

# 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

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



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



```
segment .data
```

```
indirect2.asm
```

```
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



```
segment .data
```

```
indirect2.asm
```

```
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



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?



```
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?



```
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?



```
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");
```

```
}
```

```
segment .data
```

simplefunc1.asm

```
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");
```

```
}
```

What's wrong?

```
segment .data
```

simplefunc1.asm

```
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");
```

```
}
```

```
segment .data
```

simplefunc2.asm

```
msg:  db "Salaaaaam!", 10, 0
```

```
segment .text
```

```
    :
```

```
    jmp print_salam
```

```
l1:  _____→ return address
```

```
    :
```

```
    jmp print_salam
```

```
l2:  _____→
```

```
    :
```

```
print_salam:
```

```
    mov eax, msg
```

```
    call print_string
```

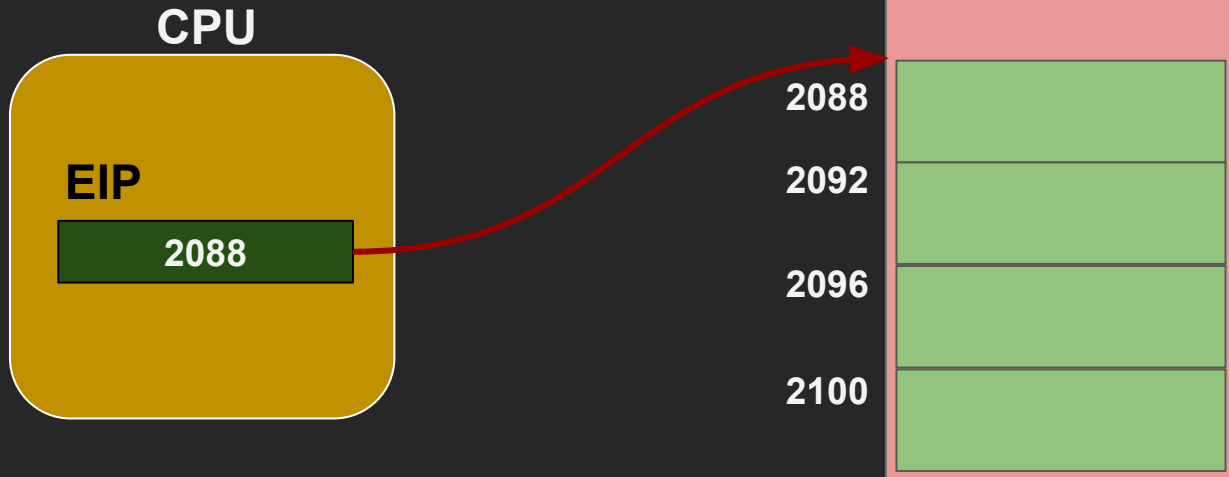
```
    jmp ?
```

# Looking closer at the jmp command



K. N. Toosi  
University of Technology

The **instruction pointer**  
(program counter) IP, EIP, RIP



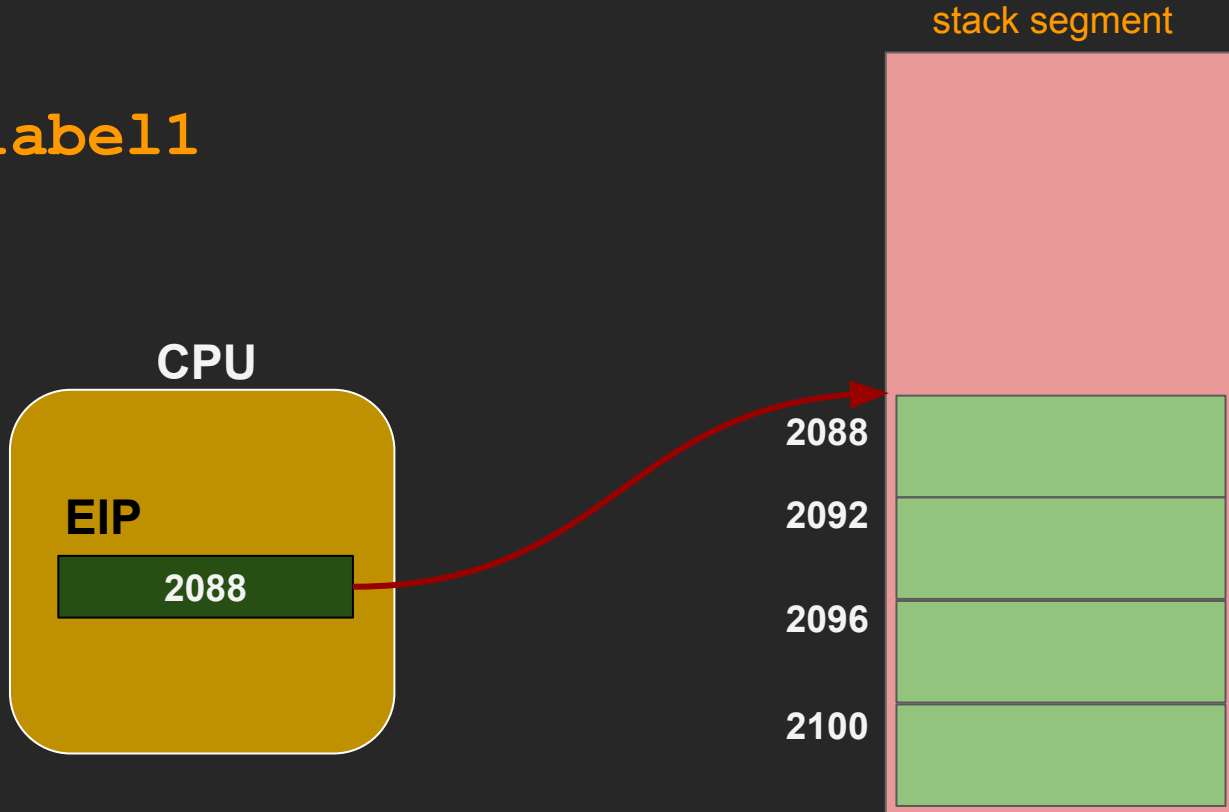


# Looking closer at the jmp command



K. N. Toosi  
University of Technology

jmp label1



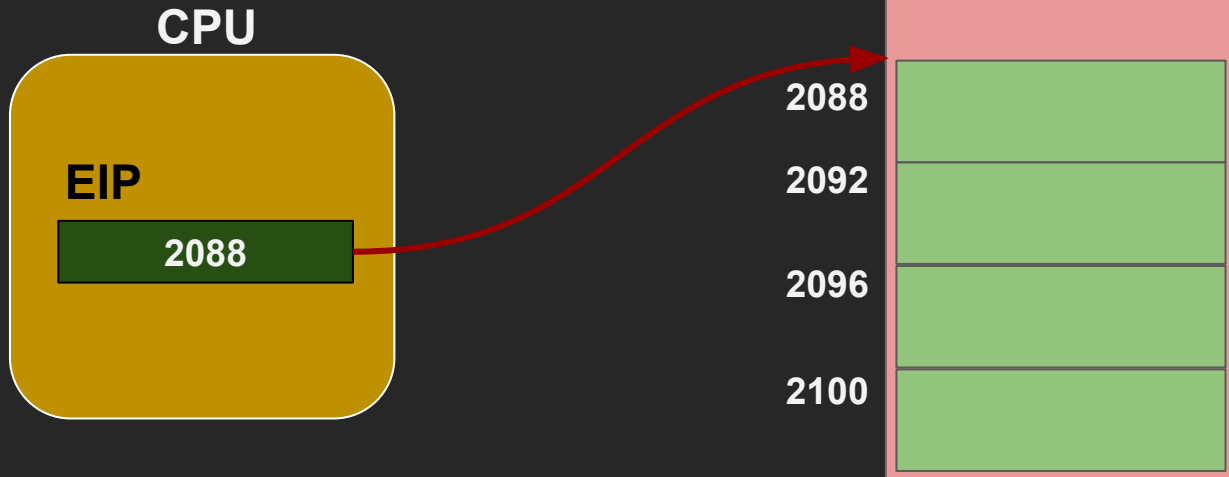
# Looking closer at the `jmp` command



K. N. Toosi  
University of Technology

`jmp label1`

How are `mov` and `jmp` similar?

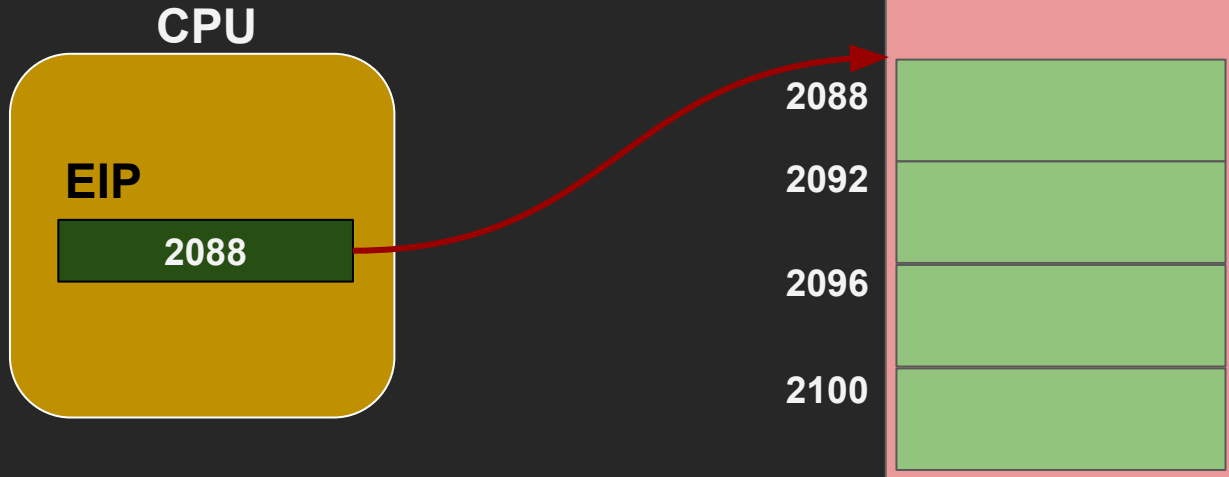


# Looking closer at the `jmp` command



K. N. Toosi  
University of Technology

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

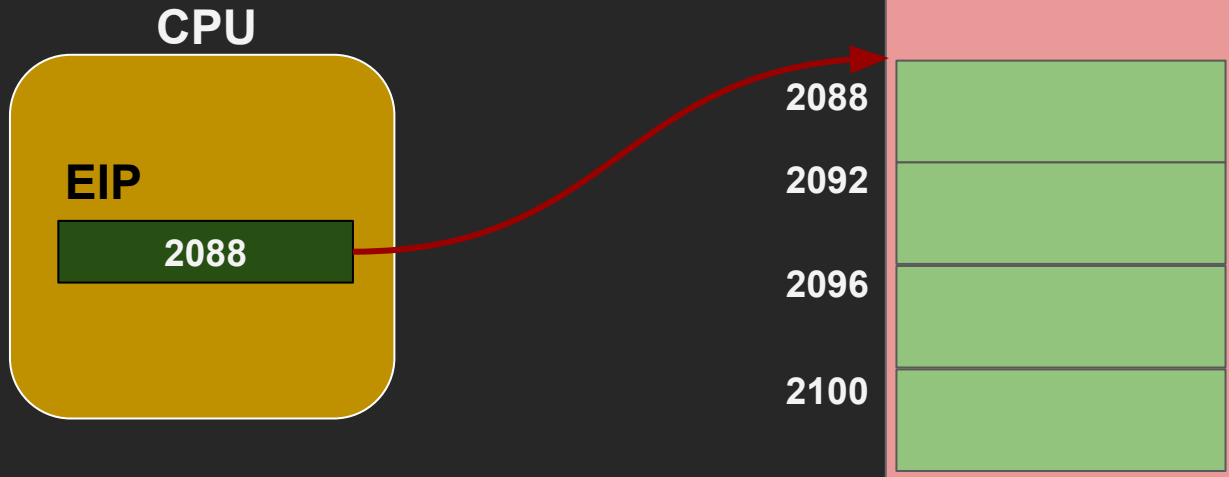


# Looking closer at the jmp command



K. N. Toosi  
University of Technology

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

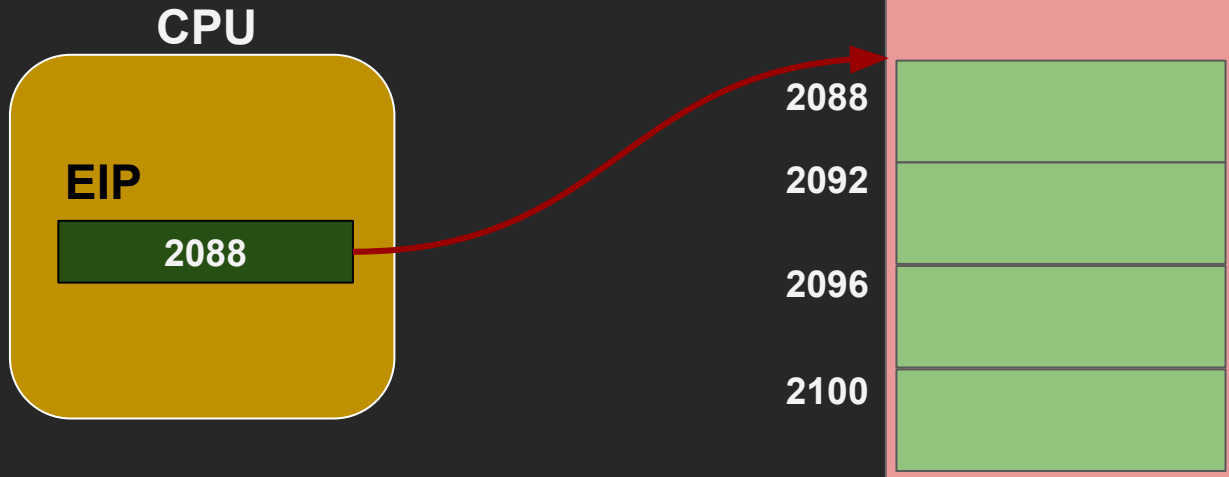


# Looking closer at the jmp command



K. N. Toosi  
University of Technology

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

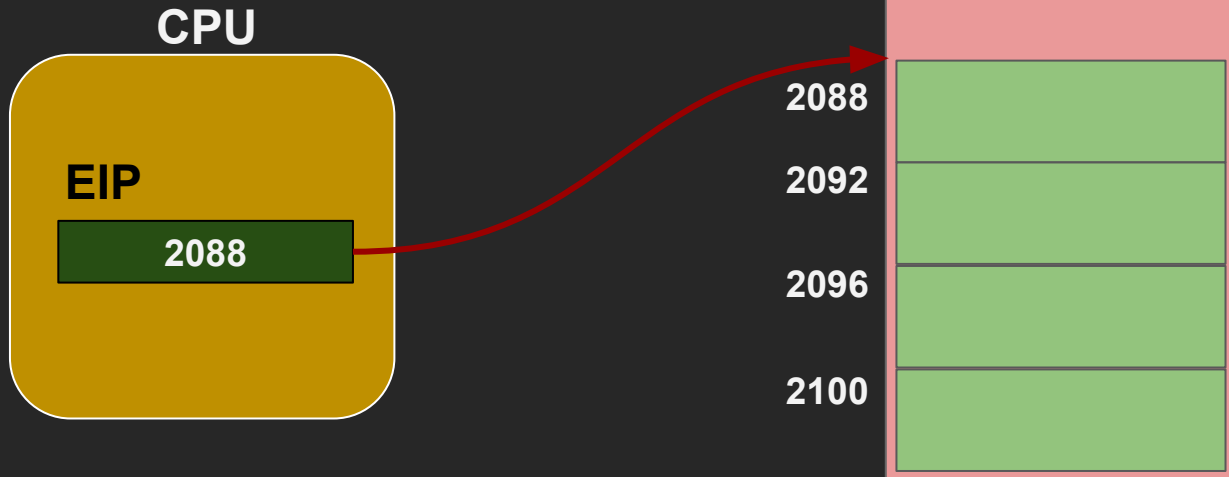


# Looking closer at the jmp command



K. N. Toosi  
University of Technology

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

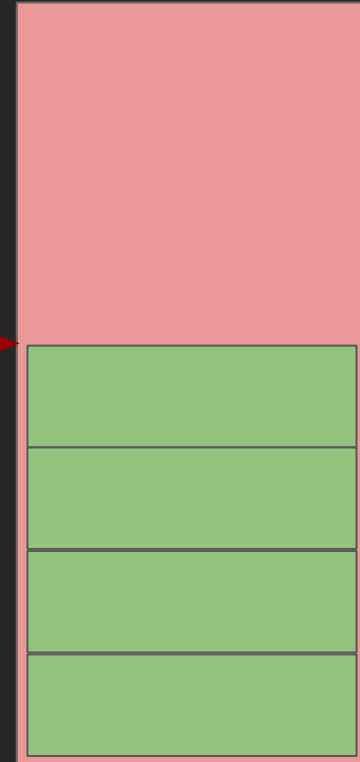


# Looking closer at the `jmp` command



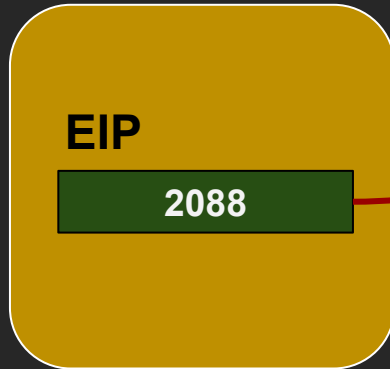
K. N. Toosi  
University of Technology

stack segment



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

CPU



2088

2092

2096

2100

# Indirect jump



K. N. Toosi  
University of Technology

**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");
```

```
}
```

```
segment .data
```

```
msg: db "Salaaaaam!", 10, 0
```

```
segment .text
```

```
:
```

```
    jmp print_salam
```

```
I1: → return address
```

```
    jmp print_salam
```

```
I2: →
```

```
:
```

```
print_salam:
```

```
    mov eax, msg
```

```
    call print_string
```

```
    jmp ?
```

# 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
```

```
I1:  → return address
```

```
        jmp print_salam
```

```
I2:  →
```

```
    :
```

```
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");
```

```
}
```

```
segment .data
```

```
msg:  db "Salaaaaam!", 10, 0
```

```
segment .text
```

```
    :
```

```
    mov edx, I1
```

```
    jmp print_salam
```

```
I1:  → return address
```

```
    mov edx, I2
```

```
    jmp print_salam
```

```
I2:  →
```

```
    :
```

```
print_salam:
```

```
    mov eax, msg
```

```
    call print_string
```

```
    jmp edx
```

simplefunc3.asm



# How to implement subprograms?

```
void print_salam(void);
```

```
int main() {
```

```
    print_salam();
```

```
}
```

```
void print_salam() {
```

```
    printf("Salaaaaam!\n");
```

```
}
```

Limitations?

```
segment .data
```

simplefunc3.asm

```
msg:  db "Salaaaaam!", 10, 0
```

```
segment .text
```

```
    :
```

```
    mov edx, I1
```

```
    jmp print_salam
```

```
I1:  → return address
```

```
    mov edx, I2
```

```
    jmp print_salam
```

```
I2:  →
```

```
    :
```

```
print_salam:
```

```
    mov eax, msg
```

```
    call print_string
```

```
    jmp edx
```

# The stack



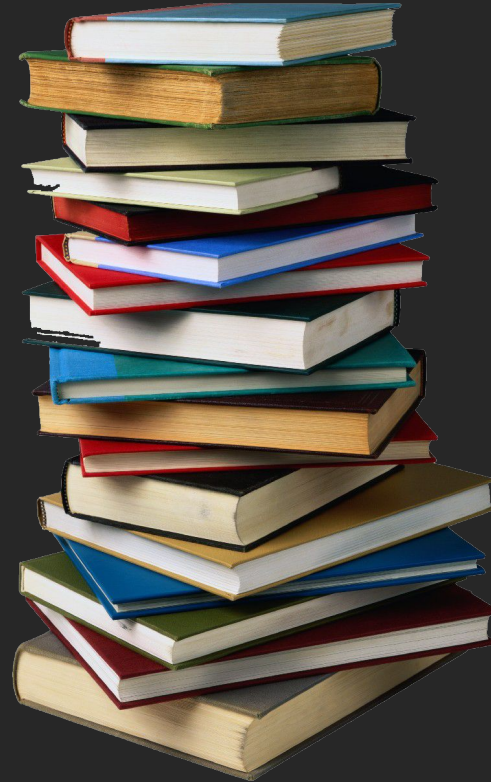
K. N. Toosi  
University of Technology



<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/>

# The stack



# Implementing the stack

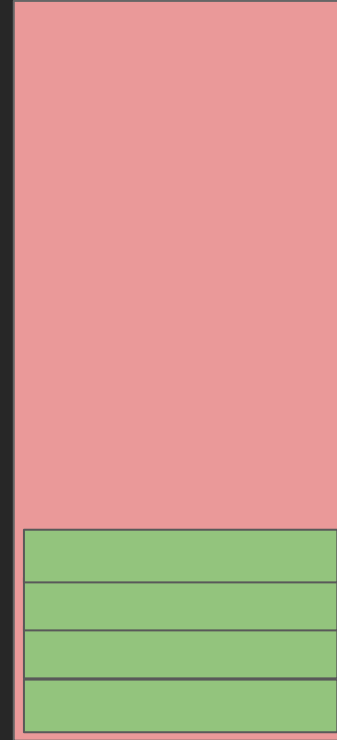


K. N. Toosi  
University of Technology

Stack Segment



Stack Segment



# Implementing the stack

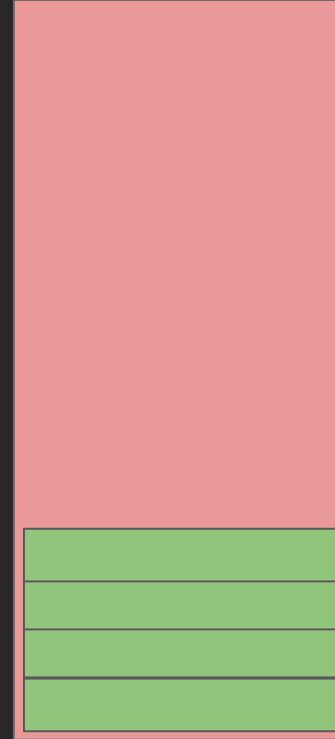


K. N. Toosi  
University of Technology

Stack Segment



Stack Segment



x86



# Implementing the stack

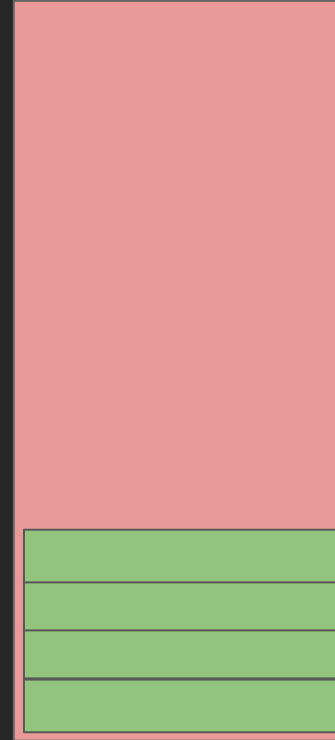


K. N. Toosi  
University of Technology

Stack Segment



Stack Segment

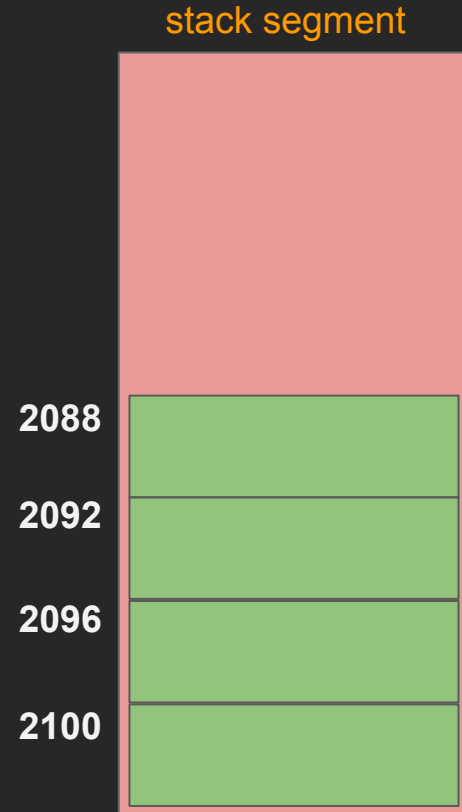


x86  
(why?)

# Implementing the stack



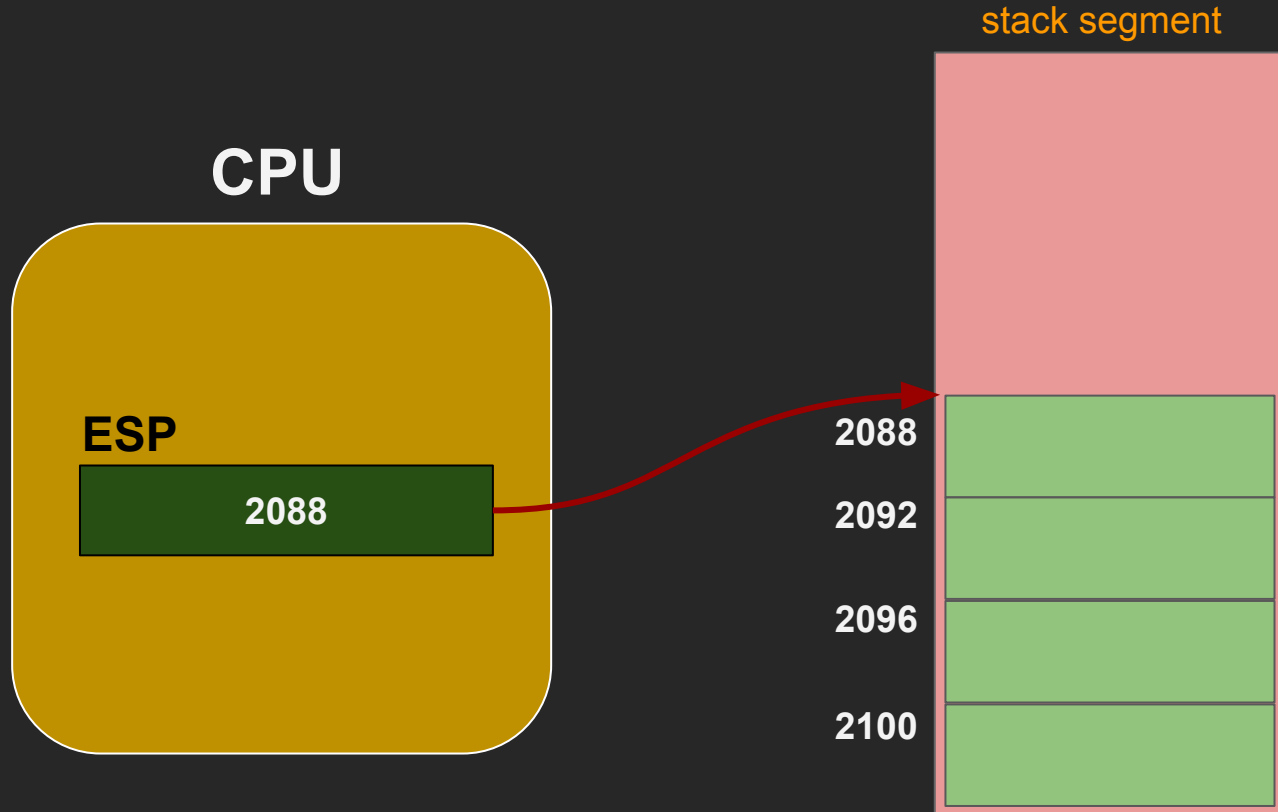
K. N. Toosi  
University of Technology



# Stack Pointer (SP, ESP, RSP)



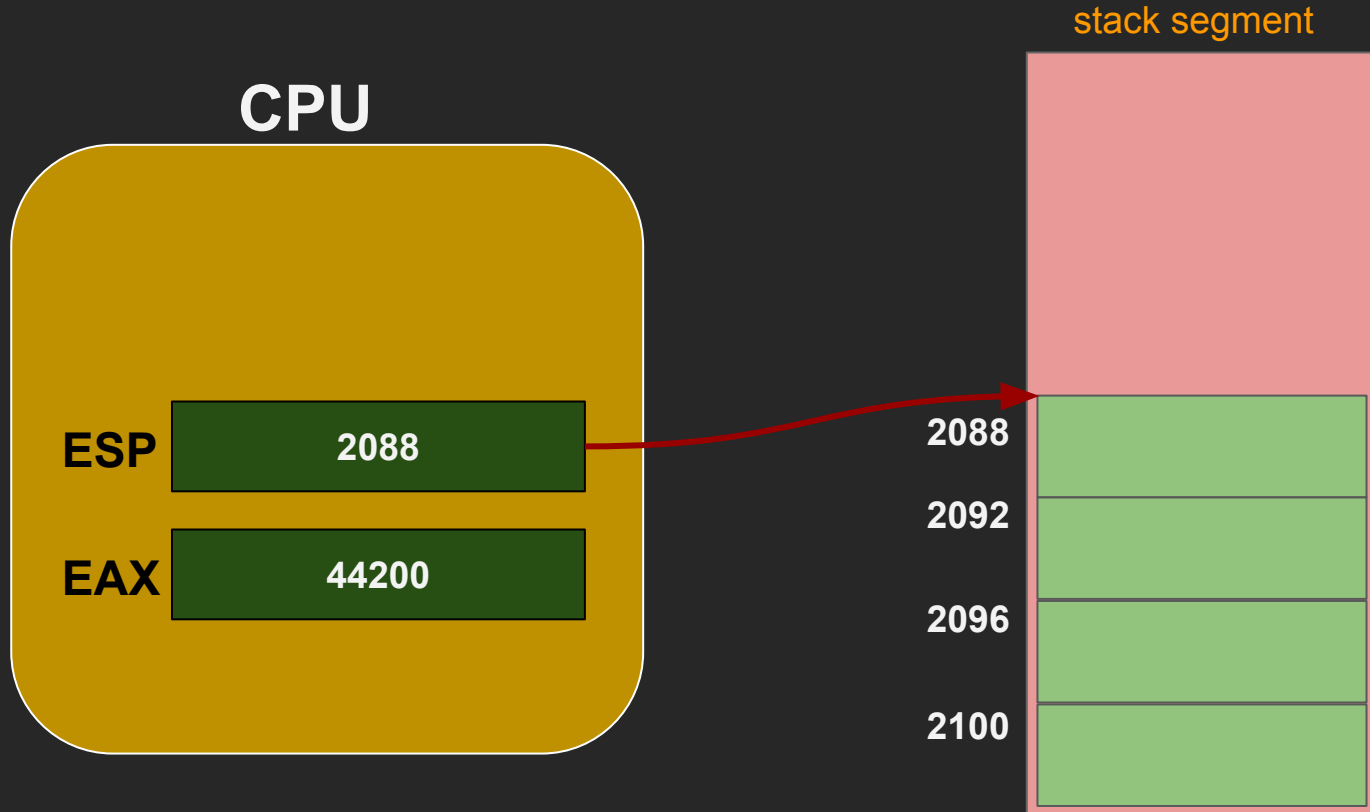
K. N. Toosi  
University of Technology



# Pushing on the stack



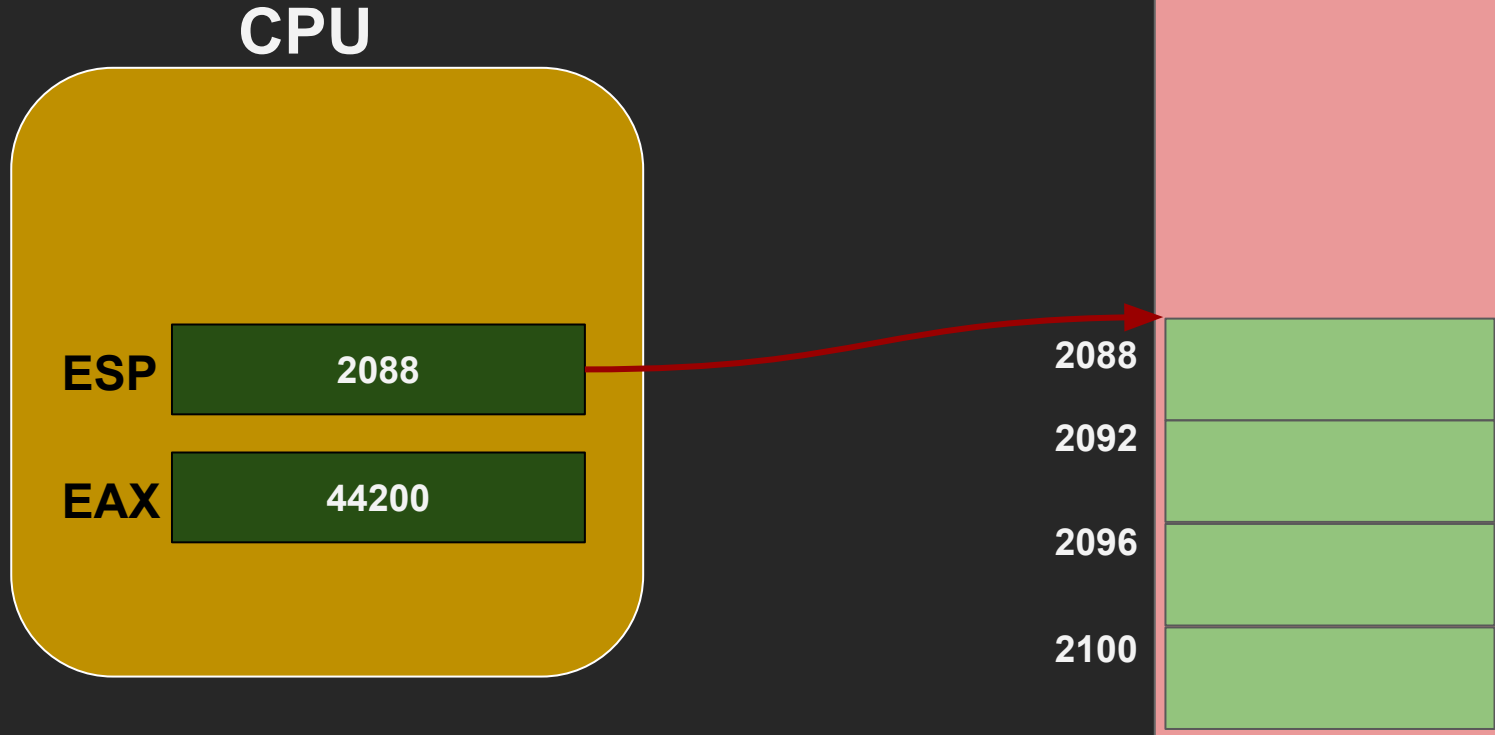
K. N. Toosi  
University of Technology



# Push EAX on the stack



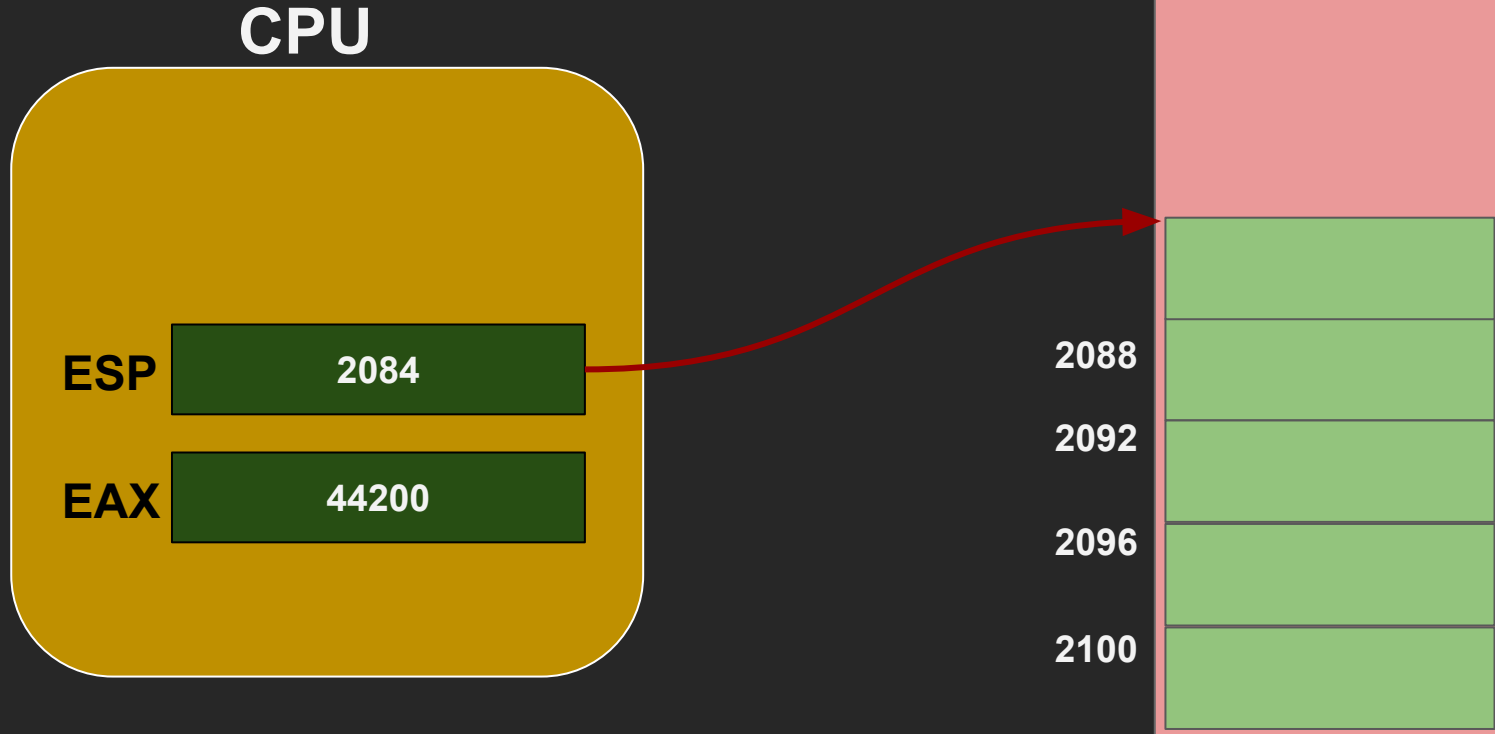
K. N. Toosi  
University of Technology



# Push EAX on the stack



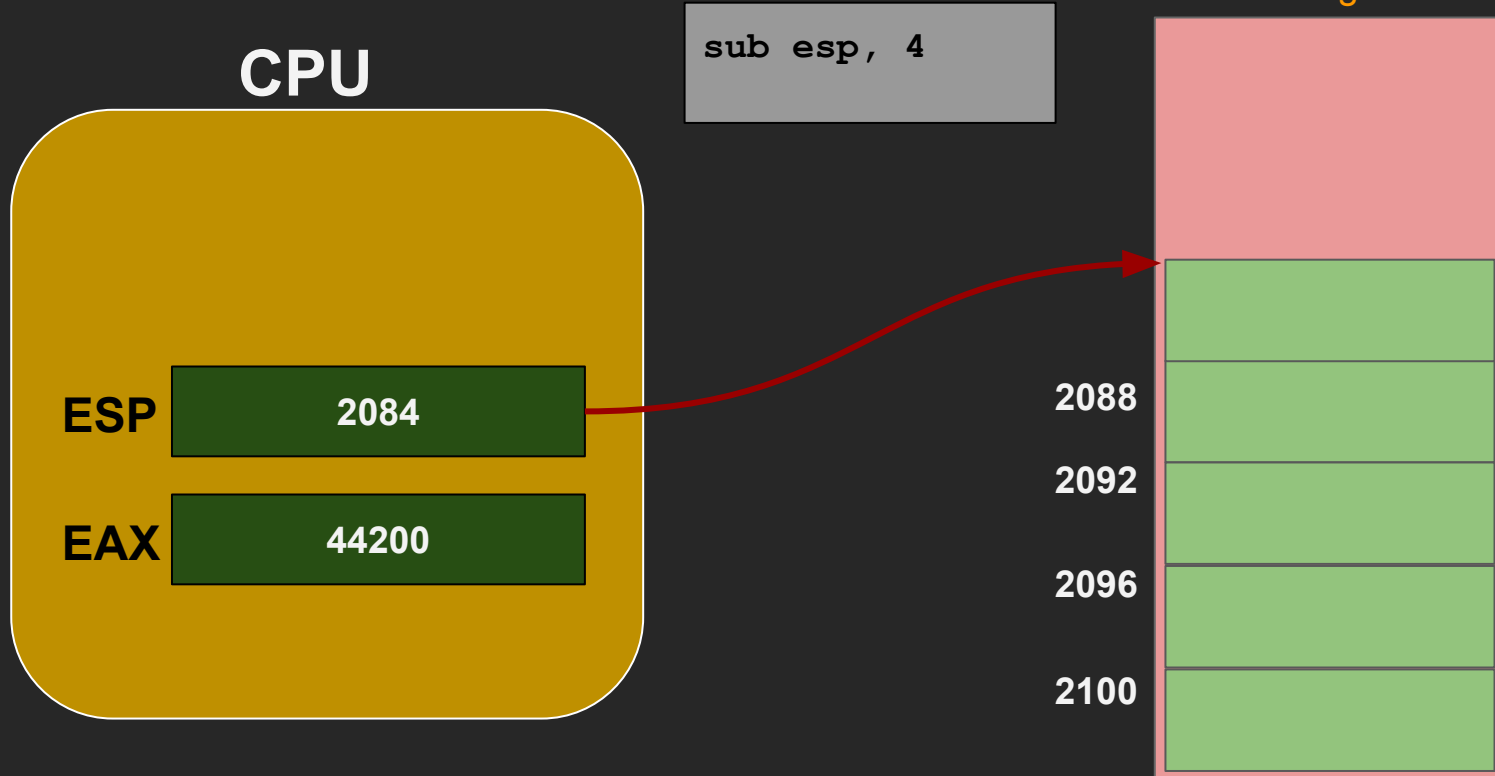
K. N. Toosi  
University of Technology



# Push EAX on the stack



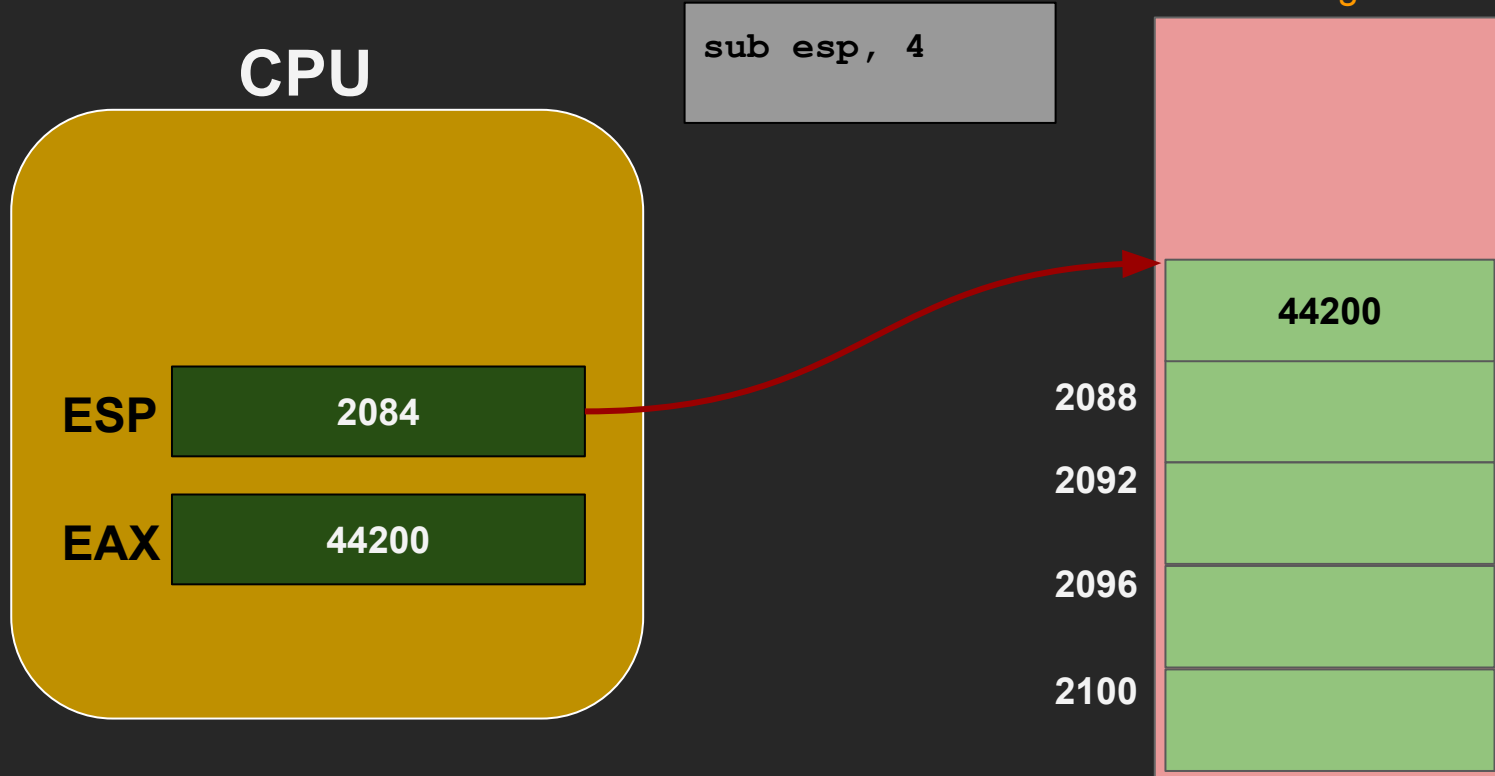
K. N. Toosi  
University of Technology



# Push EAX on the stack



K. N. Toosi  
University of Technology





# Push EAX on the stack

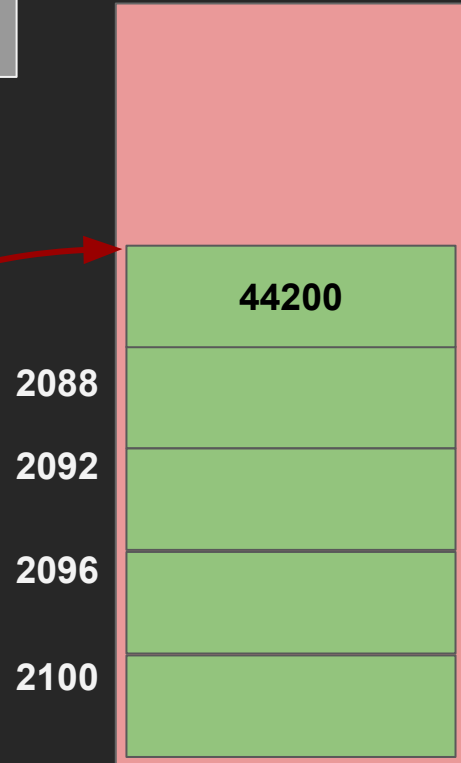


```
sub esp, 4  
mov [esp], eax
```

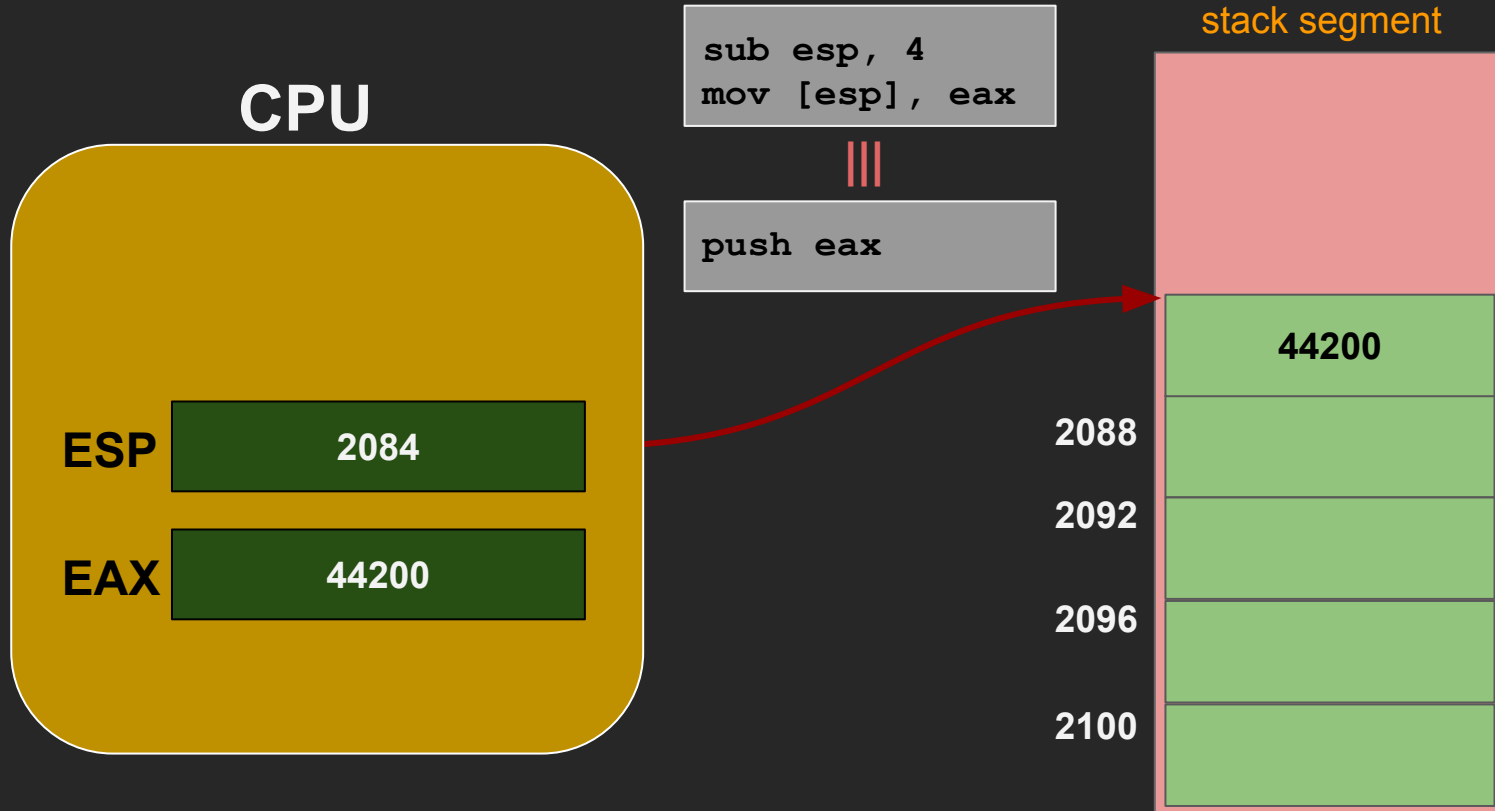
CPU



stack segment



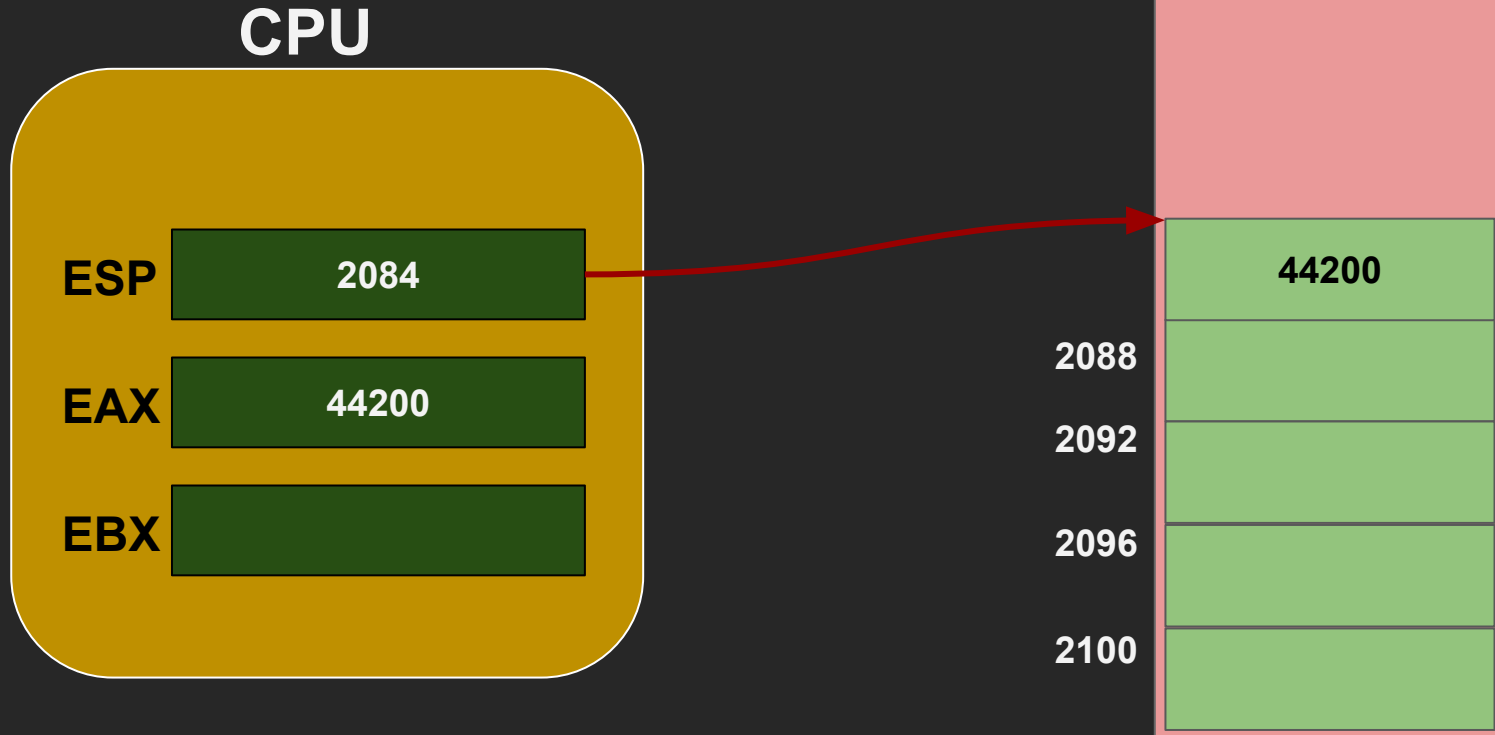
# Push EAX on the stack



# Pop into EBX



K. N. Toosi  
University of Technology

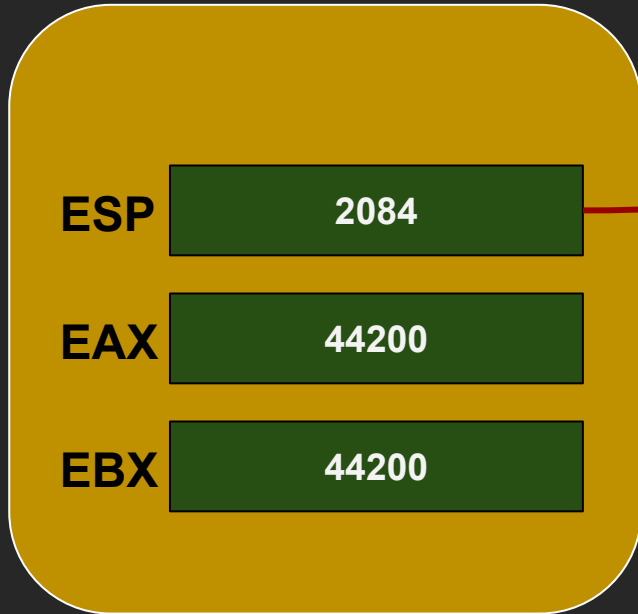


# Pop into EBX

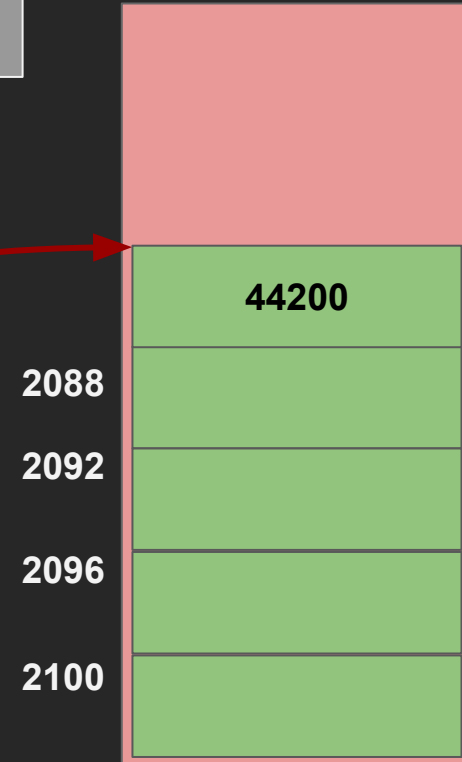


```
mov ebx, [esp]
```

CPU



stack segment

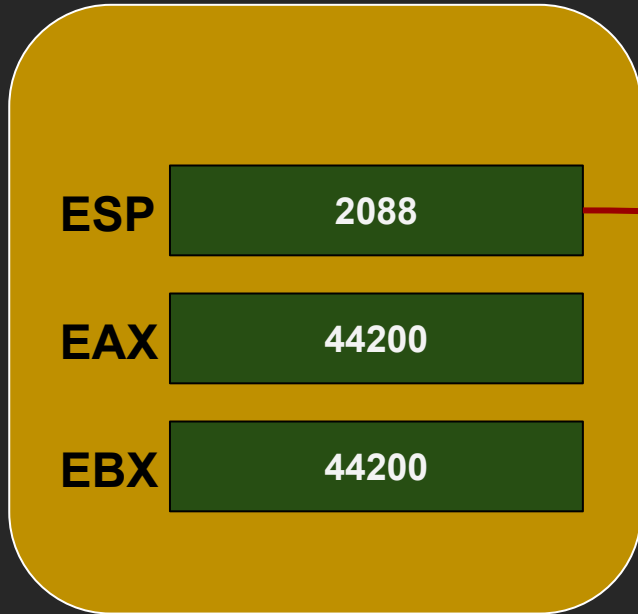


# Pop into EBX

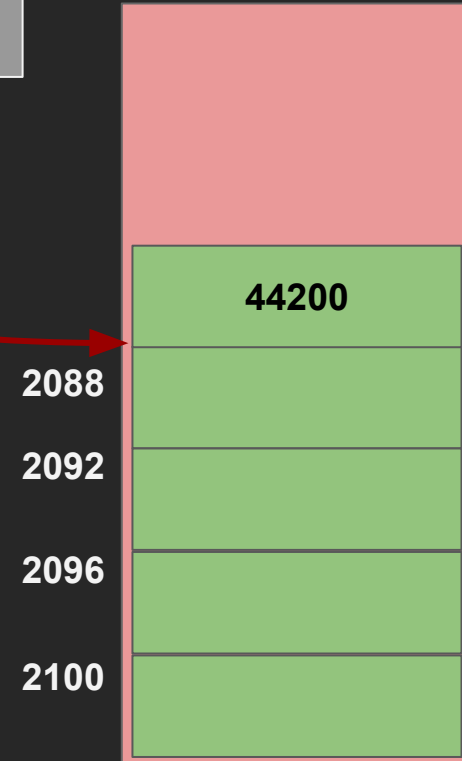


```
mov ebx, [esp]  
add esp, 4
```

## CPU



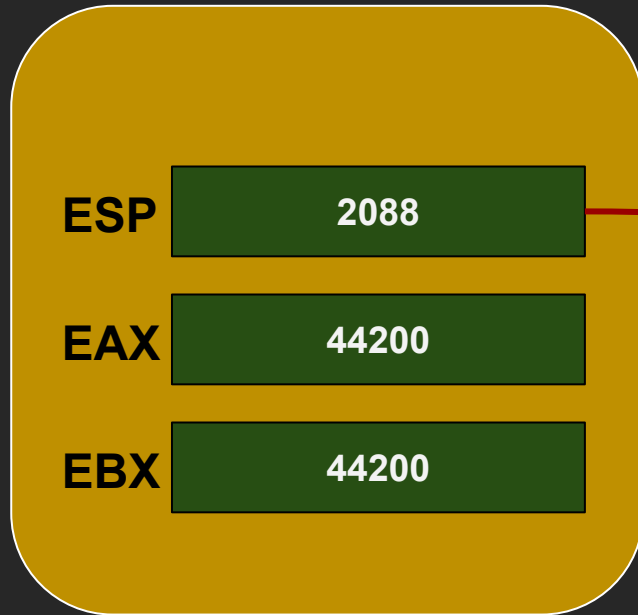
## stack segment



# Pop into EBX



## CPU

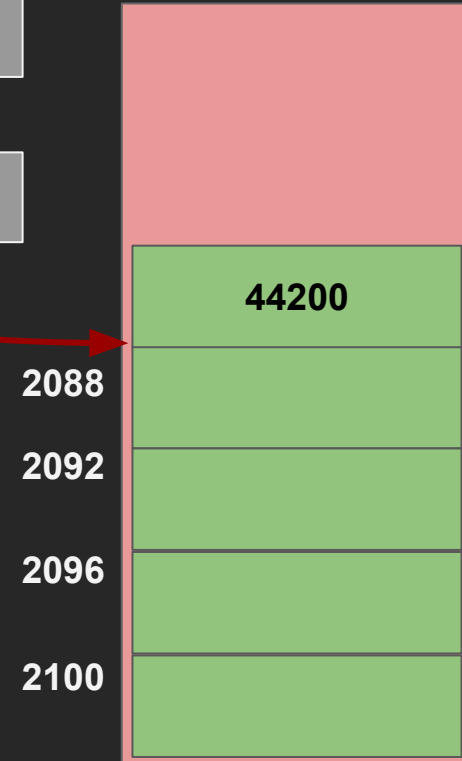


```
mov ebx, [esp]  
add esp, 4
```



```
pop ebx
```

## stack segment

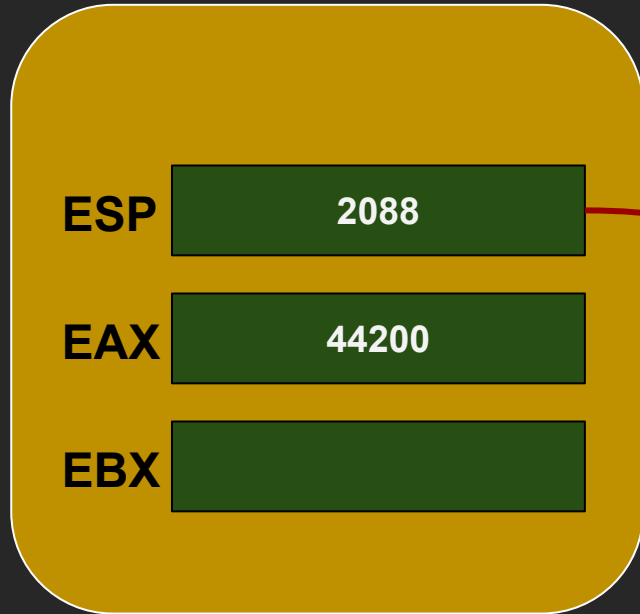


# just pop 4 bytes (store nowhere)

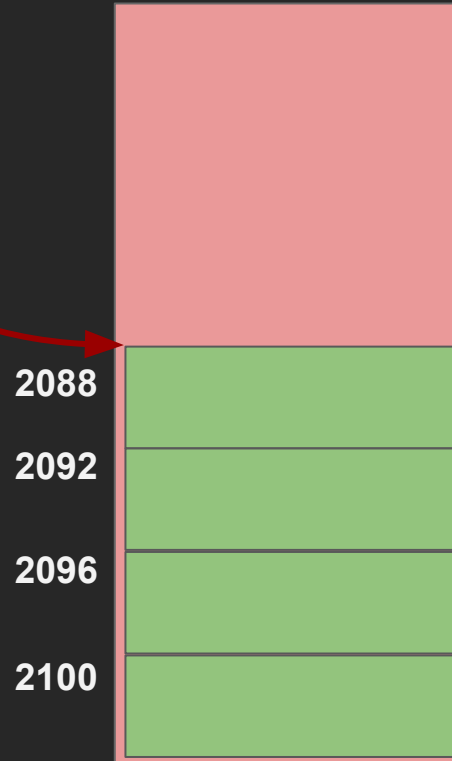


K. N. Toosi  
University of Technology

## CPU



stack segment

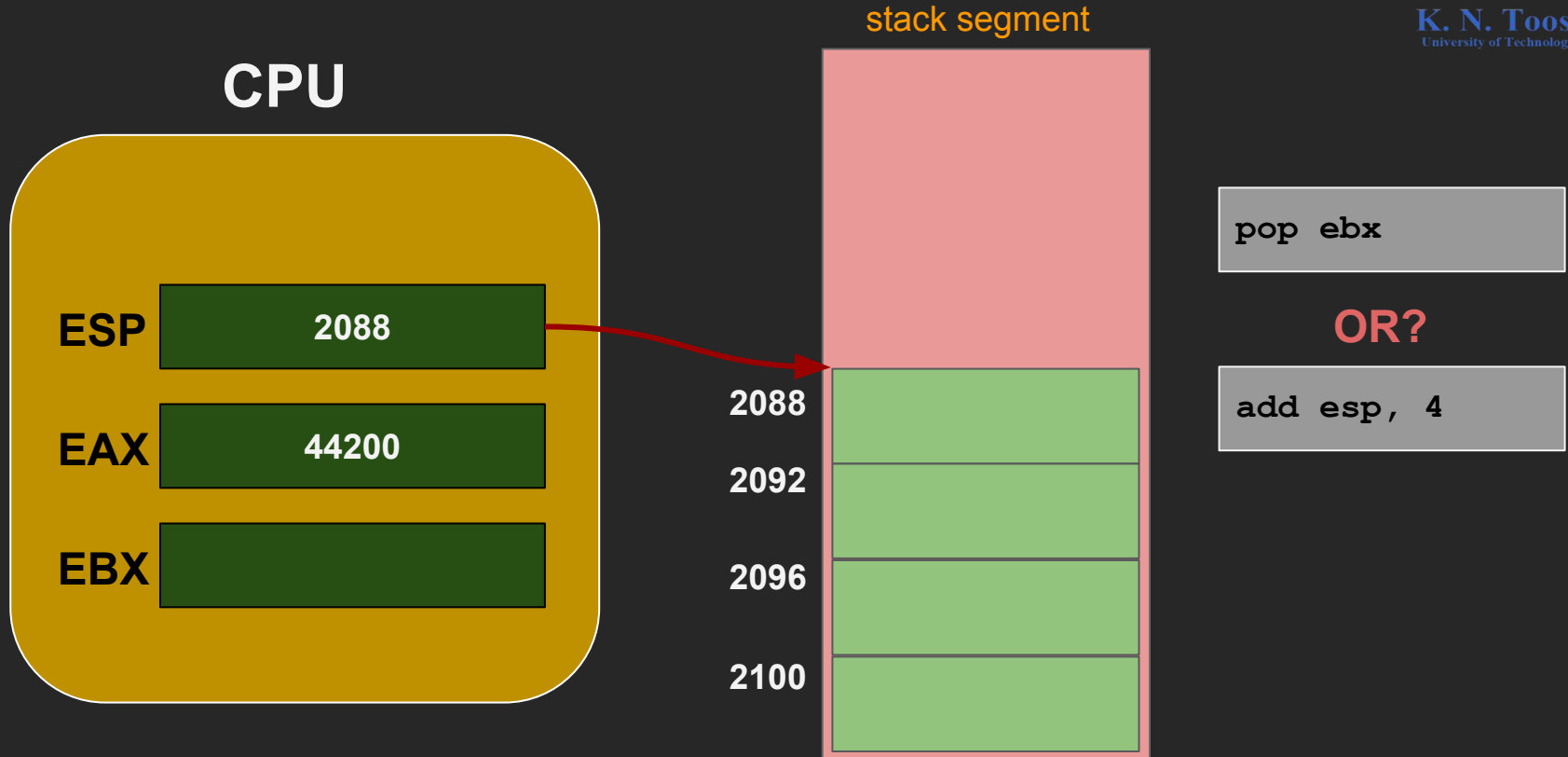


```
pop ebx
```

# just pop 4 bytes (store nowhere)



K. N. Toosi  
University of Technology

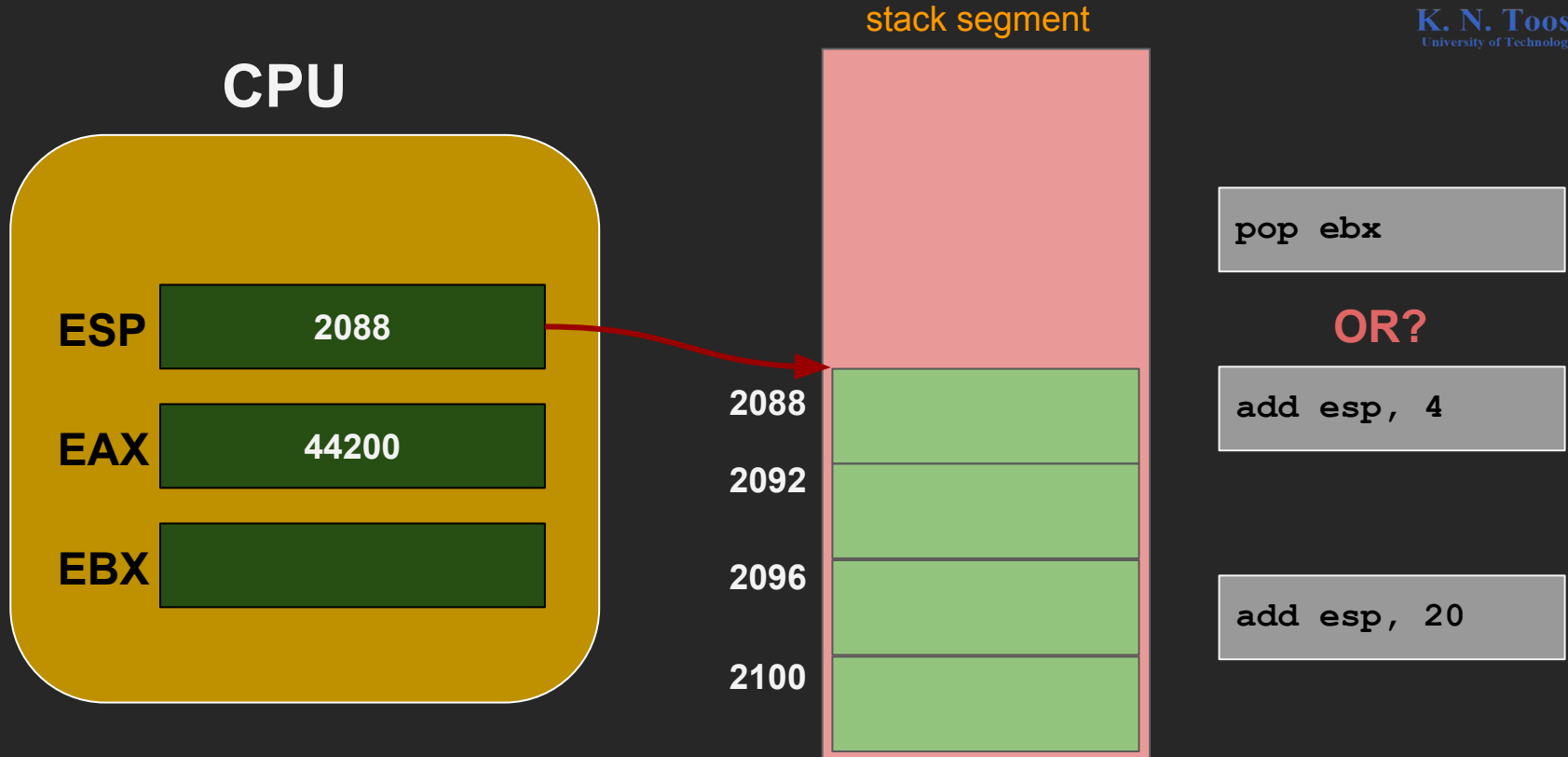




# just pop 4 bytes (store nowhere)



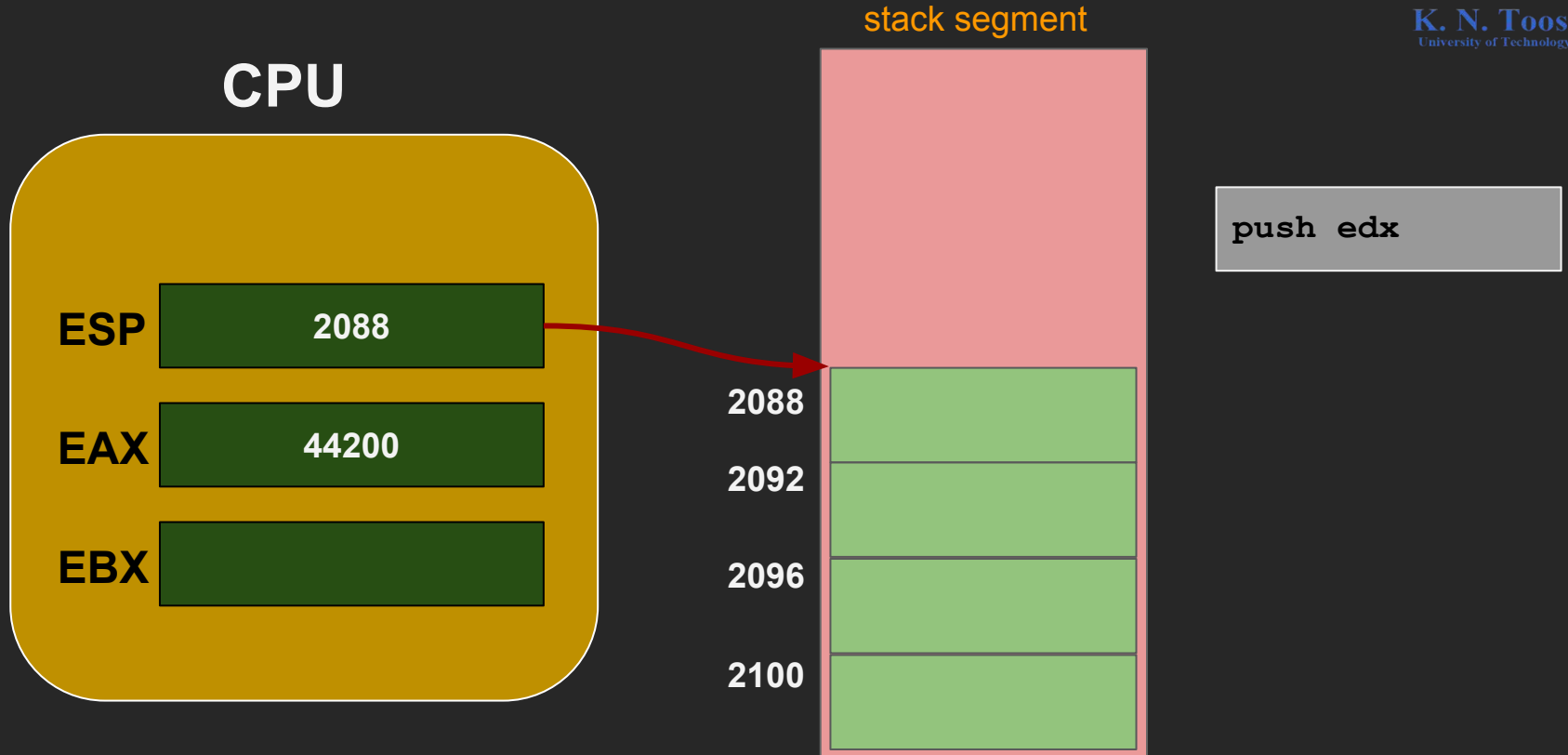
K. N. Toosi  
University of Technology



# reserve memory on stack



K. N. Toosi  
University of Technology

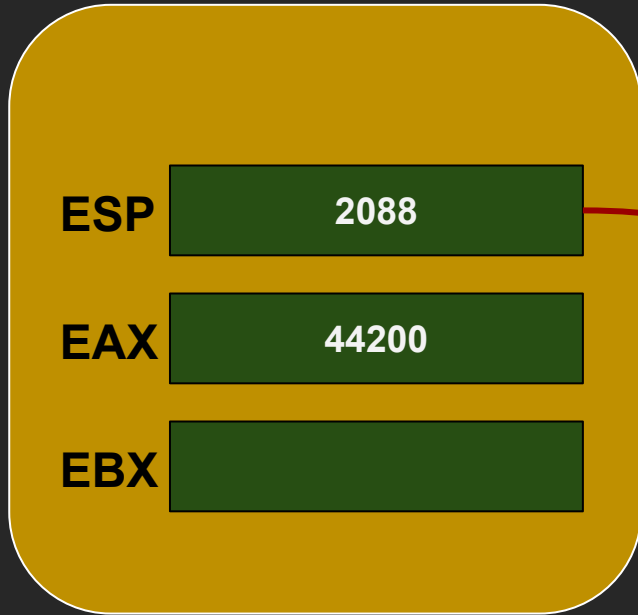


# reserve memory on stack

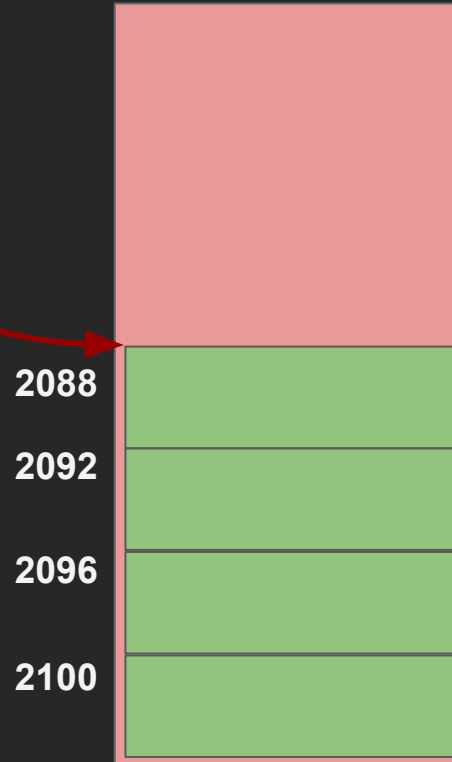


K. N. Toosi  
University of Technology

## CPU



## stack segment



```
push edx
```

**OR?**

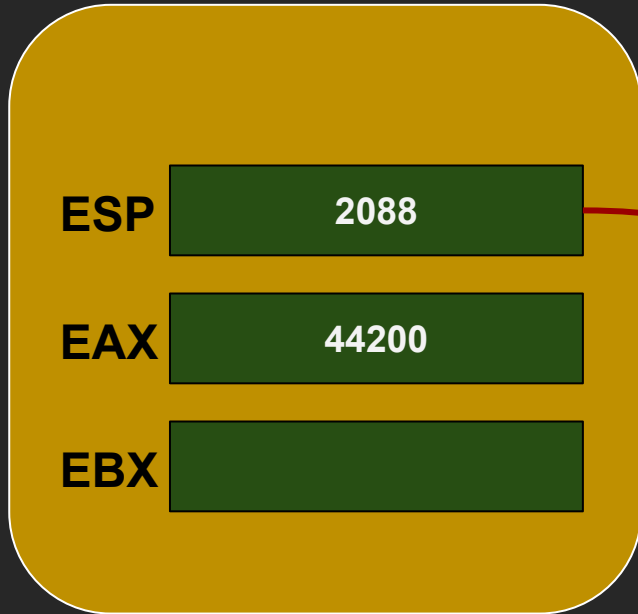
```
sub esp, 4
```

# reserve memory on stack

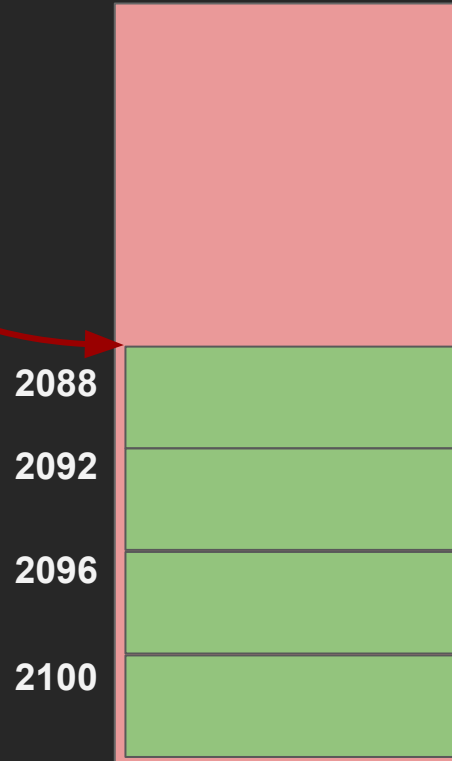


K. N. Toosi  
University of Technology

CPU



stack segment



```
push edx
```

OR?

```
sub esp, 4
```

```
sub esp, 24
```

# Push and Pop

Push reg/mem/immed

Pop reg/mem



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

# 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



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

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



# Back to subroutines



```
segment .data simplefunc3.asm
msg:  db "Salaaaaam!", 10, 0
segment .text
    :
    mov edx, I1
    jmp print_salam
I1:
    mov edx, I2
    jmp print_salam
I2:
    :
print_salam:
    mov eax, msg
    call print_string
    jmp edx
```

# Back to subroutines



**segment .data** simplefunc3.asm

**msg:** db "Salaaaaam!", 10, 0

**segment .text**

⋮

mov **edx**, I1

jmp print\_salam

**I1:**

mov **edx**, I2

jmp print\_salam

**I2:**

⋮

**print\_salam:**

mov **eax**, msg

call print\_string

jmp **edx**

**segment .data** simplefunc4.asm

**msg:** db "Salaaaaam!", 10, 0

**segment .text**

⋮

push I1

jmp print\_salam

**I1:**

push I2

jmp print\_salam

**I2:**

⋮

**print\_salam:**

mov **eax**, msg

call print\_string

??

# Back to subroutines



**segment .data** simplefunc3.asm

**msg:** db "Salaaaaam!", 10, 0

**segment .text**

⋮

mov **edx**, I1

jmp print\_salam

**I1:**

mov **edx**, I2

jmp print\_salam

**I2:**

⋮

**print\_salam:**

mov **eax**, msg

call print\_string

jmp **edx**

**segment .data** simplefunc4.asm

**msg:** db "Salaaaaam!", 10, 0

**segment .text**

⋮

push I1

jmp print\_salam

**I1:**

push I2

jmp print\_salam

**I2:**

⋮

**print\_salam:**

mov **eax**, msg

call print\_string

pop **edx**

jmp **edx**

# the CALL instruction



**segment .data** simplefunc3.asm

**msg:** db "Salaaaaam!", 10, 0

**segment .text**

⋮

mov **edx**, I1

jmp print\_salam

**I1:**

mov **edx**, I2

jmp print\_salam

**I2:**

⋮

**print\_salam:**

mov **eax**, msg

call print\_string

jmp **edx**

**segment .data** simplefunc4.asm

**msg:** db "Salaaaaam!", 10, 0

**segment .text**

⋮

push I1

jmp print\_salam

**I1:**

push I2

jmp print\_salam

**I2:**

⋮

**print\_salam:**

mov **eax**, msg

call print\_string

pop **edx**

jmp **edx**

**segment .data** simplefunc5.asm

**msg:** db "Salaaaaam!", 10, 0

**segment .text**

⋮

call print\_salam

**I1:**

call print\_salam

**I2:**

⋮

**print\_salam:**

mov **eax**, msg

call print\_string

pop **edx**

jmp **edx**

# the CALL instruction



```
segment .data simplefunc3.asm
msg: db "Salaaaaam!", 10, 0
segment .text
:
mov edx, I1
jmp print_salam
I1:
mov edx, I2
jmp print_salam
I2:
:
print_salam:
mov eax, msg
call print_string
jmp edx
```

```
segment .data simplefunc4.asm
msg: db "Salaaaaam!", 10, 0
segment .text
:
push I1
jmp print_salam
I1:
push I2
jmp print_salam
I2:
:
print_salam:
mov eax, msg
call print_string
pop edx
jmp edx
```

```
segment .data simplefunc5.asm
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!



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



# the CALL instruction

CALL is merely a form of jump!

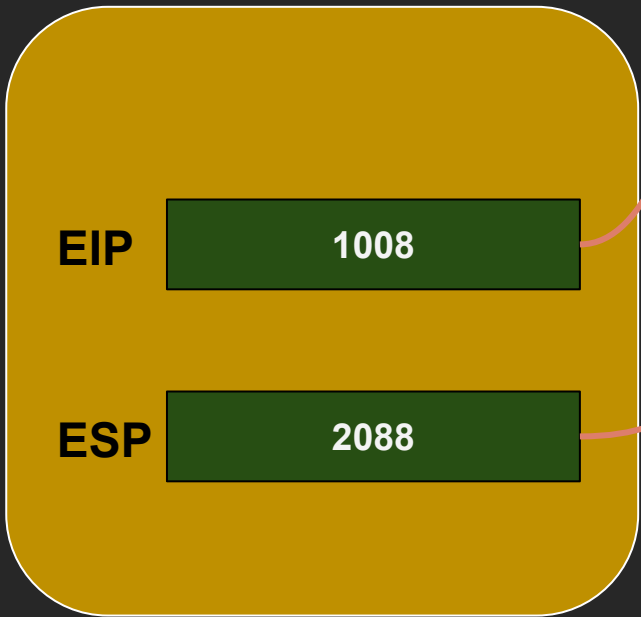
```
call label1
```

- Push return address on stack
- jump to label1



call print\_salam

# CPU

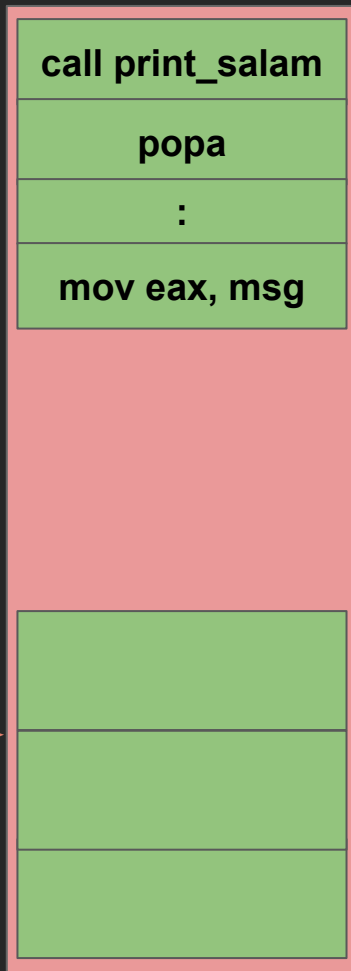


print\_salam = 1024

1008

2088

2092



```
segment .data
msg:  db "Salaaaaam!", 10, 0
segment .text
:
→ call print_salam
popa
:

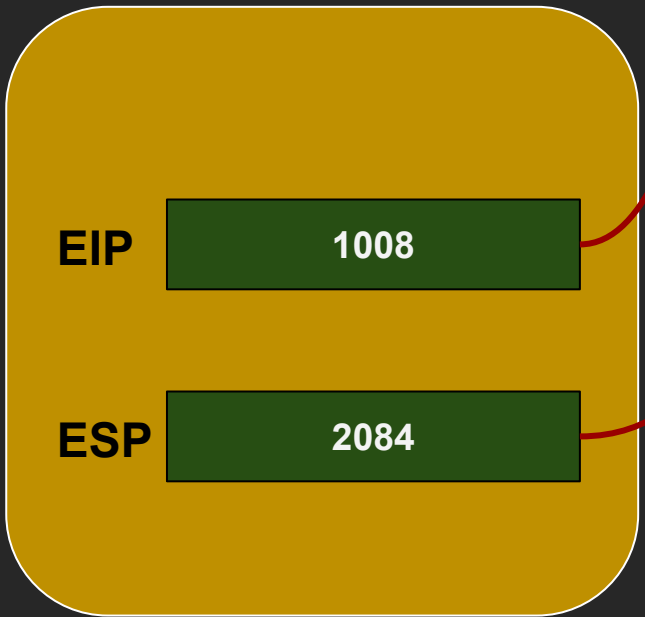
print_salam:
mov eax, msg
call print_string
pop edx
jmp edx
```





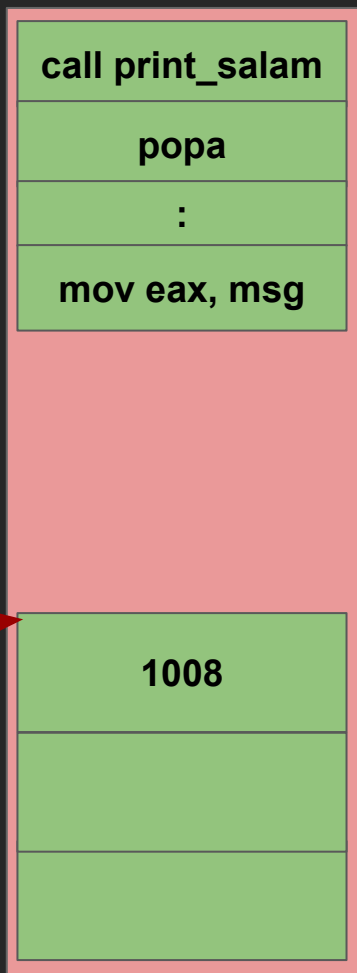
call print\_salam

# CPU



print\_salam = 1024

1008



```
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



```
segment .data simplefunc5.asm
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
```

```
segment .data simplefunc6.asm
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



```
segment .data simplefunc5.asm
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
```

```
segment .data simplefunc6.asm
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!



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

# 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

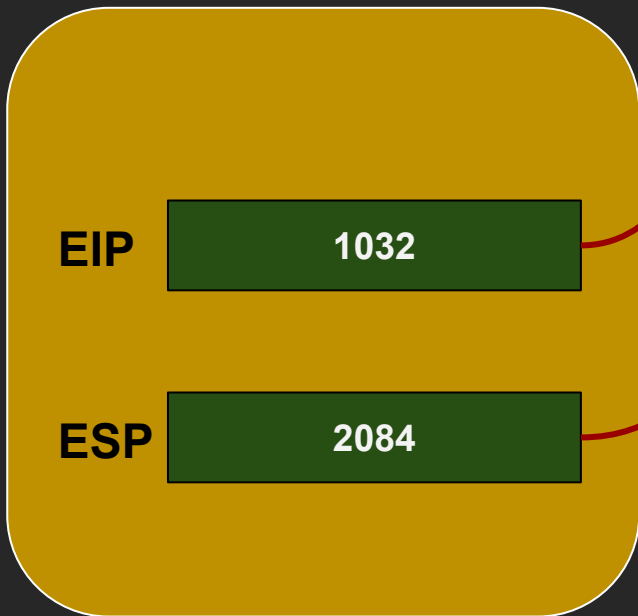


ret

1008

print\_salam = 1024

CPU



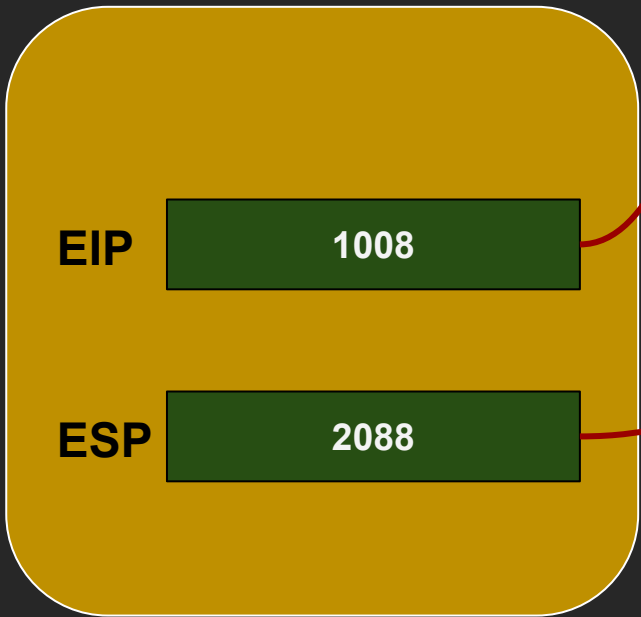
```
segment .data
msg: db "Salaaaaam!", 10, 0
segment .text
:
call print_salam
popa
:
print_salam:
mov eax, msg
call print_string
→ ret
```



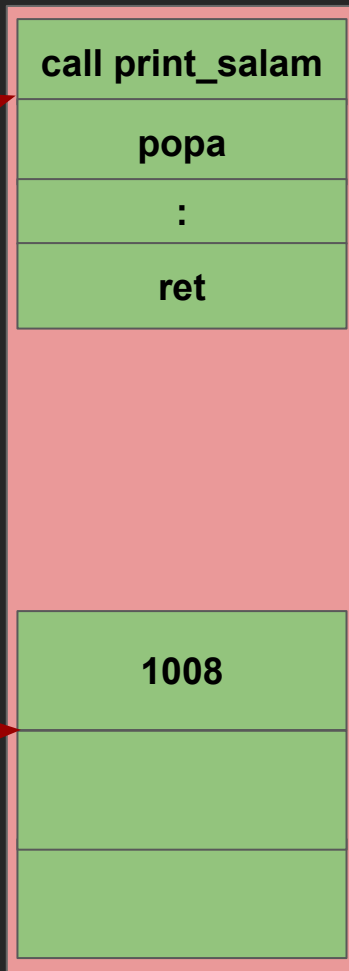


ret

CPU



print\_salam = 1024



```
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

