pop ebp

ret
```

C e Assembly

- Usando Funções de Tempo do C
  - Há uma estrutura que agrupa nove informações de 32 bits relacionadas com o tempo: a estrutura *tm*.
  - Valores contidos na estrutura *tm*:

OFFSET IN BYTES	C LIBRARY NAME	DEFINITION
0	tm_sec	Seconds after the minute, from 0
4	tm_min	Minutes after the hour, from 0
8	tm_hour	Hour of the day, from 0
12	tm_mday	Day of the month, from 1
16	tm_mon	Month of the year, from 0
20	tm_year	Year since 1900, from 0
24	tm_wday	Days since Sunday, from 0
28	tm_yday	Day of the year, from 0
32	tm_isdst	Daylight Savings Time flag

C e Assembly

- Usando Funções de Tempo do C

```
SECTION .data
yrmsg      db “O ano e’ %d.”,10,0

SECTION .bss
oldtime    resb        4

SECTION .text
           global main
           extern time
           extern localtime
           extern printf
main :
           push ebp; enter 0,0
            mov ebp,esp

           push ebx
           push esi
           push edi
```

```
push dword 0
call time
add esp,4
mov [oldtime],eax

push dword oldtime
call   localtime
mov  edx, dword[eax+20]
add edx,1900
push  edx
push dword yrmsg
call   printf
add esp, 12

pop edi
pop esi
pop ebx

mov esp,ebp ;leave
pop ebp

ret
```