Smith College

Computer Science

CSC231 — Assembly

Week #12 —Thanksgiving 2017

Dominique Thiébaut
dthiebaut@smith.edu

Summary

```
;;; FUNCTION SIDE

function:   push    ebp             ;save old ebp
            mov     ebp, esp        ;make ebp point
                                    ;to stack frame

            xxx     dword[ebp+8]    ;access paramN
            xxx     dword[ebp+12]   ;access paramN-1


            pop     ebp             ;restore ebp
            ret     4N              ;return and pop
                                    ;4N bytes from stack
```

```
;;; CALLER SIDE

            push    param1
            push    param2
            …
            push    paramN
            call    function
```

Summary
(clean function)

```
;;; FUNCTION SIDE

function:    push    ebp             ;save old ebp
             mov     ebp, esp        ;make ebp point
                                     ;to stack frame
             push-registers-you-will-use

             xxx     dword[ebp+8]  ;access param1
             xxx     dword[ebp+12];access param2

             pop-register-you-used

             pop     ebp             ;restore ebp
             ret     4N              ;return and pop
                                     ;4N bytes from stack

;;; CALLER SIDE

                push    param1
                push    param2
                …
                push    paramN
                call    function
```

Useful Instructions for Functions

```
pushad              ;push EAX, ECX, EDX,
                    ;EBX, original ESP,
                    ; EBP, ESI, and EDI.

popad               ;pop them back in
                    ;reverse order
```

Example 1

```
;;; ; ------------------------------------------------------------
;;; ; _printString:         prints a string whose address is in
;;; ;                      ecx, and whose total number of chars
;;; ;                      is in edx.
;;; ; Examples:
;;; ; Assume a string labeled msg, containing "Hello World!",
;;; ; and a constant MSGLEN equal to 12.  To print this string:
;;; ;
;;; ;           mov        ecx, msg
;;; ;           mov        edx, MSGLEN
;;; ;           call       _printSTring
;;; ;
;;; ; REGISTERS MODIFIED:  NONE
;;; ; ------------------------------------------------------------

;;; ;save eax and ebx so that they are not modified by the function

_printString:
                push            eax
                push            ebx

                mov             eax,SYS_WRITE
                mov             ebx,STDOUT
                int             0x80

                pop             ebx
                pop             eax
                ret
```

Example 2

```
;;; ;-----------------------------------------------------
;;; ;-----------------------------------------------------
;;; ; getInput: gets a numerical input from the keyboard.
;;; ;           returns the resulting number in eax (32 bits).
;;; ;           recognizes - as the first character of
;;; ;           negative numbers.  Does not skip whitespace
;;; ;           at the beginning.  Stops on first not decimal
;;; ;           character encountered.
;;; ;
;;; ; NO REGISTERS MODIFIED, except eax
;;; ;
;;; ; Example of call:
;;; ;
;;; ;          call  getInput
;;; ;          mov   dword[x], eax ; put integer in x
;;; ;
;;; ;-----------------------------------------------------
;;; ;-----------------------------------------------------
_getInput:
                section .bss
buffer          resb    120
intg            resd    1
isneg           resb    1

                section .text
                pushad                  ; save all registers

                mov     esi, buffer     ; eci --> buffer
                mov     edi, 0          ; edi = counter of chars
.loop1:
                mov     eax, 03         ; input
                mov     ebx, 0          ; stdin
                mov     ecx, esi        ; where to put the next char

                ; cut for the sake of simplicity.  Look in 231Lib.asm for more info!
```

Dot-Labels

```
;;; ;-----------------------------------------------------
;;; ;-----------------------------------------------------
printLine:      mov     esi, buffer     ; esi --> buffer
                mov     edi, 0          ; edi = counter of chars
.loop1:
                mov     eax, 03         ; input
                mov     ebx, 0          ; stdin
                mov     ecx, esi        ; where to put the next char
                loop    .loop1

.for1:          mov     …

.for2:          mov     …
                . . .
                ret

;;; ;-----------------------------------------------------
;;; ;-----------------------------------------------------
printReg:       mov     esi, buffer
                mov     edi, 0

.for1:          mov     …

.for2:          mov     …
                . . .
                . . .
                ret
```

Dot-Labels

```
;;; ;-------------------------------------------------------
;;; ;-------------------------------------------------------
printLine:      mov     esi, buffer     ; esi --> buffer
                mov     edi, 0          ; edi = counter of chars
.loop1:
                mov     eax, 03         ; input
                mov     ebx, 0          ; stdin
                mov     ecx, esi        ; where to put the next char
                loop    .loop1

.for1:          mov     …

.for2:          mov     …
                . . .
                ret

;;; ;-------------------------------------------------------
;;; ;-------------------------------------------------------
printReg:       mov     esi, buffer
                mov     edi, 0

.for1:          mov     …

.for2:          mov     …
                . . .
                . . .
                ret
```

printLine.for2:

printReg.for2:

Exercise

- Write a function that copies an array of N bytes into another array of N bytes. The function receives three parameters: the *address* of the first array, the *address* of the second array, and the *number* of bytes.

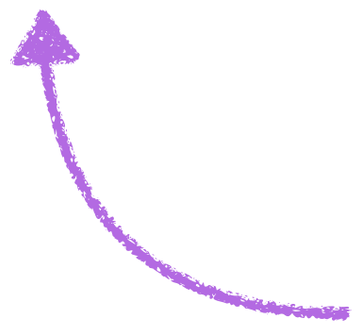
```
void copyArrays( table1, table2, N );
```

- Passing through **registers** ✓
- Passing through the **stack** ✓
- Passing by **Value** ✓
- Passing by **Reference**

Rule for Writing Functions:

Pushing and *popping* operations into/from the stack must always cancel each other out!

(a + 3 (b^2 + (c+1) ^(d-1)))



Same as with parentheses

An Example

```
# example.py
from __future__ import print_function

def incrementAll( array ):
    for i in range( len( array ) ):
        array[ i ] += 1



Table = [1, 2, 3, 4]
print( str( Table ) )

incrementAll( Table )

print( str( Table ) )
```

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
```

An Example

```
# example.py
from __future__ import print_function

def incrementAll( array ):
    for i in range( len( array ) ):
        array[ i ] += 1


Table = [1, 2, 3, 4]
print( str( Table ) )

incrementAll( Table )

print( str( Table ) )
```

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
```

An Example

```
# example.py
from __future__ import print_function

def incrementAll( array ):
    for i in range( len( array ) ):
        array[ i ] += 1


Table = [1, 2, 3, 4]
print( str( Table ) )

incrementAll( Table )

print( str( Table ) )
```

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
```

An Example

```
# example.py
from __future__ import print_function

def incrementAll( array ):
    for i in range( len( array ) ):
        array[ i ] += 1


Table = [1, 2, 3, 4]
print( str( Table ) )

incrementAll( Table )

print( str( Table ) )
```

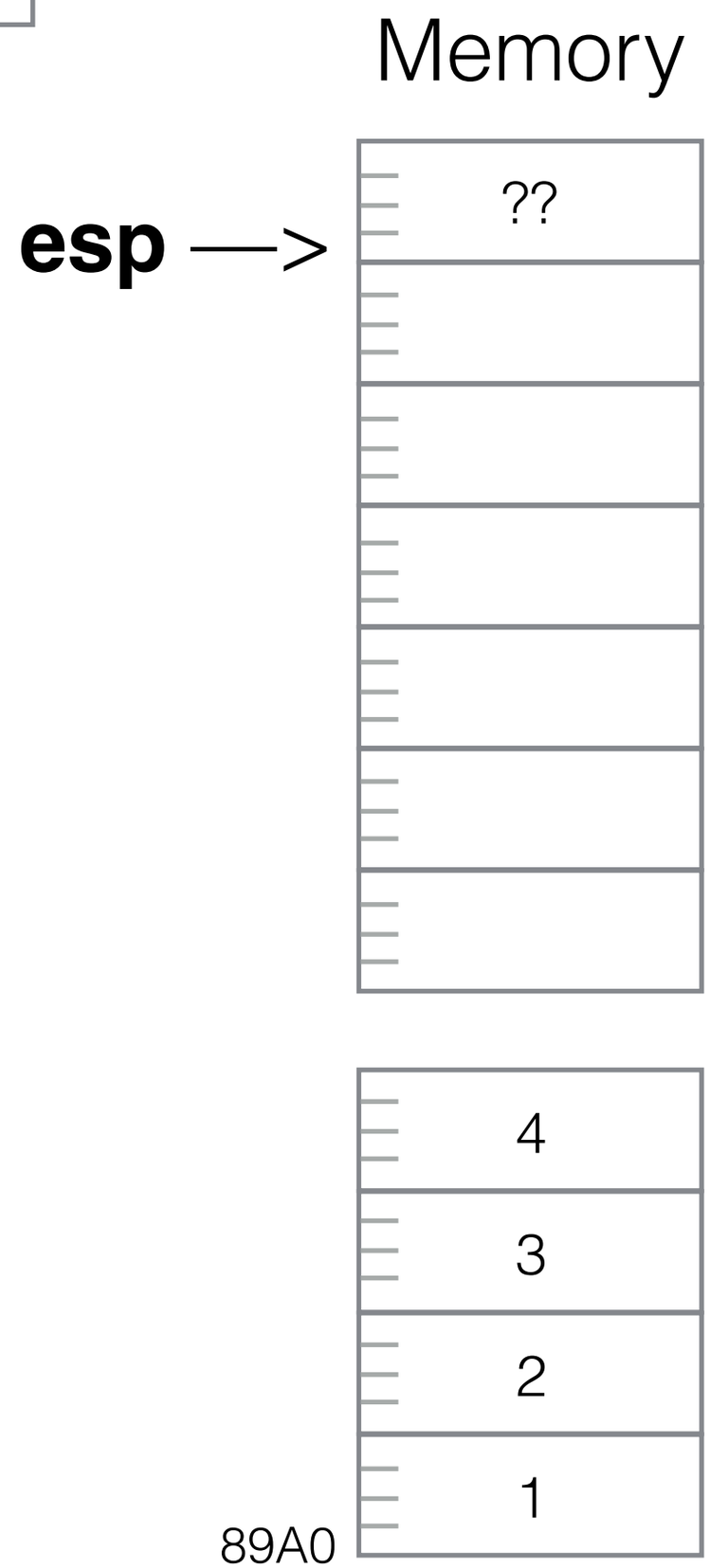
```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
```

Single-Step Execution

Passing Parameters by **Reference**: an Example

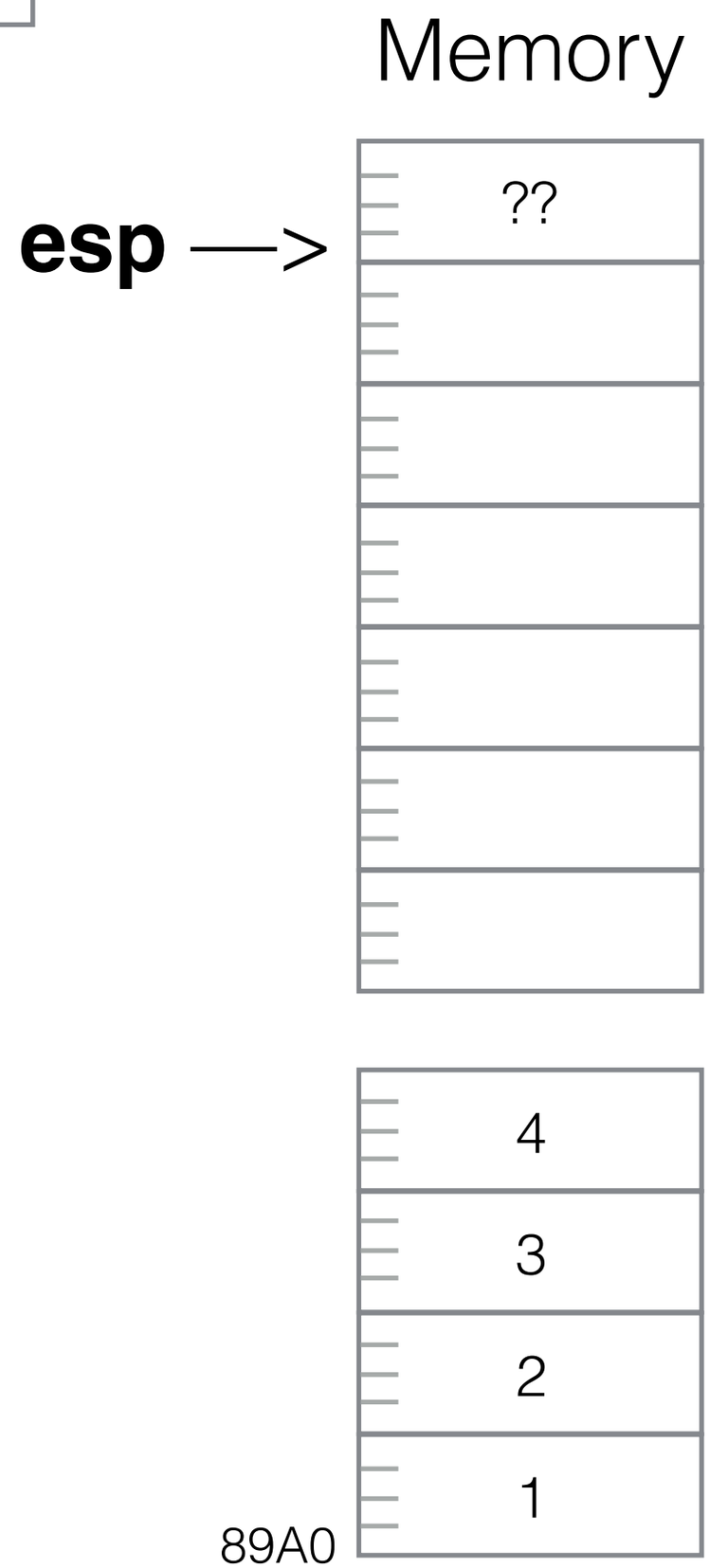
eax ?? **ebx** ?? **ecx** ??

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table      <— eip
                push    eax
                call    incrementAll
                xxx
```



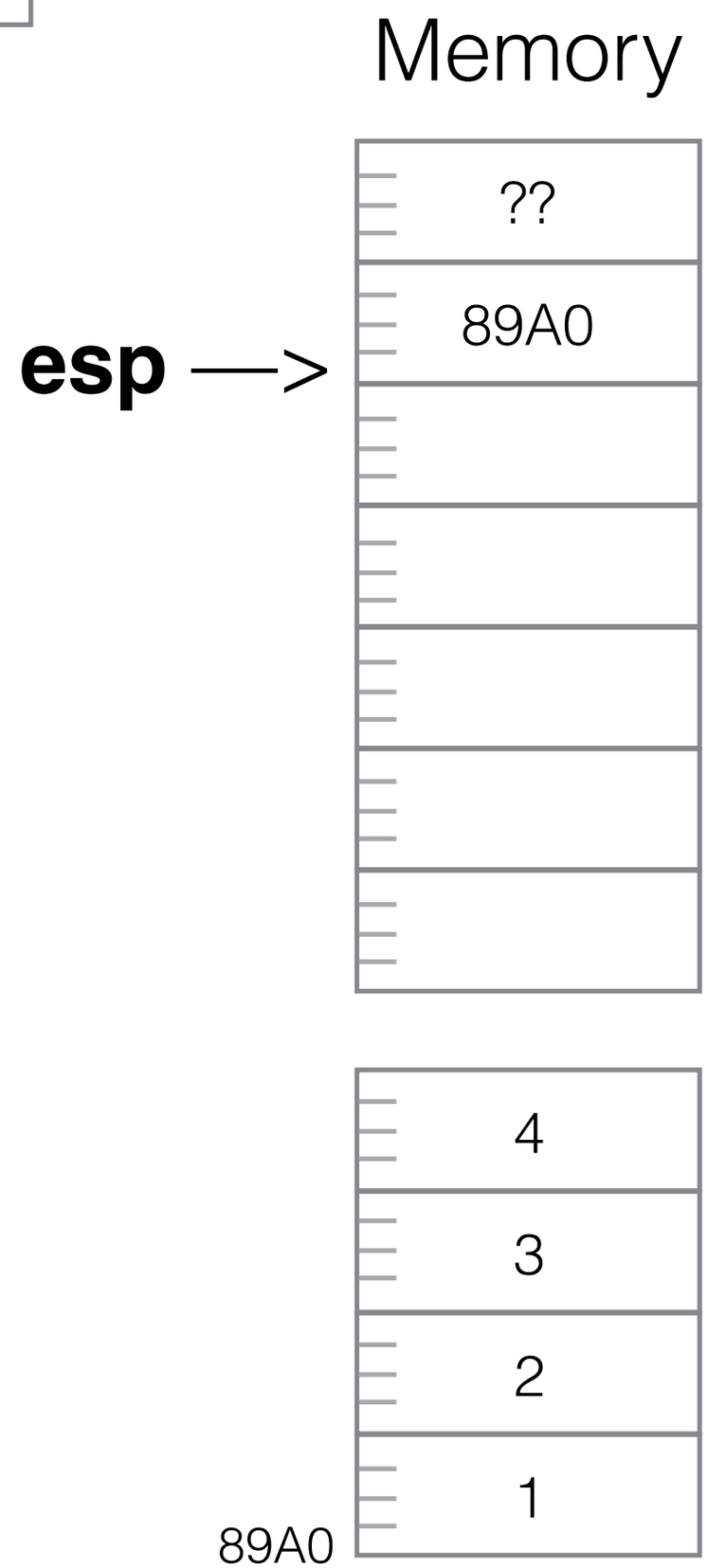
eax 89A0 **ebx** ?? **ecx** ??

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax             <— eip
                call    incrementAll
                xxx
```



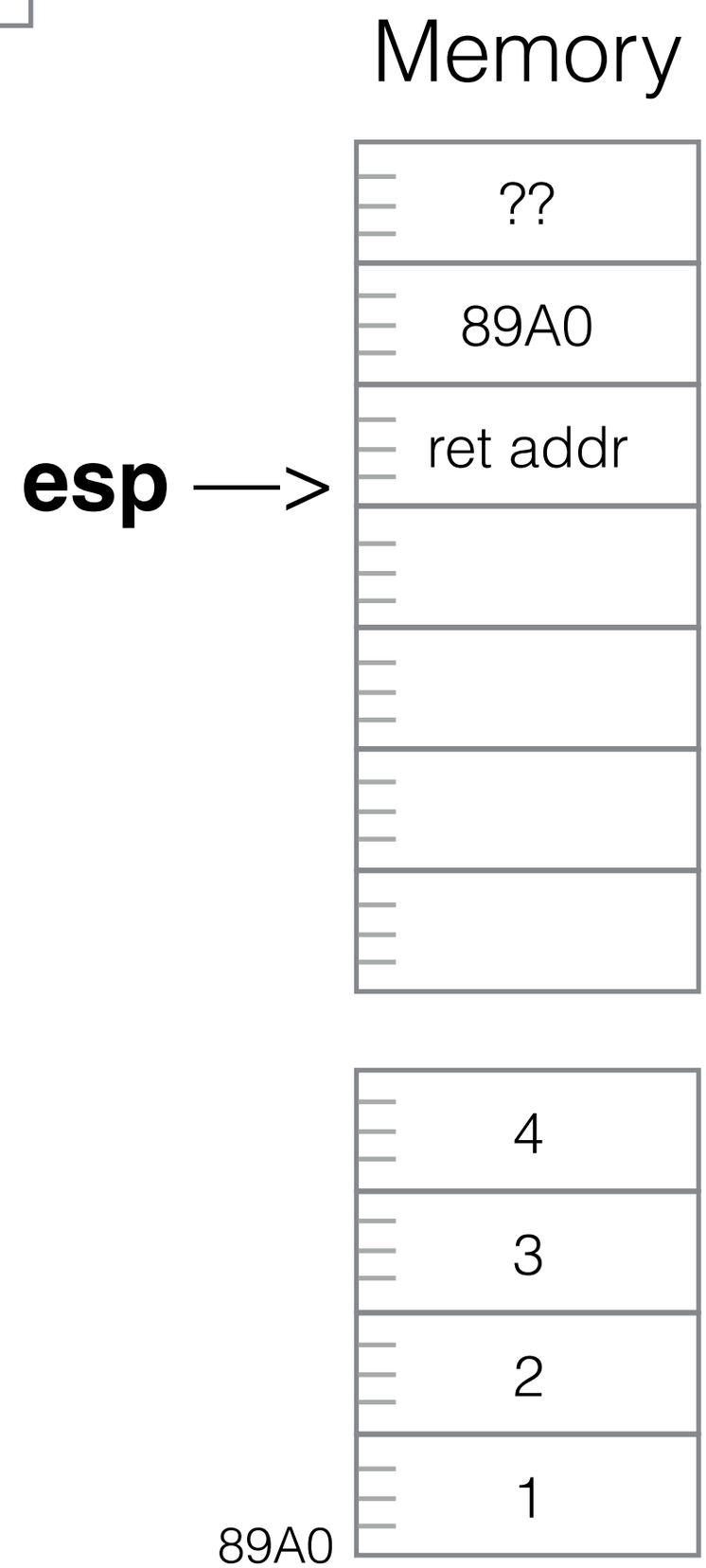
eax 89A0 **ebx** ?? **ecx** ??

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll    <— eip
                xxx
```



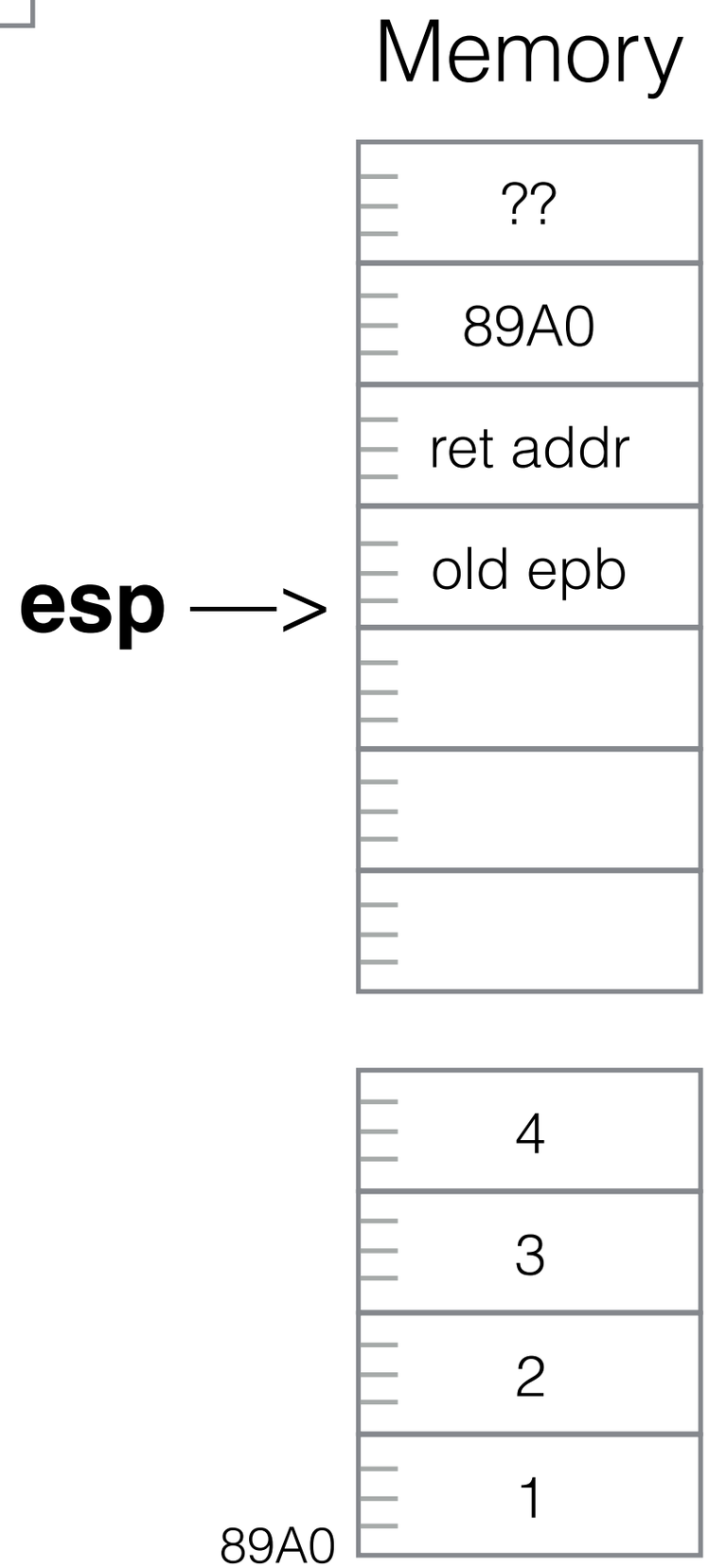
eax 89A0 **ebx** ?? **ecx** ??

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp             <— eip
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx
```



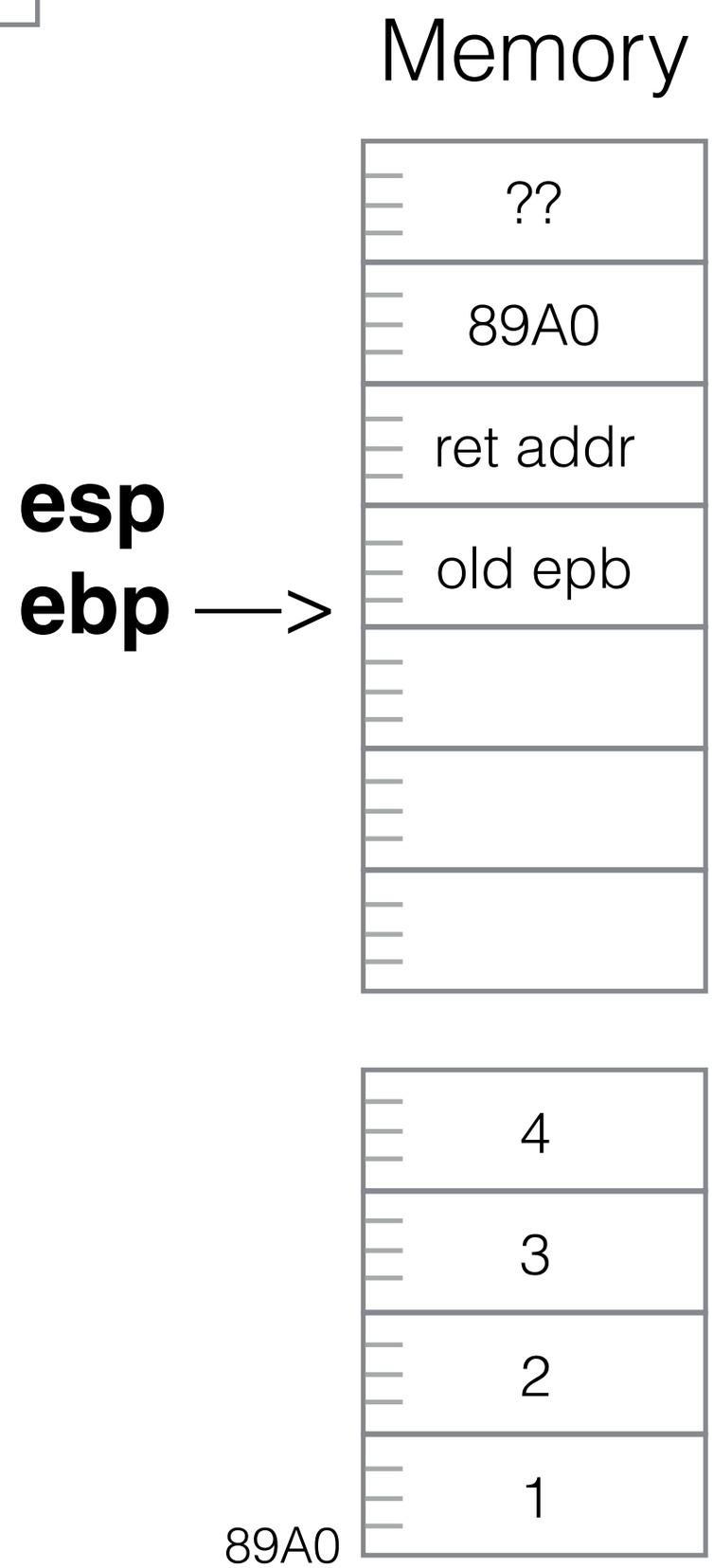
eax 89A0 **ebx** ?? **ecx** ??

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp        <— eip
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx
```



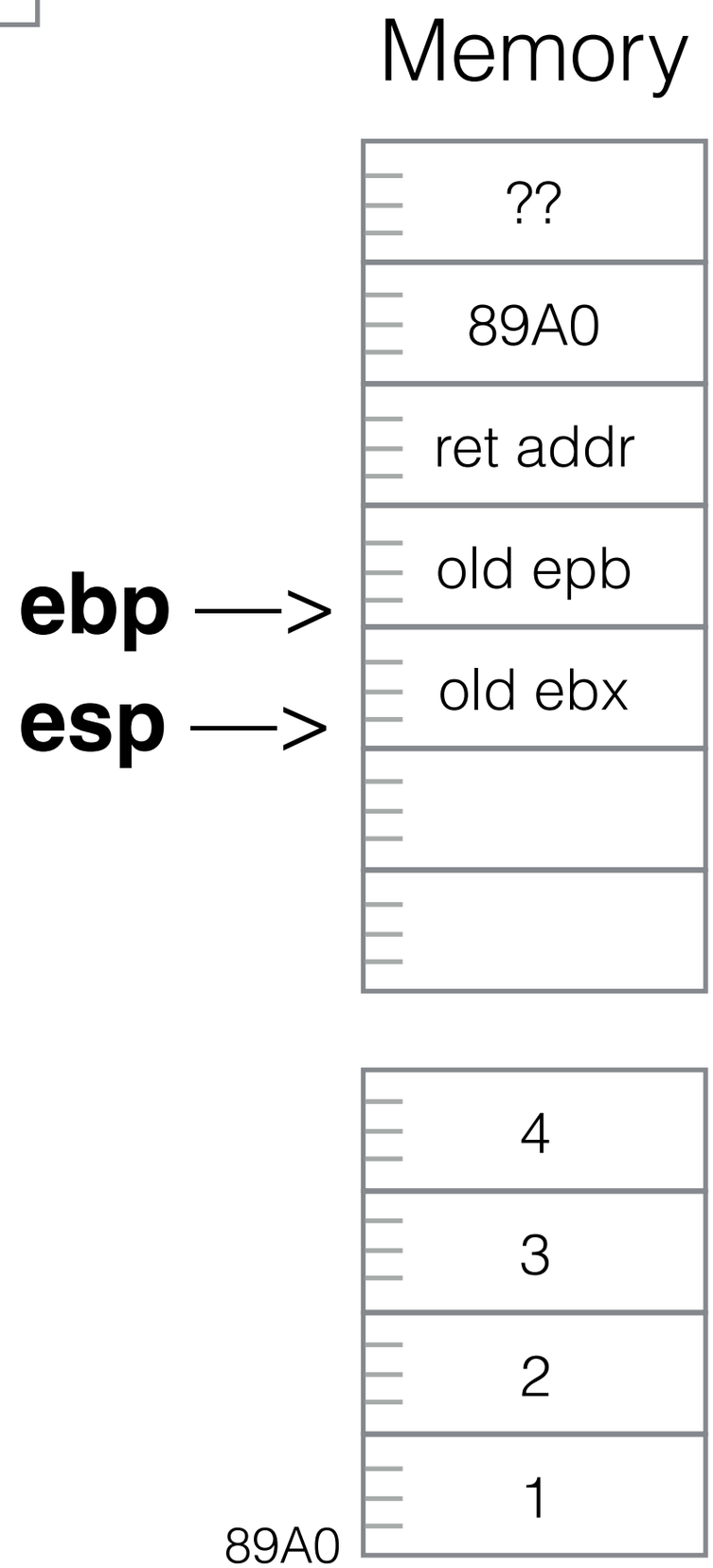
eax 89A0 **ebx** ?? **ecx** ??

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx             <— eip
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx
```



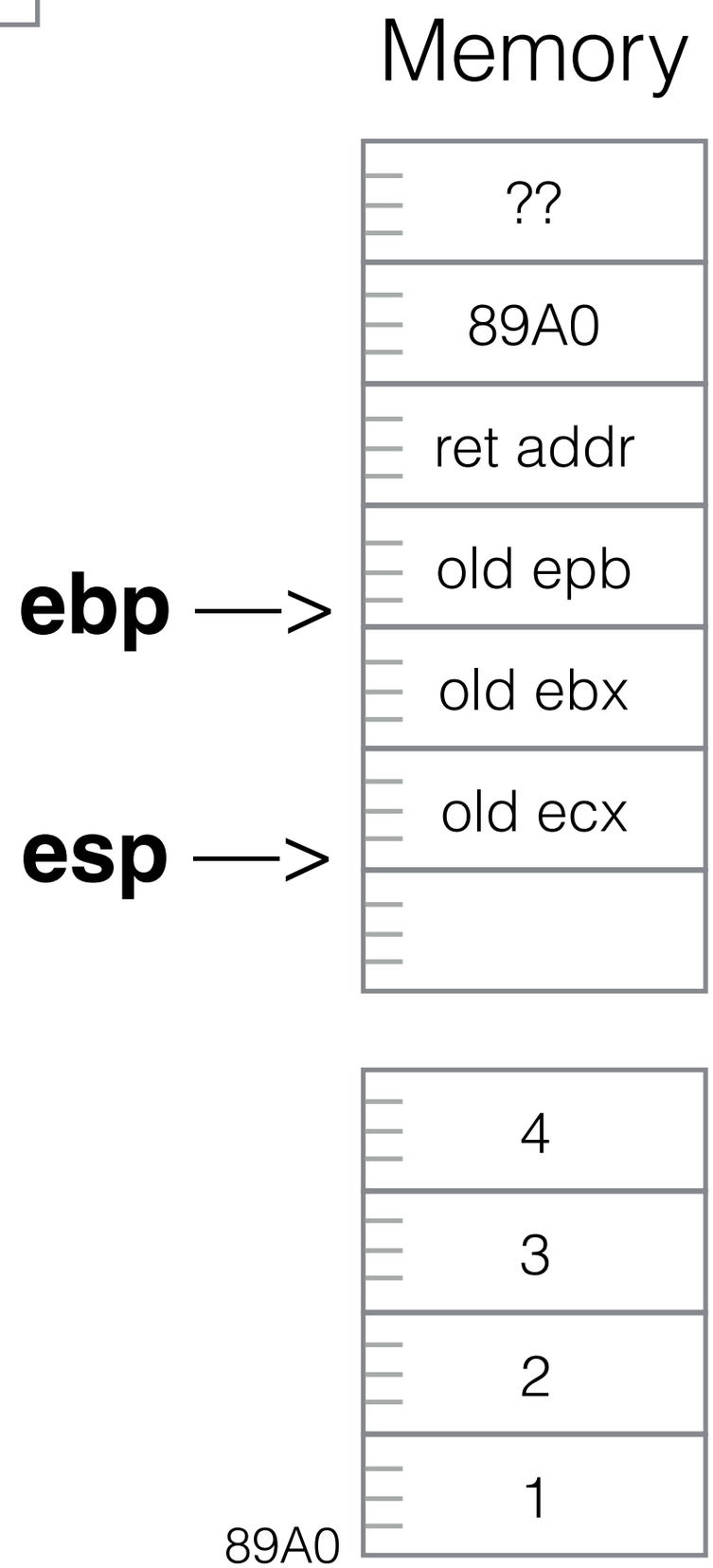
eax 89A0 **ebx** ?? **ecx** ??

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx             <— eip
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx
```



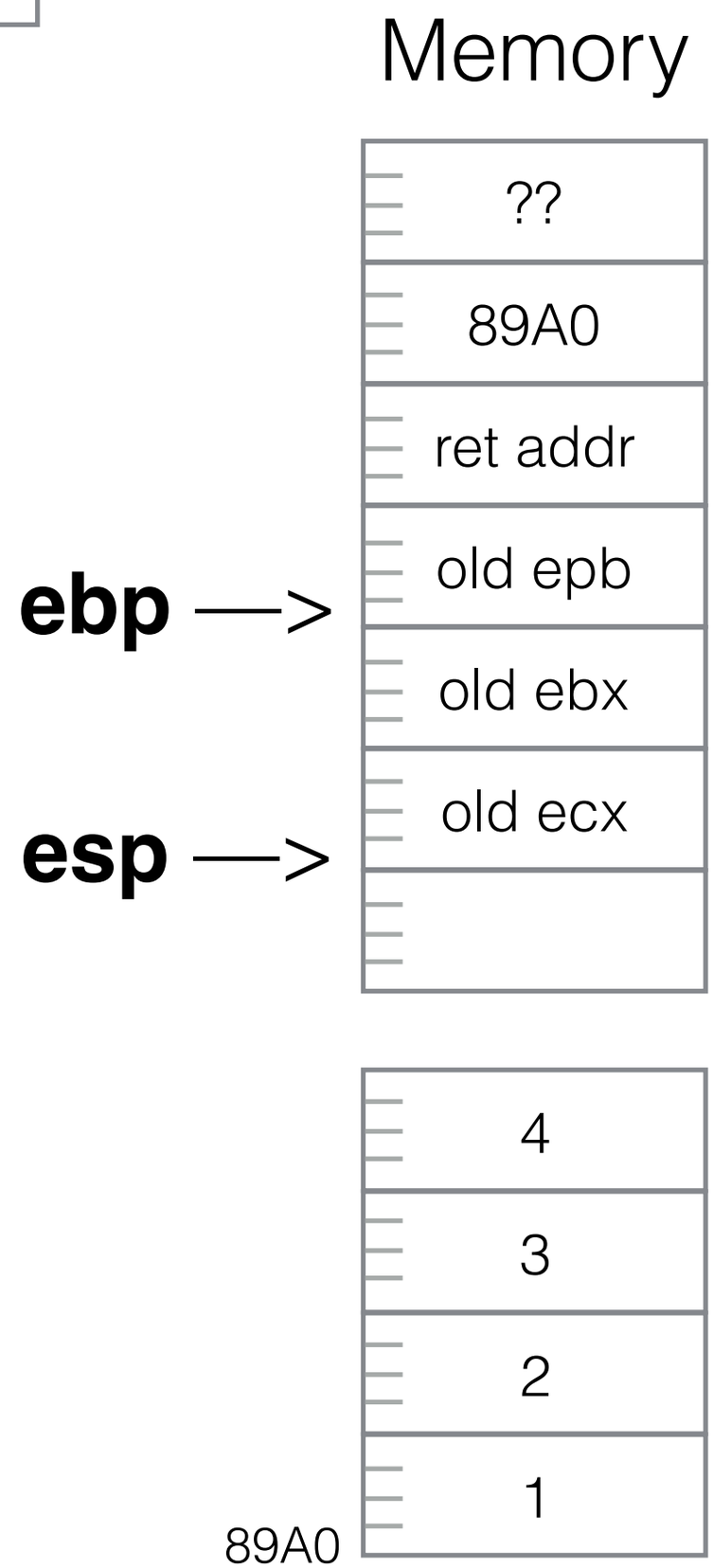
eax 89A0 **ebx** ?? **ecx** ??

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4          <— eip
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx
```



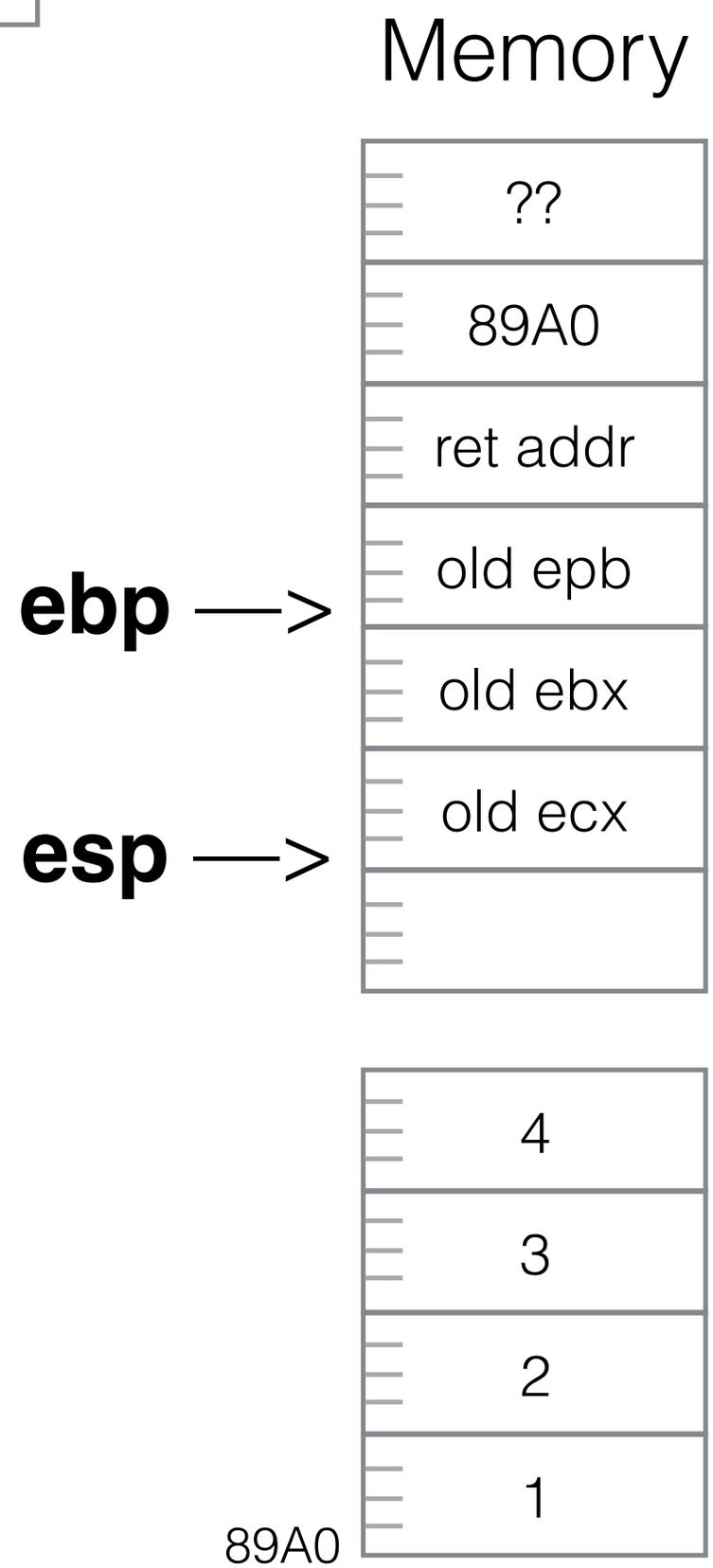
eax 89A0 **ebx** ?? **ecx** 4

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]   <— eip
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx
```



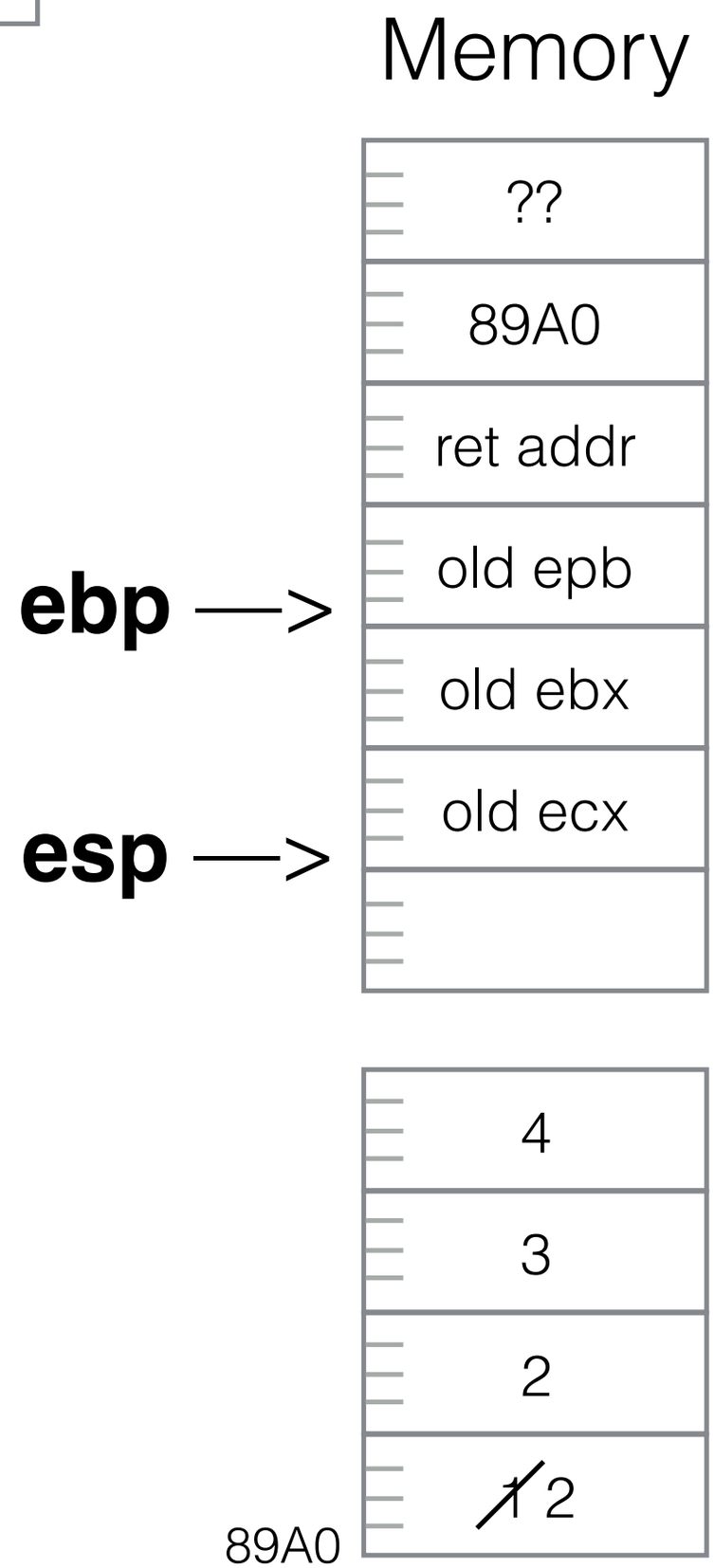
eax 89A0 **ebx** 89A0 **ecx** 4

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]          <— eip
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx
```



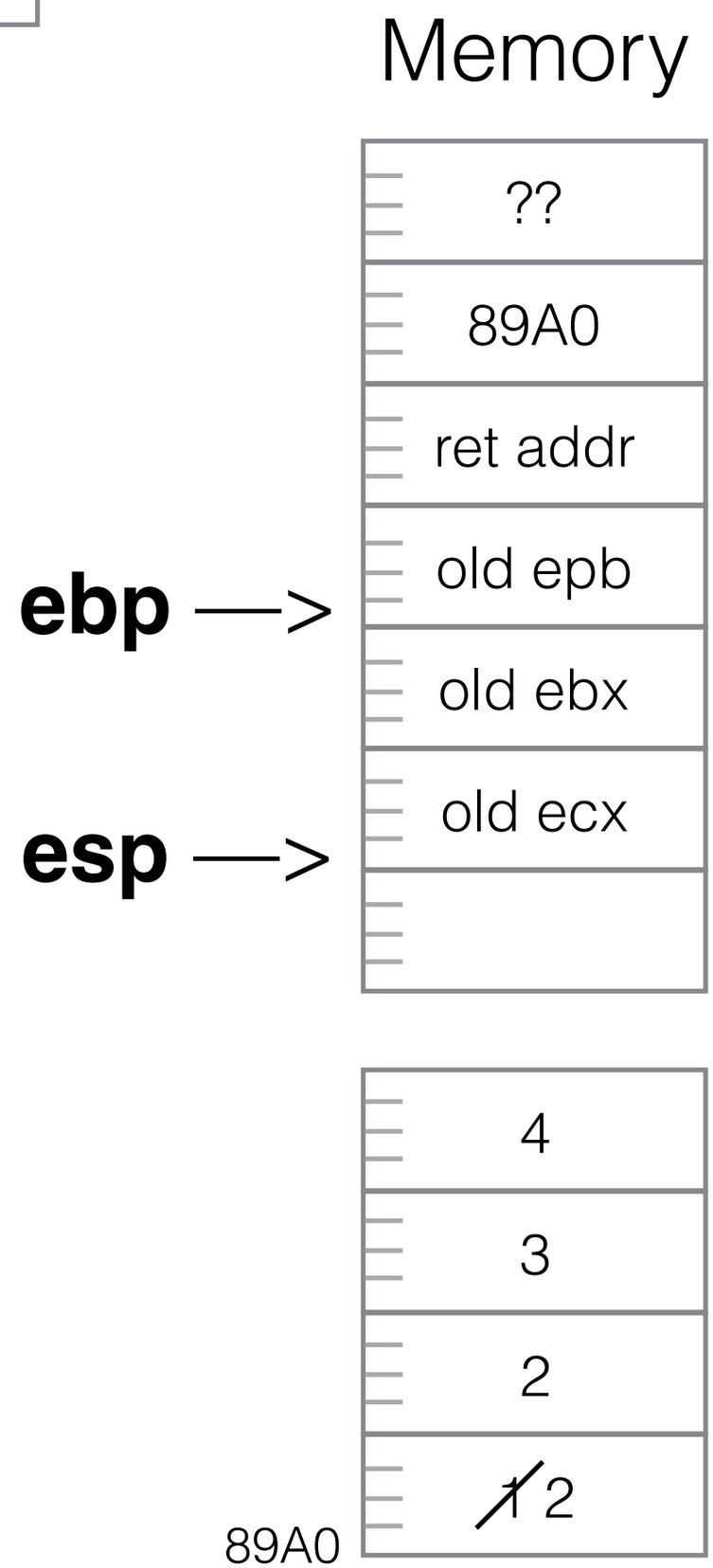
eax 89A0 **ebx** 89A0 **ecx** 4

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4          <— eip
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx
```



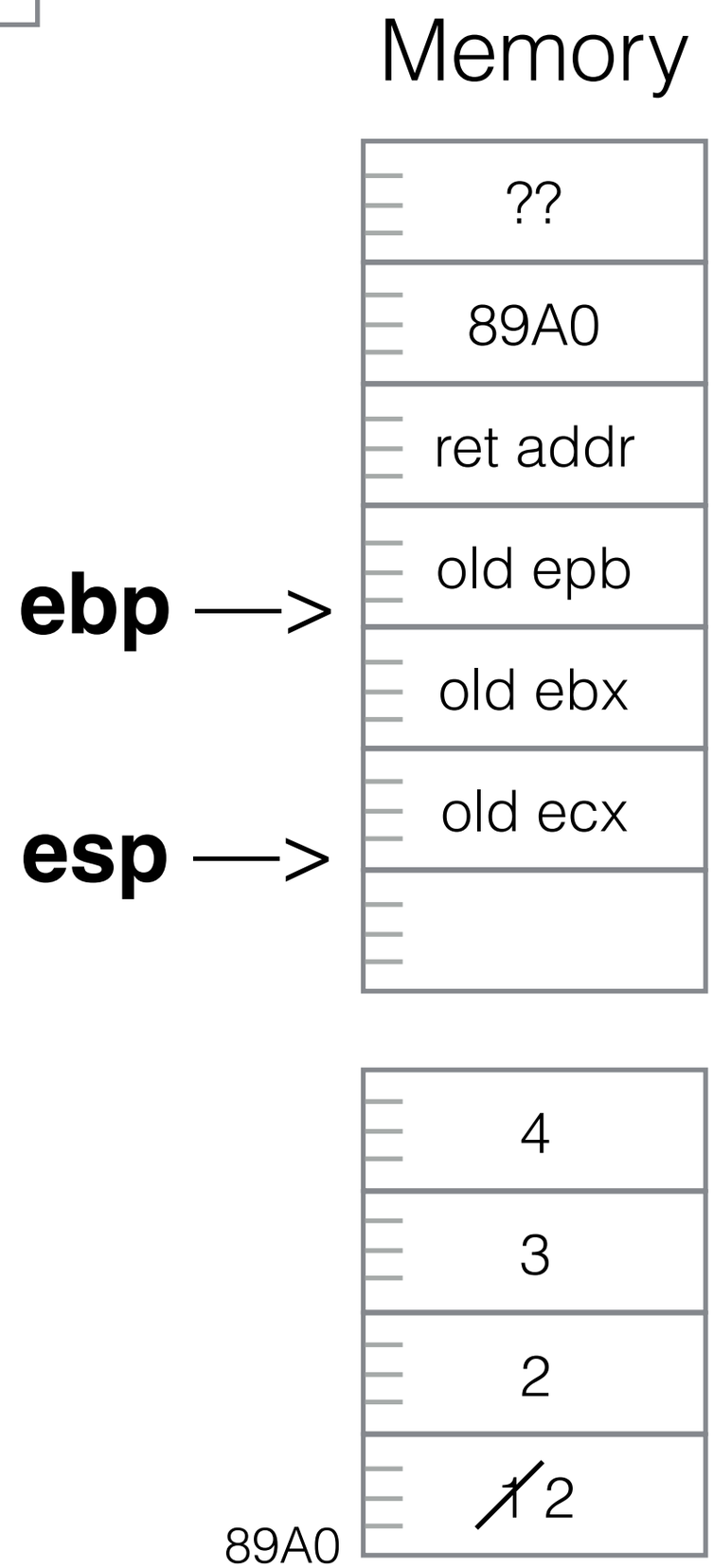
eax 89A0 **ebx** 89A4 **ecx** 4

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for            <— eip
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx
```



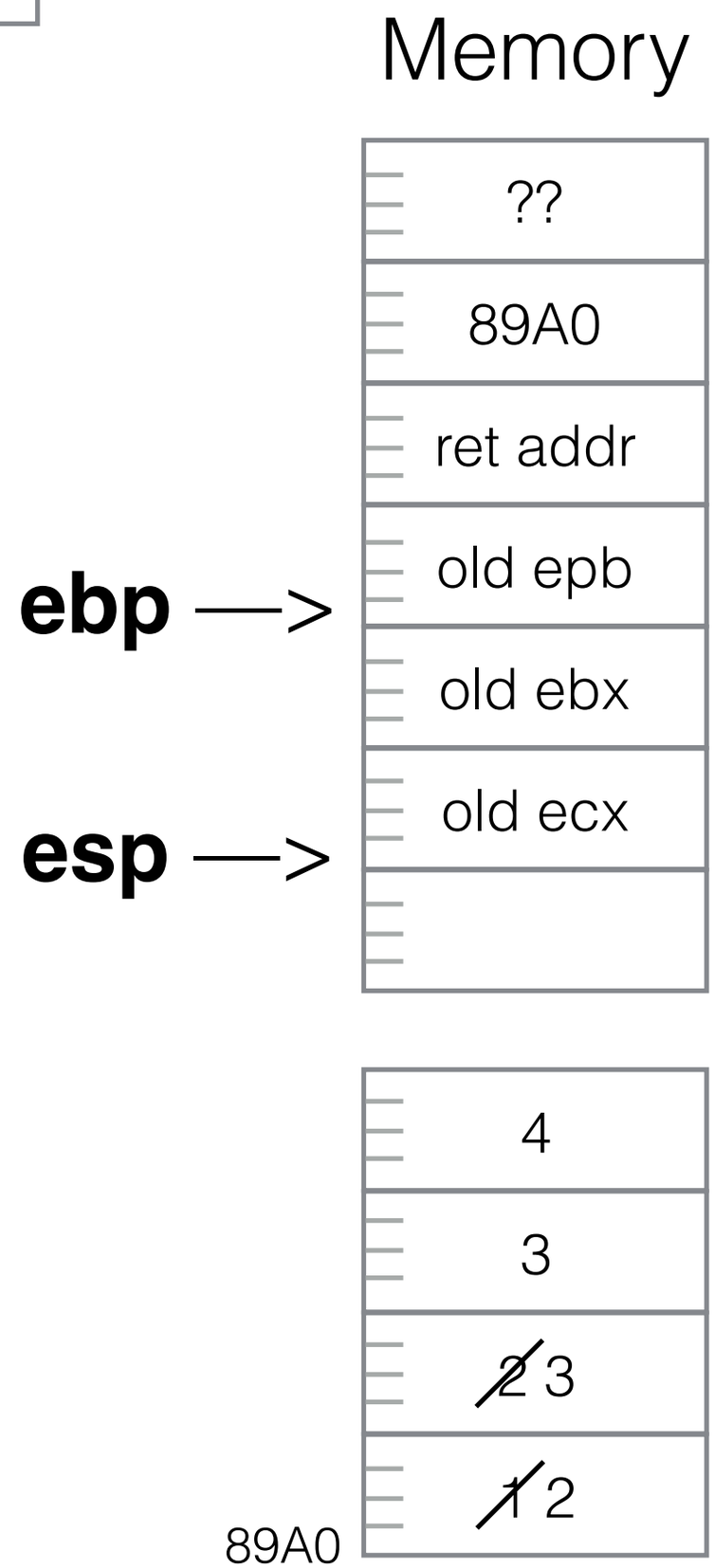
eax 89A0 **ebx** 89A4 **ecx** ~~4~~ 3

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]          <— eip
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx
```



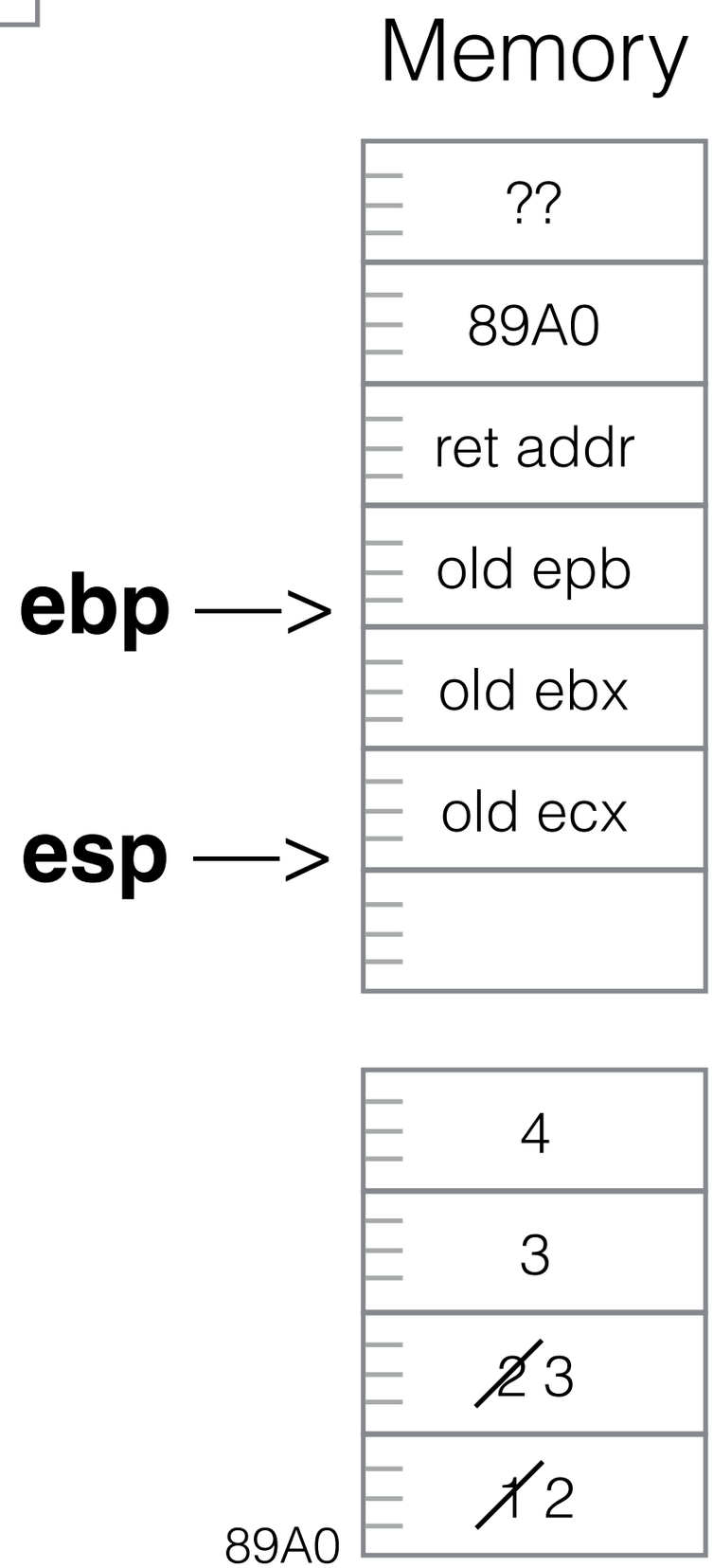
eax 89A0 **ebx** 89A4 **ecx** ~~4~~ 3

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4          <— eip
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx
```



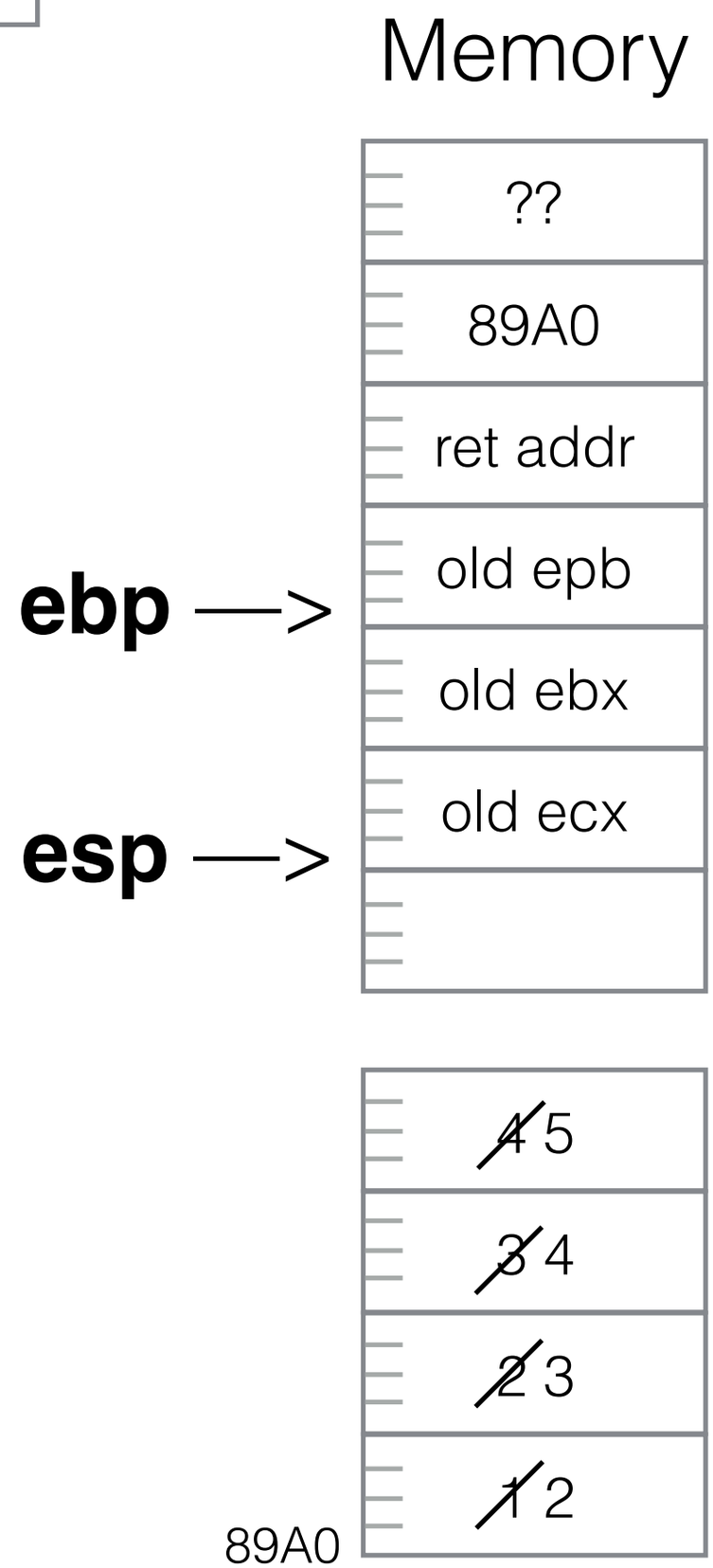
eax 89A0 **ebx** 89A8 **ecx** ~~4~~ 3

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for        <— eip
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx
```



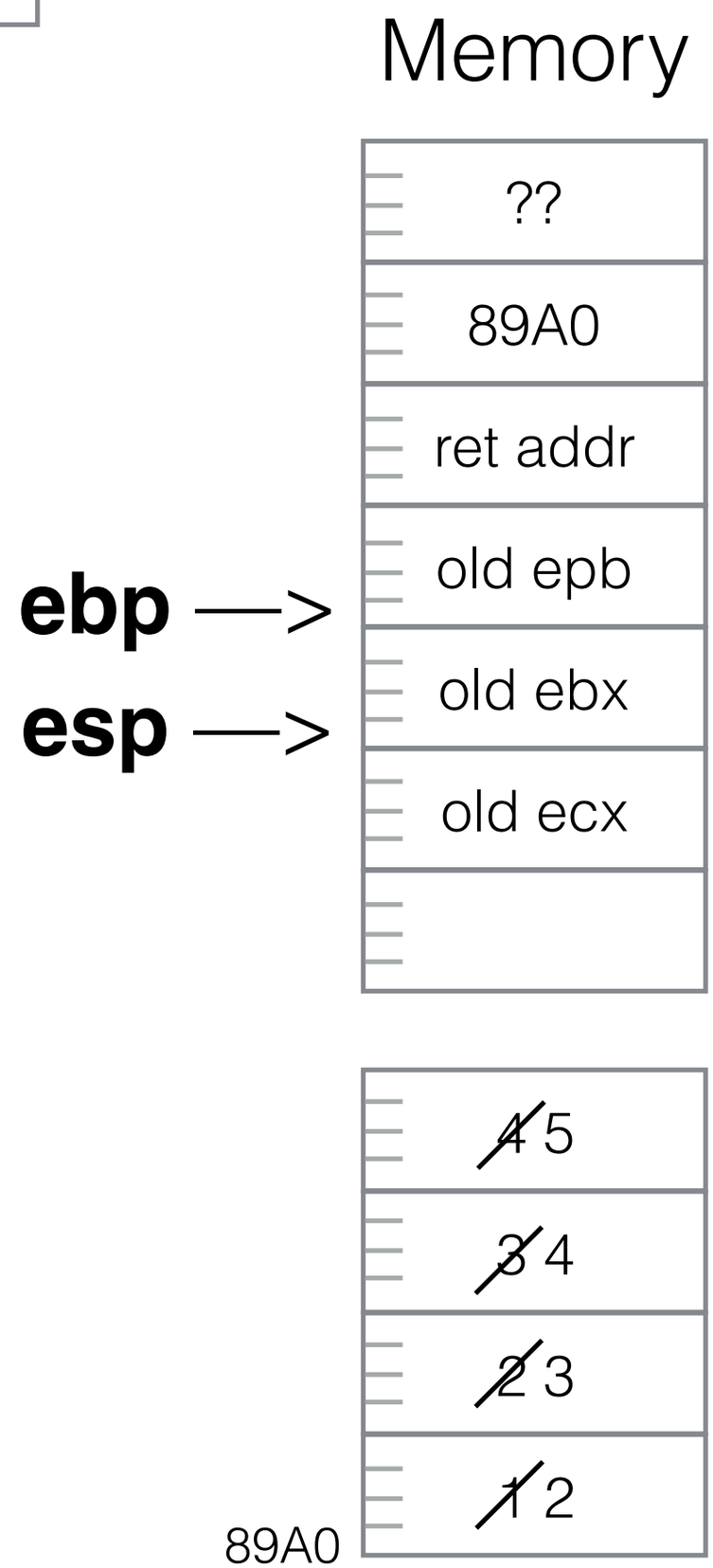
eax 89A0 **ebx** 89B2 **ecx** ~~4~~ ~~3~~ ~~2~~ ~~1~~ 0

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx             <— eip
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx
```



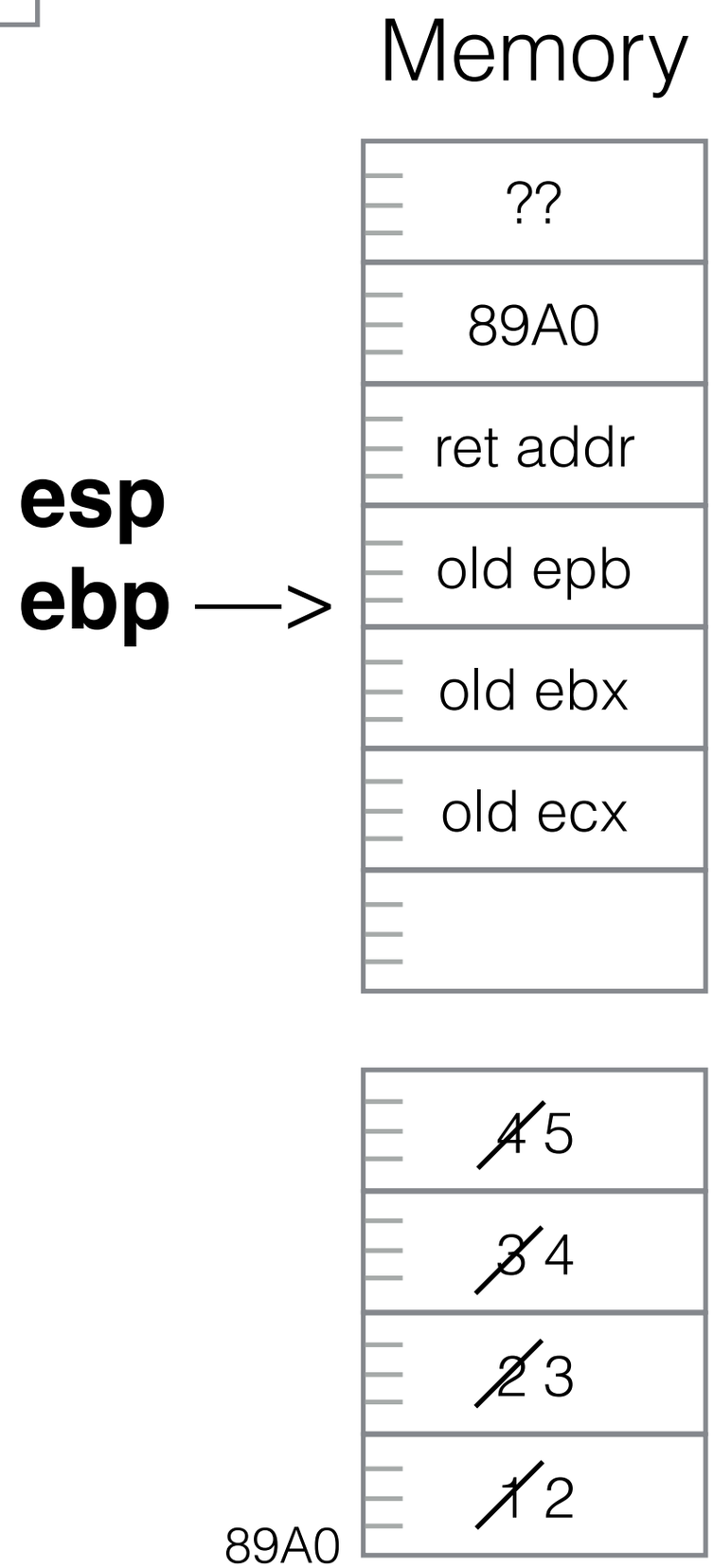
eax 89A0 **ebx** 89B2 **ecx** ??

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx          <— eip
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx
```



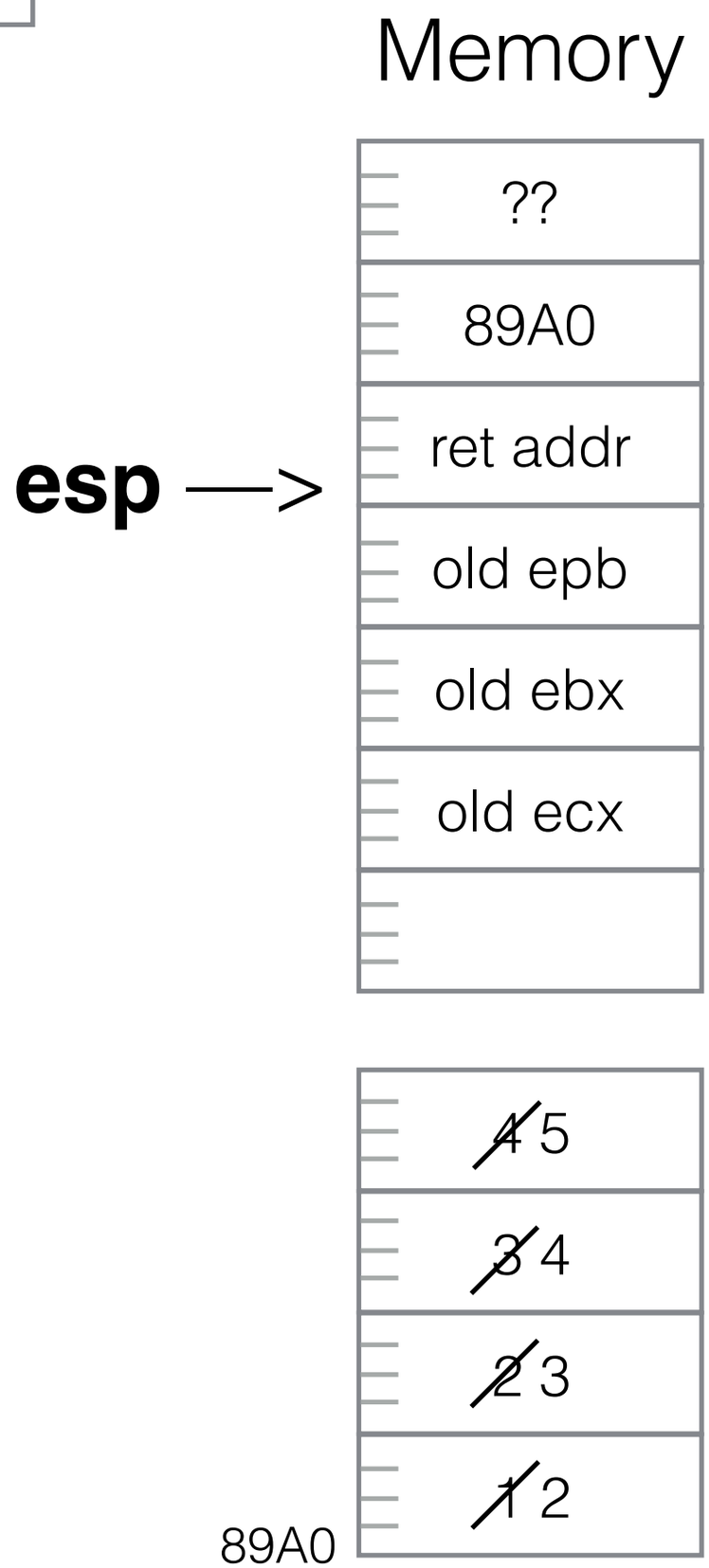
eax 89A0 **ebx** ?? **ecx** ??

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp             <— eip
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx
```



eax 89A0 **ebx** ?? **ecx** ??

```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4               <— eip


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx
```

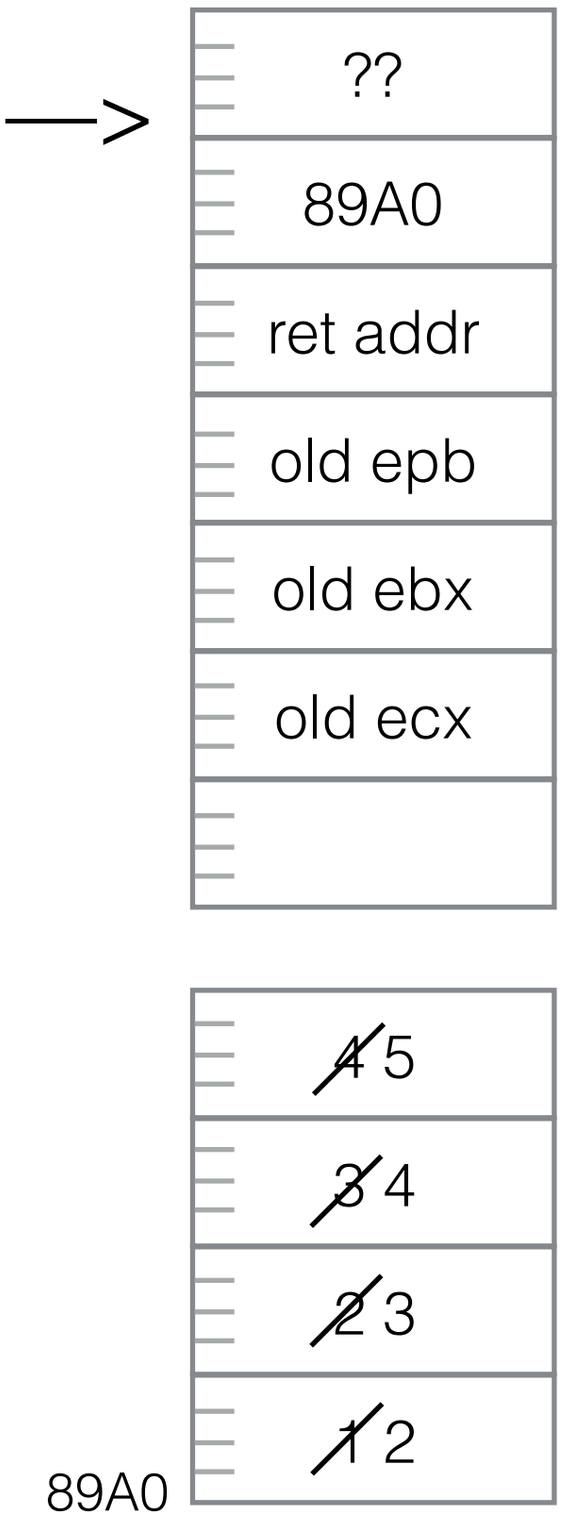


eax 89A0 **ebx** ?? **ecx** ??

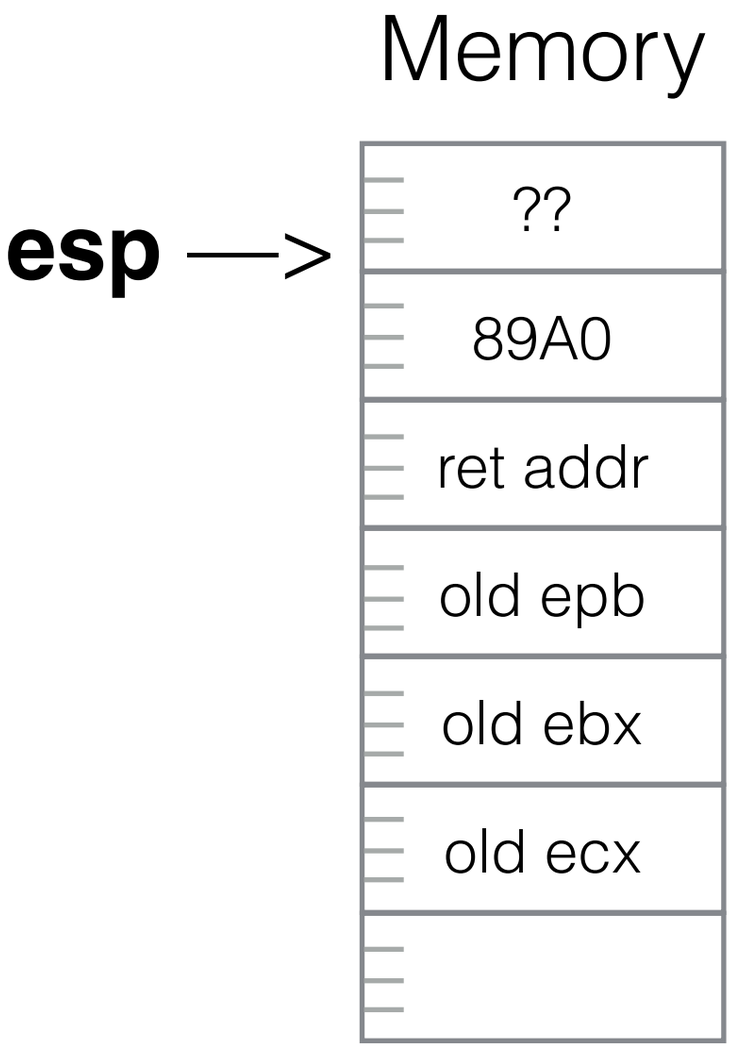
```
                section .data
Table           dd      1,2,3,4

                section .text
incrementAll:
                push    ebp
                mov     ebp, esp
                push    ebx
                push    ecx
                mov     ecx, 4
                mov     ebx, dword[ebp+8]
.for:           inc     dword[ebx]
                add     ebx, 4
                loop    .for
                pop     ecx
                pop     ebx
                pop     ebp
                ret     4


_Start:
                mov     eax, Table
                push    eax
                call    incrementAll
                xxx                          <— eip
```

Memory

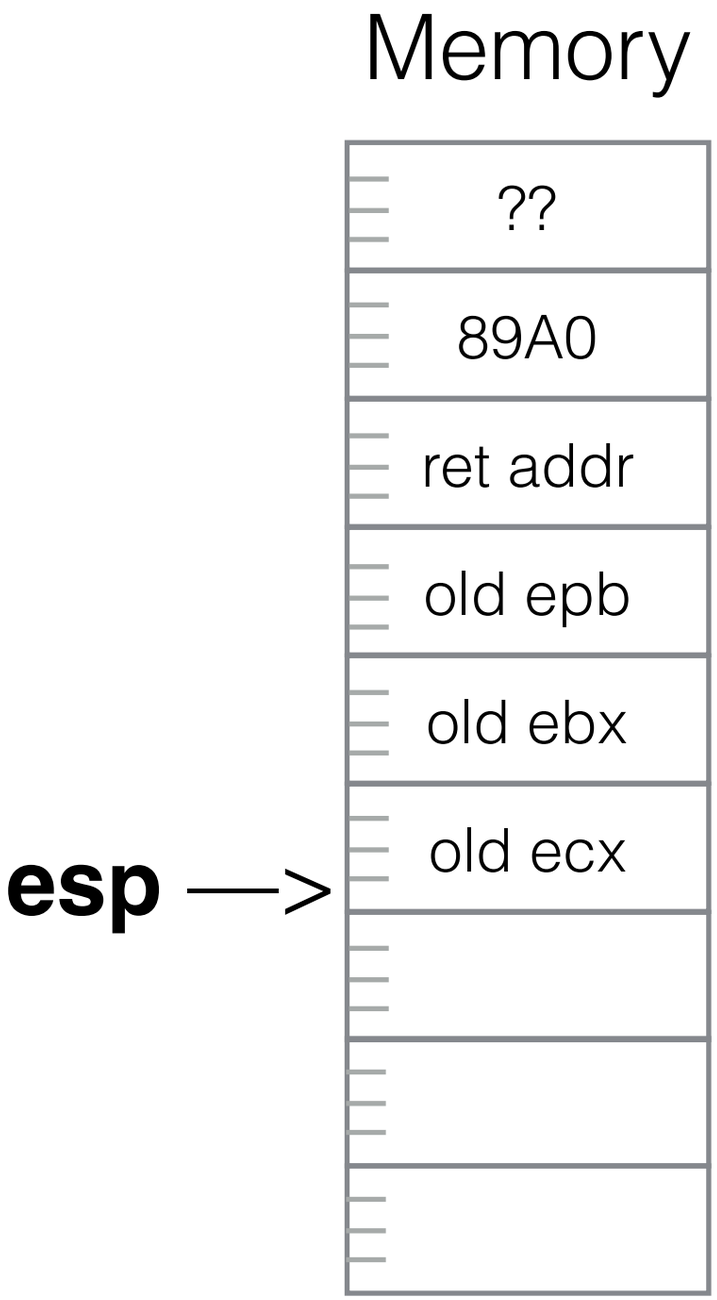
esp —>



- Passing through **registers** ✓
- Passing through the **stack** ✓
- Passing by **Value** ✓
- Passing by **Reference** ✓

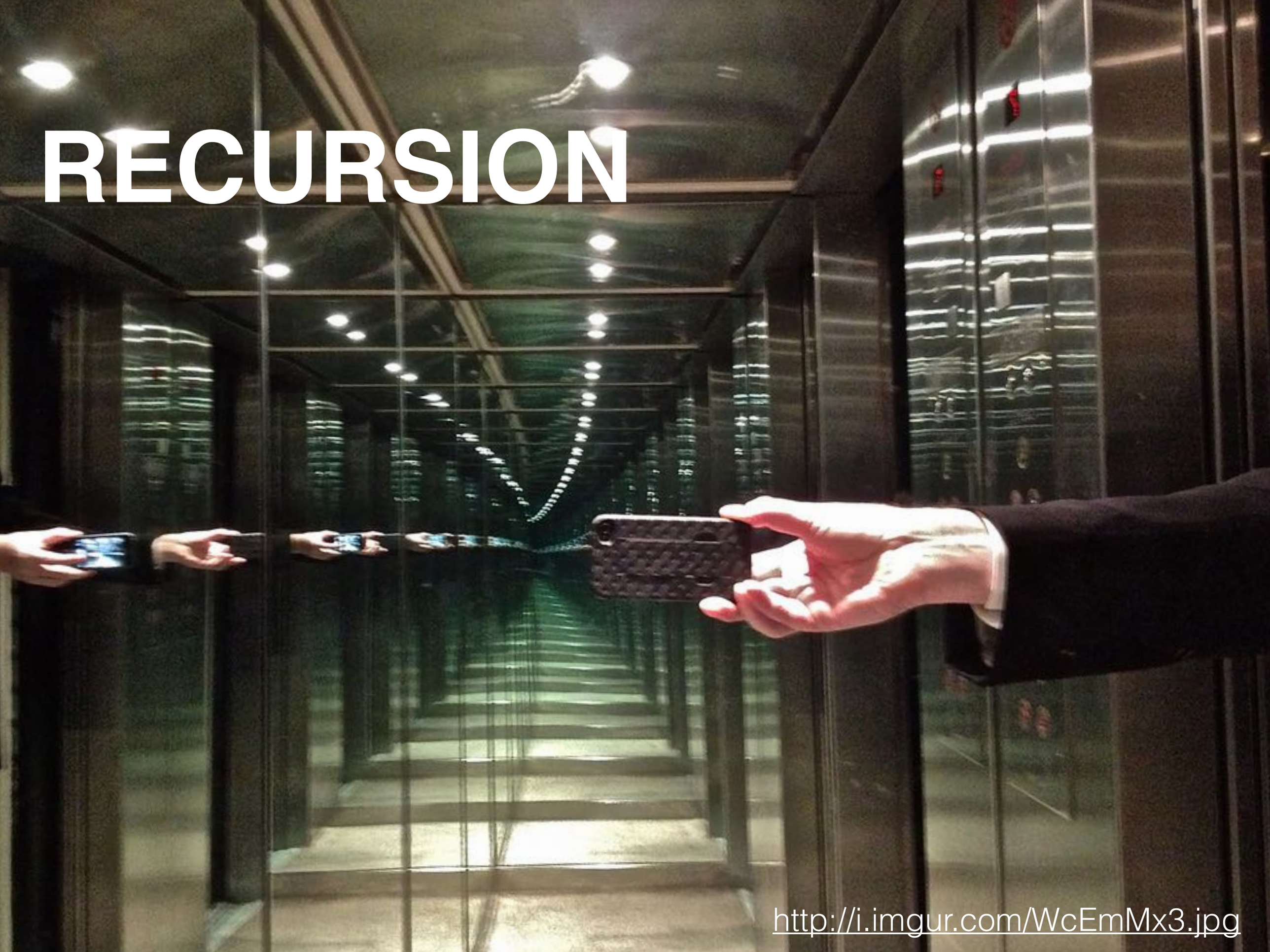


Question 1: What happens to the data "below" esp? Can it be used?



Question 2: What about local variables?

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
>>>
```

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

Write the function **fact(n)** and call it from **main()**, *in Assembly*

Compare to Non-Recursive Version

```
;;; ------------------------------------------------------------
;;; fact:         computes the factorial of n passed in eax
;;;               and returns result in eax
;;; ------------------------------------------------------------
fact:   push    ebp             ; create stack frame
        mov     ebp, esp        ; point to it

        push    edx             ; save what we use
        push    ecx
        push    edx

        mov     ecx, eax        ; loop N times
        mov     eax, 1          ; product = 1
.for:   mul     ecx             ; product *= ecx--
        loop    .for

        pop     edx             ; restore what we used
        pop     ecx
        pop     edx

        pop     ebp             ; return
        ret
```

Question 1

- Compare the execution time of the recursive version of ***factorial()*** to its non-recursive version.

Question 2

- If the maximum stack size given to a program is 8 GBytes, how many terms could ***fact()*** compute, at most, if we didn't care about multiplication overflow?

Note: We can get the default stack size linux uses with

```
ulimit -s
```