Introduction to 8086 Assembly

Lecture 9

Introduction to Subprograms

Indirect addressing

indirect.asm

```
segment .data
l1:     dd 111
segment .text
        ⋮
        mov eax, l1
        call print_int
        call print_nl

        mov eax, [l1]
        call print_int
        call print_nl
```

Indirect addressing

indirect.asm

```
segment .data
l1:     dd 111
segment .text
        ⋮
        mov eax, l1
        call print_int
        call print_nl

        mov eax, [l1]
        call print_int
        call print_nl

        mov ecx, l1
        mov eax, [ecx]
        call print_int
        call print_nl
```

Indirect addressing

indirect.asm

```
segment .data
l1:     dd 111
segment .text
        ⋮
        mov eax, l1
        call print_int
        call print_nl

        mov eax, [l1]
        call print_int
        call print_nl

        mov ecx, l1
        mov eax, [ecx]
        call print_int
        call print_nl
```

Indirect addressing

indirect2.asm

```
segment .data
l1:    dd 111
       dd 222
       dd 444

segment .text
       ⋮
       mov ecx, l1

       mov eax, [ecx]
       call print_int
       call print_nl
```

indirect2.asm (cont.)

```
       mov eax, [ecx+1]
       call print_int
       call print_nl

       mov eax, [ecx+4]
       call print_int
       call print_nl

       mov eax, [ecx+8]
       call print_int
       call print_nl
```

Indirect addressing

indirect2.asm

```
segment .data
l1:    dd 111
       dd 222
       dd 444

segment .text
       ⋮
       mov ecx, l1

       mov eax, [ecx]
       call print_int
       call print_nl
```

indirect2.asm (cont.)

```
       mov eax, [ecx+1]
       call print_int
       call print_nl

       mov eax, [ecx+4]
       call print_int
       call print_nl

       mov eax, [ecx+8]
       call print_int
       call print_nl
```

How does the assembler do this?

Indirect addressing


```
mov eax, [ecx]
mov ax,  [ecx]
mov al,  [ecx]
```

How to implement subprograms?

- Subprogram
- function
- subroutine
- procedure
- routine
- method
- callable

```
void print_salam(void);

int main() {

  print_salam();

}

void print_salam() {
  printf("Salaaaaam!\n");
}
```

How to implement subprograms?


```
void print_salam(void);

int main() {

  print_salam();

}

void print_salam() {
  printf("Salaaaaam!\n");
}
```

```
segment .data
msg:    db "Salaaaaam!", 10, 0

segment .text
        ⋮



        ⋮

print_salam:
        mov eax, msg
        call print_string
```

How to implement subprograms?


```c
void print_salam(void);

int main() {

  print_salam();

}

void print_salam() {
  printf("Salaaaaam!\n");
}
```

```
segment .data
msg:    db "Salaaaaam!", 10, 0

segment .text
        ⋮
        jmp print_salam

        ⋮

print_salam:
        mov eax, msg
        call print_string
```

How to implement subprograms?


```c
void print_salam(void);

int main() {

  print_salam();

}

void print_salam() {
  printf("Salaaaaam!\n");
}
```

```
segment .data
msg:    db "Salaaaaam!", 10, 0

segment .text
        ⋮
        jmp print_salam
l1:
        ⋮

print_salam:
        mov eax, msg
        call print_string
```

How to implement subprograms?


```
void print_salam(void);

int main() {

  print_salam();

}

void print_salam() {
  printf("Salaaaaam!\n");
}
```

```
segment .data
msg:    db "Salaaaaam!", 10, 0

segment .text
        ⋮
        jmp print_salam
l1:                          return address
        ⋮

print_salam:
        mov eax, msg
        call print_string
```

How to implement subprograms?


```
void print_salam(void);

int main() {

  print_salam();

}

void print_salam() {
  printf("Salaaaaam!\n");
}
```

simplefunc1.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0

segment .text
        ⋮
        jmp print_salam
l1:
        ⋮

print_salam:
        mov eax, msg
        call print_string

        jmp l1
```

return address

How to implement subprograms?


```c
void print_salam(void);

int main() {

  print_salam();

}

void print_salam() {
  printf("Salaaaaam!\n");
}
```

What's wrong?

simplefunc1.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0

segment .text
        ⋮
        jmp print_salam
l1:                          return address
        ⋮

print_salam:
        mov eax, msg
        call print_string

        jmp l1
```

How to implement subprograms?


```
void print_salam(void);

int main() {

  print_salam();

}

void print_salam() {
  printf("Salaaaaam!\n");
}
```

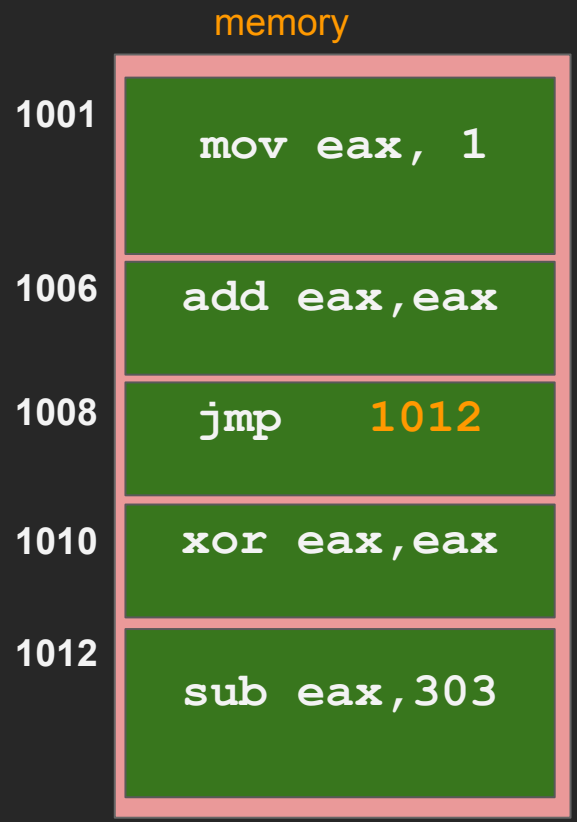
simplefunc2.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0

segment .text
        ⋮
        jmp print_salam
l1:
        ⋮
        jmp print_salam
l2:
        ⋮
print_salam:
        mov eax, msg
        call print_string
        jmp ?
```

return address

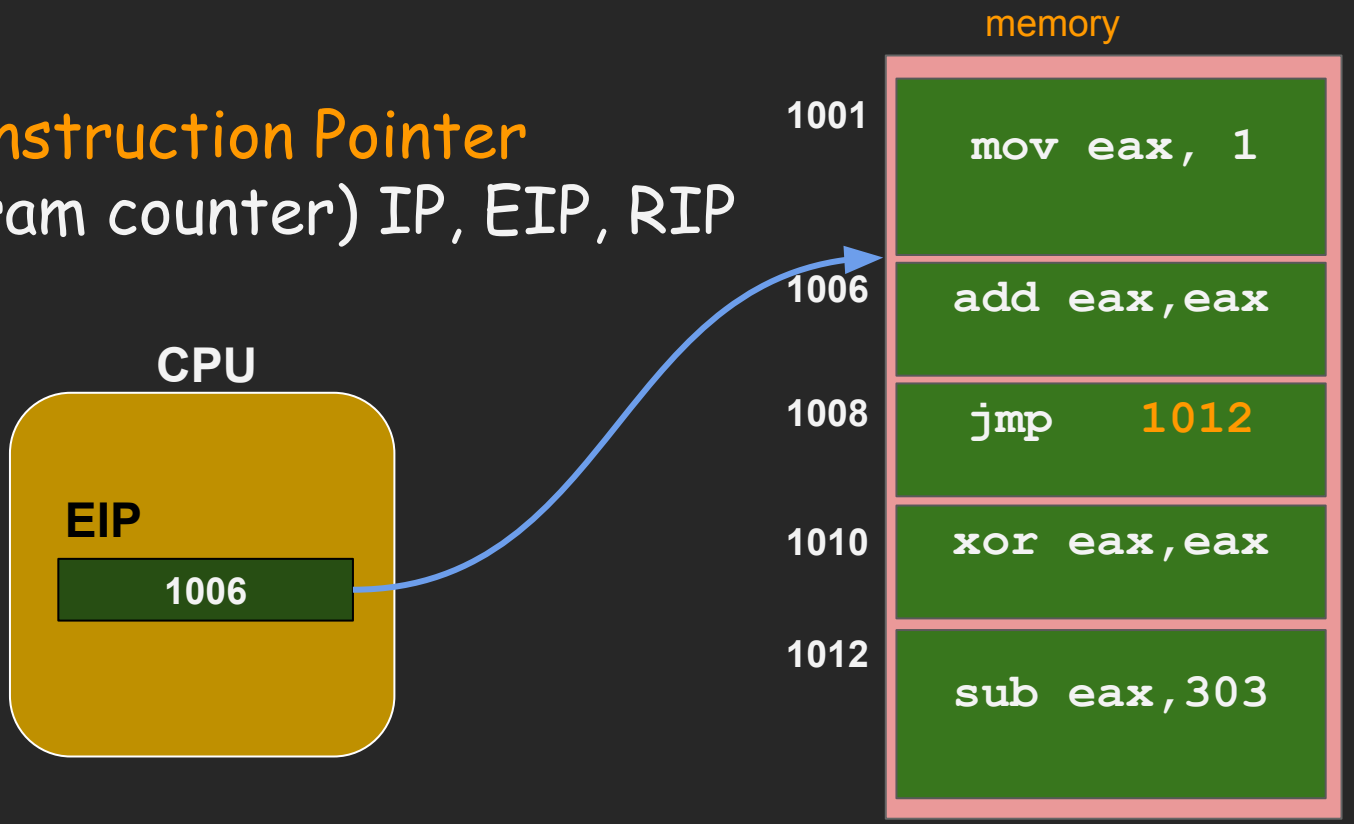
Looking closer at the jmp command


```
mov eax, 1

add eax, eax

jmp label1

xor eax, eax

label1:

sub eax, 303
```

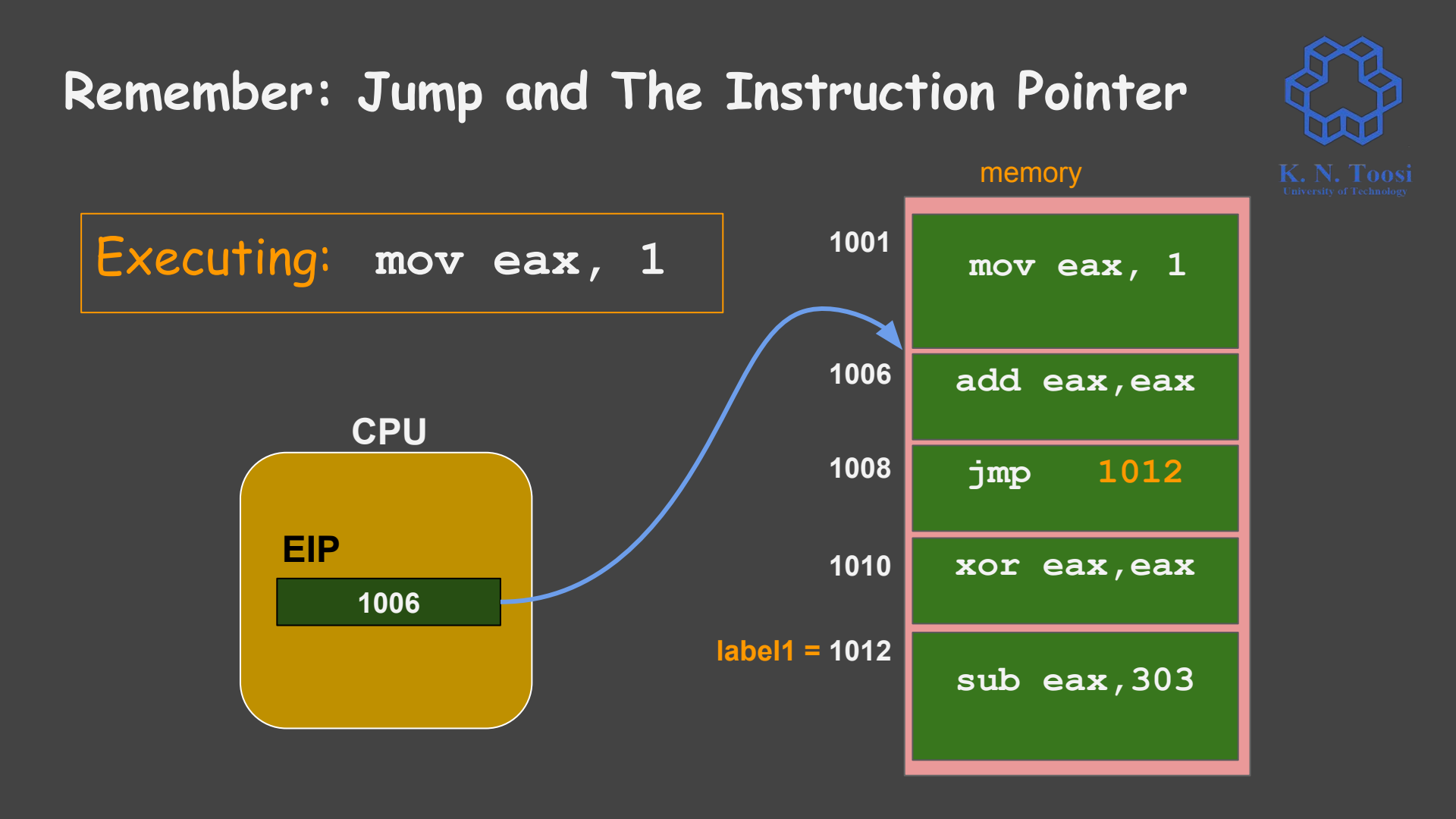


Remember: Jump and The Instruction Pointer

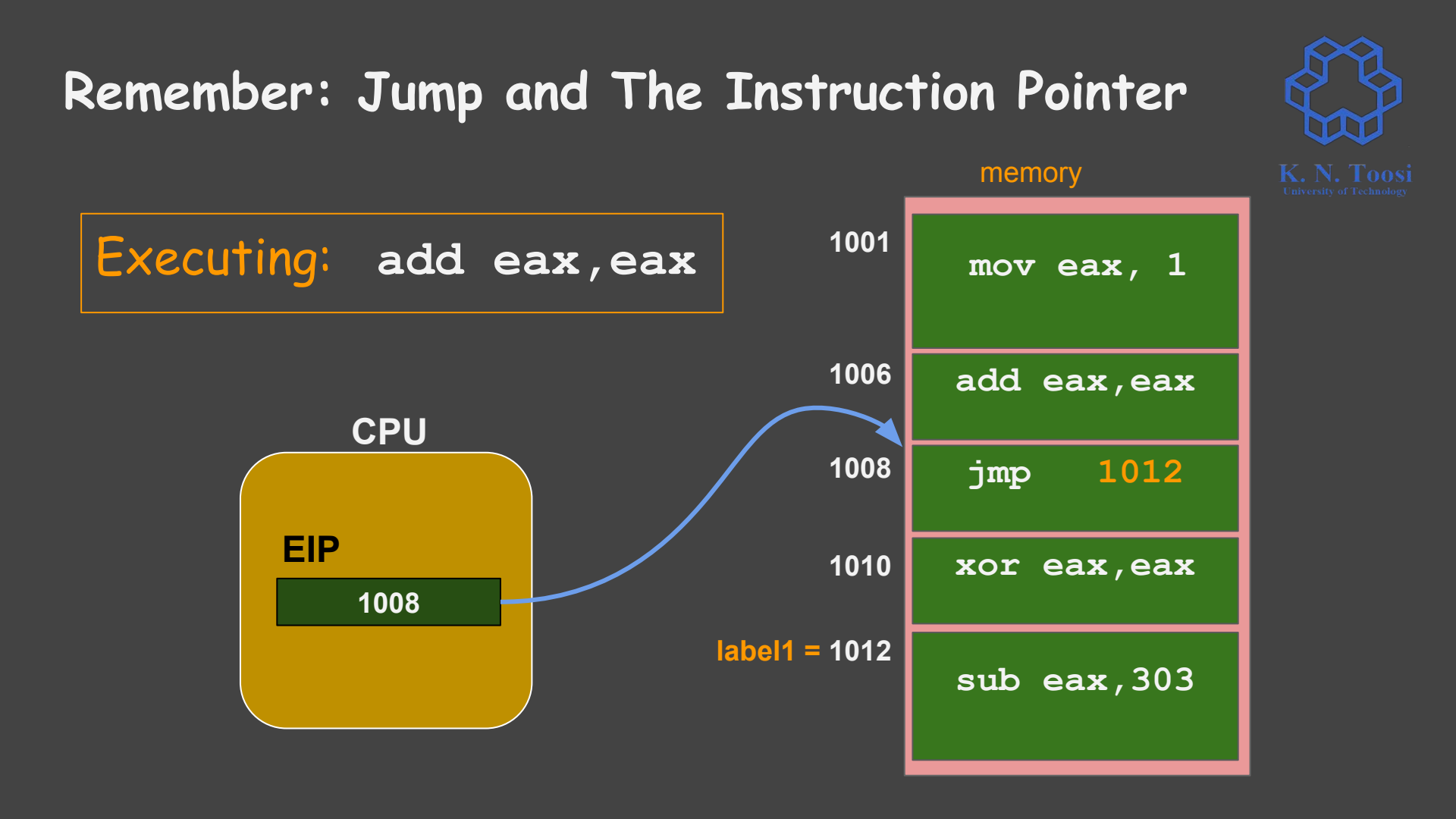
The Instruction Pointer
(program counter) IP, EIP, RIP



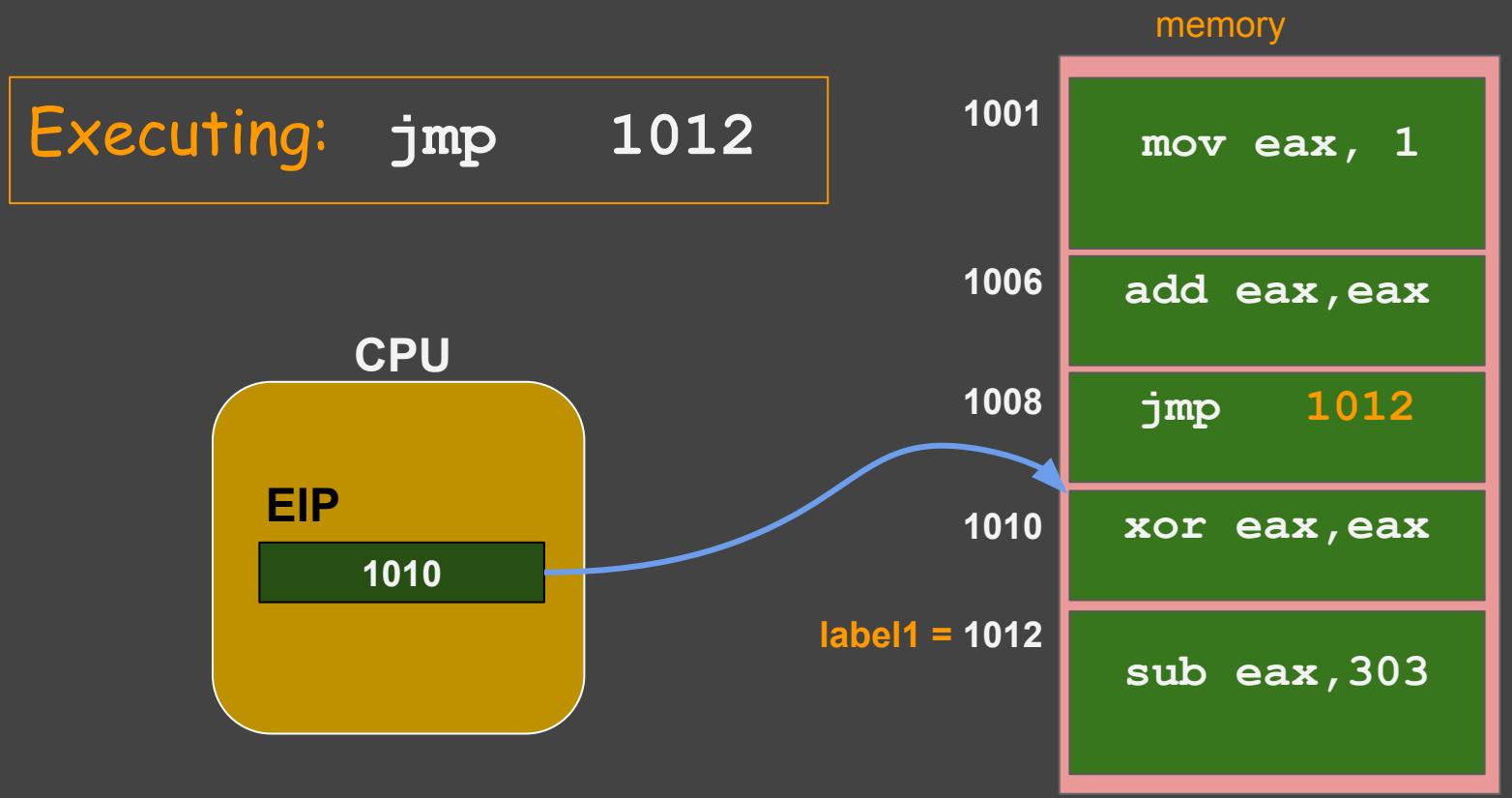
Remember: Jump and The Instruction Pointer



Remember: Jump and The Instruction Pointer



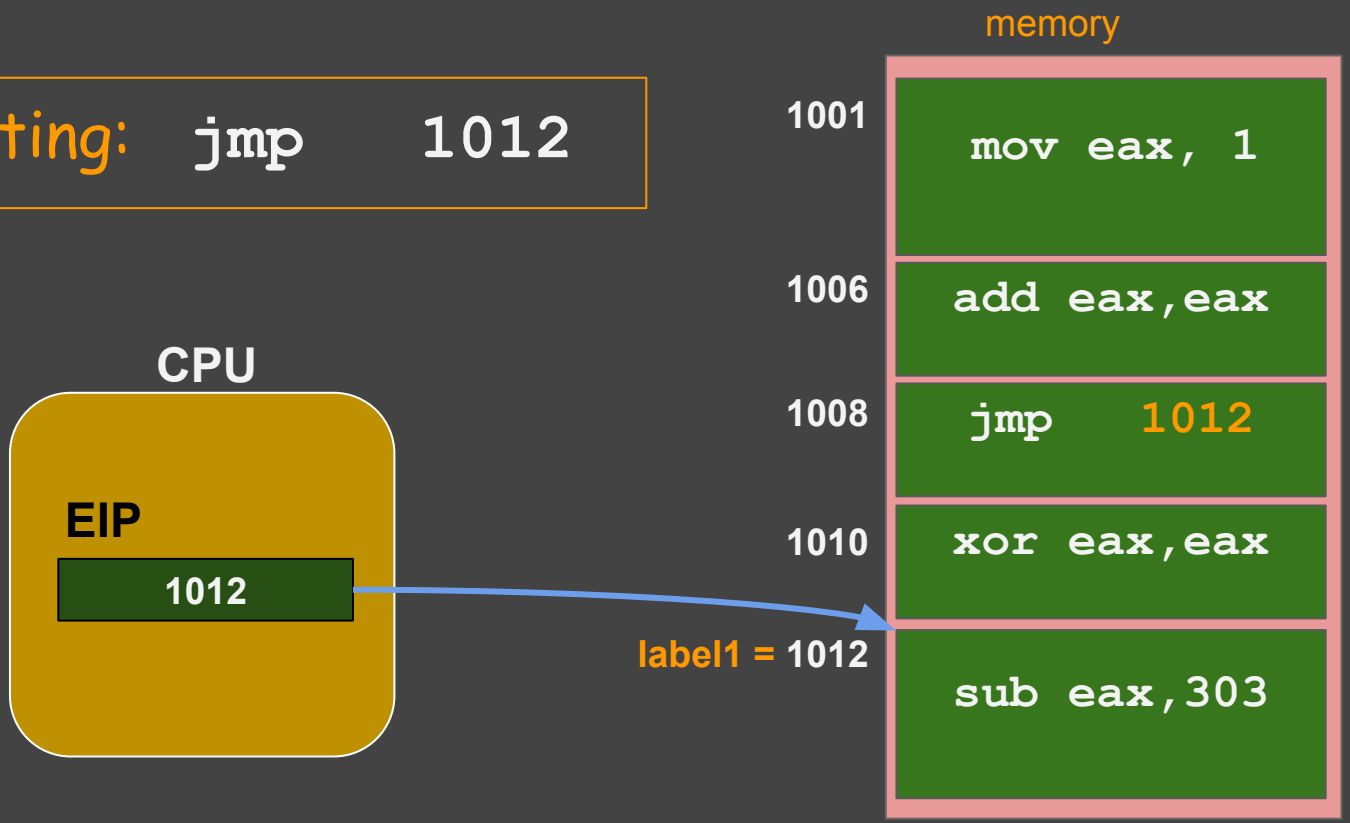
Remember: Jump and The Instruction Pointer



Remember: Jump and The Instruction Pointer

K. N. Toosi
University of Technology

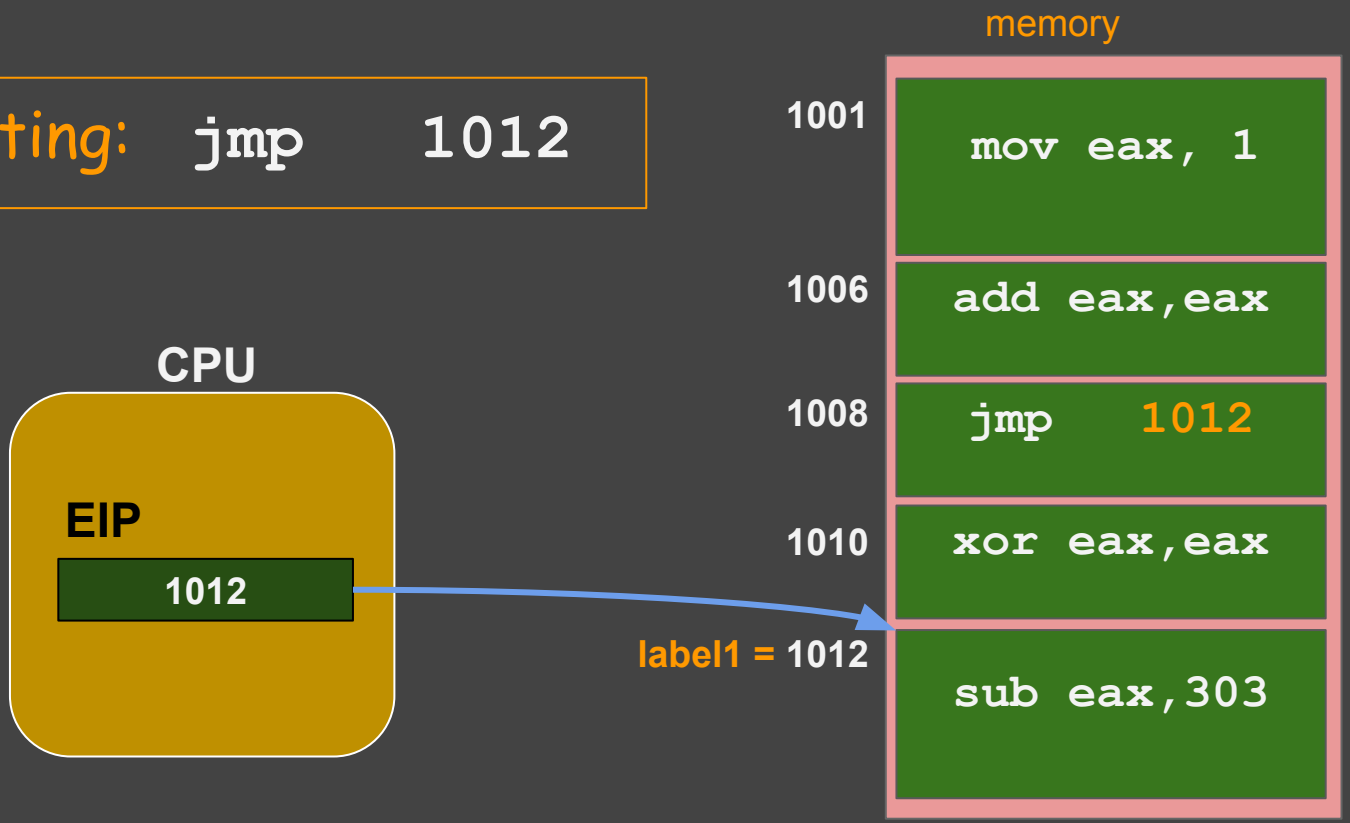
Executing: `jmp 1012`



Remember: Jump and The Instruction Pointer

K. N. Toosi
University of Technology

Executing: `jmp 1012`

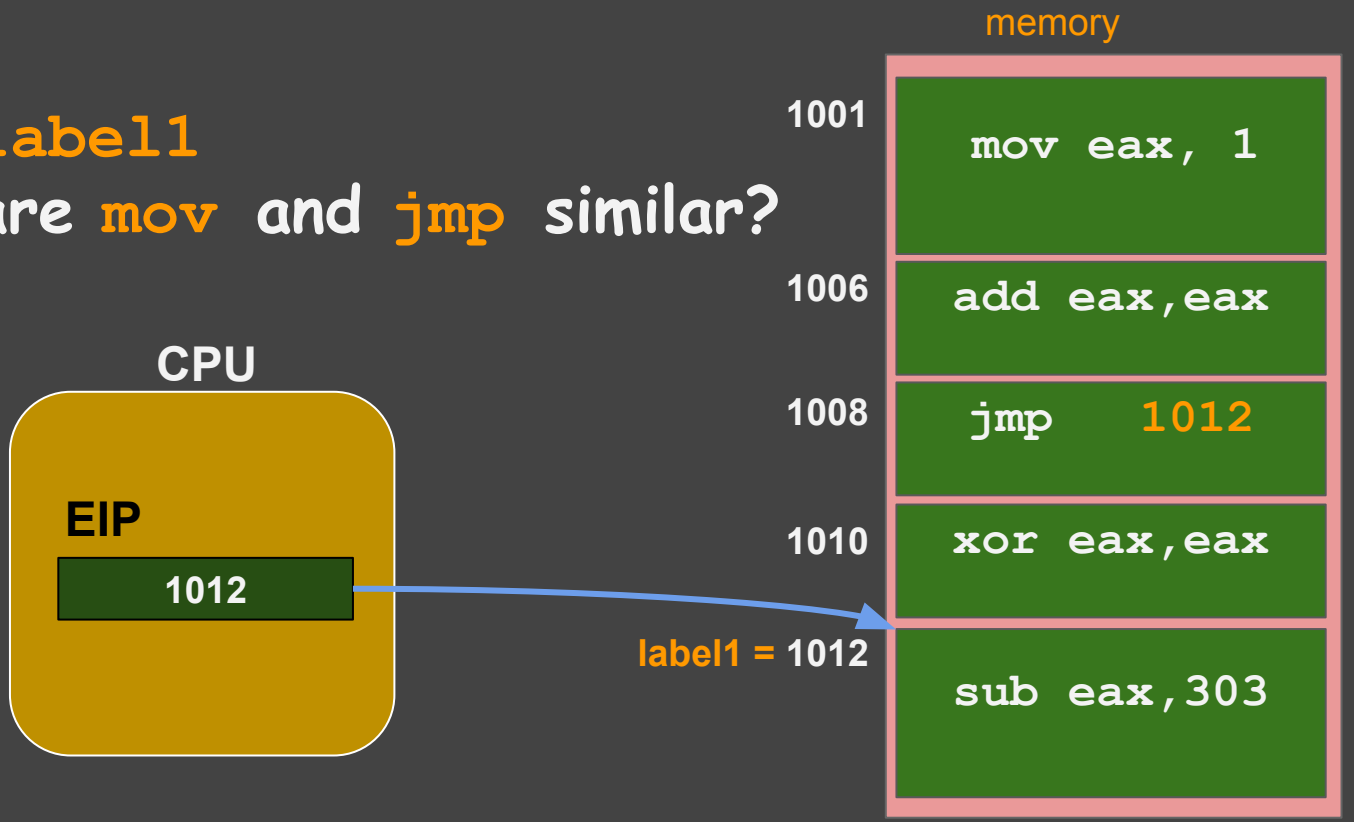


Remember: Jump and The Instruction Pointer

K. N. Toosi
University of Technology

`jmp label1`

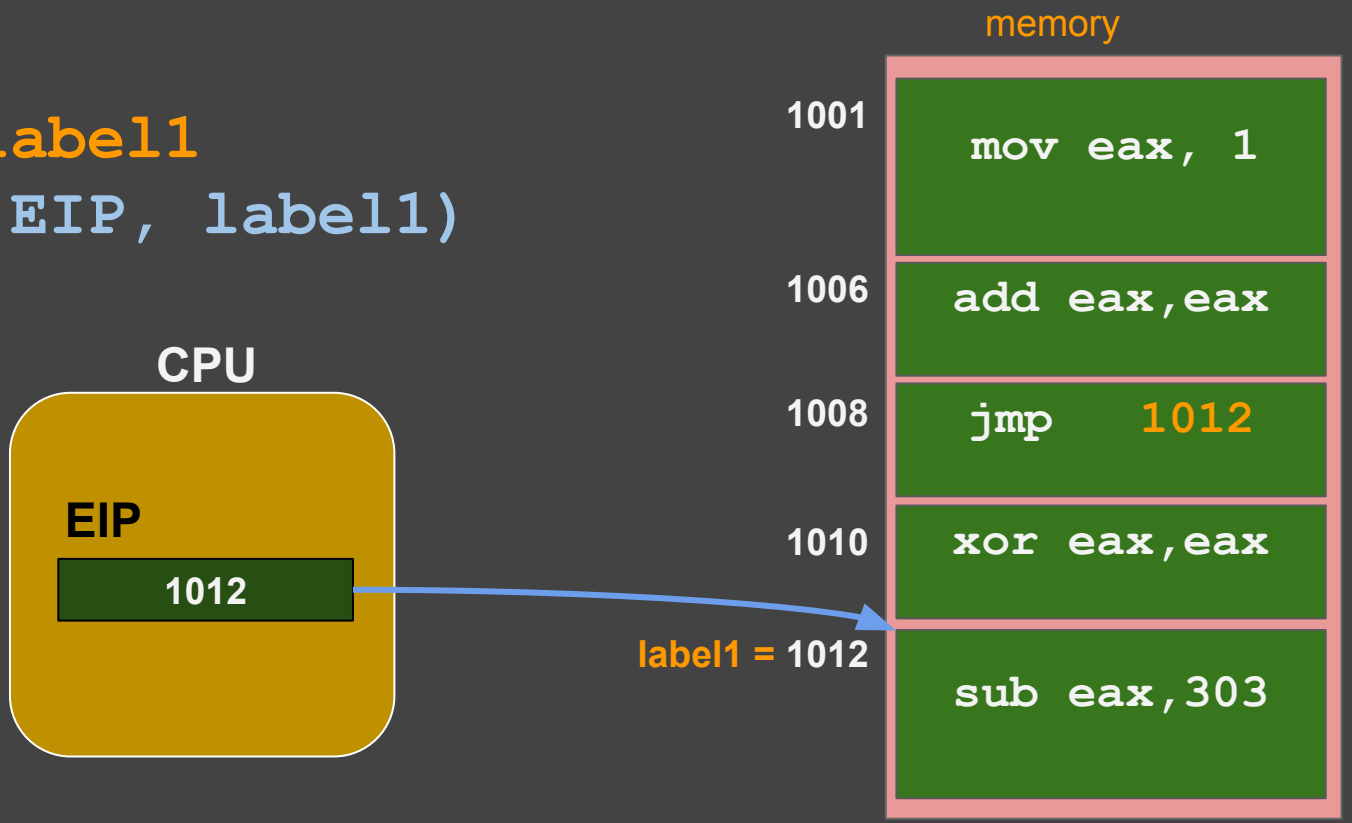
How are `mov` and `jmp` similar?



Remember: Jump and The Instruction Pointer

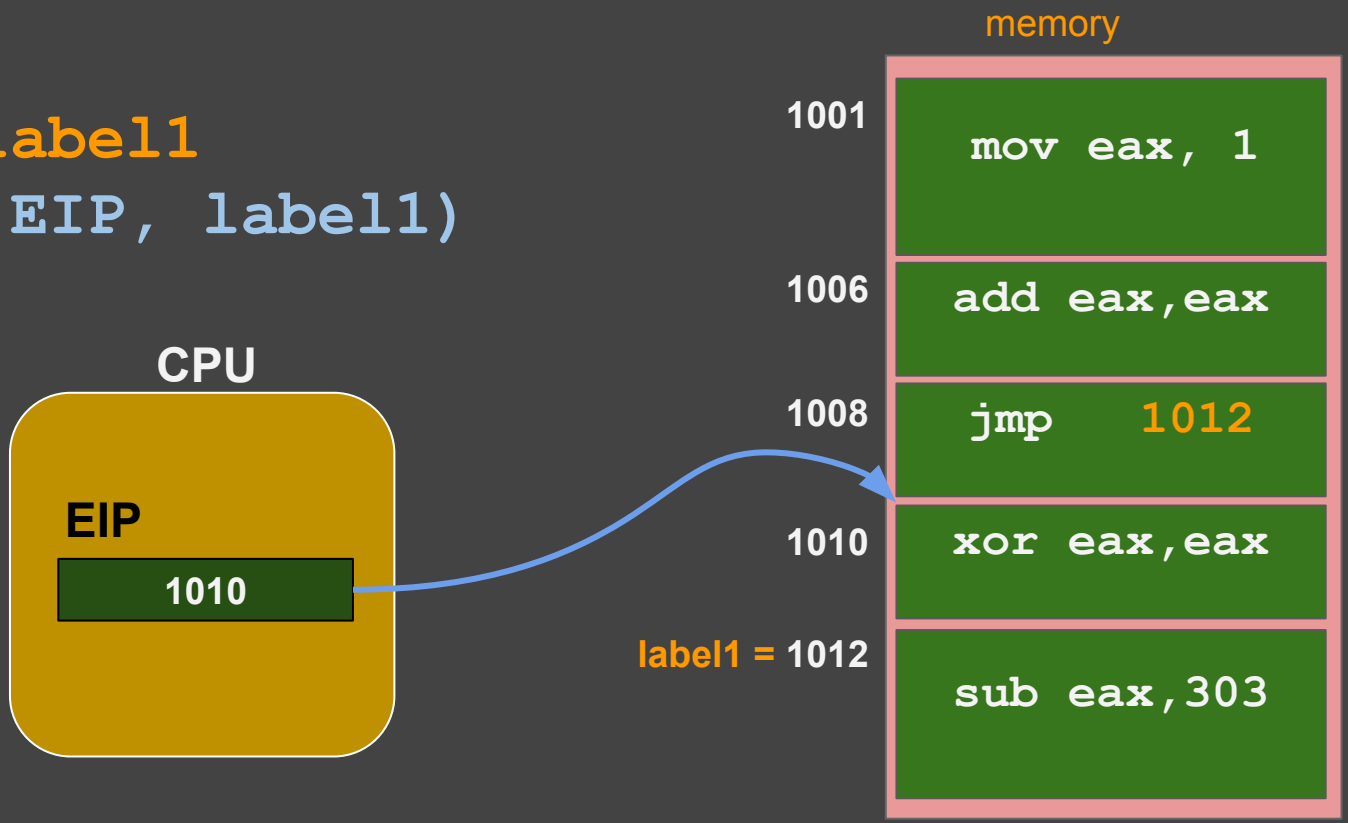
K. N. Toosi
University of Technology

```
jmp label1
(mov EIP, label1)
```



Remember: Jump and The Instruction Pointer

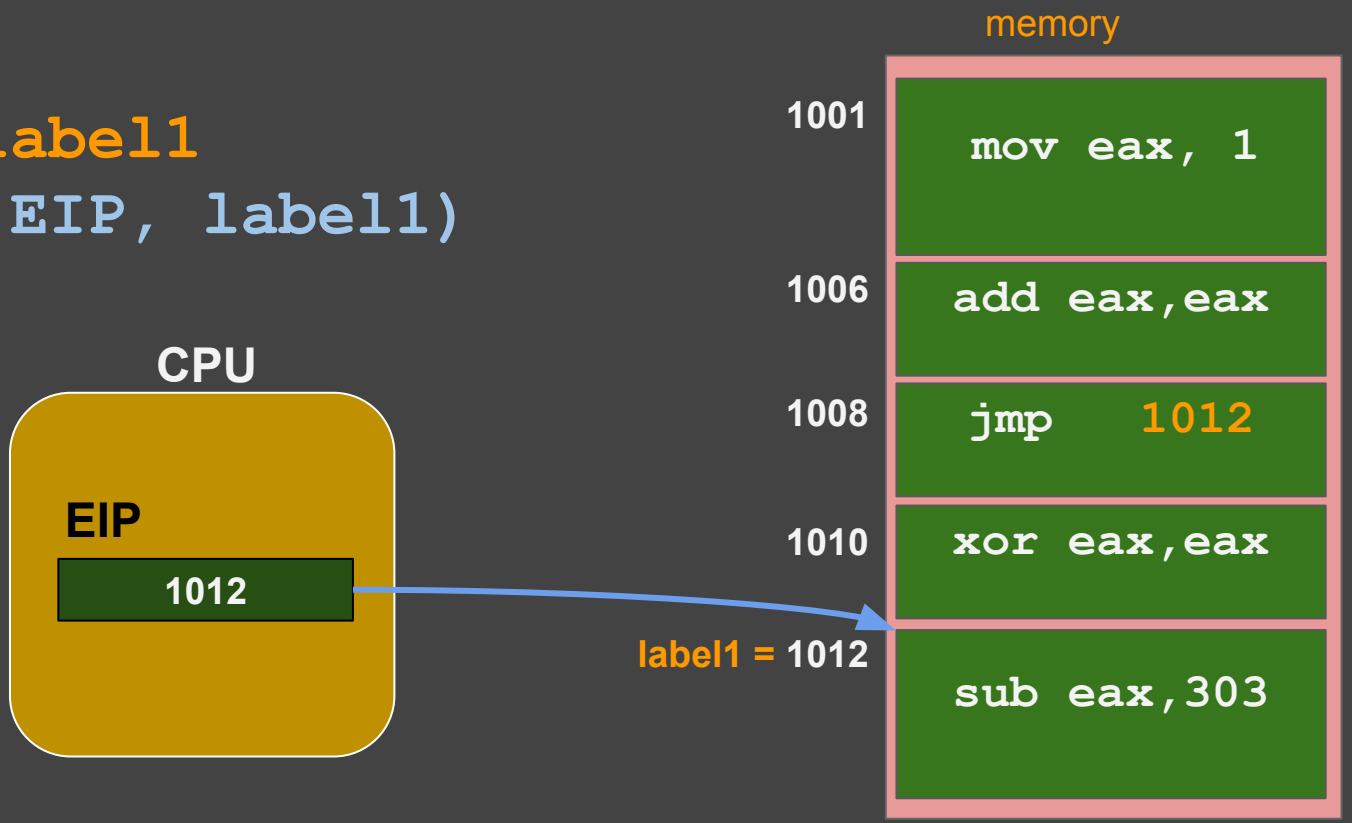

```
jmp label1
(mov EIP, label1)
```



Remember: Jump and The Instruction Pointer

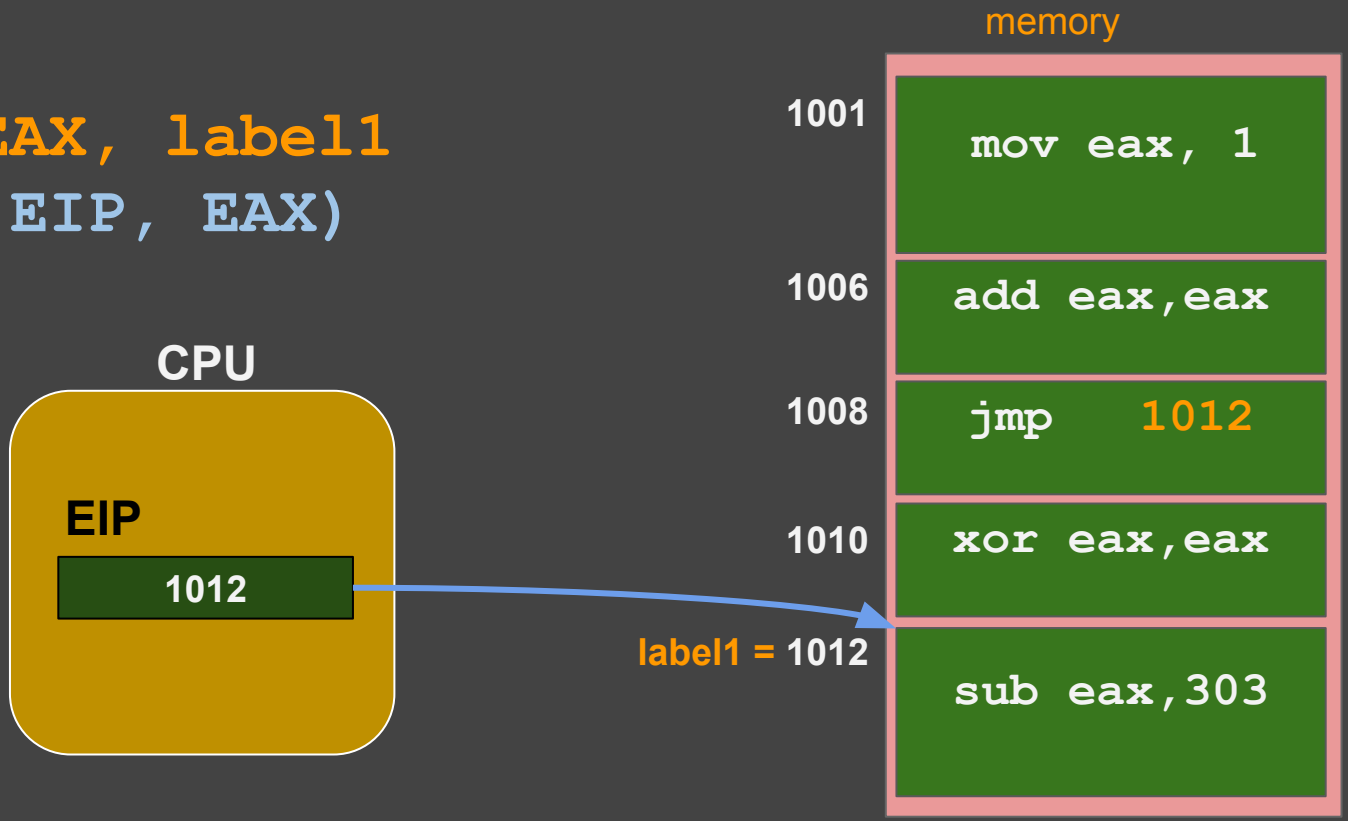
K. N. Toosi
University of Technology

```
jmp label1
(mov EIP, label1)
```



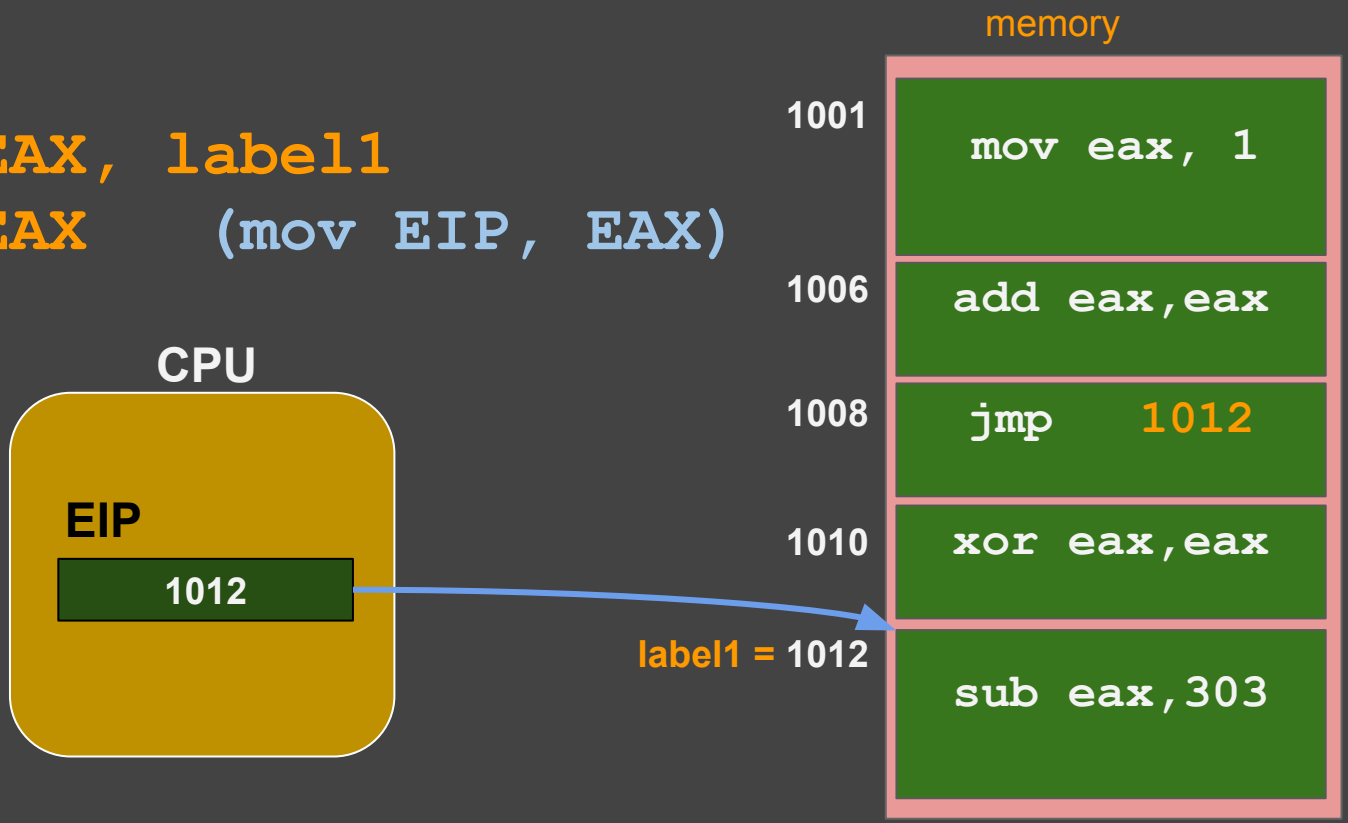
Remember: Jump and The Instruction Pointer


```
mov EAX, label1
(mov EIP, EAX)
```



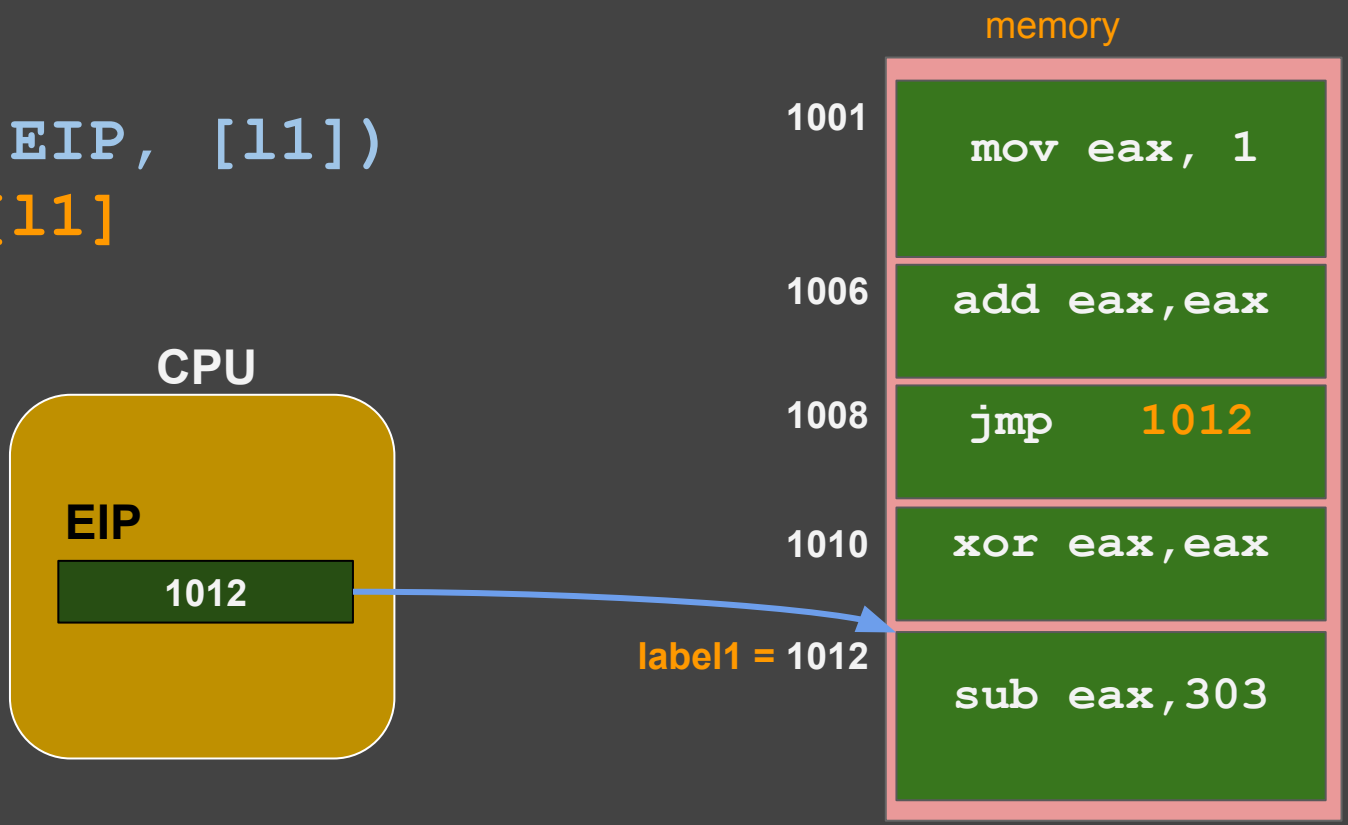
Remember: Jump and The Instruction Pointer


```
mov EAX, label1
jmp EAX     (mov EIP, EAX)
```



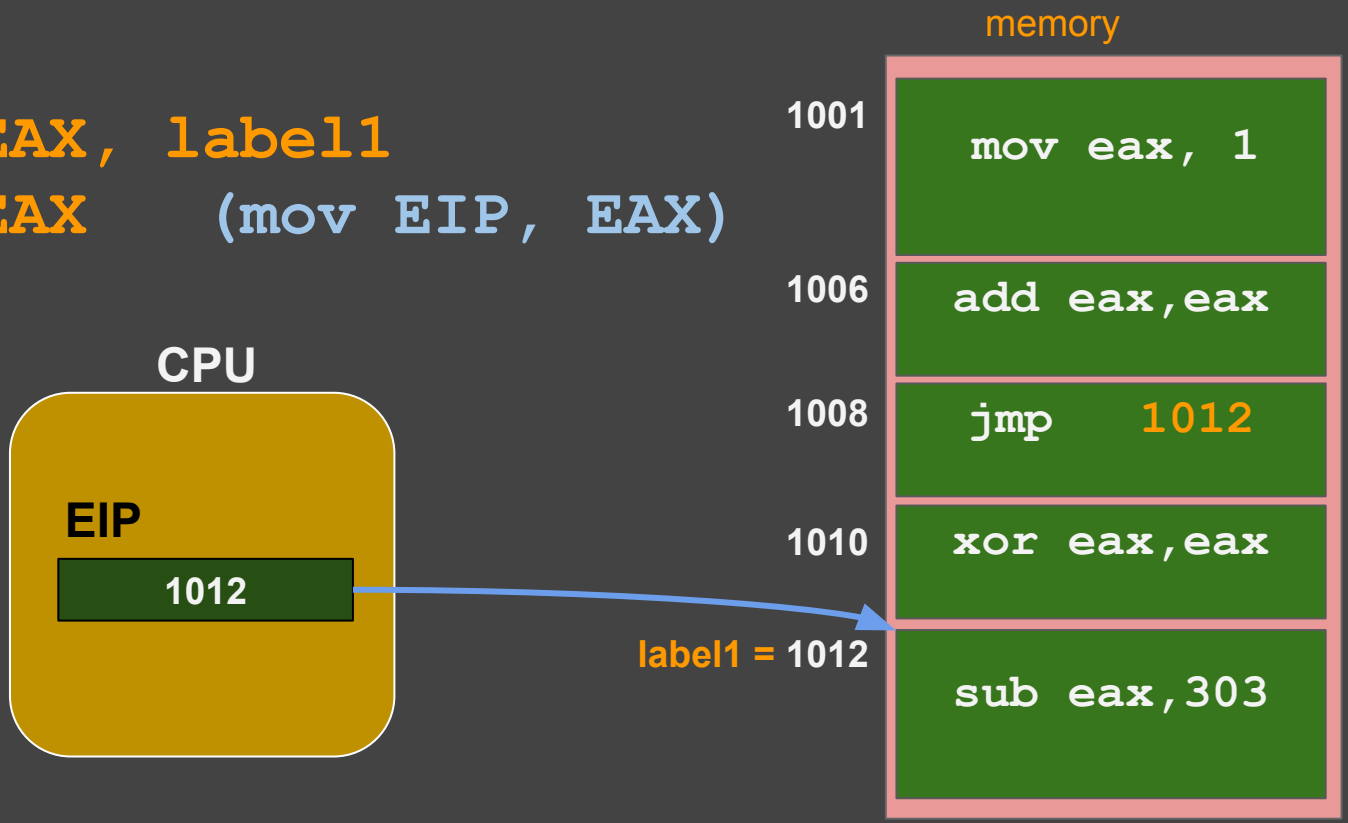
Remember: Jump and The Instruction Pointer


```
(mov EIP, [11])
jmp [11]
```



Remember: Jump and The Instruction Pointer


```
mov EAX, label1
jmp EAX     (mov EIP, EAX)
```



Indirect jump

Direct Jump: `jmp l1`

Indirect Jump:

```
mov  eax, l1
jmp  eax
```

How to implement subprograms?


```
void print_salam(void);

int main() {

  print_salam();

}

void print_salam() {
  printf("Salaaaaam!\n");
}
```

simplefunc3.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮

        jmp print_salam
l1:

        jmp print_salam
l2:
        ⋮
print_salam:
        mov eax, msg
        call print_string
        jmp ?
```

return address

How to implement subprograms?


```c
void print_salam(void);

int main() {

  print_salam();

}

void print_salam() {
  printf("Salaaaaam!\n");
}
```

simplefunc3.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮

        jmp print_salam
l1:

        jmp print_salam
l2:
        ⋮
print_salam:
        mov eax, msg
        call print_string
        jmp edx
```

return address

How to implement subprograms?


```
void print_salam(void);

int main() {

  print_salam();

}

void print_salam() {
  printf("Salaaaaam!\n");
}
```

simplefunc3.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮
        mov edx, l1
        jmp print_salam
l1:                          return address
        mov edx, l2
        jmp print_salam
l2:
        ⋮
print_salam:
        mov eax, msg
        call print_string
        jmp edx
```

How to implement subprograms?


```
void print_salam(void);

int main() {

  print_salam();

}

void print_salam() {
  printf("Salaaaaam!\n");
}
```

Limitations?

simplefunc3.asm

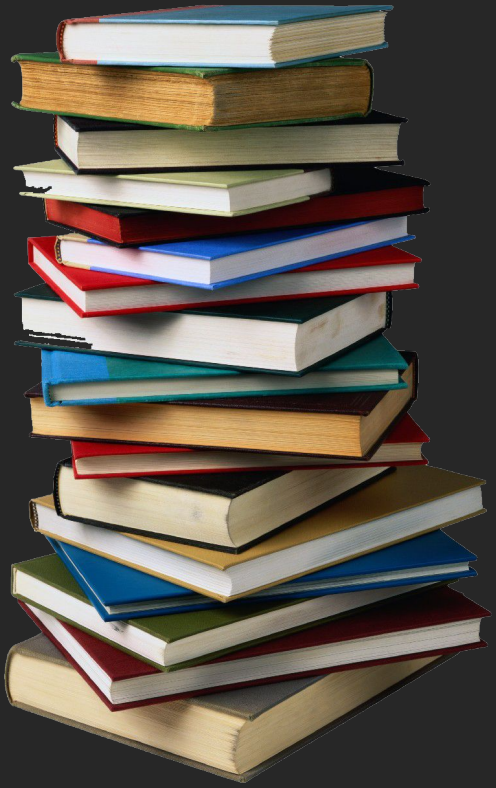
```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮
        mov edx, l1
        jmp print_salam
l1:
        mov edx, l2
        jmp print_salam
l2:
        ⋮
print_salam:
        mov eax, msg
        call print_string
        jmp edx
```

return address

The stack

http://freepngimg.com/png/25783-coin-stack-transparent-image

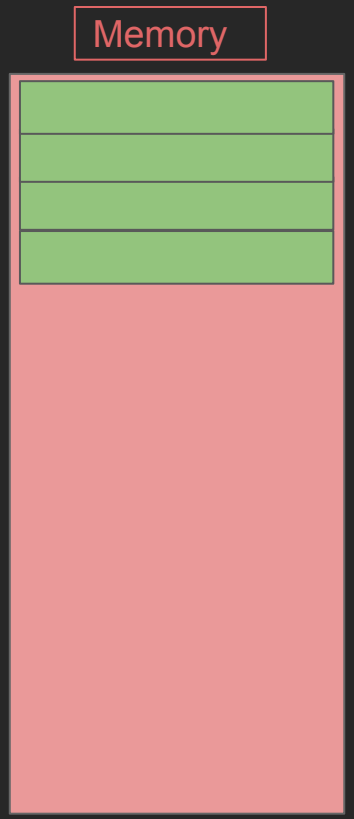
https://pixabay.com/en/plate-stack-tableware-plate-stack-629970/



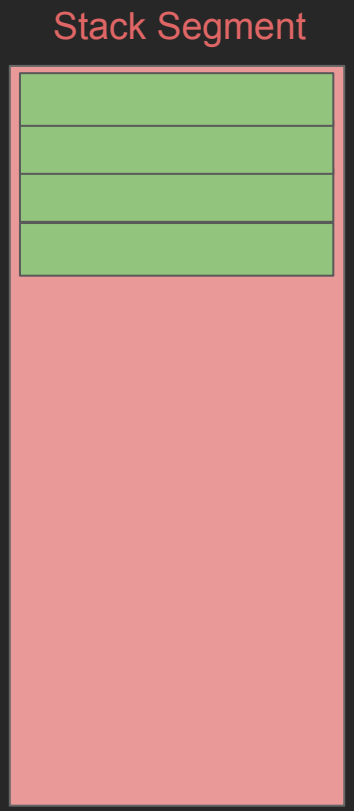
http://carbon.materialwitness.co/book-stack/

K. N. Toosi
University of Technology

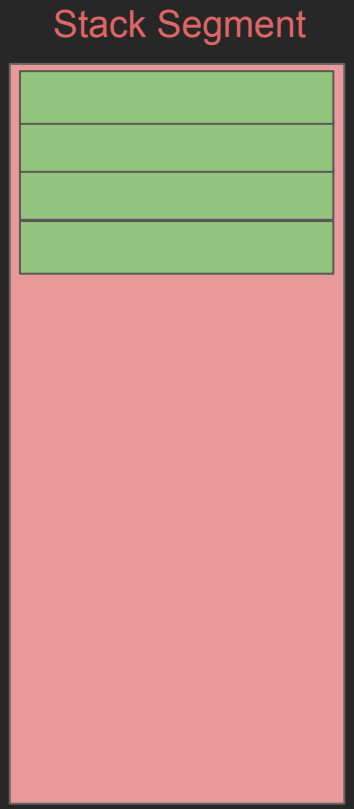
The stack



Implementing the stack



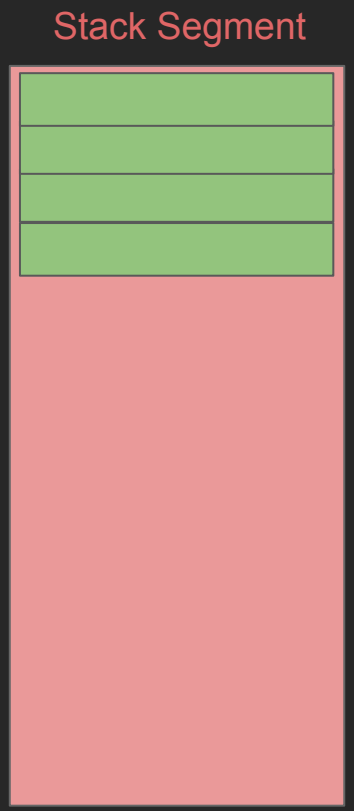
Implementing the stack



x86

K. N. Toosi
University of Technology

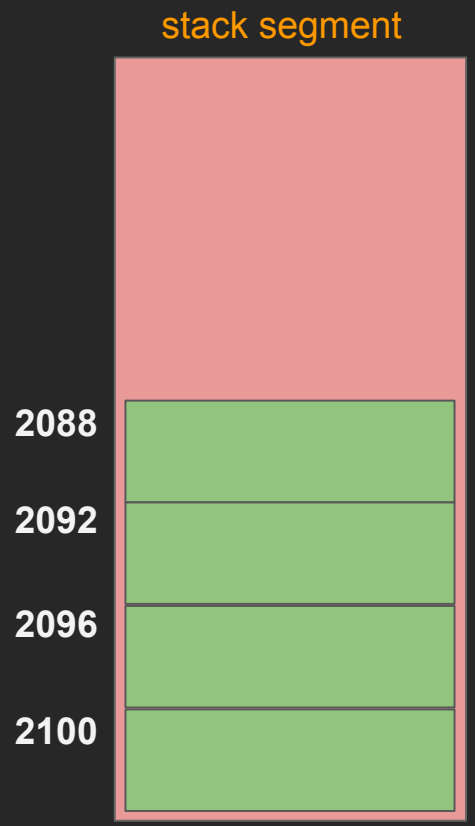
Implementing the stack



x86
(why?)

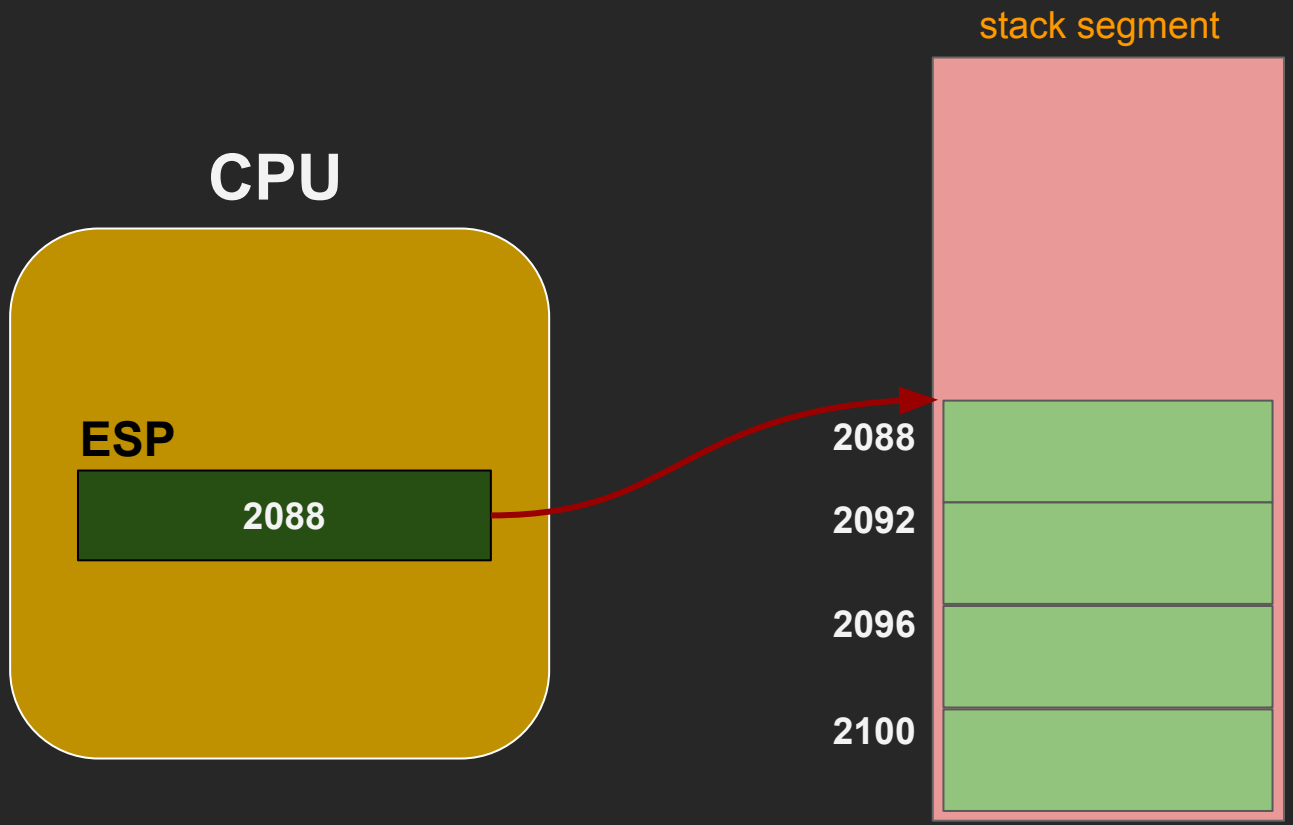
K. N. Toosi
University of Technology

Implementing the stack



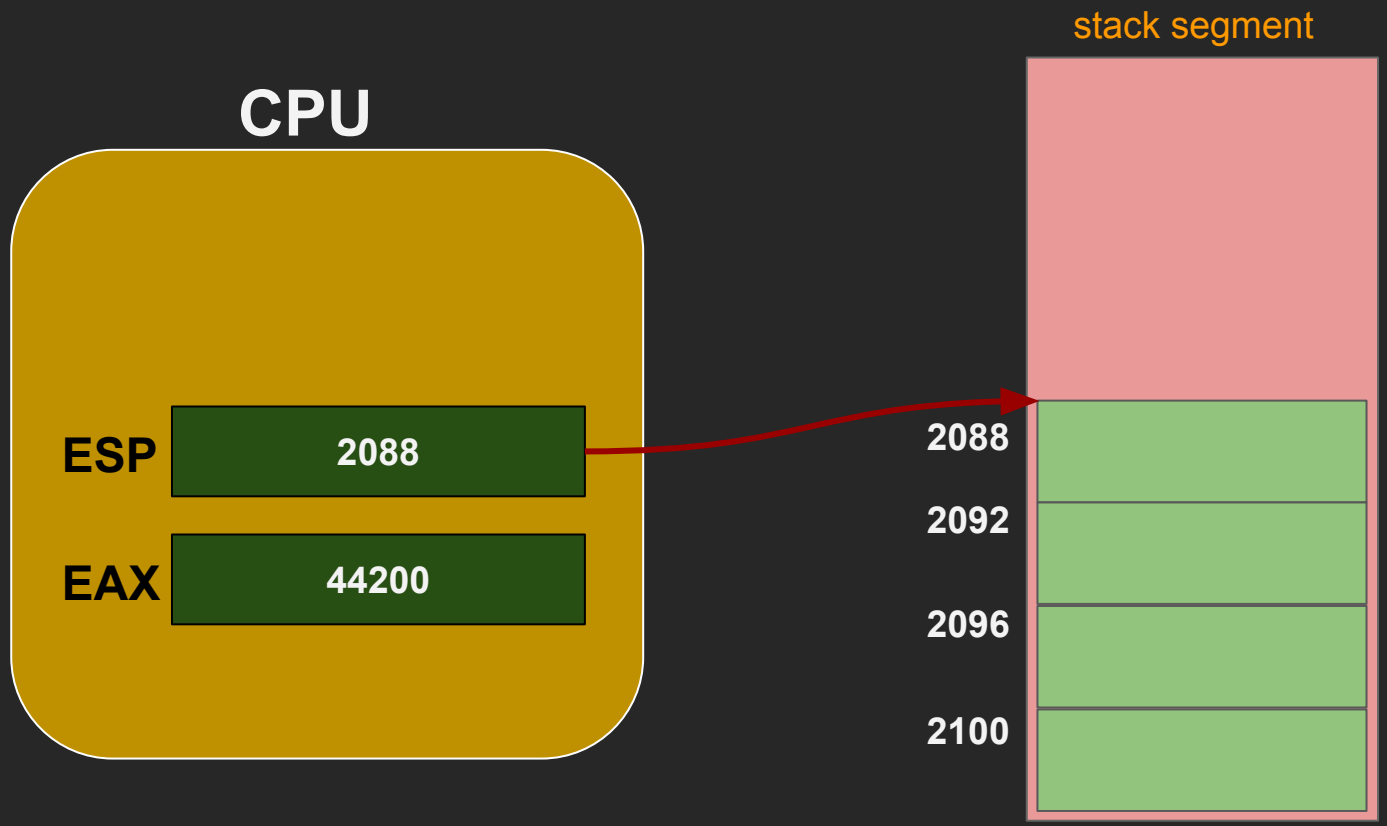
Stack Pointer (SP, ESP, RSP)

K. N. Toosi
University of Technology

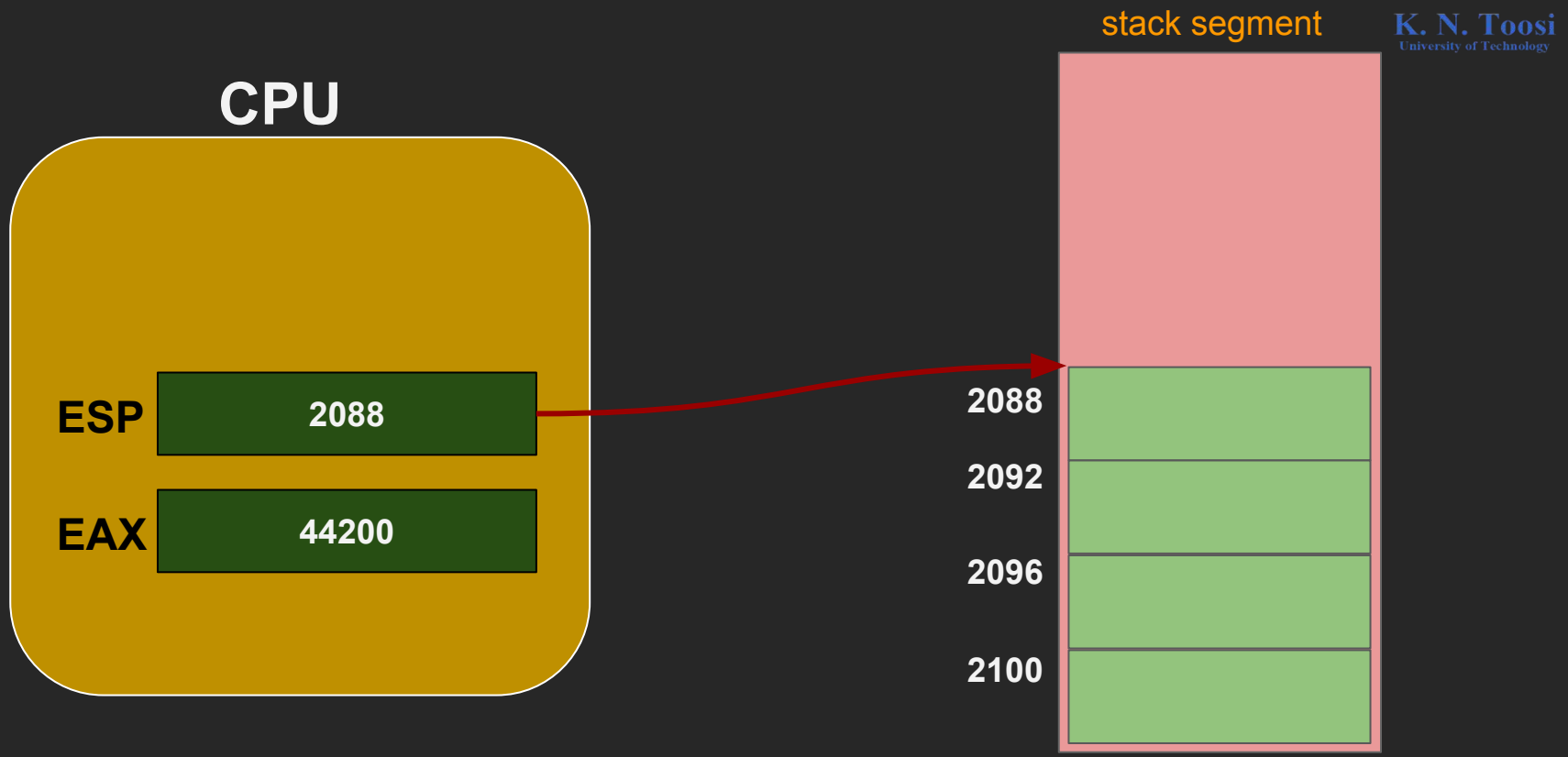


Pushing on the stack

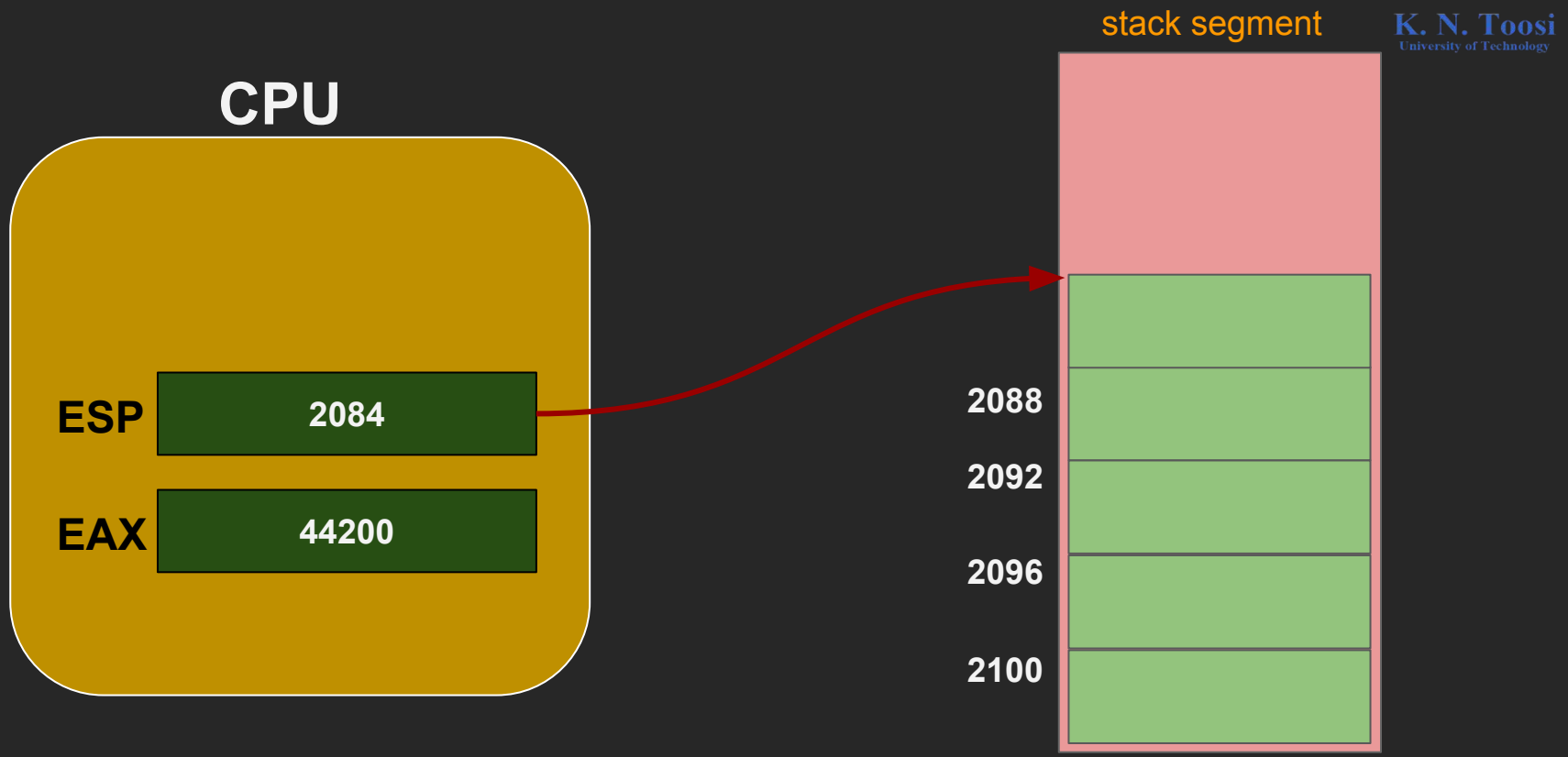
K. N. Toosi
University of Technology



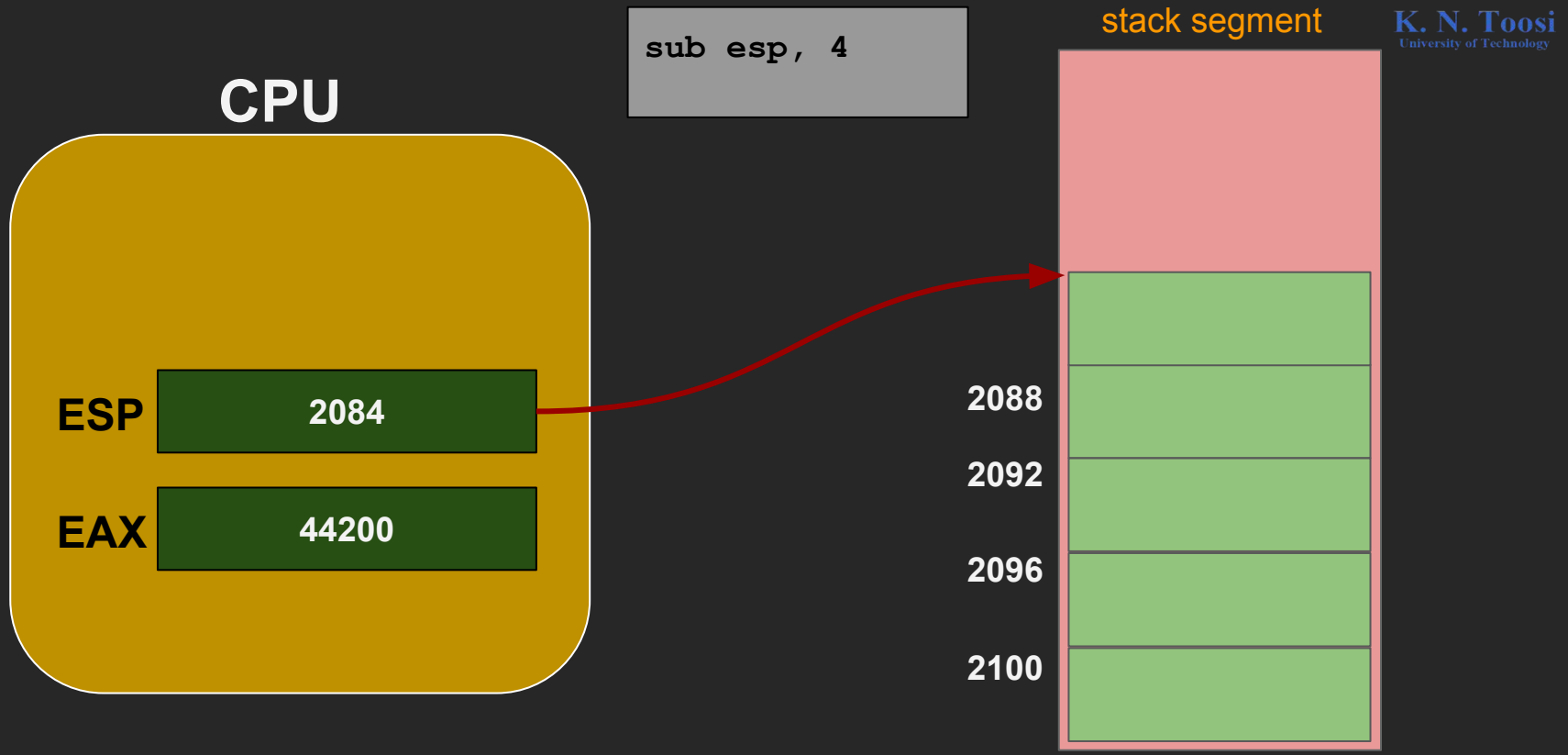
Push EAX on the stack



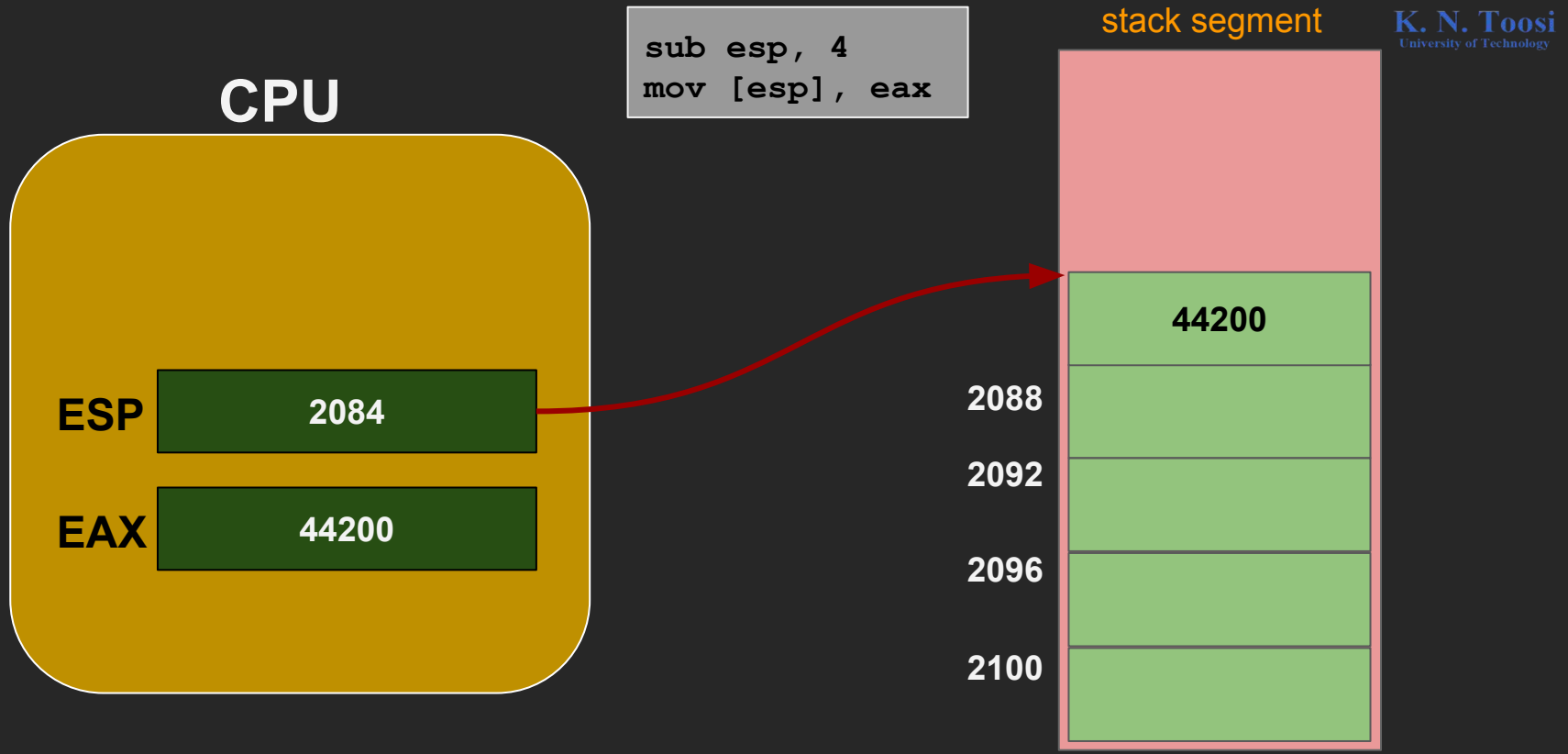
Push EAX on the stack



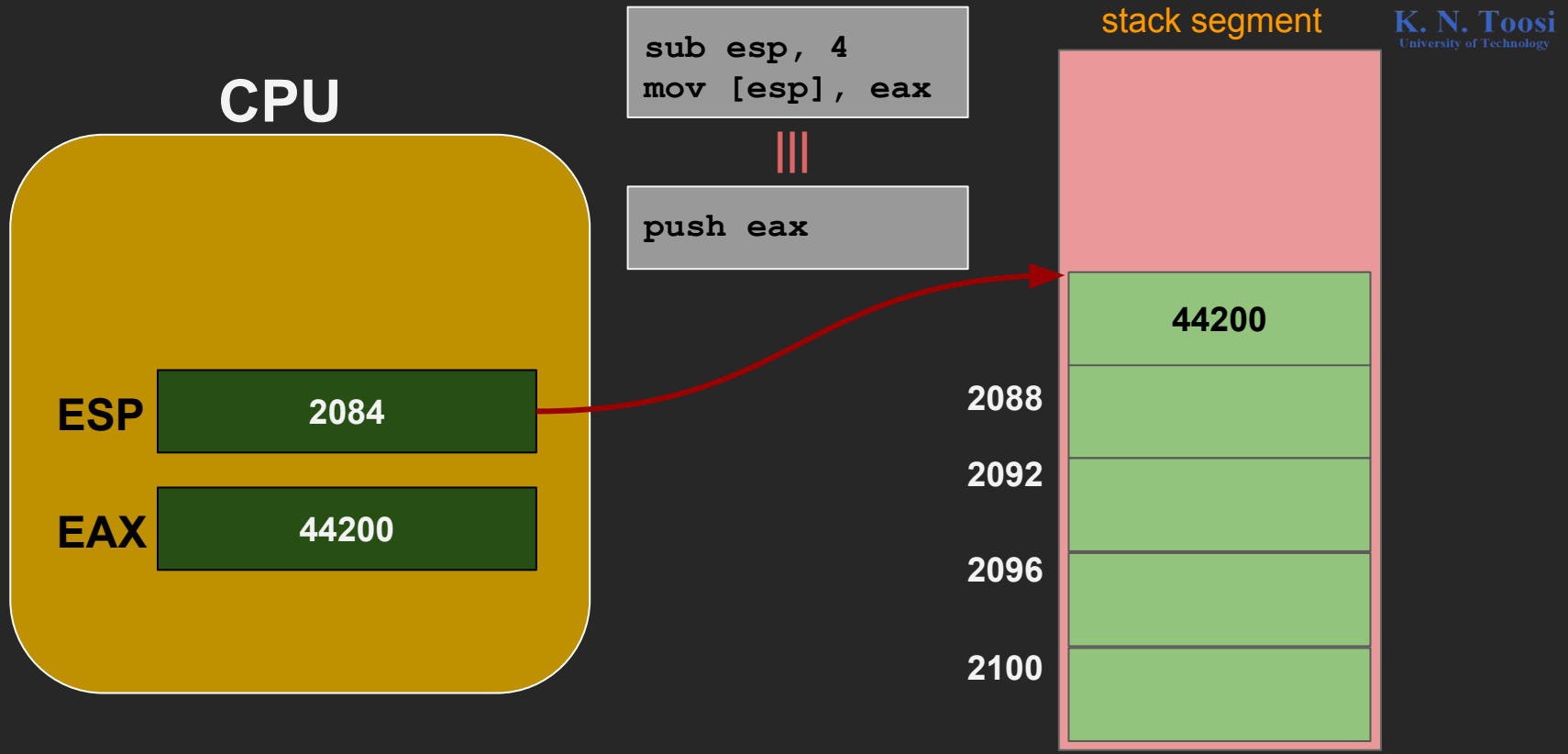
Push EAX on the stack



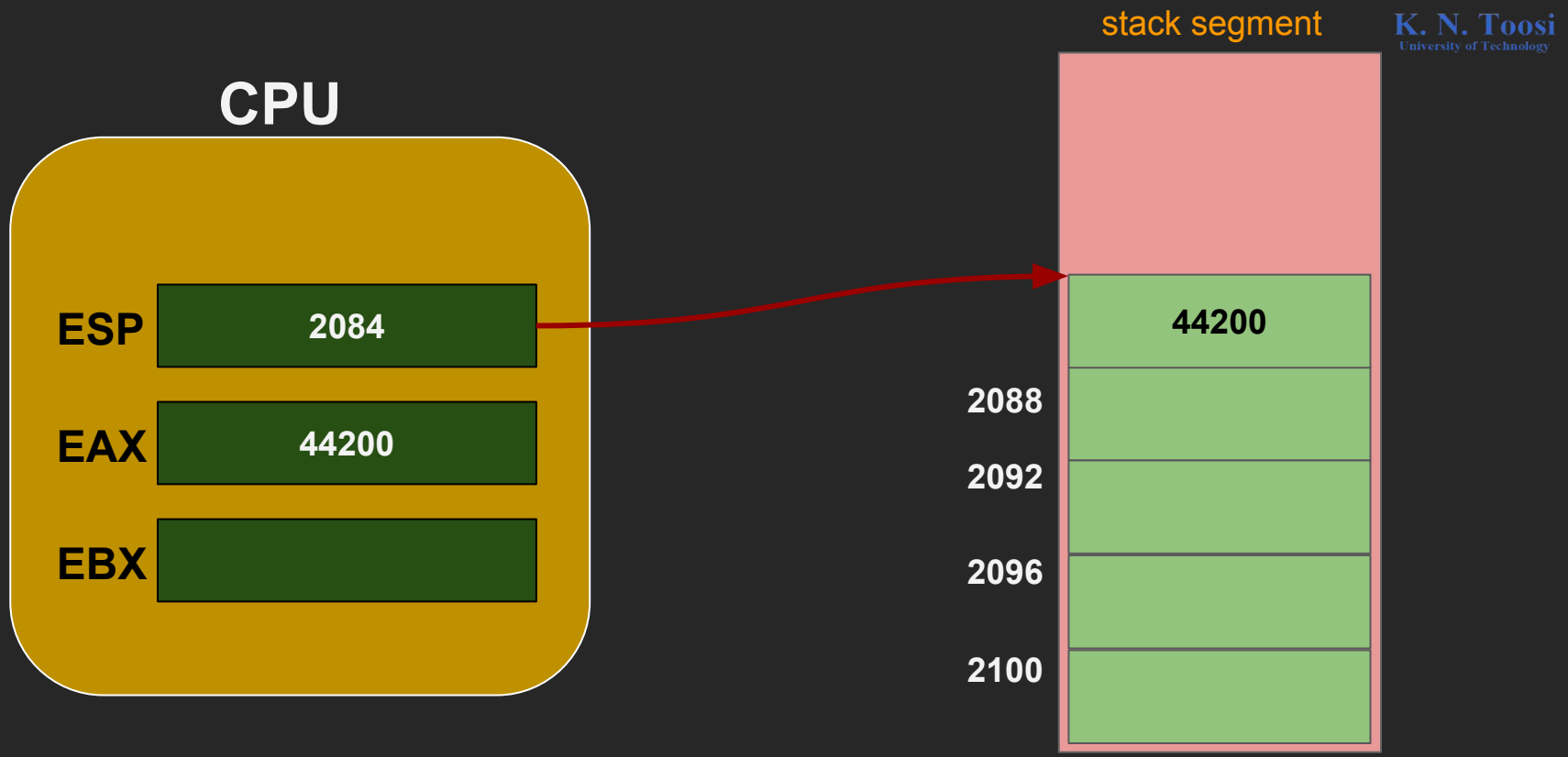
Push EAX on the stack



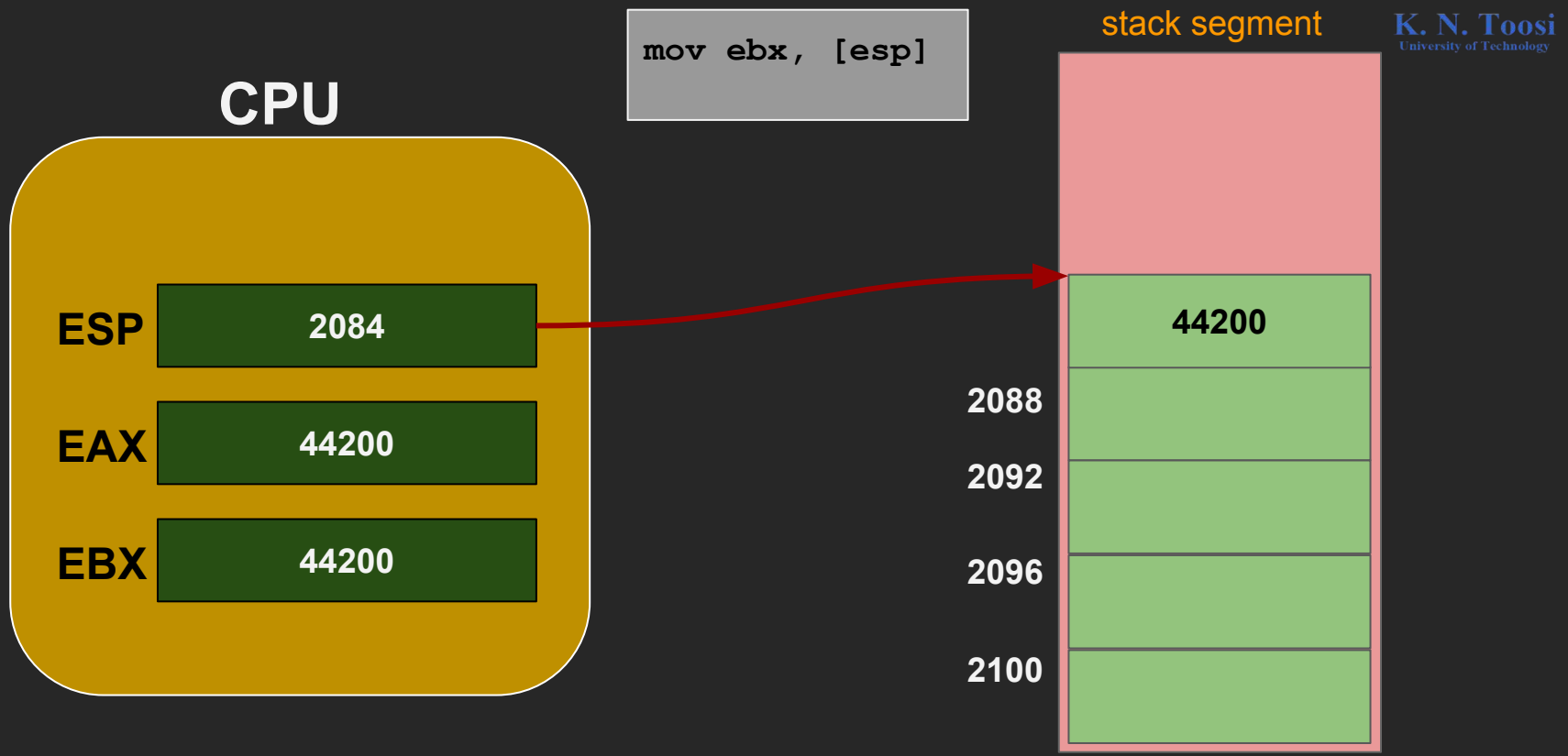
Push EAX on the stack



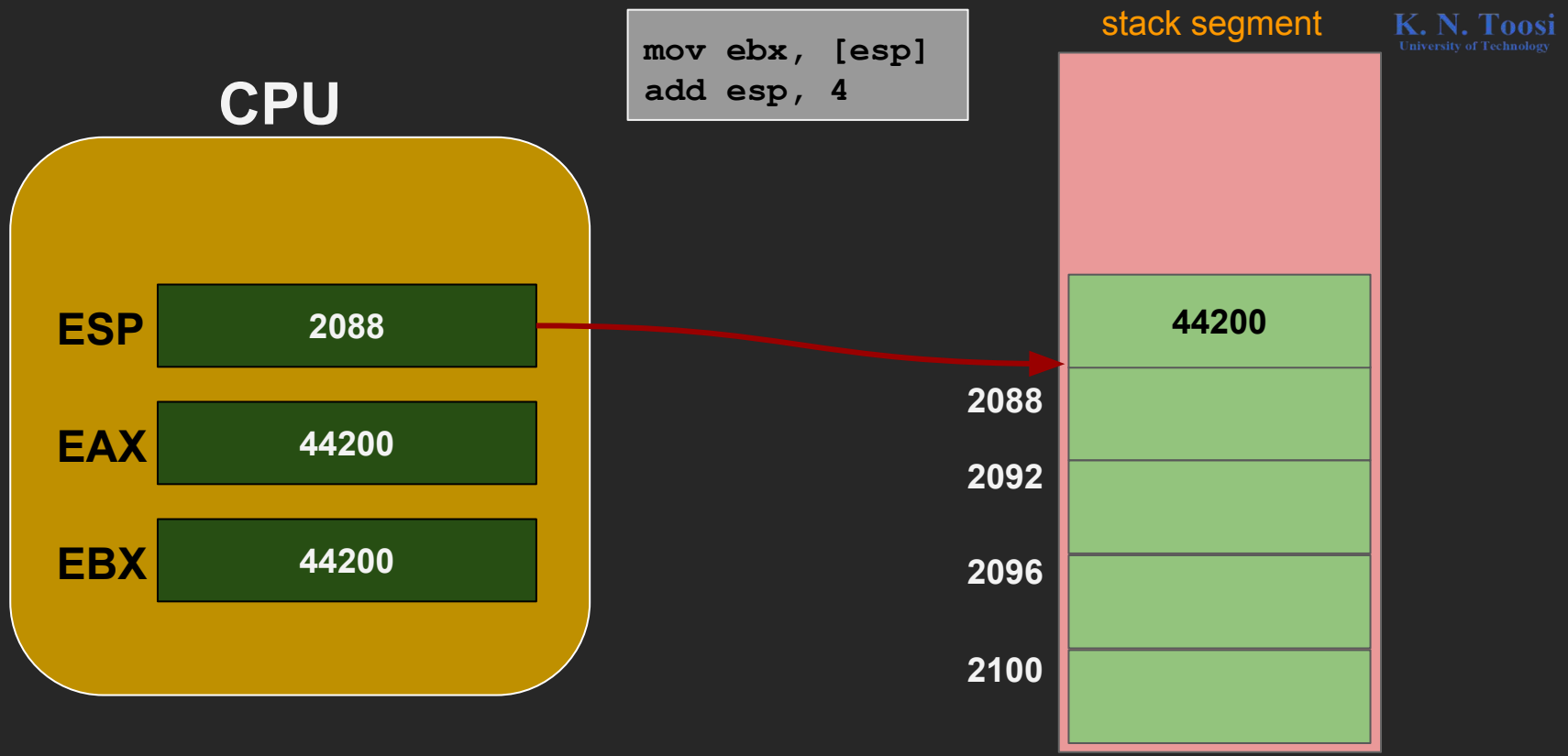
Pop into EBX


CPU
ESP
2084
EAX
44200
EBX
stack segment
44200
2088
2092
2096
2100

Pop into EBX

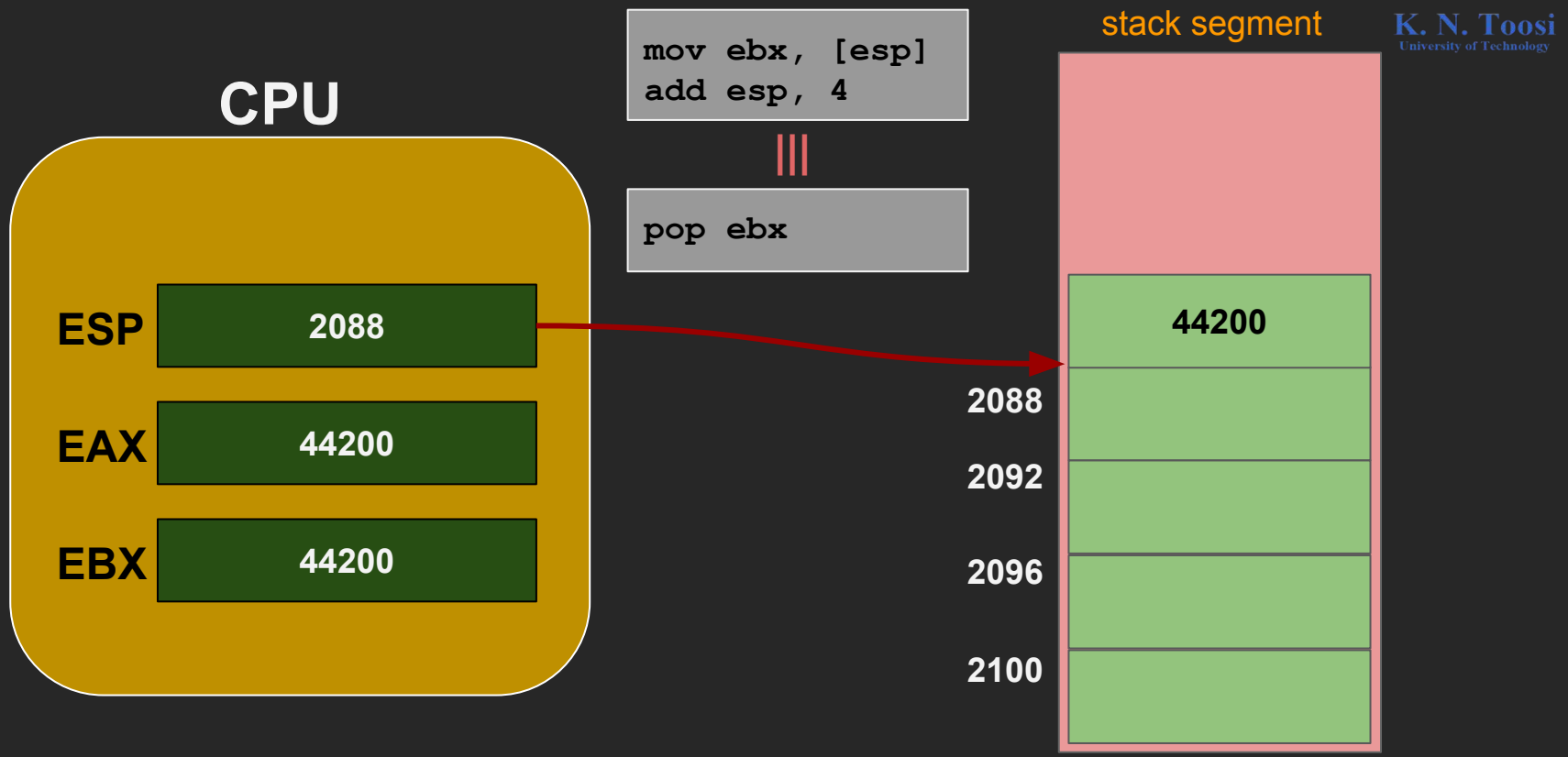


Pop into EBX

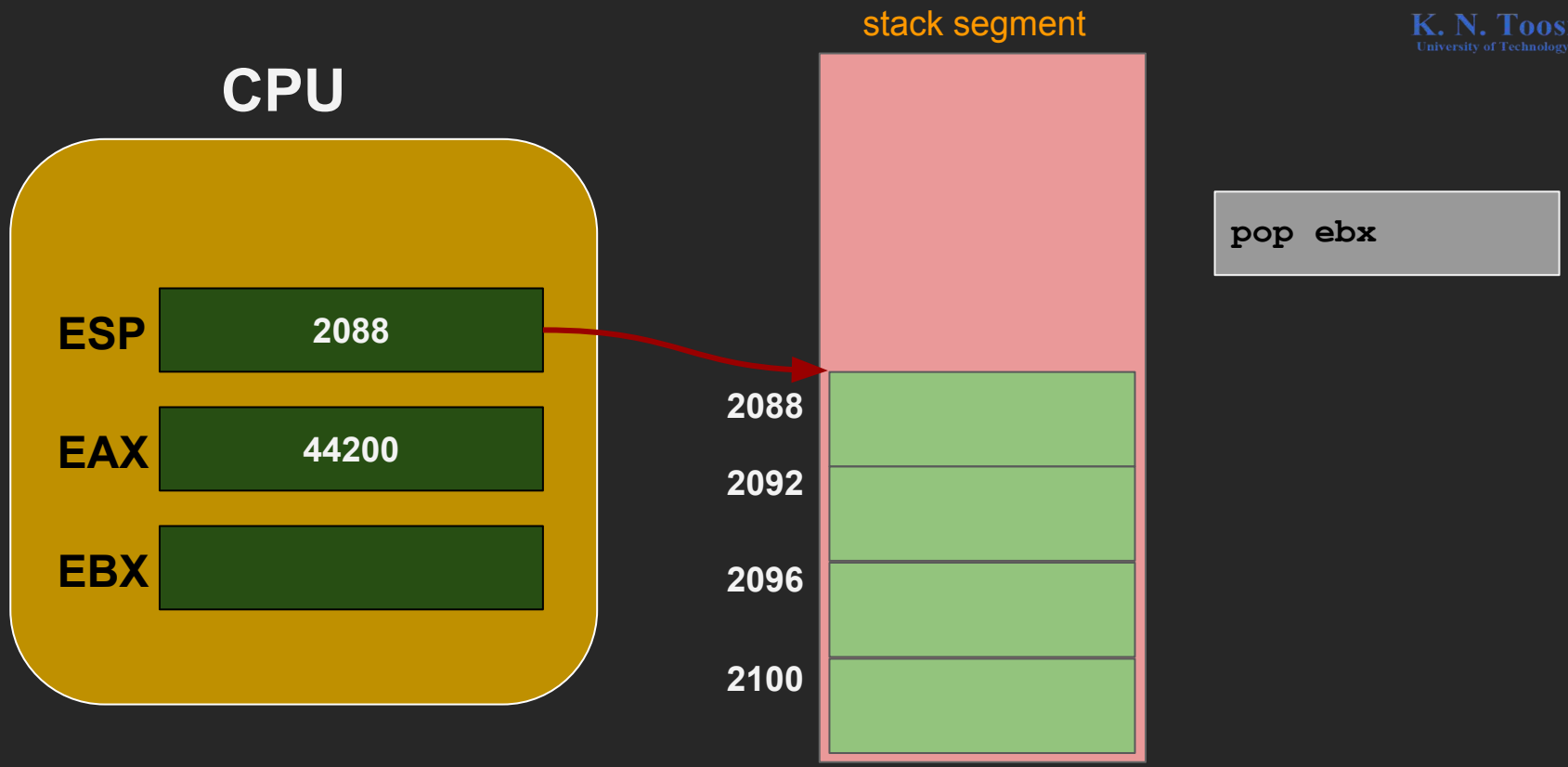


K. N. Toosi
University of Technology

Pop into EBX

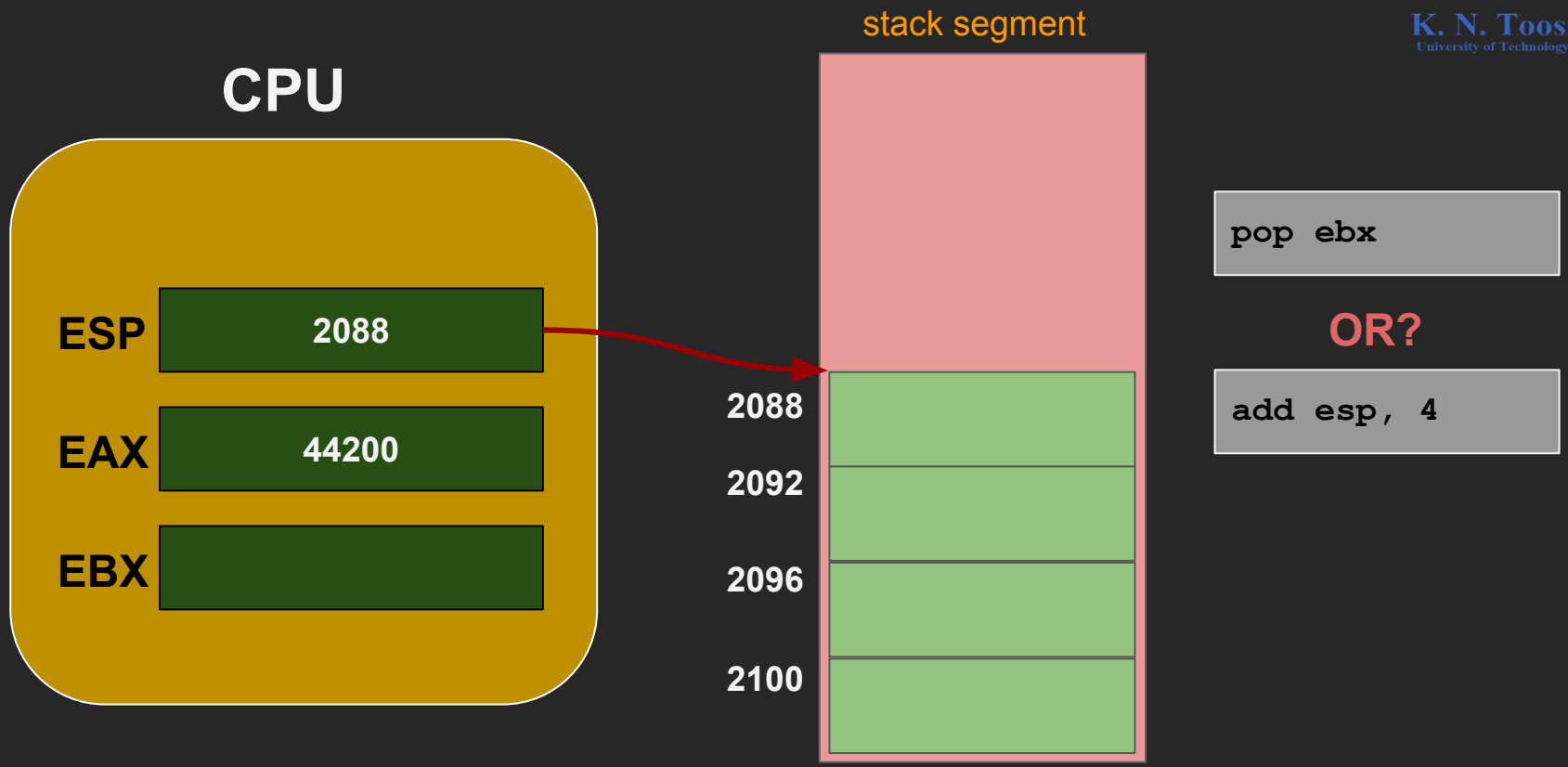


just pop 4 bytes (store nowhere)



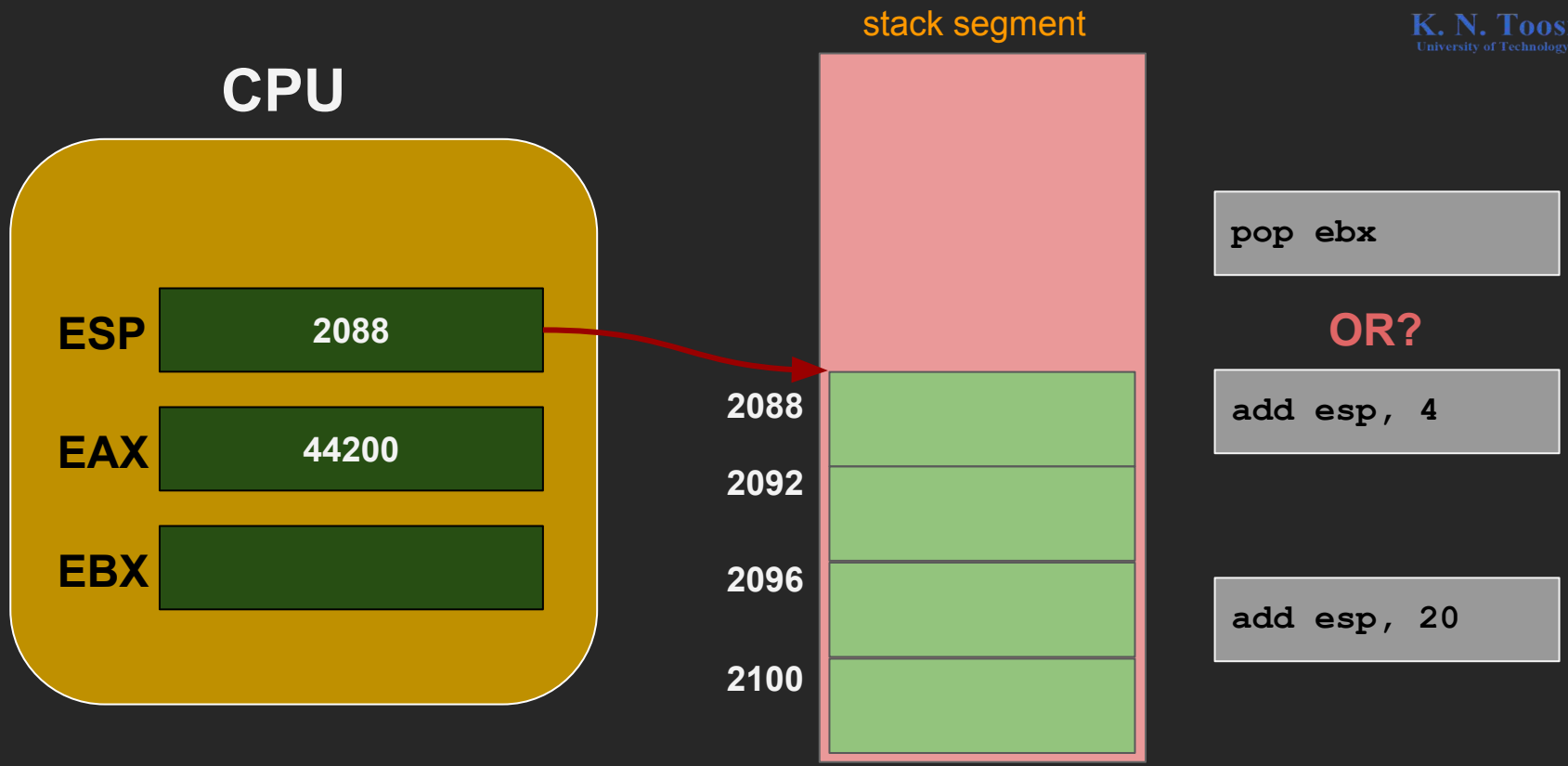
just pop 4 bytes (store nowhere)

K. N. Toosi
University of Technology

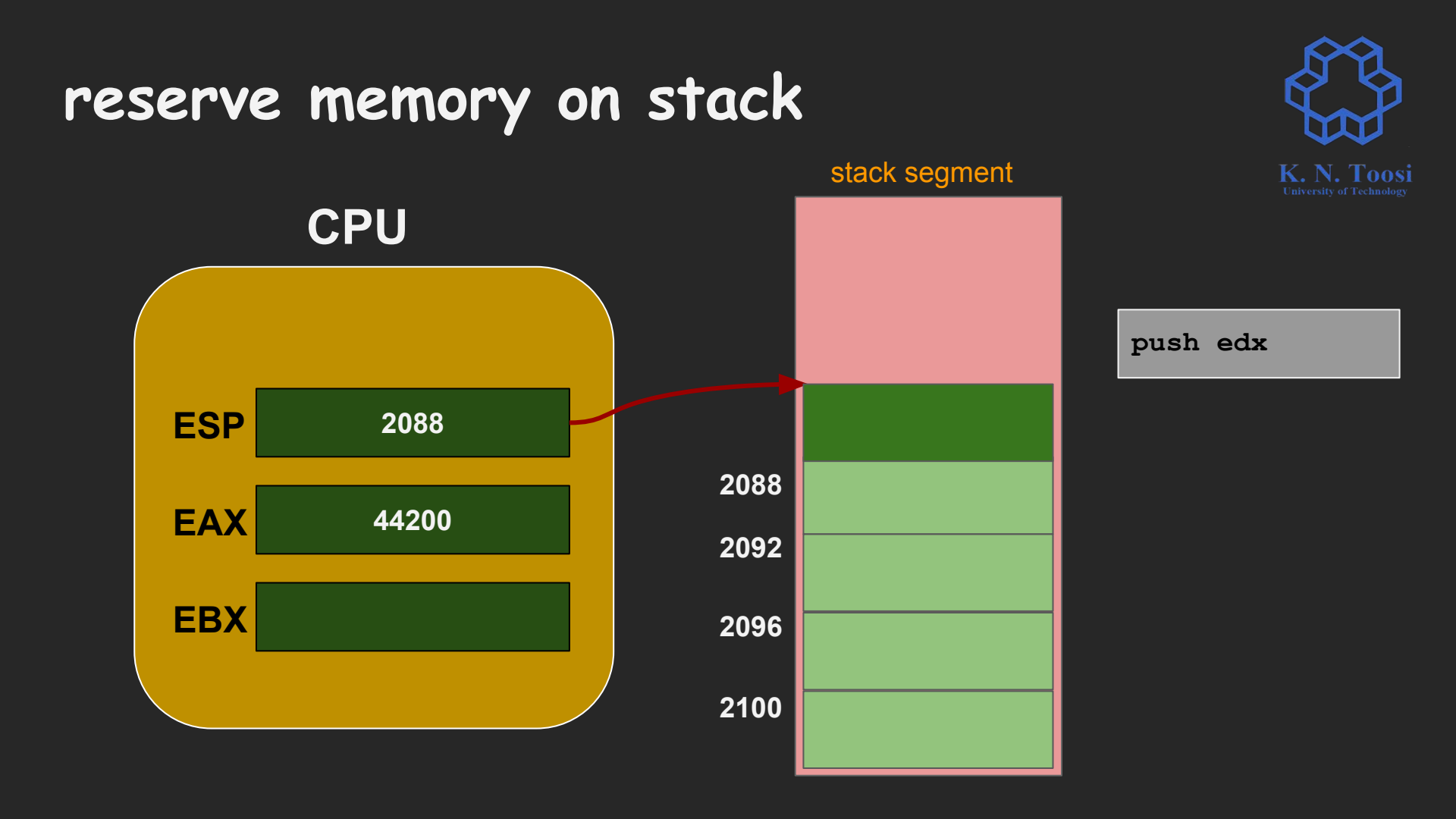


just pop 4 bytes (store nowhere)

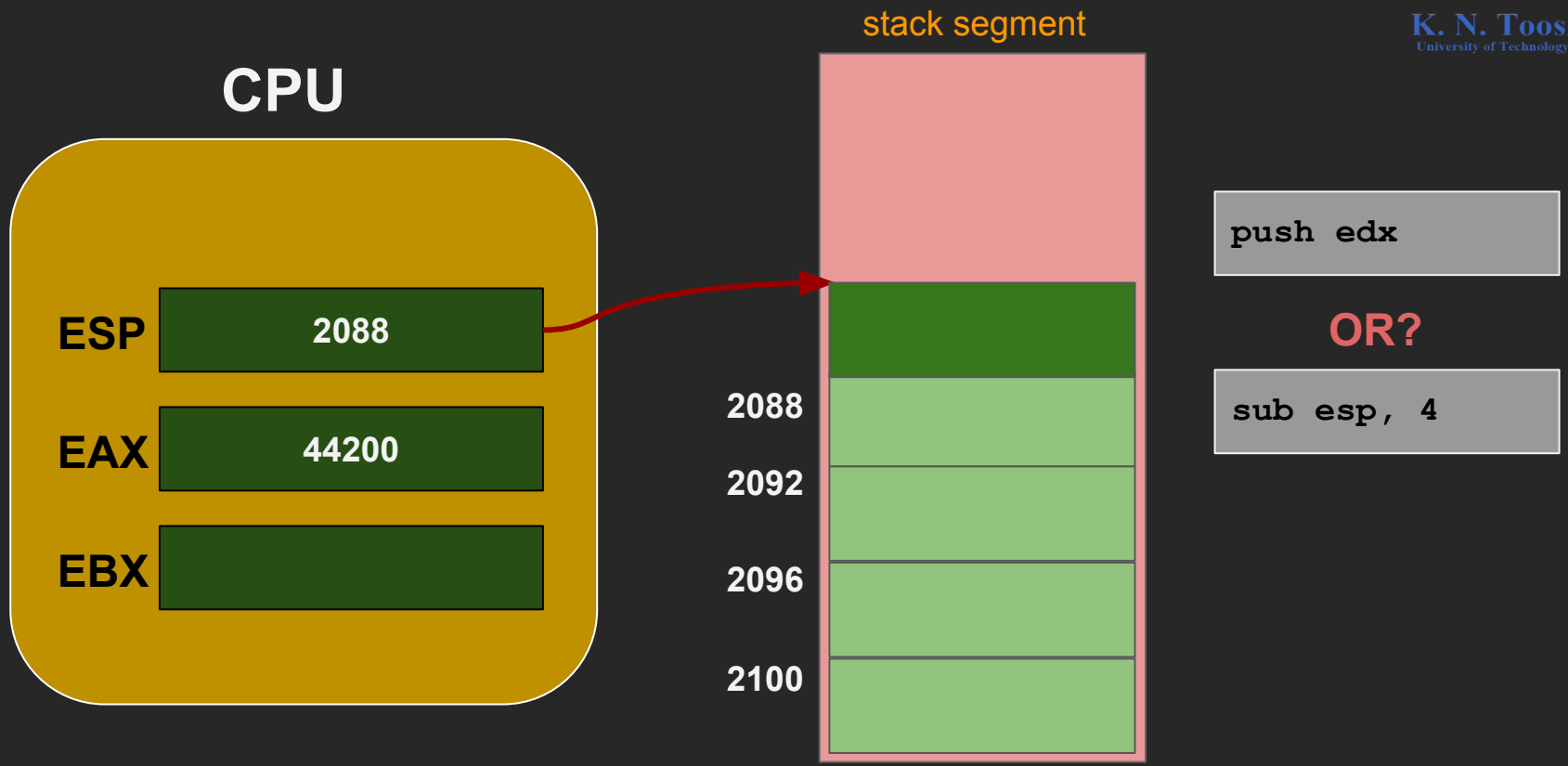
K. N. Toosi
University of Technology



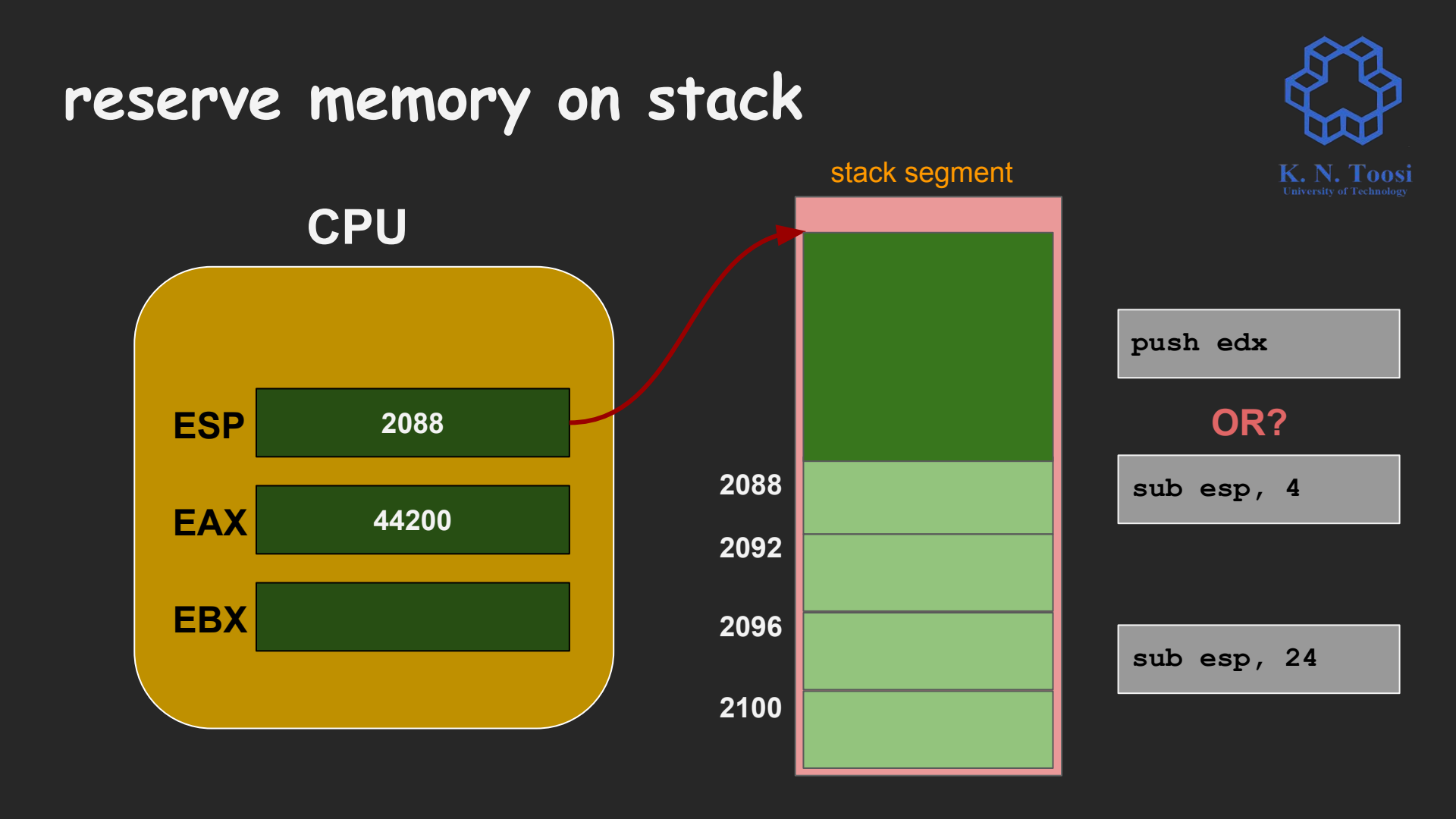
reserve memory on stack



reserve memory on stack



reserve memory on stack



Push and Pop


```
Push reg/mem/immed

Pop  reg/mem
```

Practice


```
push eax
push ebx

pop  eax
pop  ebx
```

pusha and popa

- 8086:
  - pusha: Push AX, CX, DX, BX, SP, BP, SI, DI
  - popa: Pop DI, SI, BP, BX, DX, CX, AX.
- 80386: netwide assembler (what we use)
  - pusha, pushad: Push EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI
  - popa, popad: Pop EDI, ESI, EBP, EBX, EDX, ECX, EAX.
- 80386: some other assemblers
  - pusha: Push AX, CX, DX, BX, SP, BP, SI, DI
  - pushad: Push EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI
  - popa: Pop DI, SI, BP, BX, DX, CX, AX.
  - popad: Pop EDI, ESI, EBP, EBX, EDX, ECX, EAX
- 64 bit
  - no pusha/popa in 64-bit mode

pushf and popf

- push and pop FLAGS/EFLAGS register
- some assemblers use (pushf/pushfd/pushfq, etc.)

Back to subroutines

simplefunc3.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮
        mov edx, l1
        jmp print_salam
l1:
        mov edx, l2
        jmp print_salam
l2:
        ⋮
print_salam:
        mov eax, msg
        call print_string
        jmp edx
```

Back to subroutines

simplefunc3.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮
        mov edx, l1
        jmp print_salam
l1:
        mov edx, l2
        jmp print_salam
l2:
        ⋮
print_salam:
        mov eax, msg
        call print_string
        jmp edx
```

simplefunc4.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮
        push l1
        jmp print_salam
l1:
        push l2
        jmp print_salam
l2:
        ⋮
print_salam:
        mov eax, msg
        call print_string
        ??
```

Back to subroutines

simplefunc3.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮
        mov edx, l1
        jmp print_salam
l1:
        mov edx, l2
        jmp print_salam
l2:
        ⋮
print_salam:
        mov eax, msg
        call print_string
        jmp edx
```

simplefunc4.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮
        push l1
        jmp print_salam
l1:
        push l2
        jmp print_salam
l2:
        ⋮
print_salam:
        mov eax, msg
        call print_string
        pop edx
        jmp edx
```

the CALL instruction

simplefunc3.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮
        mov edx, l1
        jmp print_salam
l1:
        mov edx, l2
        jmp print_salam
l2:
        ⋮
print_salam:
        mov eax, msg
        call print_string
        jmp edx
```

simplefunc4.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮
        push l1
        jmp print_salam
l1:
        push l2
        jmp print_salam
l2:
        ⋮
print_salam:
        mov eax, msg
        call print_string
        pop edx
        jmp edx
```

simplefunc5.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮

        call print_salam
l1:

        call print_salam
l2:
        ⋮
print_salam:
        mov eax, msg
        call print_string
        pop edx
        jmp edx
```

the CALL instruction

simplefunc3.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮
        mov edx, l1
        jmp print_salam
l1:
        mov edx, l2
        jmp print_salam
l2:
        ⋮
print_salam:
        mov eax, msg
        call print_string
        jmp edx
```

simplefunc4.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮
        push l1
        jmp print_salam
l1:
        push l2
        jmp print_salam
l2:
        ⋮
print_salam:
        mov eax, msg
        call print_string
        pop edx
        jmp edx
```

simplefunc5.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮

        call print_salam


        call print_salam

        ⋮
print_salam:
        mov eax, msg
        call print_string
        pop edx
        jmp edx
```

the CALL instruction

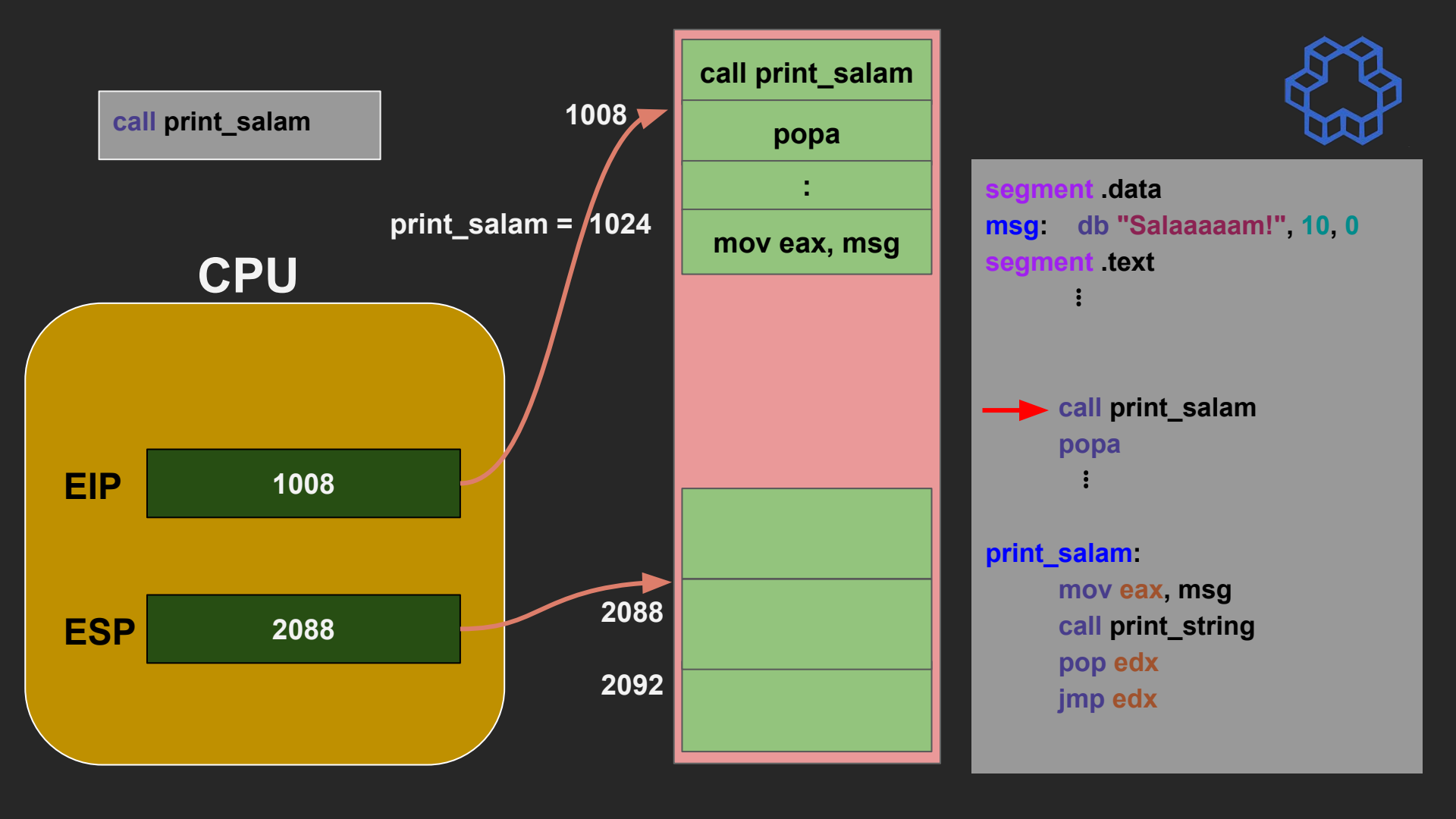
CALL is merely a form of jump!

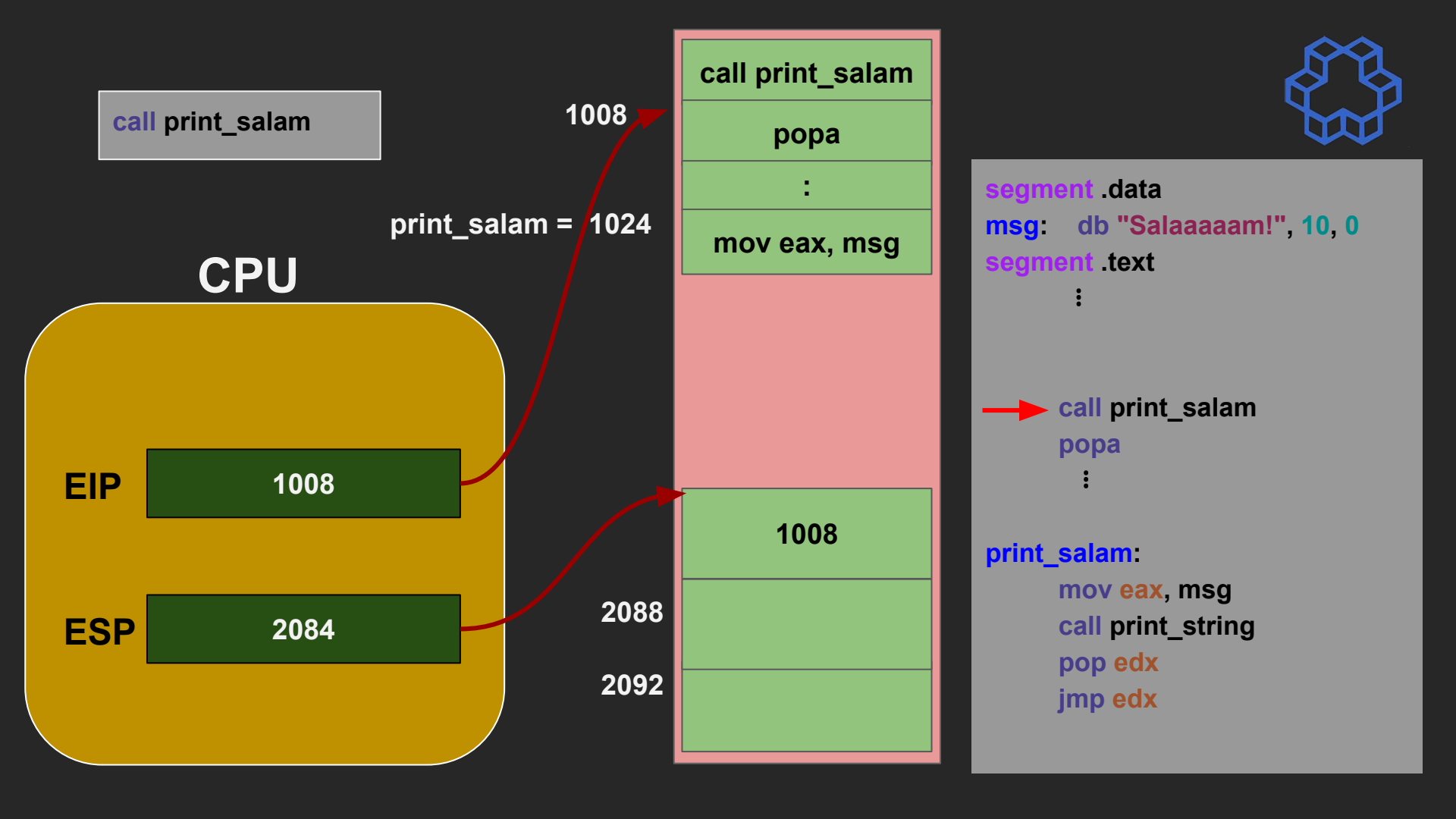
the CALL instruction

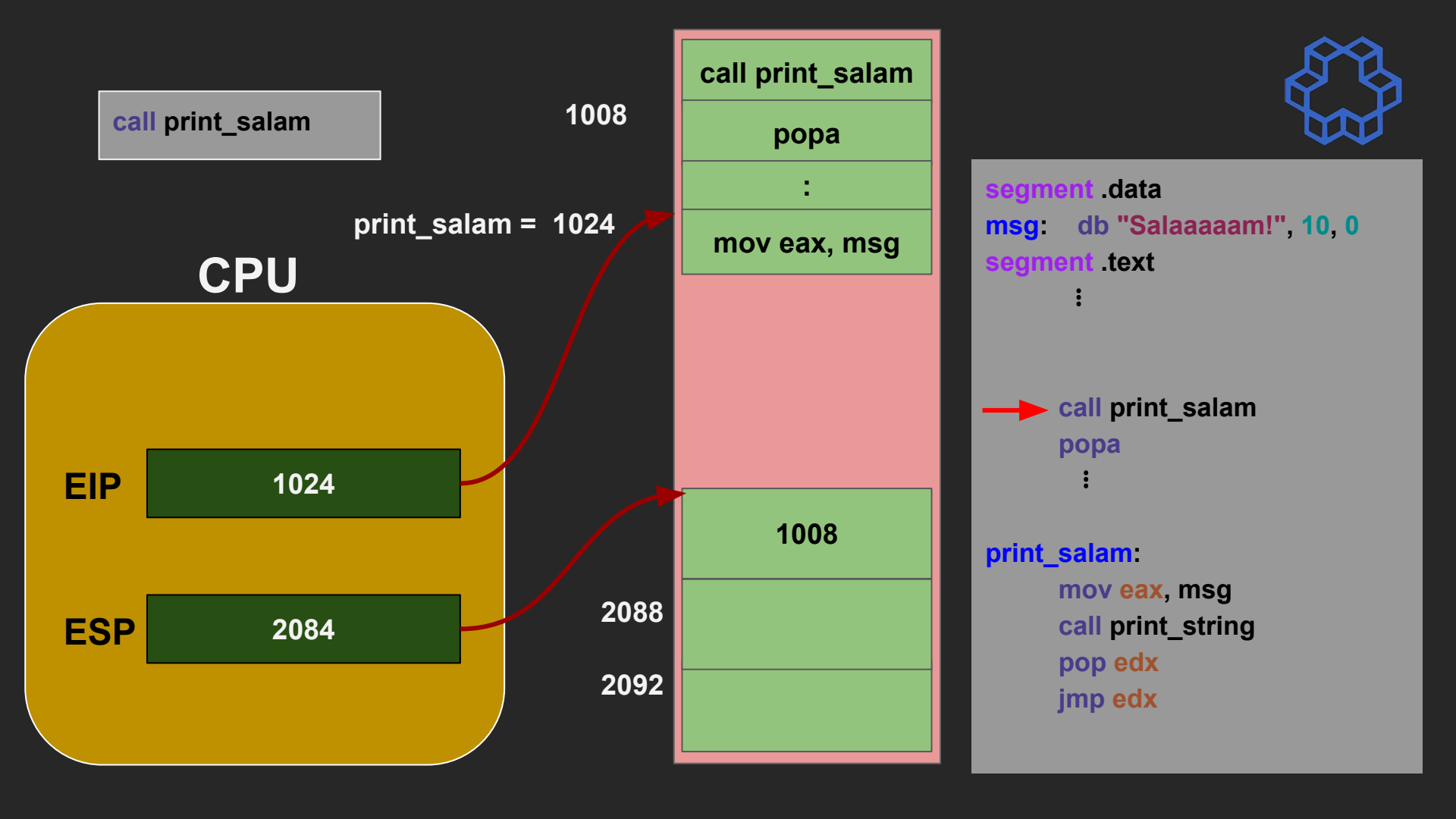
CALL is merely a form of jump!

```
call label1
```

- Push return address (EIP) on stack
- jump to `label1`


call print_salam
1008
print_salam = 1024
CPU
EIP
1008
ESP
2088
call print_salam
popa
:
mov eax, msg
2088
2092
segment .data
msg: db "Salaaaaam!", 10, 0
segment .text
call print_salam
popa
print_salam:
mov eax, msg
call print_string
pop edx
jmp edx


call print_salam
CPU
EIP
1008
ESP
2084
1008
print_salam = 1024
call print_salam
popa
:
mov eax, msg
1008
2088
2092
segment .data
msg: db "Salaaaaam!", 10, 0
segment .text
call print_salam
popa
print_salam:
mov eax, msg
call print_string
pop edx
jmp edx


call print_salam
1008
print_salam = 1024
CPU
EIP
1024
ESP
2084
call print_salam
popa
:
mov eax, msg
1008
2088
2092
segment .data
msg: db "Salaaaaam!", 10, 0
segment .text
call print_salam
popa
print_salam:
mov eax, msg
call print_string
pop edx
jmp edx

returning from a subroutine

simplefunc5.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮

        call print_salam


        call print_salam

        ⋮
print_salam:
        mov eax, msg
        call print_string
        pop edx
        jmp edx
```

the RET instruction

simplefunc5.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮

        call print_salam


        call print_salam

        ⋮
print_salam:
        mov eax, msg
        call print_string
        pop edx
        jmp edx
```

simplefunc6.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮

        call print_salam


        call print_salam

        ⋮
print_salam:
        mov eax, msg
        call print_string
        ret
```

the RET instruction

simplefunc5.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮

        call print_salam


        call print_salam

        ⋮
print_salam:
        mov eax, msg
        call print_string
        pop edx
        jmp edx
```

simplefunc6.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮

        call print_salam


        call print_salam

        ⋮
print_salam:
        mov eax, msg
        call print_string
        ret
```

ret=?

the RET instruction

simplefunc5.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮

        call print_salam


        call print_salam

        ⋮
print_salam:
        mov eax, msg
        call print_string
        pop edx
        jmp edx
```

simplefunc6.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮

        call print_salam


        call print_salam

        ⋮
print_salam:
        mov eax, msg
        call print_string
        ret
```

ret (pop EIP)

the RET instruction

simplefunc5.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮

        call print_salam


        call print_salam

        ⋮
print_salam:
        mov eax, msg
        call print_string
        pop edx
        jmp edx
```

simplefunc6.asm

```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮

        call print_salam


        call print_salam

        ⋮
print_salam:
        mov eax, msg
        call print_string
        ret
```

the RET instruction

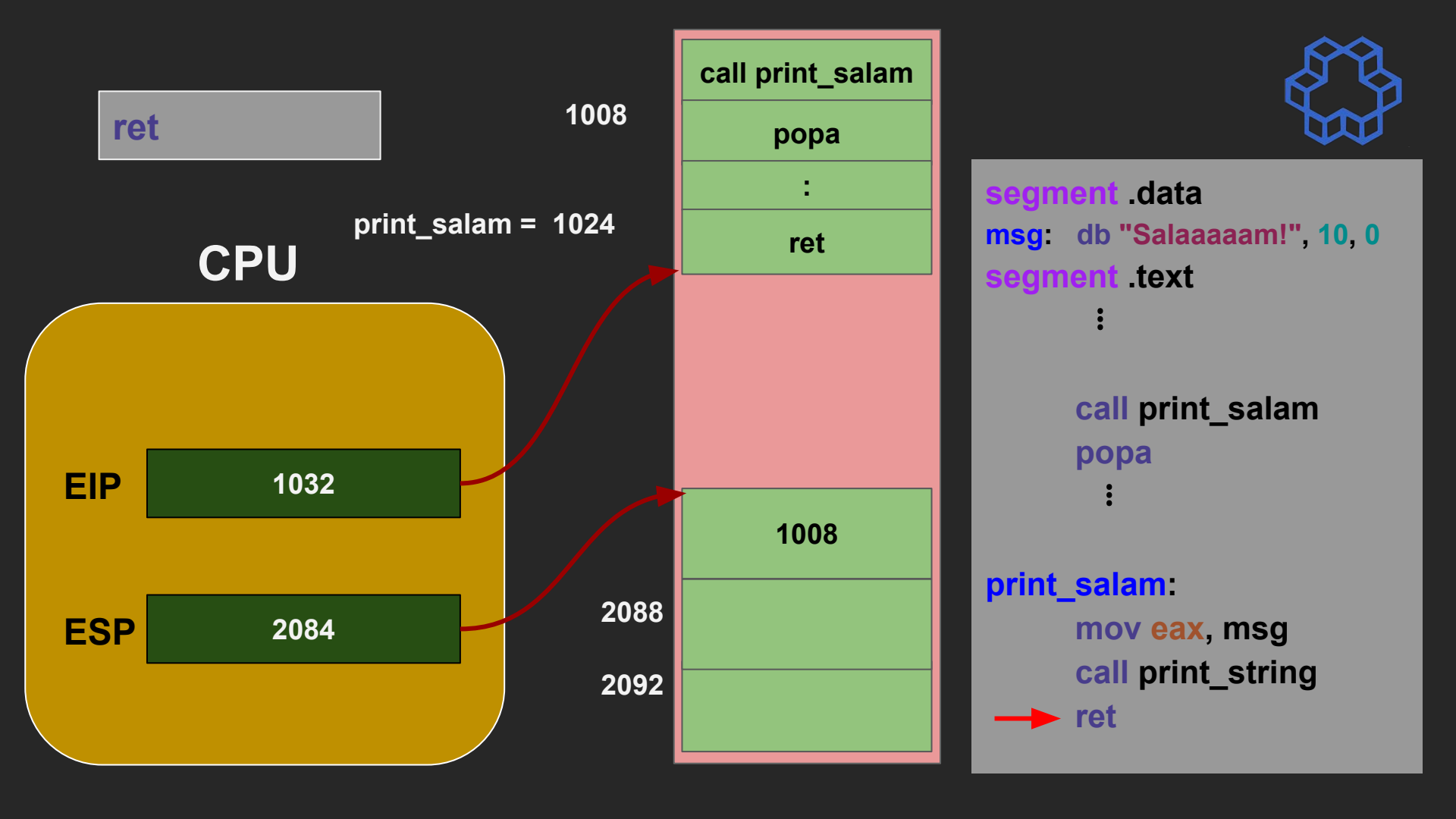
RET is merely a form of jump!

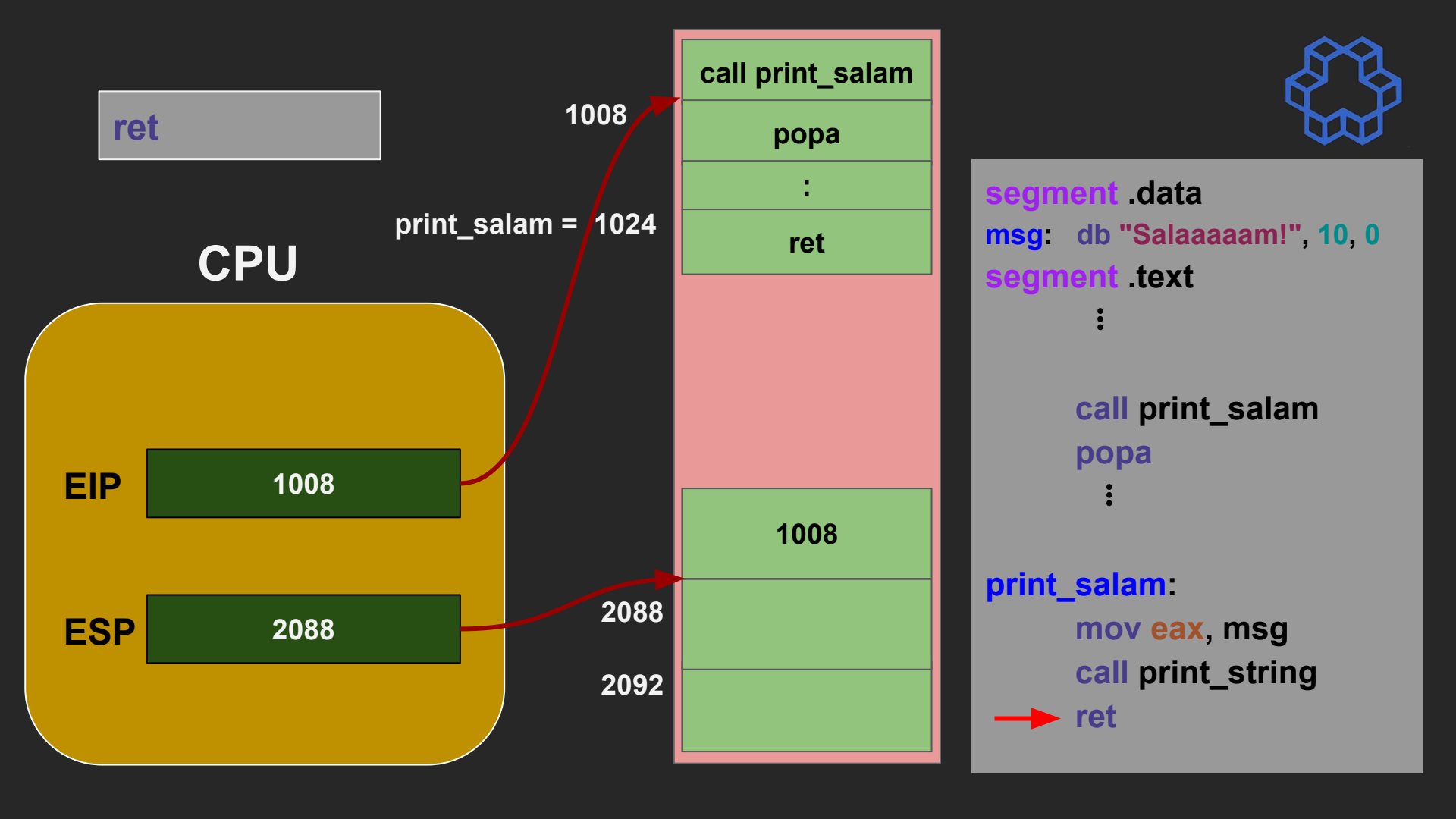
the RET instruction

RET is merely a form of jump!

```
ret
```

- jump to the address stored on top of stack
- pop stack


ret
1008
call print_salam
popa
:
ret
print_salam = 1024
CPU
EIP
1032
ESP
2084
1008
2088
2092
segment .data
msg: db "Salaaaaam!", 10, 0
segment .text
call print_salam
popa
print_salam:
mov eax, msg
call print_string
ret


ret
CPU
EIP
1008
ESP
2088
1008
print_salam = 1024
call print_salam
popa
:
ret
1008
2088
2092
segment .data
msg: db "Salaaaaam!", 10, 0
segment .text
call print_salam
popa
print_salam:
mov eax, msg
call print_string
ret

What else?

- parameters (arguments)
- local variables
- return values