Smith College

Computer Science

CSC231 — Assembly

Week #13 — Fall 2017

Dominique Thiébaut
dthiebaut@smith.edu

Pascal Triangle

```
for n in 0 1 2 5 ; do
        echo "n =$n"
        ./hw7b $n
        echo ""
done
```

```
n =0

n =1
   1

n =2
   1   0
   1   1

n =5
   1   0   0   0   0
   1   1   0   0   0
   1   2   1   0   0
   1   3   3   1   0
   1   4   6   4   1
```

Short/Long Jumps

```
if ( x < y ) {
   f1(x);
}
else {
   f2(y);
}
```

```
if ( x >= y ) {
   f2(y);
}
else {
   f1(x);
}
```

Short/Long Jumps

```
if:   cmp eax, ebx
      jge else
then:…
      …
      jmp  endIf
else:…
      …
endIf:
```

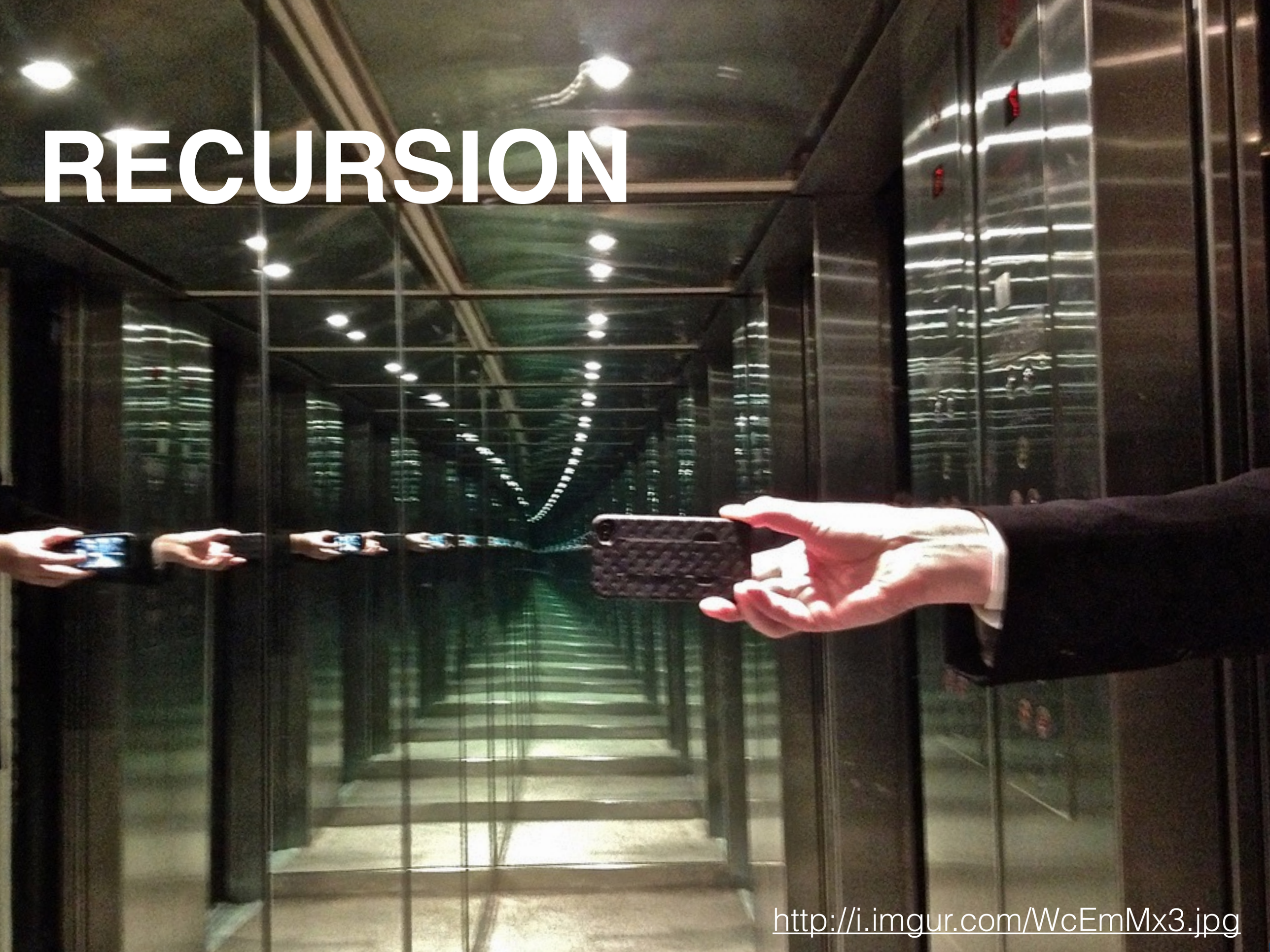
Short/Long Jumps

```
if:   cmp eax, ebx
      jge else
then:…
      …
      jmp  endIf
else:…
      …
endIf:
```

same

```
if:   cmp eax, ebx
      jl  then
      jmp else
then:…
      …
      jmp  endIf
else:…
      …
endIf:
```

RECURSION



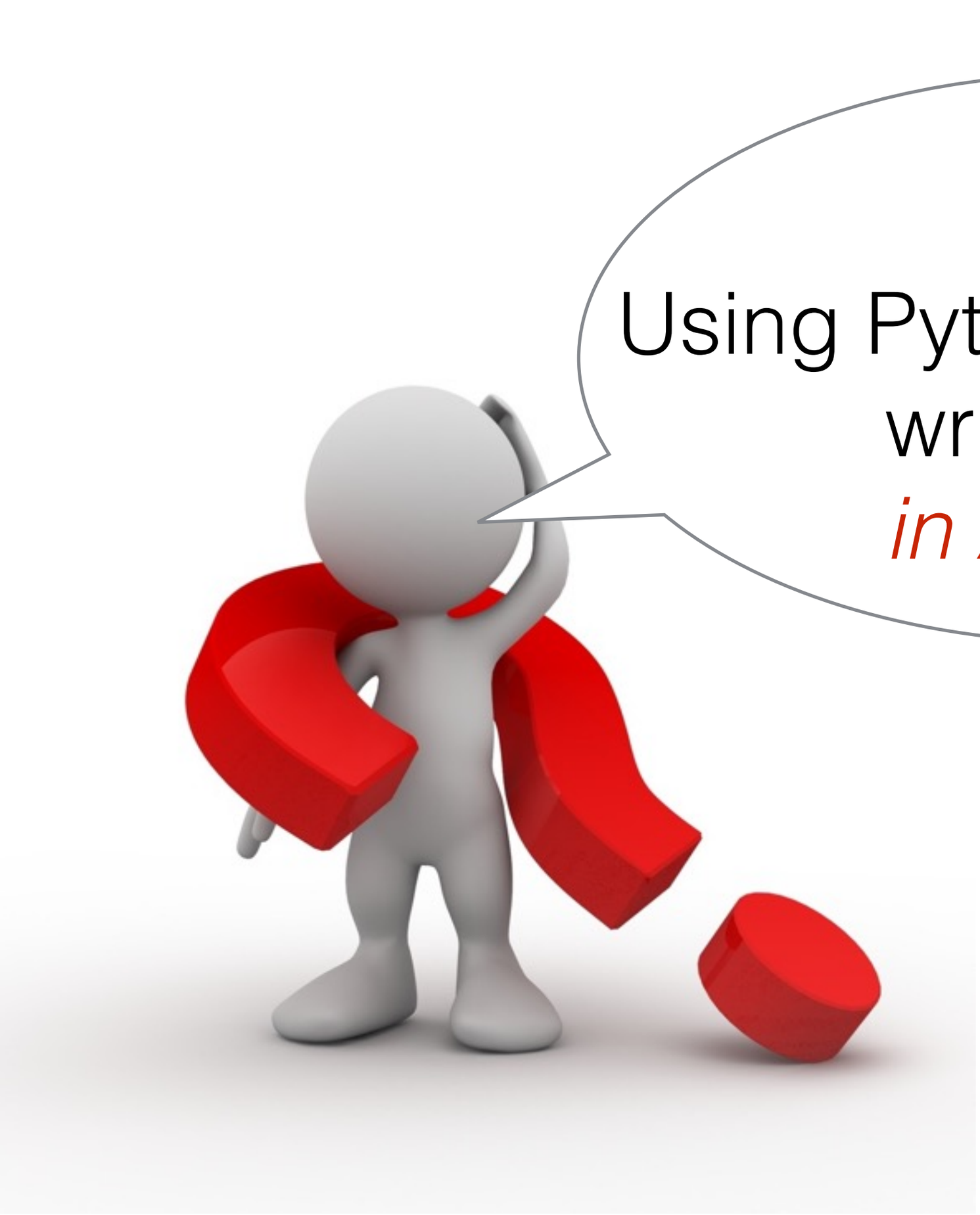
http://i.imgur.com/WcEmMx3.jpg

```
Python 3.5.0b1 (v3.5.0b1:071fefbb5e3d, May 23 2015, 18:22:54)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.

>>> def fact( n ):
      if n <= 1:
         return 1
      return n * fact( n - 1 )

>>> fact( 3 )
6
>>> fact( 5 )
120
>>> fact( 20 )
2432902008176640000
>>> fact( 100 )
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
>>>
```

$$n! = \begin{cases} 1 & \text{if } n<=1 \\ n * (n-1)! & \text{otherwise} \end{cases}$$

Class Exercise:
Act out the Play "Compute Fact(5)"

Class Exercise 2: Write a Python Program with a main() function and a fact() function



Using Python as example,
write **fact(n)**
in Assembly

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

```
;;; def main():
;;;     n = 5
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println

exit:
        mov     eax, 1
        mov     ebx, 0
        int     0x80
```

Question 1

- What is the largest value of *n* the assembly and recursive version of ***fact*** can compute?

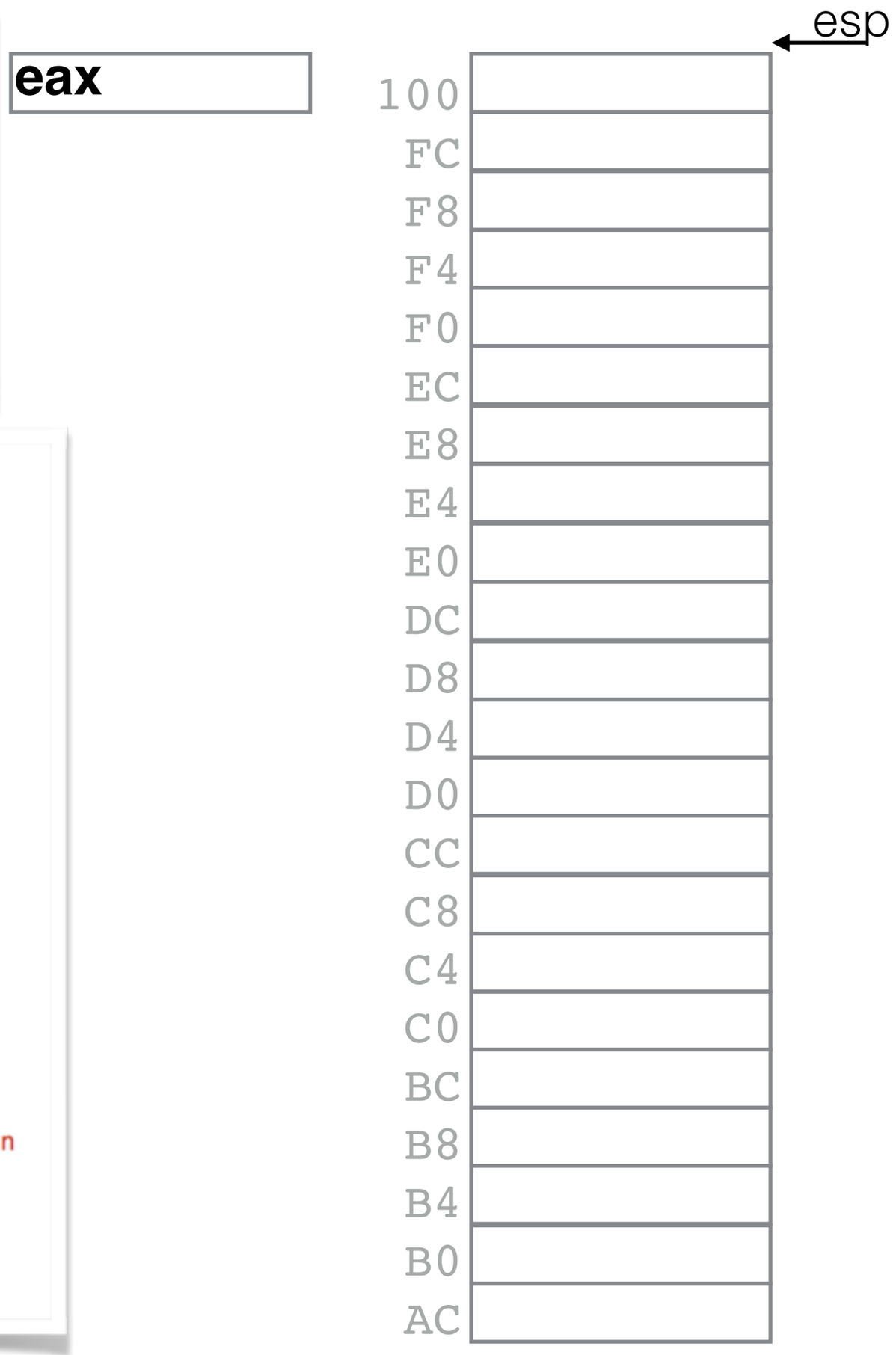
Note: there are several possible good answers…

We stopped here last time…

Single-Stepping the Assembly Code Computing *Fact(3)*

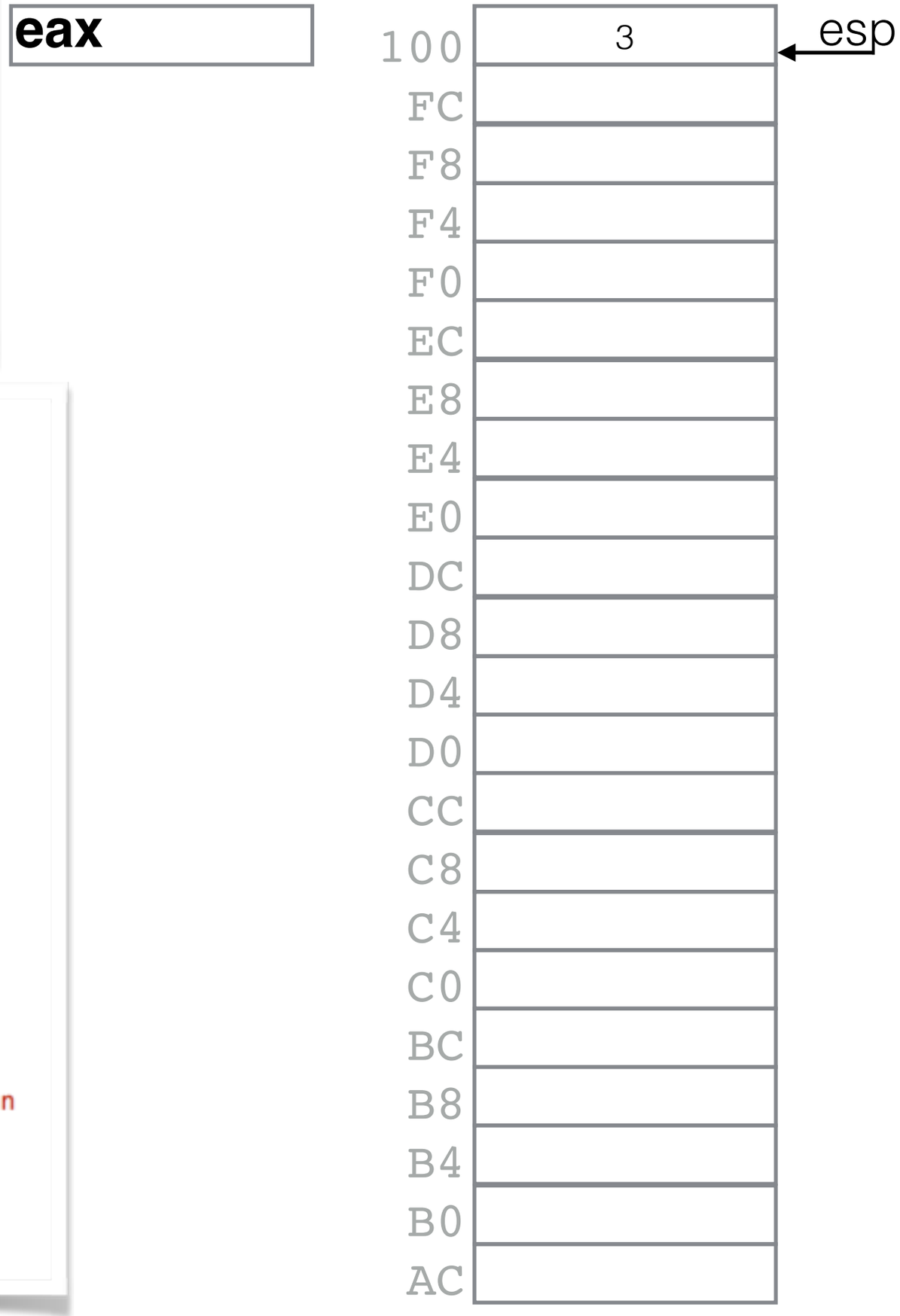
```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```



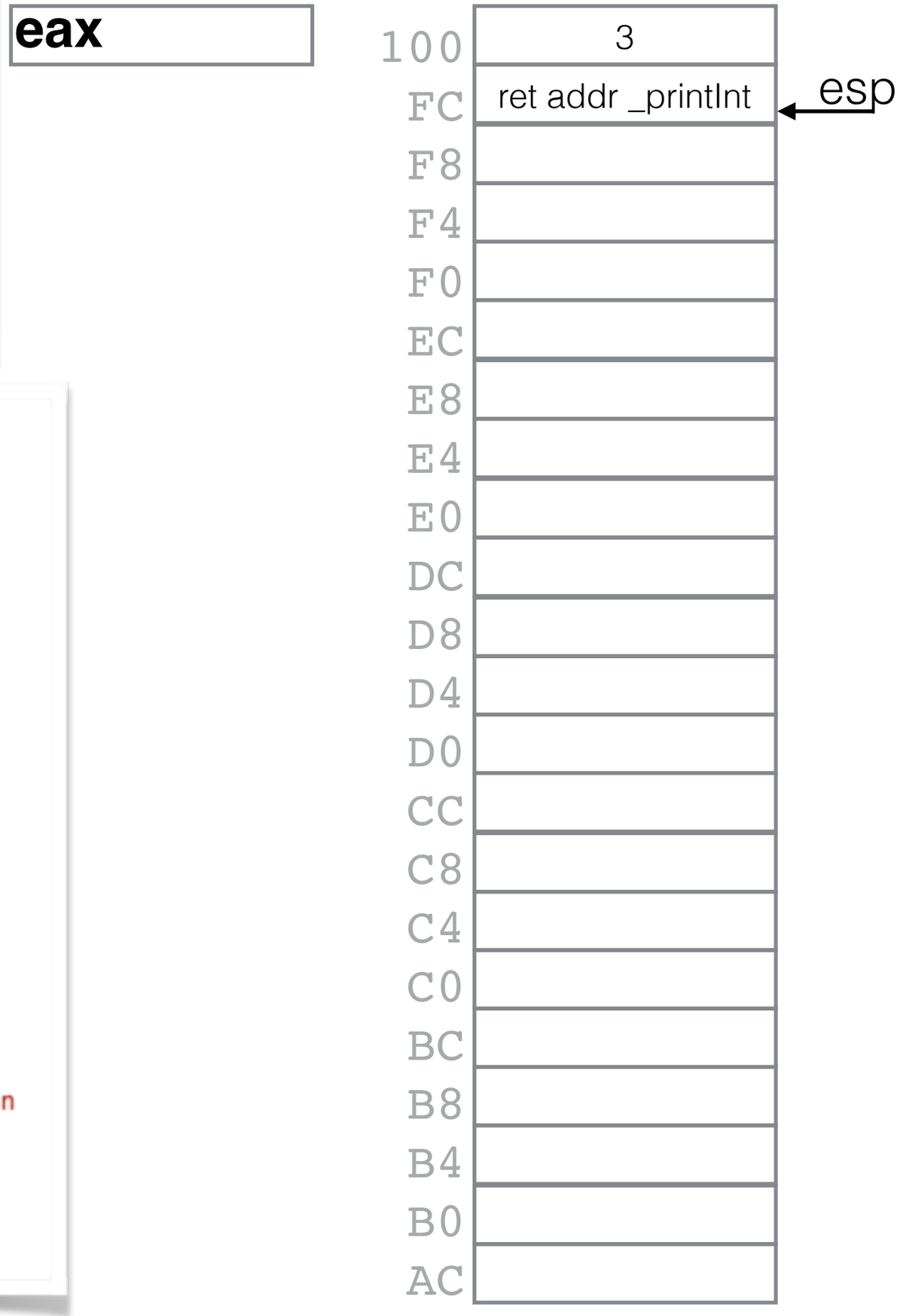
```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```



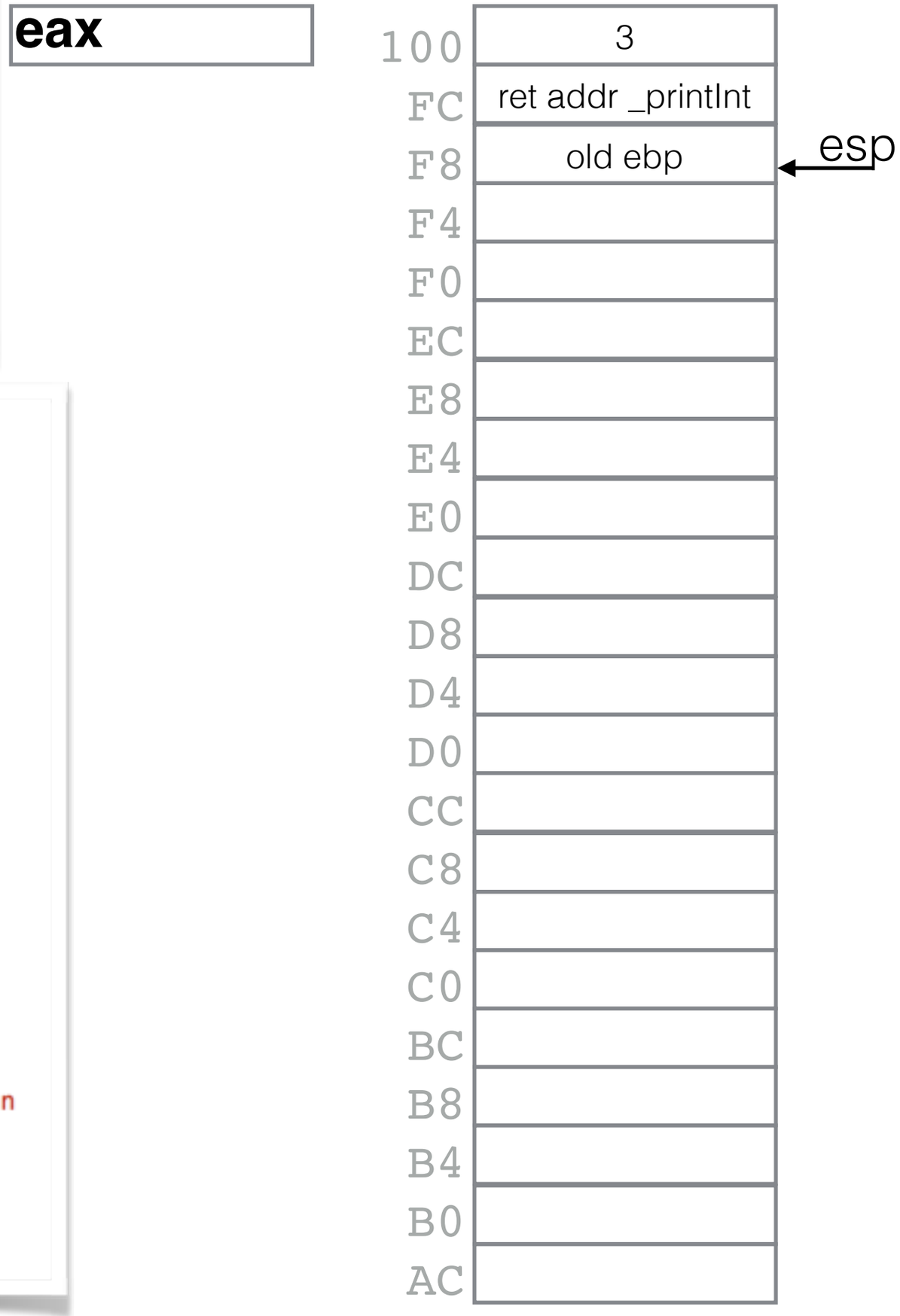
```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```



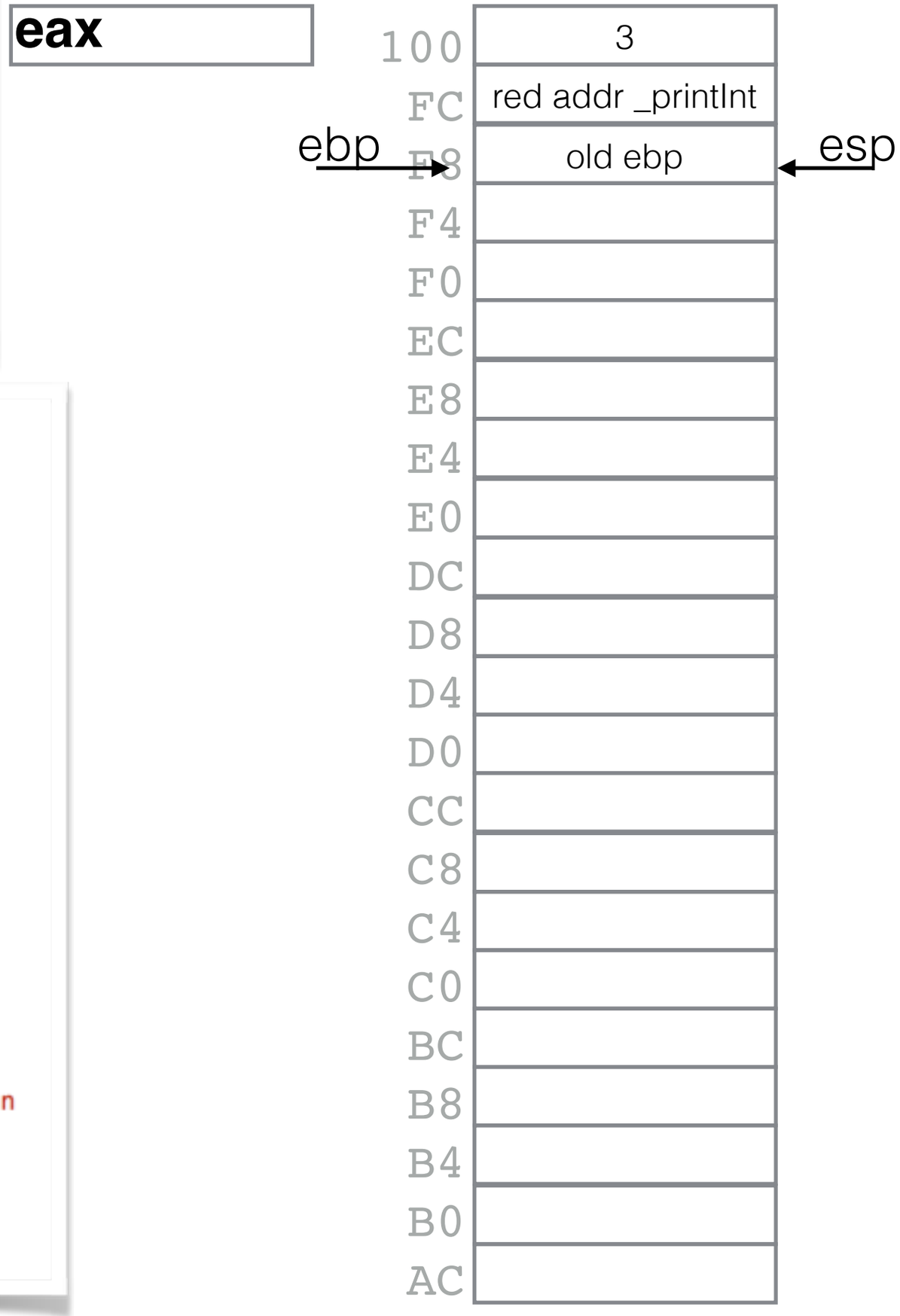
```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```



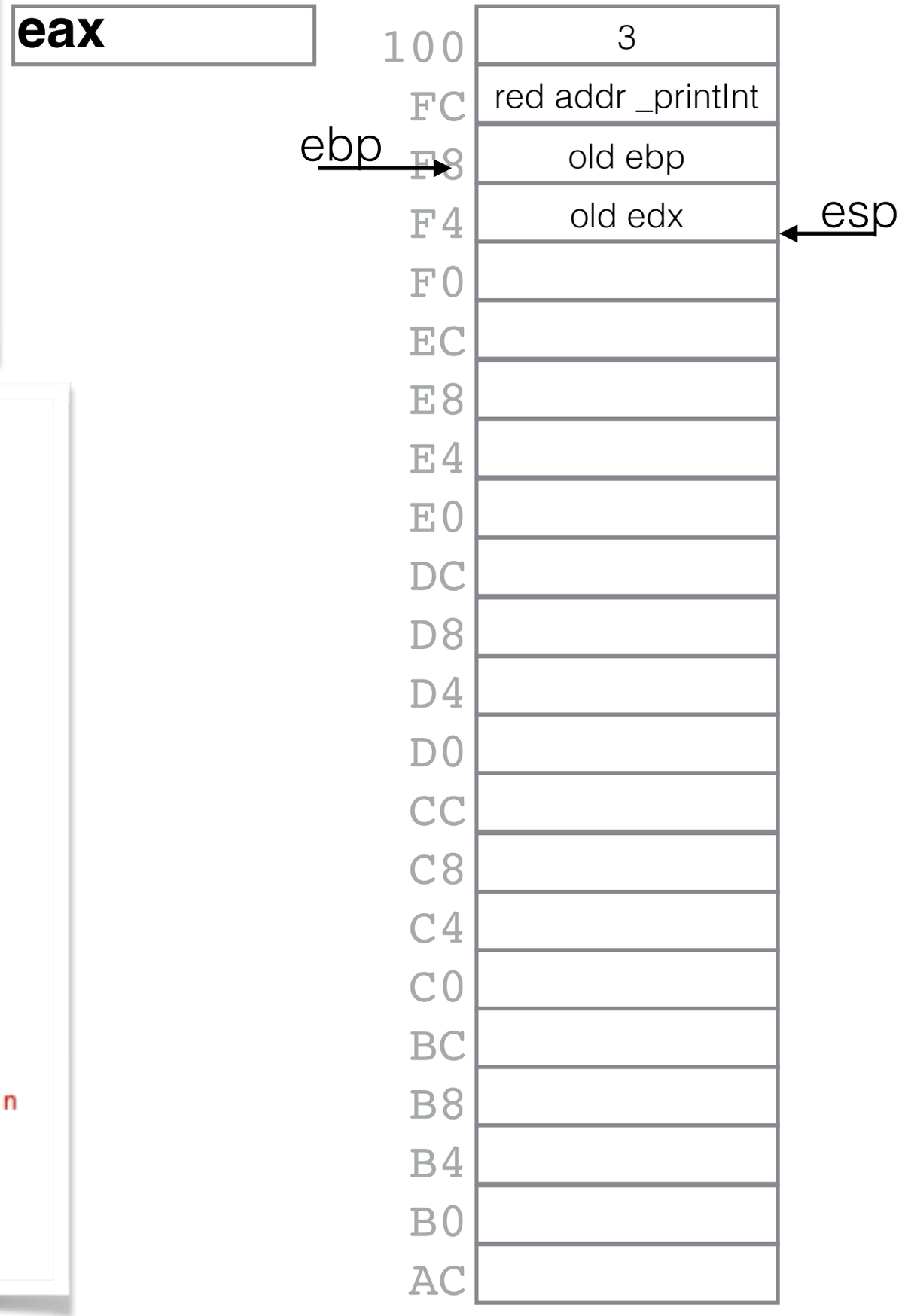
```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```



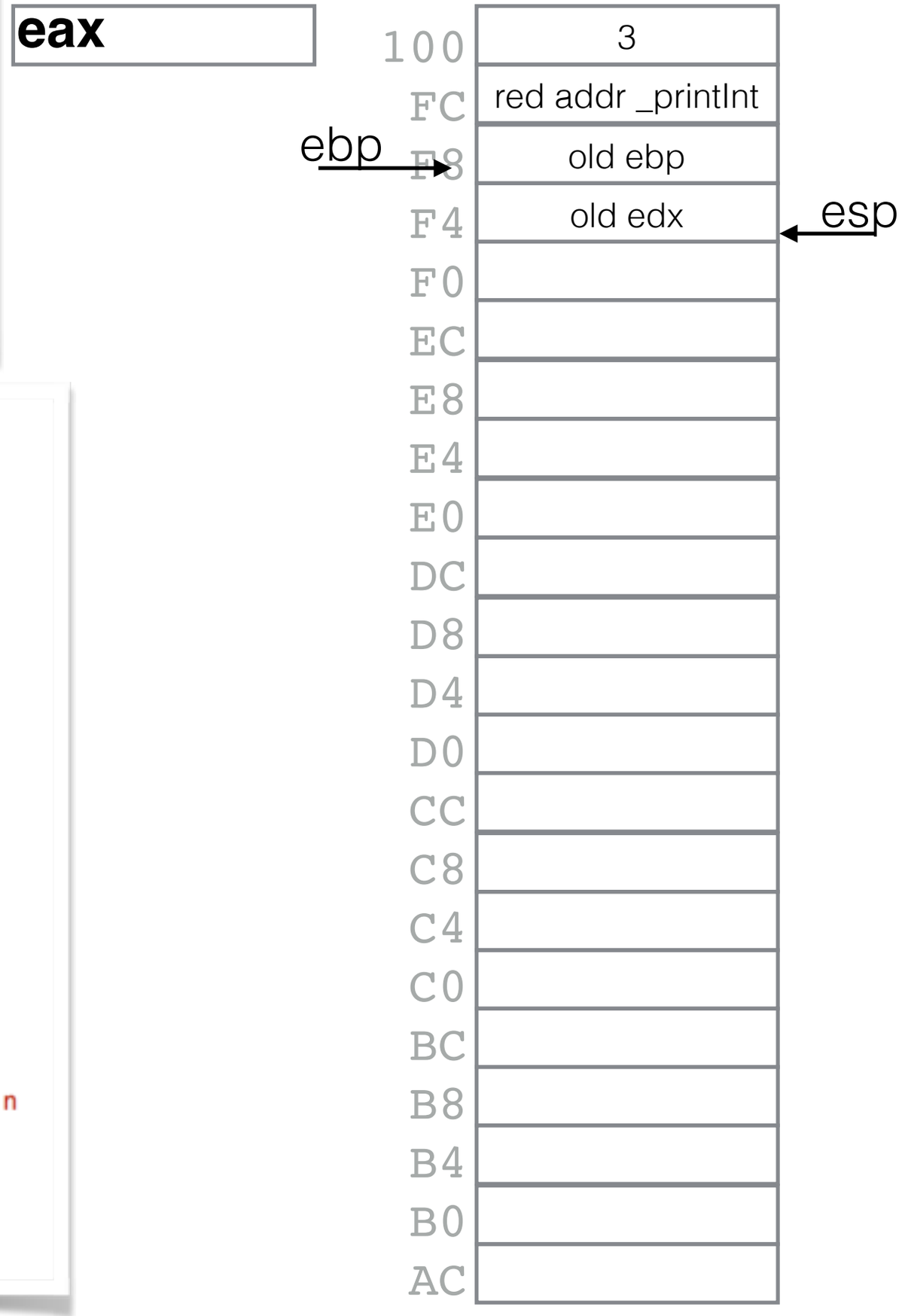
```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```



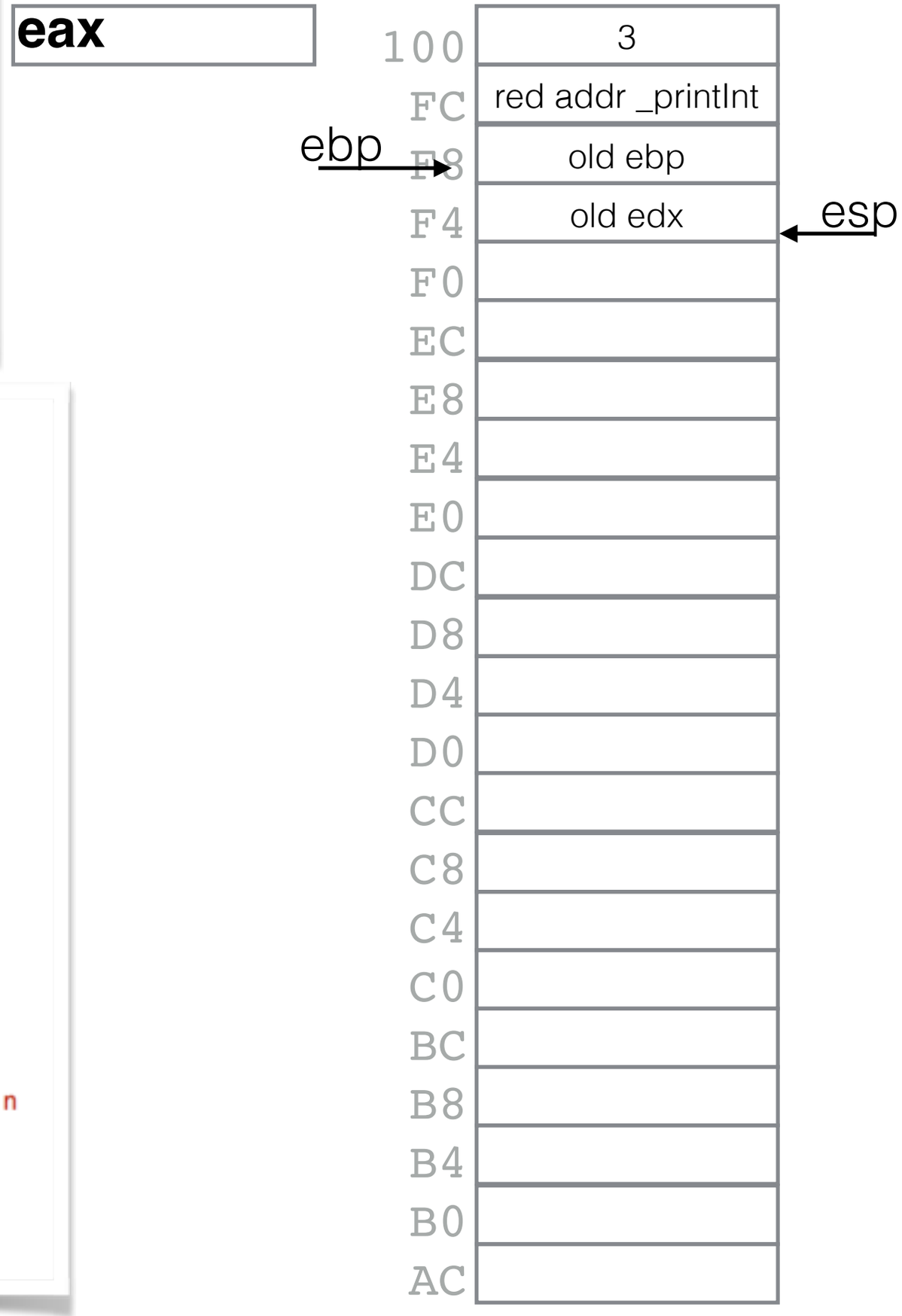
```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

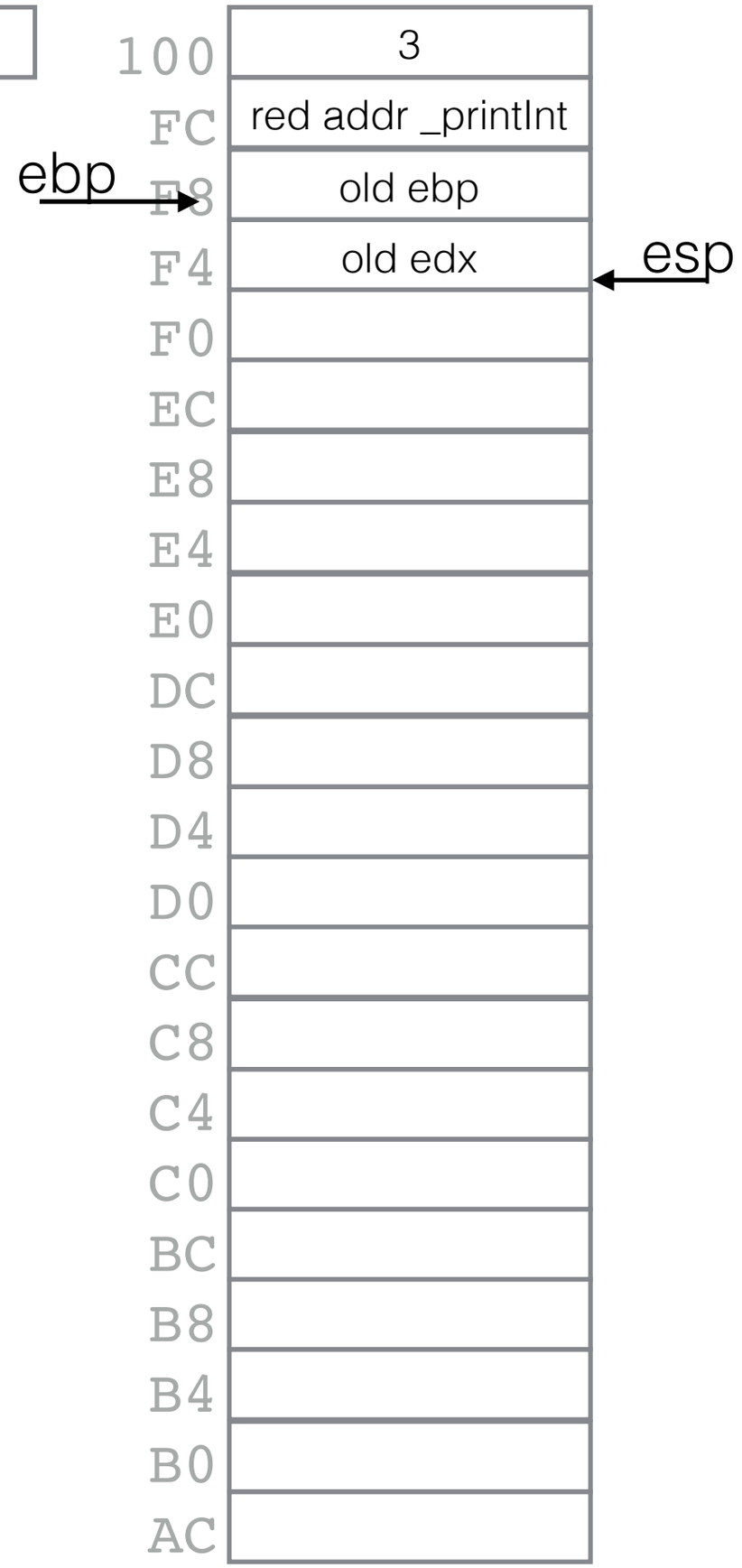
```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

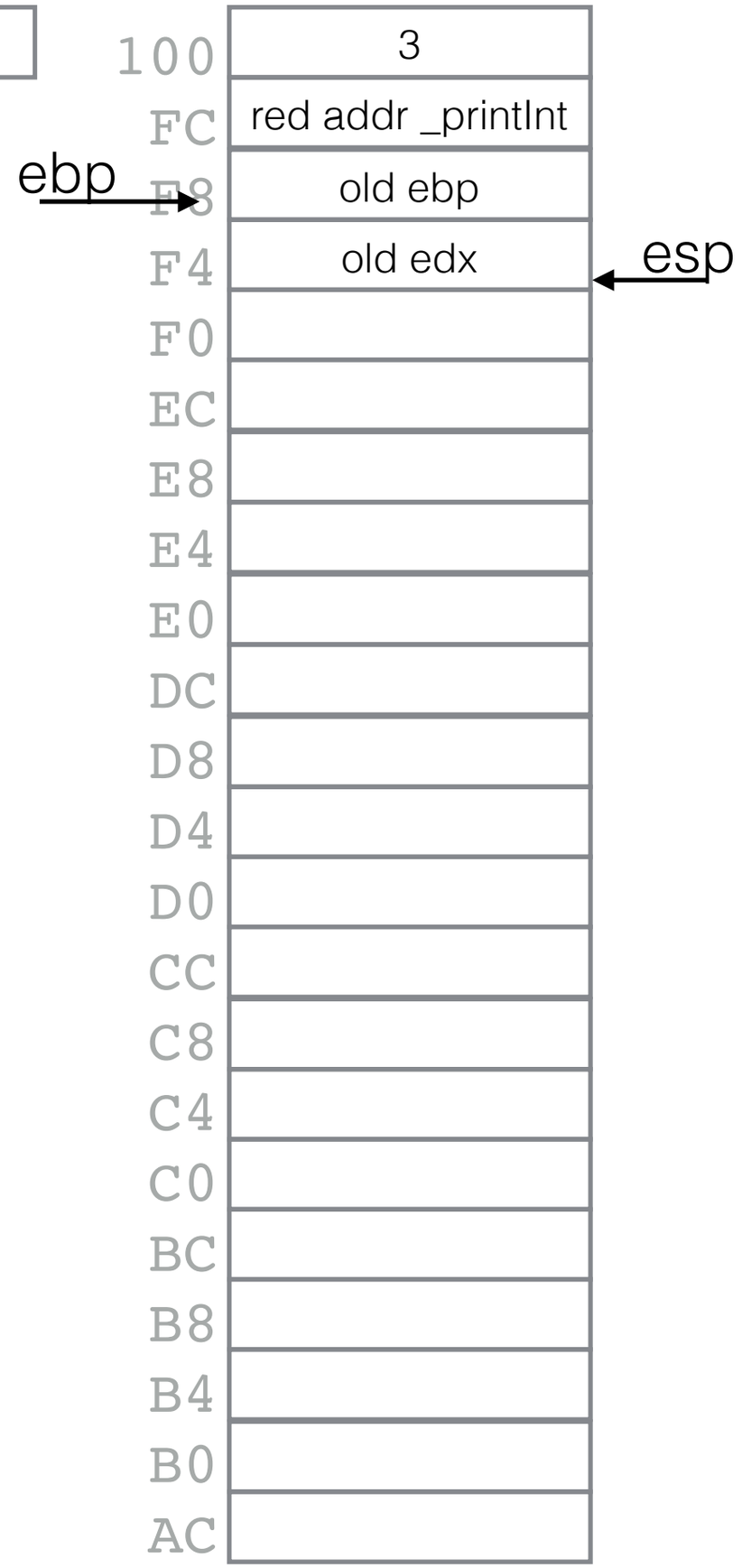
eax: 3



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

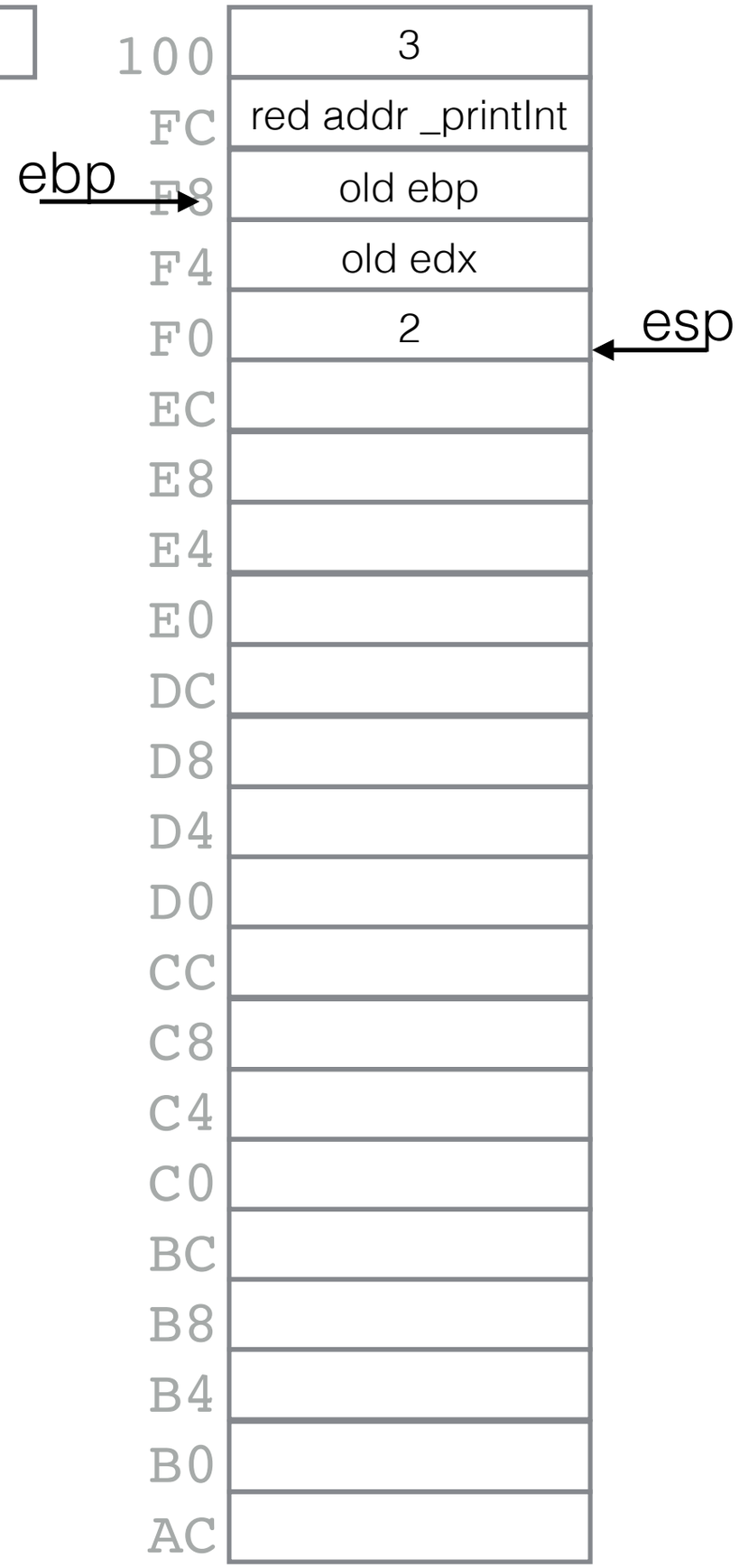
eax: 2



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

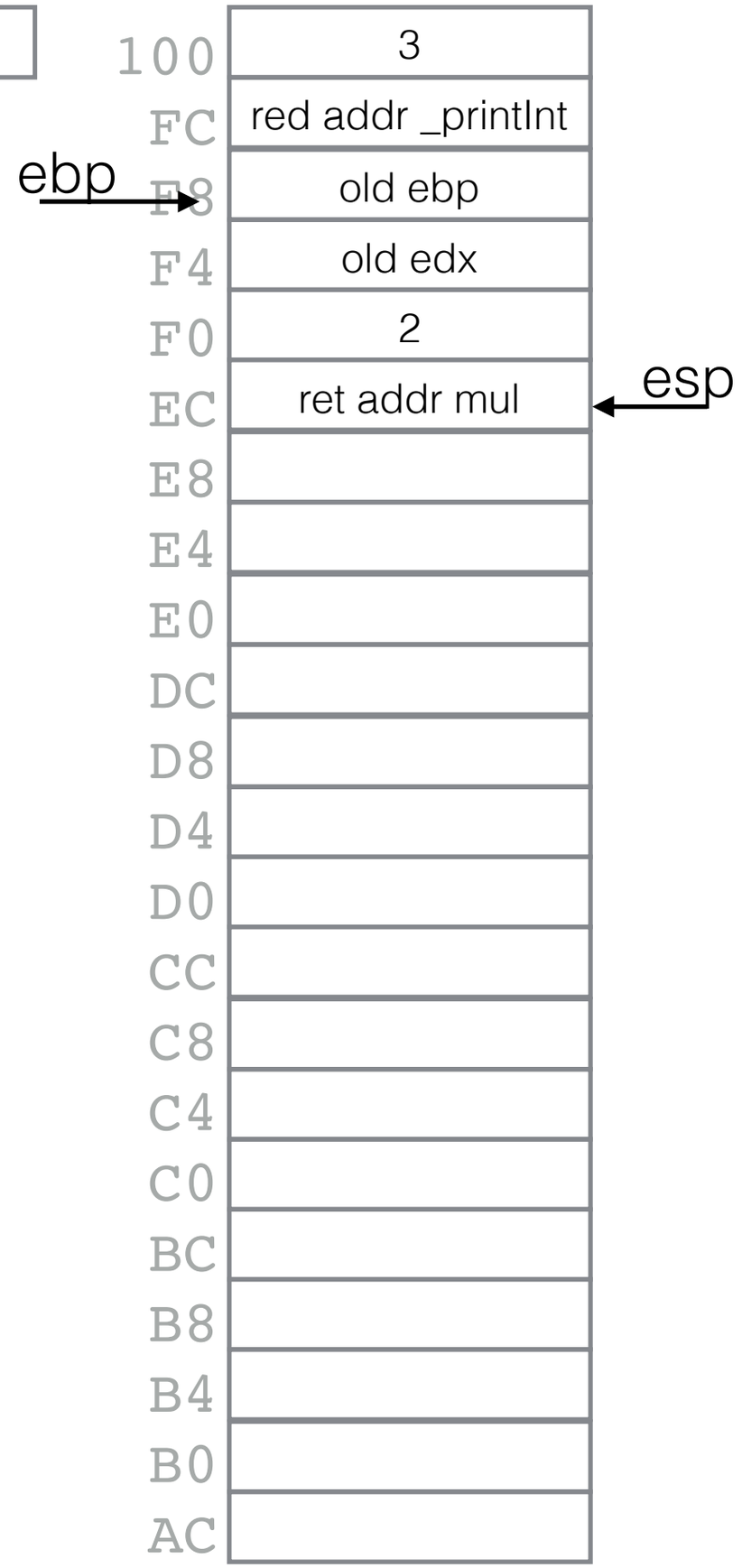
eax: 2



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

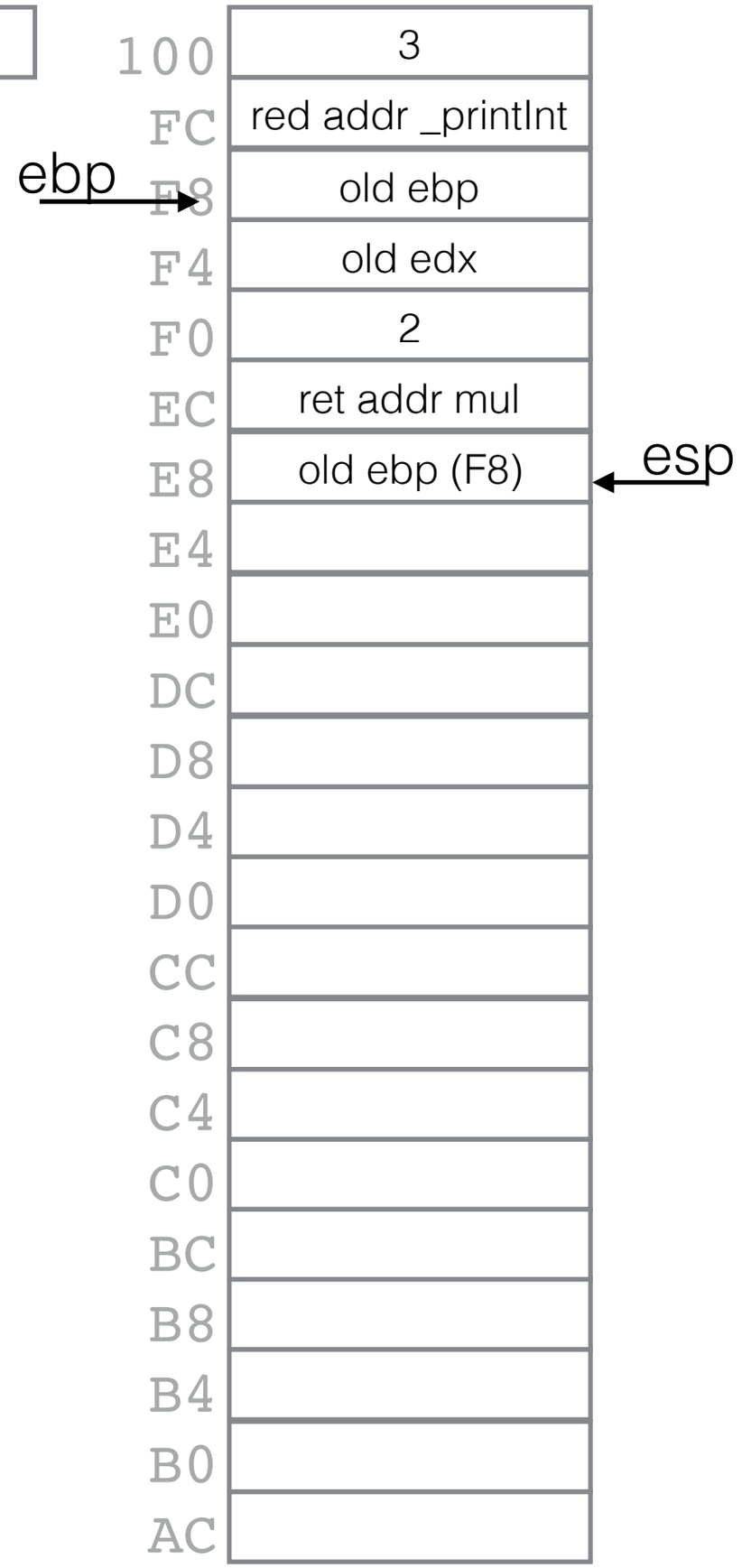
eax: 2



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

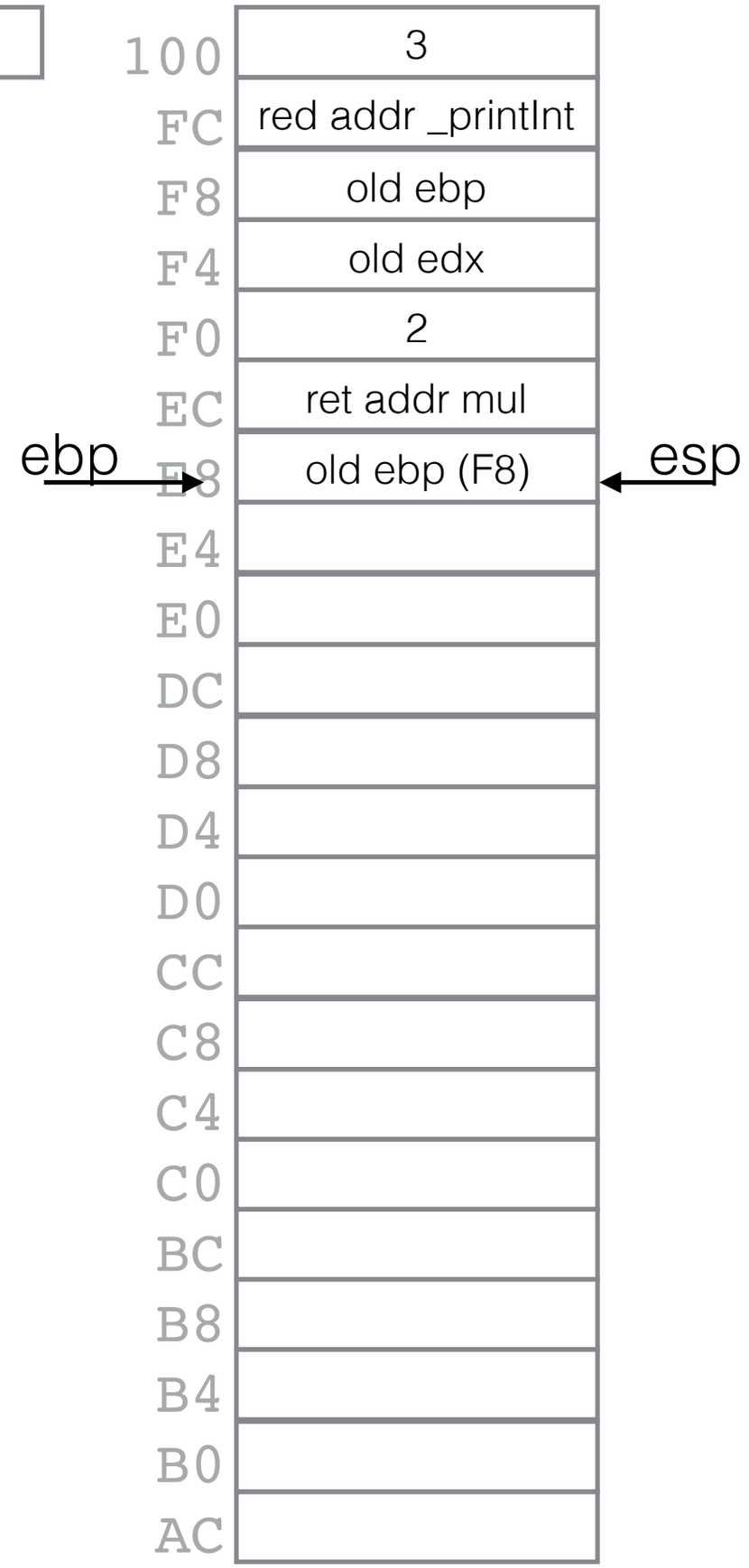
eax: 2



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

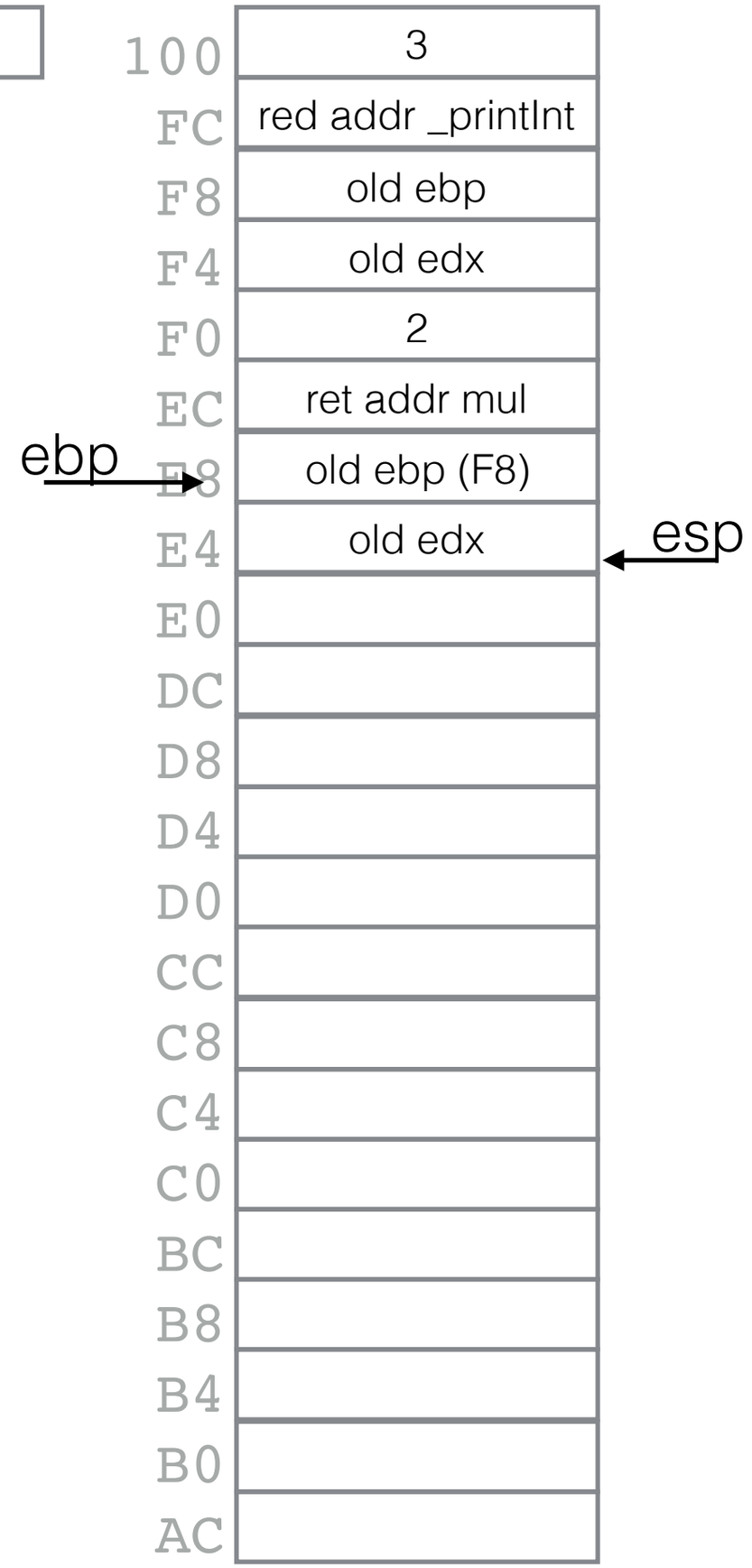
eax: 2



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

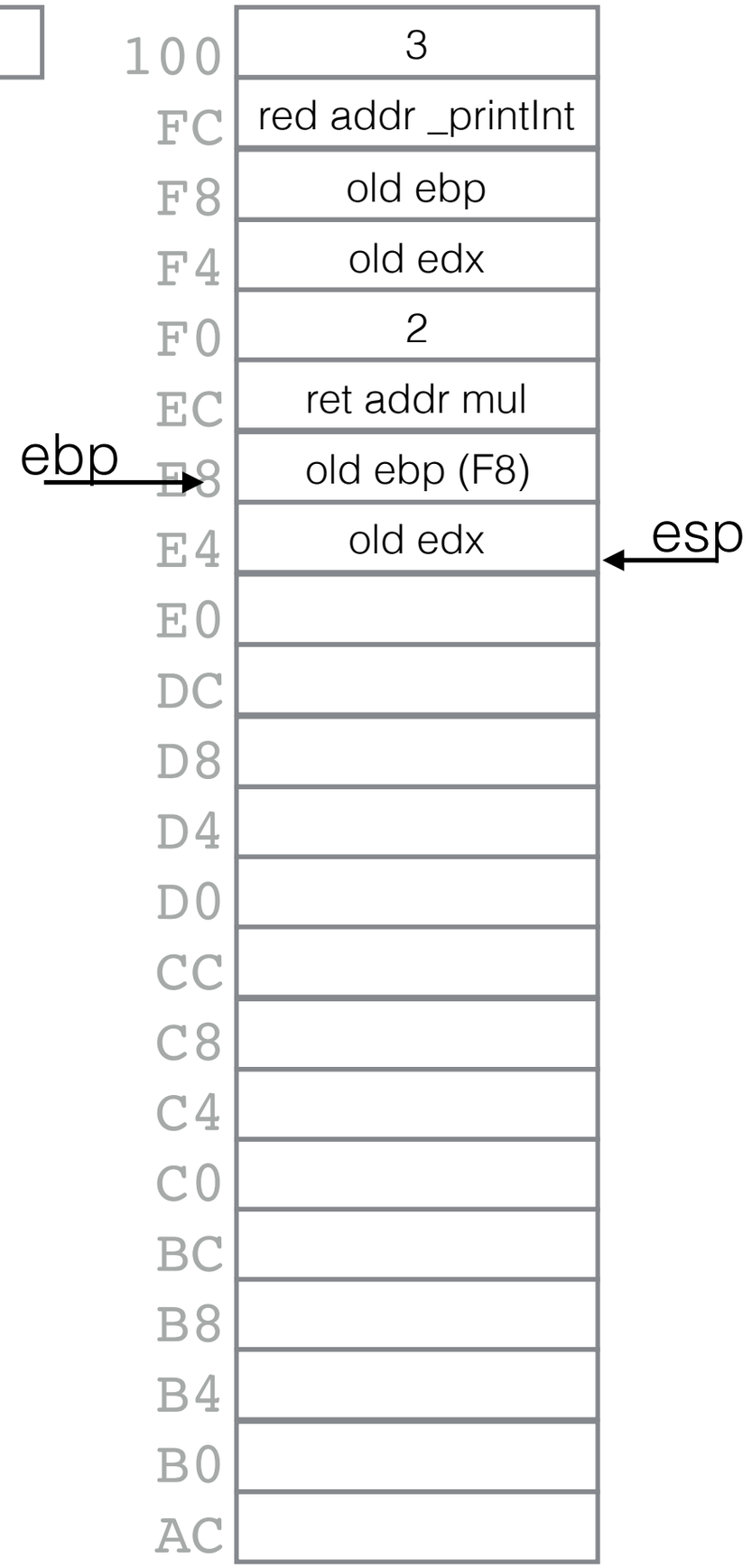
eax: 2



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

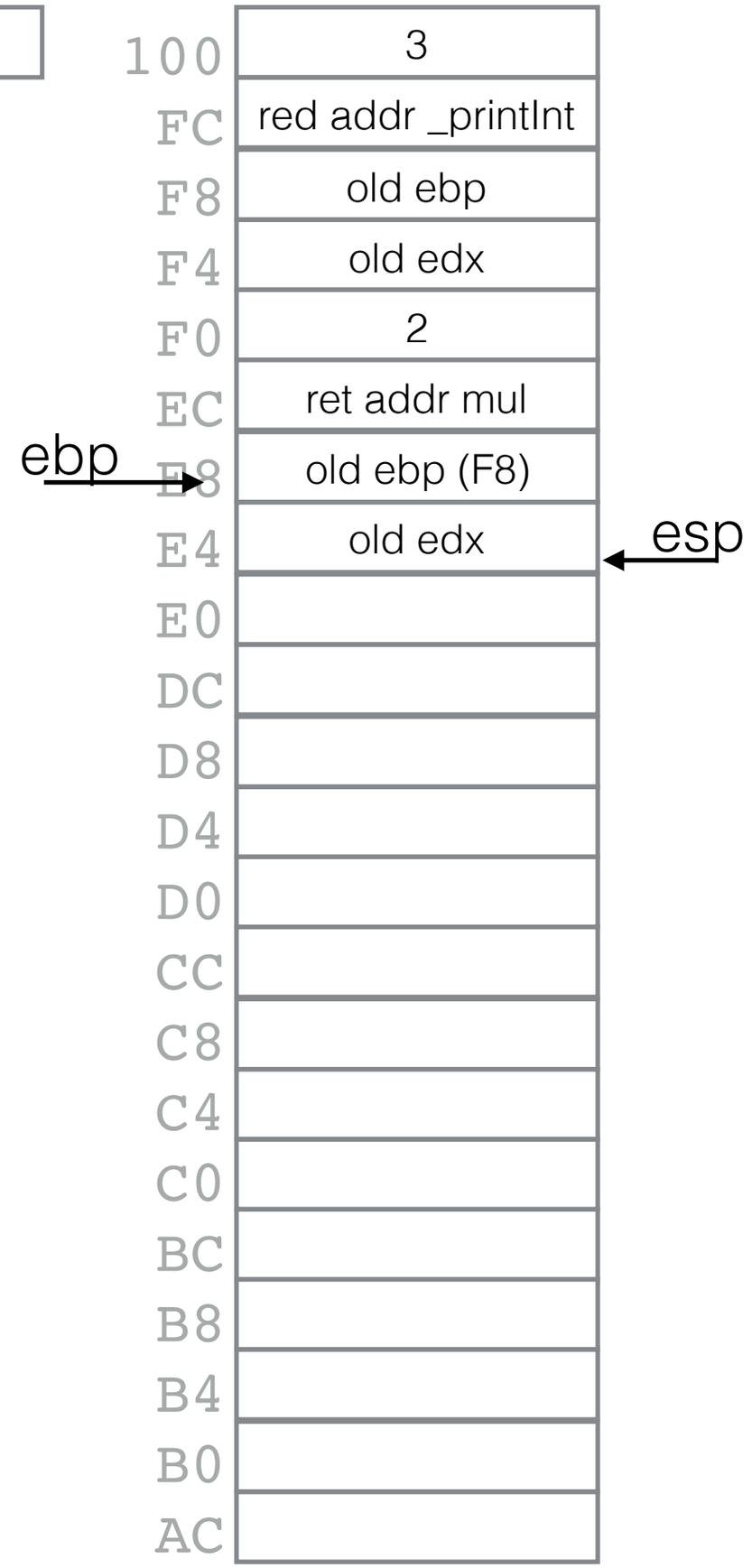
eax: 2



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

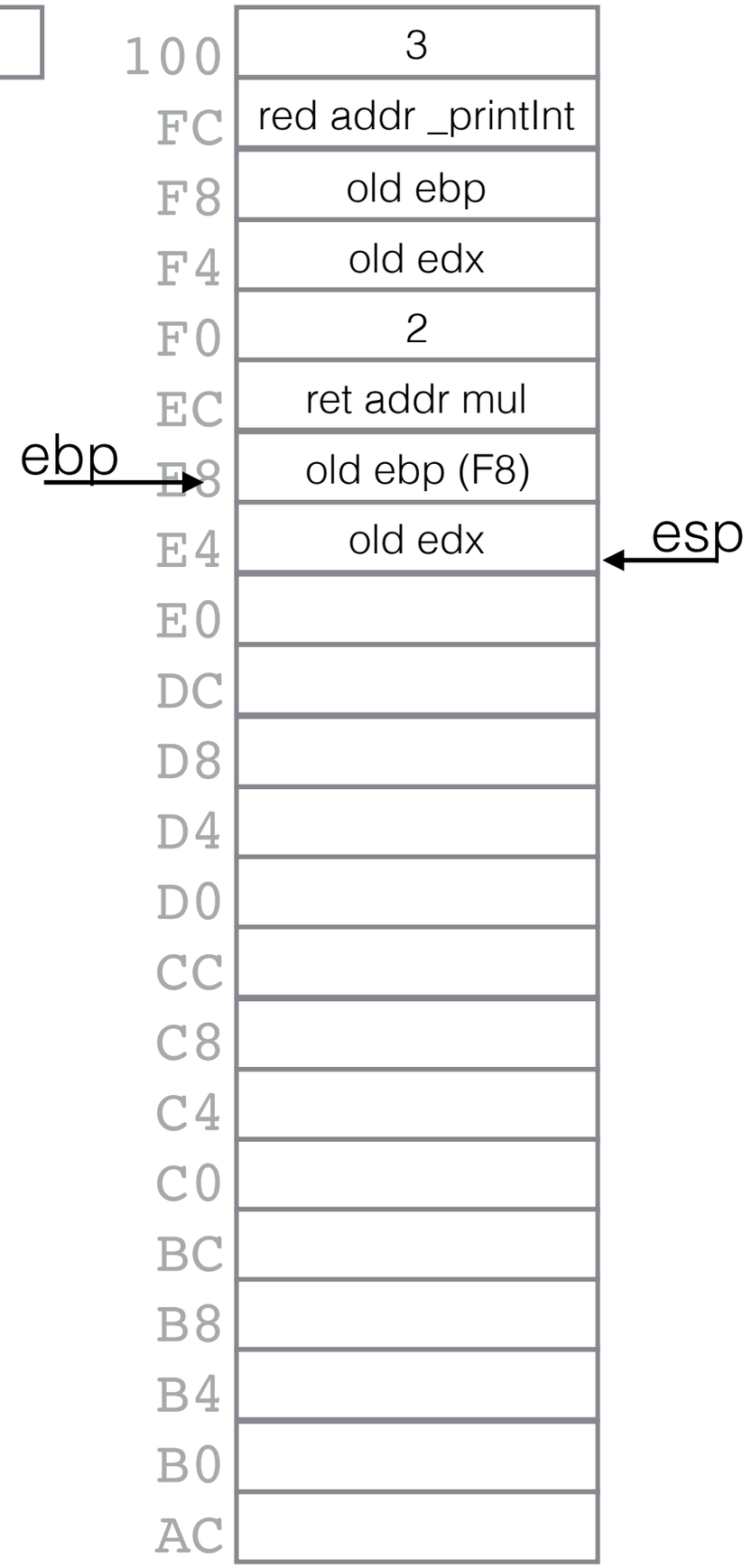
eax: 2



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

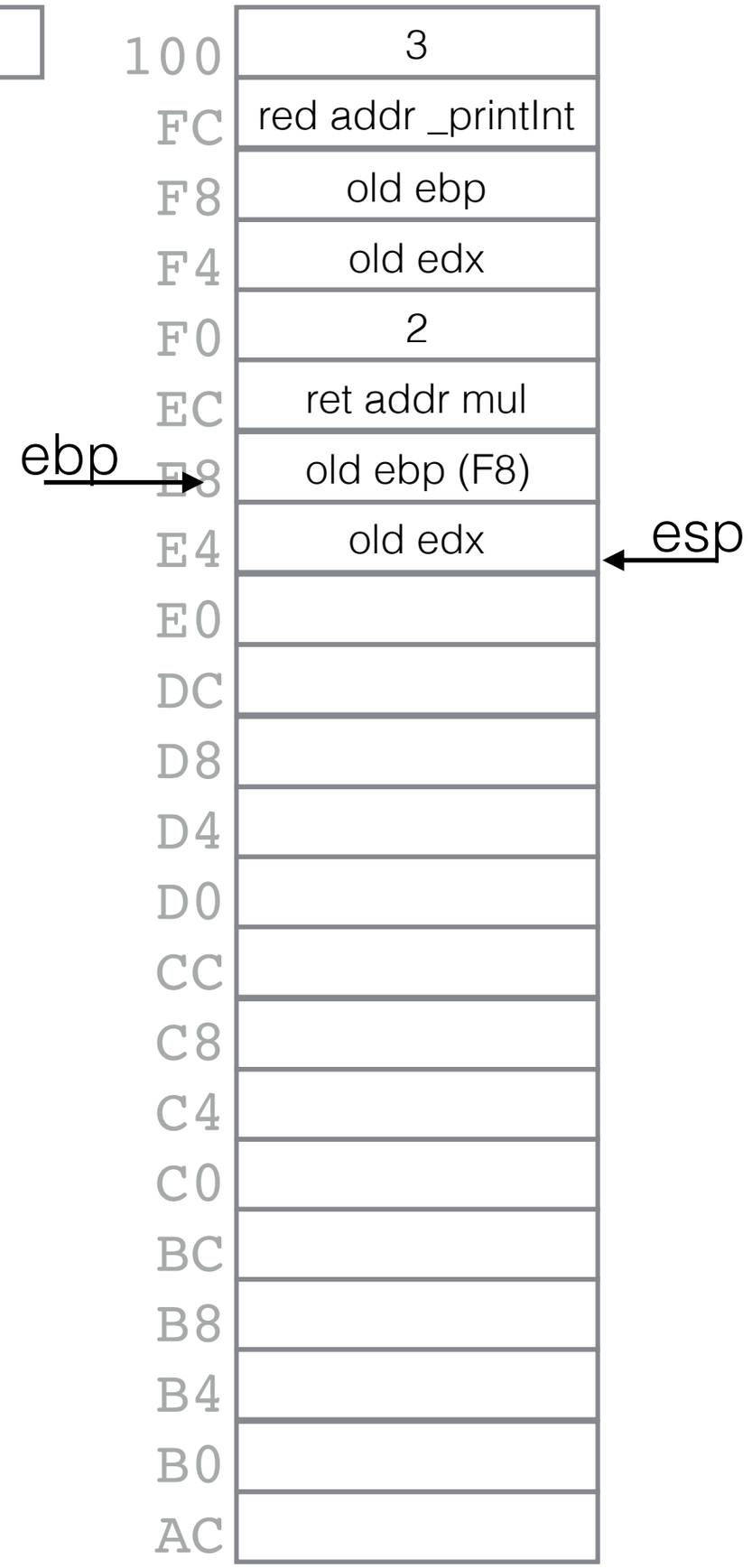
eax: 2



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

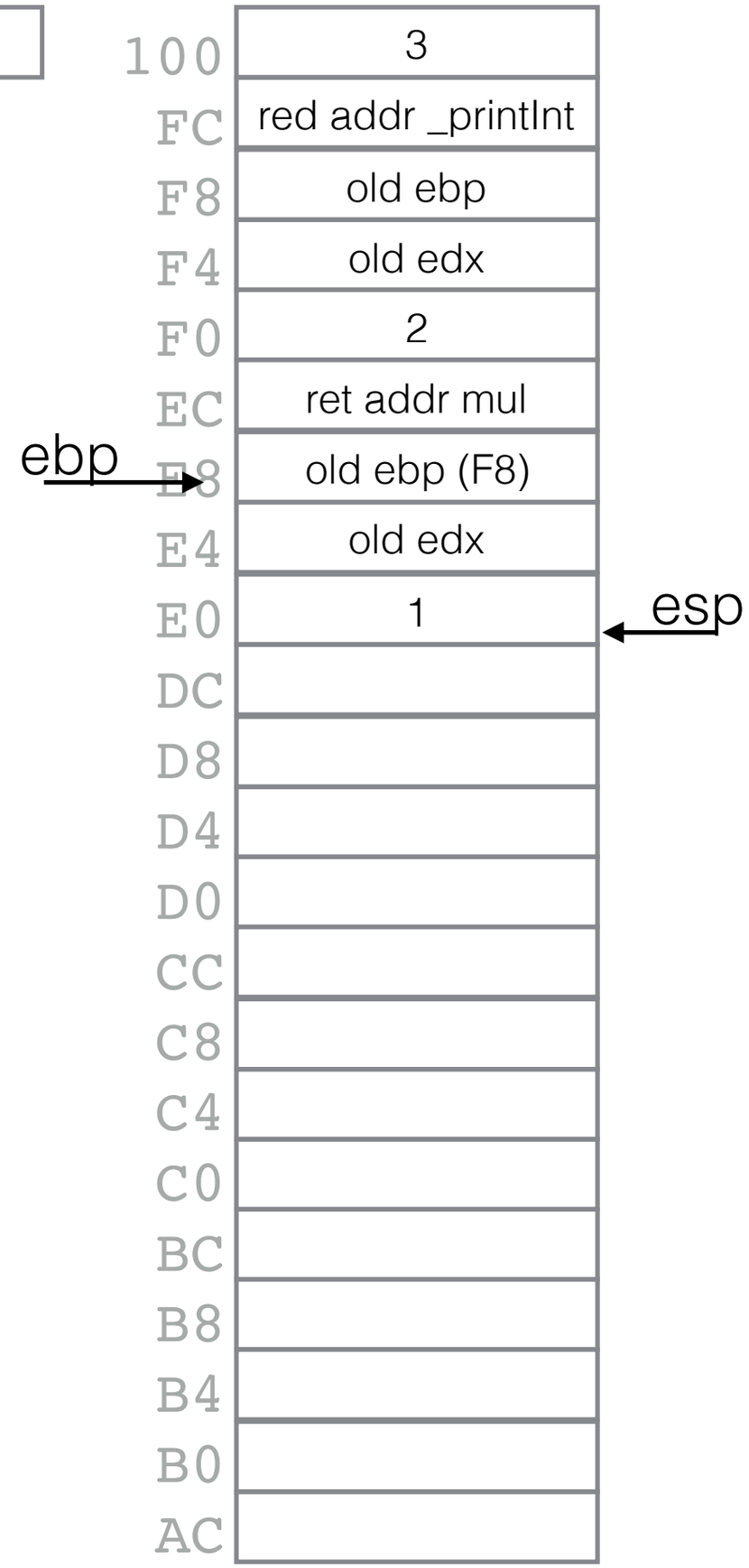
eax: 1



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

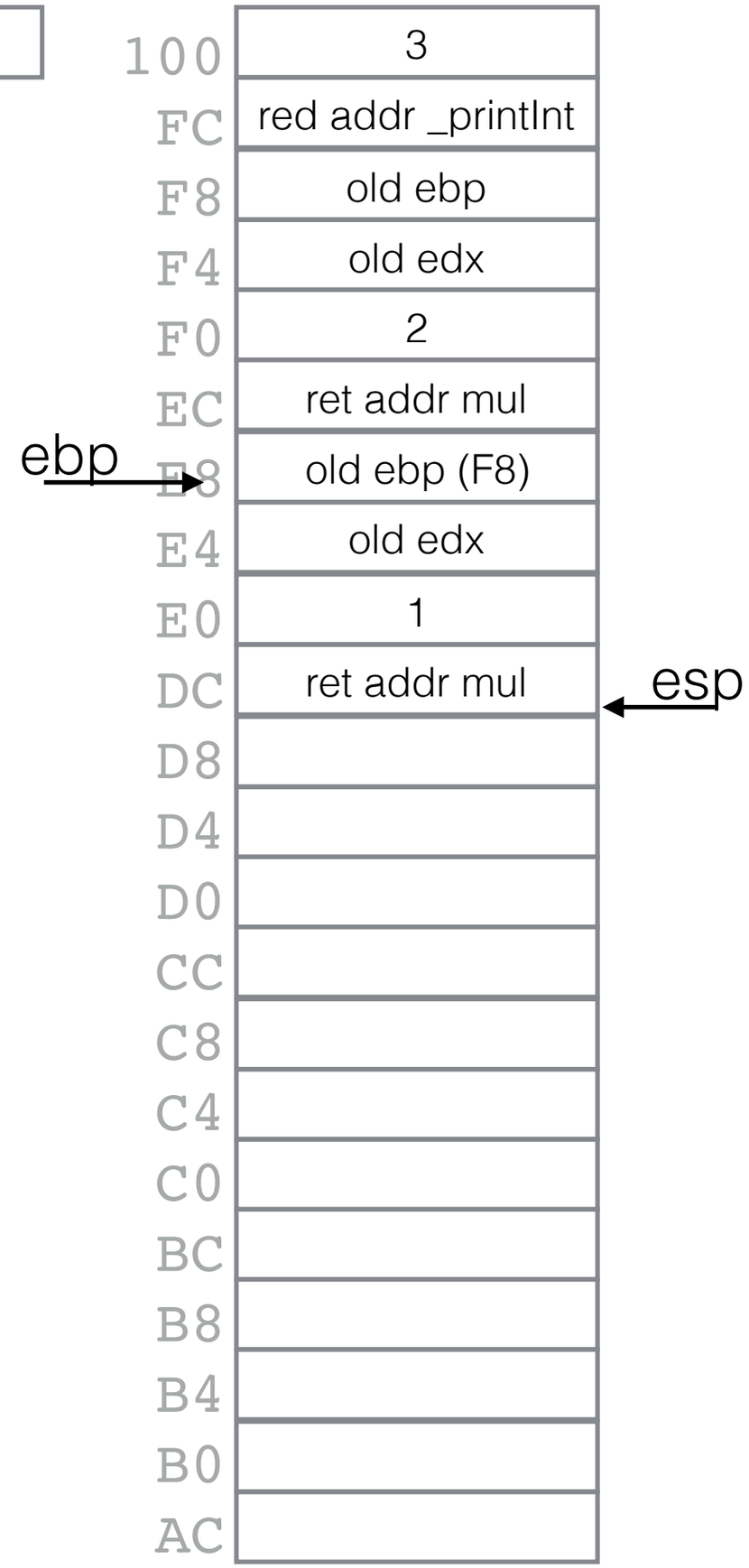
eax: 1



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

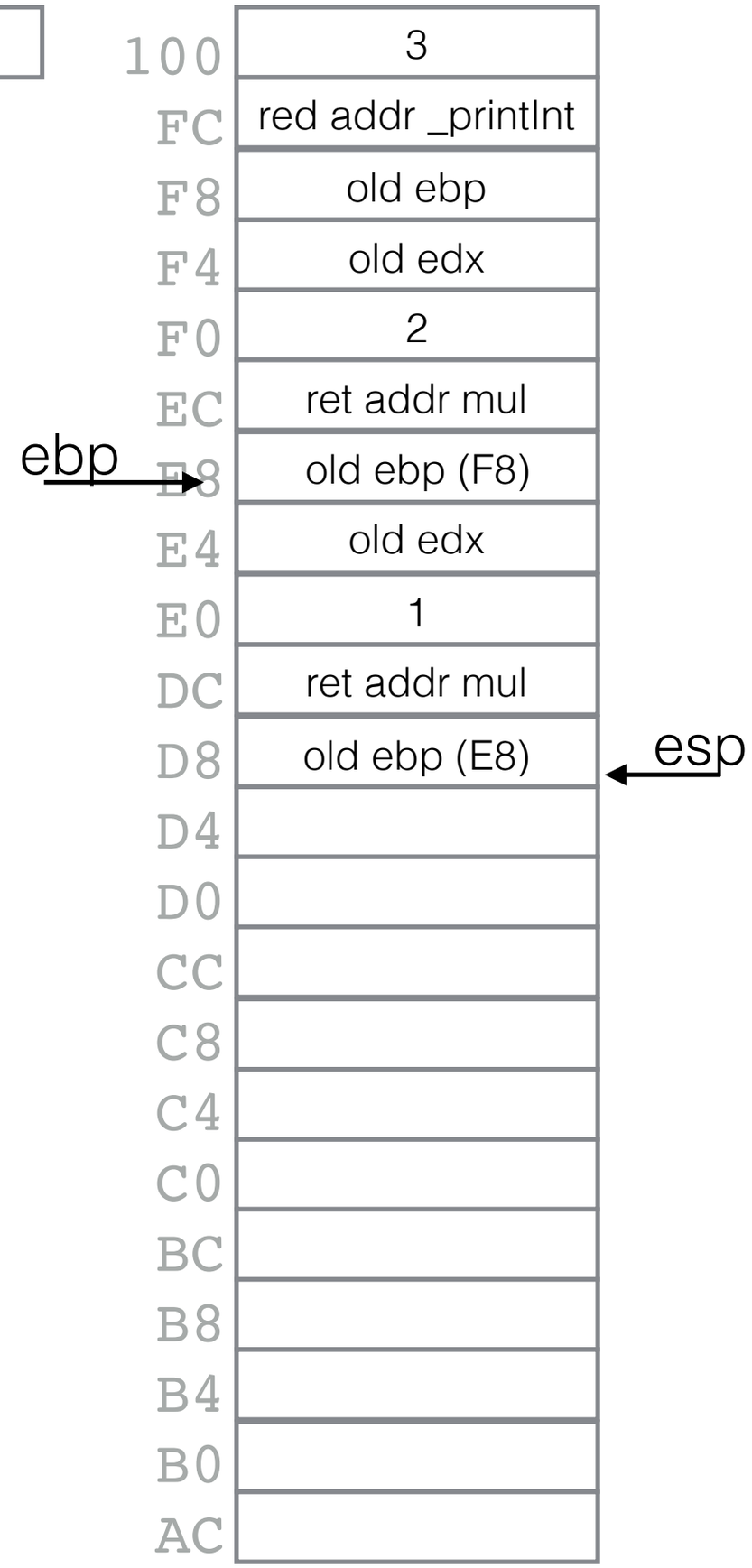
eax: 1



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

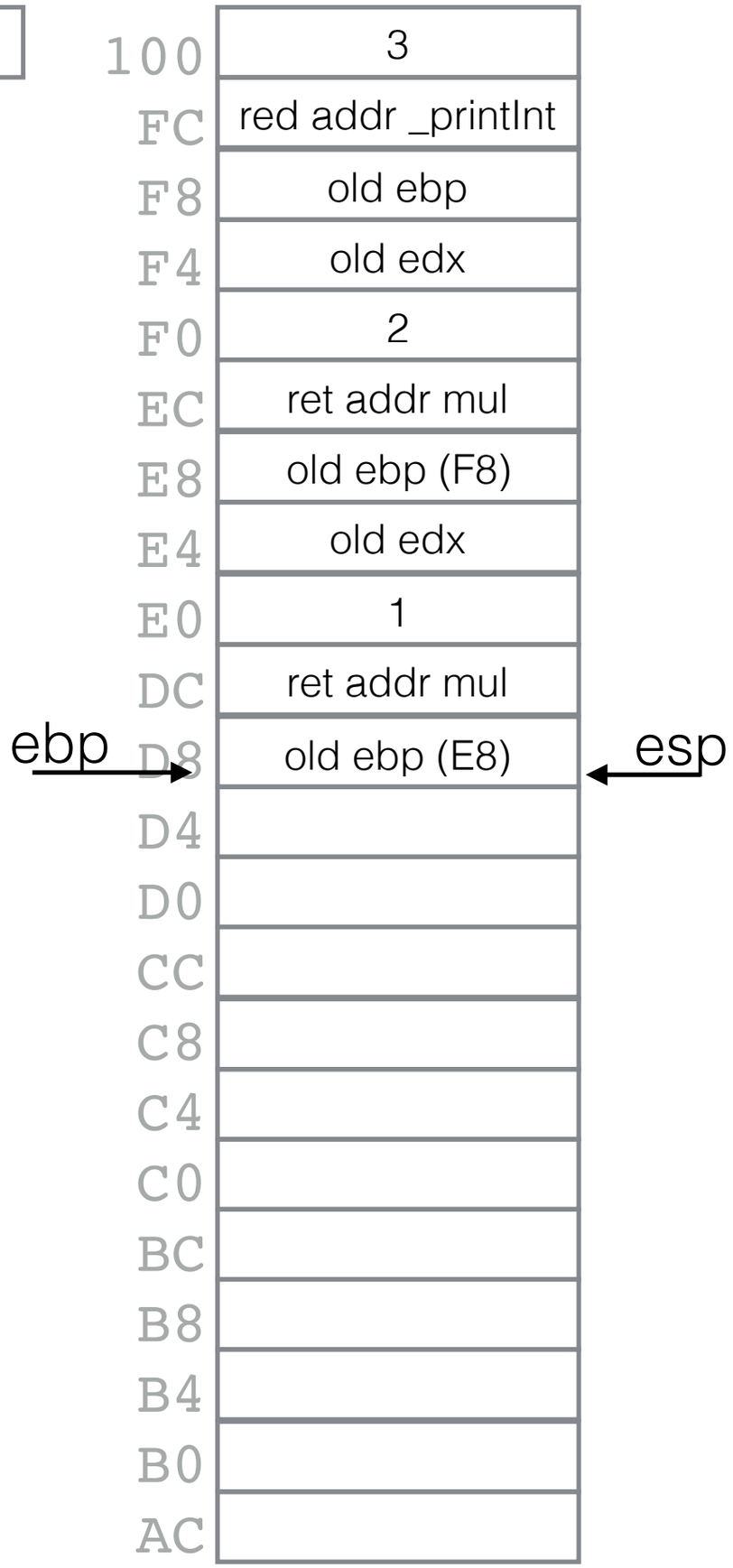
eax: 1



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

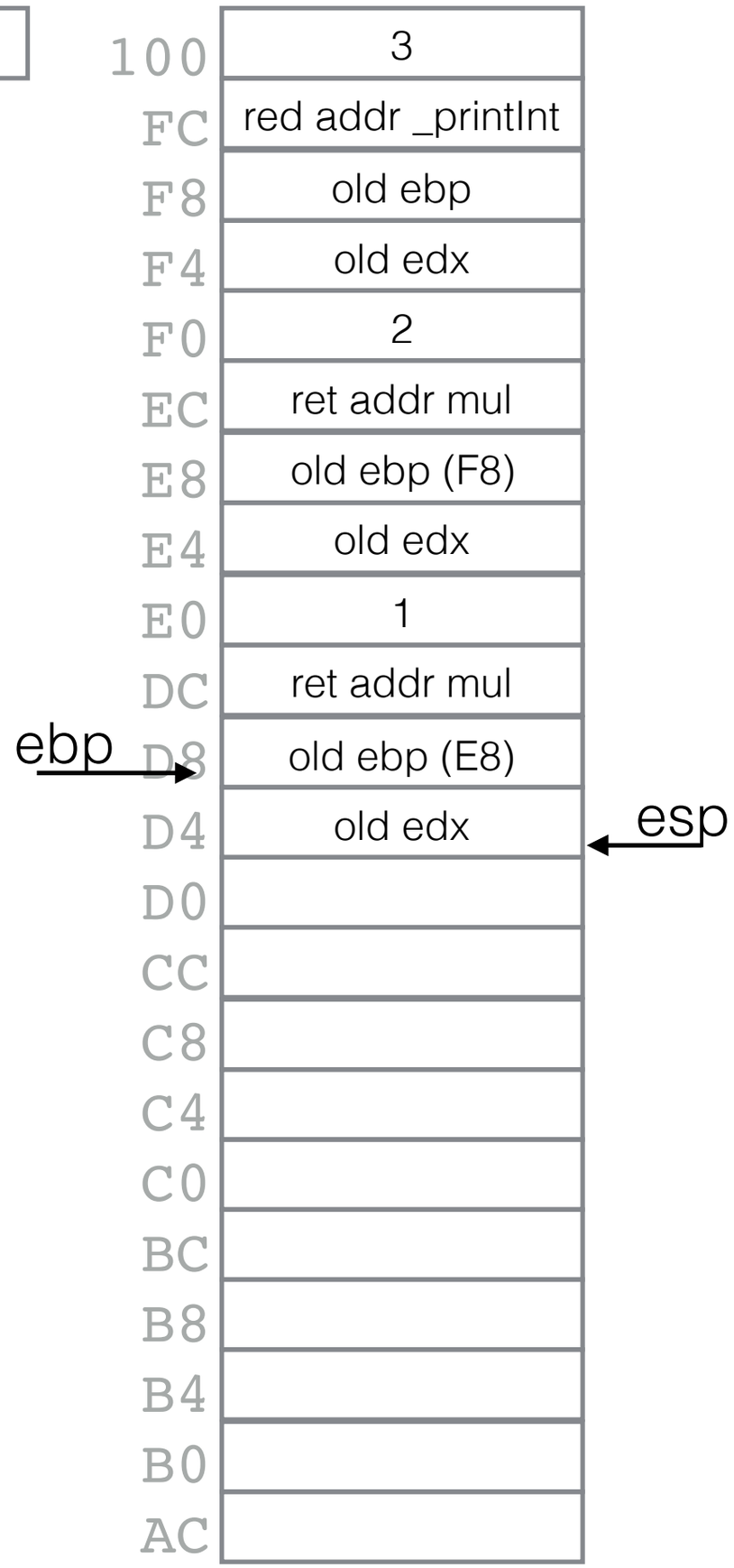
eax: 1



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

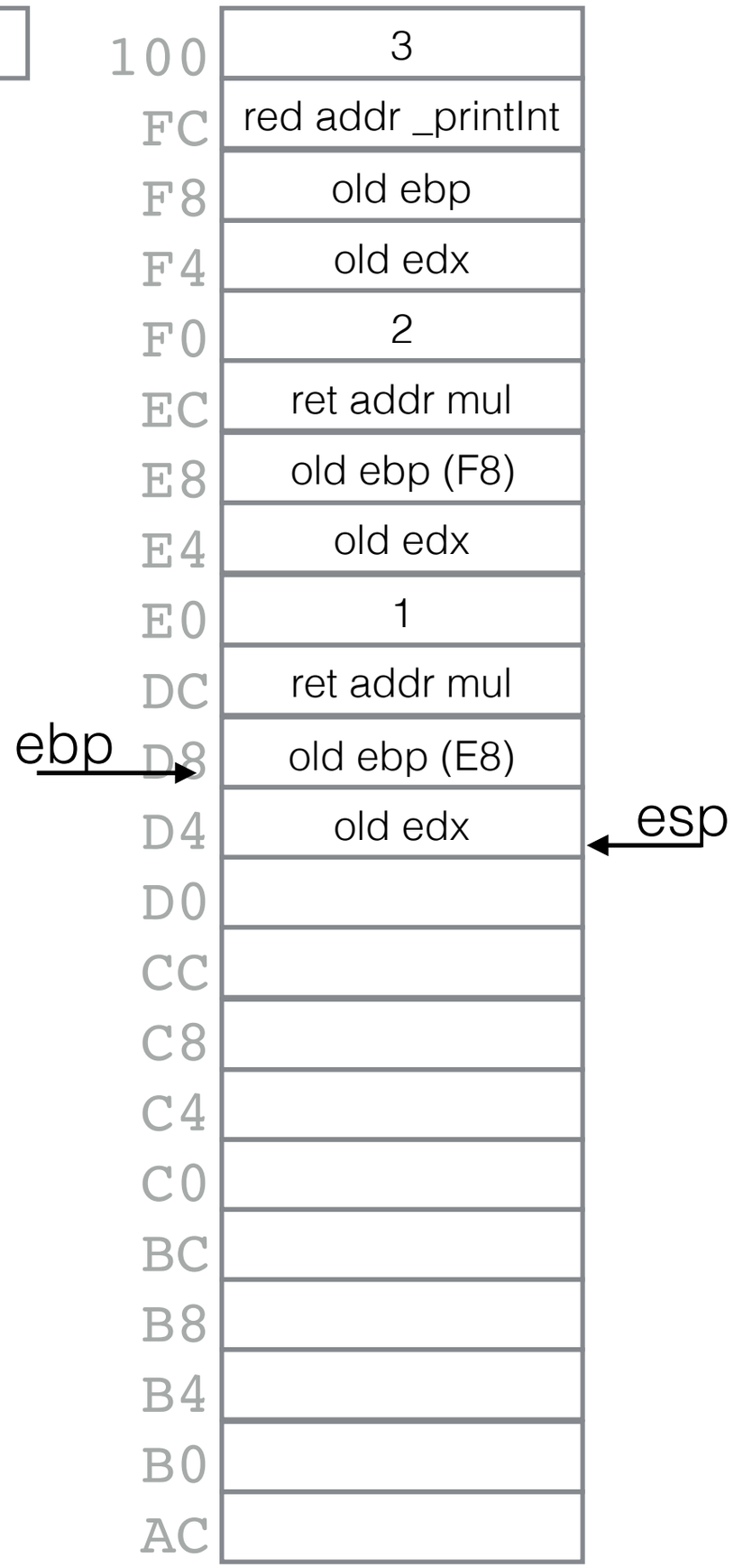
eax: 1



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

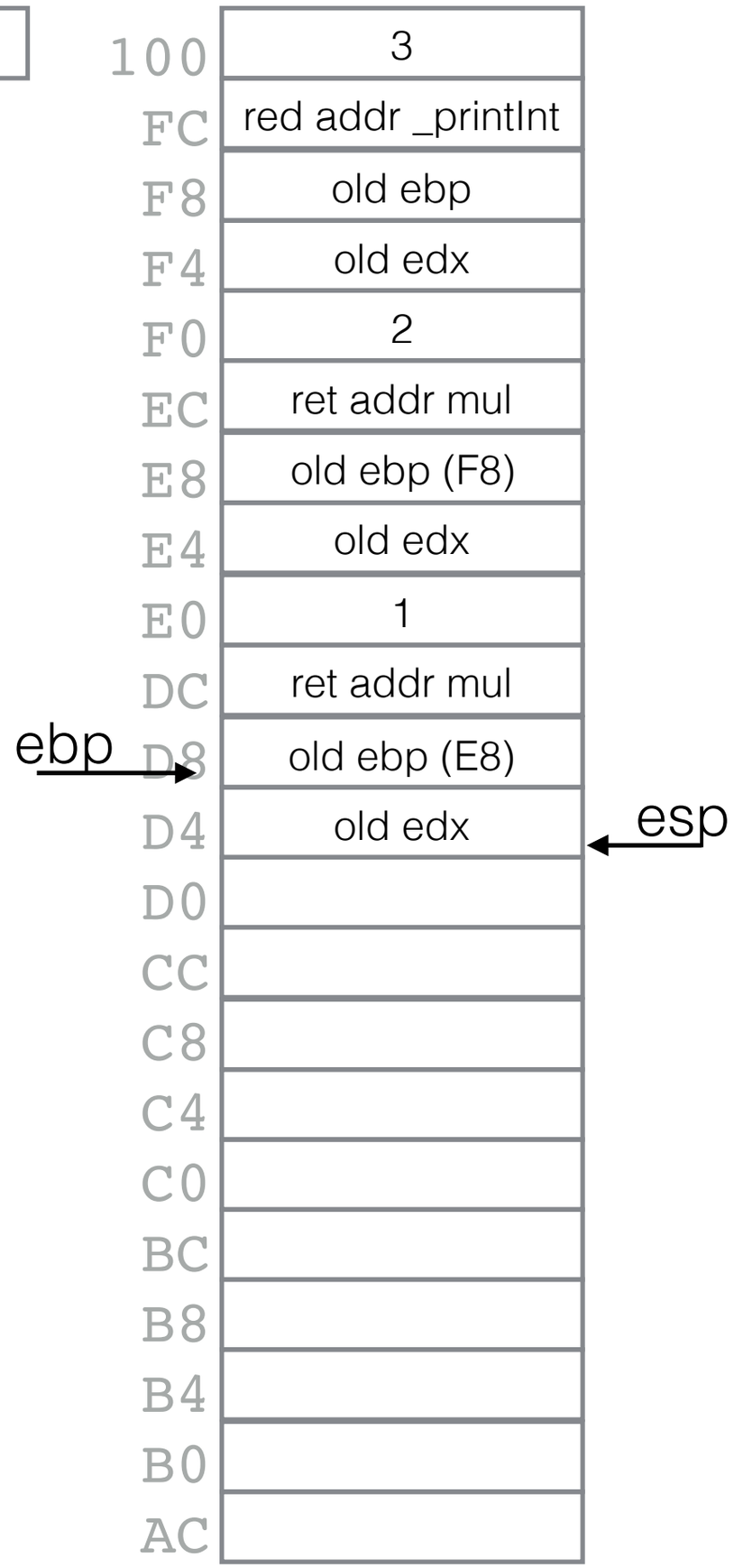
eax: 1



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

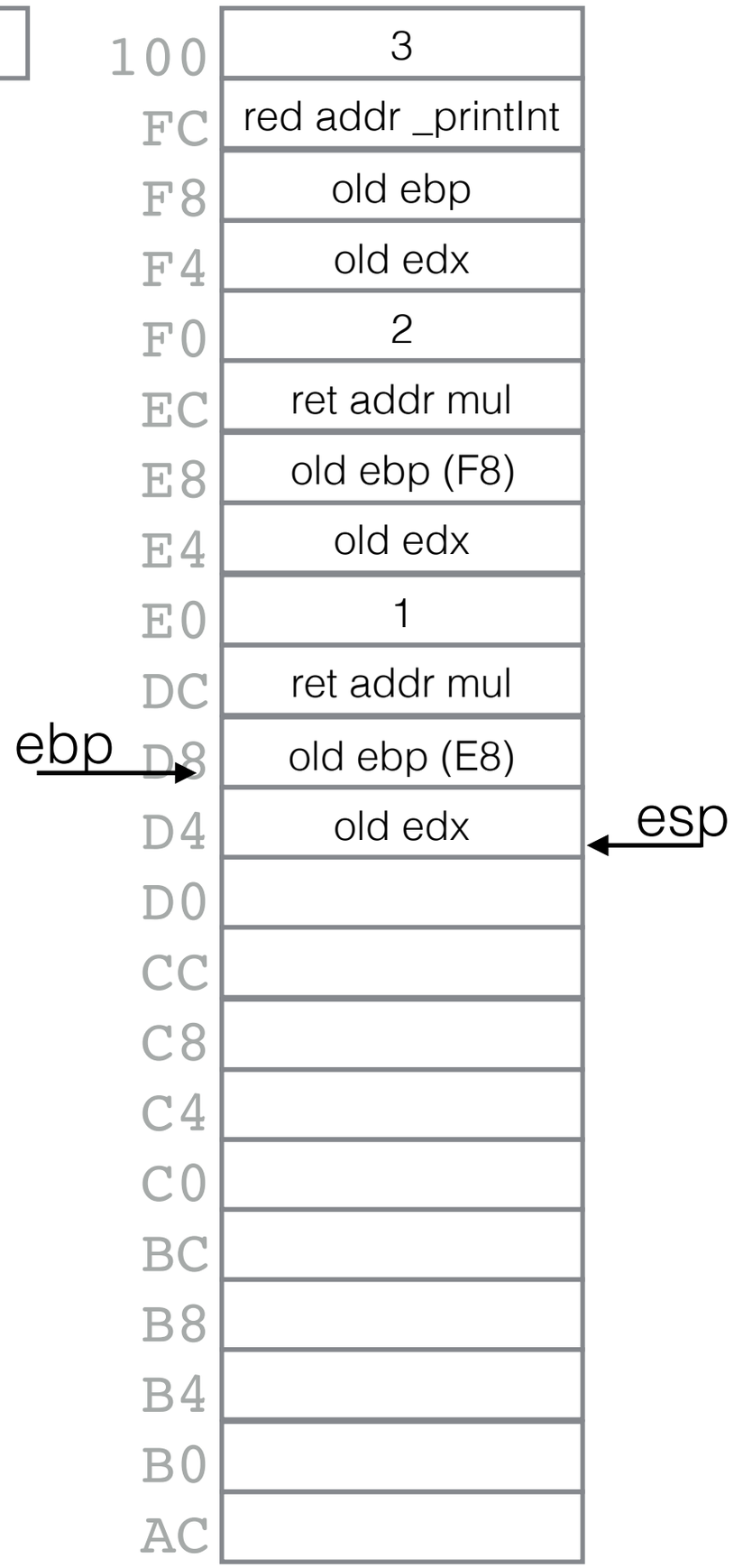
eax: 1



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

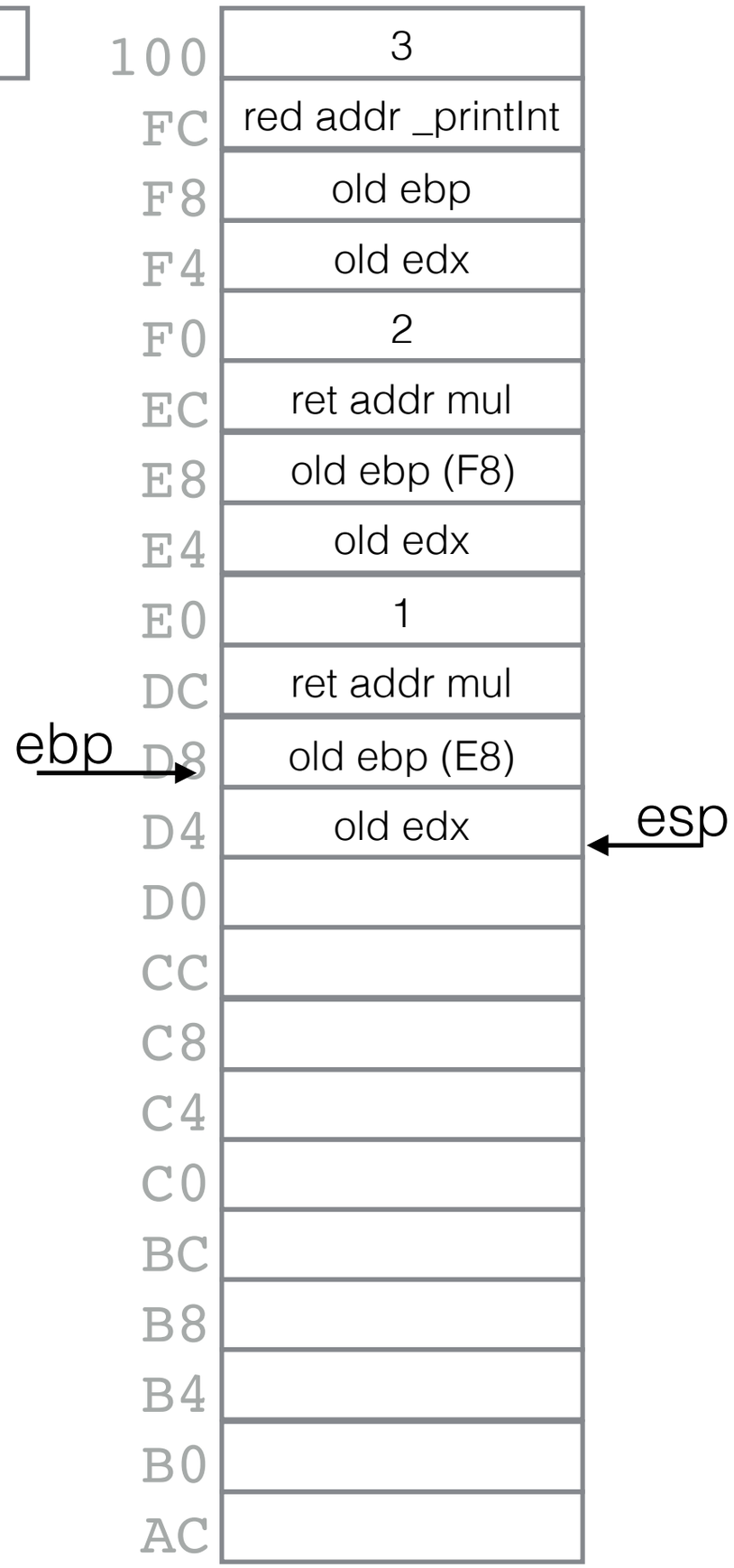
eax: 1



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

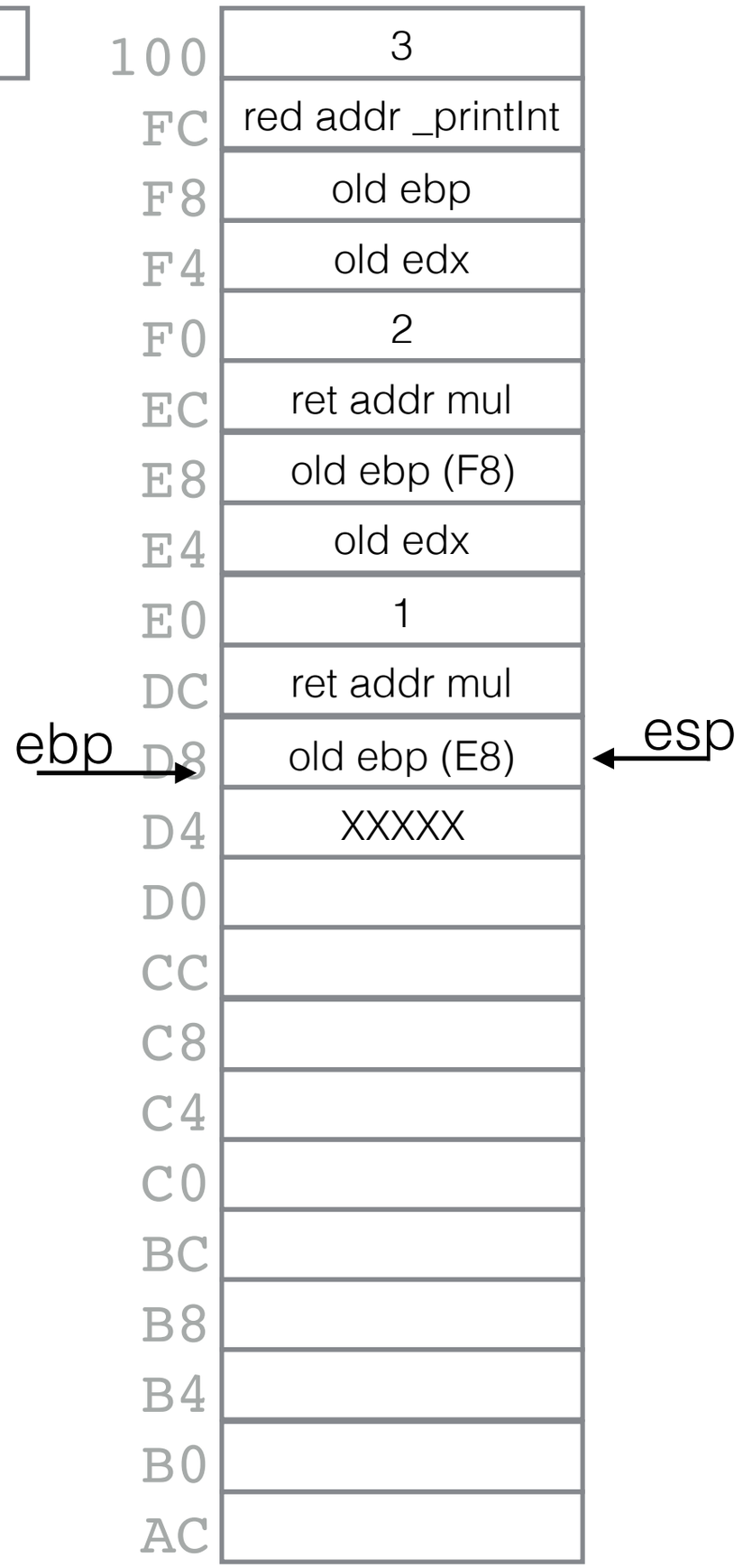
eax: 1



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

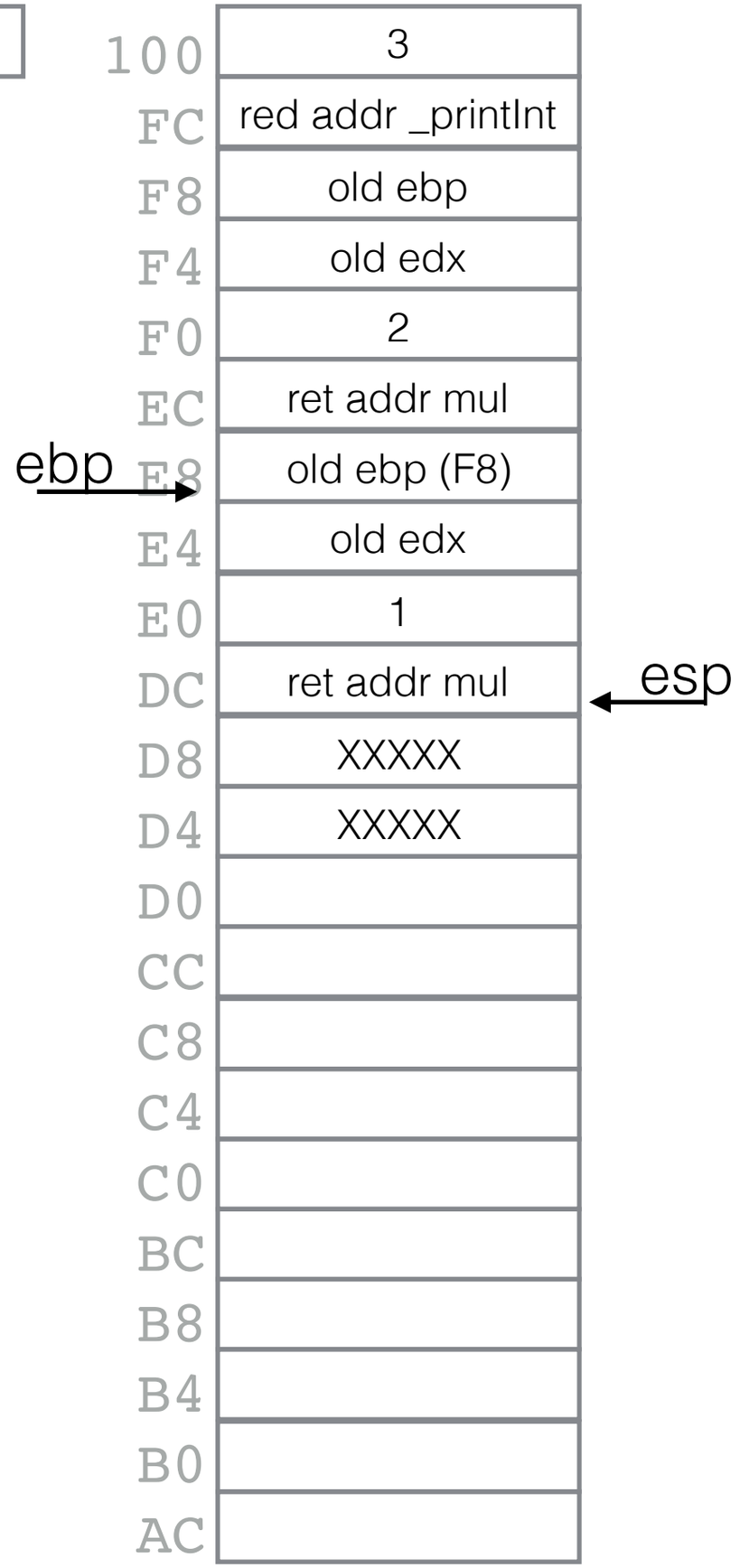
eax: 1



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

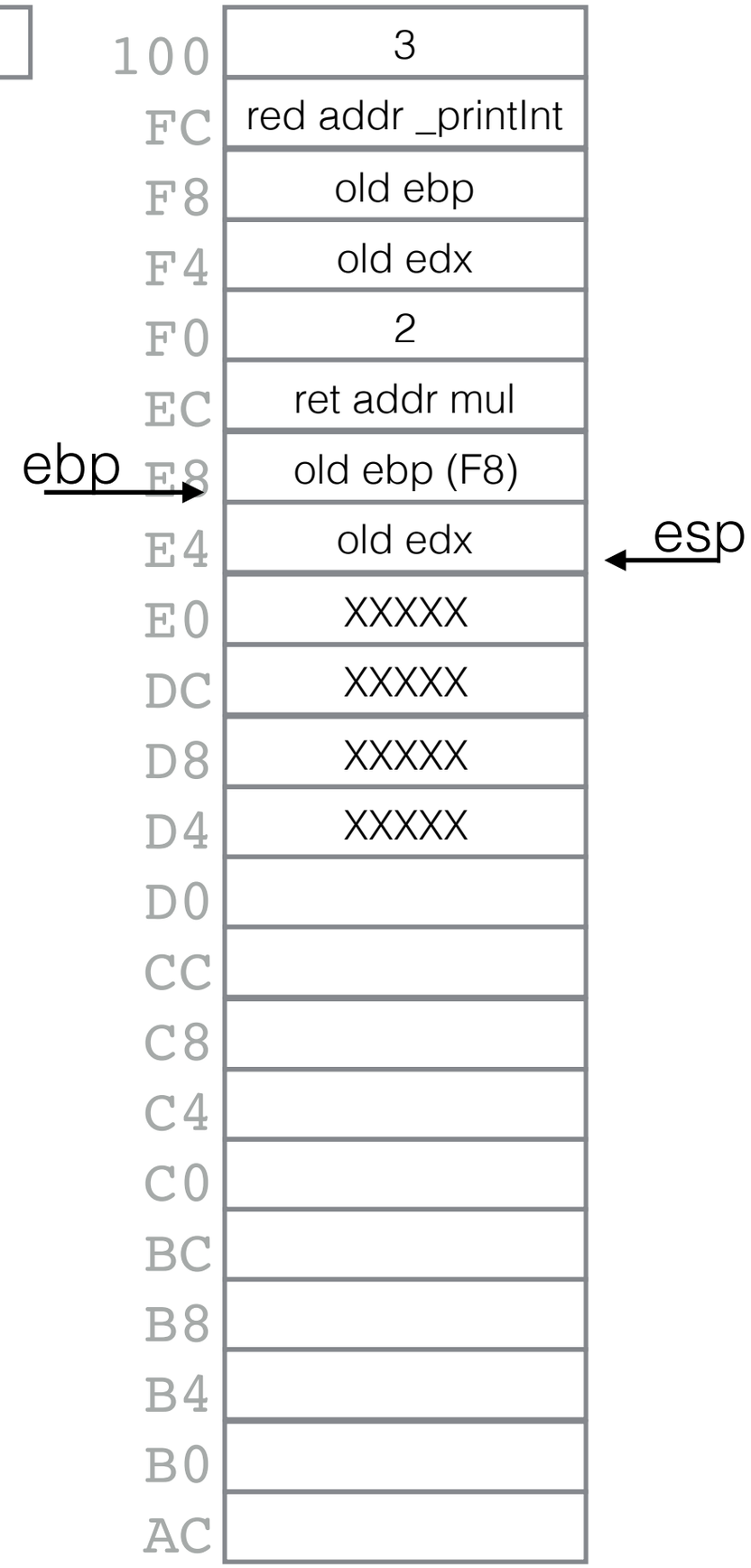
eax: 1



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

eax: 1

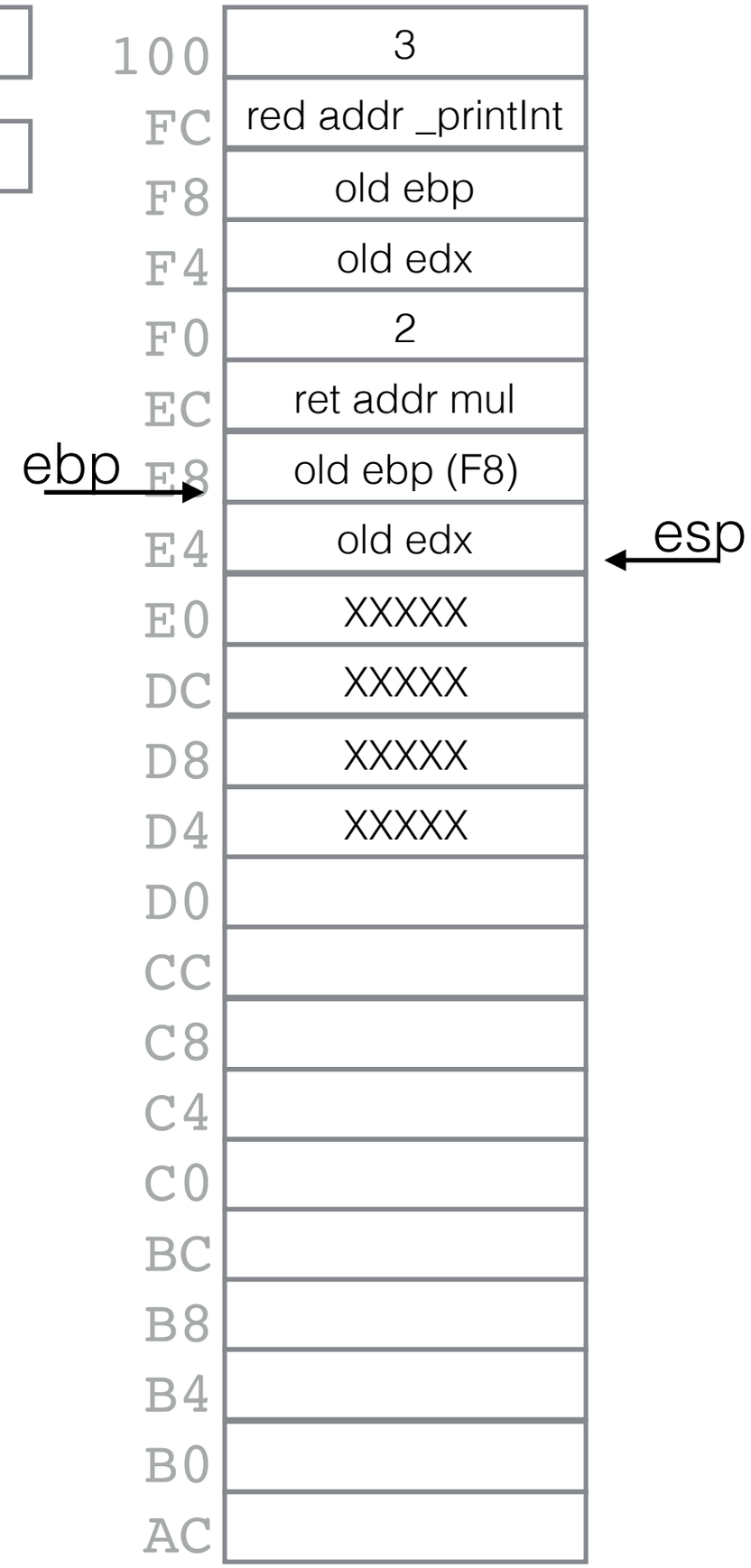


```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

eax: 2

edx: 0

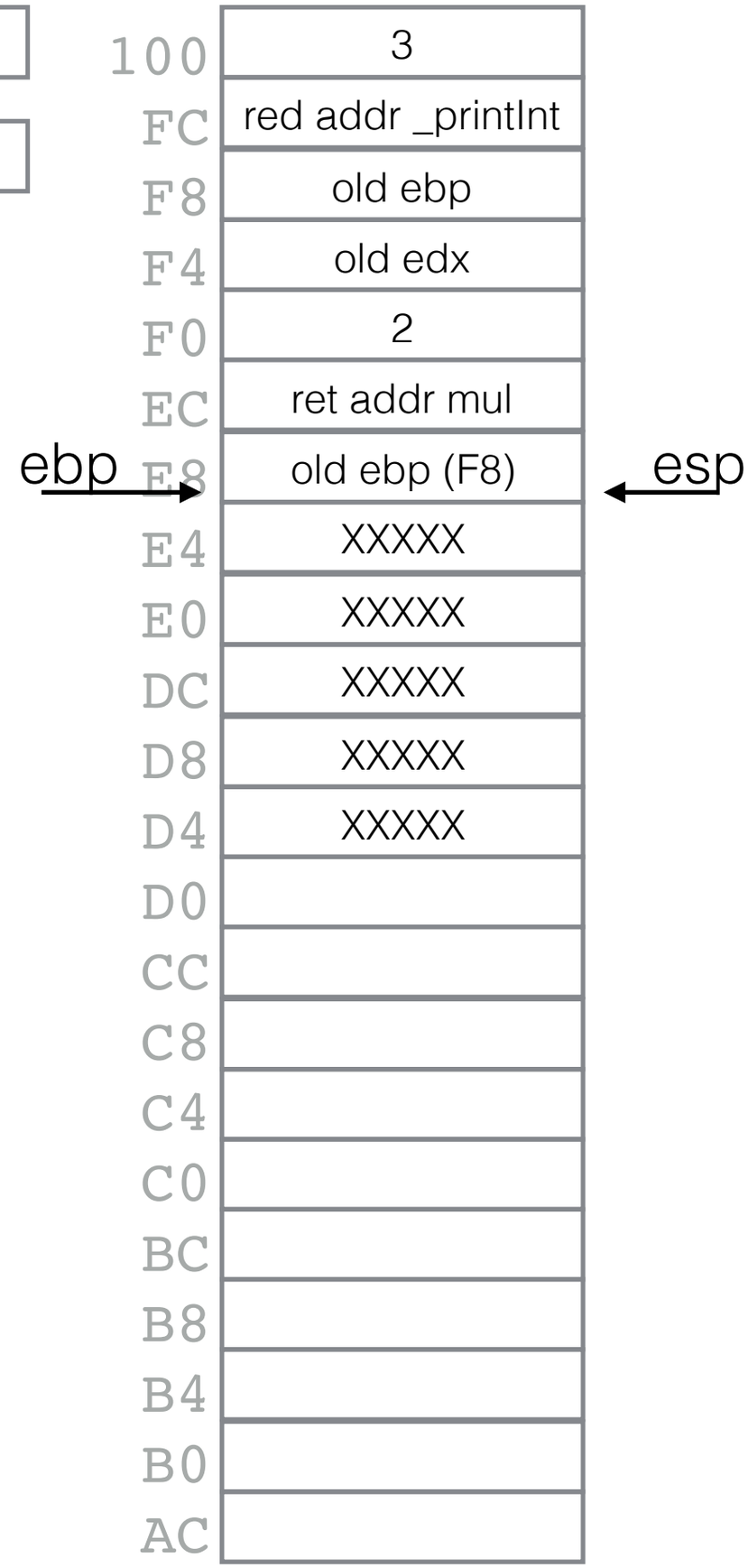


```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

eax: 2

edx: xxxx

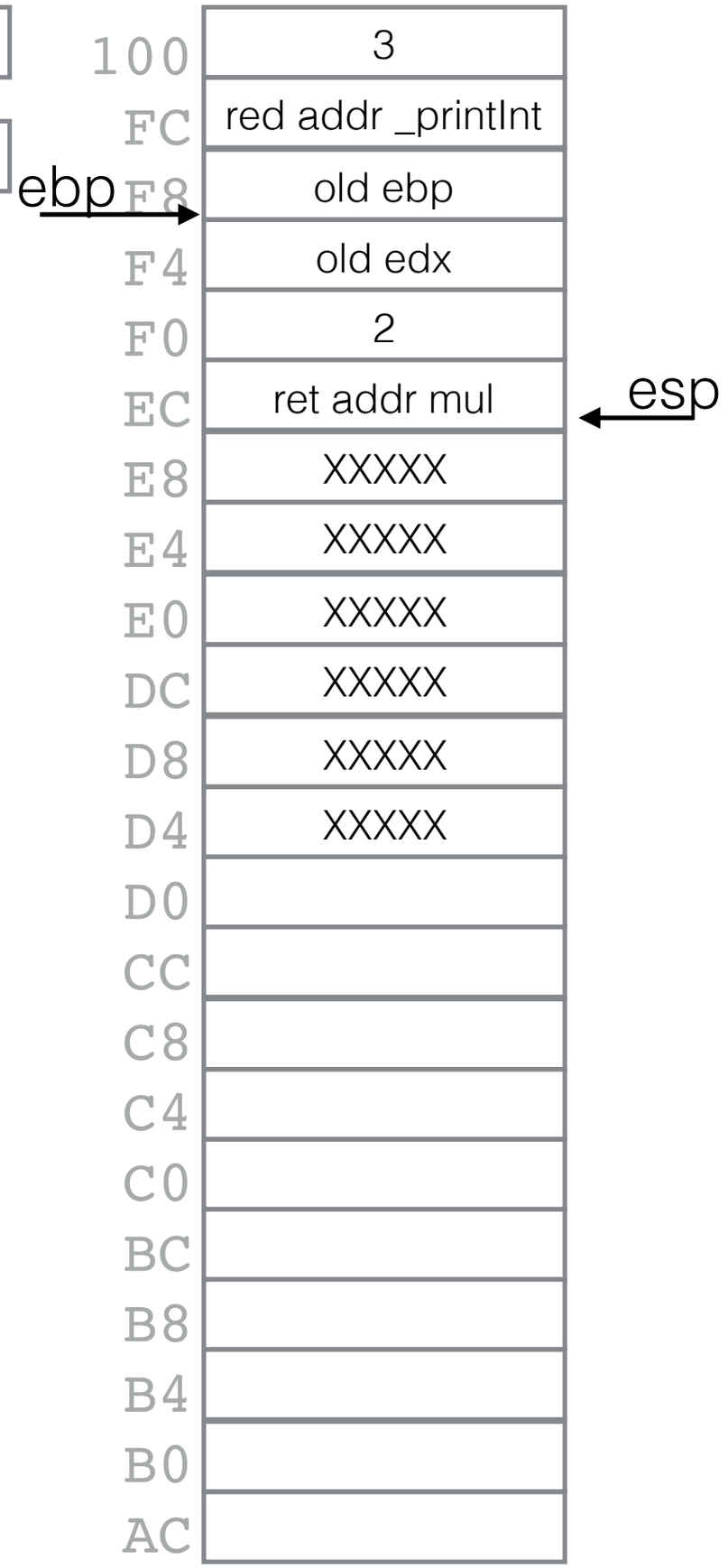


```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

eax: 2

edx: xxxx

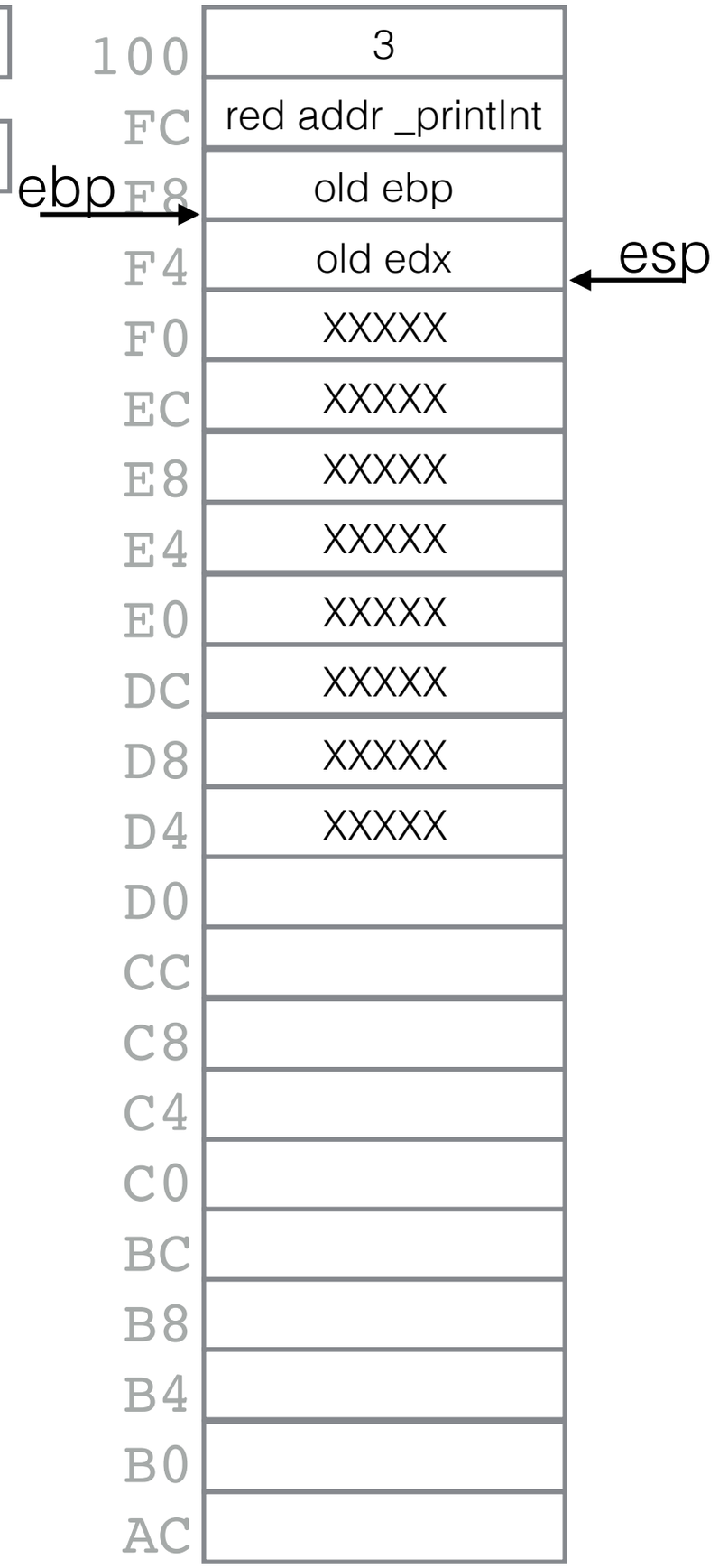


```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

eax: 2

edx: xxxx

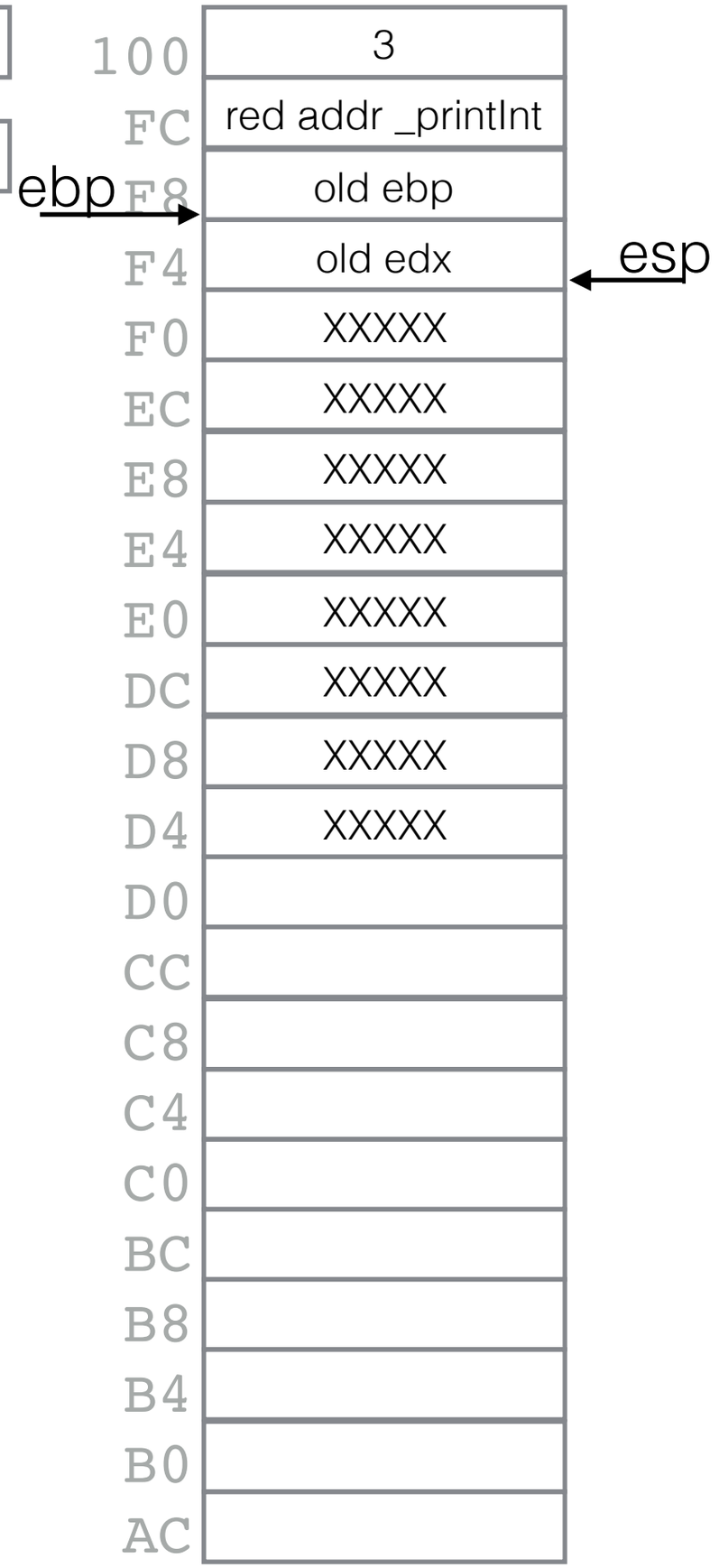


```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

eax: 6

edx: 0

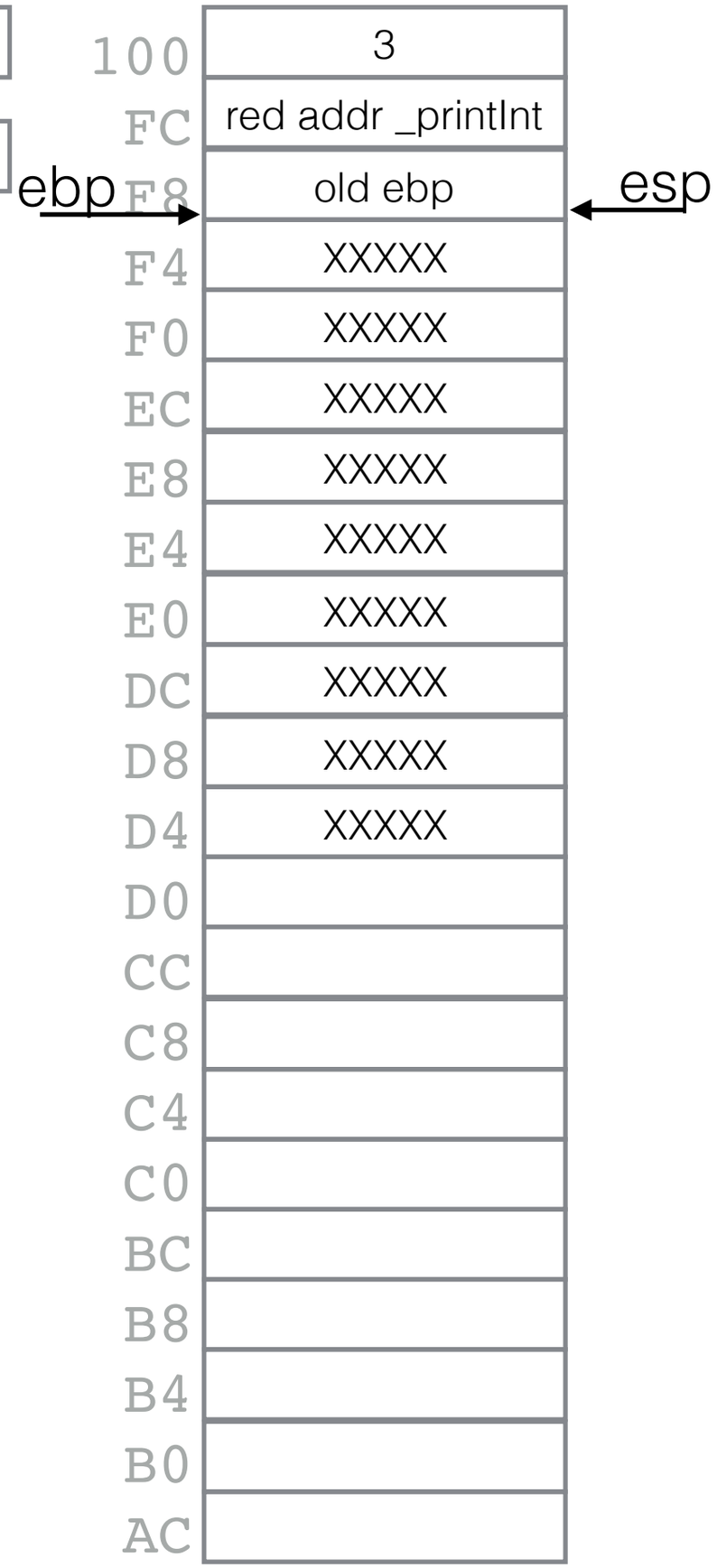


```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

eax: 6

edx: XXX



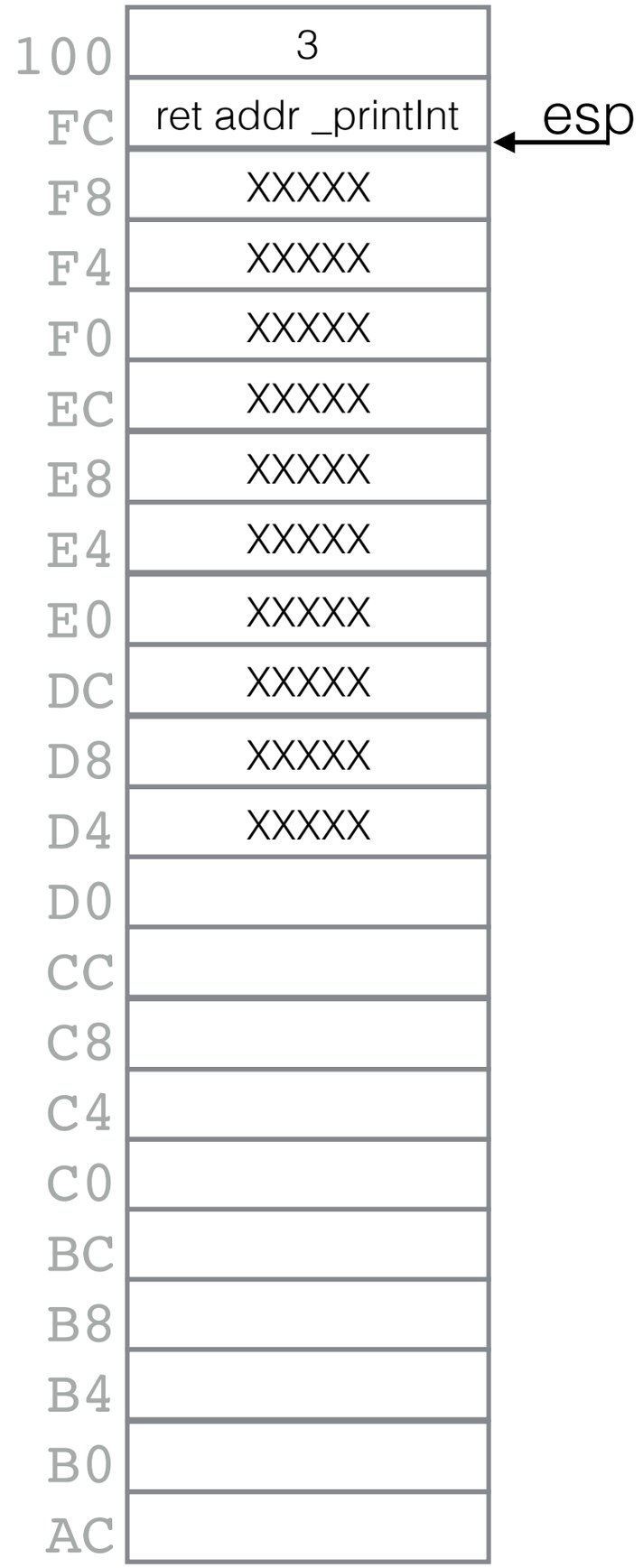
```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

eax: 6

edx: XXX

ebp



```
;;; def main():
;;;     n = 3
;;;     print( "fact(", n, ") =", fact( n ) )
;;; main()

_start: push    dword[n]
        call    fact
        call    _printInt
        call    _println
```

```
;;; ------------------------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------------------------
;;; passes n in stack, returns result in eax
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax             ;eax <- n-1
        push    eax             ;pass n-1 to fact
        call    fact            ;eax <- fact(n-1)
        mul     dword[ebp+8]    ;edx:eax <- eax * n = fact(n-1) * n

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

eax: 6

edx: XXX

ebp

esp

100	XXXXX
FC	XXXXX
F8	XXXXX
F4	XXXXX
F0	XXXXX
EC	XXXXX
E8	XXXXX
E4	XXXXX
E0	XXXXX
DC	XXXXX
D8	XXXXX
D4	XXXXX
D0	
CC	
C8	
C4	
C0	
BC	
B8	
B4	
B0	
AC	

Following the path of the Execution for fact(3)…

```
;;; -------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; -------------------------
;;; passes n in stack, returns r
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax
        push    eax
        call    fact
        mul     dword[ebp+8]

.endFact:
        pop     edx
        pop     ebp
        ret     4
```


```
;;; -------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; -------------------------
;;; passes n in stack, returns r
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax
        push    eax
        call    fact
        mul     dword[ebp+8]

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

```
;;; -------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; -------------------------
;;; passes n in stack, returns r
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax
        push    eax
        call    fact
        mul     dword[ebp+8]

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

```
;;; ------------------------------
;;; def fact( n ):
;;;     if n==1:
;;;         return 1
;;;     res = fact( n-1 )
;;;     return n * res
;;; ------------------------------
;;; passes n in stack, returns r
fact:   push    ebp
        mov     ebp, esp
        push    edx

.if:    cmp     dword[ebp+8], 1
        jg      .recurse
        mov     eax, 1
        jmp     .endFact

.recurse:
        mov     eax,dword[ebp+8]
        dec     eax
        push    eax
        call    fact
        mul     dword[ebp+8]

.endFact:
        pop     edx
        pop     ebp
        ret     4
```

Java Version

Does it break down at some point?

```
class Factorial {

    private static int fact( int n ) {
        if ( n <= 1 ) return 1;
        return n * fact( n-1 );
    }

    public static void main( String args[] ) {
        int n =  Integer.parseInt ( args[0] );
        System.out.print( "fact(" + n + ")=" + fact(n) +"\n\n" );
    }

}
```

getcopy Factorial.java

```
java Factorial 10000
java Factorial 10000 2> javaErrors.log
less javaErrors.log
cat  javaErrors.log  | wc -l
```

Compare to Non-Recursive Version

```
;;; ------------------------------------------------------------------
;;; fact: receives n in eax and returns n! in eax
;;; ------------------------------------------------------------------
fact:           push    edx         ; save registers affected
                push    ecx
                mov     ecx, eax    ; ecx <- n
                mov     eax, 1      ; eax accumulates n!
.for:           mul     ecx         ; edx:eax <- eax * ecx
                loop    .for

                pop     ecx         ; restore ecx
                pop     edx         ; restore edx
                ret
```

getcopy factorialNonRecursive.asm

Question 2

- Compare the execution time of the recursive version of ***factorial()*** to its non-recursive version. (Use the assembly versions to answer this question)

- count the number of cycles for each
- for N!, non-recursive # steps= k1 + 2(N)
- for N! recursive # steps = k2 + 13(N)

Question 3

- If the maximum stack size given to a program is 8 GBytes, how many terms could the assembly ***fact()*** function compute, at most, if we didn't care about multiplication overflow?
  Note: We can get the default stack size linux uses with

```
ulimit -a
```

```
cs231a@aurora ~/handout $ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 15770
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 15770
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

Question 4

- What are the space complexities for the recursive and non-recursive versions of Factorial?

- For the recursive version, we have *n* stack frames in the stack when computing *n!* Each stack frame contains 1) *n*, 2) the return address, 3) old ebp, 4) old edx, or 4x4 bytes = 16 bytes.
  Total stack space= 16*n* bytes = O(*n*)

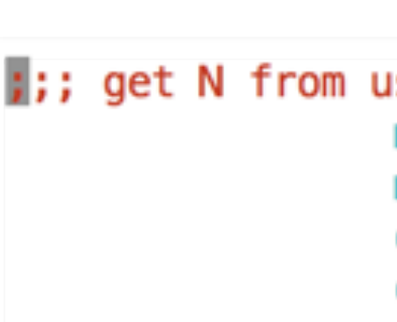
- For the non-recursive version, we have 3 dwords in the stack: 1) return address, 2) old edx, and 3) old ecx.
  Total stack space = 3x3 = 9 bytes = O(1)

Towers of Hanoi… in Assembly

- In Python first
- In Assembly next

https://media-cdn.tripadvisor.com/media/photo-s/0f/00/ee/18/ulun-danu-bratan-temple.jpg

```
;;; get N from user
                mov     ecx, prompt
                mov     edx, 2
                call    _printString
                call    _getInput

;;; define the 3 pegs and pass them in bl, cl, and dl.

                mov     bl, 'A'
                mov     cl, 'B'
                mov     dl, 'C'

;;; moveDisks( N, 'A', 'B', 'C' )          ; eax <- N
                call    moveDisks          ; bl  <- 'A'
                                           ; cl  <- 'B'
                                           ; dl  <- 'C'
```



main program calling hanoi
User provides *N*, # of disks

hanoi.py - /Users/thiebaut/Desktop/hanoi.py (3.5.4)

```
#
# D. Thiebaut
#
# A program demonstrating a recursive solution to
# the towers of Hanoi puzzle.
#
from __future__ import print_function

def moveDisks( N, src, dest, extra ):
    if N==1:
        print( src, dest )
    else:
        moveDisks( N-1, src, extra, dest )
        moveDisks( 1, src, dest, extra )
        moveDisks( N-1, extra, dest, src )

def main():
    N = int( input( "Number of disks?> " ) )
    #                src  dest  extra
    moveDisks( N, 'A', 'C', 'B' )

main()
```

Ln: 23 Col: 0

Version 1

```
;;; ------------------------------------------------------------------
;;; moveDisks( n, source, dest, extra )
;;;            eax   bl     cl     dl
;;; Moves the n disks from source to dest using extra if necessary.
;;; Uses recursion to move the N-1 disks above the last one.
;;; Does not modify any of the registers
;;; ------------------------------------------------------------------
moveDisks:      pushad

;;; if n==1:
;;;    print( source, dest )
                cmp     eax, 1
                jg      recurse
                mov     al, bl
                call    printChar
                mov     al, ' '
                call    printChar
                mov     al, cl
                call    printChar
                call    _println

                popad
                ret
```

```
recurse:
;;; moveDisks( n-1, source, temp, dest )
                dec     eax                 ; eax <- n-1
                xchg    cl, dl              ; swap cl & dl
                call    moveDisks           ; move n-1
                xchg    cl, dl              ; swap them back


;;; print( source, dest )
                mov     al, bl              ; print source
                call    printChar
                mov     al, ' '             ; print space
                call    printChar
                mov     al, cl              ; print dest
                call    printChar
                call    _println            ; print \n

;;; moveDisks( n-1, temp, dest, source )
                popad
                pushad                      ; makes sense, but not needed
                xchg    bl, dl
                dec     eax
                call    moveDisks

                popad                       ; makes sense, by not needed
                ret
```

Version 2

```
;;; ------------------------------------------------------------------
;;; moveDisks( n, source, dest, extra )
;;;            eax   bl     cl     dl
;;; Moves the n disks from source to dest using extra if necessary.
;;; Uses recursion to move the N-1 disks above the last one.
;;; Does not modify any of the registers
;;; ------------------------------------------------------------------
moveDisks:      pushad

;;; if n==1:
;;;    print( source, dest )
                cmp     eax, 1
                jg      recurse
                mov     al, bl
                call    printChar
                mov     al, ' '
                call    printChar
                mov     al, cl
                call    printChar
                call    _println

                popad
                ret
```

```
recurse:
;;; moveDisks( n-1, source, temp, dest )
                dec     eax                 ; eax <- n-1
                xchg    cl, dl              ; swap cl & dl
                call    moveDisks           ; move n-1
                xchg    cl, dl              ; swap them back

;;; moveDisks( 1, source, temp, dest )
                push    eax
                mov     eax, 1
                call    moveDisks
                pop     eax

;;; moveDisks( n-1, temp, dest, source )
                popad
                pushad                      ; makes sense, but not needed
                xchg    bl, dl
                dec     eax
                call    moveDisks

                popad                       ; makes sense, by not needed
                ret
```