Programação em *Assembly*

Procedimentos e funções

IA32

Contexto da função – *Stack Frame*

O contexto de cada função, definido como **o conjunto de dados e informação de controlo usado pela função**, é armazenado na *stack*, numa estrutura designada por ***stack frame*** ou *activation record*.

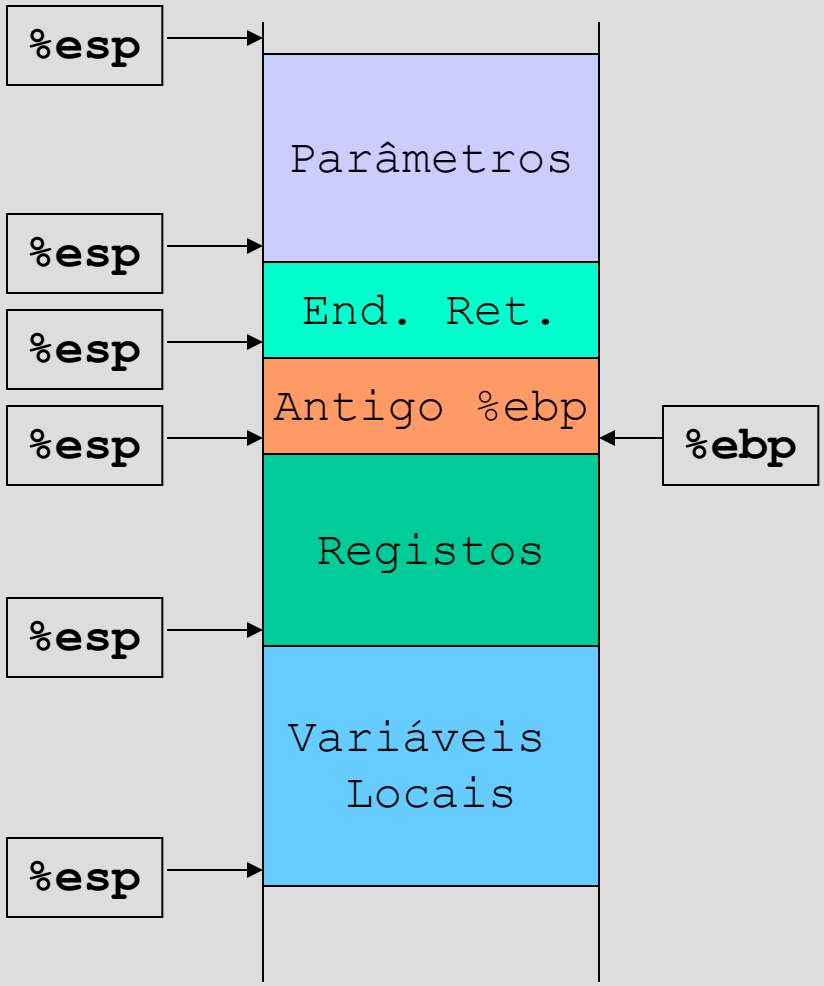
Cada função tem a sua própria *stack frame.*

De um modo geral esta contém:

- Argumentos
- Endereço de retorno
- *Frame pointer* anterior
- Registos salvaguardados
- Variáveis locais

IA32 – *Stack frame*

1. A função que invoca coloca os parâmetros na *stack* (`push op`)
2. A função que invoca coloca o endereço de retorno na *stack* (`call addr`)
3. A função invocada guarda %ebp (`pushl %ebp`)
4. A função invocada copia %esp para %ebp (`movl %esp, %ebp`)
5. A função invocada salvaguarda alguns registos (`pushl regs`)
6. A função invocada reserva espaço para variáveis locais (`subl $imm, %esp`)

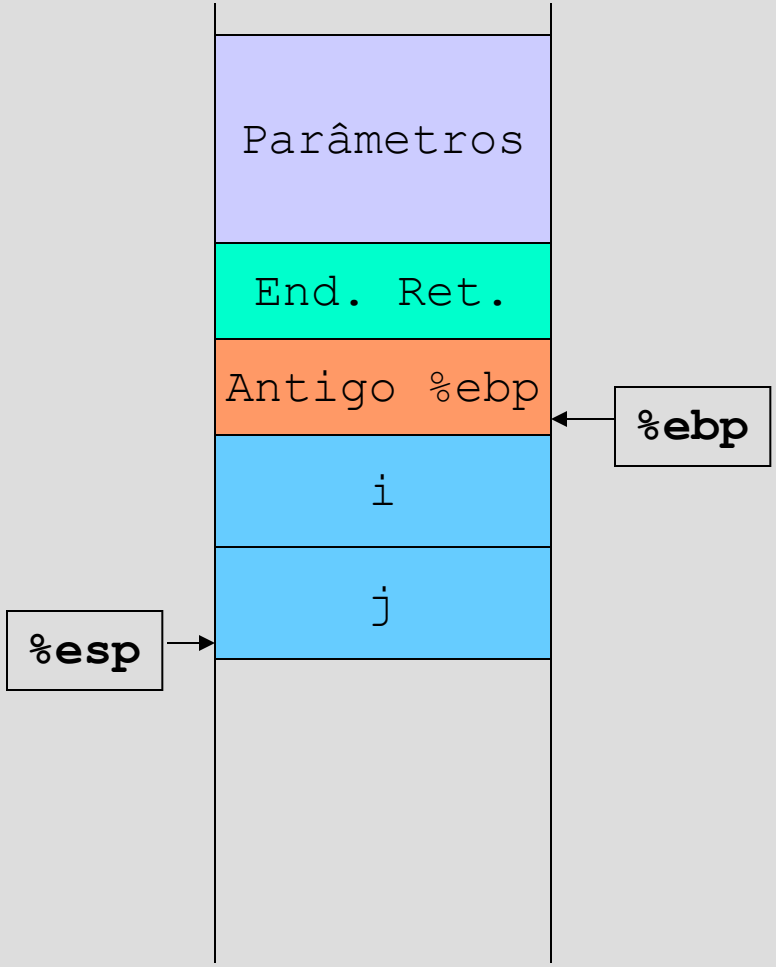


IA32 – *Stack frame*

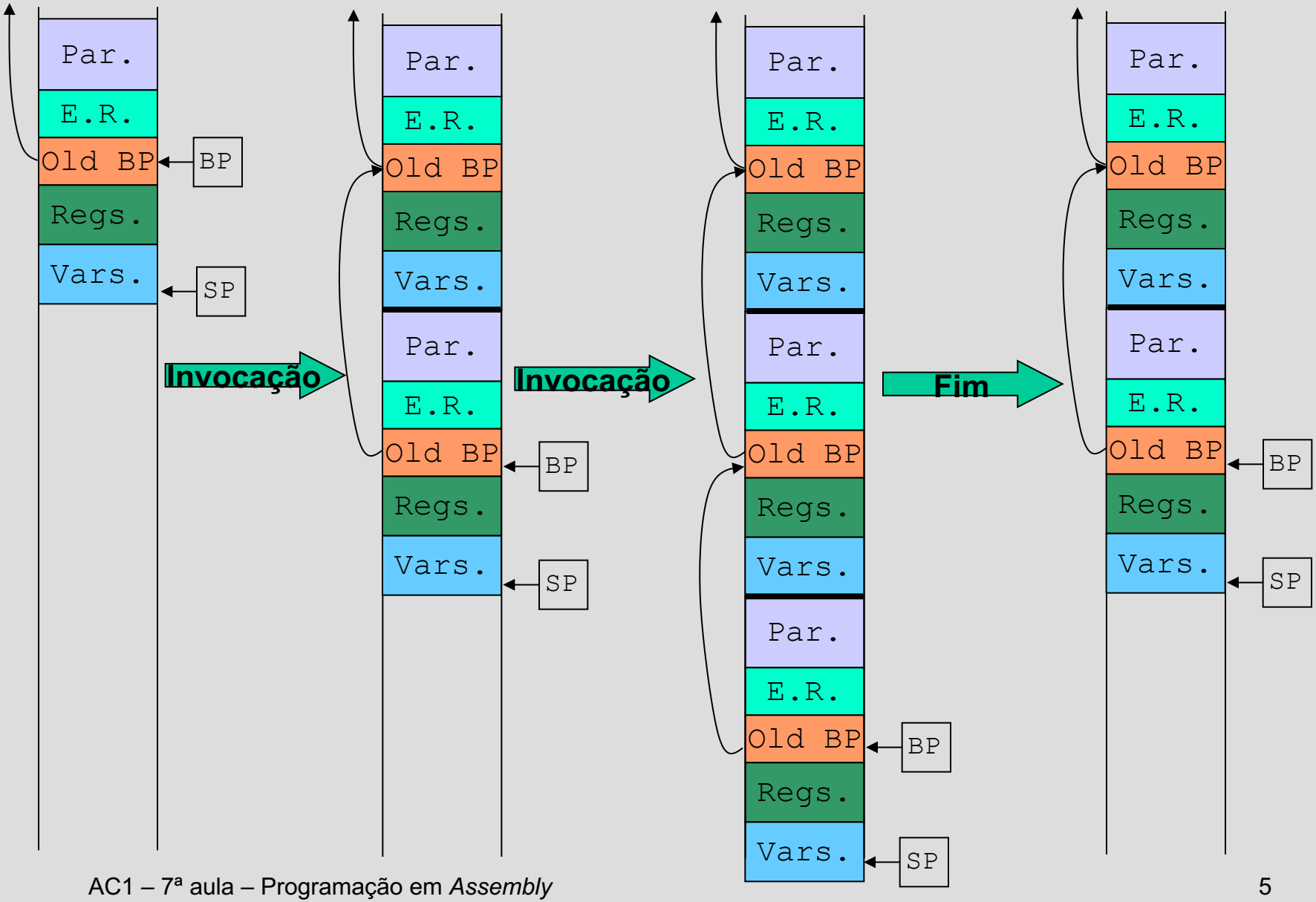
```
int main (int argc, char **argv)
{  int i, j=10;

   i = j * 4; }
```

```
main:
  pushl %ebp
  movl %esp, %ebp
  subl $4, %esp          ;espaço p/ i
  pushl $10              ; j = 10
  movl -8($ebp), %eax     ; %eax=j
  sall 2, %eax           ; %eax = j*4
  movl %eax, -4(%ebp)    ; i=%eax
  leave
  ret
```

```
leave ⇔ movl %ebp,%esp;popl %ebp
```



IA32 – *Stack Frame*



IA32 – Salvaguarda de registos

```
main:
   ...
   movl $10, %eax
   call func
   ...
```

Qual o valor que está em **`%eax`** após o **`call`**?

Convenciona-se que:

1. ***caller save*** – alguns registos podem ser alterados pela função invocada. Se a função que invoca precisar de os manter, então guarda-os na *stack.*

2. ***callee save*** – alguns registos **não** podem ser alterados pela função invocada. Se esta os alterar, deve guardar o valor anterior na *stack.*

caller save	**callee save**
%eax, %ecx, %edx	%ebx, %esi, %edi %ebp

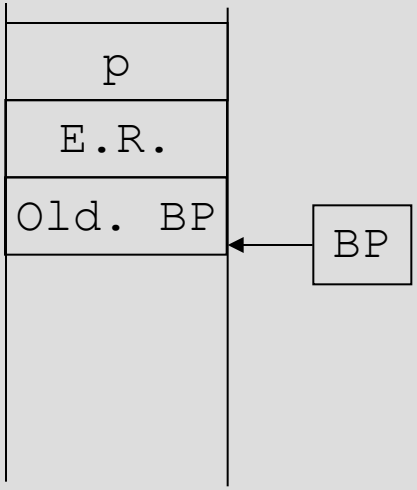
IA32 – Funções e procedimentos

```
int main ()
{ int i, accum=0;

  for (i=0;i<100;i++)
    accum += f(i);
}
```

```
main:
  pushl %ebp
  movl %esp, %ebp
  pushl %ebx
  pushl %esi
  movl $0, %ebx   ; accum=0
  movl $0, %esi   ; i=0
ciclo:
  pushl %esi      ; parâmetro
  call f
  addl $4, %esp   ; tirar parâm.
  addl %eax, %ebx
  incl %esi
  cmpl $100, %esi
  jl ciclo
  pop %esi
  pop %ebx
  leave
  ret
```

IA32 – Funções e procedimentos

```
int f (int p)
{ int j, ret=1;

  for (j=p;j>1;j--)
    ret *= j;
  return (ret);
}
```

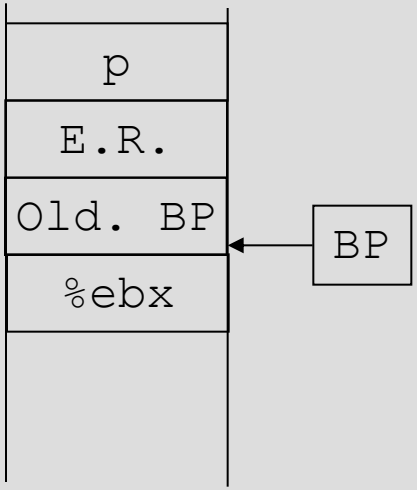



```
f:
  pushl %ebp
  movl %esp, %ebp
  movl $1, %eax   ; ret=1
  movl 8(%ebp), %ecx   ; j=p
  jmp f_ciclo
ciclo:
  mull %ecx, %eax    ; ret*=j
  decl %ecx          ; j—
f_ciclo:
  cmpl $1, %ecx
  jg ciclo
  leave
  ret
```

NOTA: `f()` não invoca nenhuma função.

IA32 – Funções e procedimentos

```
int f (int p)
{ int ret;

  if (p>1)
    ret= p * f(p-1);
  else ret=1;
  return (ret);
}
```




```
f:      pushl %ebp
        movl %esp, %ebp
        pushl %ebx
        movl 8(%ebp), %ebx
        cmpl $1, %ebx    ; p>1??
        jle else
        leal -1(%ebx), %ecx
        pushl %ecx ; parâmetro
        call f
        addl $4, %esp   ; tirar parâm.
        mull %ebx, %eax ;ret=p*f(p-1)
        jmp f_if
else:   movl $1, %eax    ; ret=1
f_if:   pop %ebx
        leave
        ret
```

NOTA: `f()` recursiva.

IA32 – Funções e procedimentos

Tema	Hennessy [COD]	Bryant [CS:APP]
IA32 – Procedimentos e funções		Sec 3.7 e 3.11