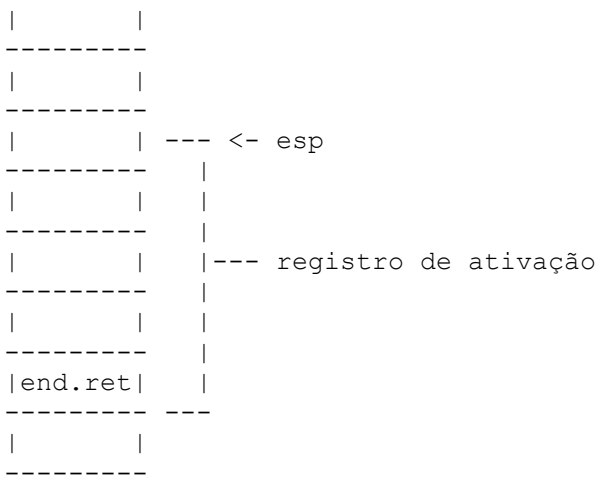
Extraído de http://www.inf.puc-rio.br/~inf1612/corrente/aula9.html.
Material original desenvolvido, pela PUC-RIO.
Baseado no livro: Computer Systems, A programmer's perspective. Randal Bryant and David O'Hallaron. Prentice Hall. 2003

Registros de Ativação

A medida que um programa executa e rotinas vão sendo chamadas, a pilha de execução cresce, refletindo as chamadas.

Além dos endereços de retorno, é comum cada procedimento usar a pilha para armazenar variáveis locais e outros temporários. Por isso, associamos a cada chamada de procedimento uma área da pilha, chamada de *registro de ativação*.



Para facilitar o acesso aos valores armazenados no registro de ativação, em geral se mantém um ponteiro para o "início" dele. No pentium, o registrador ebp é utilizado para guardar o endereço do início do registro de ativação corrente.

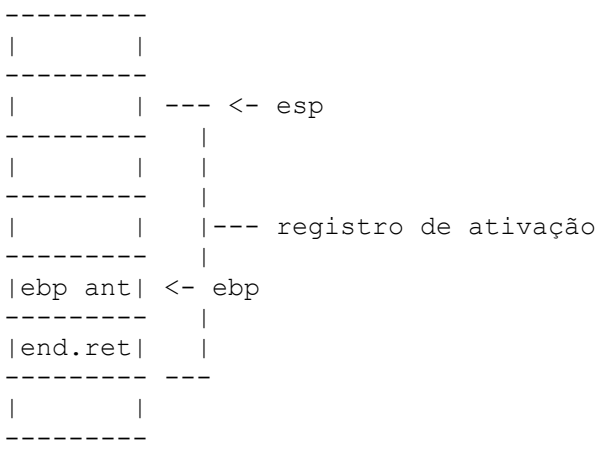
É por isso que o código tipicamente gerado por um compilador para um procedimento começa com:

```
pushl %ebp ; salva o endereço do registro de ativação anterior
movl %esp, %ebp
```

e termina com:

```
movl %ebp, %esp
popl %ebp ; restaura o endereço do registro de ativação anterior
```

ou seja, em cada instante temos na pilha de execução:



Variáveis Locais

O código de uma rotina pode reservar espaço para uma ou mais variáveis locais. Repare que para essa reserva não existem convenções de ordem, já que apenas o código interno à própria rotina faz acesso a essas variáveis.

É comum termos o seguinte trecho de çódigo para o início de um procedimento:

```
      push %ebp
      mov %esp, %ebp
      sub $20, %esp    /* 20 bytes para variáveis locais */
```

O acesso às variáveis locais é feito através de endereços como -4(%ebp), -8(%ebp), etc...

Como exemplo, considere o seguinte código em C:

```
void troca (int *x, int *y) {
  int tmp;
  tmp = *x;
  *x = *y;
  *y = tmp;
}
```

A compilação desse código poderia gerar algo como:

```
troca:  push %ebp
        mov  %esp, %ebp
        sub  $4, %esp       /* reserva espaço na pilha para tmp */

        mov 8(%ebp), %eax   /* primeiro parâmetro: endereço de x */
        mov (%eax), %edx    /* pega valor de x */
        mov %edx, -4(%ebp)   /* tmp = *x */
        mov 12(%ebp), %ebx  /* segundo parâmetro: endereço de y */
        mov (%ebx), %edx     /* pega valor de y */
        mov %edx, (%eax)    /* *x = *y */
        mov -4(%ebp), %edx  /* le valor de temp */
```

```
mov %edx, (%ebx)     /* *y = tmp */

mov %ebp, %esp
pop %ebp
ret
```

Referências:

- CS:APP, 3.7
- notas de aula Karen Miller, cap. 10