Introduction to 8086 Assembly

Lecture 14

Recursion

Recursion

factorial.asm

```
        :
;; compute fact(4)
push 4
call fact
add esp, 4

call print_int
call print_nl
        :
```

factorial.asm (cont.)

```
fact:
        push ebp
        mov  ebp, esp

        mov eax, [ebp+8]
        cmp eax, 0
        jg  recur

        mov eax, 1
        jmp endfact

recur:
        dec eax
        push eax
        call fact
        add esp, 4

        imul dword [ebp+8]

endfact:
        pop ebp
        ret
```

K. N. Toosi
University of Technology

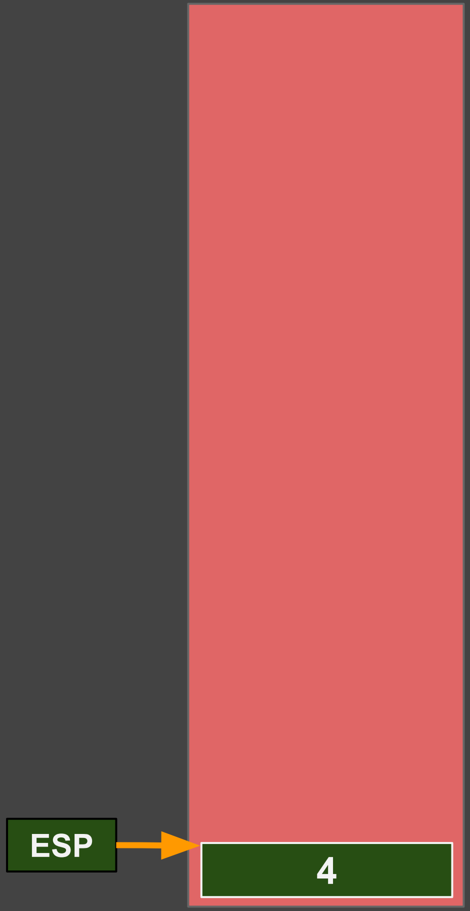
Recursion

factorial.asm

```
        :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
        :
```

factorial.asm (cont.)

```
fact:
     push ebp
     mov  ebp, esp

     mov eax, [ebp+8]
     cmp eax, 0
     jg  recur

     mov eax, 1
     jmp endfact

recur:
     dec eax
     push eax
     call fact
L2:  add esp, 4

     imul dword [ebp+8]

endfact:
     pop ebp
     ret
```



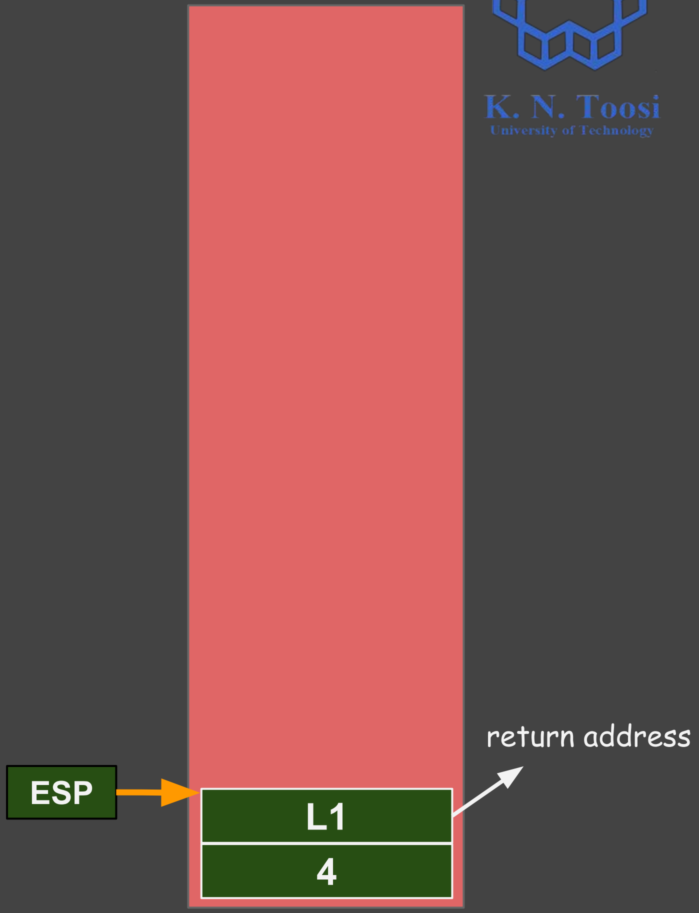
Recursion

factorial.asm

```
            :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
            :
```

factorial.asm (cont.)

```
fact:
    push ebp
    mov  ebp, esp

    mov eax, [ebp+8]
    cmp eax, 0
    jg  recur

    mov eax, 1
    jmp endfact

recur:
    dec eax
    push eax
    call fact
L2: add esp, 4

    imul dword [ebp+8]

endfact:
    pop ebp
    ret
```



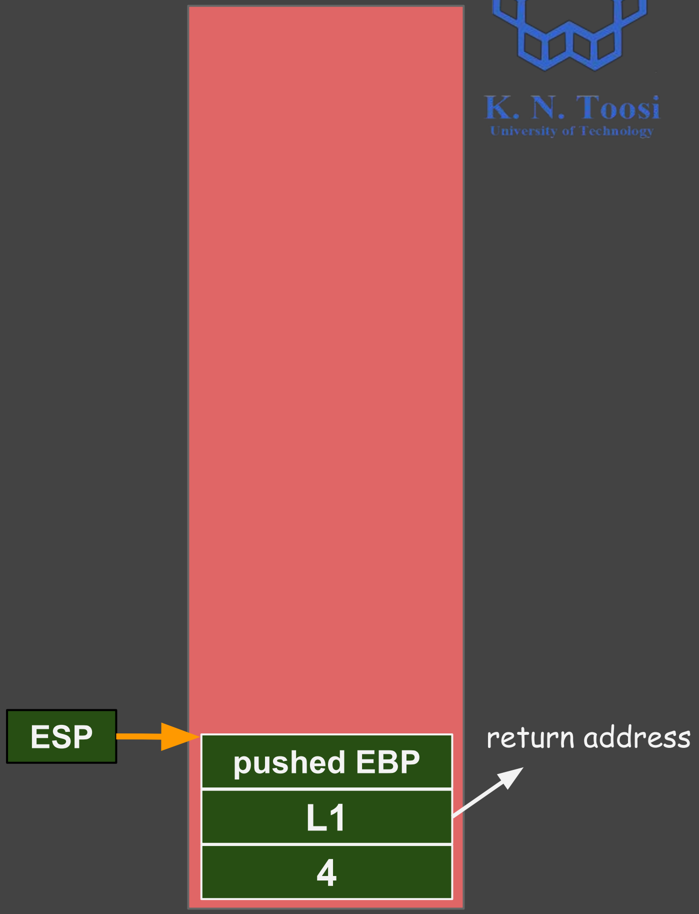
Recursion

factorial.asm

```
            :
      ;; compute fact(4)
      push 4
      call fact
L1:   add esp, 4

      call print_int
      call print_nl
            :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
      jg  recur

      mov eax, 1
      jmp endfact

recur:
      dec eax
      push eax
      call fact
L2:   add esp, 4

      imul dword [ebp+8]

endfact:
      pop ebp
      ret
```



K. N. Toosi
University of Technology

Recursion

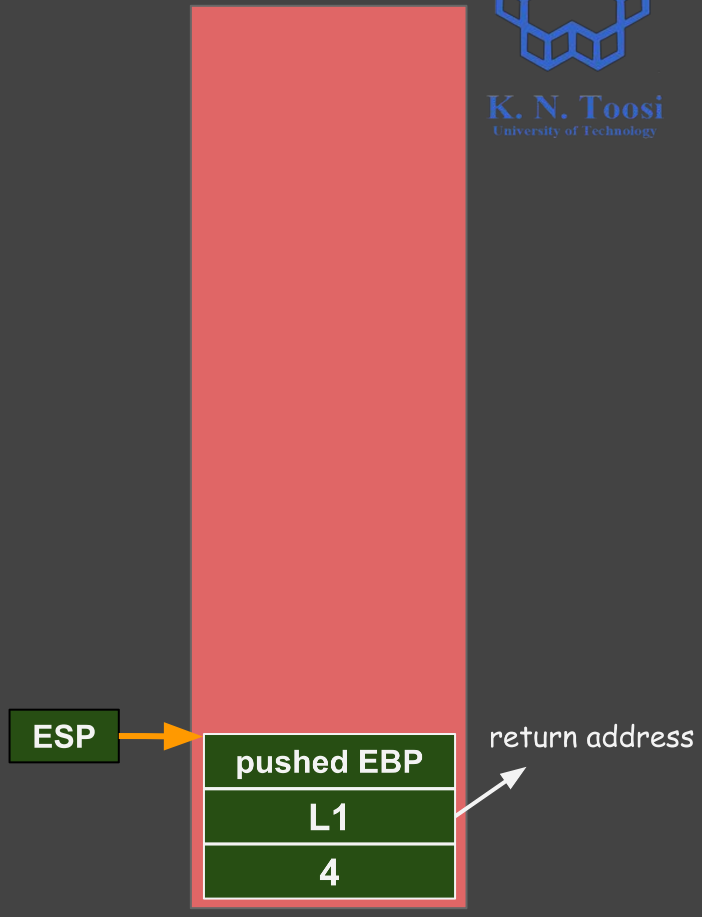
factorial.asm

```
            :
      ;; compute fact(4)
      push 4
      call fact
L1:   add esp, 4

      call print_int
      call print_nl
            :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
→     jg  recur

      mov eax, 1
      jmp endfact

recur:
      dec eax
      push eax
      call fact
L2:   add esp, 4

      imul dword [ebp+8]

endfact:
      pop ebp
      ret
```

EAX=4



Recursion

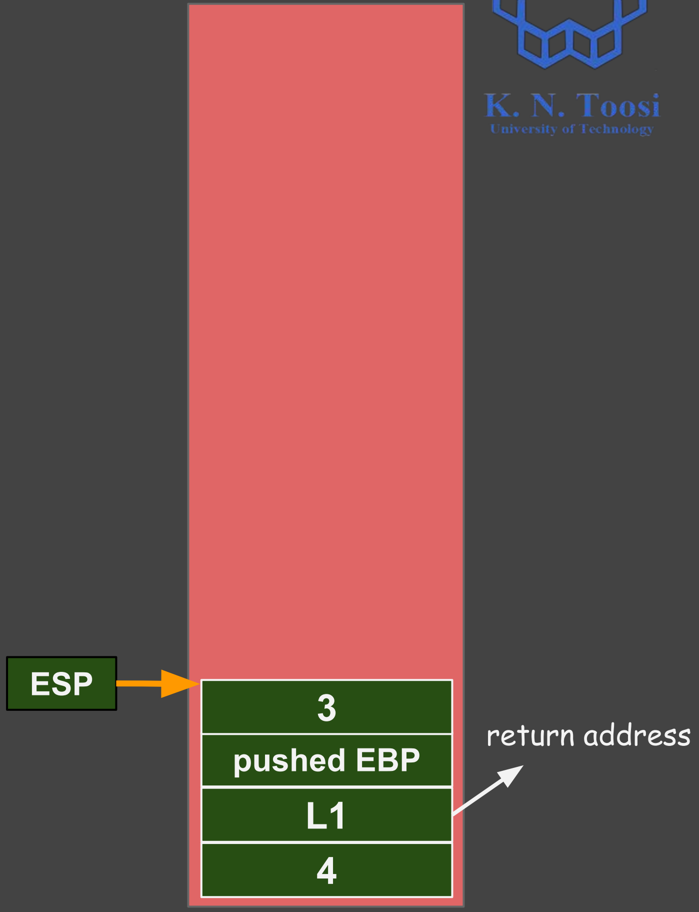
factorial.asm

```
        :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
        :
```

factorial.asm (cont.)

```
fact:
    push ebp
    mov  ebp, esp

    mov eax, [ebp+8]
    cmp eax, 0
    jg  recur

    mov eax, 1
    jmp endfact

recur:
    dec eax
    push eax
    call fact
L2: add esp, 4

    imul dword [ebp+8]

endfact:
    pop ebp
    ret
```

EAX=3



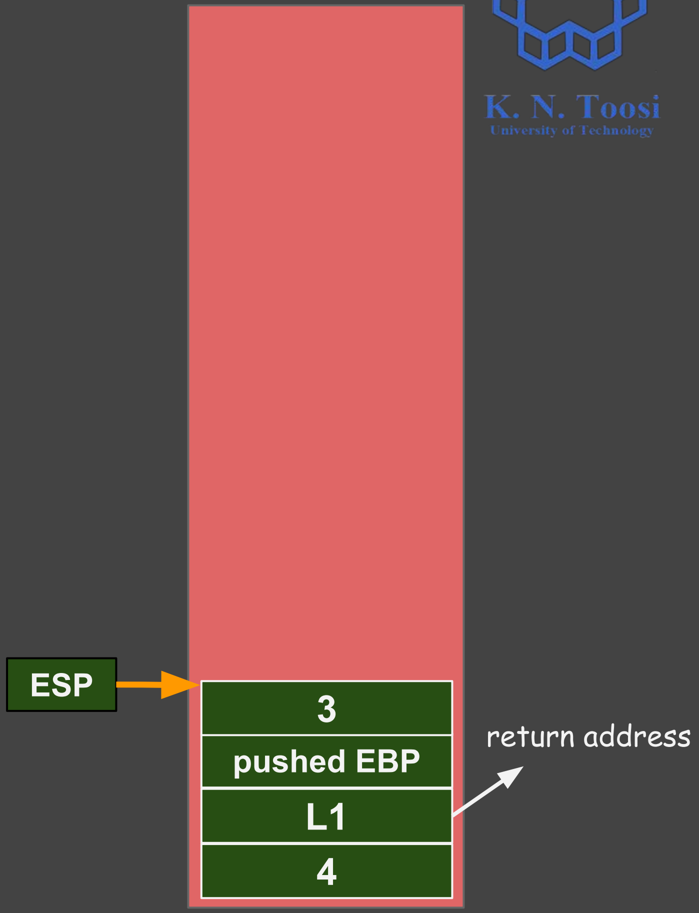
Recursion

factorial.asm

```
          :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
          :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
      jg  recur

      mov eax, 1
      jmp endfact

recur:
      dec eax
      push eax
  →   call fact
L2:   add esp, 4

      imul dword [ebp+8]

endfact:
      pop ebp
      ret
```



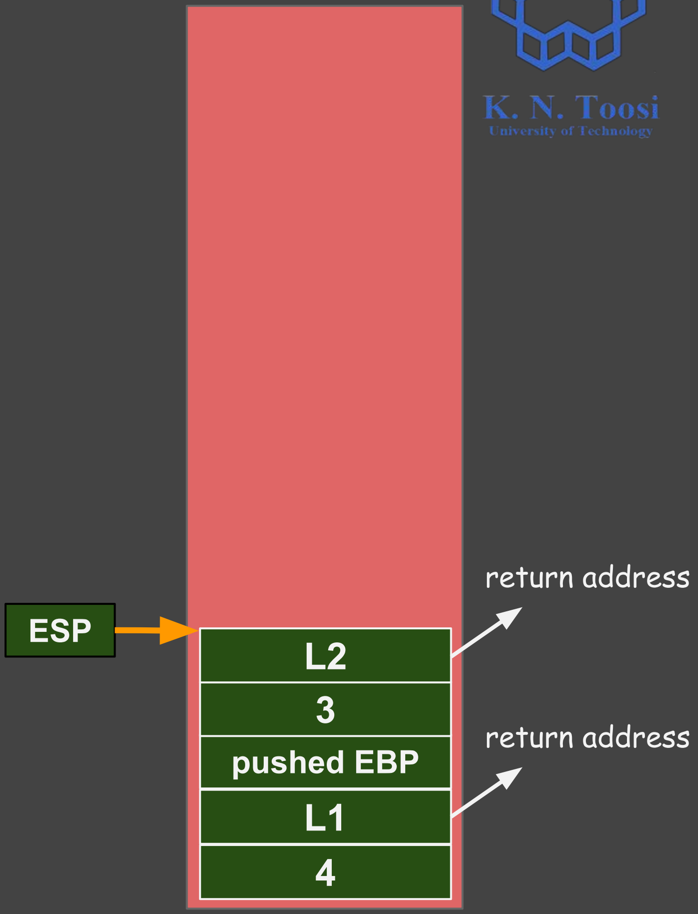
Recursion

factorial.asm

```
          :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
          :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
      jg  recur

      mov eax, 1
      jmp endfact

recur:
      dec eax
      push eax
      call fact
L2:   add esp, 4

      imul dword [ebp+8]

endfact:
      pop ebp
      ret
```



ESP

L2

3

pushed EBP

L1

4

return address

return address

Recursion

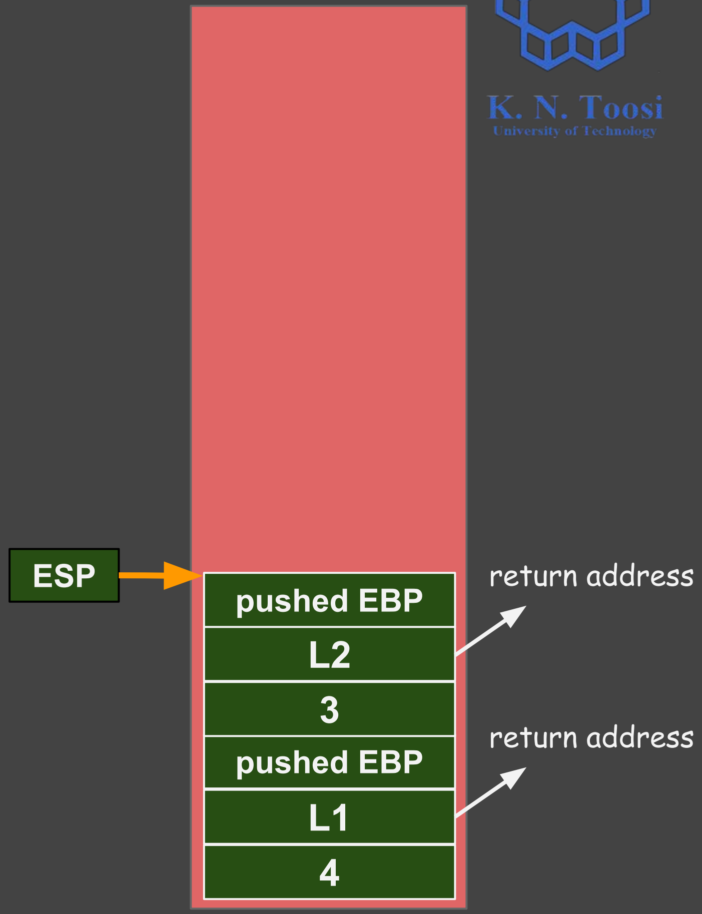
factorial.asm

```
        :
  ;; compute fact(4)
  push 4
  call fact
L1: add esp, 4

  call print_int
  call print_nl
        :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
    → jg  recur

      mov eax, 1
      jmp endfact

recur:
      dec eax
      push eax
      call fact
L2:   add esp, 4

      imul dword [ebp+8]

endfact:
      pop ebp
      ret
```

EAX=3



Recursion

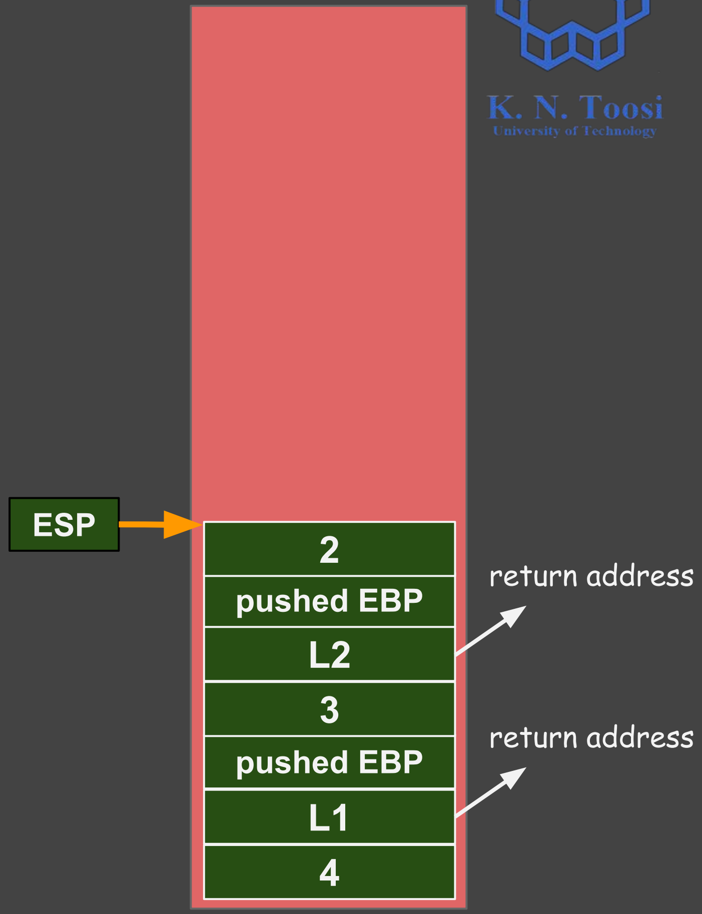
factorial.asm

```
          :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
          :
```

factorial.asm (cont.)

```
fact:
    push ebp
    mov  ebp, esp

    mov eax, [ebp+8]
    cmp eax, 0
    jg  recur

    mov eax, 1
    jmp endfact

recur:
    dec eax
    push eax
    call fact
L2: add esp, 4

    imul dword [ebp+8]

endfact:
    pop ebp
    ret
```

EAX=2



ESP

2
pushed EBP
L2
3
pushed EBP
L1
4

return address

return address

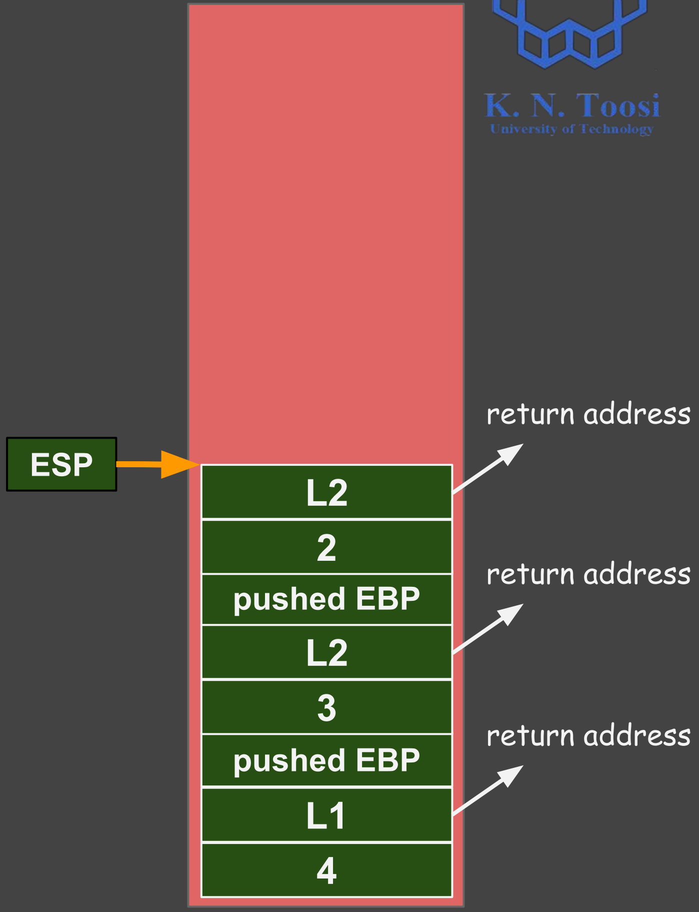
Recursion

factorial.asm

```
            :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
            :
```

factorial.asm (cont.)

```
fact:
    push ebp
    mov  ebp, esp

    mov eax, [ebp+8]
    cmp eax, 0
    jg  recur

    mov eax, 1
    jmp endfact

recur:
    dec eax
    push eax
    call fact
L2: add esp, 4

    imul dword [ebp+8]

endfact:
    pop ebp
    ret
```



L2
2
pushed EBP
L2
3
pushed EBP
L1
4

K. N. Toosi
University of Technology

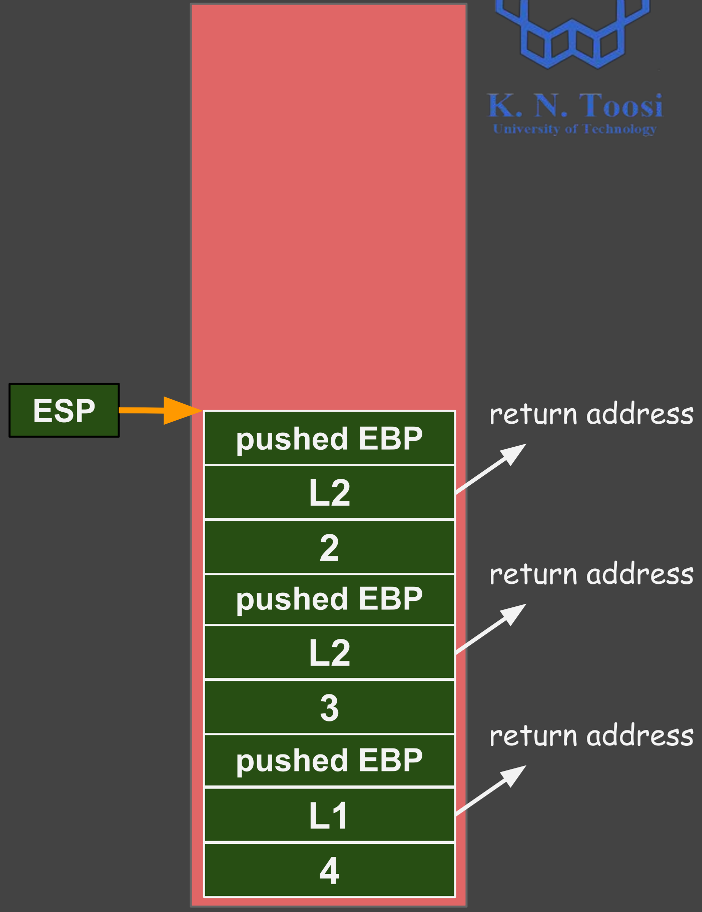
Recursion

factorial.asm

```
            :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
            :
```

factorial.asm (cont.)

```
fact:
        push ebp
        mov  ebp, esp

        mov eax, [ebp+8]
        cmp eax, 0
        jg  recur

        mov eax, 1
        jmp endfact

recur:
        dec eax
        push eax
        call fact
L2:     add esp, 4

        imul dword [ebp+8]

endfact:
        pop ebp
        ret
```



K. N. Toosi
University of Technology

Recursion

K. N. Toosi
University of Technology

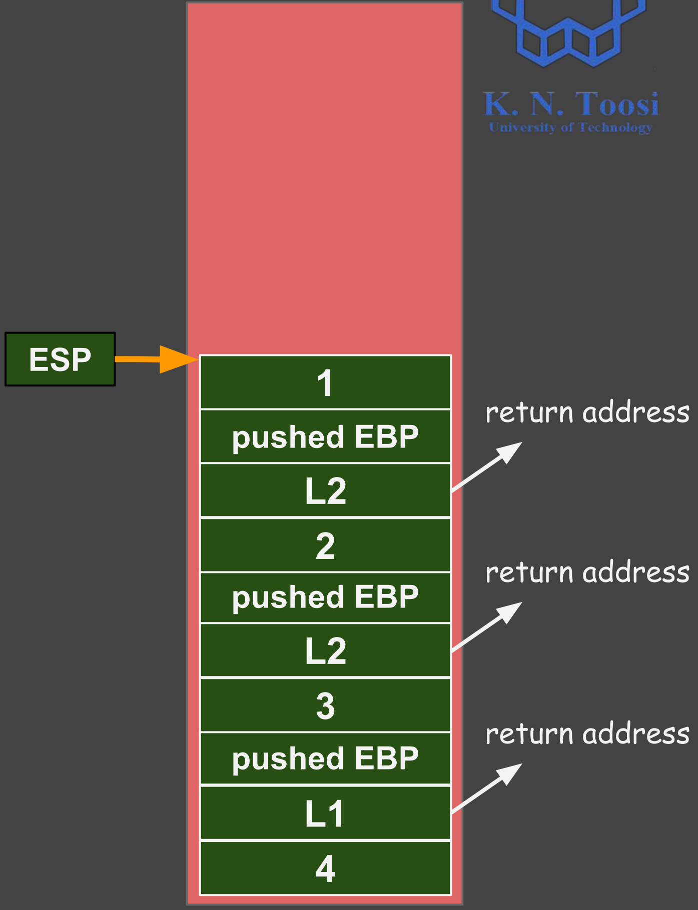
factorial.asm

```
            :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
            :
```

factorial.asm (cont.)

```
fact:
    push ebp
    mov  ebp, esp

    mov eax, [ebp+8]
    cmp eax, 0
    jg  recur

    mov eax, 1
    jmp endfact

recur:
    dec eax
    push eax
    call fact
L2: add esp, 4

    imul dword [ebp+8]

endfact:
    pop ebp
    ret
```

EAX=1



Recursion

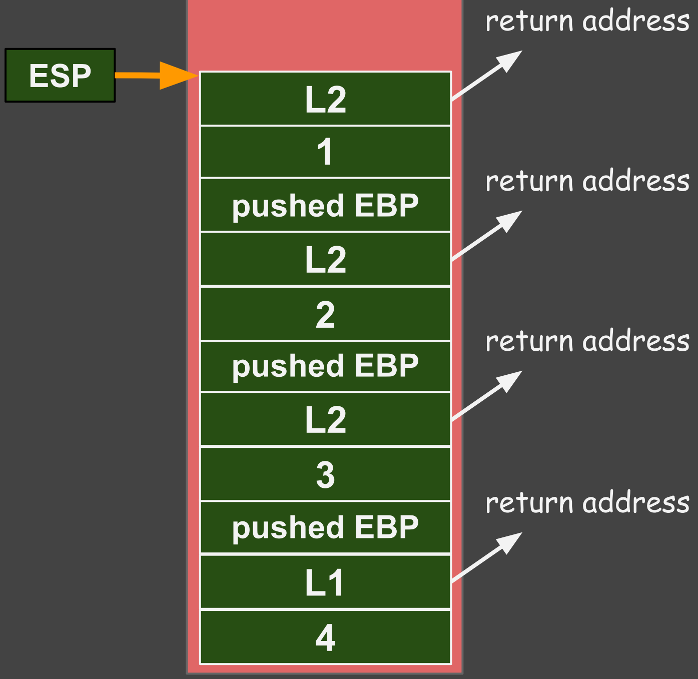
K. N. Toosi
University of Technology

factorial.asm

```
          :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
          :
```

factorial.asm (cont.)

```
fact:
    push ebp
    mov  ebp, esp

    mov eax, [ebp+8]
    cmp eax, 0
    jg  recur

    mov eax, 1
    jmp endfact

recur:
    dec eax
    push eax
    call fact
L2: add esp, 4

    imul dword [ebp+8]

endfact:
    pop ebp
    ret
```



Recursion

K. N. Toosi
University of Technology

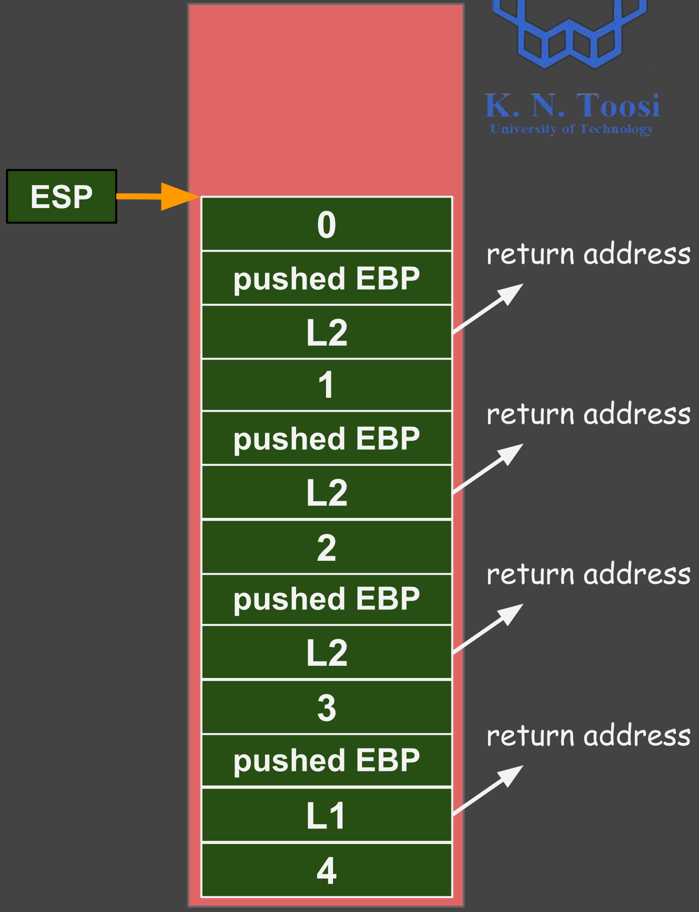
factorial.asm

```
        :
   ;; compute fact(4)
   push 4
   call fact
L1: add esp, 4

   call print_int
   call print_nl
        :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
      jg  recur

      mov eax, 1
      jmp endfact

recur:
      dec eax
      push eax
      call fact
L2:   add esp, 4

      imul dword [ebp+8]

endfact:
      pop ebp
      ret
```

EAX=0



ESP

0
pushed EBP
L2
1
pushed EBP
L2
2
pushed EBP
L2
3
pushed EBP
L1
4

return address

return address

return address

return address

Recursion

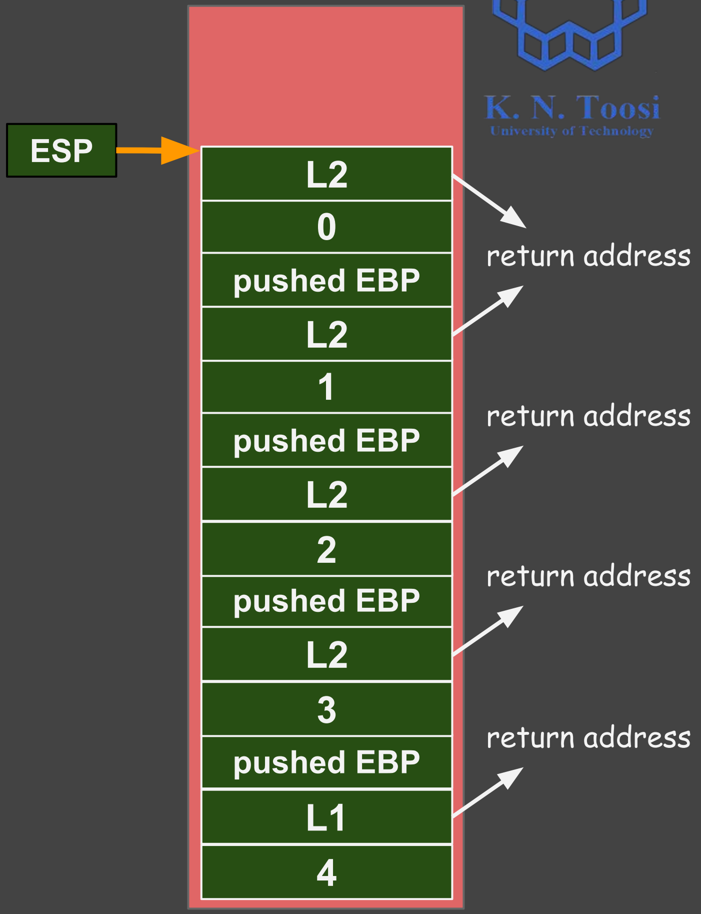
K. N. Toosi
University of Technology

factorial.asm

```
            :
      ;; compute fact(4)
      push 4
      call fact
L1:   add esp, 4

      call print_int
      call print_nl
            :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
      jg  recur

      mov eax, 1
      jmp endfact

recur:
      dec eax
      push eax
      call fact
L2:   add esp, 4

      imul dword [ebp+8]

endfact:
      pop ebp
      ret
```



Recursion

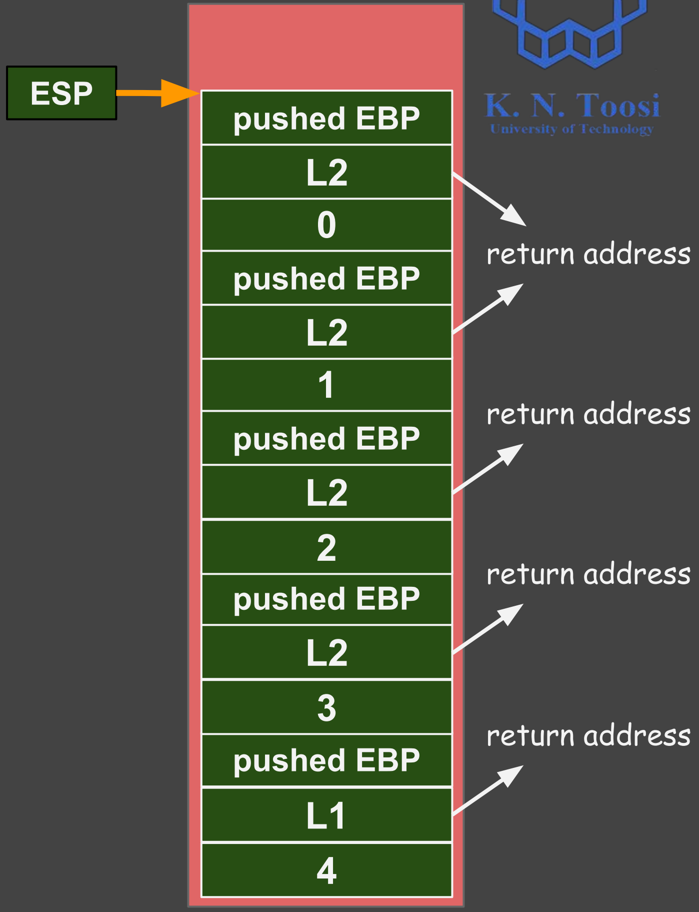
K. N. Toosi
University of Technology

factorial.asm

```
            :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
            :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
      jg  recur

      mov eax, 1
      jmp endfact

recur:
      dec eax
      push eax
      call fact
L2:   add esp, 4

      imul dword [ebp+8]

endfact:
      pop ebp
      ret
```



ESP

pushed EBP
L2
0
pushed EBP
L2
1
pushed EBP
L2
2
pushed EBP
L2
3
pushed EBP
L1
4

return address
return address
return address
return address

Recursion

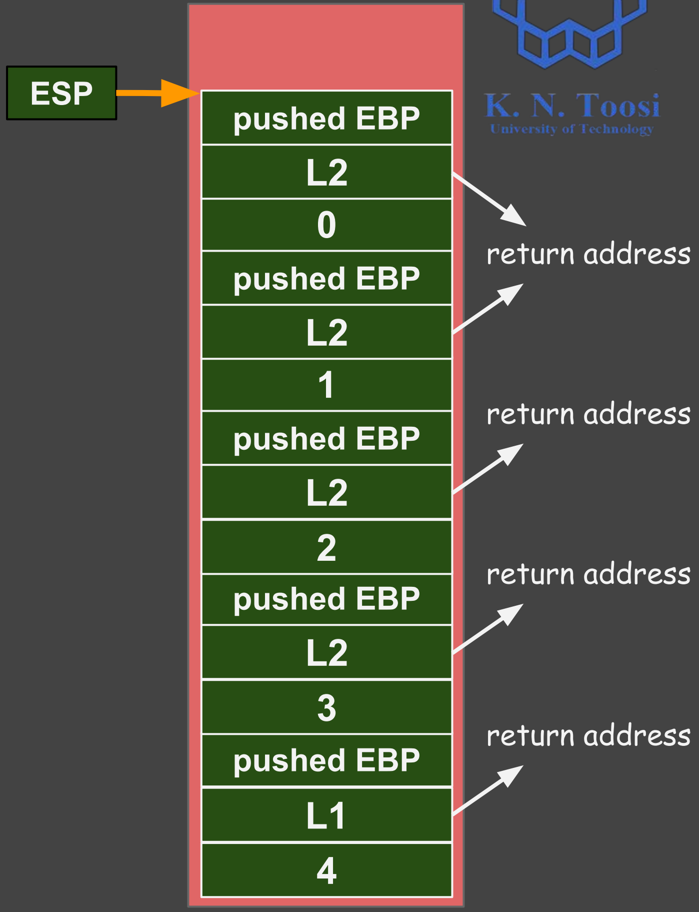
factorial.asm

```
            :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
            :
```

factorial.asm (cont.)

```
fact:
     push ebp
     mov  ebp, esp

     mov eax, [ebp+8]
     cmp eax, 0
     jg  recur

     mov eax, 1
     jmp endfact

recur:
     dec eax
     push eax
     call fact
L2:  add esp, 4

     imul dword [ebp+8]

endfact:
     pop ebp
     ret
```

EAX=0



K. N. Toosi
University of Technology

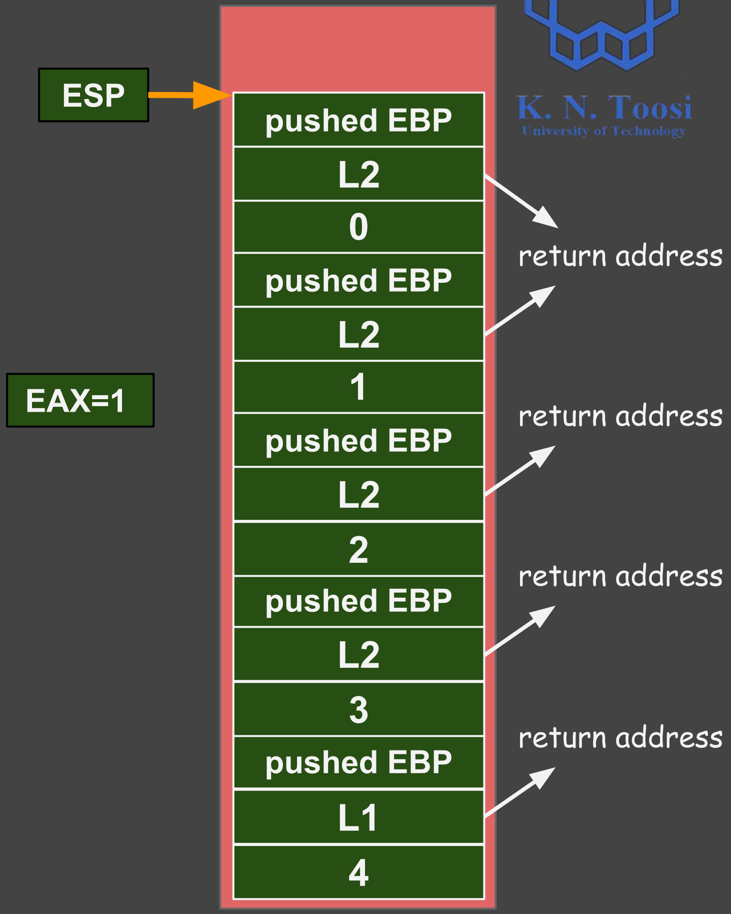
Recursion

factorial.asm

```
            :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
            :
```

factorial.asm (cont.)

```
fact:
    push ebp
    mov  ebp, esp

    mov eax, [ebp+8]
    cmp eax, 0
    jg  recur

    mov eax, 1
    jmp endfact

recur:
    dec eax
    push eax
    call fact
L2: add esp, 4

    imul dword [ebp+8]

endfact:
    pop ebp
    ret
```



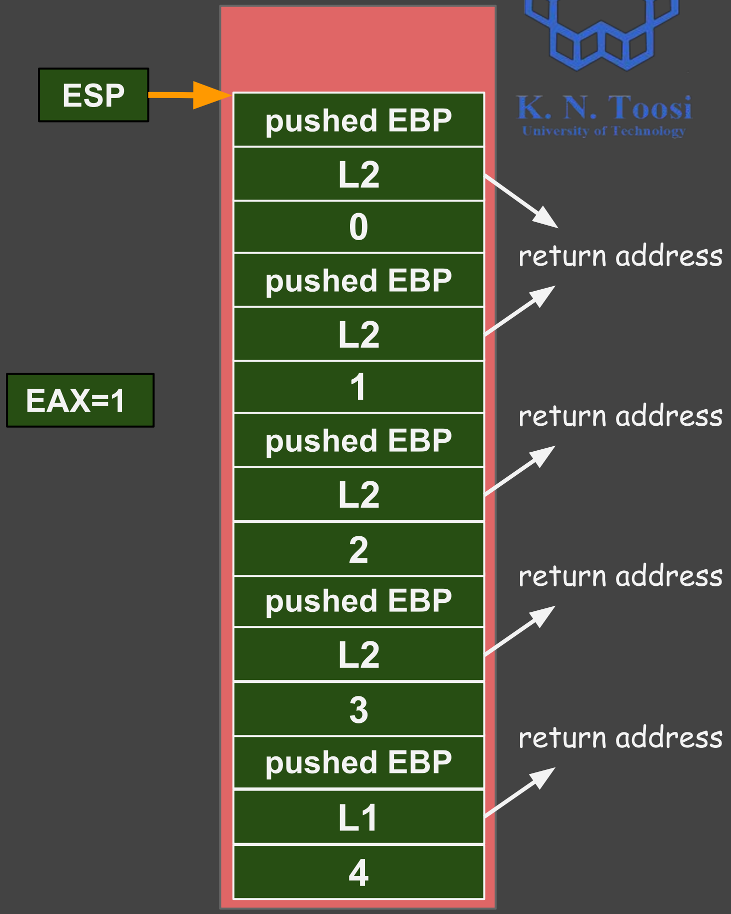
Recursion

factorial.asm

```
            :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
            :
```

factorial.asm (cont.)

```
fact:
    push ebp
    mov  ebp, esp

    mov eax, [ebp+8]
    cmp eax, 0
    jg  recur

    mov eax, 1
    jmp endfact

recur:
    dec eax
    push eax
    call fact
L2: add esp, 4

    imul dword [ebp+8]

endfact:
    pop ebp
    ret
```



ESP

EAX=1

pushed EBP
L2
0
pushed EBP
L2
1
pushed EBP
L2
2
pushed EBP
L2
3
pushed EBP
L1
4

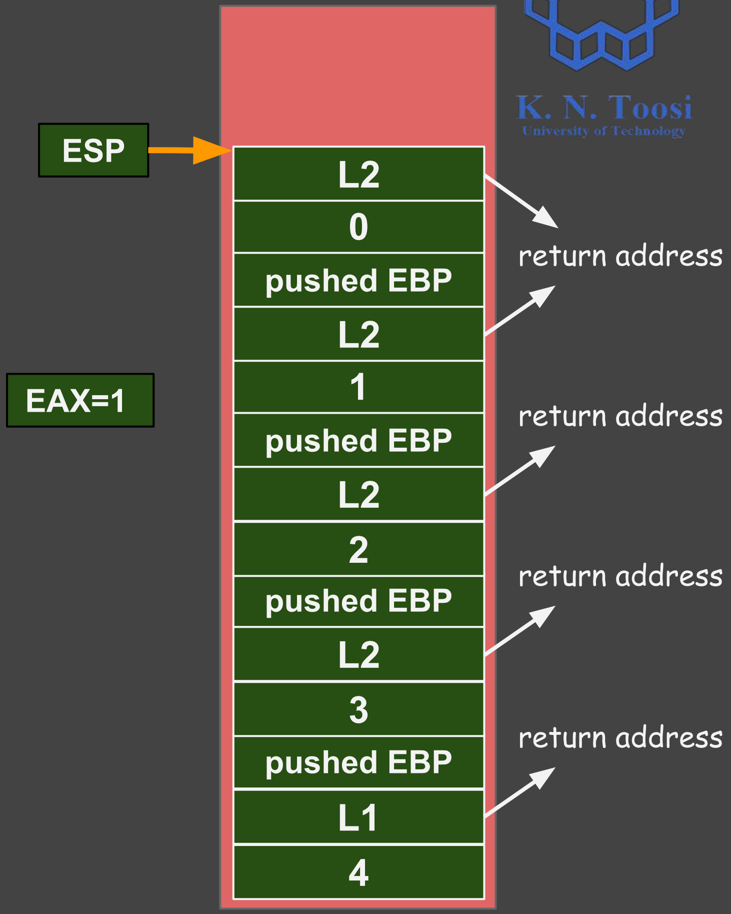
Recursion

factorial.asm

```
              :
     ;; compute fact(4)
     push 4
     call fact
L1:  add esp, 4

     call print_int
     call print_nl
              :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
      jg  recur

      mov eax, 1
      jmp endfact

recur:
      dec eax
      push eax
      call fact
L2:   add esp, 4

      imul dword [ebp+8]

endfact:
      pop ebp
      ret
```



ESP
EAX=1

L2
0
pushed EBP
L2
1
pushed EBP
L2
2
pushed EBP
L2
3
pushed EBP
L1
4

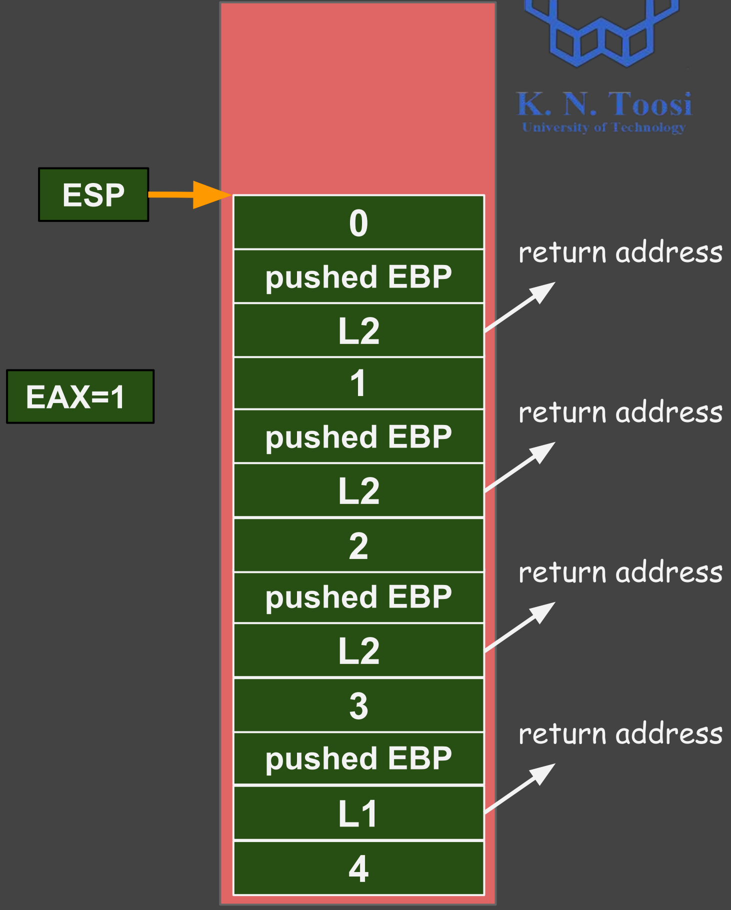
Recursion

factorial.asm

```
        :
;; compute fact(4)
push 4
call fact
L1:  add esp, 4

     call print_int
     call print_nl
        :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
      jg  recur

      mov eax, 1
      jmp endfact

recur:
      dec eax
      push eax
      call fact
L2:   add esp, 4

      imul dword [ebp+8]

endfact:
      pop ebp
      ret
```



0
pushed EBP
L2
1
pushed EBP
L2
2
pushed EBP
L2
3
pushed EBP
L1
4

K. N. Toosi
University of Technology

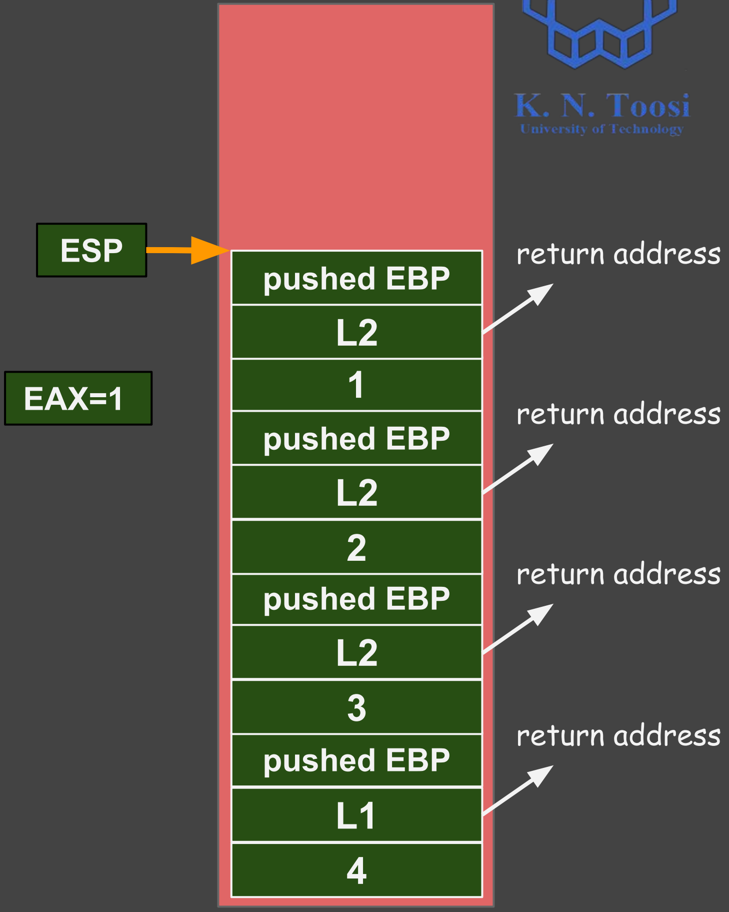
Recursion

factorial.asm

```
            :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
            :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
      jg  recur

      mov eax, 1
      jmp endfact

recur:
      dec eax
      push eax
      call fact
L2:   add esp, 4

      imul dword [ebp+8]

endfact:
      pop ebp
      ret
```



ESP

EAX=1

pushed EBP

L2

1

pushed EBP

L2

2

pushed EBP

L2

3

pushed EBP

L1

4

return address

return address

return address

return address

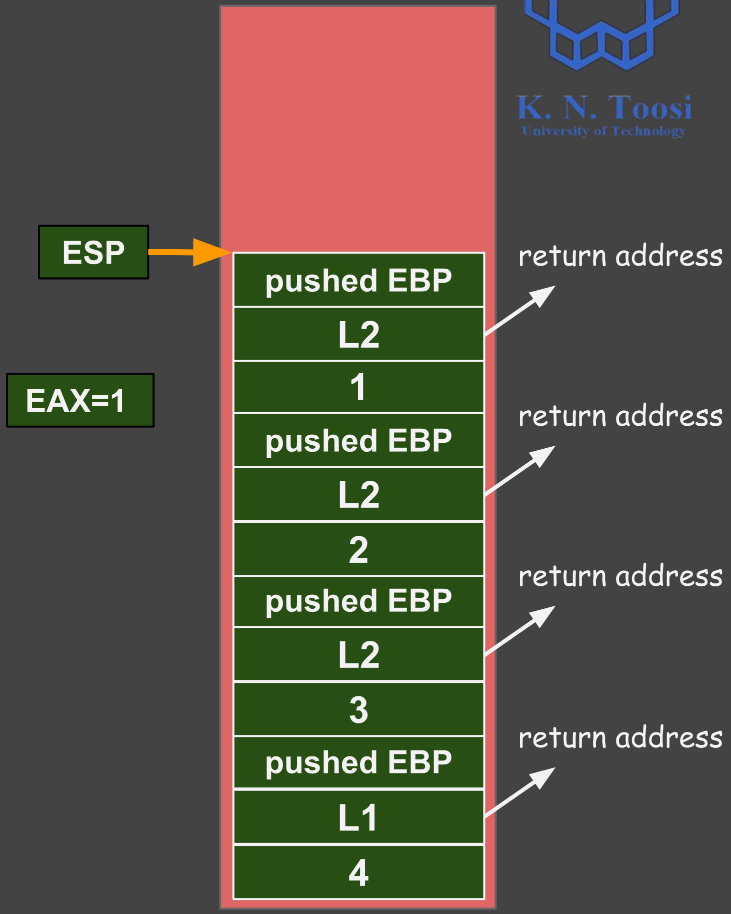
Recursion

factorial.asm

```
            :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
            :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
      jg  recur

      mov eax, 1
      jmp endfact

recur:
      dec eax
      push eax
      call fact
L2:   add esp, 4

      imul dword [ebp+8]   EAX*=1

endfact:
      pop ebp
      ret
```


ESP

EAX=1

pushed EBP
L2
1
pushed EBP
L2
2
pushed EBP
L2
3
pushed EBP
L1
4

K. N. Toosi
University of Technology

Recursion

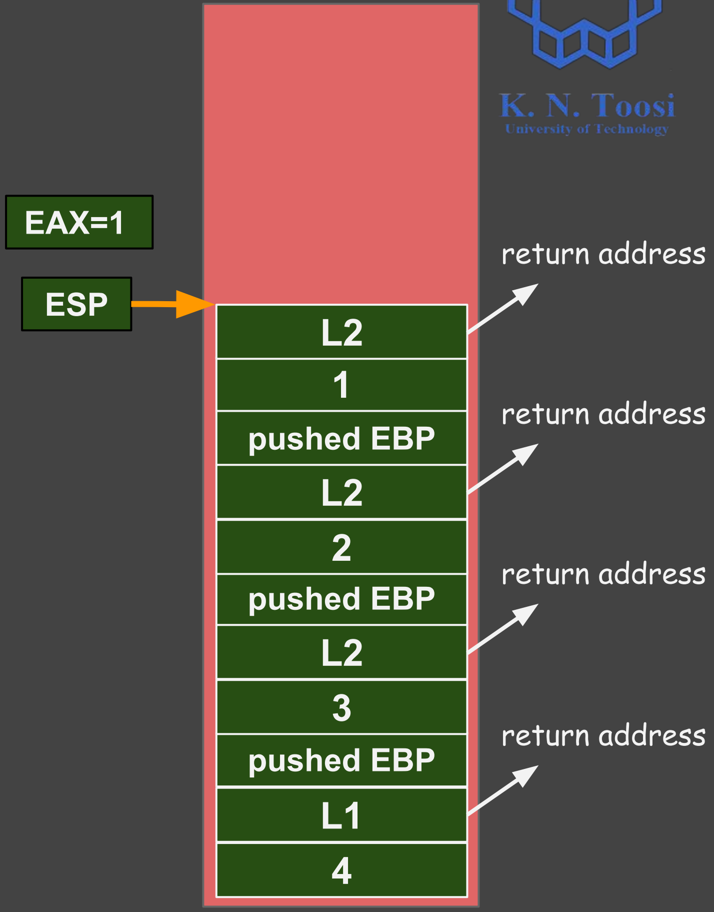
K. N. Toosi
University of Technology

factorial.asm

```
         :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
         :
```

factorial.asm (cont.)

```
fact:
    push ebp
    mov  ebp, esp

    mov eax, [ebp+8]
    cmp eax, 0
    jg  recur

    mov eax, 1
    jmp endfact

recur:
    dec eax
    push eax
    call fact
L2: add esp, 4

    imul dword [ebp+8]

endfact:
    pop ebp
    ret
```



EAX=1

ESP

L2
1
pushed EBP
L2
2
pushed EBP
L2
3
pushed EBP
L1
4

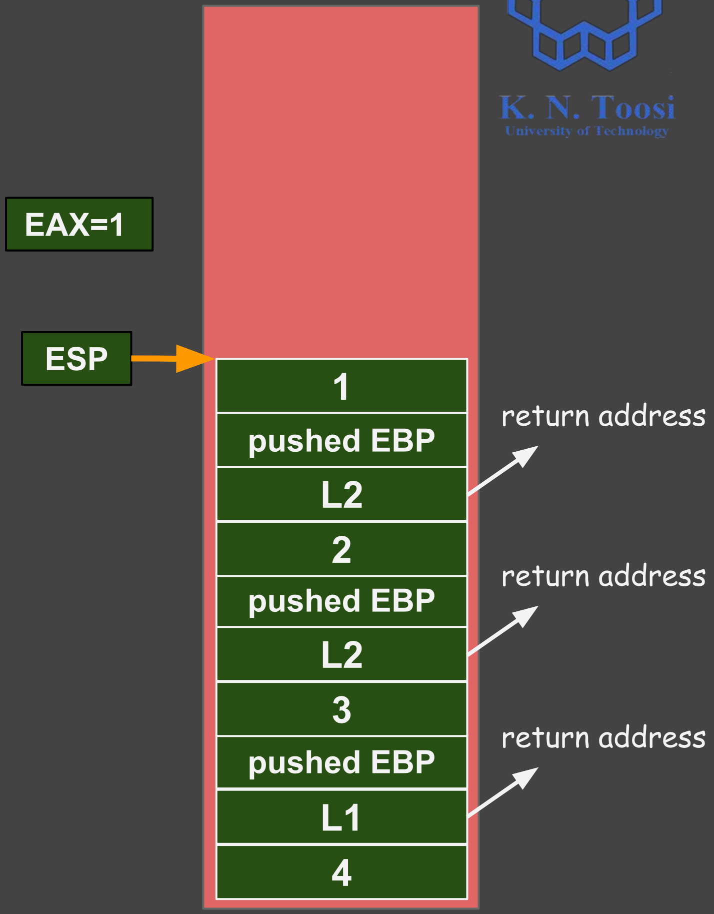
Recursion

factorial.asm

```
              :
      ;; compute fact(4)
      push 4
      call fact
L1:   add esp, 4

      call print_int
      call print_nl
              :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
      jg  recur

      mov eax, 1
      jmp endfact

recur:
      dec eax
      push eax
      call fact
L2:   add esp, 4

      imul dword [ebp+8]

endfact:
      pop ebp
      ret
```



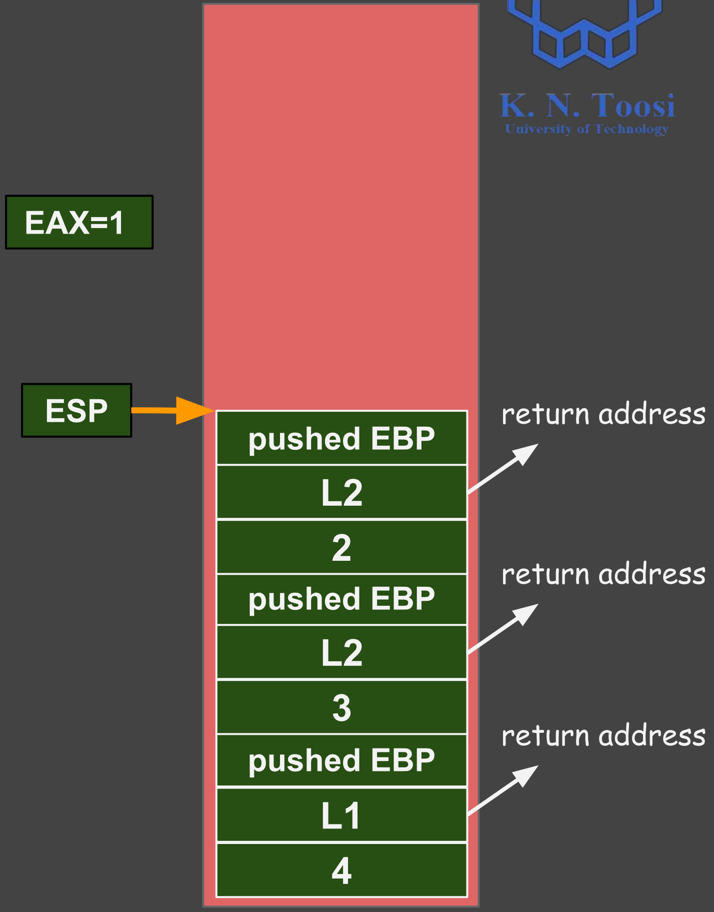
Recursion

factorial.asm

```
            :
      ;; compute fact(4)
      push 4
      call fact
L1:   add esp, 4

      call print_int
      call print_nl
            :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
      jg  recur

      mov eax, 1
      jmp endfact

recur:
      dec eax
      push eax
      call fact
L2:   add esp, 4

→     imul dword [ebp+8]

endfact:
      pop ebp
      ret
```



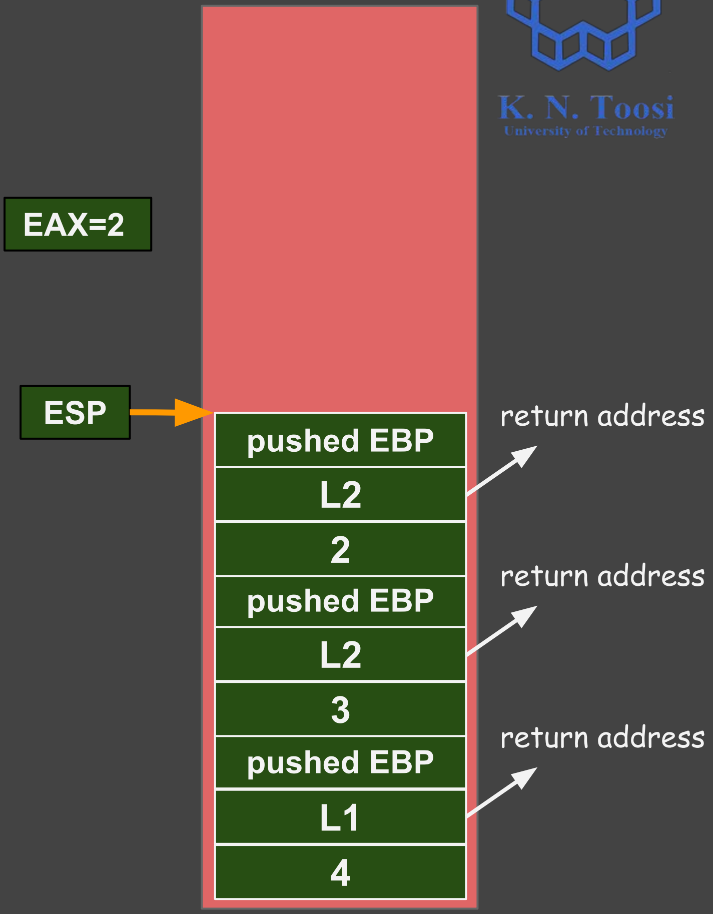
Recursion

factorial.asm

```
            :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
            :
```

factorial.asm (cont.)

```
fact:
    push ebp
    mov  ebp, esp

    mov eax, [ebp+8]
    cmp eax, 0
    jg  recur

    mov eax, 1
    jmp endfact

recur:
    dec eax
    push eax
    call fact
L2: add esp, 4

    imul dword [ebp+8]   EAX*=2

endfact:
    pop ebp
    ret
```



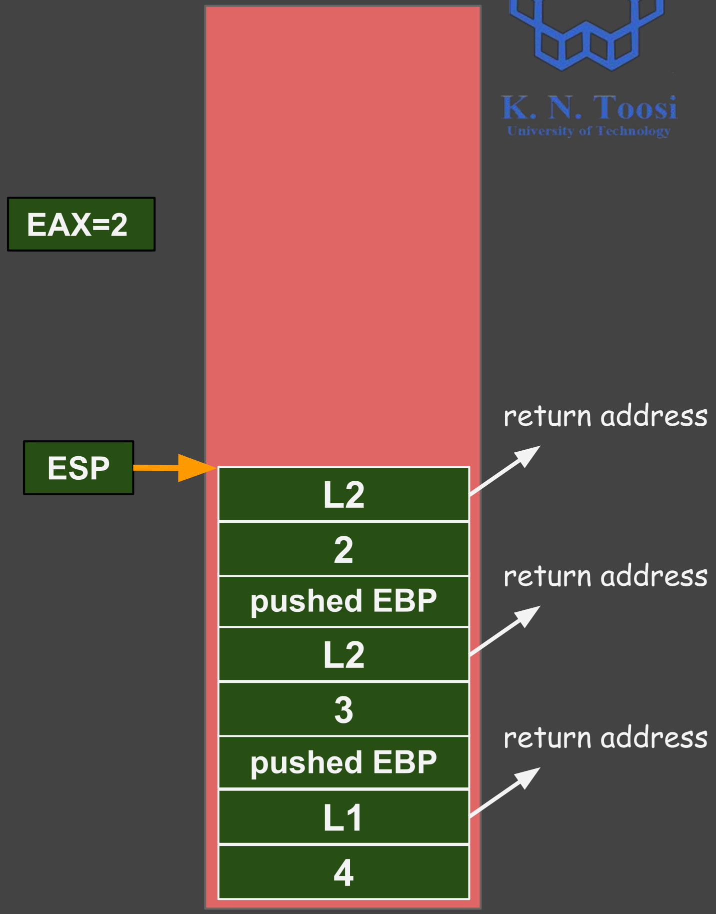
Recursion

factorial.asm

```
             :
      ;; compute fact(4)
      push 4
      call fact
L1:   add esp, 4

      call print_int
      call print_nl
             :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
      jg  recur

      mov eax, 1
      jmp endfact

recur:
      dec eax
      push eax
      call fact
L2:   add esp, 4

      imul dword [ebp+8]

endfact:
      pop ebp
  →   ret
```



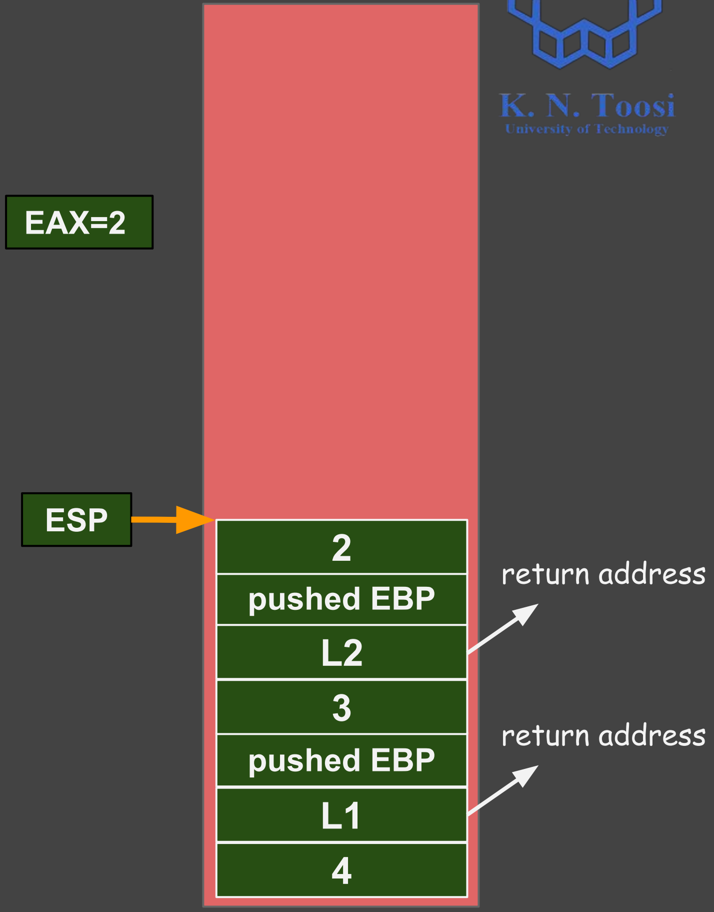
Recursion

factorial.asm

```
        :
;; compute fact(4)
push 4
call fact
L1:  add esp, 4

     call print_int
     call print_nl
        :
```

factorial.asm (cont.)

```
fact:
        push ebp
        mov  ebp, esp

        mov eax, [ebp+8]
        cmp eax, 0
        jg  recur

        mov eax, 1
        jmp endfact

recur:
        dec eax
        push eax
        call fact
L2:     add esp, 4

        imul dword [ebp+8]

endfact:
        pop ebp
        ret
```



Recursion

K. N. Toosi
University of Technology

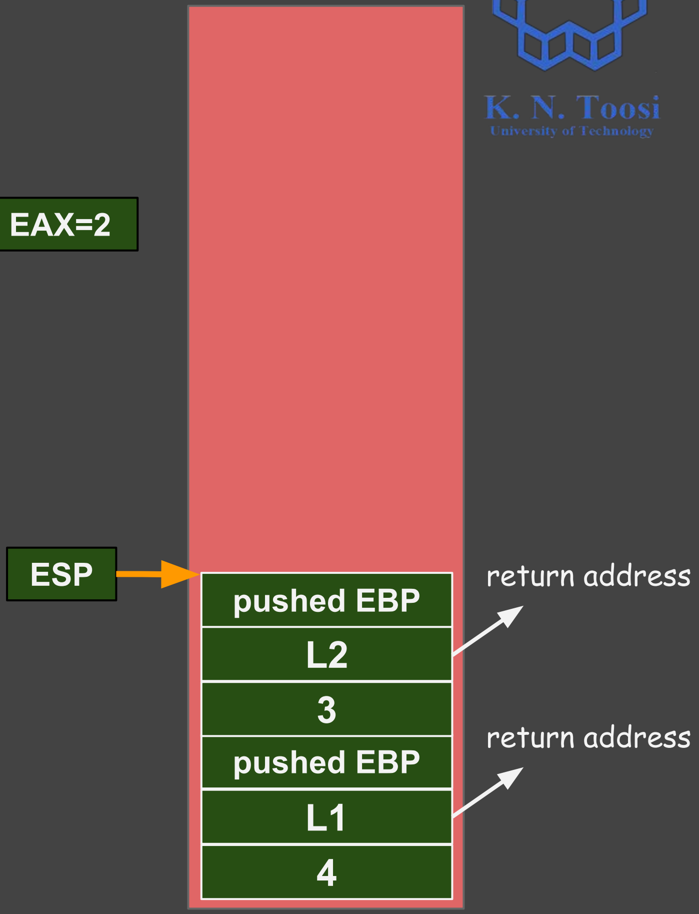
factorial.asm

```
        :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
        :
```

factorial.asm (cont.)

```
fact:
    push ebp
    mov  ebp, esp

    mov eax, [ebp+8]
    cmp eax, 0
    jg  recur

    mov eax, 1
    jmp endfact

recur:
    dec eax
    push eax
    call fact
L2: add esp, 4

    imul dword [ebp+8]

endfact:
    pop ebp
    ret
```

EAX=2



Recursion

factorial.asm

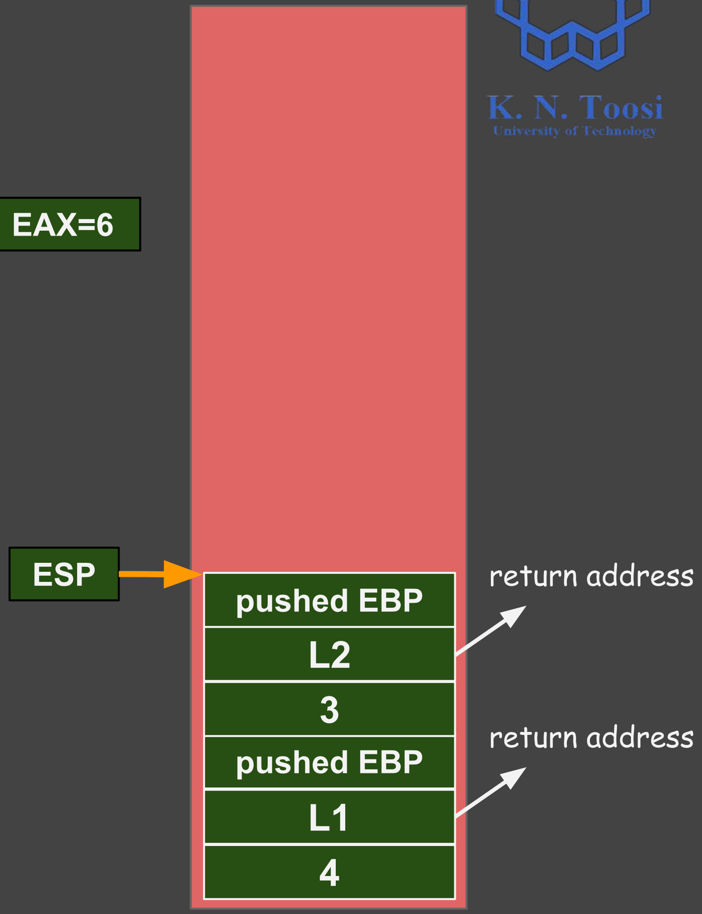
```
            :
        ;; compute fact(4)
        push 4
        call fact
L1:     add esp, 4

        call print_int
        call print_nl
            :
```

factorial.asm (cont.)

```
fact:
        push ebp
        mov  ebp, esp

        mov eax, [ebp+8]
        cmp eax, 0
        jg  recur

        mov eax, 1
        jmp endfact

recur:
        dec eax
        push eax
        call fact
L2:     add esp, 4

        imul dword [ebp+8]

endfact:
        pop ebp
        ret
```

EAX*=3

EAX=6



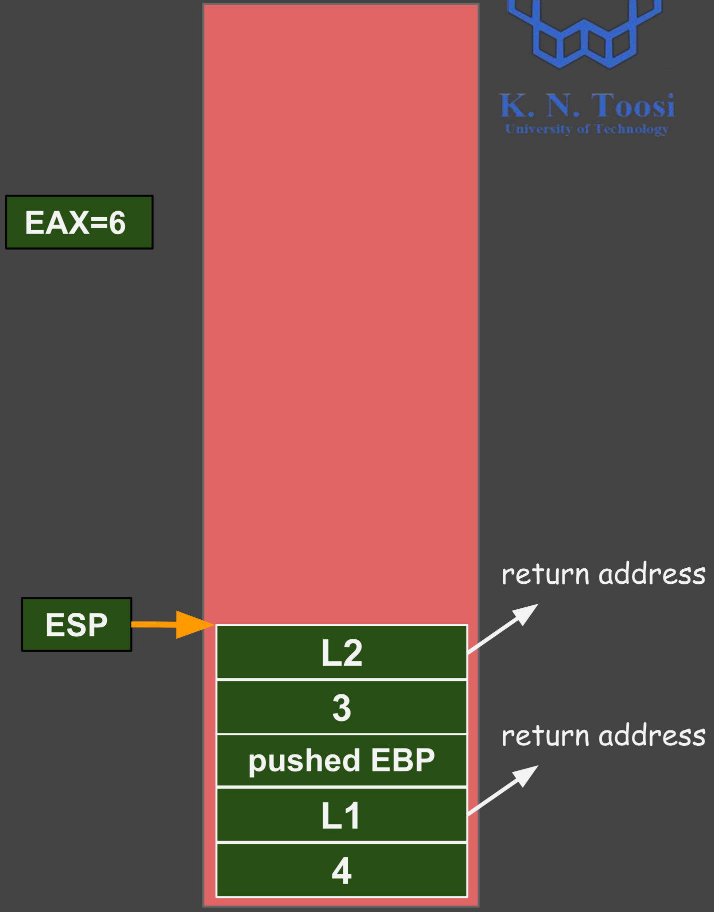
Recursion

factorial.asm

```
            :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
            :
```

factorial.asm (cont.)

```
fact:
        push ebp
        mov  ebp, esp

        mov eax, [ebp+8]
        cmp eax, 0
        jg  recur

        mov eax, 1
        jmp endfact

recur:
        dec eax
        push eax
        call fact
L2:     add esp, 4

        imul dword [ebp+8]

endfact:
        pop ebp
        ret
```



Recursion

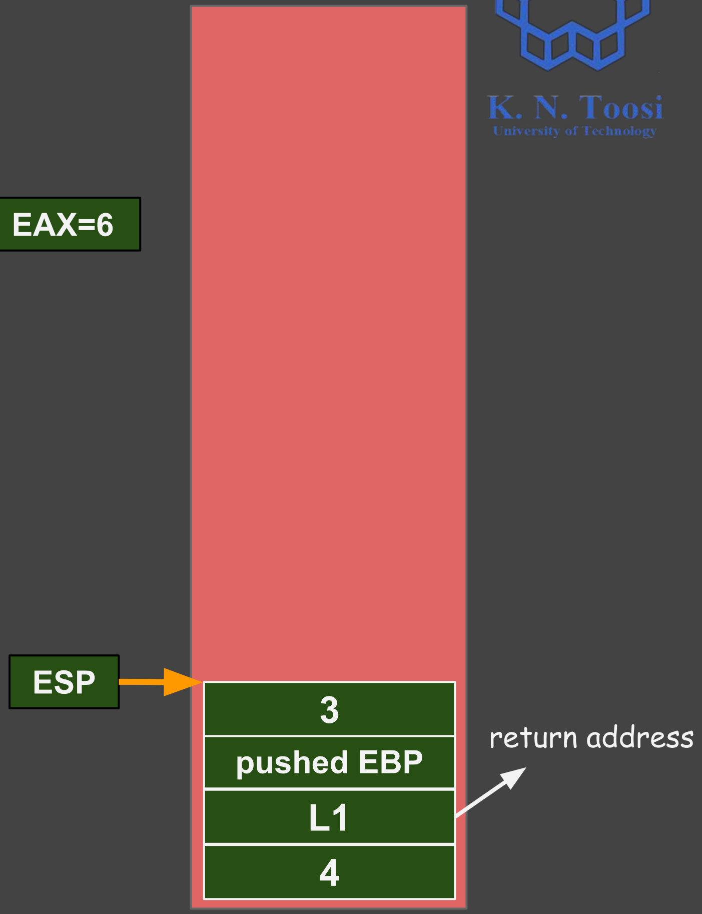
factorial.asm

```
            :
     ;; compute fact(4)
     push 4
     call fact
L1:  add esp, 4

     call print_int
     call print_nl
            :
```

factorial.asm (cont.)

```
fact:
     push ebp
     mov  ebp, esp

     mov eax, [ebp+8]
     cmp eax, 0
     jg  recur

     mov eax, 1
     jmp endfact

recur:
     dec eax
     push eax
     call fact
L2:  add esp, 4

     imul dword [ebp+8]

endfact:
     pop ebp
     ret
```

EAX=6



Recursion

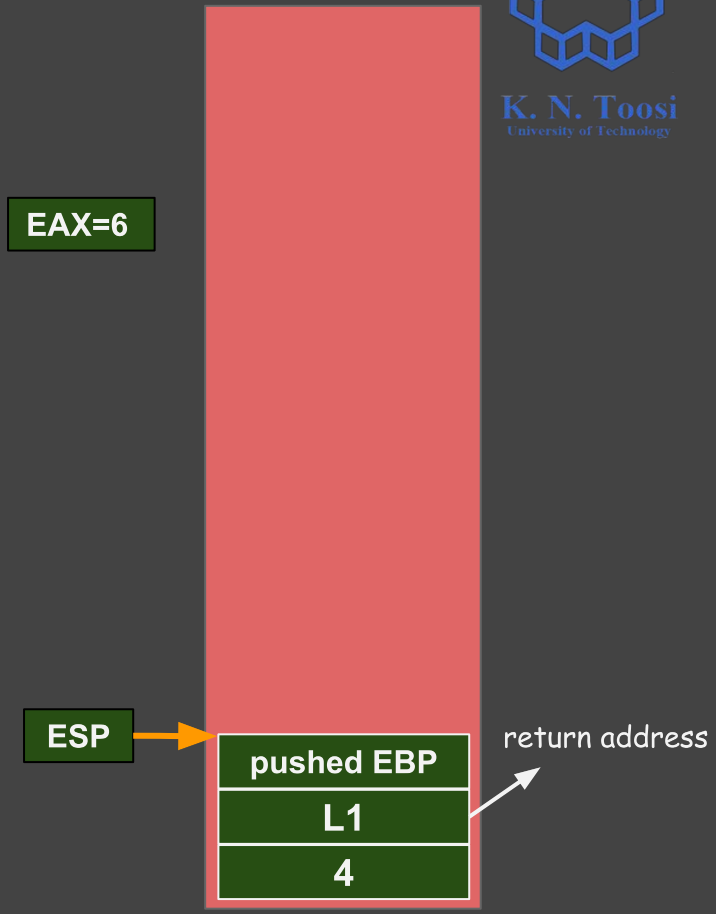
K. N. Toosi
University of Technology

factorial.asm

```
        :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
        :
```

factorial.asm (cont.)

```
fact:
    push ebp
    mov  ebp, esp

    mov eax, [ebp+8]
    cmp eax, 0
    jg  recur

    mov eax, 1
    jmp endfact

recur:
    dec eax
    push eax
    call fact
L2: add esp, 4
→
    imul dword [ebp+8]

endfact:
    pop ebp
    ret
```



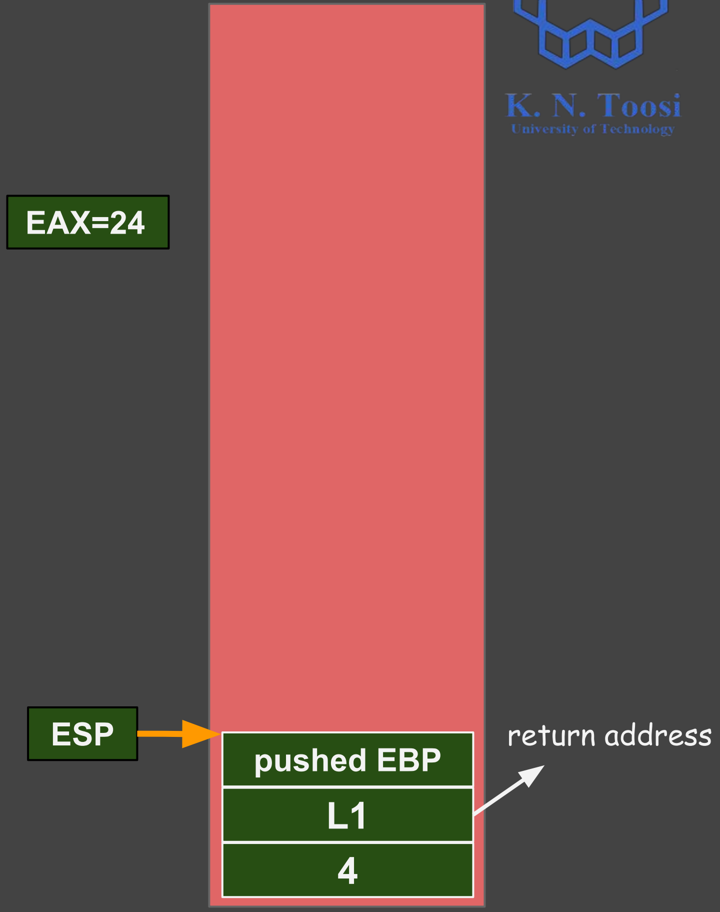
Recursion

factorial.asm

```
            :
     ;; compute fact(4)
     push 4
     call fact
L1:  add esp, 4

     call print_int
     call print_nl
            :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
      jg  recur

      mov eax, 1
      jmp endfact

recur:
      dec eax
      push eax
      call fact
L2:   add esp, 4

      imul dword [ebp+8]   EAX*=4

endfact:
      pop ebp
      ret
```



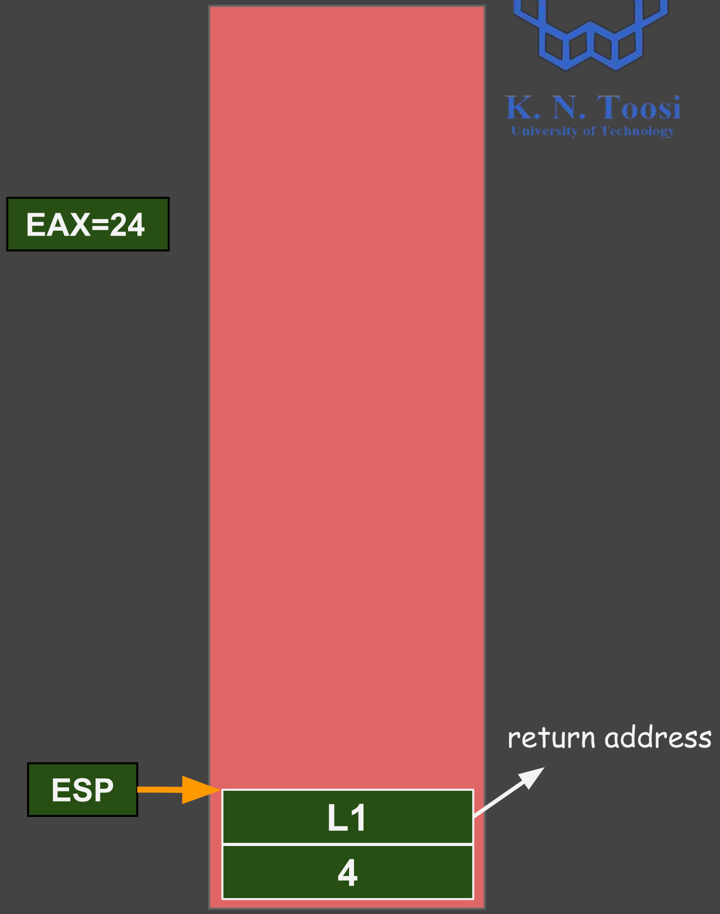
Recursion

factorial.asm

```
          :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
          :
```

factorial.asm (cont.)

```
fact:
        push ebp
        mov  ebp, esp

        mov eax, [ebp+8]
        cmp eax, 0
        jg  recur

        mov eax, 1
        jmp endfact

recur:
        dec eax
        push eax
        call fact
L2:     add esp, 4

        imul dword [ebp+8]

endfact:
        pop ebp
        ret
```



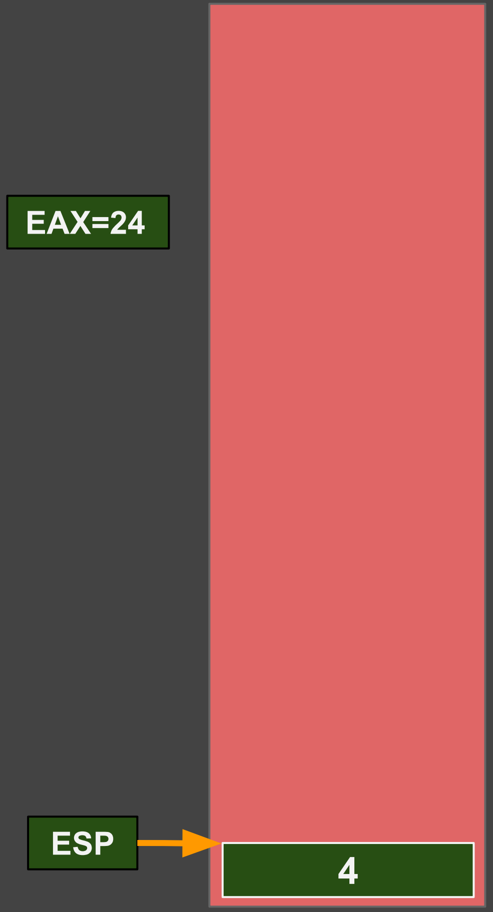
Recursion

factorial.asm

```
        :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4

    call print_int
    call print_nl
        :
```

factorial.asm (cont.)

```
fact:
    push ebp
    mov  ebp, esp

    mov eax, [ebp+8]
    cmp eax, 0
    jg  recur

    mov eax, 1
    jmp endfact

recur:
    dec eax
    push eax
    call fact
L2: add esp, 4

    imul dword [ebp+8]

endfact:
    pop ebp
    ret
```



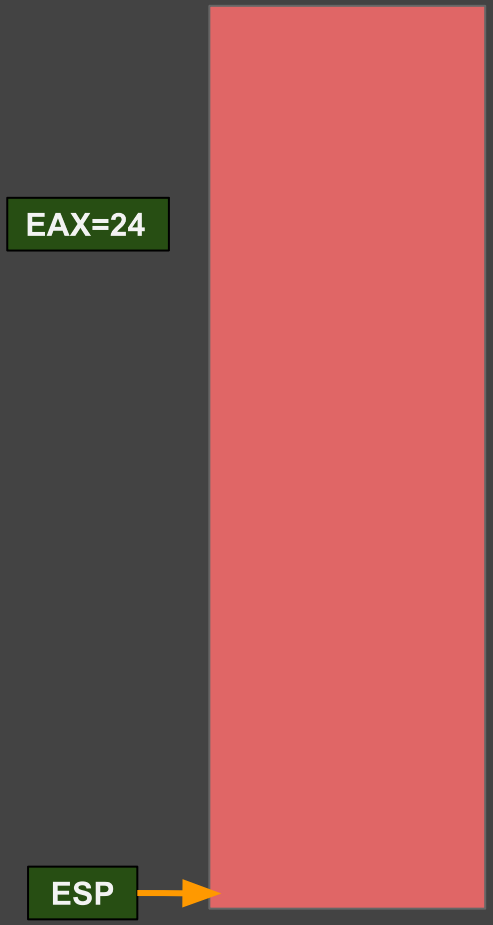
Recursion

factorial.asm

```
          :
    ;; compute fact(4)
    push 4
    call fact
L1: add esp, 4
→
    call print_int
    call print_nl
          :
```

factorial.asm (cont.)

```
fact:
      push ebp
      mov  ebp, esp

      mov eax, [ebp+8]
      cmp eax, 0
      jg    recur

      mov eax, 1
      jmp  endfact

recur:
      dec eax
      push eax
      call  fact
L2:   add esp, 4

      imul dword [ebp+8]

endfact:
      pop ebp
      ret
```



Practice:

print_int_rec.c

```c
int main() {
  print_integer(12340);
  putchar('\n');

  print_integer(-842101);
  putchar('\n');
}

void print_integer(int n) {
  if (n < 0) {
    putchar('-');
    print_integer(-n);
  }
  else if (n < 10) {
    putchar('0'+n);
    return;
  }
  else {
    print_integer(n / 10);
    putchar('0' + n % 10);
  }
}
```

Practice:

print_int_rec.c

```c
int main() {
  print_integer(12340);
  putchar('\n');

  print_integer(-842101);
  putchar('\n');
}

void print_integer(int n) {
  if (n < 0) {
    putchar('-');
    print_integer(-n);
  }
  else if (n < 10) {
    putchar('0'+n);
    return;
  }
  else {
    print_integer(n / 10);
    putchar('0' + n % 10);
  }
}
```

print_int_rec.asm

```nasm
section .data
c:     db    0

section .text

myputchar:
       pusha

       mov  [c], al
       mov  ecx, c  ; address of start of message
       mov  edx, 1  ; length of message
       mov  ebx,1   ; file descriptor (1: stdout)
       mov  eax,4   ; syscall number  (4: sys_write)
       int    0x80

       popa
       ret
```

Practice:

print_int_rec.c

```c
int main() {
  print_integer(12340);
  putchar('\n');

  print_integer(-842101);
  putchar('\n');
}

void print_integer(int n) {
  if (n < 0) {
    putchar('-');
    print_integer(-n);
  }
  else if (n < 10) {
    putchar('0'+n);
    return;
  }
  else {
    print_integer(n / 10);
    putchar('0' + n % 10);
  }
}
```

print_int_rec.asm (cont.)

```asm
global _start

_start:
        push 12340
        call print_integer
        ;; callee clears the stack

        mov al, 10
        call myputchar

        push -842101
        call print_integer

        mov al, 10
        call myputchar

        push 0
        call print_integer

        mov al, 10
        call myputchar

        mov eax, 1
        int 0x80
```

Practice:

print_int_rec.c

```c
int main() {
  print_integer(12340);
  putchar('\n');

  print_integer(-842101);
  putchar('\n');
}

void print_integer(int n) {
  if (n < 0) {
    putchar('-');
    print_integer(-n);
  }
  else if (n < 10) {
    putchar('0'+n);
    return;
  }
  else {
    print_integer(n / 10);
    putchar('0' + n % 10);
  }
}
```

print_int_rec.asm (cont.)

```asm
print_integer:
        push ebp
        mov  ebp, esp
        pusha

        mov eax, [ebp+8]

        cmp eax , 0
        jnl check2

        mov  al, '-'
        call myputchar
        mov eax, [ebp+8]

        neg  eax
        push eax
        call print_integer
        jmp endfunc
check2:
        cmp eax, 10
        jge recur

        add  al, '0'
        call myputchar
        jmp endfunc
```

print_int_rec.asm (cont.)

```asm
recur:
        mov edx, 0
        mov ecx, 10
        div ecx

        push eax
        call print_integer

        mov al, dl
        add al, '0'
        call myputchar

endfunc:
        popa
        mov esp, ebp
        pop ebp
        ret 4
```

Practice:

print_int_rec.c

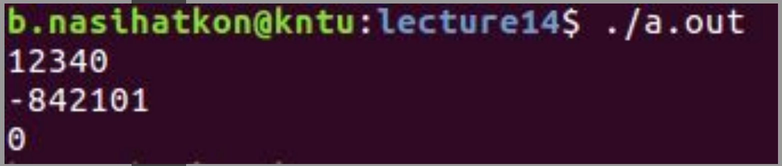
```c
int main() {
  print_integer(12340);
  putchar('\n');

  print_integer(-842101);
  putchar('\n');
}

void print_integer(int n) {
  if (n < 0) {
    putchar('-');
    print_integer(-n);
  }
  else if (n < 10) {
    putchar('0'+n);
    return;
  }
  else {
    print_integer(n / 10);
    putchar('0' + n % 10);
  }
}
```

print_int_rec.asm (cont.)

```
print_integer:
       push ebp
       mov  ebp, esp
       pusha

       mov eax, [ebp+8]

       cmp eax , 0
       jnl check2

       mov  al, '-'
       call myputchar
       mov eax, [ebp+8]

       neg  eax
       push eax
       call print_integer
       jmp endfunc
check2:
       cmp eax, 10
       jge recur

       add  al, '0'
       call myputchar
       jmp endfunc
```

print_int_rec.asm (cont.)

```
recur:
       mov edx, 0
       mov ecx, 10
       div ecx

       push eax
       call print_integer

       mov al, dl
       add al, '0'
       call myputchar

endfunc:
       popa
       mov esp, ebp
       pop ebp
       ret 4
```



```
b.nasihatkon@kntu:lecture14$ ./a.out
12340
-842101
0
```

Indirect call

```
jmp label1

call label1
```

Indirect call

```
jmp eax
call eax
```

Indirect call

```
jmp eax
call eax

jmp [label]
call [label]

jmp [eax]
call [eax]
```

Indirect call

```
jmp eax
call eax

jmp [label]
call [label]

jmp [eax]
call [eax]
```

Applications?

Indirect call

```
jmp eax
call eax

jmp [label]
call [label]

jmp [eax]
call [eax]
```

Applications?

pointer to functions