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
```

K. N. Toosi
University of Technology

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?

K. N. Toosi
University of Technology

Indirect addressing

K. N. Toosi
University of Technology

```
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?

K. N. Toosi
University of Technology

```c
void print_salam(void);

int main() {

  print_salam();

}

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

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

segment .text
        ⋮



        ⋮


print_salam:
        mov eax, msg
        call print_string
```

How to implement subprograms?

K. N. Toosi
University of Technology

```c
void print_salam(void);

int main() {

  print_salam();

}

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

```asm
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");
}
```

```nasm
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?

K. N. Toosi
University of Technology

```
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
```

$l1: \longrightarrow$ return address

How to implement subprograms?

```c
void print_salam(void);

int main() {

  print_salam();

}

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

simplefunc1.asm

```nasm
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?

K. N. Toosi
University of Technology

```c
void print_salam(void);

int main() {

  print_salam();

}

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

What's wrong?

simplefunc1.asm

```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?

K. N. Toosi
University of Technology

```c
void print_salam(void);

int main() {

  print_salam();

}

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

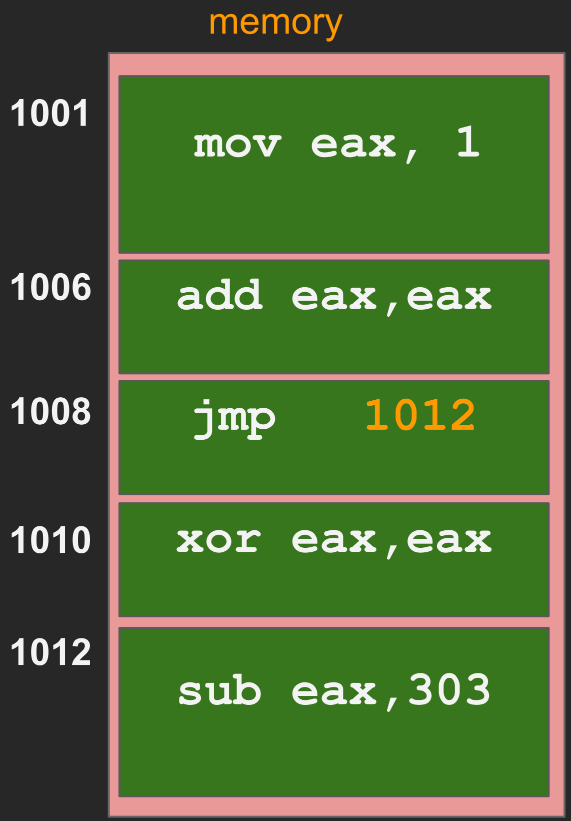
simplefunc2.asm

```nasm
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

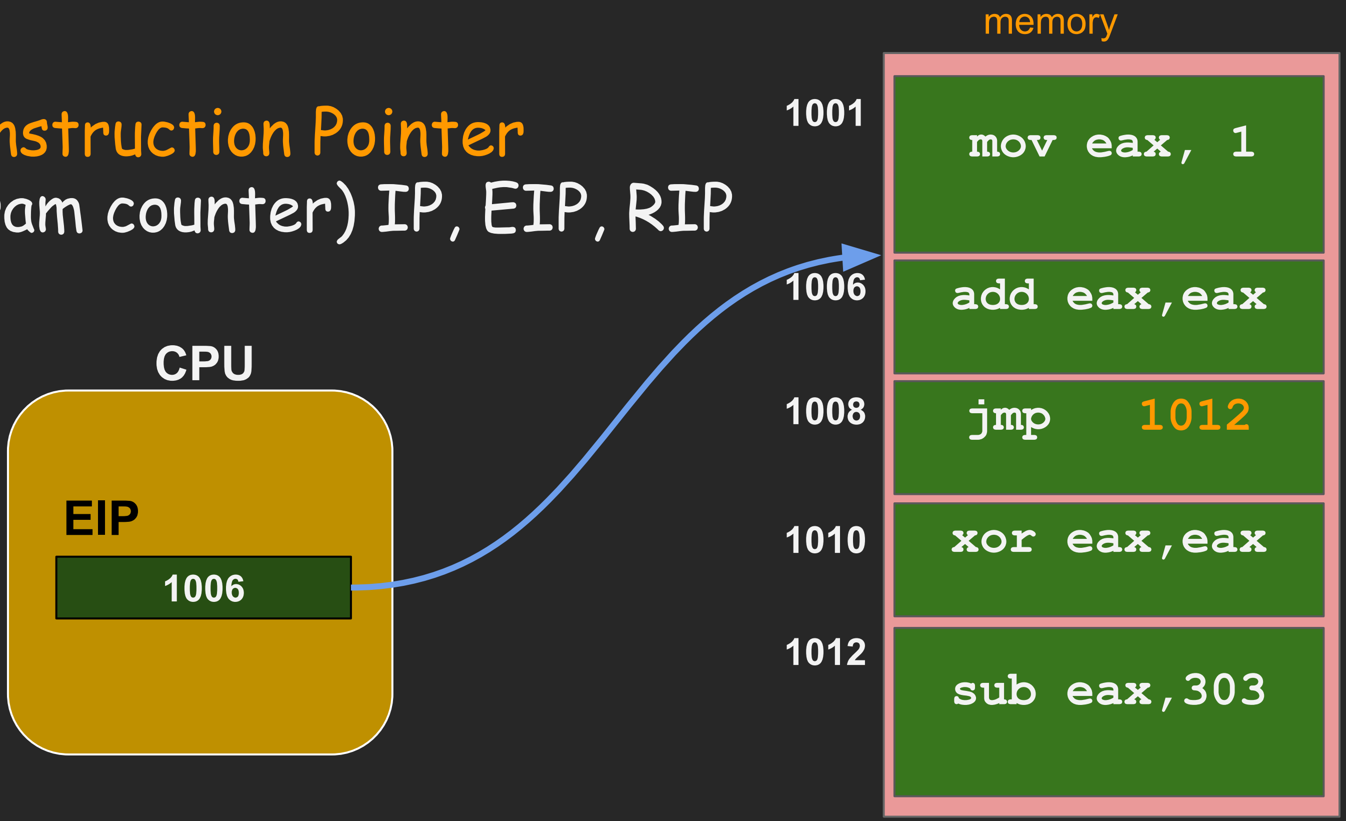
Looking closer at the jmp command

```
    mov eax, 1

    add eax, eax

    jmp label1

    xor eax, eax

label1:

    sub eax, 303
```



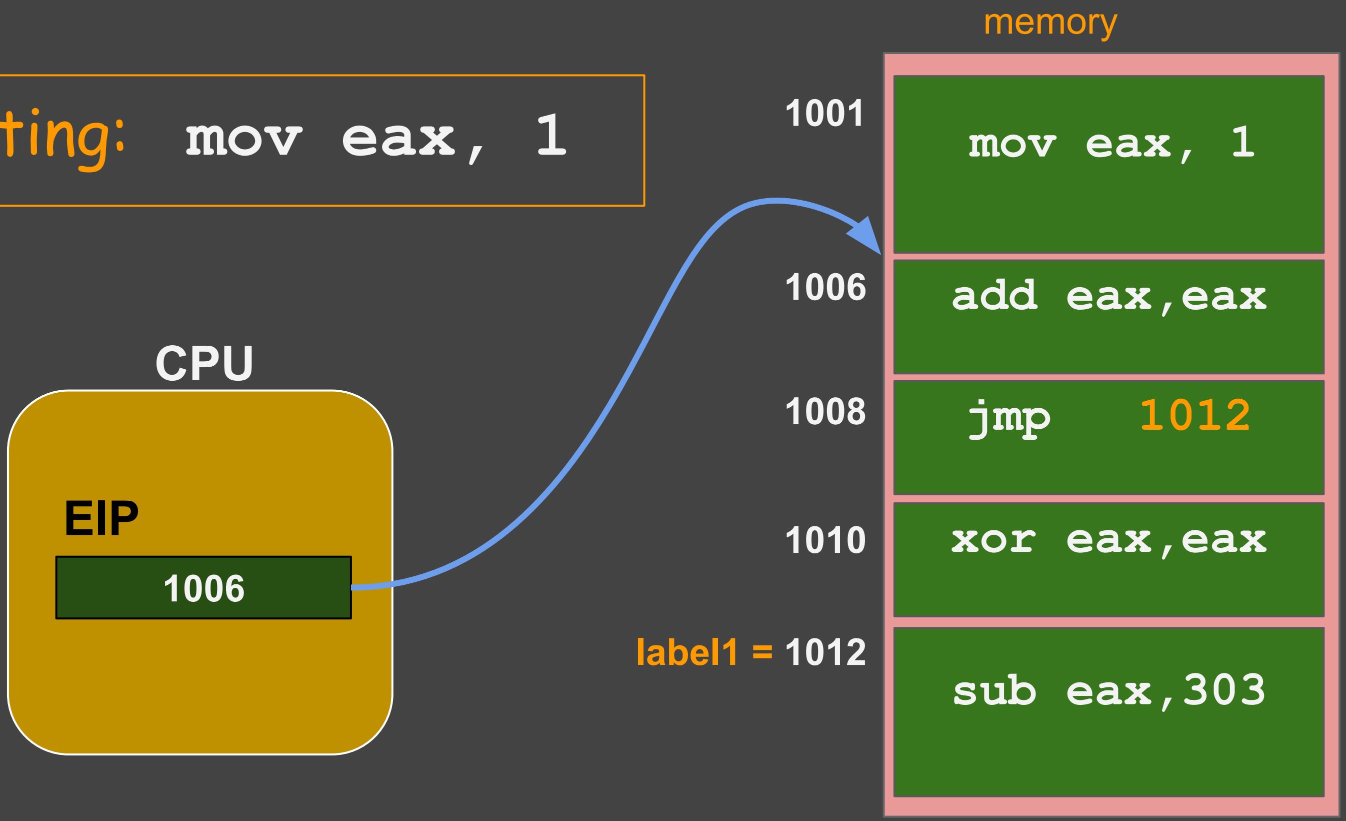
Remember: Jump and The Instruction Pointer

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



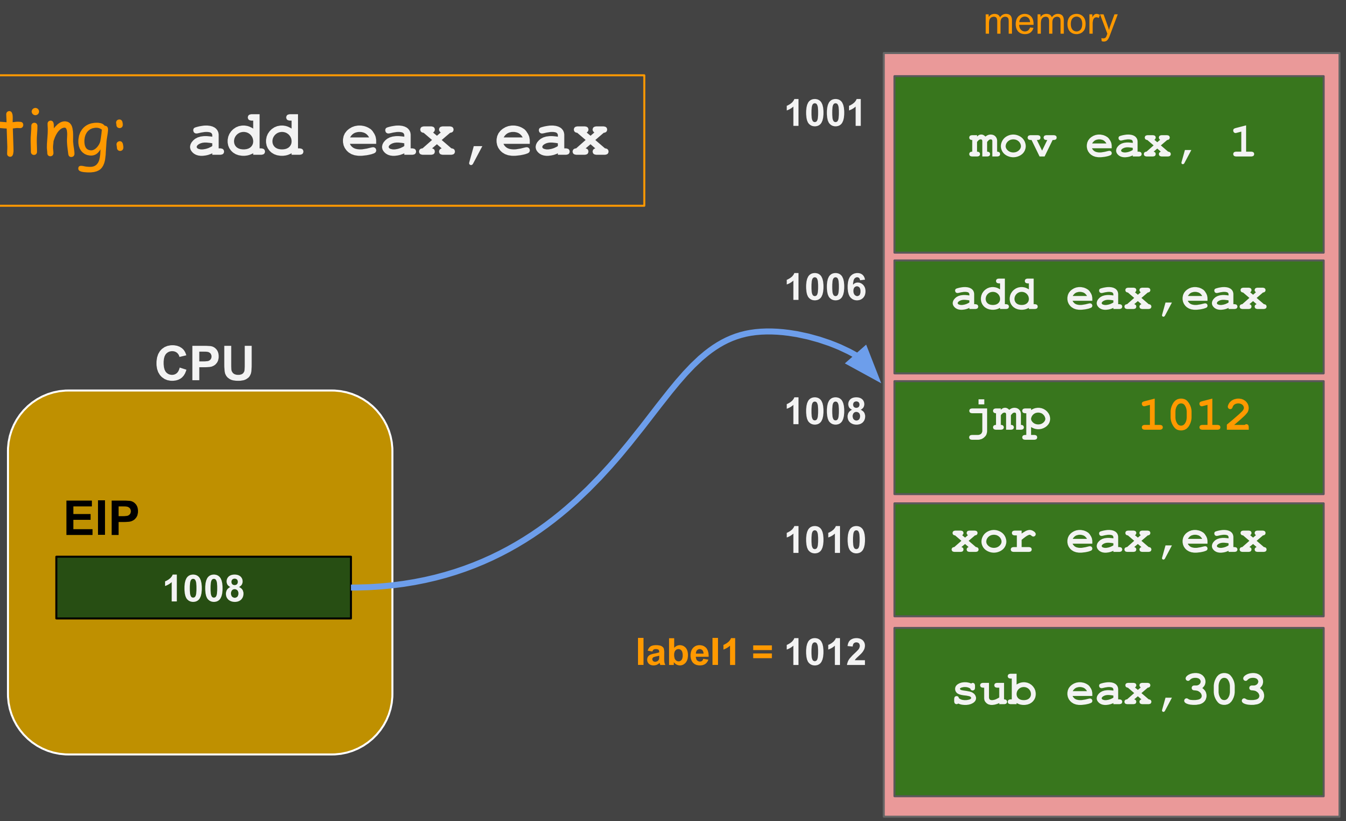
Remember: Jump and The Instruction Pointer

Executing: `mov eax, 1`



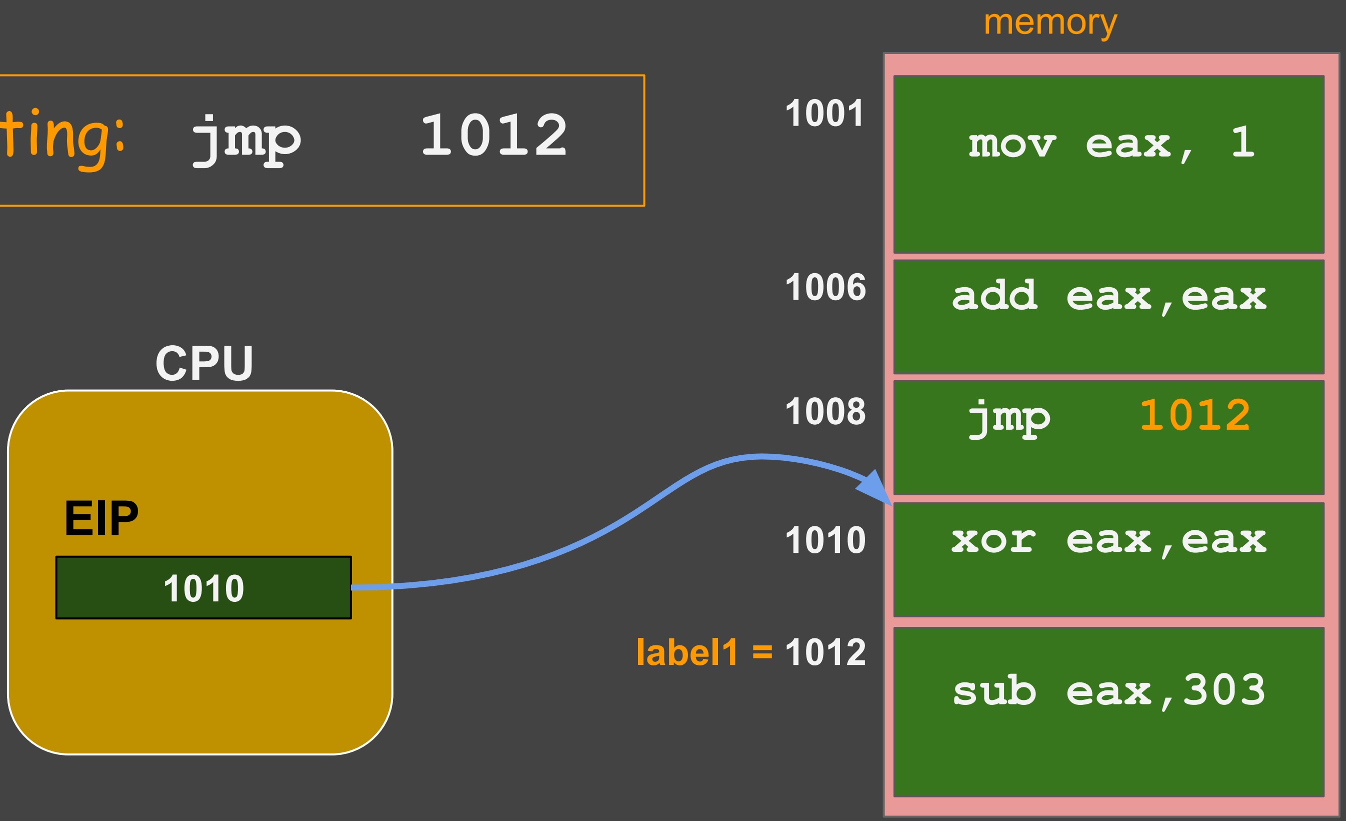
Remember: Jump and The Instruction Pointer

Executing: `add eax,eax`



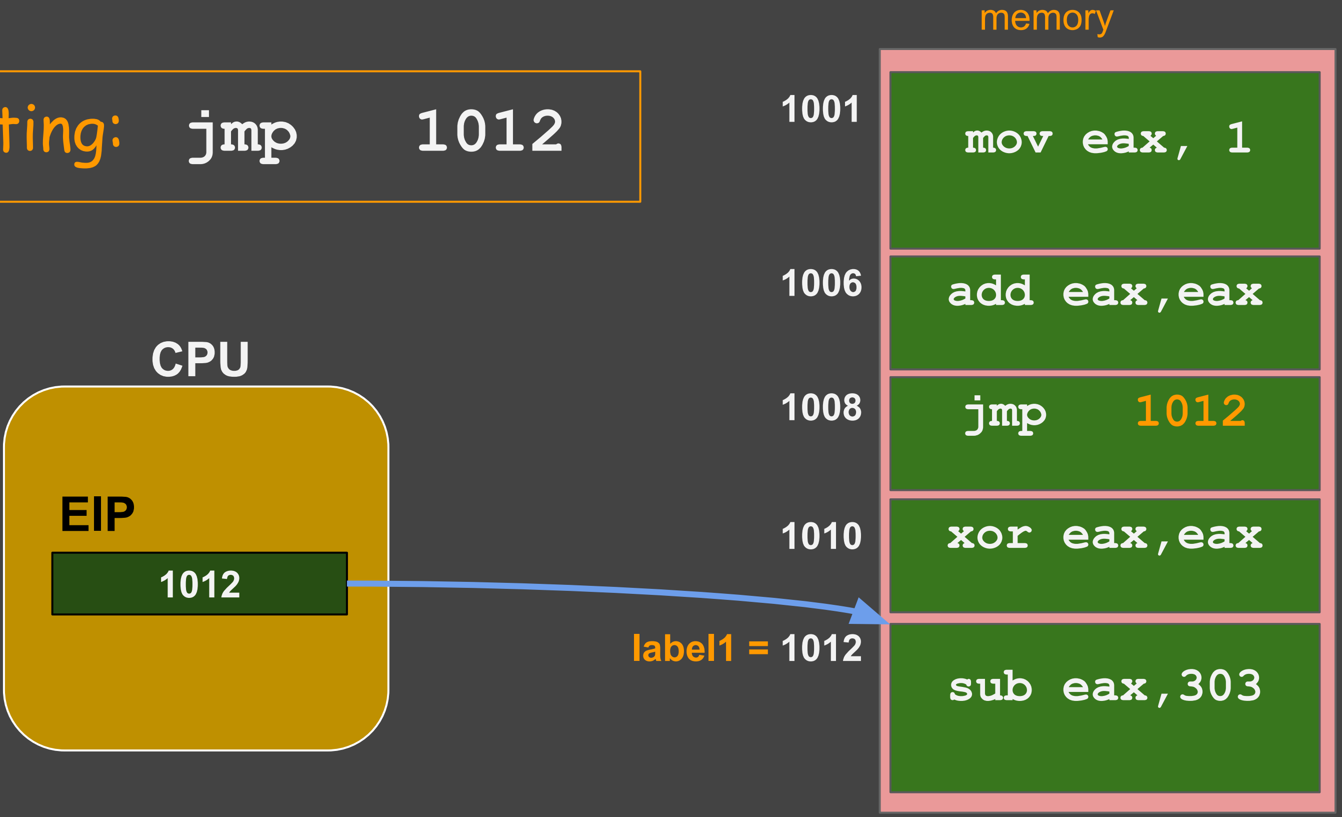
Remember: Jump and The Instruction Pointer

Executing: jmp 1012



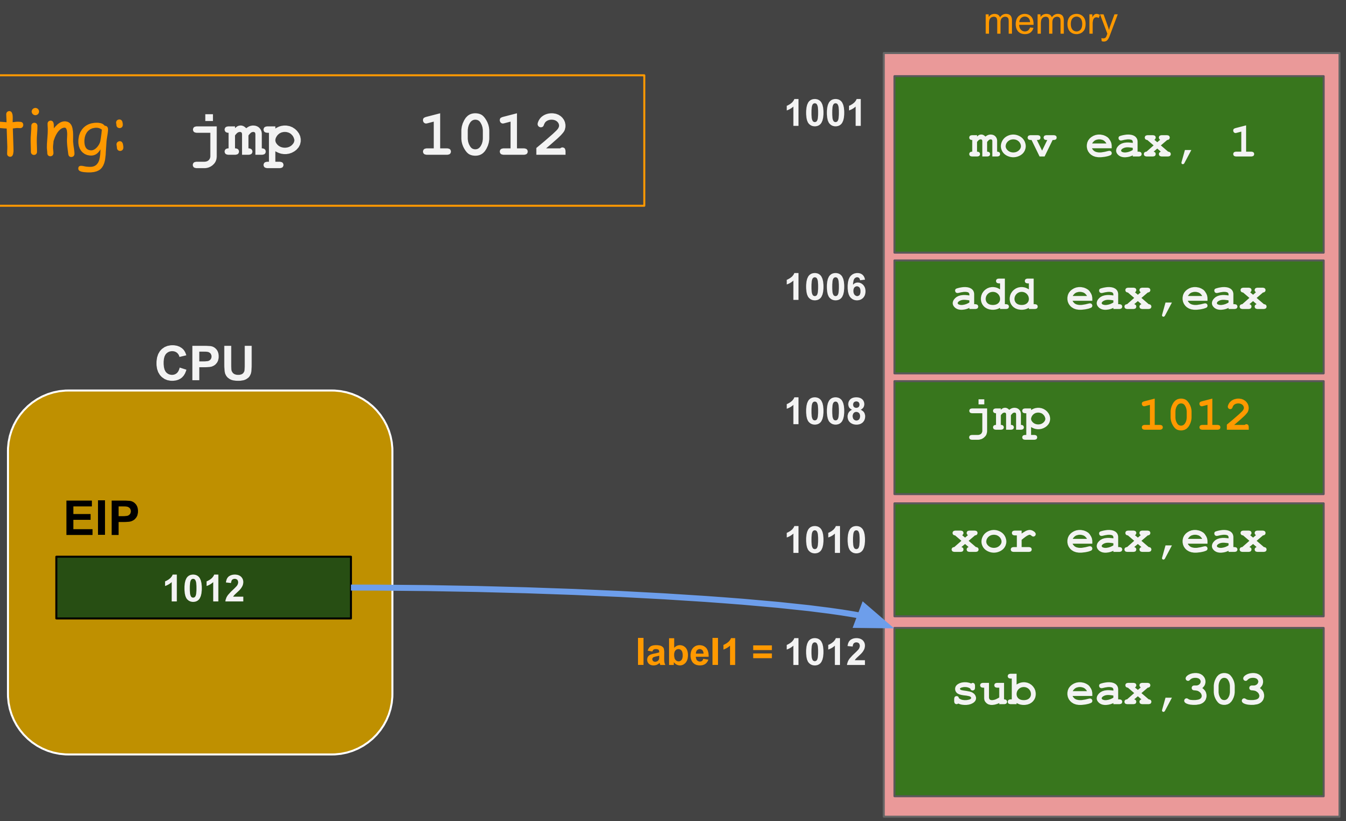
Remember: Jump and The Instruction Pointer

Executing: `jmp    1012`



Remember: Jump and The Instruction Pointer

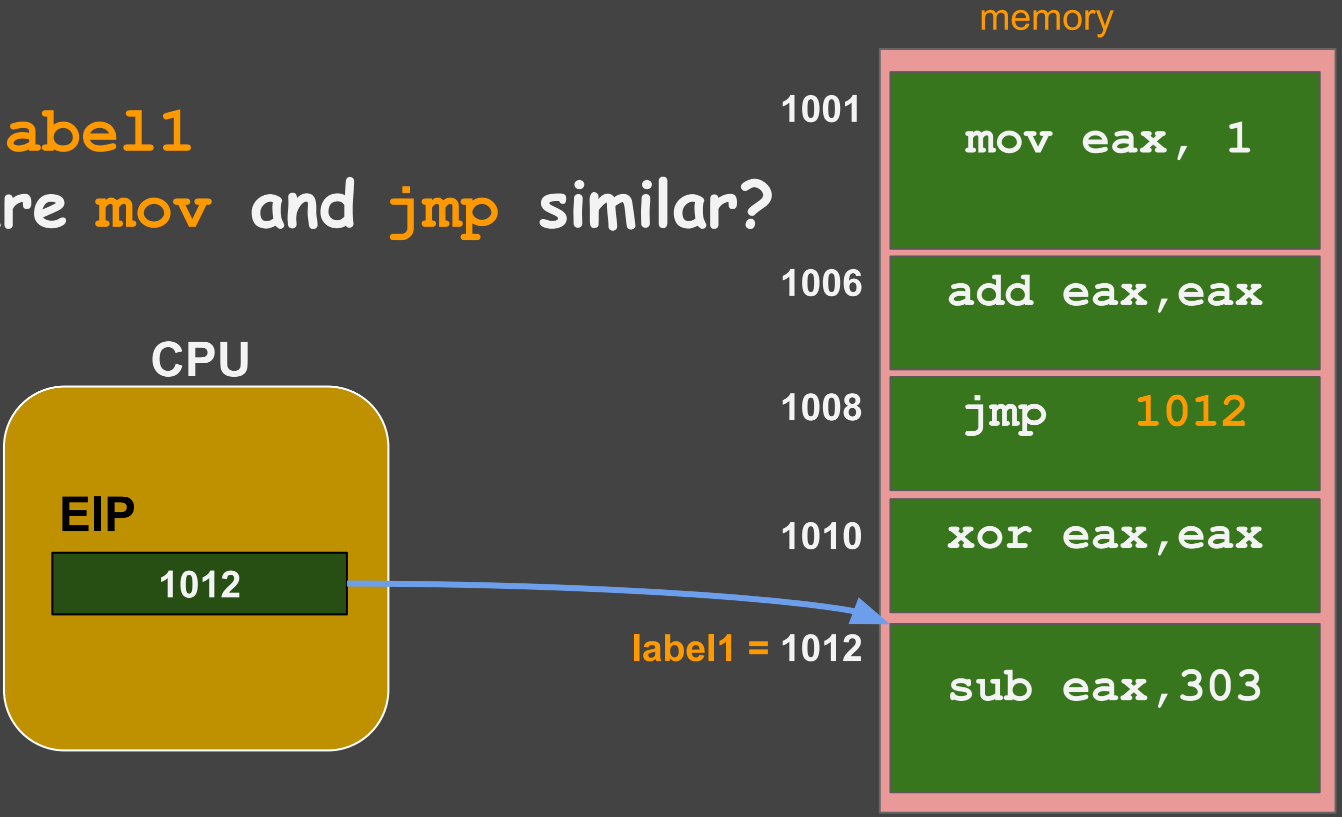
Executing: `jmp 1012`



Remember: Jump and The Instruction Pointer

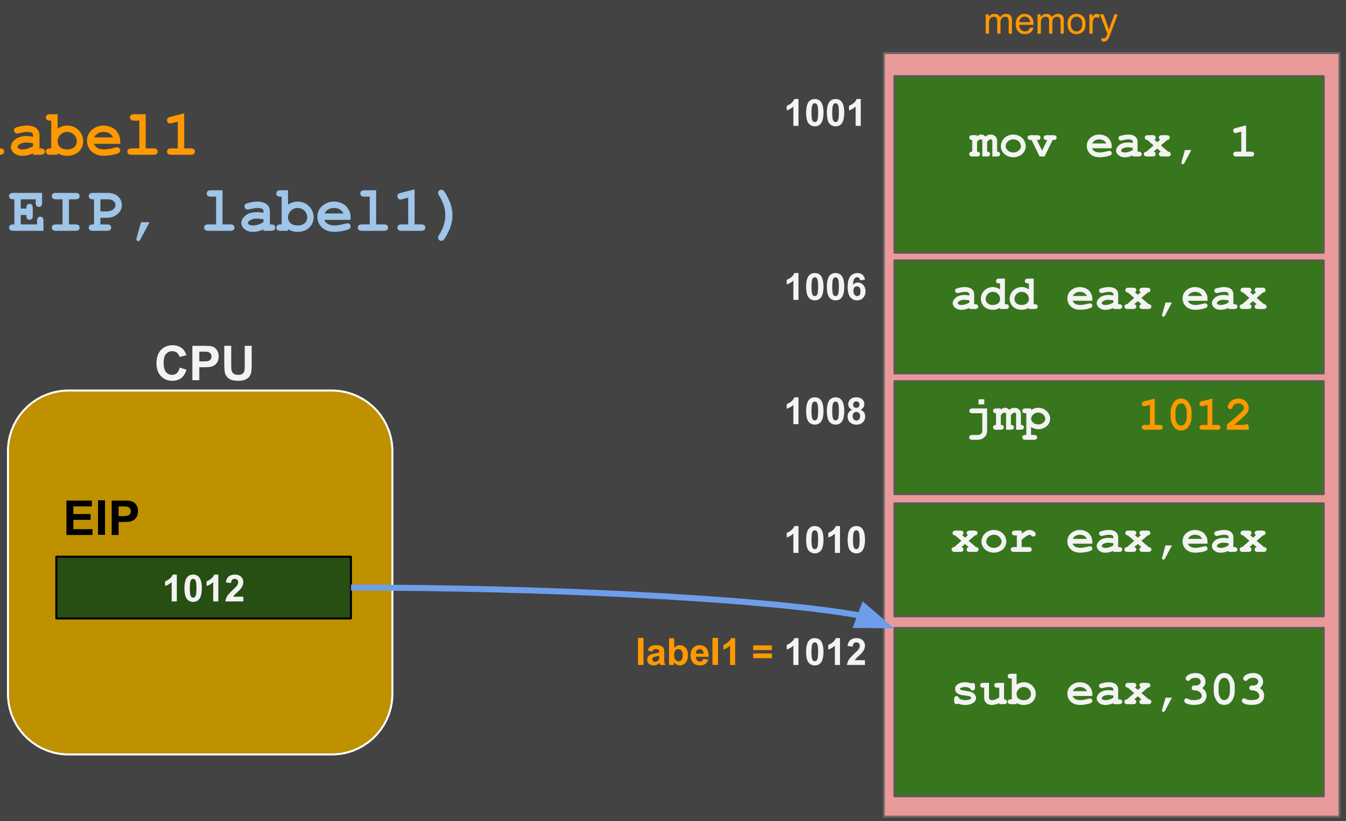
jmp label1

How are mov and jmp similar?



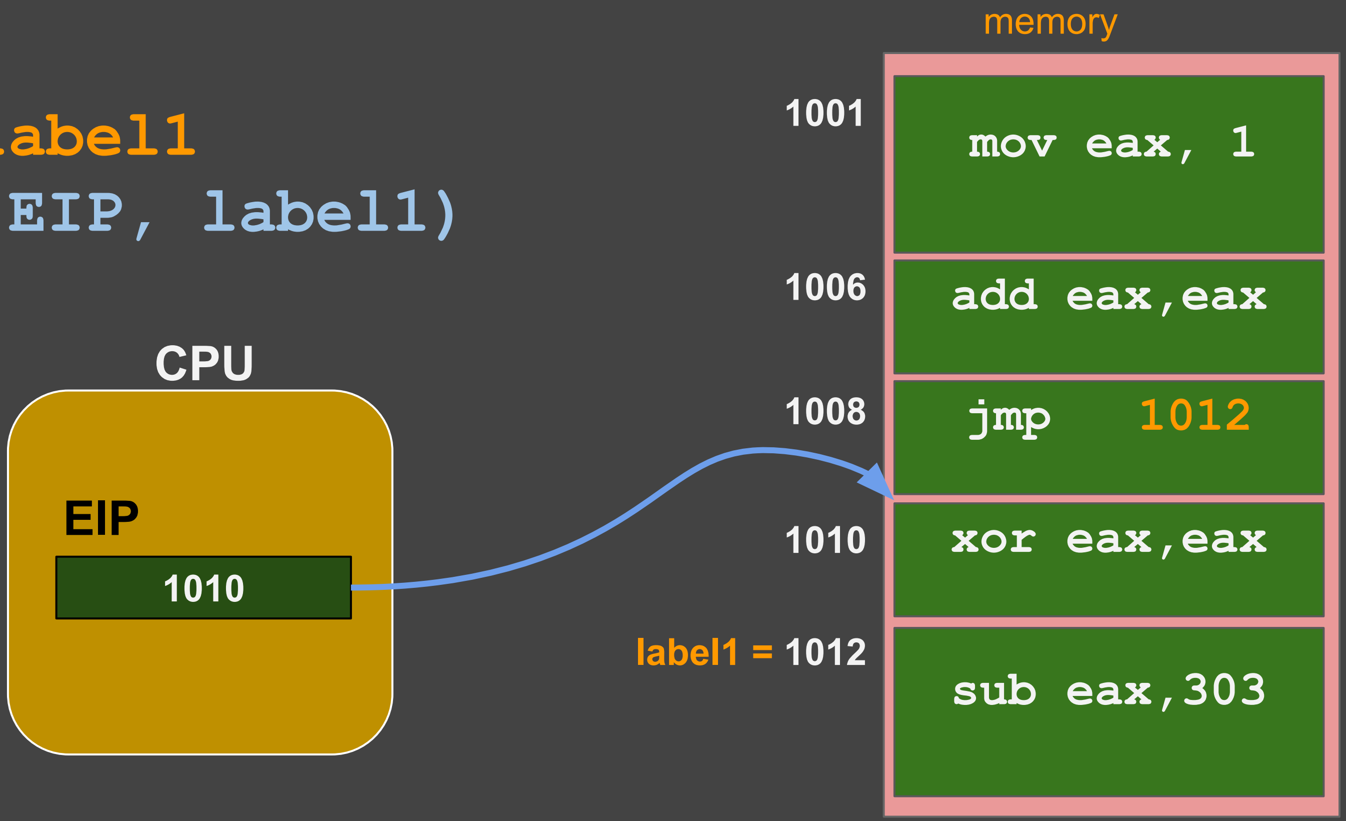
Remember: Jump and The Instruction Pointer

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



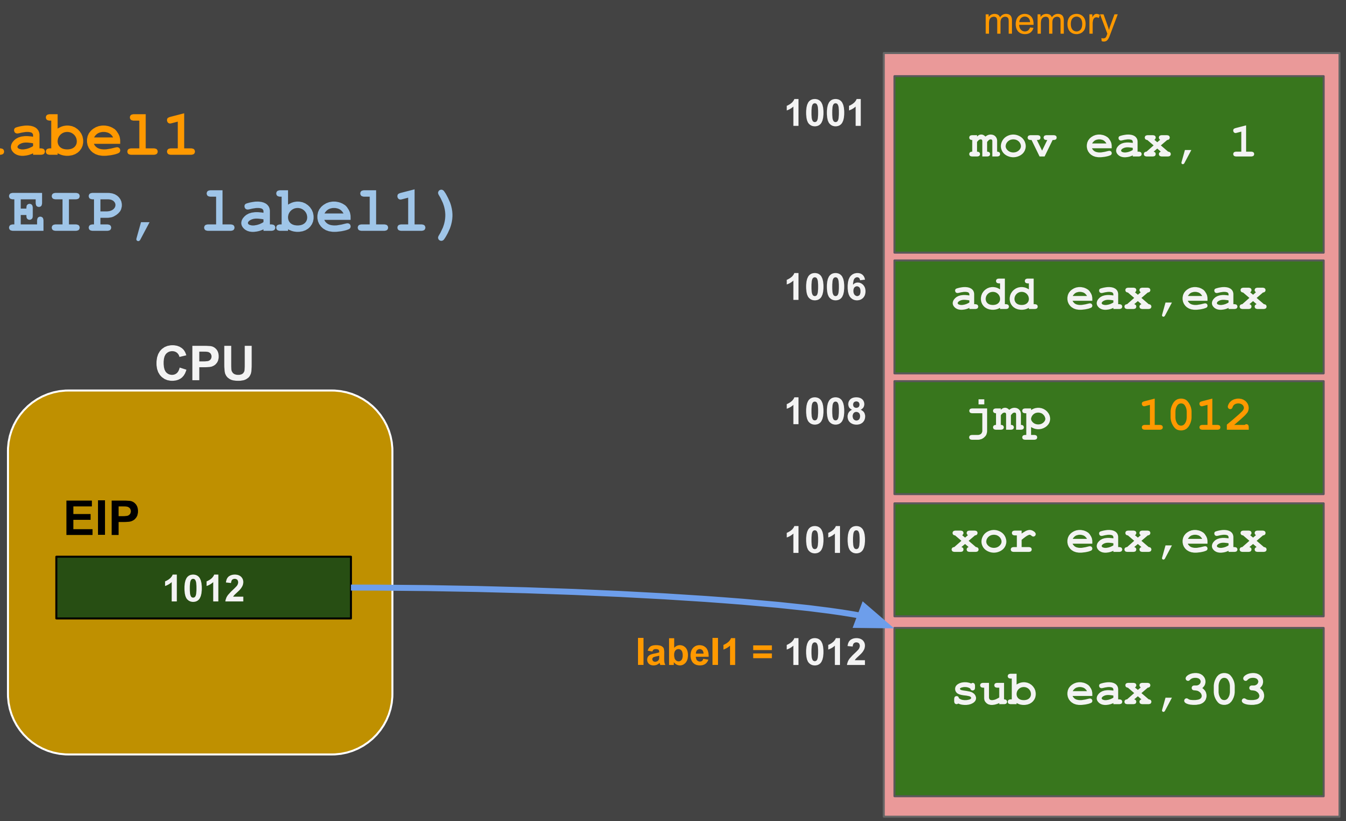
Remember: Jump and The Instruction Pointer

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



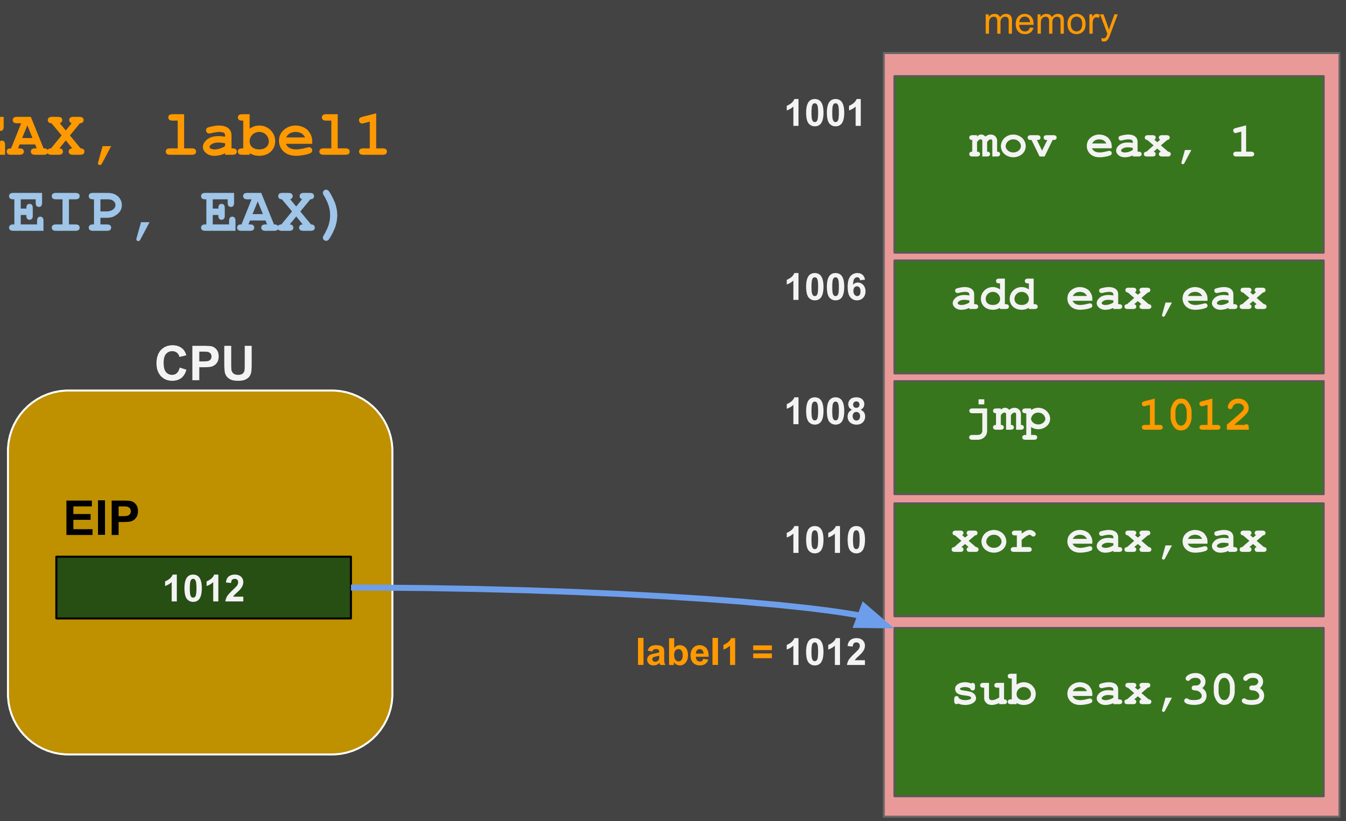
Remember: Jump and The Instruction Pointer

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



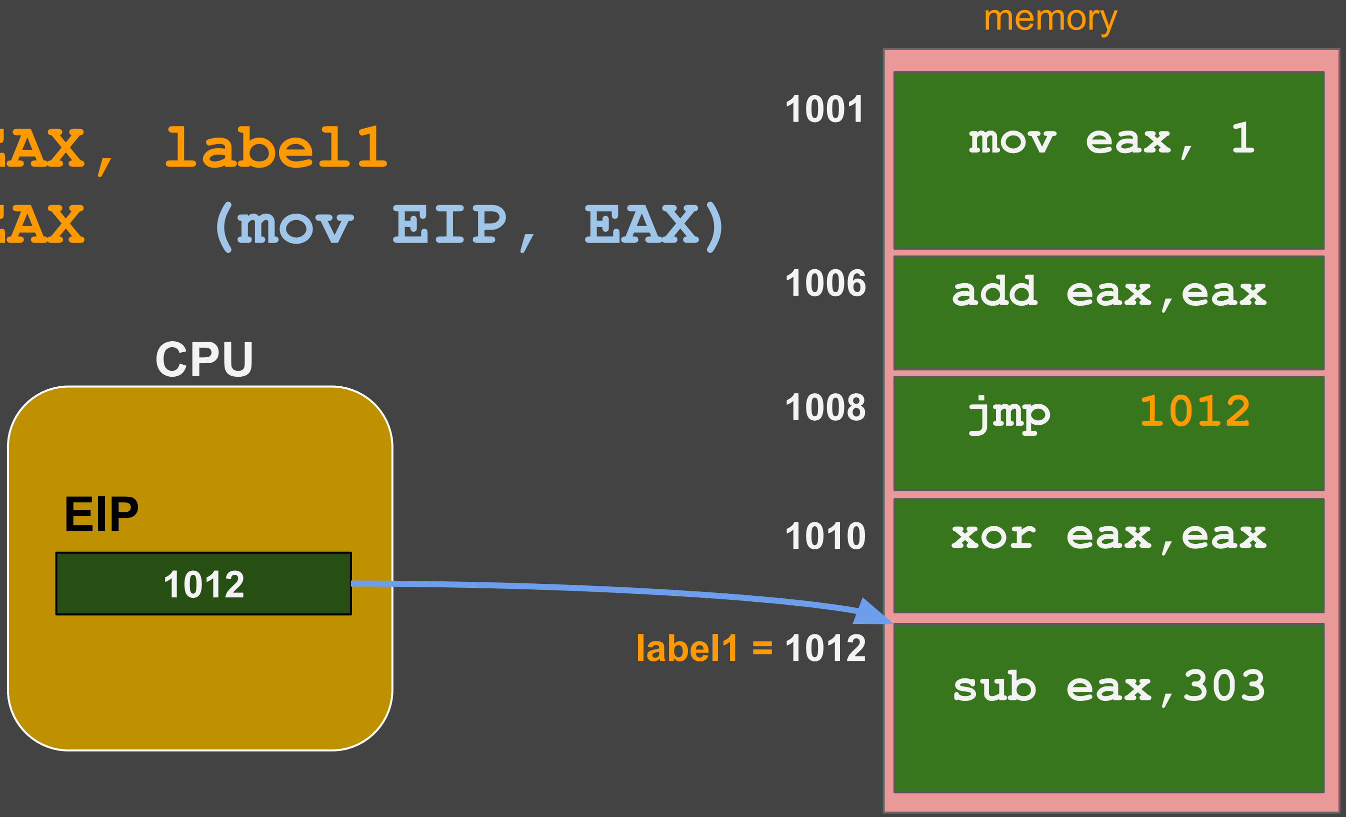
Remember: Jump and The Instruction Pointer

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



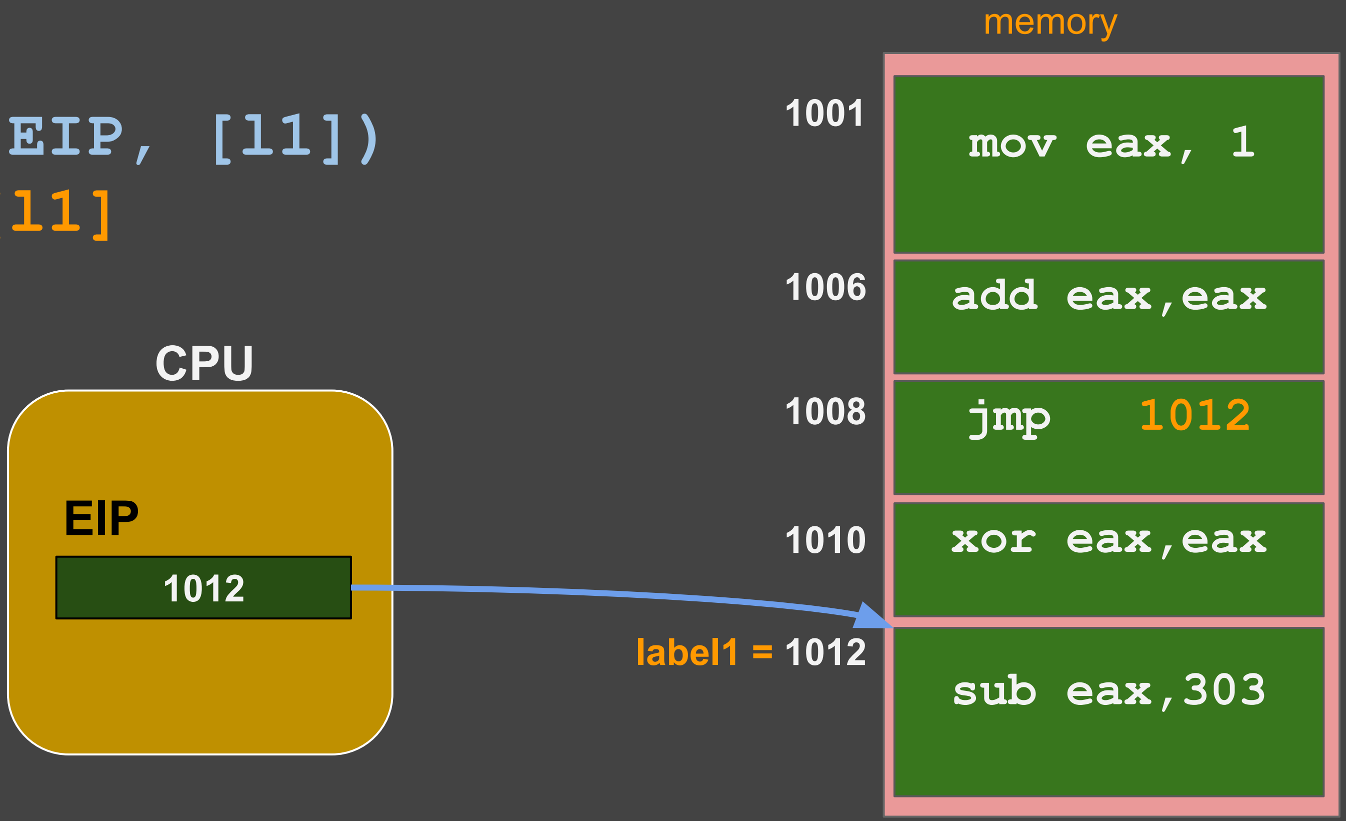
Remember: Jump and The Instruction Pointer

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



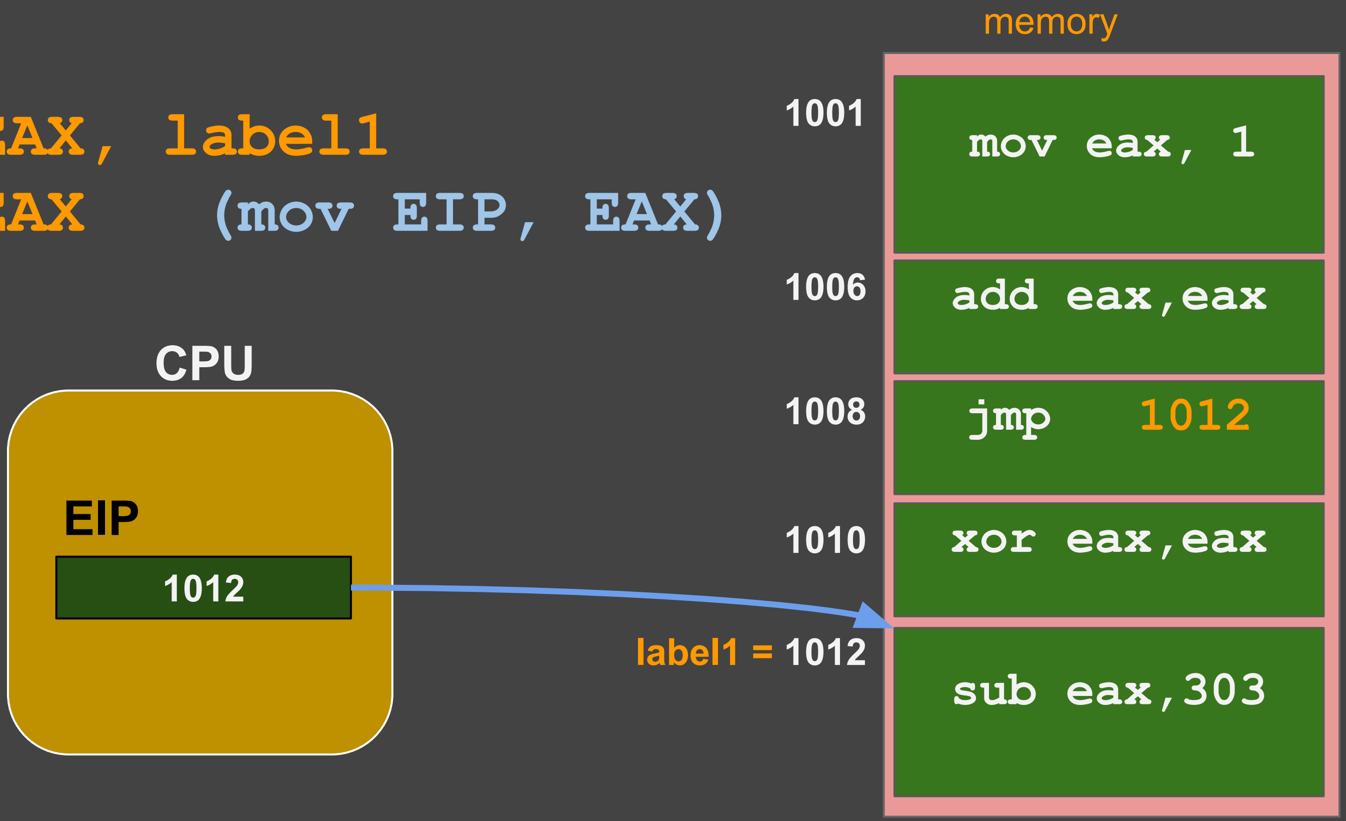
Remember: Jump and The Instruction Pointer

```
(mov EIP, [l1])
jmp [l1]
```



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?

K. N. Toosi
University of Technology

```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 ?
```

return address

How to implement subprograms?

```c
void print_salam(void);

int main() {

  print_salam();

}

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

simplefunc3.asm

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

return address

How to implement subprograms?

K. N. Toosi
University of Technology

```c
void print_salam(void);

int main() {

  print_salam();

}

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

Limitations?

simplefunc3.asm

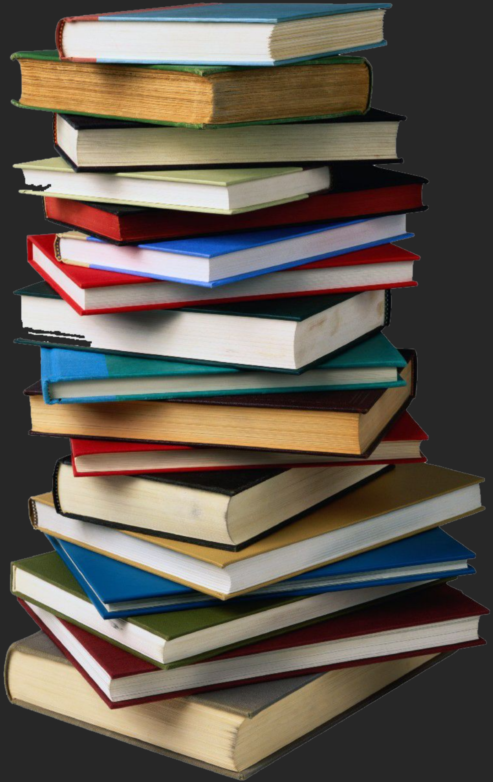
```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
         ⋮
        mov edx, l1
        jmp print_salam
l1:
        mov edx, l2
        jmp print_salam
l2:
         ⋮
print_salam:
        mov eax, msg
        call print_string
        jmp edx
```

return address

The stack

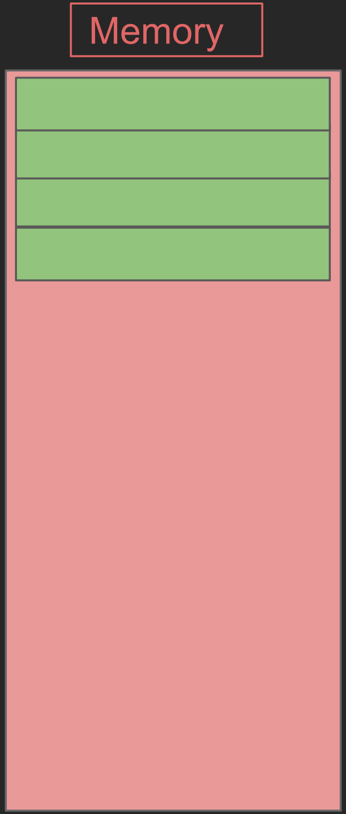
http://freepngimg.com/png/25783-coin-stack-transparent-image

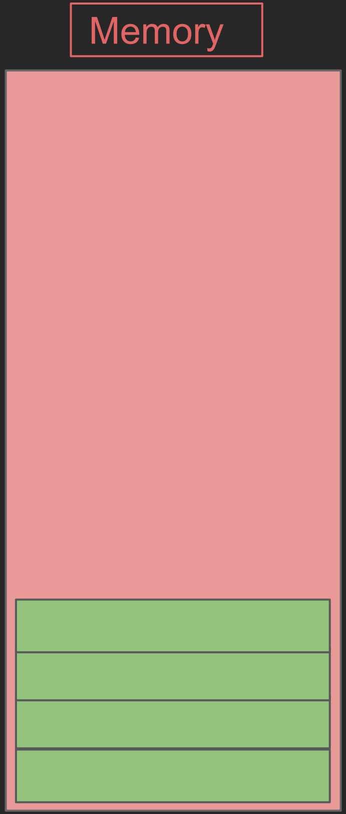
https://pixabay.com/en/plate-stack-tableware-plate-stack-629970/


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

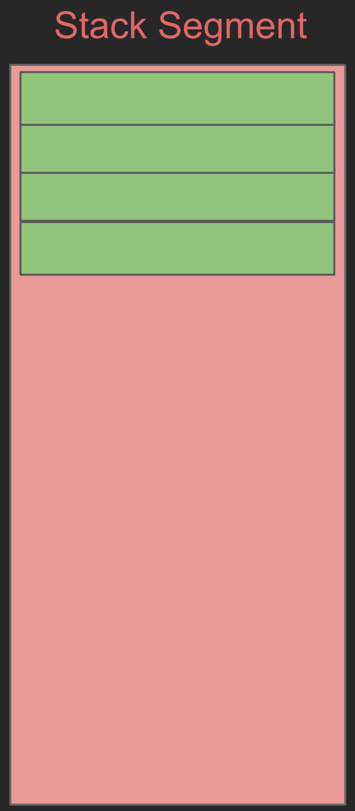
K. N. Toosi
University of Technology

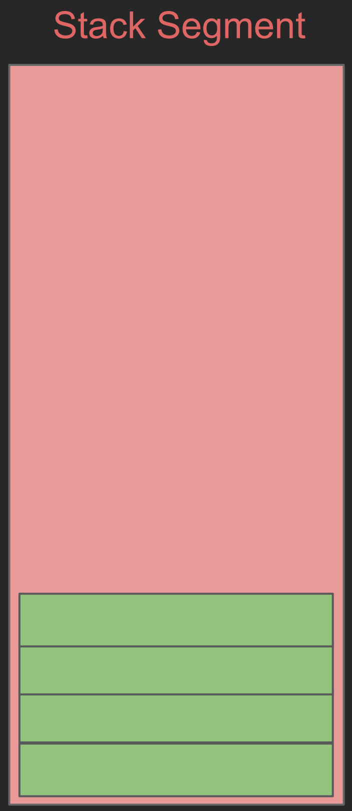
The stack



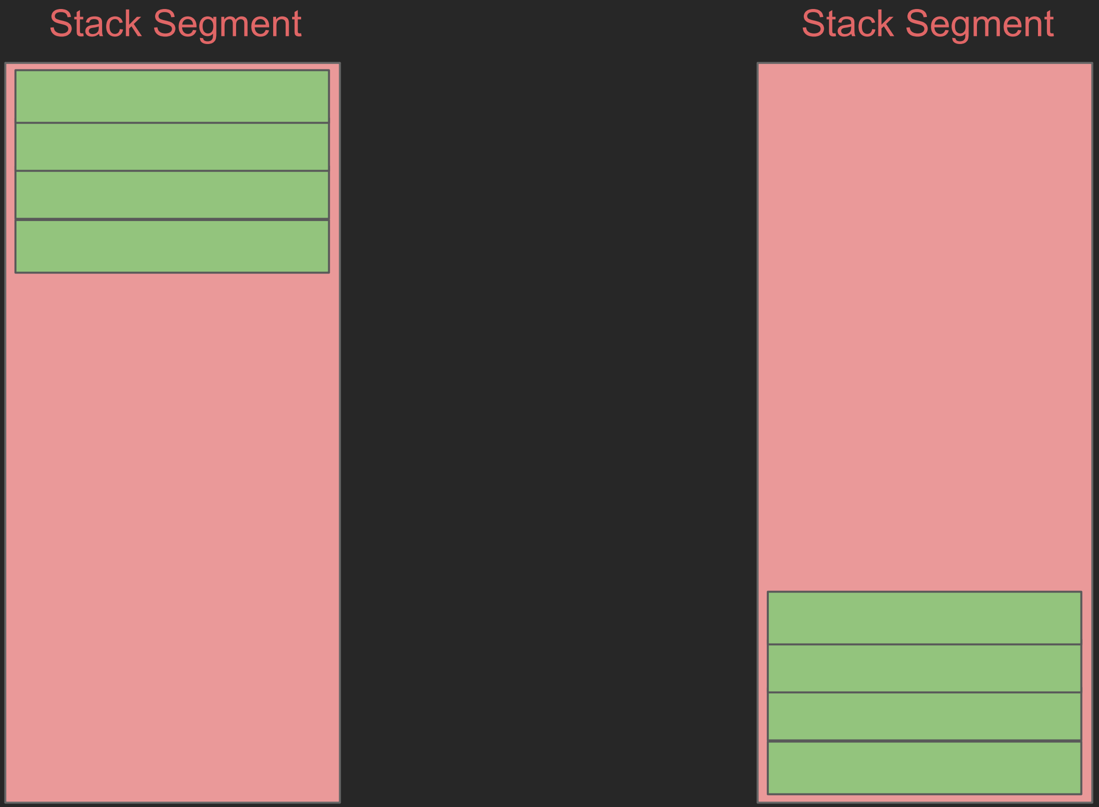


Implementing the stack



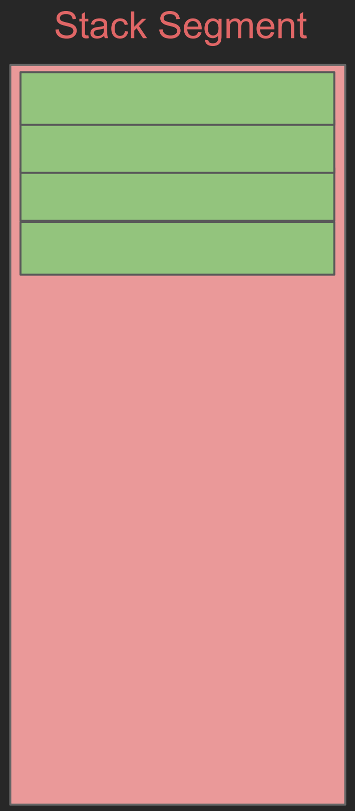


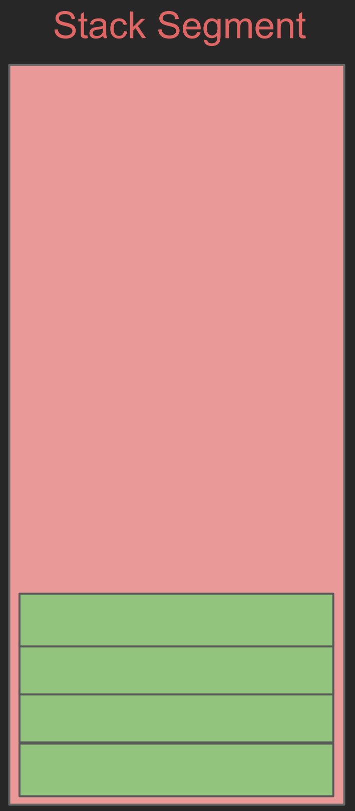
Implementing the stack



x86

Implementing the stack

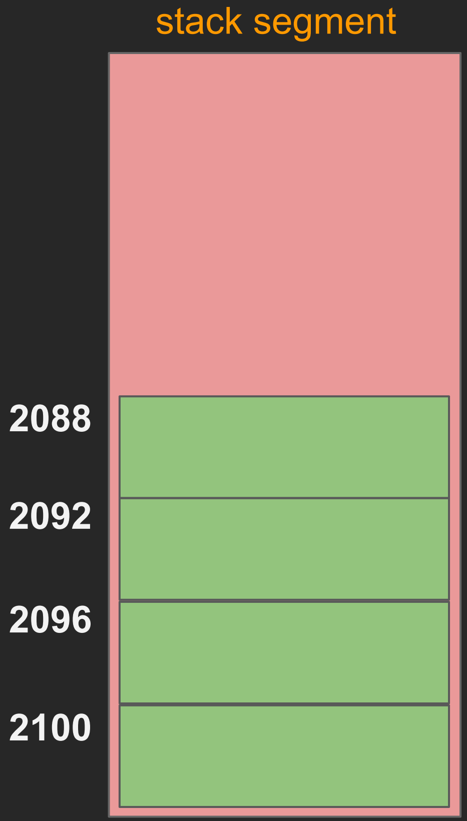




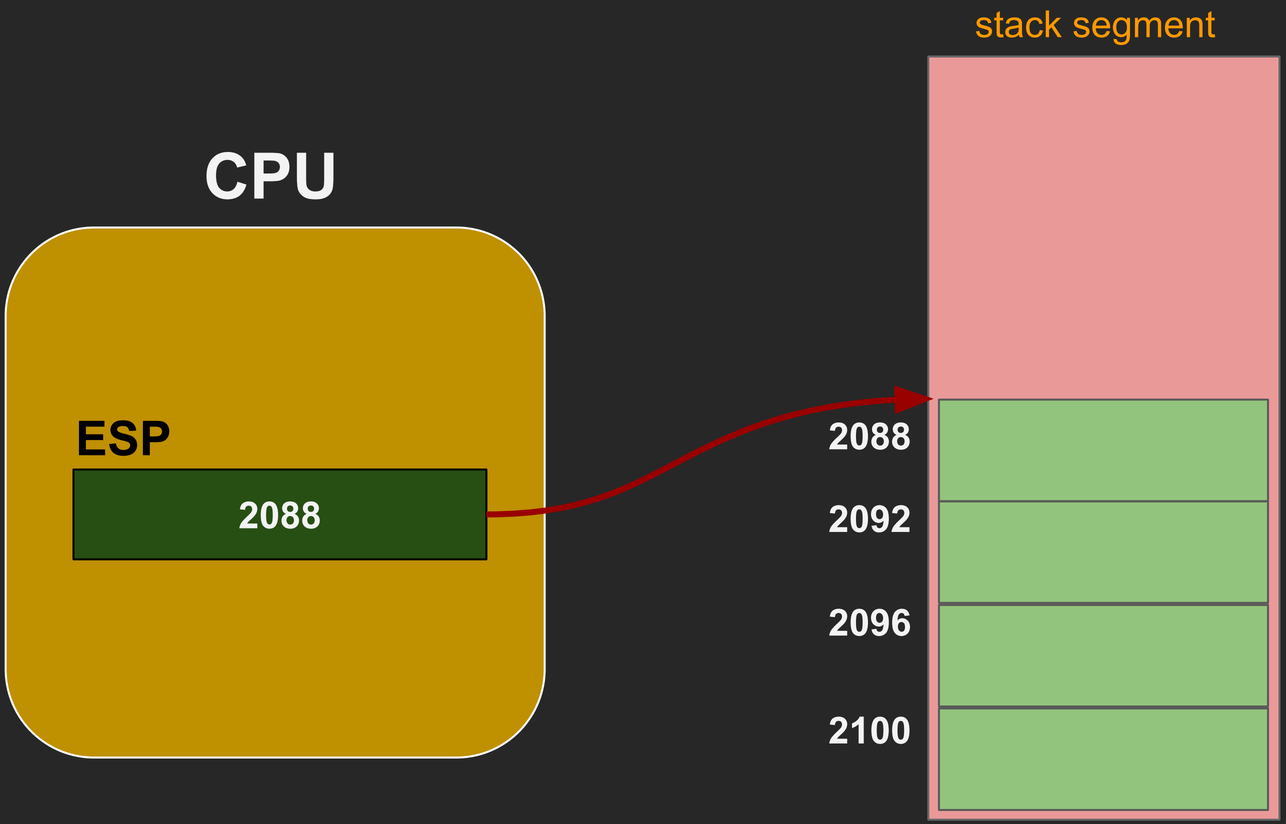
x86
(why?)

Implementing the stack

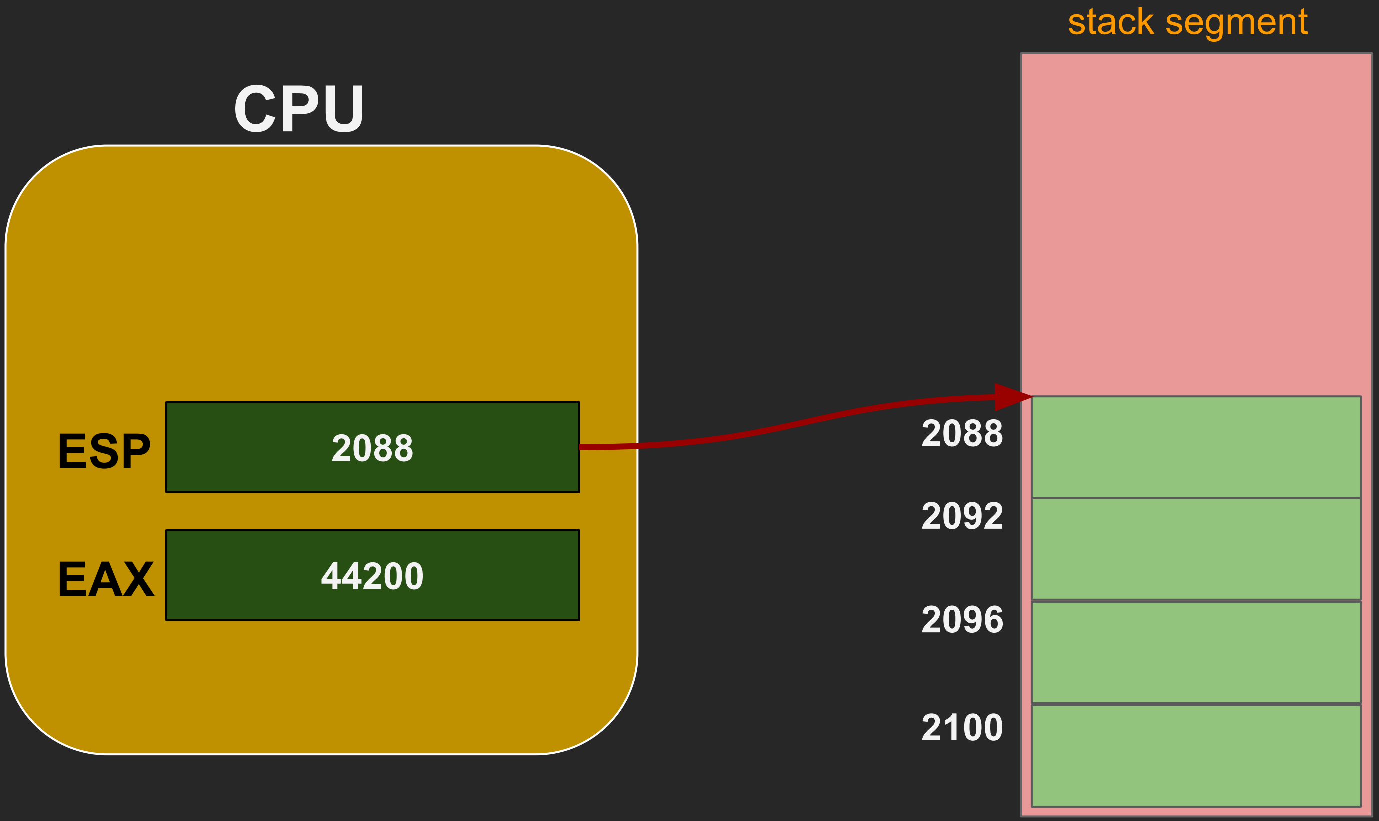
K. N. Toosi University of Technology



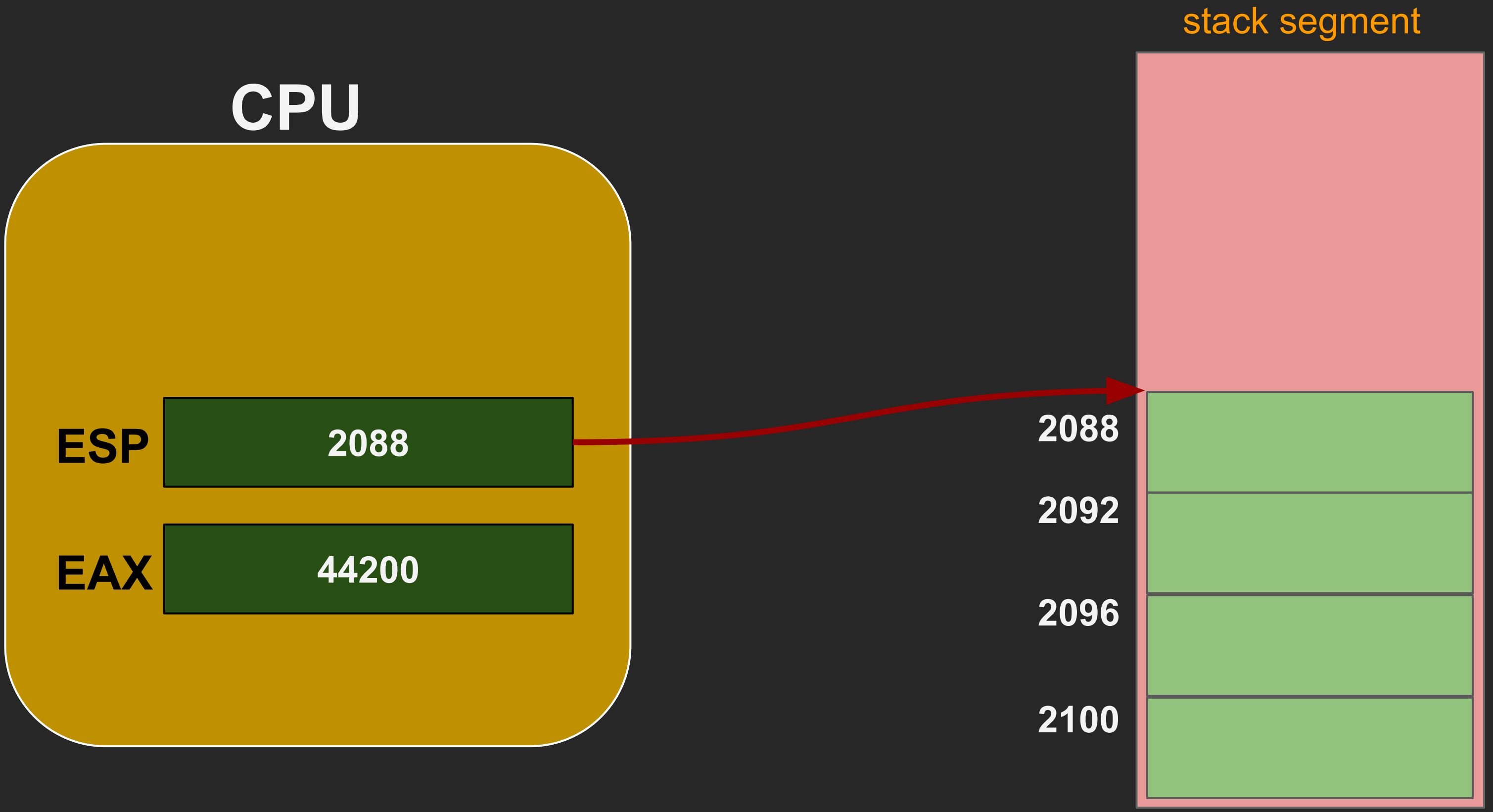
Stack Pointer (SP, ESP, RSP)



Pushing on the stack

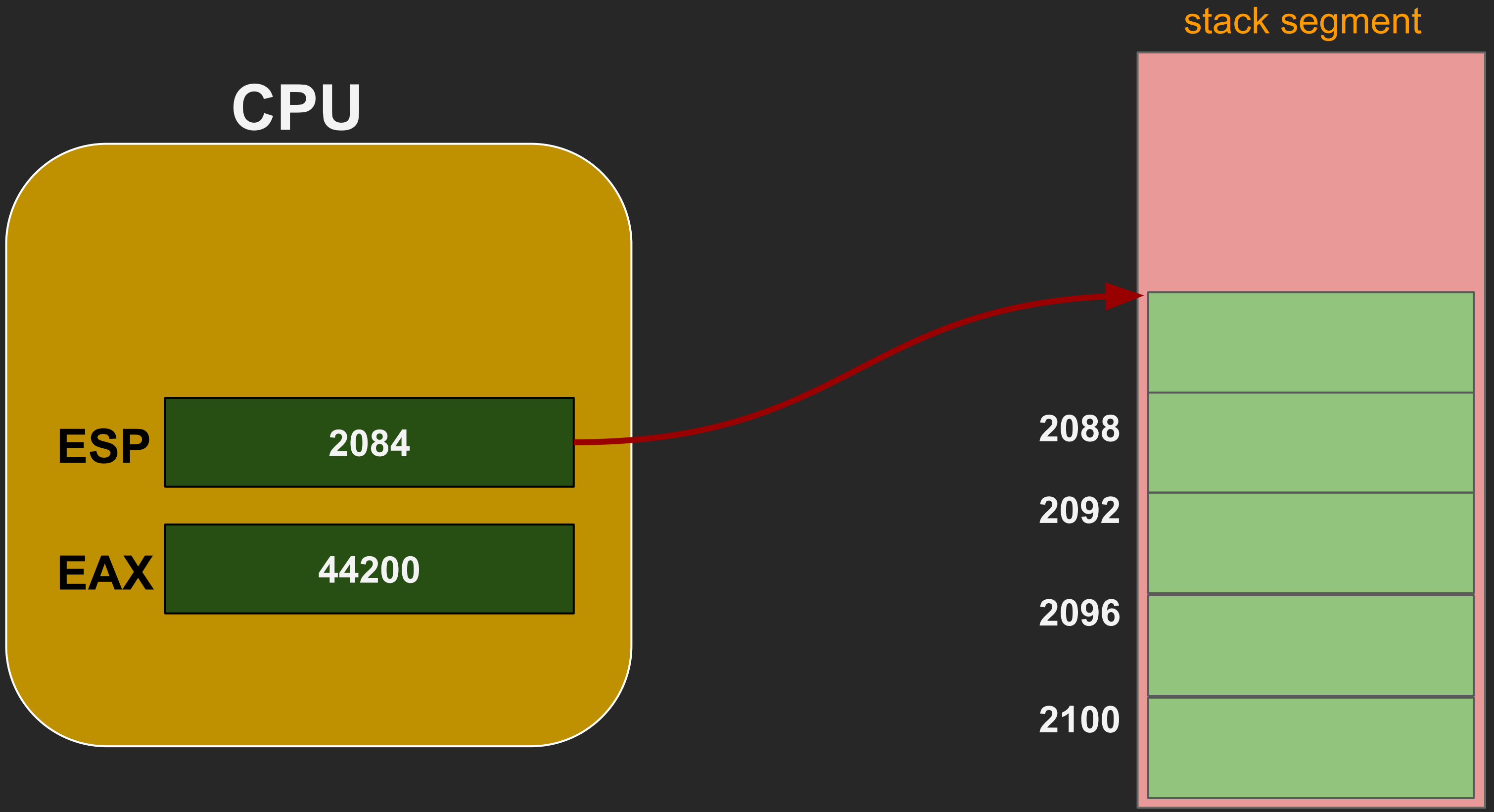


Push EAX on the stack



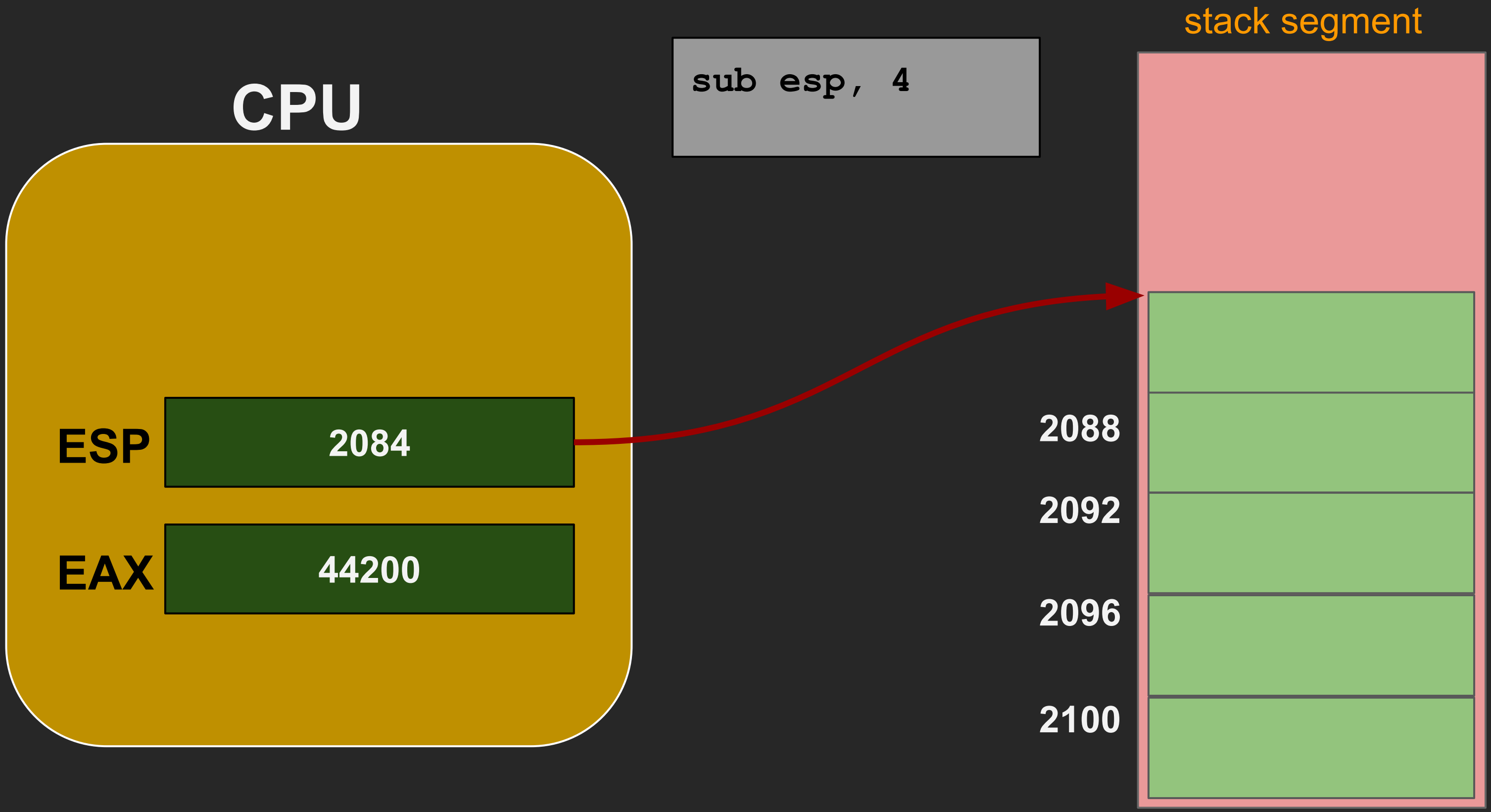
K. N. Toosi
University of Technology

Push EAX on the stack



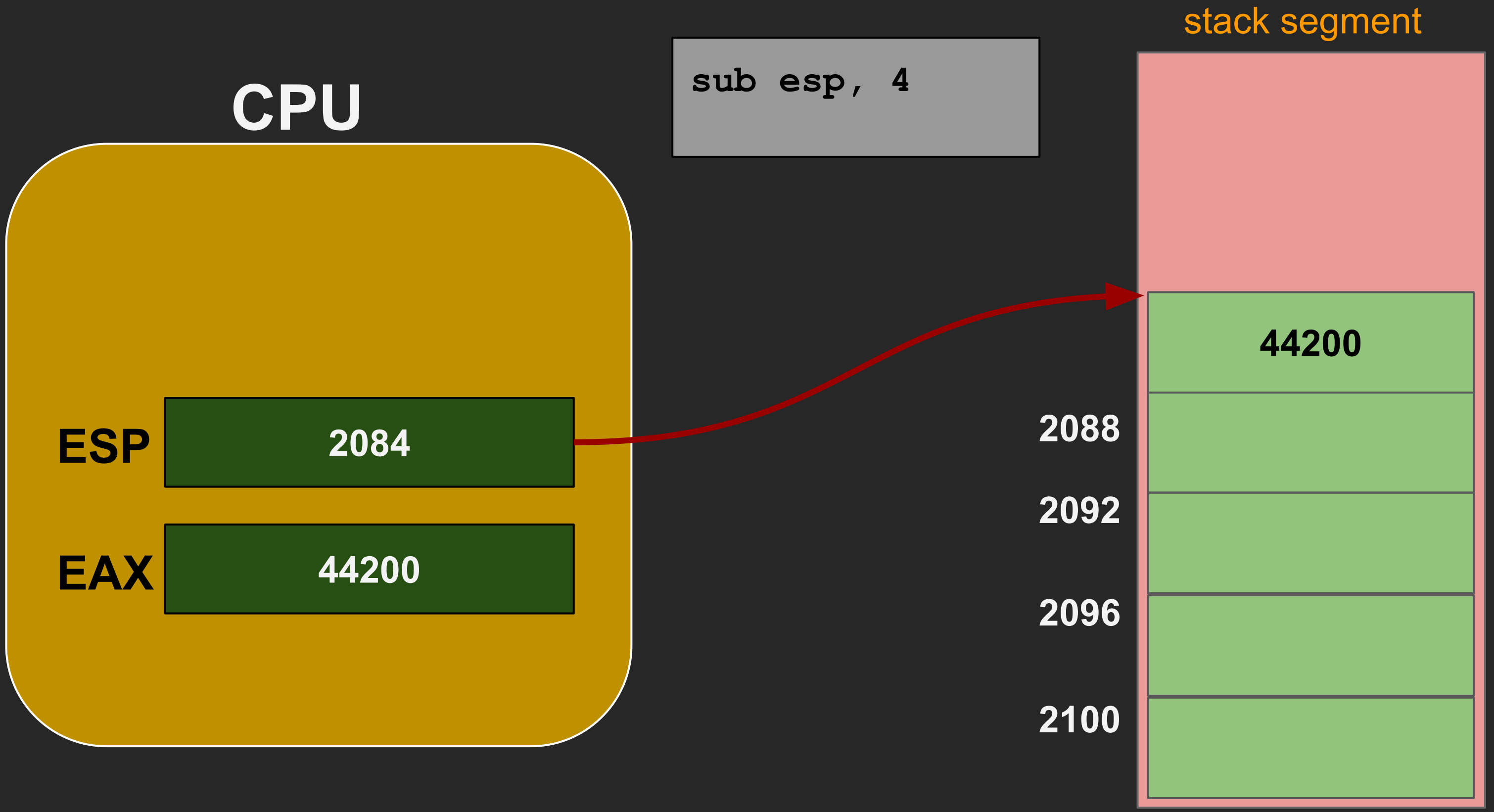
Push EAX on the stack

```
sub esp, 4
```



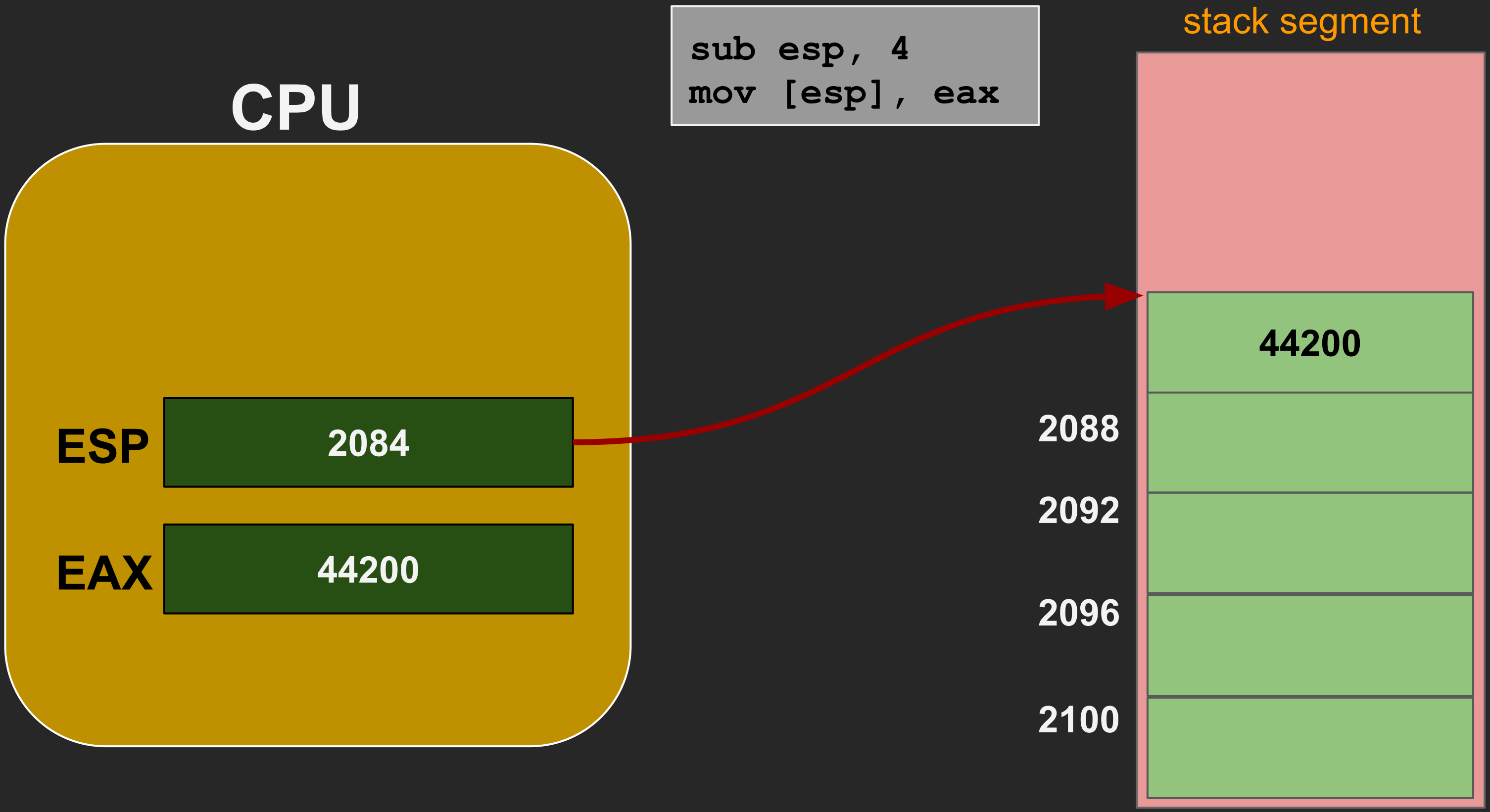
Push EAX on the stack

```
sub esp, 4
```



Push EAX on the stack

```
sub esp, 4
mov [esp], eax
```

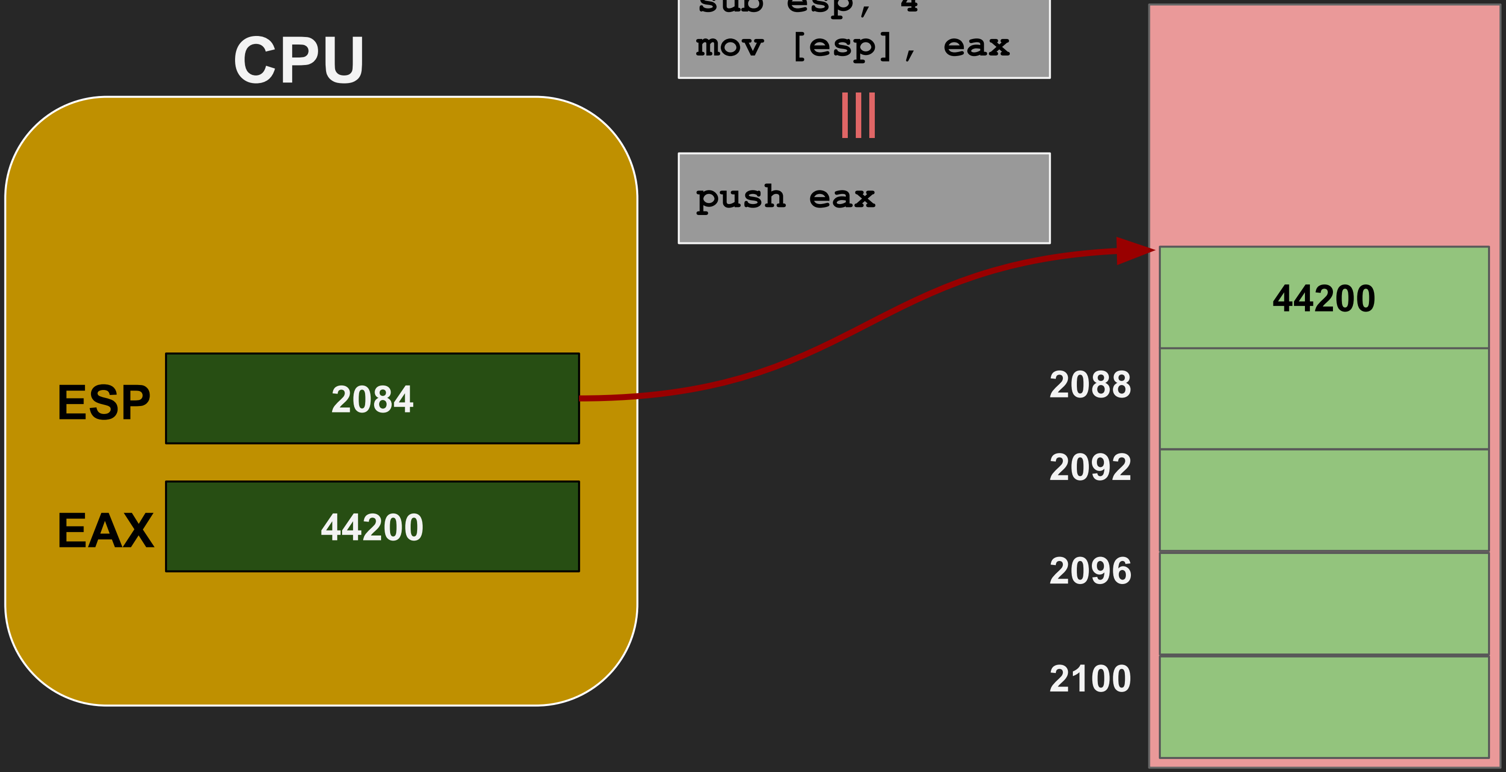


Push EAX on the stack

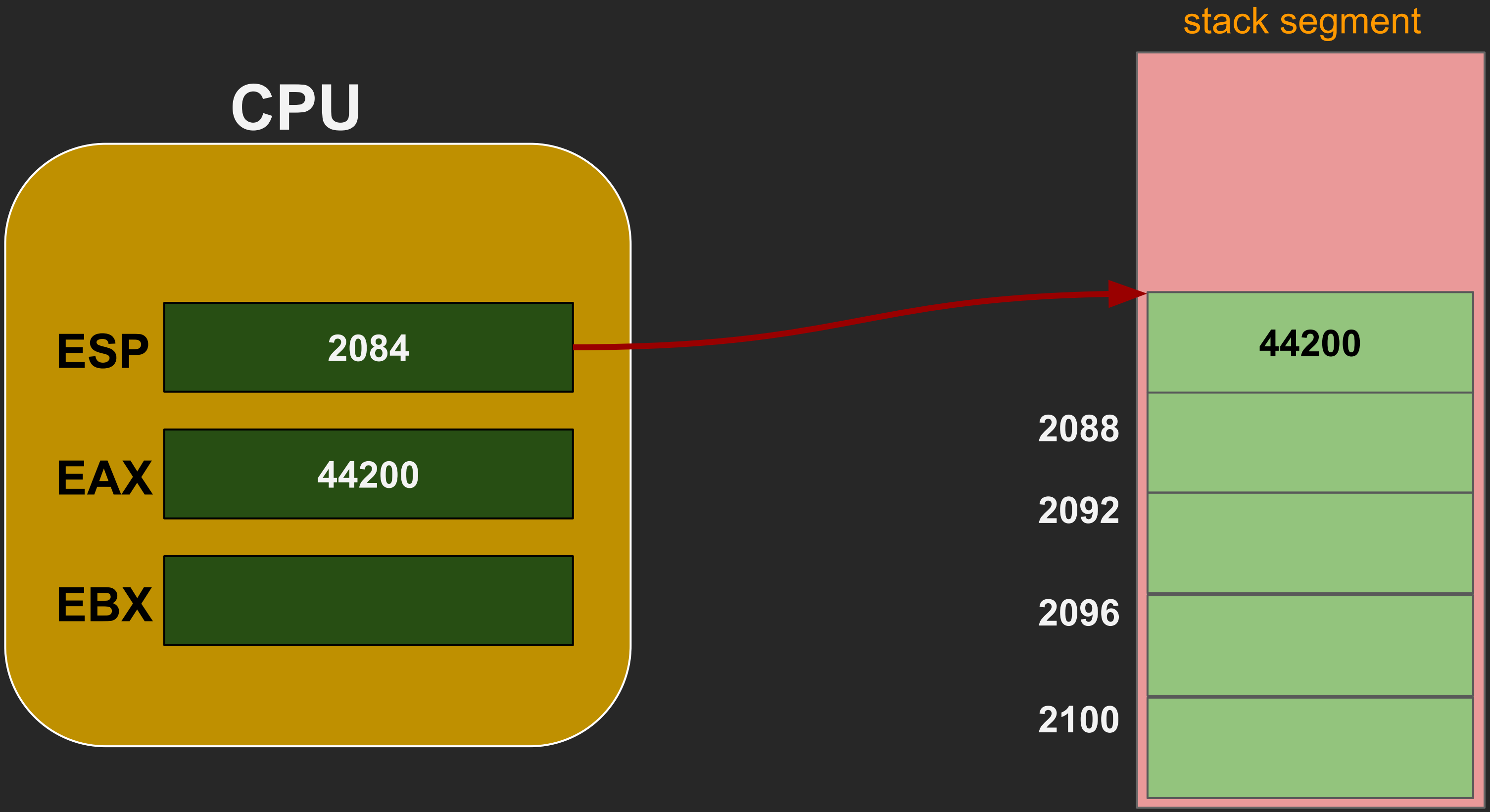
```
sub esp, 4
mov [esp], eax
```

$\equiv$

```
push eax
```

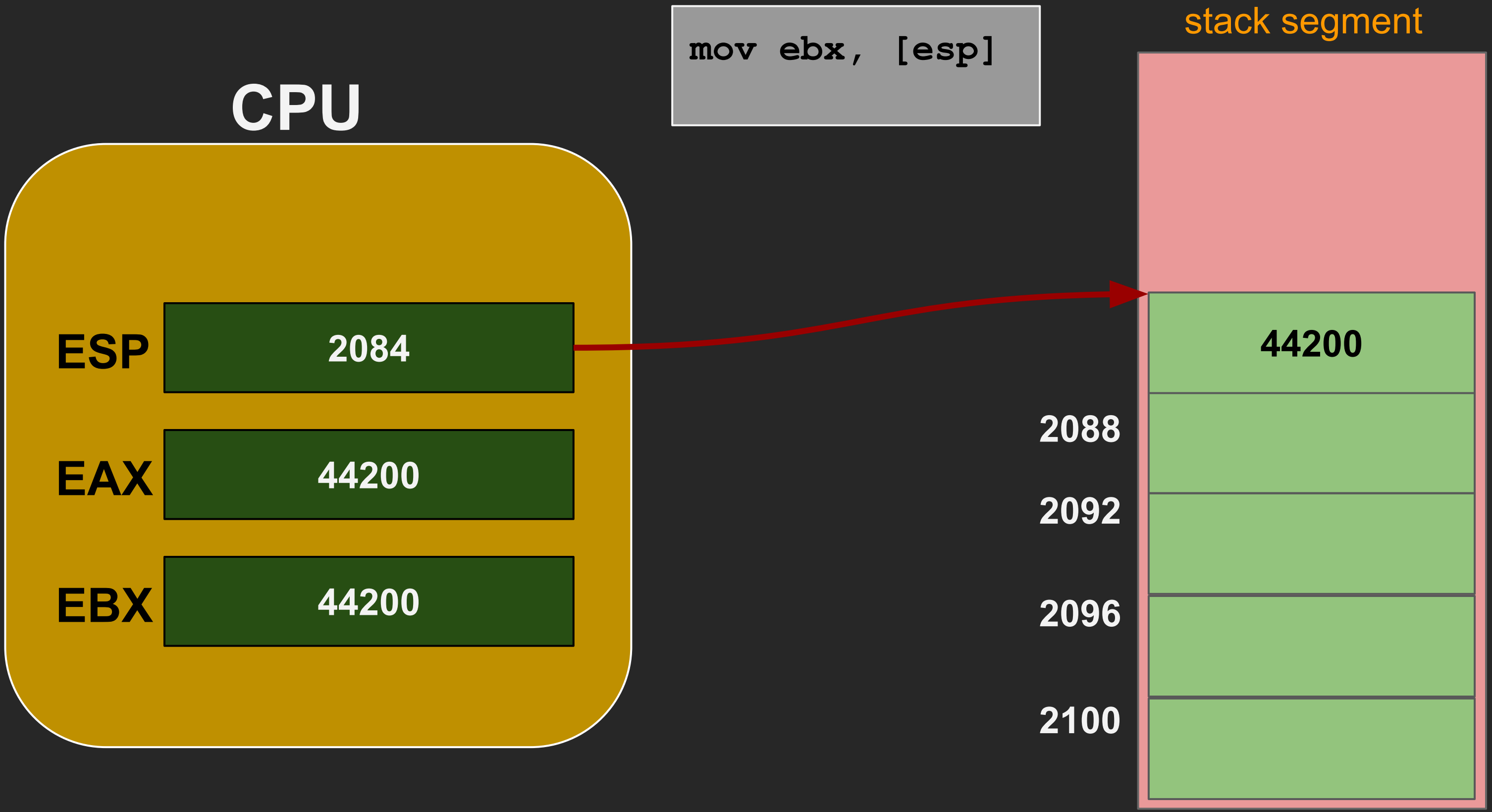


Pop into EBX


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

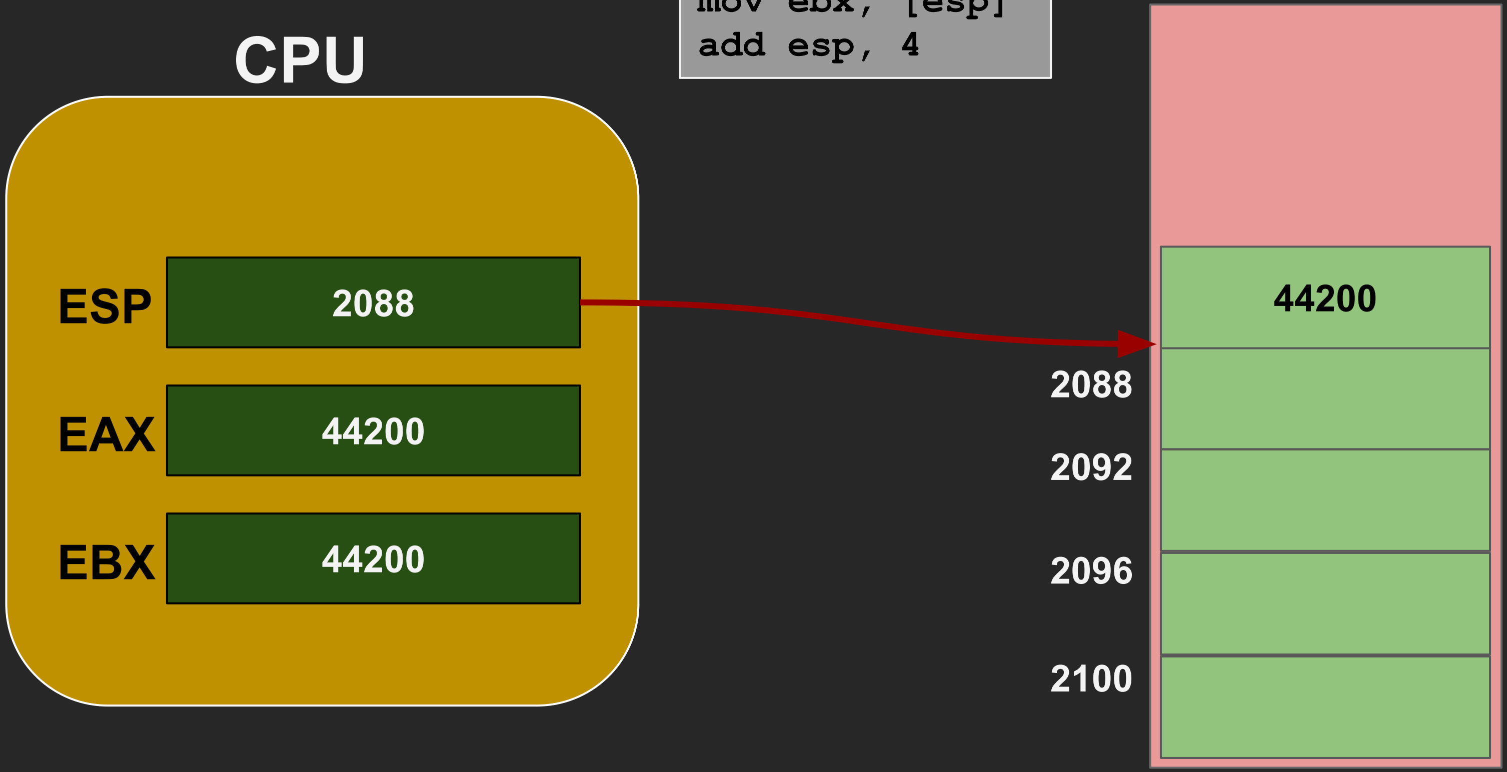
Pop into EBX

```
mov ebx, [esp]
```



Pop into EBX

```
mov ebx, [esp]
add esp, 4
```



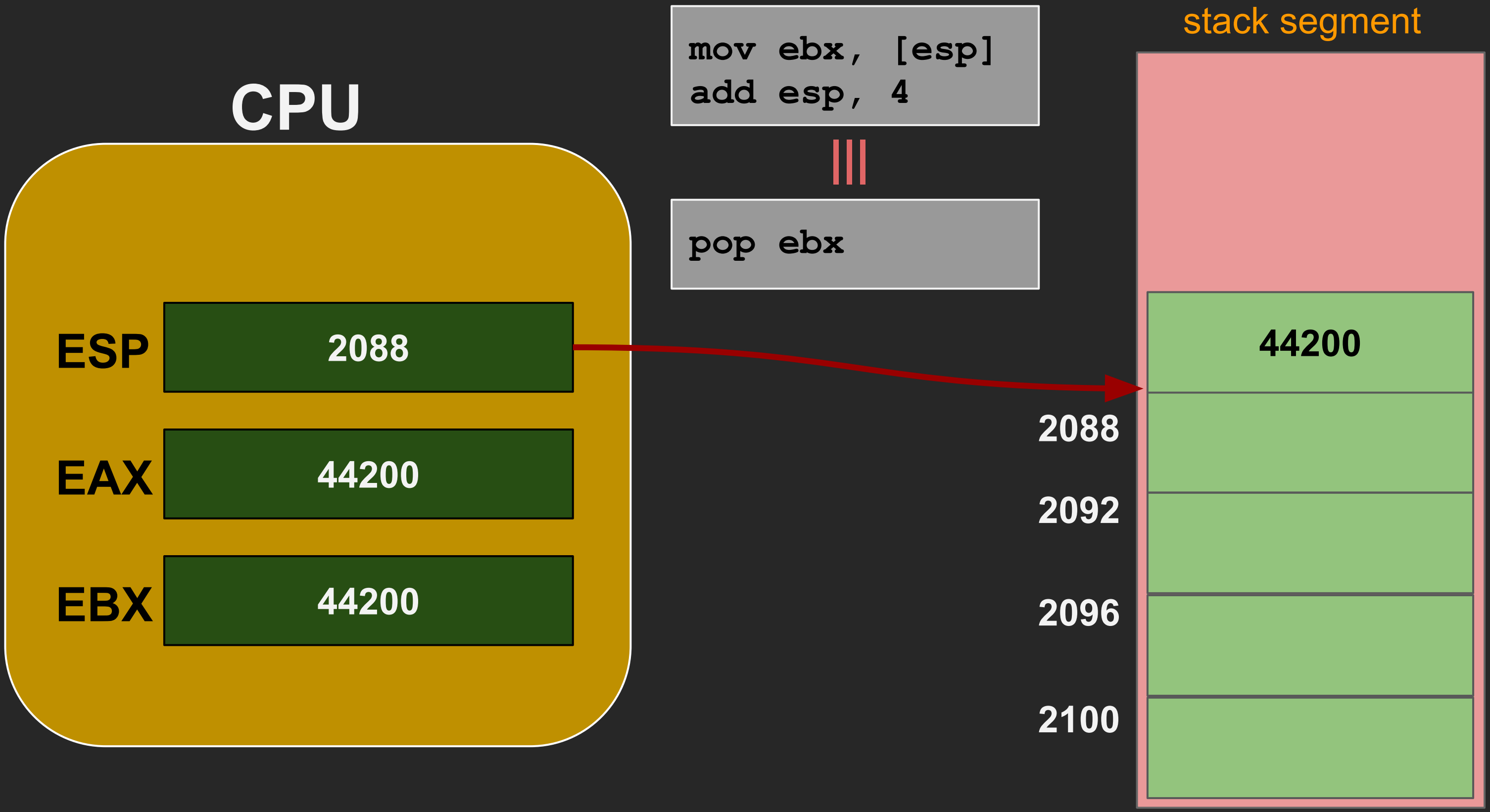
K. N. Toosi
University of Technology

Pop into EBX

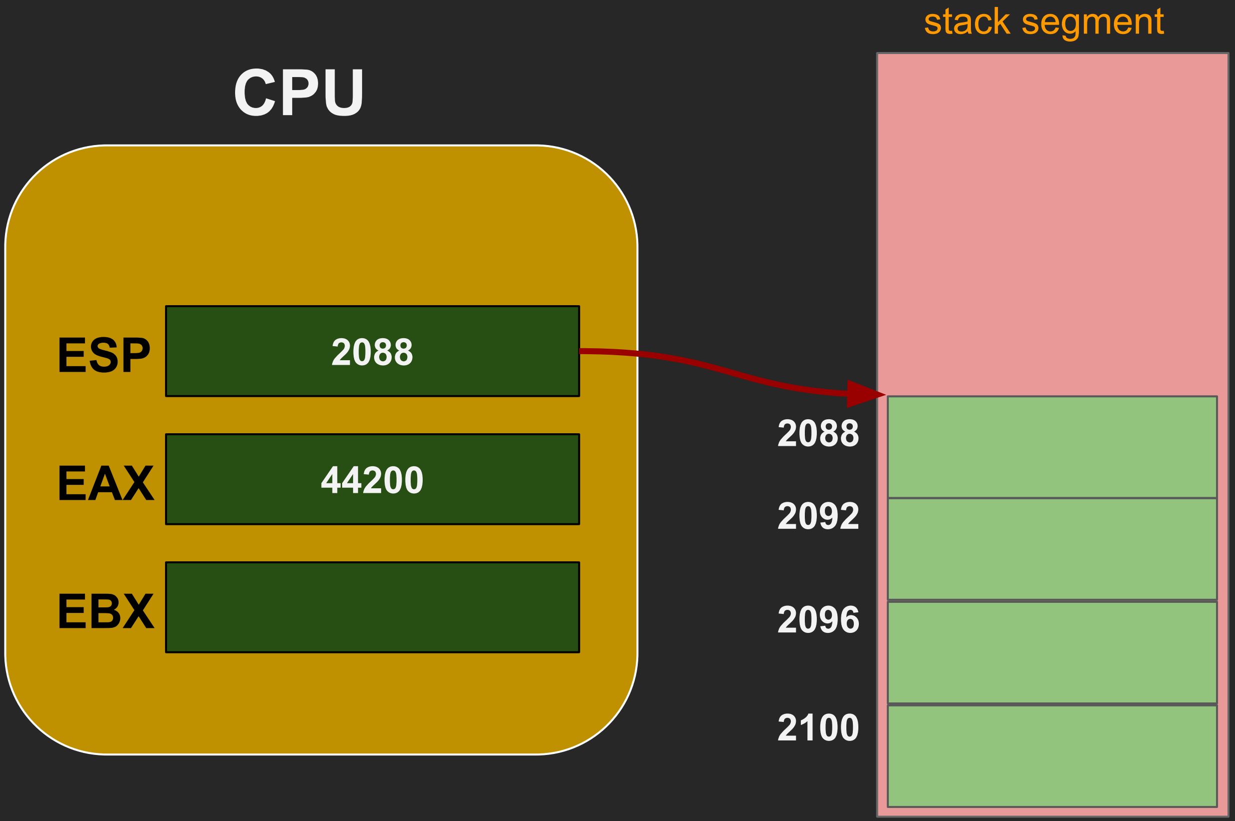
```
mov ebx, [esp]
add esp, 4
```

≡

```
pop ebx
```

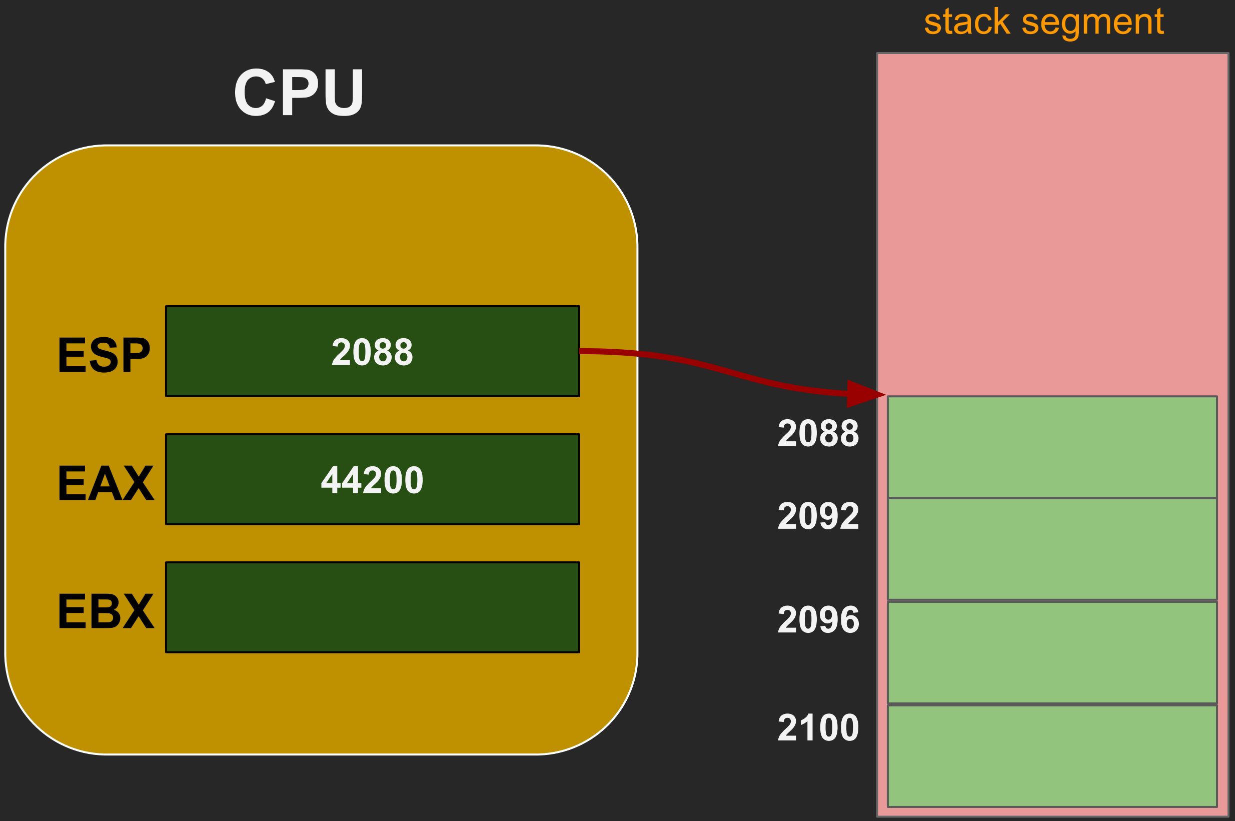


just pop 4 bytes (store nowhere)



```
pop ebx
```

just pop 4 bytes (store nowhere)

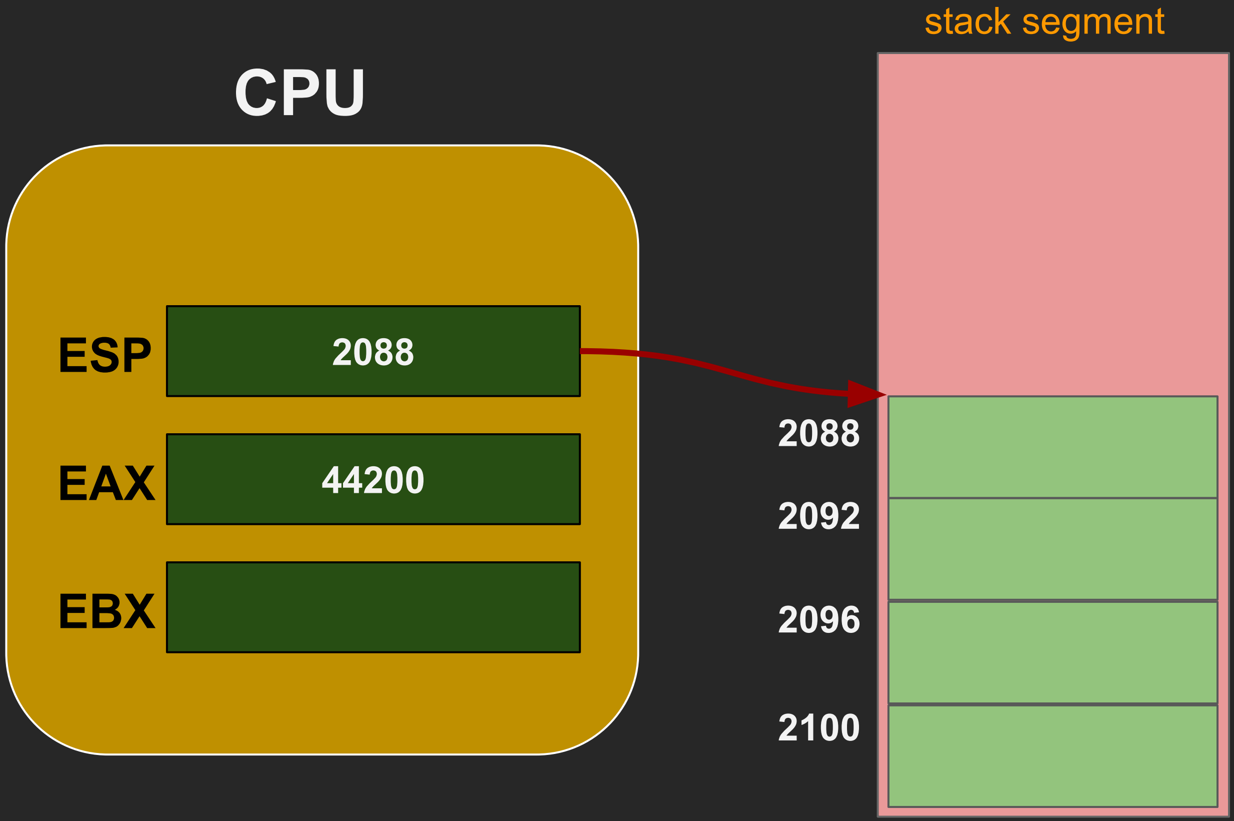


```
pop ebx
```

OR?

```
add esp, 4
```

just pop 4 bytes (store nowhere)



```
pop ebx
```

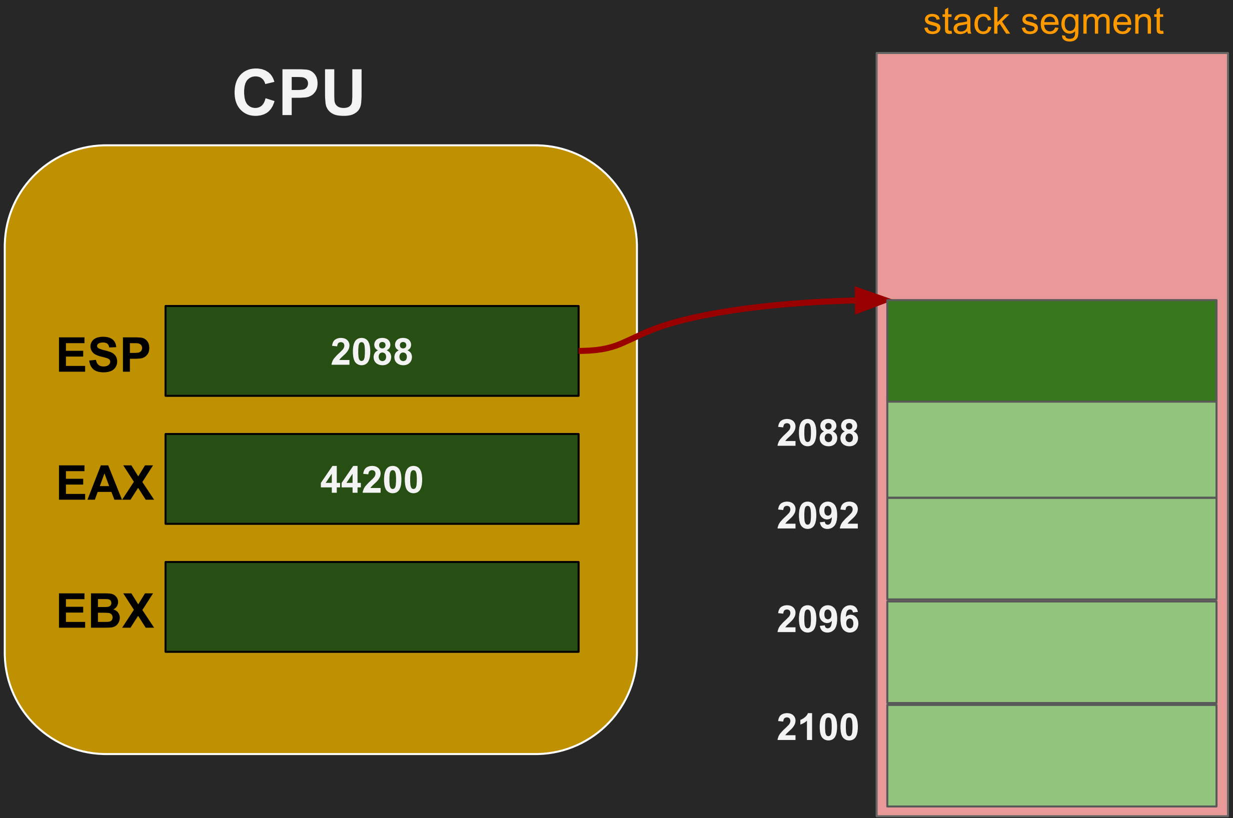
OR?

```
add esp, 4
```

```
add esp, 20
```

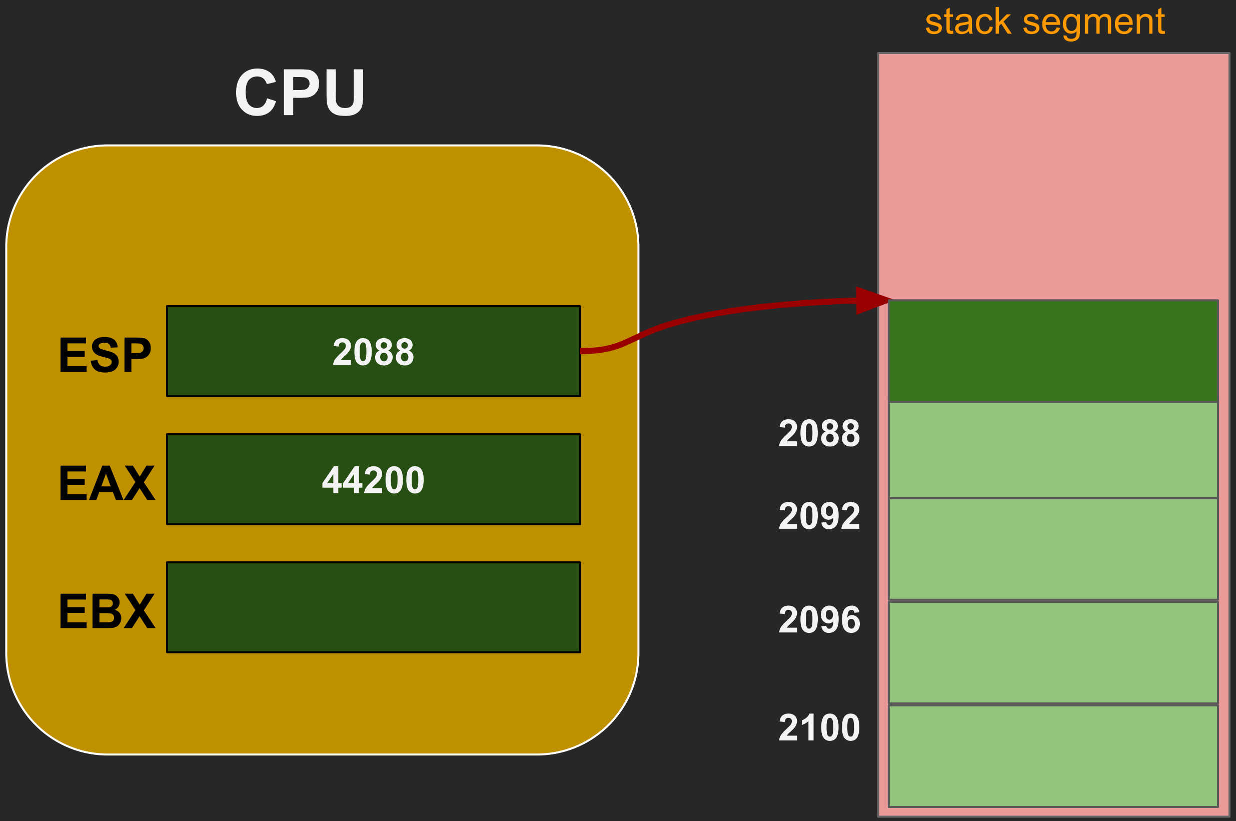
reserve memory on stack

K. N. Toosi
University of Technology



```
push edx
```

reserve memory on stack

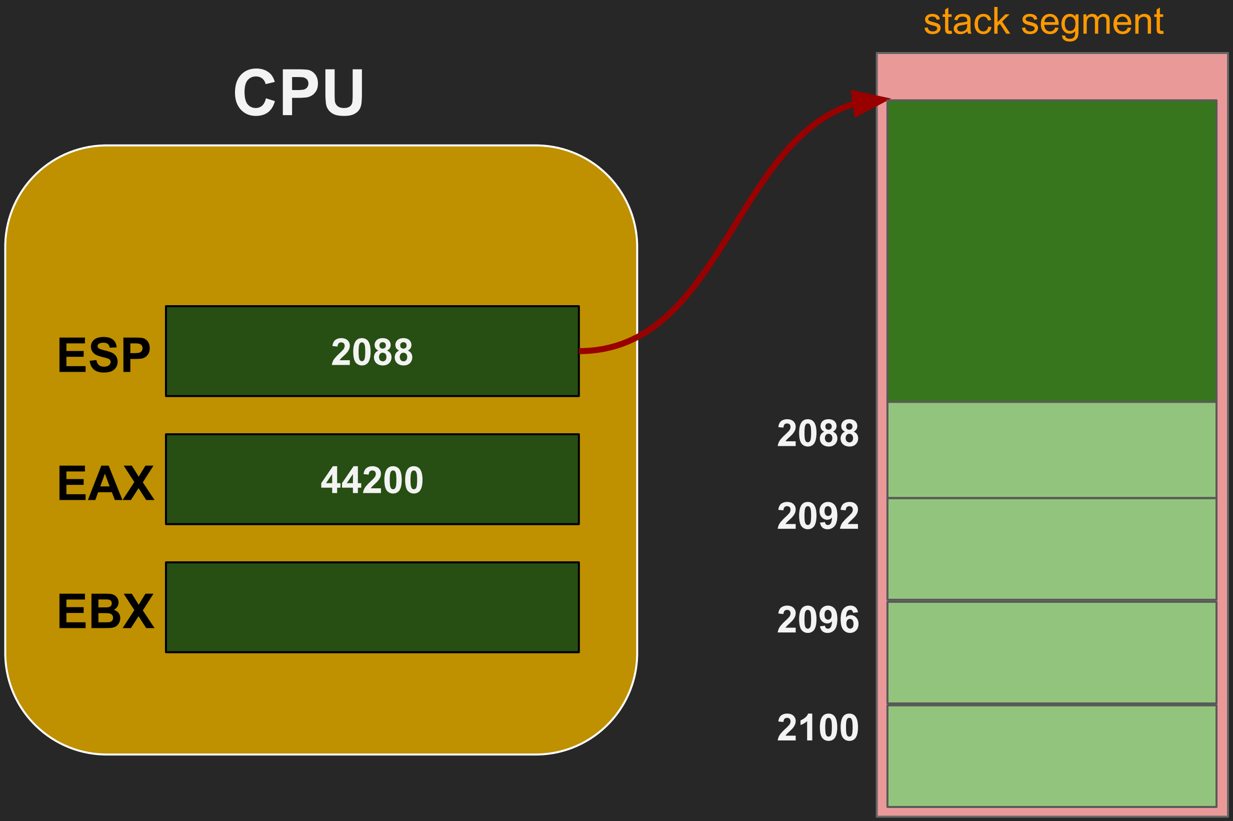


```
push edx
```

OR?

```
sub esp, 4
```

reserve memory on stack



```
push edx
```

OR?

```
sub esp, 4
```

```
sub esp, 24
```

Push and Pop

K. N. Toosi
University of Technology

```
Push reg/mem/immed

Pop  reg/mem
```

Practice

K. N. Toosi
University of Technology

```
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
        ??
```

K. N. Toosi
University of Technology

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
```

K. N. Toosi
University of Technology

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

K. N. Toosi
University of Technology

CALL is merely a form of jump!

the CALL instruction

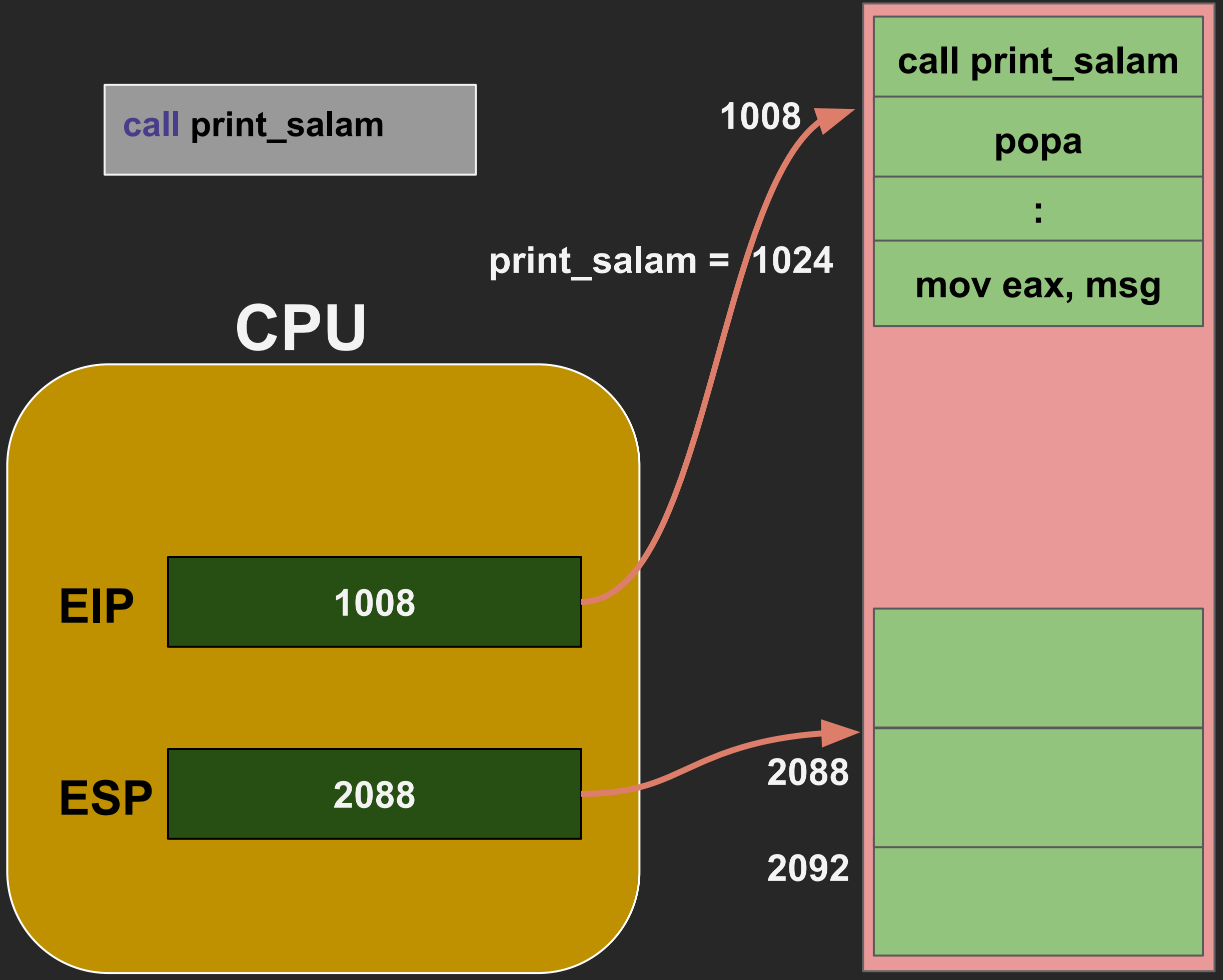
K. N. Toosi
University of Technology

CALL is merely a form of jump!

```
call label1
```

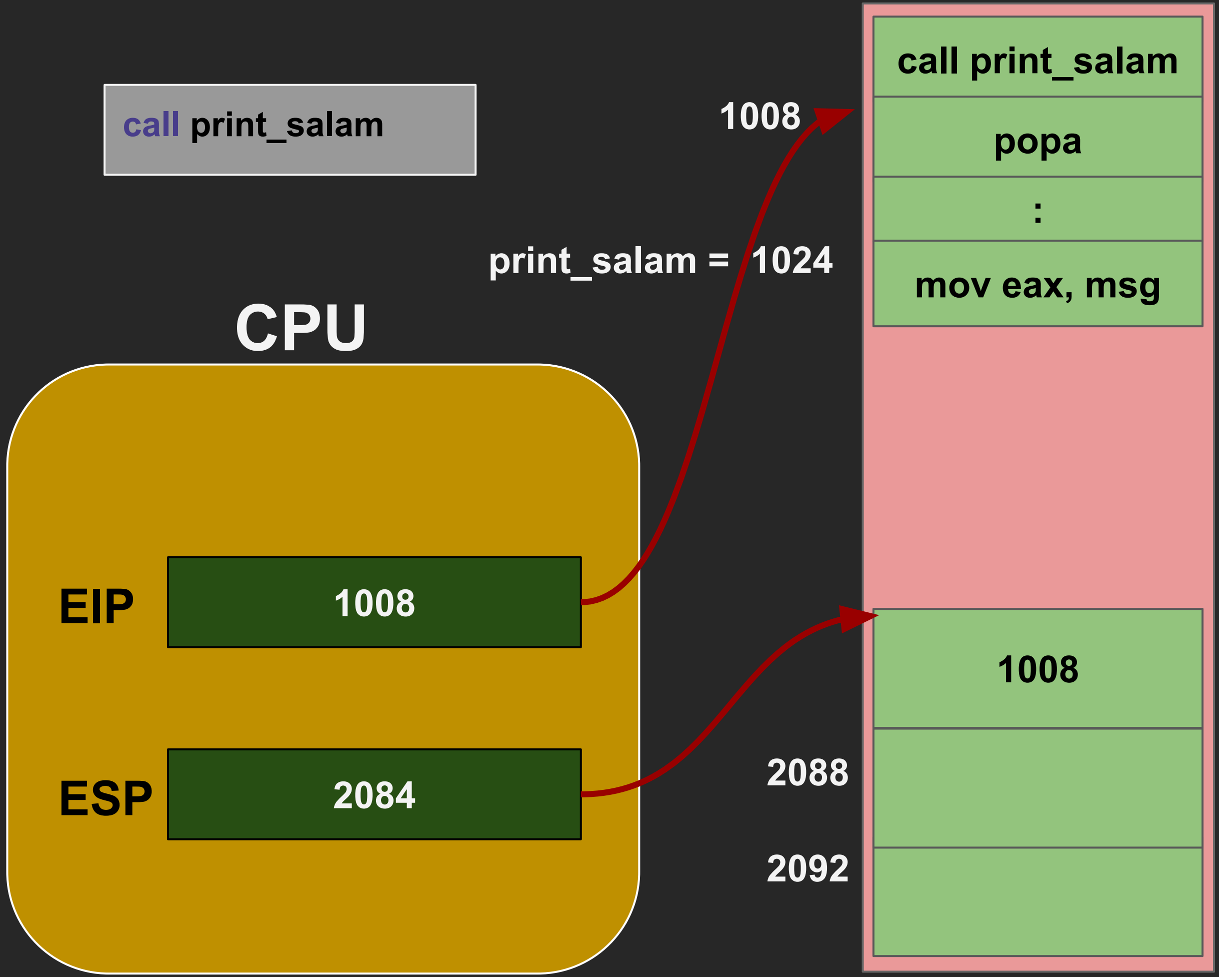
- Push return address (EIP) on stack
- jump to `label1`

```
call print_salam
```



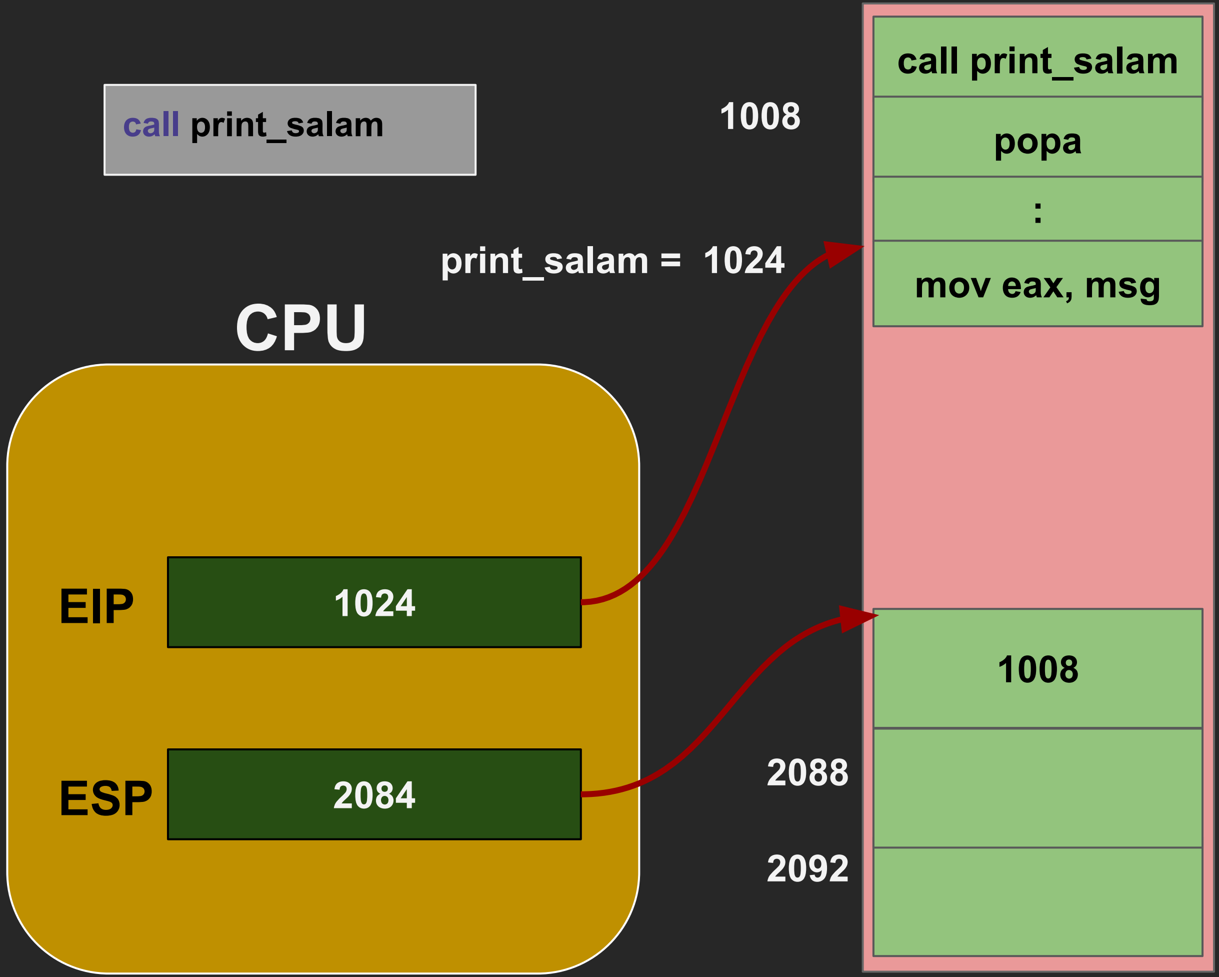
```
segment .data
msg:    db "Salaaaaam!", 10, 0
segment .text
        ⋮



    call print_salam
    popa
        ⋮

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

```
call print_salam
```



```
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
```



```
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

K. N. Toosi
University of Technology

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
```

K. N. Toosi
University of Technology

the RET instruction

RET is merely a form of jump!

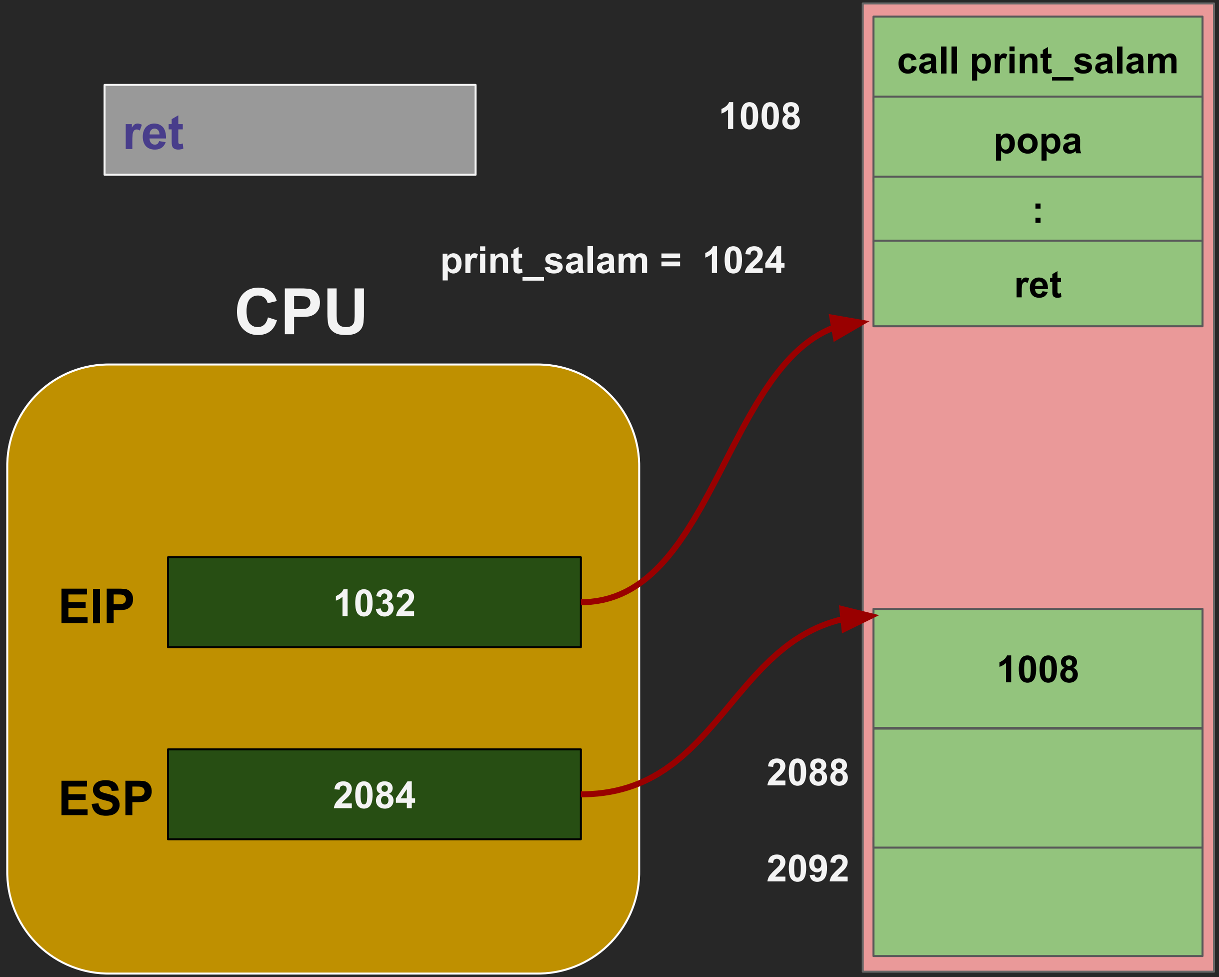
the RET instruction

K. N. Toosi
University of Technology

RET is merely a form of jump!

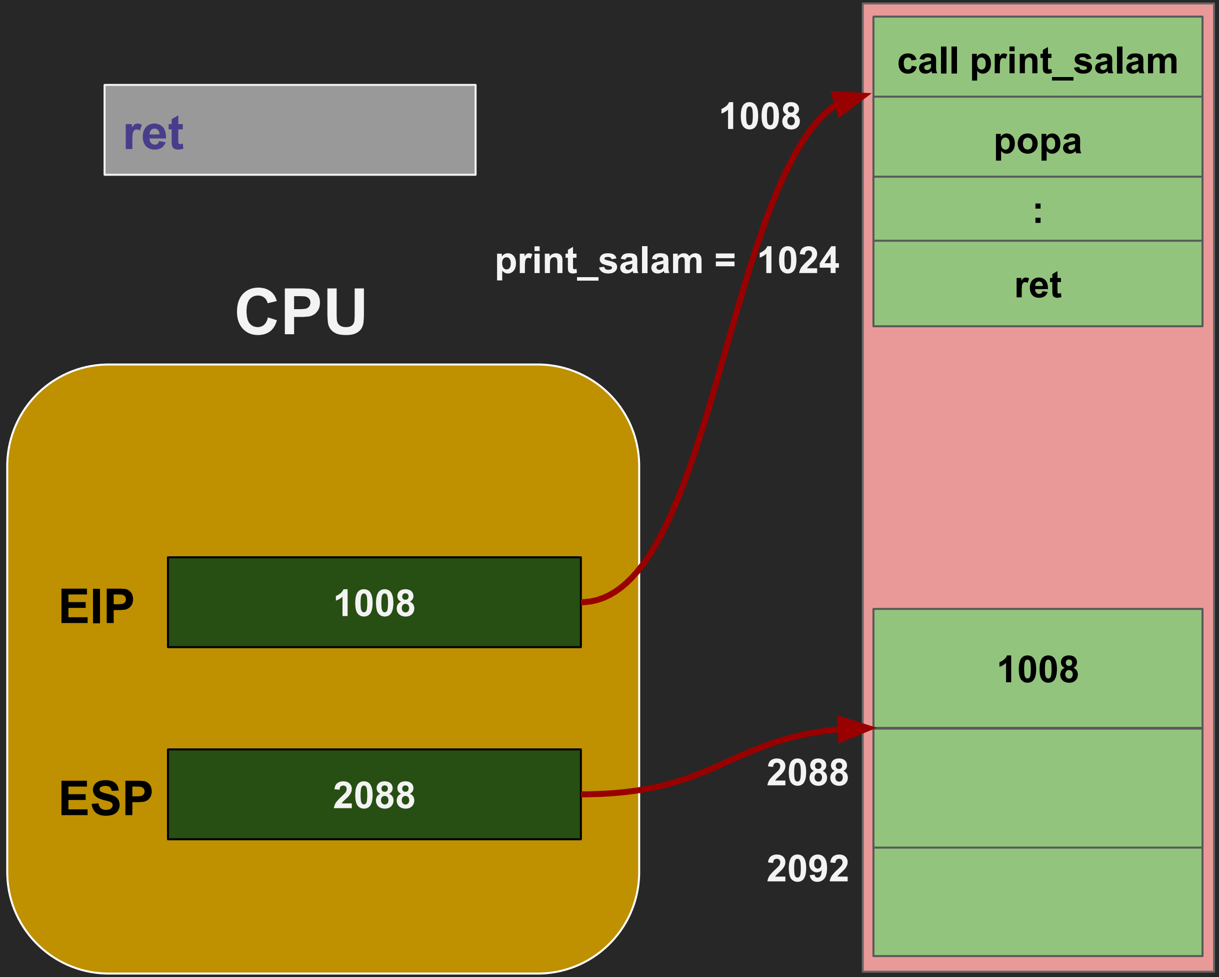
```
ret
```

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



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

           call print_salam
           popa
             ⋮

print_salam:
           mov eax, msg
           call print_string
  ──►      ret
```

ret



```
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