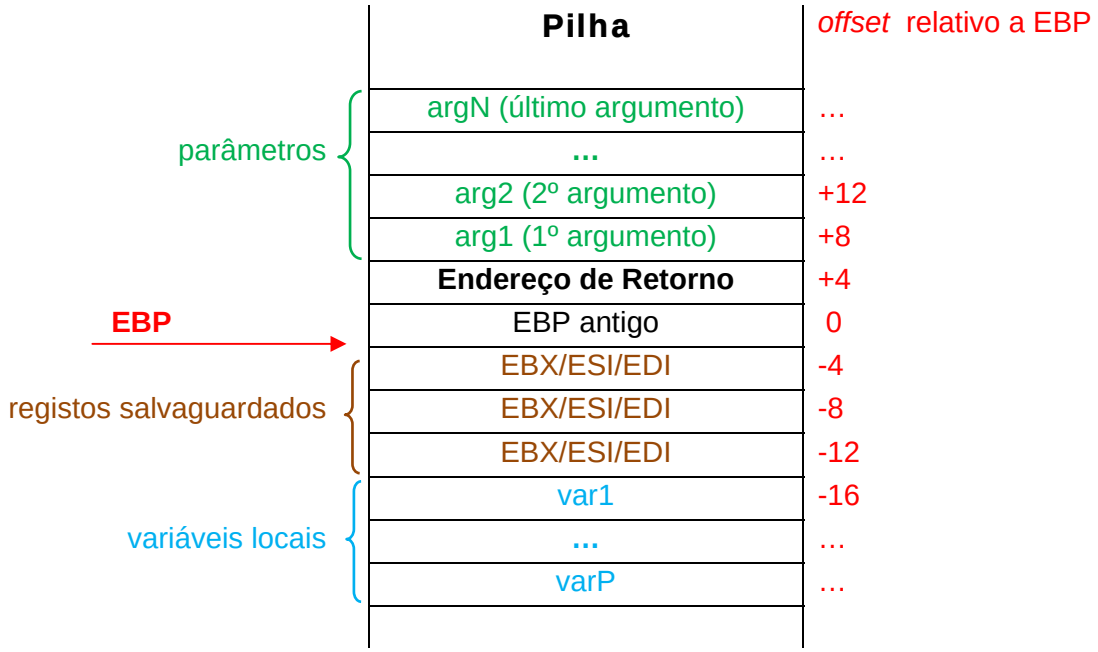
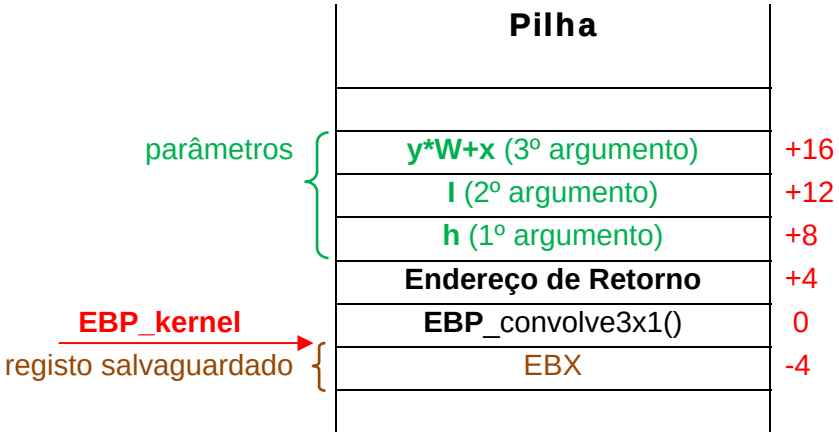
# Módulo 1 :: AC :: LEI 2012/13
(resolução)

***Activation Record*** genérico:




## Convolução 1D / assembly

1. *Activation record* da chamada da função **kernel()** em convolve3x1():



2. Cálculo de endereços e acesso a essas posições de memória:

inp[ndx-1]

```
movl   16(%ebp), %eax        // EAX = M[EBP+16] = ndx
subl   $1, %eax              // EAX = ndx-1
sall   $2, %eax              // EAX = (ndx-1) << 2 = (ndx-1)*4 (4 = tamanho dum INTEIRO)
addl   12(%ebp), %eax        // EAX = (ndx-1)*4 + M[EBP+12] = (ndx-1)*4 + inp
movl   (%eax), %edx          // EDX = M[(ndx-1)*4+inp]  (<=> inp[ndx-1] em C)
```

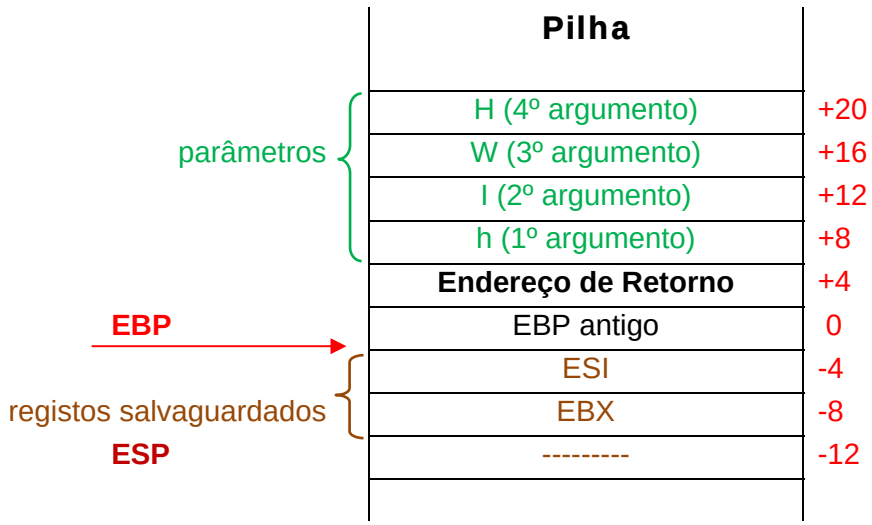
inp[ndx]

```
movl   16(%ebp), %eax        // EAX = M[EBP+16] = ndx
sall   $2, %eax              // EAX = ndx << 2 = ndx*4 (4 = tamanho dum INTEIRO)
addl   12(%ebp), %eax        // EAX = ndx*4 + M[EBP+12] = ndx*4 + inp
movl   (%eax), %eax          // EAX = M[ndx*4+inp]  (<=> inp[ndx] em C)
```

inp[ndx+1]

```
movl   16(%ebp), %eax        // EAX = M[EBP+16] = ndx
addl   $1, %eax              // EAX = ndx+1
sall   $2, %eax              // EAX = (ndx+1) << 2 = (ndx+1)*4 (4 = tamanho dum INTEIRO)
addl   12(%ebp), %eax        // EAX = (ndx+1)*4 + M[EBP+12] = (ndx+1)*4 + inp
movl   (%eax), %eax          // EAX = M[(ndx+1)*4+inp]  (<=> inp[ndx+1] em C)
```

3. *Activation record* da função **convolve3x1()**:



4. Instruções usadas no teste dos 2 ciclos for da função **convolve3x1()**:

// Teste do ciclo ***for*** em relação a ***y*** → *for (…; **y<H;** …) {…}*

```
cmpl   20(%ebp), %ebx        // compara y com M[EBP+20] = y-H
setl   %al                   // AL=1 se y<H
testb  %al, %al              // afetar as FLAGS de acordo com o resultado de AL&AL
jne    .L6                   // salta se (AL!=0) <=> salta se (AL=1) <=> salta se (y<H)
```

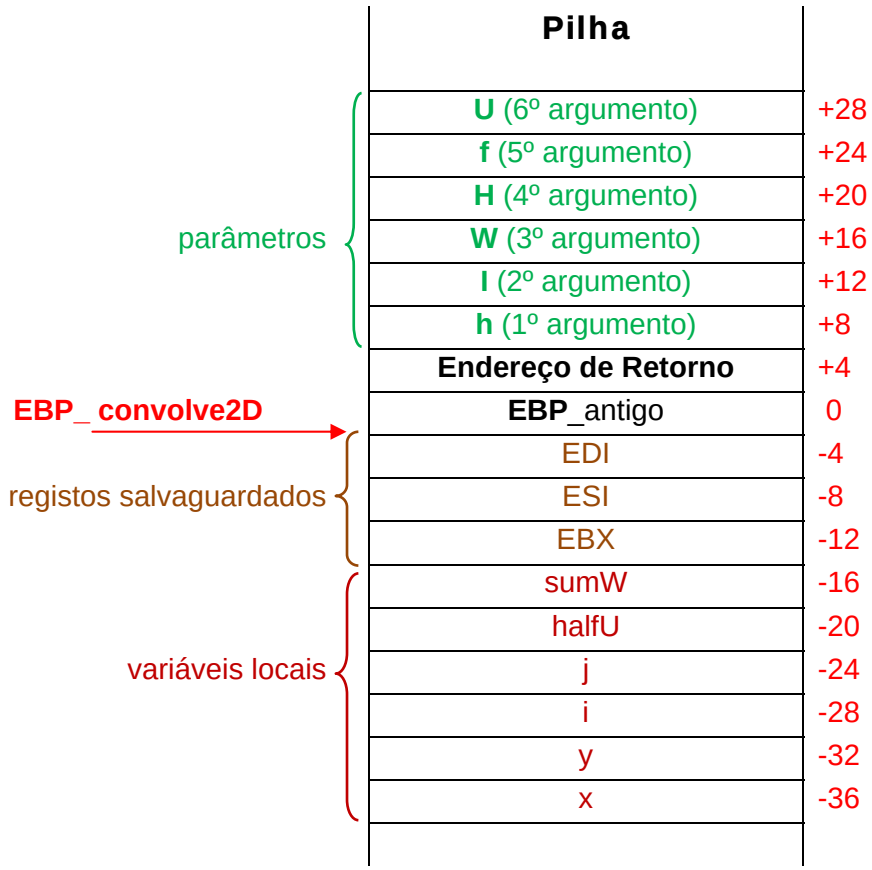
// Teste do ciclo ***for*** em relação a ***x*** → *for (…; **x<(W-1);** …) {…}*

```
movl   16(%ebp), %eax        // EAX = M[EBP+16] = W
subl   $1, %eax              // EAX = W - 1
cmpl   %esi, %eax            // compara (W-1) com x = (W-1)-x
setg   %al                   // AL=1 se (W-1)>x
testb  %al, %al              // afetar as FLAGS de acordo com o resultado de AL&AL
jne    .L7       // salta se (AL!=0) <=> salta se (AL=1) <=> salta se (W-1)>x <=> x<(W-1)
```

---

**Convolução 2D**

1. *Activation record* da função **convolve2D()**:



2. Instruções que calculam o endereço de ***f[]*** e ***I[]*** e que leem os seus valores de memória:

```
movl  -20(%ebp), %edx   // EDX = halfU
movl  -24(%ebp), %ebx   // EBX = j
leal  (%ebx,%edx), %edx // EDX = j+halfU   (LEAL usado para somar)
imull 28(%ebp), %edx    // EDX = (j+halfU)*U
addl  -28(%ebp), %edx   // EDX = (j+halfU)*U+i
addl  -20(%ebp), %edx   // EDX = (j+halfU)*U+i+halfU
sall  $2, %edx          // EDX = ((j+halfU)*U+i+halfU)*4
addl  24(%ebp), %edx    // EDX = ((j+halfU)*U+i+halfU)*4+f  <=> endereço de f[]
movl  (%edx), %ebx      // EBX = f[(j+halfU)*U+i+halfU]

movl  -24(%ebp), %edx         // EDX = j
movl  -32(%ebp), %esi         // ESI = y
leal  (%esi,%edx), %edx // EDX = y+j   (LEAL usado para somar)
imull 16(%ebp), %edx          // EDX = (y+j)*W
movl  -28(%ebp), %esi         // ESI = i
movl  -36(%ebp), %edi         // EDI = x
leal  (%edi,%esi), %esi // ESI = x+i   (LEAL usado para somar)
addl  %esi, %edx              // EDX = (y+j)*W + x+i
sall  $2, %edx                // EDX = ((y+j)*W+x+i)*4
addl  12(%ebp), %edx          // EDX = ((y+j)*W+x+i)*4+I     <=> endereço de I[]
movl  (%edx), %edx            // EDX = I[(y+j)*W+(x+i)]
```

3. Instruções que implementam o teste de cada um dos 4 ciclos da função:

```
// teste do ciclo for(…;j<=halfU; …)

movl  -24(%ebp), %eax   // EAX = j
cmpl  -20(%ebp), %eax   // compara j com halfU --> j-halfU
setle %al               // AL=1 se j<=halfU
testb %al, %al          // afetar as FLAGS de acordo com o resultado de AL&AL
jne   .L12              // saltar para início do ciclo for(j...) se (AL!=0) <=>
                        //    <=> saltar se (AL=1) <=> saltar se (j<=halfU)

// teste do ciclo for(…;i<=halfU; …)

movl  -28(%ebp), %eax   // EAX = i
cmpl  -20(%ebp), %eax   // compara i com halfU --> i-halfU
setle %al               // AL=1 se i<=halfU
testb %al, %al          // afetar as FLAGS de acordo com o resultado de AL&AL
jne   .L13              // saltar para início ciclo for(i...) se (AL!=0) <=>
                        //    <=> saltar se (AL=1) <=> saltar se (i<=halfU)

// teste do ciclo for(…;y<H; …)

movl  -32(%ebp), %eax   // EAX = y
cmpl  20(%ebp), %eax    // compara y com H --> y-H
setl  %al               // AL=1 se y<H
testb %al, %al          // afetar as FLAGS de acordo com o resultado de AL&AL
jne   .L16              // saltar para início ciclo for(y...) se (AL!=0) <=>
                        //   <=> saltar se (AL=1) <=> saltar se (y<H)

// teste do ciclo for(…;x<W; …)

movl  -36(%ebp), %eax   // EAX = x
cmpl  16(%ebp), %eax    // compara x com W --> x-W
setl  %al               // AL=1 se x<W
testb %al, %al          // afetar as FLAGS de acordo com o resultado de AL&AL
jne   .L17              // saltar para início ciclo for(x...) se (AL!=0) <=>
                        //   <=> saltar se (AL=1) <=> saltar se (x<W)
```