

# Introduction to 8086 Assembly

## Lecture 14

Recursion

# Recursion



factorial.asm

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

factorial.asm

```
fact:  
  
      mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
  
      mov eax, 1  
      jmp endfact  
  
recur:  
      dec eax  
      push eax  
      call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact:  
      ret
```

# Recursion



factorial.asm

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

factorial.asm

```
fact:  
      mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
  
      mov eax, 1  
      jmp endfact  
  
recur:  
      dec eax  
      push eax  
      call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact:  
      ret
```

ESP

4


# Recursion



factorial.asm

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

factorial.asm

```
fact:   mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
  
      mov eax, 1  
      jmp endfact  
  
recur:  
      dec eax  
      push eax  
      call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact:  
      ret
```

ESP

L1

4

return address

# Recursion



factorial.asm

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

factorial.asm

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       → jg recur  
       mov eax, 1  
       jmp endfact  
  
recur: dec eax  
       push eax  
       call fact  
L2:   add esp, 4  
  
       imul dword [esp+4]  
  
endfact: ret
```

EAX=4

ESP

L1

4

return address


# Recursion



factorial.asm

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

factorial.asm

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
  
       mov eax, 1  
       jmp endfact  
  
recur:  dec eax  
       push eax  
       call fact  
L2:   add esp, 4  
  
       imul dword [esp+4]  
  
endfact: ret
```

EAX=4

ESP

L1

4

return address

# Recursion



factorial.asm

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

factorial.asm

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
  
       mov eax, 1  
       jmp endfact  
  
recur: dec eax  
       → push eax  
       call fact  
L2:   add esp, 4  
  
       imul dword [esp+4]  
  
endfact: ret
```

EAX=3

ESP

L1

4

return address

# Recursion



factorial.asm

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

factorial.asm

```
fact:  
      mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
  
      mov eax, 1  
      jmp endfact  
  
recur:  
      dec eax  
      push eax  
      → call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact:  
      ret
```

EAX=3

ESP

3

L1

4

return address




# Recursion



factorial.asm

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

factorial.asm

```
fact:   mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
  
      mov eax, 1  
      jmp endfact  
  
recur:  
      dec eax  
      push eax  
      call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact:  
      ret
```

EAX=3

ESP 

L2

3

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

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       → jg recur  
  
       mov eax, 1  
       jmp endfact  
  
recur: dec eax  
       push eax  
       call fact  
L2:   add esp, 4  
  
       imul dword [esp+4]  
  
endfact: ret
```

EAX=3

ESP

L2

3

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

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
  
       mov eax, 1  
       jmp endfact  
  
recur: → dec eax  
        push eax  
        call fact  
L2:   add esp, 4  
  
       imul dword [esp+4]  
  
endfact:  ret
```

EAX=3

ESP

L2

3

L1

4

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

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
  
       mov eax, 1  
       jmp endfact  
  
recur: → dec eax  
        push eax  
        call fact  
L2:   add esp, 4  
  
       imul dword [esp+4]  
  
endfact: ret
```

EAX=2

ESP

L2

3

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

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
  
       mov eax, 1  
       jmp endfact  
  
recur: dec eax  
       push eax  
       → call fact  
L2:   add esp, 4  
  
       imul dword [esp+4]  
  
endfact: ret
```

EAX=2

ESP

2

L2

3

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

```
fact:  → mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
  
      mov eax, 1  
      jmp endfact  
  
recur:  
      dec eax  
      push eax  
      call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact:  
      ret
```

EAX=2

ESP →

L2

2

L2

3

L1

4

return address

return address

return address

# Recursion



factorial.asm

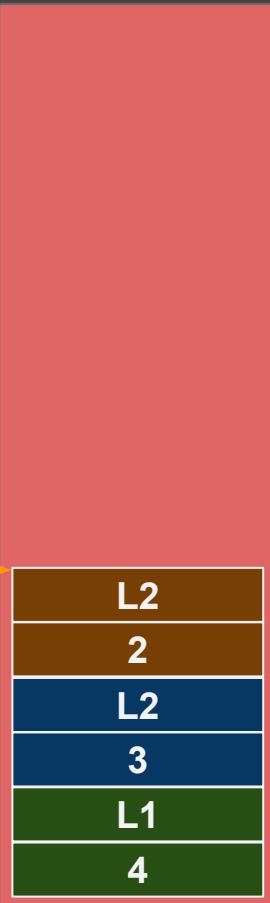
```
      :  
      ;; compute fact(4)  
      push 4  
      call fact  
L1:   add esp, 4  
  
      call print_int  
      call print_nl  
      :
```

factorial.asm

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       → jg recur  
  
       mov eax, 1  
       jmp endfact  
  
recur: dec eax  
       push eax  
       call fact  
L2:   add esp, 4  
  
       imul dword [esp+4]  
  
endfact: ret
```

EAX=2

ESP



L2

2

L2

3

L1

4

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

```
fact:  mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
  
      mov eax, 1  
      jmp endfact  
  
recur:  dec eax  
      push eax  
      call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact:  ret
```

EAX=2

ESP

L2

2

L2

3

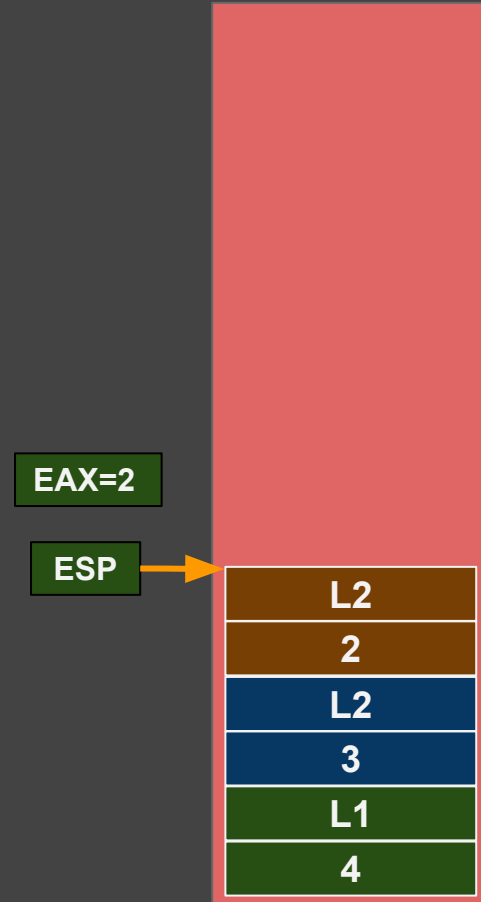
L1

4

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

```
fact:  
      mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
  
      mov eax, 1  
      jmp endfact  
  
recur:  
      dec eax  
      → push eax  
      call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact:  
      ret
```

EAX=1

ESP →



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

```
fact:  mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
  
      mov eax, 1  
      jmp endfact  
  
recur: dec eax  
      push eax  
      → call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact: ret
```

EAX=1

ESP



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

```
fact:  → mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
  
       mov eax, 1  
       jmp endfact  
  
recur:  
       dec eax  
       push eax  
       call fact  
L2:   add esp, 4  
  
       imul dword [esp+4]  
  
endfact:  
      ret
```

EAX=1

ESP

L2

1

L2

2

L2

3

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

```
fact:  mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
      mov eax, 1  
      jmp endfact  
  
recur: dec eax  
      push eax  
      call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact: ret
```

EAX=1

ESP

L2

1

L2

2

L2

3

L1

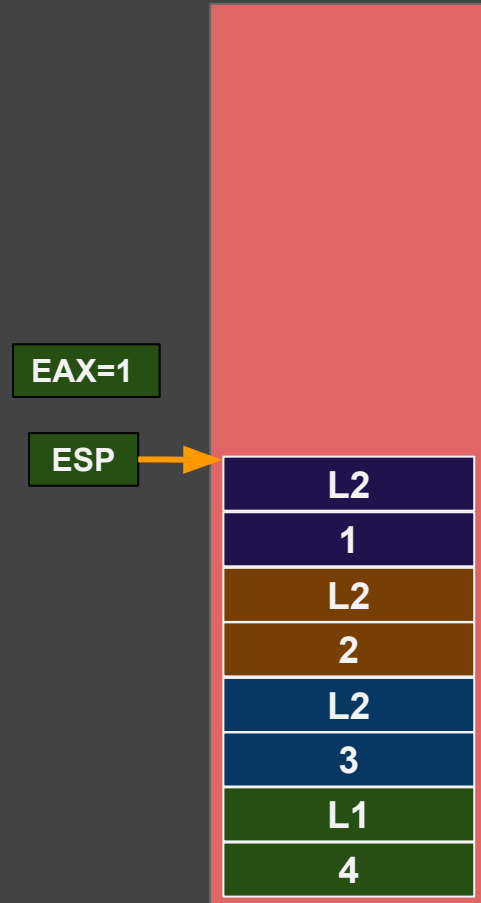
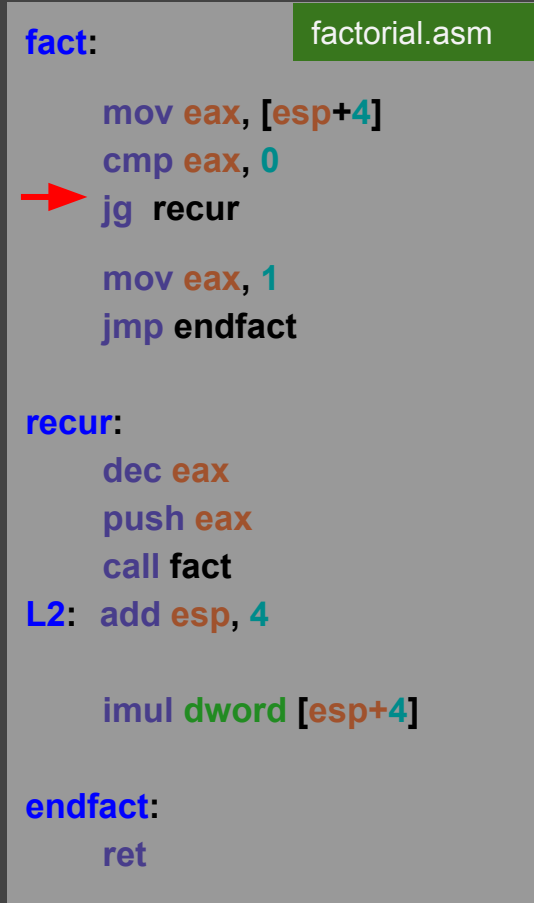
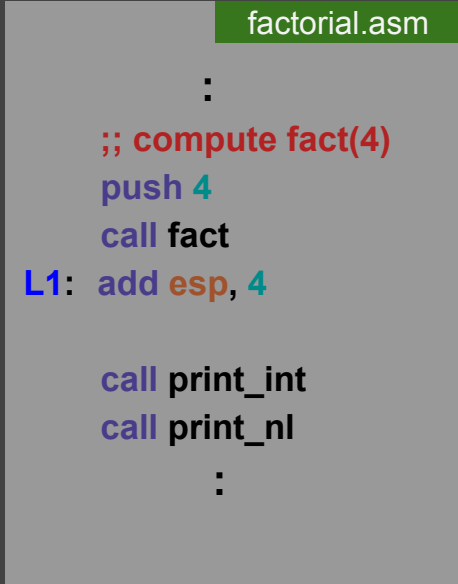
4

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

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
  
       mov eax, 1  
       jmp endfact  
  
recur:   
      dec eax  
      push eax  
      call fact  
L2:   add esp, 4  
  
       imul dword [esp+4]  
  
endfact:   
      ret
```

EAX=1

ESP

L2

1

L2

2

L2

3

L1

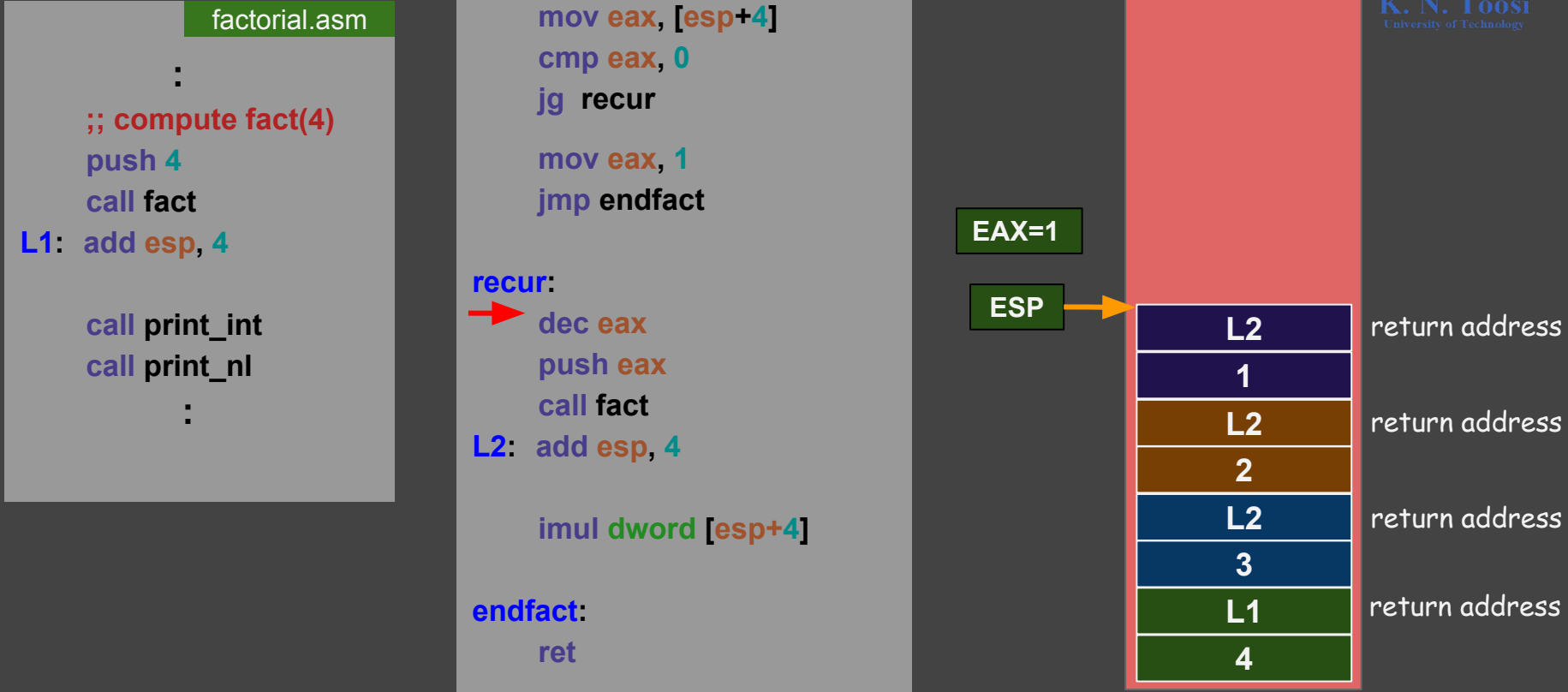
4

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

```
fact:  mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
  
      mov eax, 1  
      jmp endfact  
  
recur: dec eax  
      → push eax  
      call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact: ret
```

EAX=0

ESP

L2

1

L2

2

L2

3

L1

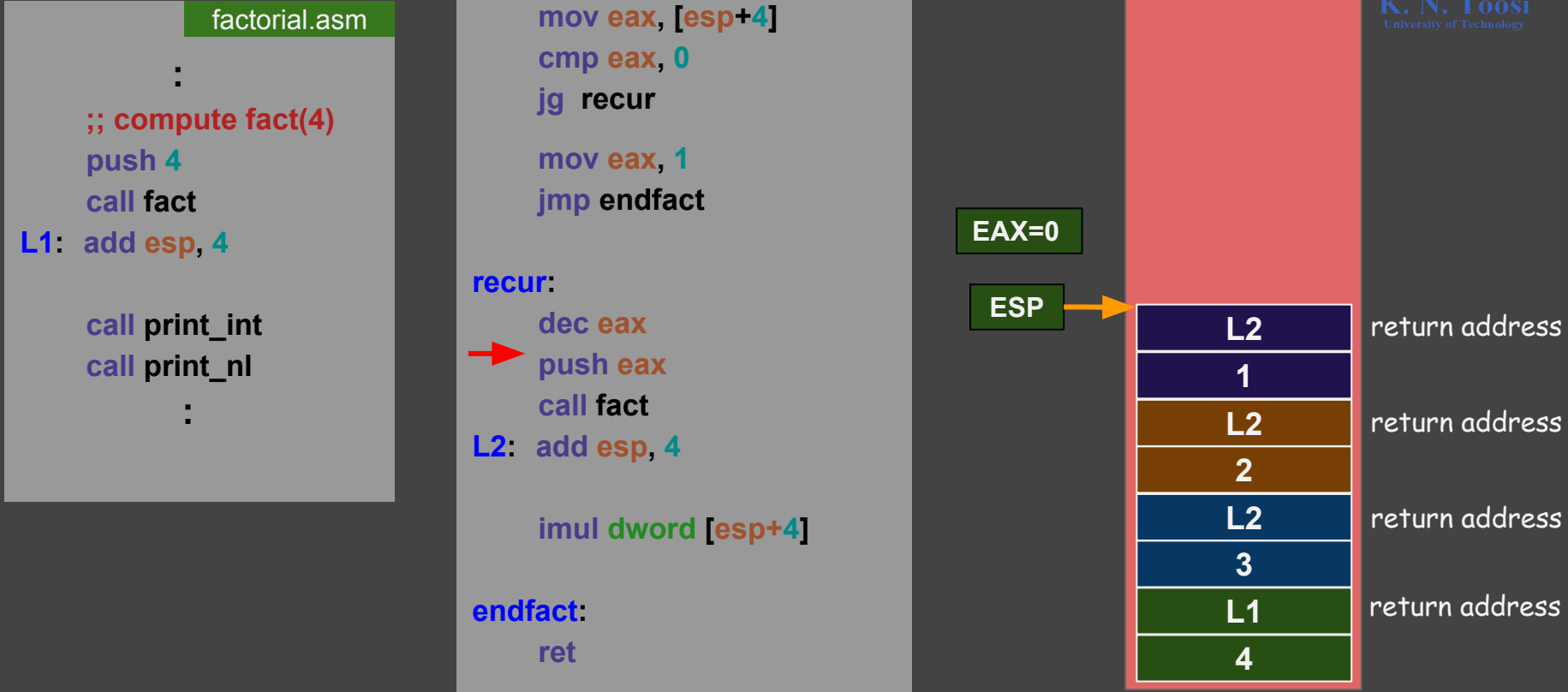
4

return address

return address

return address

return address



# Recursion



factorial.asm

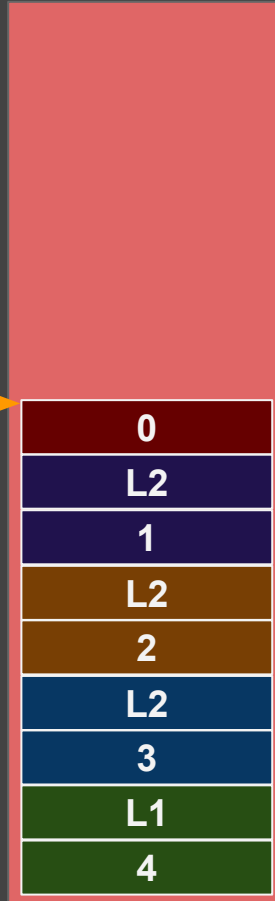
```
      :  
      ;; compute fact(4)  
      push 4  
      call fact  
L1:   add esp, 4  
  
      call print_int  
      call print_nl  
      :
```

factorial.asm

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
  
       mov eax, 1  
       jmp endfact  
  
recur: dec eax  
       push eax  
       → call fact  
L2:   add esp, 4  
  
       imul dword [esp+4]  
  
endfact: ret
```

EAX=0

ESP



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

```
fact:  → mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
  
       mov eax, 1  
       jmp endfact  
  
recur:  
       dec eax  
       push eax  
       call fact  
L2:   add esp, 4  
  
       imul dword [esp+4]  
  
endfact:  
      ret
```

EAX=0

ESP

L2

0

L2

1

L2

2

L2

3

L1

4

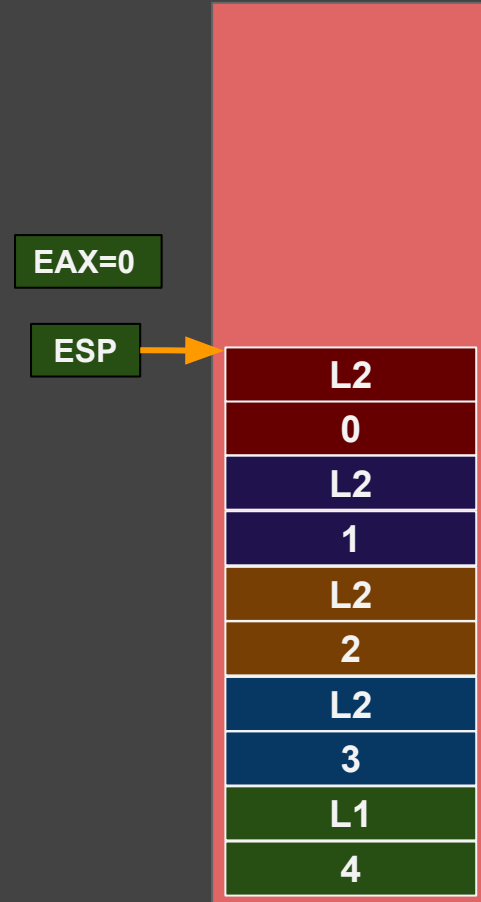
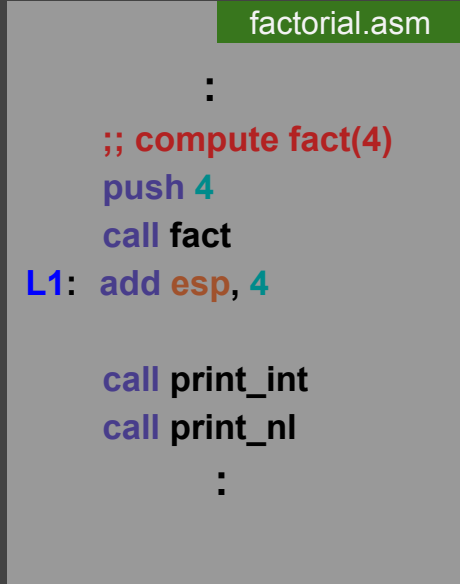
return address

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

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       → jg recur  
       mov eax, 1  
       jmp endfact  
  
recur: dec eax  
       push eax  
       call fact  
L2:   add esp, 4  
  
       imul dword [esp+4]  
  
endfact: ret
```

EAX=0

ESP

L2

0

L2

1

L2

2

L2

3

L1

4

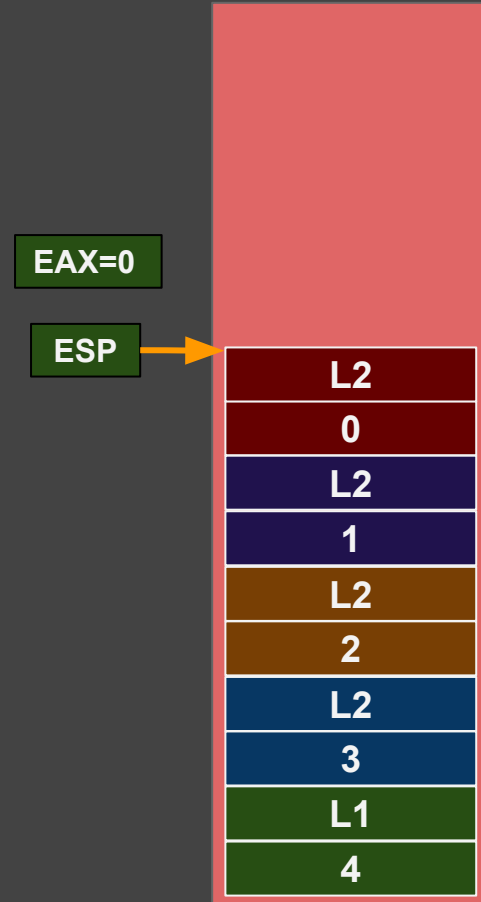
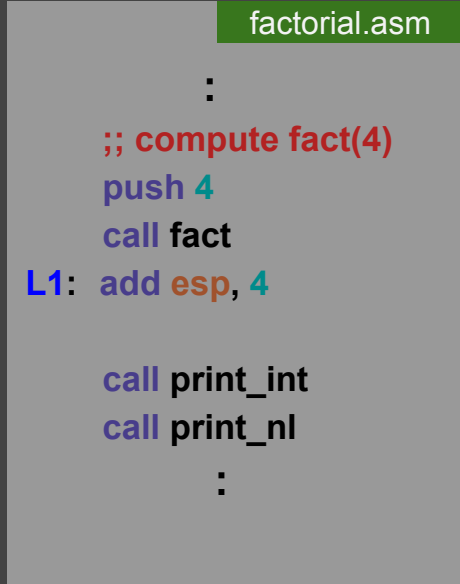
return address

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

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
       → mov eax, 1  
       jmp endfact  
  
recur: dec eax  
       push eax  
       call fact  
L2:   add esp, 4  
  
       imul dword [esp+4]  
  
endfact: ret
```

EAX=0

ESP



return address

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

```
fact:  
      mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
      → mov eax, 1  
      jmp endfact  
  
recur:  
      dec eax  
      push eax  
      call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact:  
      ret
```

EAX=1

ESP

L2

0

L2

1

L2

2

L2

3

L1

4

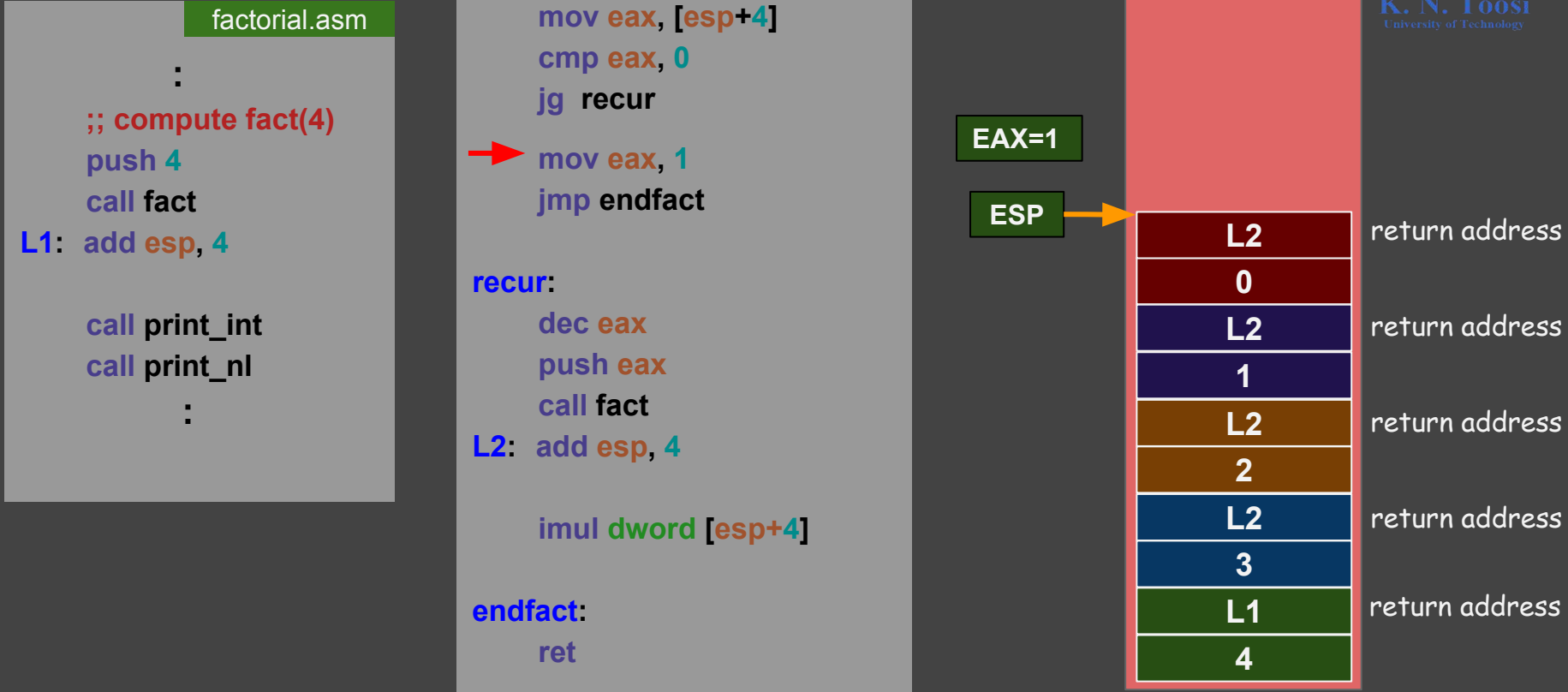
return address

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

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
       mov eax, 1  
       jmp endfact  
  
recur: dec eax  
       push eax  
       call fact  
L2:   add esp, 4  
  
       imul dword [esp+4]  
  
endfact: ret
```

EAX=1

ESP

L2

0

L2

1

L2

2

L2

3

L1

4

return address

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

```
fact:  mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
  
      mov eax, 1  
      jmp endfact  
  
recur:  
      dec eax  
      push eax  
      call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact:  
→ ret
```

EAX=1

ESP

L2

0

L2

1

L2

2

L2

3

L1

4

return address

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
fact:
mov eax, [esp+4]
cmp eax, 0
jg recur

mov eax, 1
jmp endfact

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

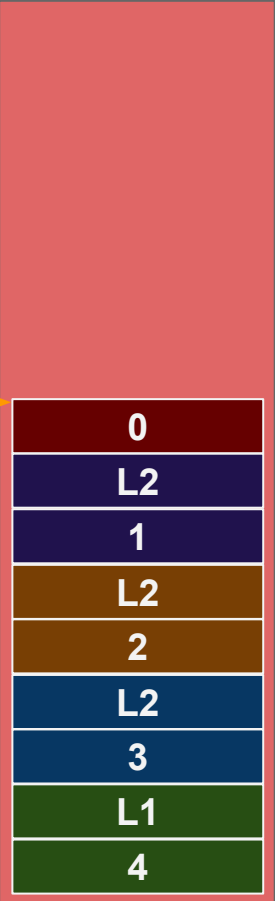
imul dword [esp+4]

endfact:
ret
```



EAX=1

ESP



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

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
  
       mov eax, 1  
       jmp endfact  
  
recur:  
       dec eax  
       push eax  
       call fact  
L2:   add esp, 4  
  
       → imul dword [esp+4]  
  
endfact:  
      ret
```

EAX=1

ESP

L2

1

L2

2

L2

3

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

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
  
       mov eax, 1  
       jmp endfact  
  
recur: dec eax  
       push eax  
       call fact  
L2:   add esp, 4  
  
→ imul dword [esp+4]  EAX*=1  
  
endfact: ret
```

EAX=1

ESP

L2

1

L2

2

L2

3

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

```
fact:  mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
  
      mov eax, 1  
      jmp endfact  
  
recur: dec eax  
      push eax  
      call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact:  → ret
```

EAX=1

ESP →

L2

return address

1

return address

L2

2

return address

L2

3

return address

L1

4

return address

# Recursion



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

call print_int
call print_nl
:
```

```
factorial.asm
fact:
mov eax, [esp+4]
cmp eax, 0
jg recur

mov eax, 1
jmp endfact

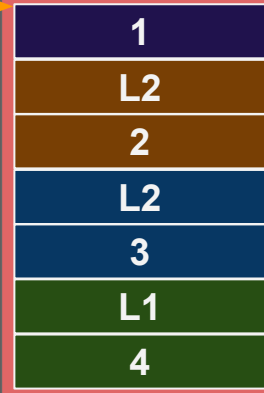
recur:
dec eax
push eax
call fact
L2: add esp, 4

imul dword [esp+4]

endfact:
ret
```

EAX=1

ESP



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

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
  
       mov eax, 1  
       jmp endfact  
  
recur: dec eax  
       push eax  
       call fact  
L2:   add esp, 4  
  
      → imul dword [esp+4]  
  
endfact: ret
```

EAX=1

ESP

L2

2

L2

3

L1

4

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

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
  
       mov eax, 1  
       jmp endfact
```

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

→ `imul dword [esp+4]`

EAX\*=2

```
endfact: ret
```

EAX=2

ESP

L2

2

L2

3

L1

4

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

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
  
       mov eax, 1  
       jmp endfact  
  
recur: dec eax  
       push eax  
       call fact  
L2:   add esp, 4  
  
       imul dword [esp+4]  
  
endfact:  
→ ret
```

EAX=2

ESP →

L2

2

L2

3

L1

4

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
fact:
mov eax, [esp+4]
cmp eax, 0
jg recur

mov eax, 1
jmp endfact

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

imul dword [esp+4]

endfact:
ret
```

EAX=2

ESP



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

```
fact:  mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
  
      mov eax, 1  
      jmp endfact  
  
recur:  
      dec eax  
      push eax  
      call fact  
L2:   add esp, 4  
  
      → imul dword [esp+4]  
  
endfact:  
      ret
```

EAX=2

ESP



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
fact:
mov eax, [esp+4]
cmp eax, 0
jg recur

mov eax, 1
jmp endfact

recur:
dec eax
push eax
call fact
L2: add esp, 4
→ imul dword [esp+4]
EAX*=3

endfact:
ret
```

EAX=6

ESP



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

```
fact:  
      mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
  
      mov eax, 1  
      jmp endfact  
  
recur:  
      dec eax  
      push eax  
      call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact:  
      → ret
```

EAX=6

ESP

L2

3

L1

4

return address

return address

# Recursion

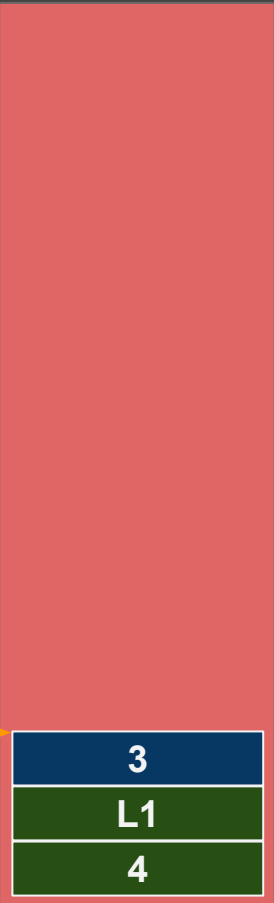


```
factorial.asm  
:  
;; compute fact(4)  
push 4  
call fact  
L1: add esp, 4  
  
call print_int  
call print_nl  
:
```

```
factorial.asm  
fact:  
mov eax, [esp+4]  
cmp eax, 0  
jg recur  
mov eax, 1  
jmp endfact  
  
recur:  
dec eax  
push eax  
call fact  
→ L2: add esp, 4  
  
imul dword [esp+4]  
  
endfact:  
ret
```

EAX=6

ESP →



return address

# Recursion



factorial.asm

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

factorial.asm

```
fact:  mov eax, [esp+4]  
       cmp eax, 0  
       jg recur  
  
       mov eax, 1  
       jmp endfact  
  
recur:  
       dec eax  
       push eax  
       call fact  
L2:   add esp, 4  
  
      → imul dword [esp+4]  
  
endfact:  
      ret
```

EAX=6

ESP

L1

4

return address

# Recursion



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

call print_int
call print_nl
:
```

```
factorial.asm
fact:
mov eax, [esp+4]
cmp eax, 0
jg recur
mov eax, 1
jmp endfact

recur:
dec eax
push eax
call fact
L2: add esp, 4
→ imul dword [esp+4]
endfact:
ret
```

EAX=24

EAX\*=4

ESP



return address

# Recursion



factorial.asm

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

factorial.asm

```
fact:  mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
  
      mov eax, 1  
      jmp endfact  
  
recur:  
      dec eax  
      push eax  
      call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact:  
      → ret
```

EAX=24

ESP

L1

4

return address

# Recursion



factorial.asm

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

factorial.asm

```
fact:  
      mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
  
      mov eax, 1  
      jmp endfact  
  
recur:  
      dec eax  
      push eax  
      call fact  
      L2: add esp, 4  
  
      imul dword [esp+4]  
  
endfact:  
      ret
```

EAX=24

ESP

4



# Recursion



factorial.asm

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

factorial.asm

```
fact:  
      mov eax, [esp+4]  
      cmp eax, 0  
      jg recur  
      mov eax, 1  
      jmp endfact  
  
recur:  
      dec eax  
      push eax  
      call fact  
L2:   add esp, 4  
  
      imul dword [esp+4]  
  
endfact:  
      ret
```

EAX=24

ESP



# Recursion

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

```
factorial.asm  
  
fact:  
mov eax, [esp+4]  
cmp eax, 0  
jg recur  
  
mov eax, 1  
jmp endfact  
  
recur:  
dec eax  
push eax  
call fact  
L2: add esp, 4  
  
imul dword [esp+4]  
  
endfact:  
ret
```

EAX=24

ESP →





# Practice:

```
int main() { print_int_rec.c
    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:

```
int main() { print_int_rec.c
    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);
    }
}
```

**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
```

**print\_int\_rec.asm**



# Practice:

```
int main() { print_int_rec.c
    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);
    }
}
```

global \_start

print\_int\_rec.asm (cont.)

\_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:



```
int main() { print_int_rec.c
    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_integer: print_int_rec.asm (cont.)
    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
```

```
recur: print_int_rec.asm (cont.)
    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:



```
int main() { print_int_rec.c
    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_integer: print_int_rec.asm (cont.)
    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
```

```
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`



**K. N. Toosi**  
University of Technology

# Indirect call

```
jmp eax  
call eax
```



**K. N. Toosi**  
University of Technology

# Indirect call

```
jmp eax  
call eax
```

```
jmp [label]  
call [label]
```

```
jmp [eax]  
call [eax]
```





# Indirect call



K. N. Toosi  
University of Technology

```
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

