

# Introduction to 8086 Assembly

## Lecture 10

Passing Arguments, Calling Conventions, Local Variables

# Passing parameters

Write a subprogram that takes  $m, n$  as arguments and returns  $m^n$



K. N. Toosi  
University of Technology

powfunc.c

```
int pow(int m, int n) {  
    int p = 1;  
    while (n > 0) {  
        p *= m;  
        n--;  
    }  
  
    return p;  
}
```

# Passing parameters



K. N. Toosi  
University of Technology

Write a subprogram that takes  $m, n$  as arguments and returns  $m^n$

1. Use registers

```
powfunc.c  
  
int pow(int m, int n) {  
    int p = 1;  
    while (n > 0) {  
        p *= m;  
        n--;  
    }  
  
    return p;  
}
```

# Passing parameters

Write a subprogram taking  $m, n$  as arguments, returning  $m^n$

```
int caller_func() {  
    pow(3,4);  
}  
  
int pow(int m, int n) {  
    int p = 1;  
    while (n > 0) {  
        p *= m;  
        n--;  
    }  
    return p;  
}
```

powfunc.c

```
mov ebx, 3  
mov eax, 4  
  
call pow
```

powfunc1.asm

```
call print_int  
call print_nl  
  
:
```

```
pow:  
    mov ecx, eax  
    mov eax, 1  
loop1:  
    imul ebx  
  
loop loop1  
  
ret
```



# Passing parameters

Write a subprogram taking  $m, n$  as arguments, returning  $m^n$

```
int caller_func() {  
    pow(3,4);  
}  
  
int pow(int m, int n) {  
    int p = 1;  
    while (n > 0) {  
        p *= m;  
        n--;  
    }  
    return p;  
}
```

```
mov ebx, 3  
mov eax, 4
```

```
call pow
```

```
call print_int  
call print_nl
```

```
⋮
```

```
pow:
```

```
mov ecx, eax  
mov eax, 1
```

```
loop1:
```

```
imul ebx
```

```
loop loop1
```

```
ret
```

powfunc1.asm

return value in eax

return value in eax



# Passing parameters

Write a subprogram taking  $m, n$  as arguments, returning  $m^n$

```
int caller_func() {  
    pow(3,4);  
}  
  
int pow(int m, int n) {  
    int p = 1;  
    while (n > 0) {  
        p *= m;  
        n--;  
    }  
    return p;  
}
```

```
powfunc1.asm  
mov ebx, 3  
mov eax, 4  
  
call pow  
  
call print_int  
call print_nl  
  
:  
pow:  
    mov ecx, eax  
    mov eax, 1  
loop1:  
    imul ebx  
  
loop loop1  
  
ret
```

return value in eax

what registers get changed?



# Passing parameters

Write a subprogram taking  $m, n$  as arguments, returning  $m^n$

```
int caller_func() {  
    pow(3,4);  
}  
  
int pow(int m, int n) {  
    int p = 1;  
    while (n > 0) {  
        p *= m;  
        n--;  
    }  
    return p;  
}
```

```
mov ebx, 3  
mov eax, 4
```

```
call pow
```

```
call print_int  
call print_nl
```

```
⋮
```

```
pow:
```

```
mov ecx, eax  
mov eax, 1
```

```
loop1:
```

```
imul ebx
```

```
loop loop1
```

```
ret
```

powfunc1.asm

return value in eax

what registers  
get changed?

EAX, ECX, EDX



# Passing parameters

Write a subprogram taking  $m, n$  as arguments, returning  $m^n$

```
int caller_func() {
    pow(3,4);
}

int pow(int m, int n) {
    int p = 1;
    while (n > 0) {
        p *= m;
        n--;
    }
    return p;
}
```

```
mov ebx, 3
mov eax, 4
```

```
call pow
```

```
call print_int
call print_nl
```

```
:
```

```
pow:
```

```
mov ecx, eax
mov eax, 1
```

```
loop1:
```

```
imul ebx
```

```
loop loop1
```

```
ret
```

powfunc1.asm

return value in eax

what registers  
get changed?

EAX, ECX, EDX





# Passing parameters

Write a subprogram taking  $m, n$  as arguments, returning  $m^n$

```
int caller_func() {  
    pow(3,4);  
}  
  
int pow(int m, int n) {  
    int p = 1;  
    while (n > 0) {  
        p *= m;  
        n--;  
    }  
    return p;  
}
```

powfunc.c

```
mov ebx, 3  
mov eax, 4  
  
call pow  
  
call print_int  
call print_nl  
:
```

powfunc2.asm

```
pow:  
    push ecx  
    push edx  
  
    mov ecx, eax  
    mov eax, 1  
loop1:  
    imul ebx  
    loop loop1  
  
    pop edx  
    pop ecx  
  
ret
```





# Passing parameters

Write a subprogram taking  $m, n$  as arguments, returning  $m^n$

```
int caller_func() {  
    pow(3,4);  
}  
  
int pow(int m, int n) {  
    int p = 1;  
    while (n > 0) {  
        p *= m;  
        n--;  
    }  
    return p;  
}
```

powfunc.c

```
mov ebx, 3  
mov eax, 4  
  
call pow  
  
call print_int  
call print_nl  
:
```

powfunc2.asm

```
pow:  
    push ecx  
    push edx  
  
    mov ecx, eax  
    mov eax, 1  
loop1:  
    imul ebx  
    loop loop1  
  
    pop edx  
    pop ecx  
  
ret
```

Is this really  
necessary?

# passing parameters

1. User registers
2. ?



# passing parameters

1. User registers
2. Use Stack





# Passing parameters on stack

Write a subprogram taking  $m, n$  as arguments, returning  $m^n$

```
int caller_func() {  
    pow(3,4);  
}  
  
int pow(int m, int n) {  
    int p = 1;  
    while (n > 0) {  
        p *= m;  
        n--;  
    }  
    return p;  
}
```

powfunc.c

```
;; pow(3,4)  
push 4 ; push n  
push 3 ; push m
```

powfunc2.asm

```
call pow  
add esp, 8
```

```
call print_int  
call print_nl  
:
```

```
pow:  
    mov ecx, [esp+8]  
    mov eax, 1  
loop1:  
    imul dword [esp+4]  
  
loop loop1  
ret
```



# Passing parameters on stack

Write a subprogram taking  $m, n$  as arguments, returning  $m^n$

```
int caller_func() {  
    pow(3,4);  
}  
  
int pow(int m, int n) {  
    int p = 1;  
    while (n > 0) {  
        p *= m;  
        n--;  
    }  
    return p;  
}
```

```
;; pow(3,4)  
push 4 ; push n  
push 3 ; push m
```

```
call pow  
add esp, 8
```

```
call print_int  
call print_nl  
:
```

```
pow:  
mov ecx, [esp+8]  
mov eax, 1
```

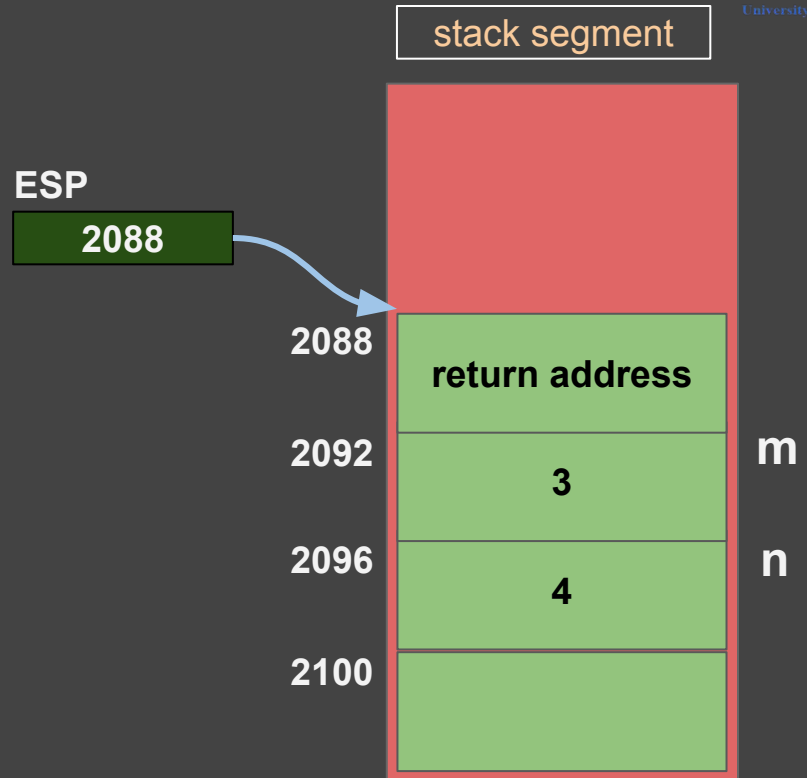
```
loop1:  
imul dword [esp+4]
```

```
loop loop1  
ret
```

accessing  
parameters via  
ESP



```
powfunc3.asm  
  
;; pow(3,4)  
push 4 ; push n  
push 3 ; push m  
  
call pow  
add esp, 8  
  
call print_int  
call print_nl  
:  
  
pow: ;; pow(m,n)  
mov ecx, [esp+8] ;; ecx = n  
mov eax, 1  
loop1:  
imul dword [esp+4] ;; eax *= m  
  
loop loop1  
ret
```





```
;; pow(3,4)
push 4 ; push n
push 3 ; push m
```

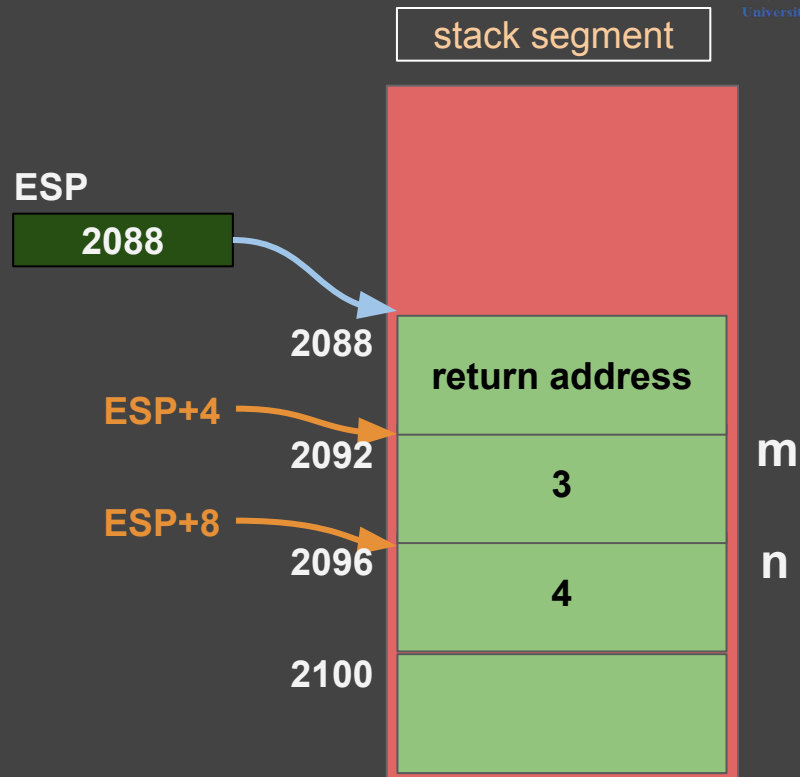
powfunc3.asm

```
call pow
add esp, 8
```

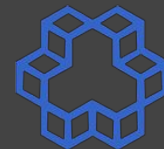
```
call print_int
call print_nl
:
```

```
pow: ;; pow(m,n)
mov ecx, [esp+8] ;; ecx = n
mov eax, 1
loop1:
imul dword [esp+4] ;; eax *= m

loop loop1
ret
```

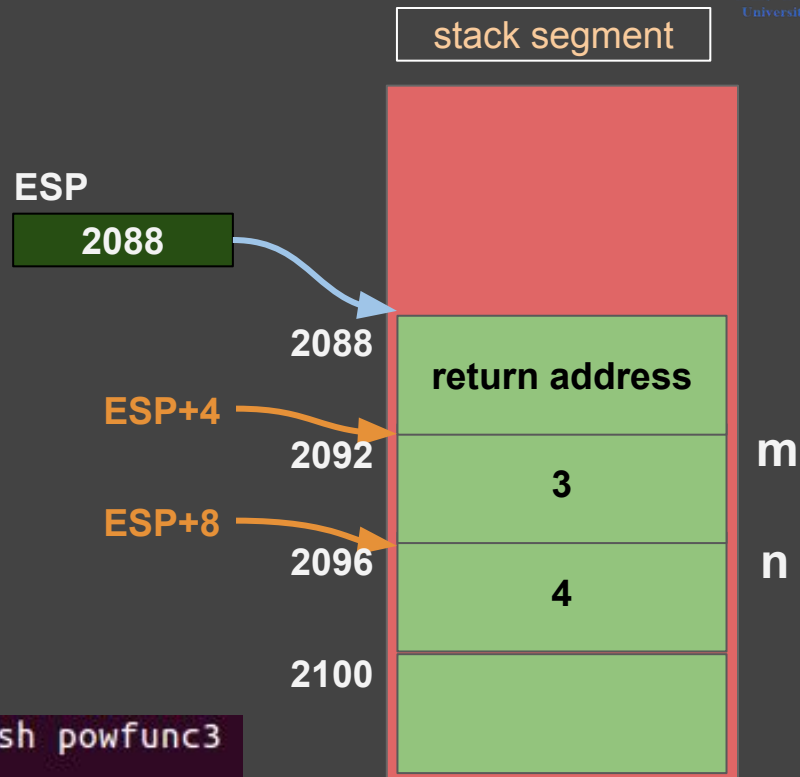






```
powfunc3.asm  
;; pow(3,4)  
push 4 ; push n  
push 3 ; push m  
  
call pow  
add esp, 8  
  
call print_int  
call print_nl  
:  
  
pow: ;; pow(m,n)  
mov ecx, [esp+8] ;; ecx = n  
mov eax, 1  
loop1:  
imul dword [esp+4] ;; eax *= m  
  
loop loop1  
ret
```

```
b.nasihatkon@kntu:lecture10$ ./run.sh powfunc3  
81
```



# Accessing parameters via ESP



```
powfunc3.asm
;; pow(3,4)
push 4 ; push n
push 3 ; push m

call pow
add esp, 8

call print_int
call print_nl
:
pow: ;; pow(m,n)

mov ecx, [esp+8] ;; ecx = n
mov eax, 1
loop1:
imul dword [esp+4] ;; eax *= m
loop loop1

ret
```

```
powfunc4.asm
;; pow(3,4)
push 4 ; push n
push 3 ; push m

call pow
add esp, 8

call print_int
call print_nl
:
pow: ;; pow(m,n)
push ecx
push edx
mov ecx, [esp+8] ;; ecx = n
mov eax, 1
loop1:
imul dword [esp+4] ;; eax *= m
loop loop1
pop edx
pop ecx
ret
```

# Accessing parameters via ESP



K. N. Toosi  
University of Technology

```
;; pow(3,4) powfunc3.asm
push 4 ; push n
push 3 ; push m

call pow
add esp, 8

call print_int
call print_nl
:
pow: ;; pow(m,n)

mov ecx, [esp+8] ;; ecx = n
mov eax, 1
loop1:
imul dword [esp+4] ;; eax *= m
loop loop1

ret
```

```
;; pow(3,4) powfunc4.asm
push 4 ; push n
push 3 ; push m

call pow
add esp, 8

call print_int
call print_nl
:
pow: ;; pow(m,n)
push ecx
push edx
mov ecx, [esp+8] ;; ecx = n
mov eax, 1
loop1:
imul dword [esp+4] ;; eax *= m
loop loop1
pop edx
pop ecx
ret
```

```
b.nasihatkon@kntu:lecture10$ ./run.sh powfunc4
```

# Accessing parameters via ESP



K. N. Toosi  
University of Technology

```
powfunc3.asm
;; pow(3,4)
push 4 ; push n
push 3 ; push m

call pow
add esp, 8

call print_int
call print_nl
:
pow: ;; pow(m,n)

mov ecx, [esp+8] ;; ecx = n
mov eax, 1
loop1:
imul dword [esp+4] ;; eax *= m
loop loop1

ret
```

```
powfunc4.asm
;; pow(3,4)
push 4 ; push n
push 3 ; push m

call pow
add esp, 8

call print_int
call print_nl
:
pow: ;; pow(m,n)
push ecx
push edx
mov ecx, [esp+8] ;; ecx = n
mov eax, 1
loop1:
imul dword [esp+4] ;; eax *= m
loop loop1
pop edx
pop ecx
ret
```

```
b.nasihatkon@kntu:lecture10$ ./run.sh powfunc4
```

0

what's wrong?

# Accessing parameters via ESP



```
powfunc4.asm
;; pow(3,4)
push 4    ; push n
push 3    ; push m

call pow
add esp, 8

call print_int
call print_nl
:
pow:    ;; pow(m,n)
push ecx
push edx
mov ecx, [esp+8]    ;; ecx = n
mov eax, 1
loop1:
imul dword [esp+4] ;; eax *= m
loop loop1
pop edx
pop ecx
ret
```

ESP

2080

2080

pushed EDX

2084

pushed ECX

2088

return address

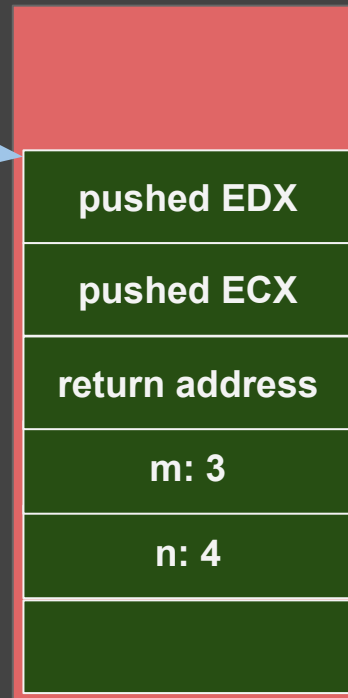
2092

m: 3

2096

n: 4

2100



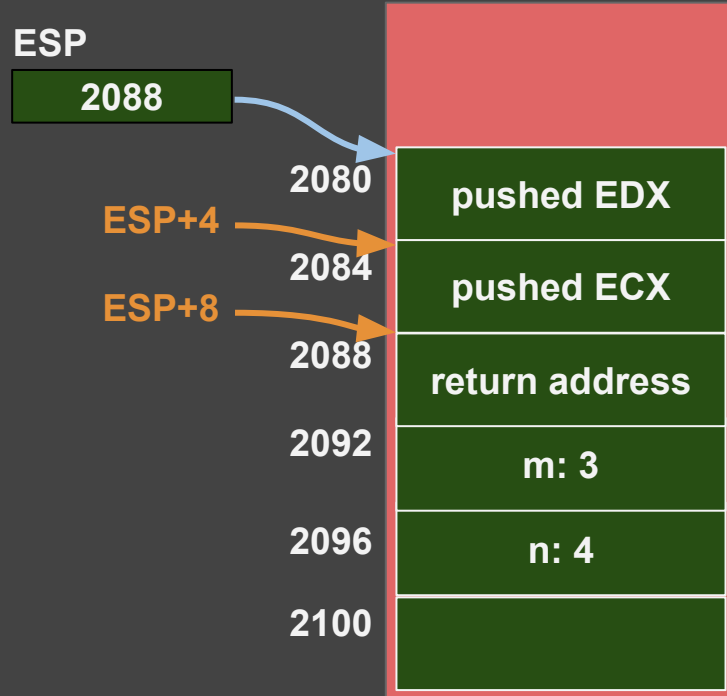
# Accessing parameters via ESP



```
powfunc4.asm
;; pow(3,4)
push 4 ; push n
push 3 ; push m

call pow
add esp, 8

call print_int
call print_nl
:
pow: ;; pow(m,n)
push ecx
push edx
mov ecx, [esp+8] ;; ecx = n
mov eax, 1
loop1:
imul dword [esp+4] ;; eax *= m
loop loop1
pop edx
pop ecx
ret
```



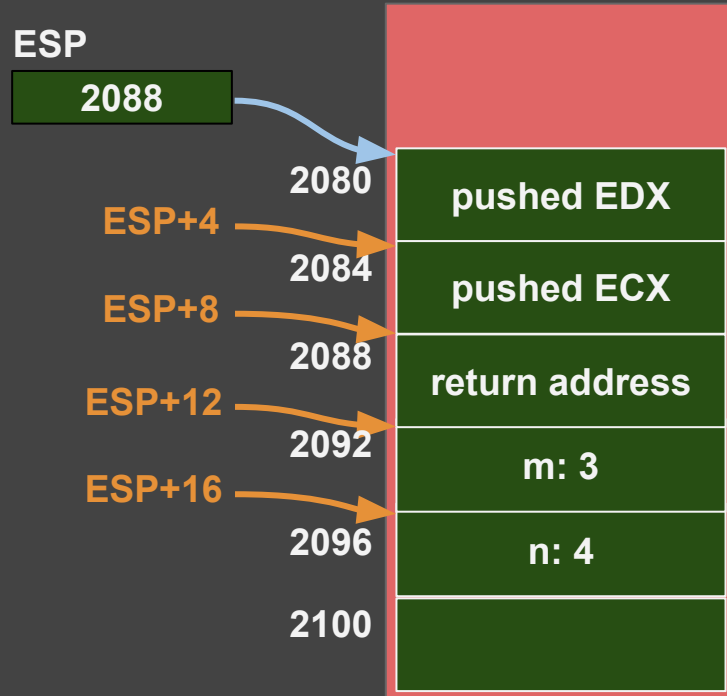
# Accessing parameters via ESP



```
powfunc4.asm
;; pow(3,4)
push 4 ; push n
push 3 ; push m

call pow
add esp, 8

call print_int
call print_nl
:
pow: ;; pow(m,n)
push ecx
push edx
mov ecx, [esp+8] ;; ecx = n
mov eax, 1
loop1:
imul dword [esp+4] ;; eax *= m
loop loop1
pop edx
pop ecx
ret
```



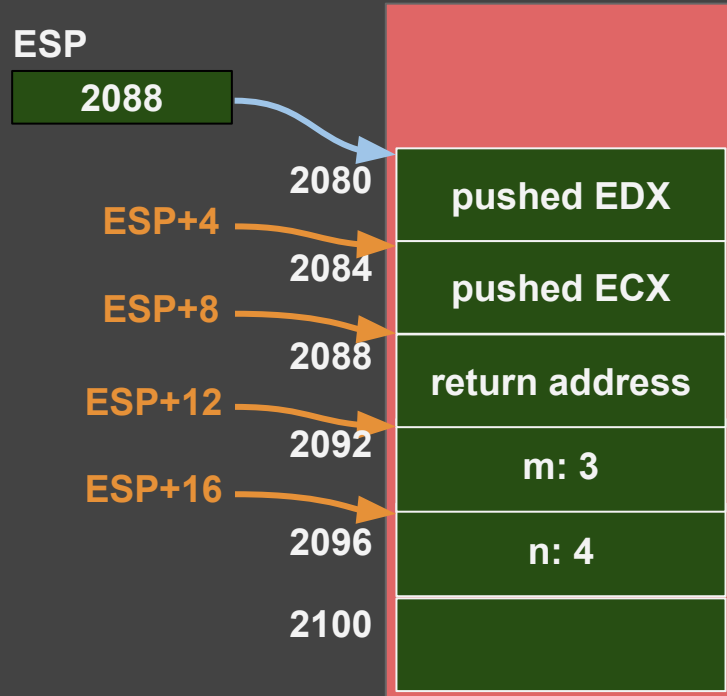
# Accessing parameters via ESP



```
powfunc6.asm
;; pow(3,4)
push 4    ; push n
push 3    ; push m

call pow
add esp, 8

call print_int
call print_nl
:
pow:    ;; pow(m,n)
push ecx
push edx
mov ecx, [esp+16]    ;; ecx = n
mov eax, 1
loop1:
imul dword [esp+12] ;; eax *= m
loop loop1
pop edx
pop ecx
ret
```





# Accessing parameters via ESP



```
powfunc6.asm
;; pow(3,4)
push 4 ; push n
push 3 ; push m

call pow
add esp, 8

call print_int
call print_nl
:
pow: ;; pow(m,n)
push ecx
push edx
mov ecx, [esp+16] ;; ecx = n
mov eax, 1
loop1:
imul dword [esp+12] ;; eax *= m
loop loop1
pop edx
pop ecx
ret
```

ESP

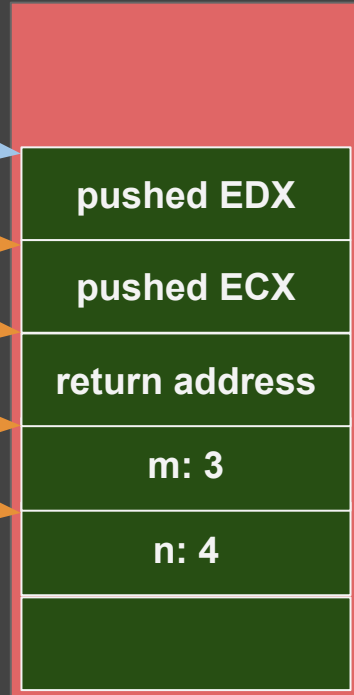
2088

ESP+4 2080  
ESP+8 2084  
ESP+12 2088  
ESP+16 2092

2096

2100

OK!



```
b.nasihatkon@kntu:lecture10$ ./run.sh powfunc6
81
```

# Accessing parameters via ESP



K. N. Toosi  
University of Technology

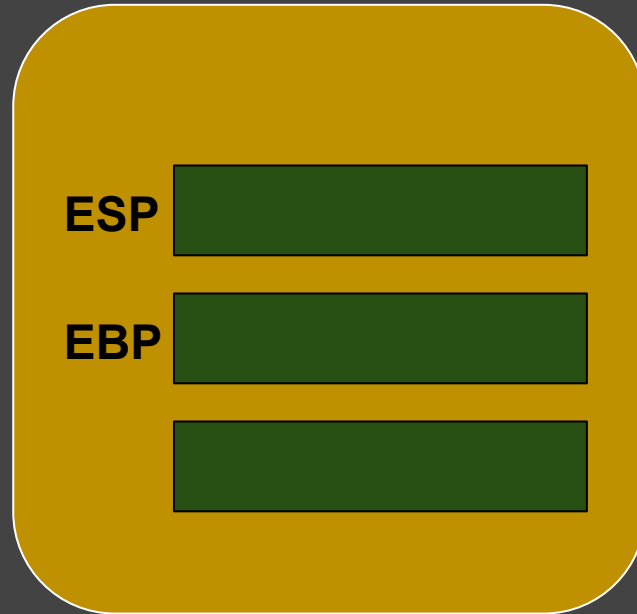
- ESP may change several times inside a function due to stack operations
- Relative address of a parameter w.r.t. ESP is not constant

# the Base Pointer Register: EBP



K. N. Toosi  
University of Technology

## x86 32-bit CPU

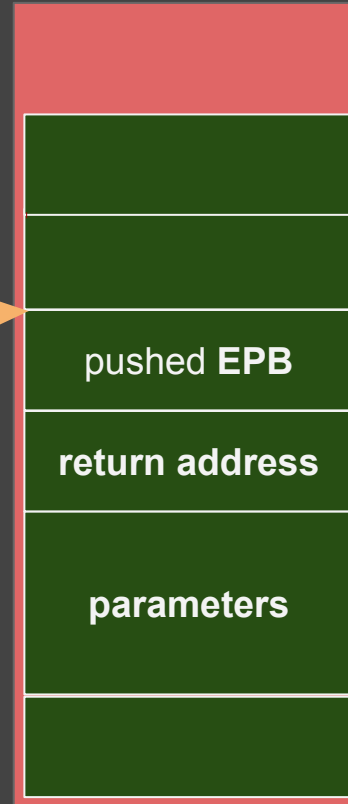
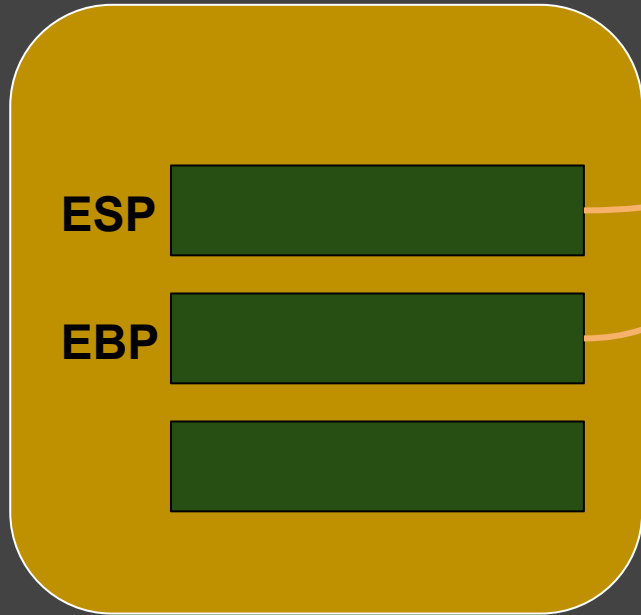


# the Base Pointer Register: EBP



K. N. Toosi  
University of Technology

CPU

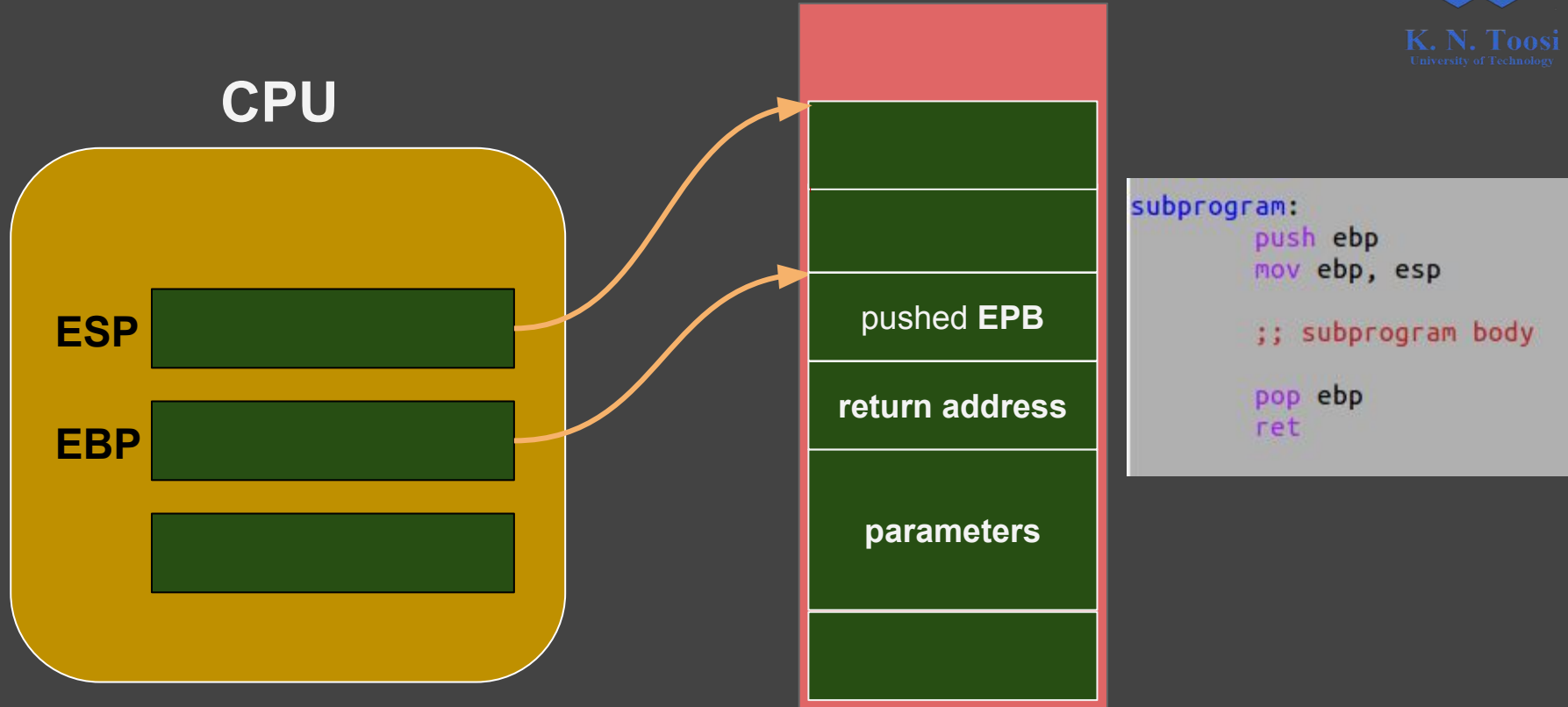


```
subprogram:  
    push ebp  
    mov  ebp, esp  
  
    ;; subprogram body  
  
    pop  ebp  
    ret
```

# the Base Pointer Register: EBP



K. N. Toosi  
University of Technology

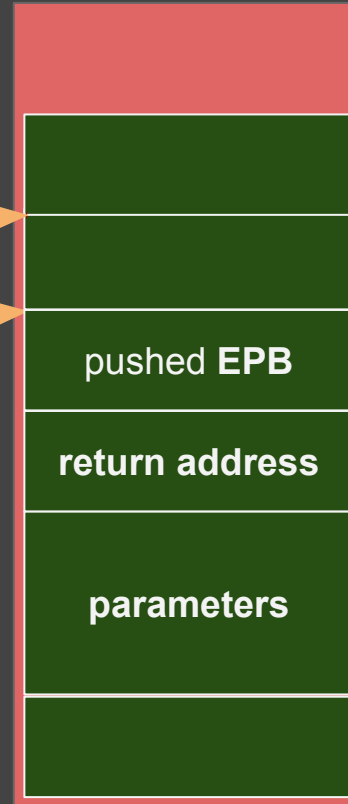
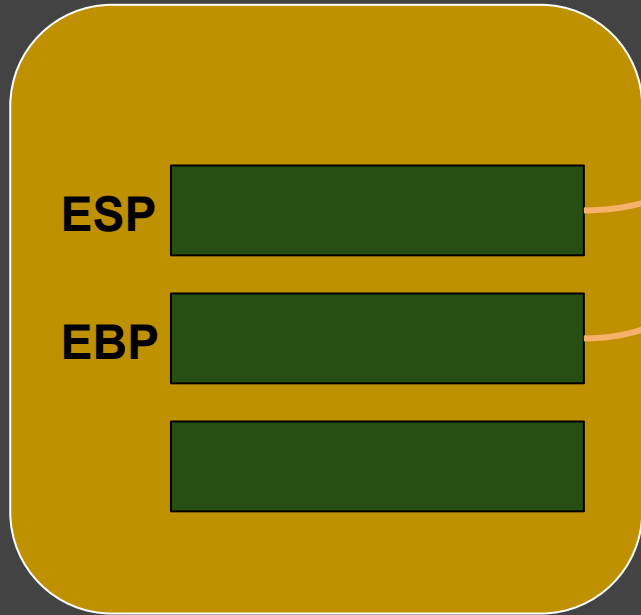


# the Base Pointer Register: EBP



K. N. Toosi  
University of Technology

CPU

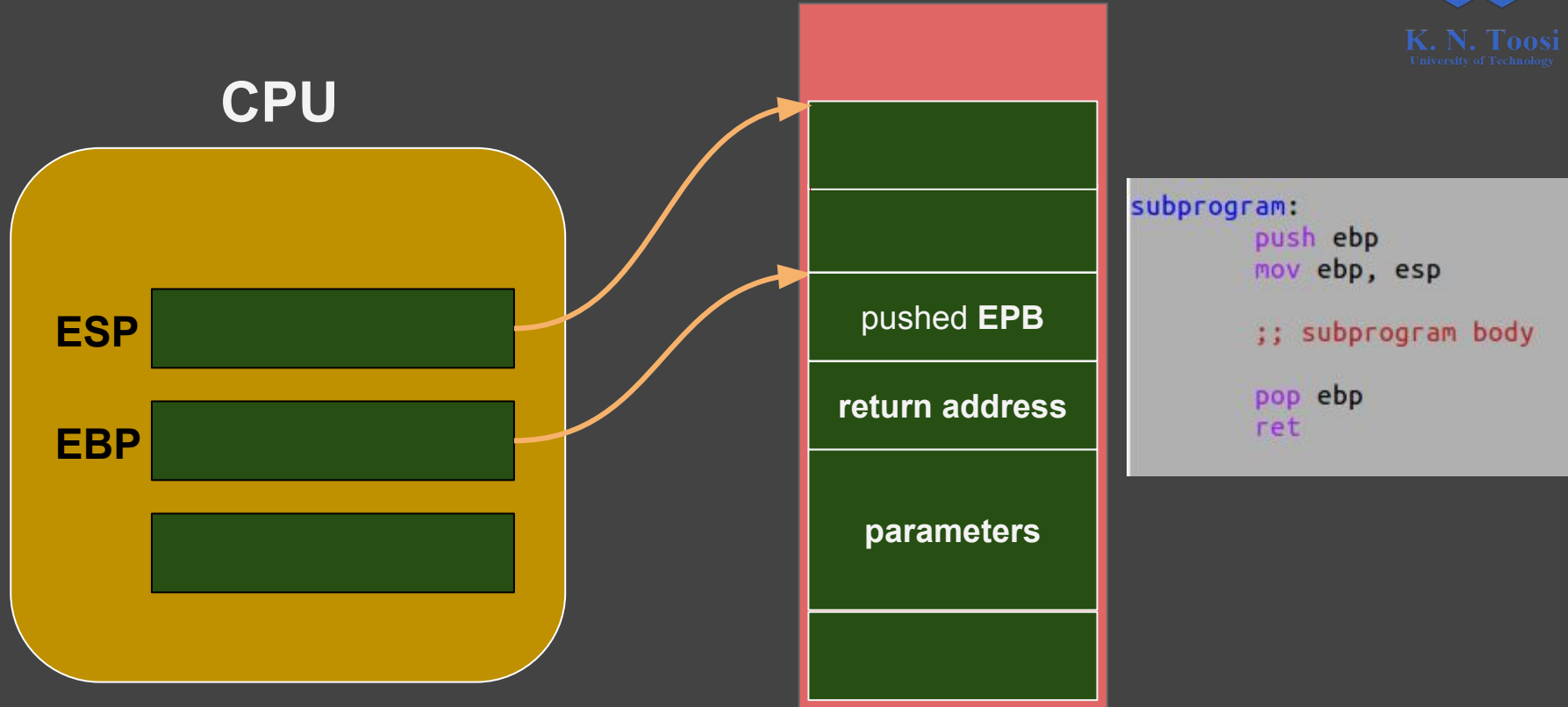


```
subprogram:  
    push ebp  
    mov  ebp, esp  
  
    ;; subprogram body  
  
    pop  ebp  
    ret
```

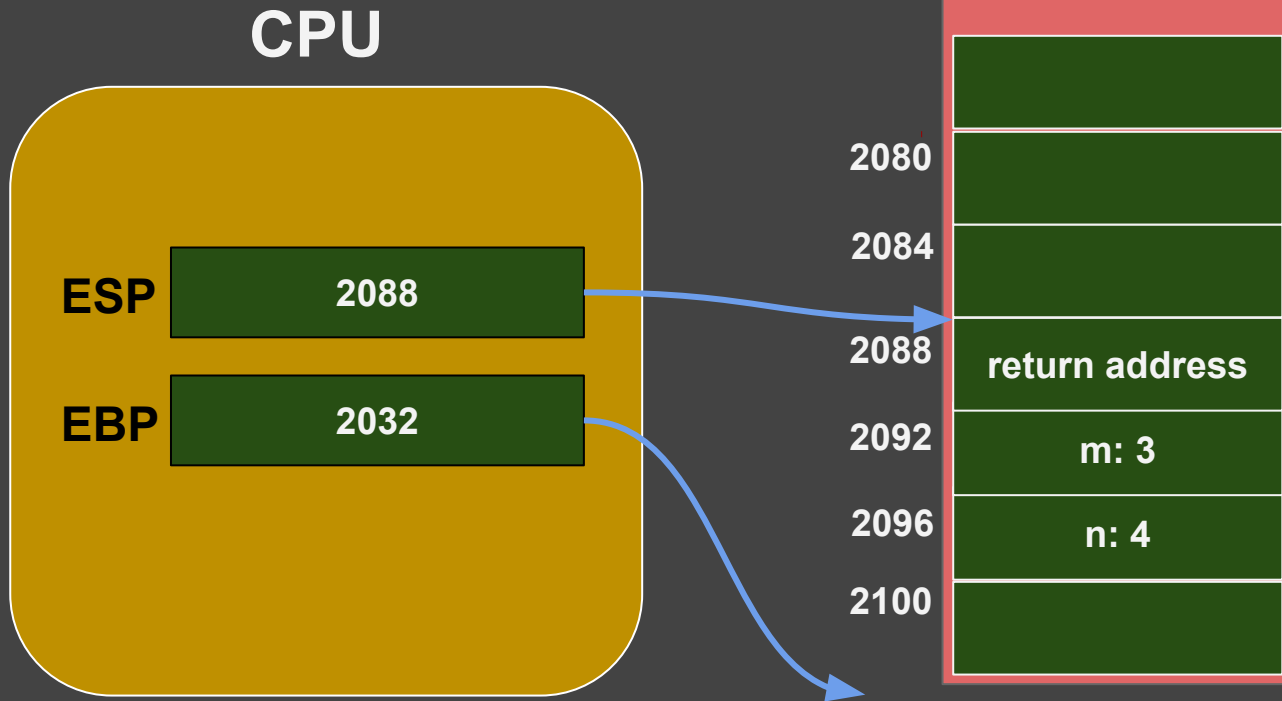
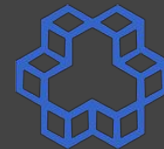
# the Base Pointer Register: EBP



K. N. Toosi  
University of Technology



# Accessing Parameters via EBP



```
pow: powfunc7.asm
    push ebp
    mov ebp, esp

    push ecx
    push edx

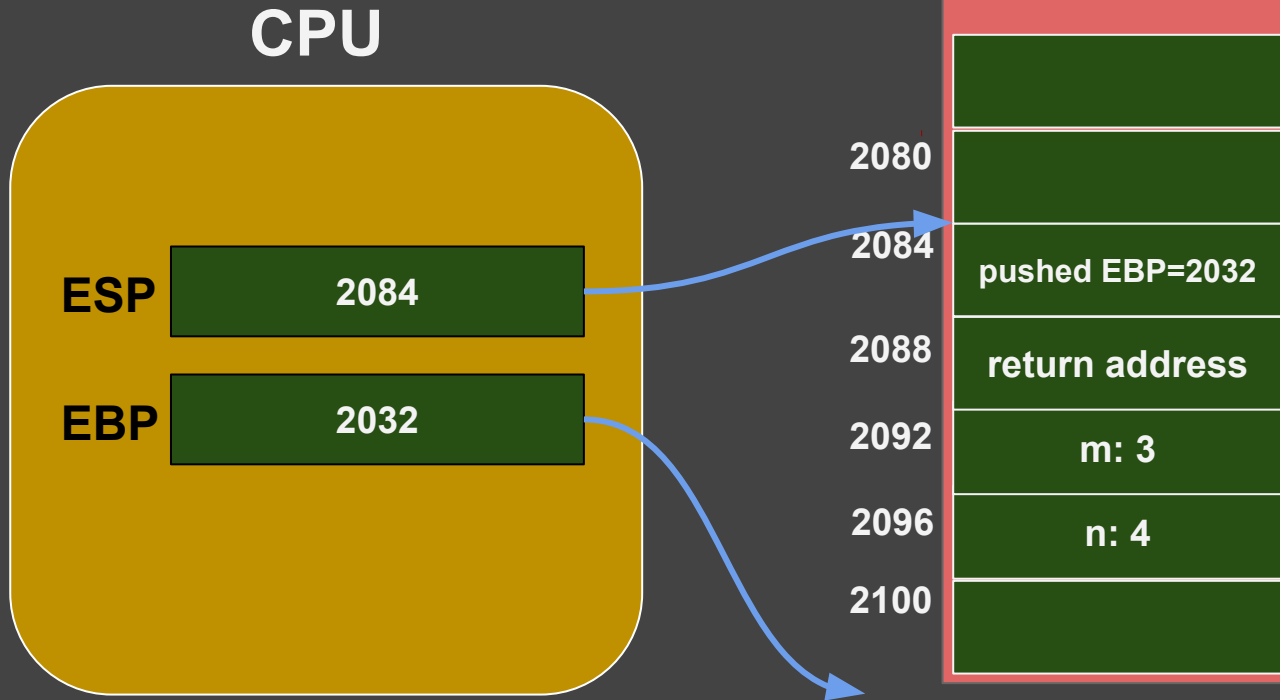
    mov ecx, [ebp+12]
    mov eax, 1
loop1:
    imul dword [ebp+8]
    loop loop1

    pop edx
    pop ecx

    pop ebp
    ret
```



# Accessing Parameters via EBP



```
pow: powfunc7.asm
→ push ebp
  mov ebp, esp

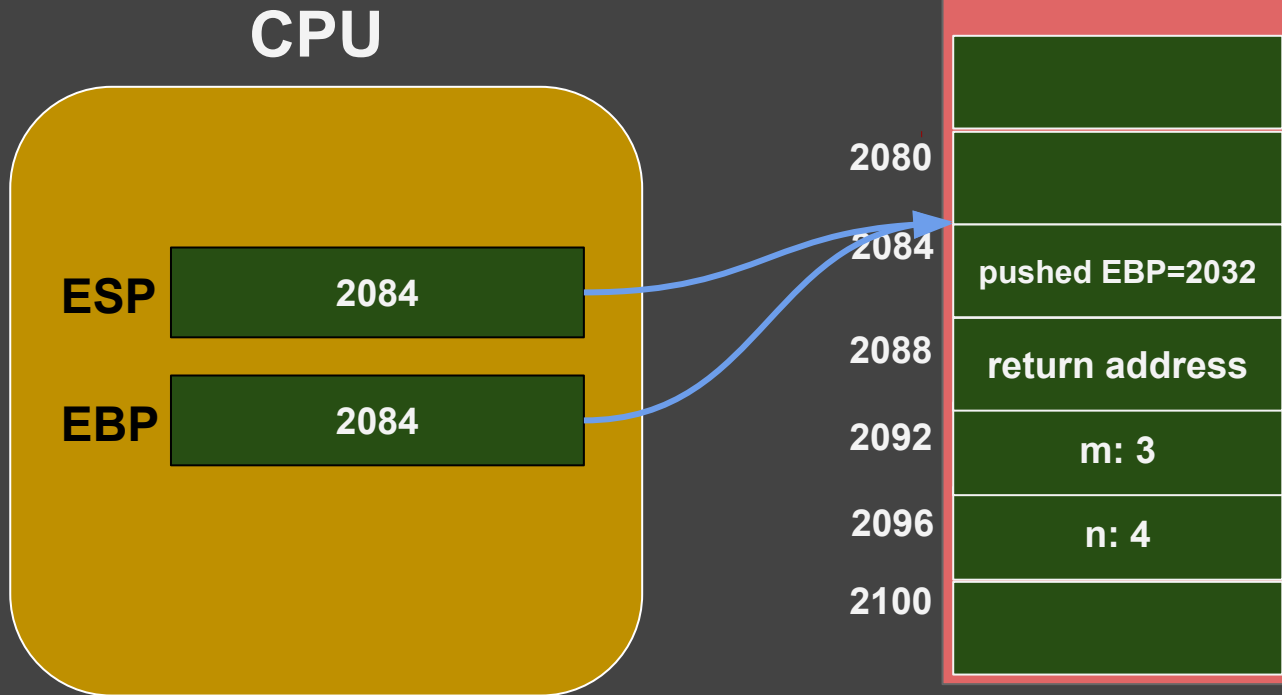
  push ecx
  push edx

  mov ecx, [ebp+12]
  mov eax, 1
loop1:
  imul dword [ebp+8]
  loop loop1

  pop edx
  pop ecx

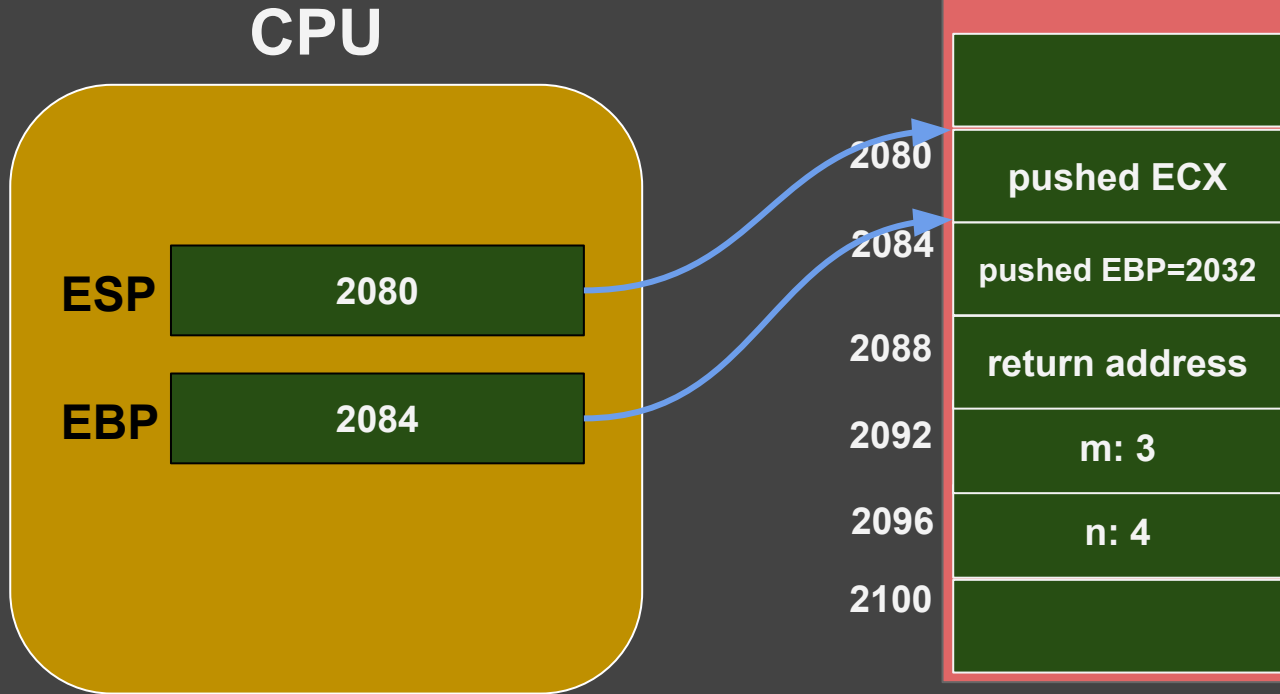
  pop ebp
  ret
```

# Accessing Parameters via EBP



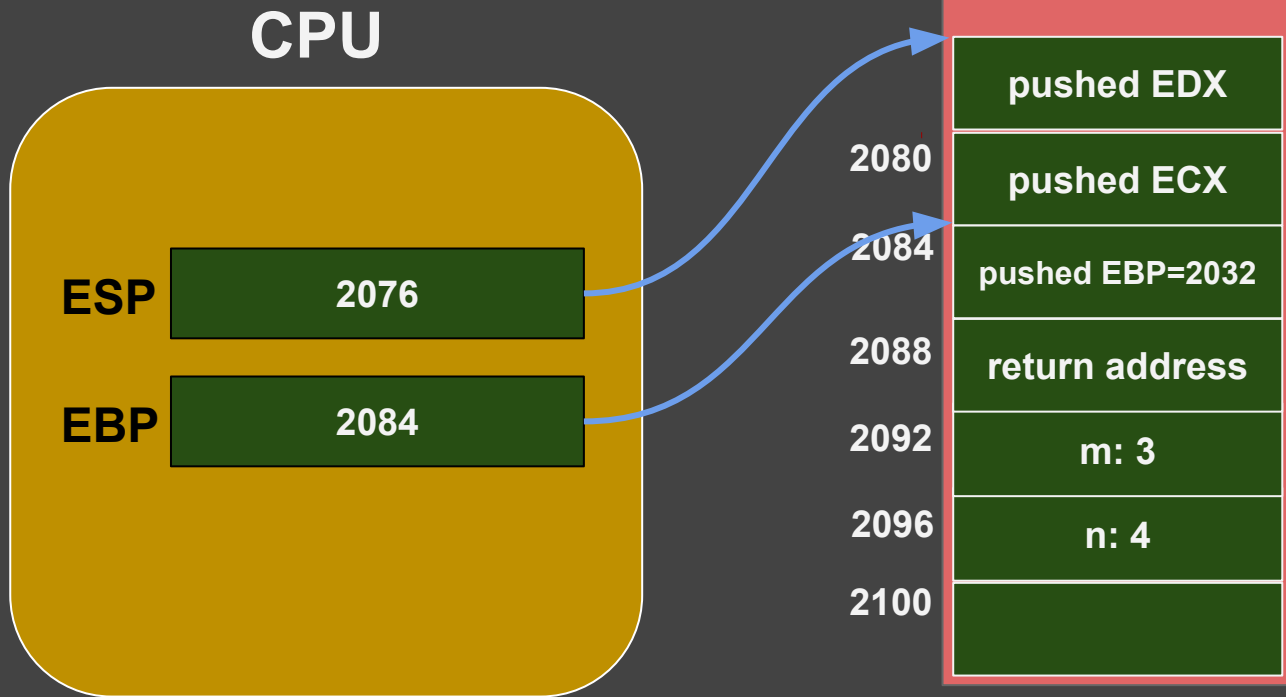
```
pow: powfunc7.asm
    push ebp
    mov ebp, esp
    push ecx
    push edx
    mov ecx, [ebp+12]
    mov eax, 1
loop1:
    imul dword [ebp+8]
    loop loop1
    pop edx
    pop ecx
    pop ebp
    ret
```

# Accessing Parameters via EBP



```
pow: powfunc7.asm
    push ebp
    mov ebp, esp
    push ecx
    push edx
    mov ecx, [ebp+12]
    mov eax, 1
loop1:
    imul dword [ebp+8]
    loop loop1
    pop edx
    pop ecx
    pop ebp
    ret
```

# Accessing Parameters via EBP



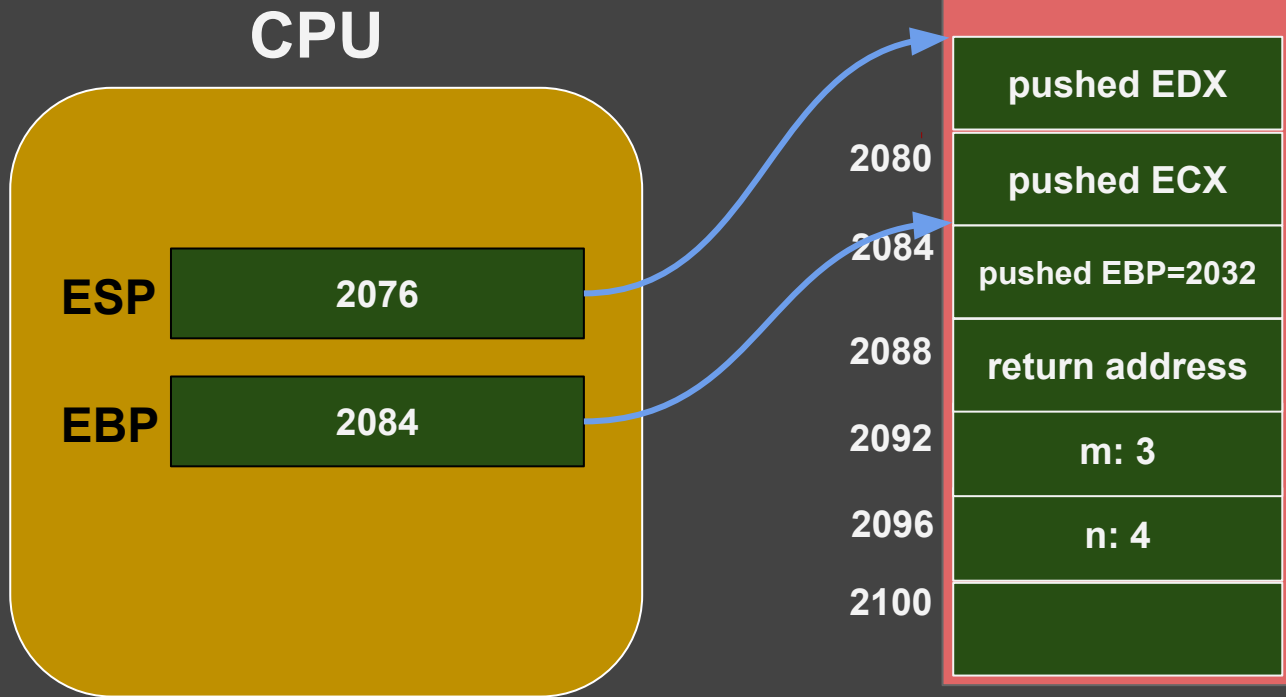
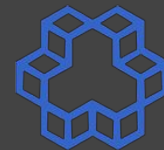
```
pow: powfunc7.asm
    push ebp
    mov ebp, esp

    push ecx
    push edx
    mov ecx, [ebp+12]
    mov eax, 1
loop1:
    imul dword [ebp+8]
    loop loop1

    pop edx
    pop ecx

    pop ebp
    ret
```

# Accessing Parameters via EBP



```
pow: powfunc7.asm
    push ebp
    mov ebp, esp

    push ecx
    push edx

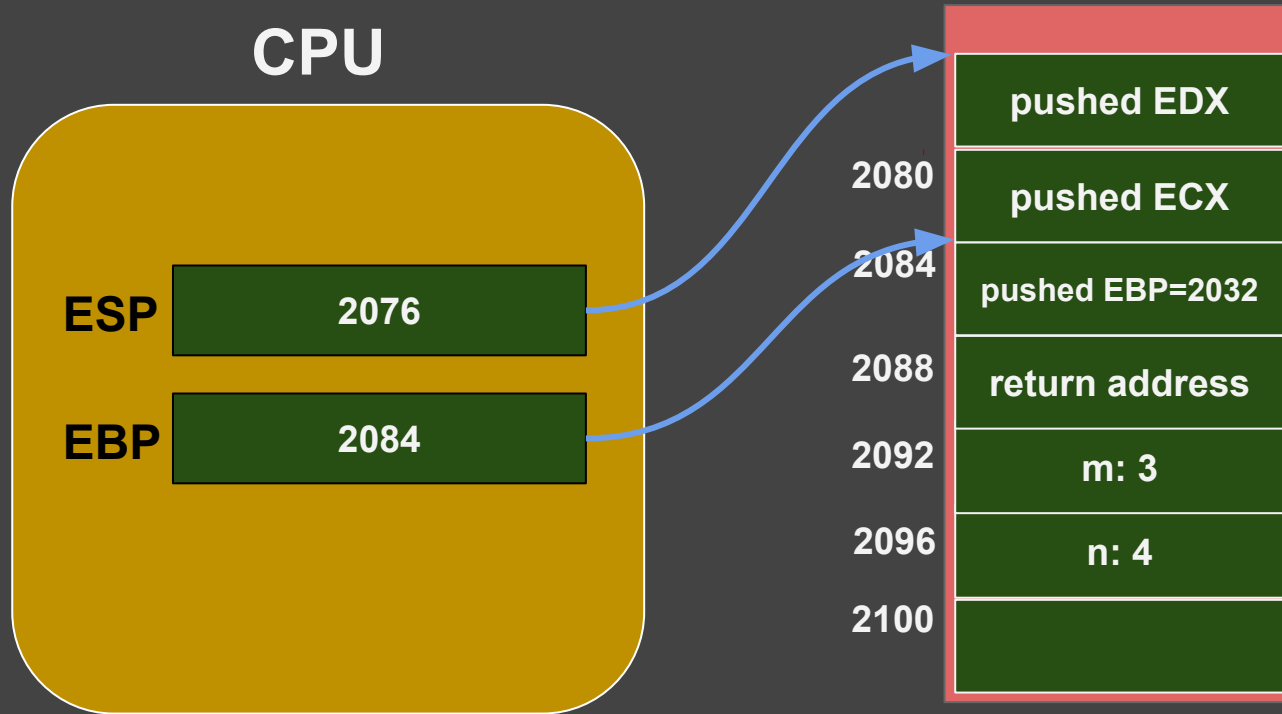
    → mov ecx, [ebp+12]
    mov eax, 1

loop1:
    imul dword [ebp+8]
    loop loop1

    pop edx
    pop ecx

    pop ebp
    ret
```

# Accessing Parameters via EBP



```
pow: powfunc7.asm
    push ebp
    mov ebp, esp

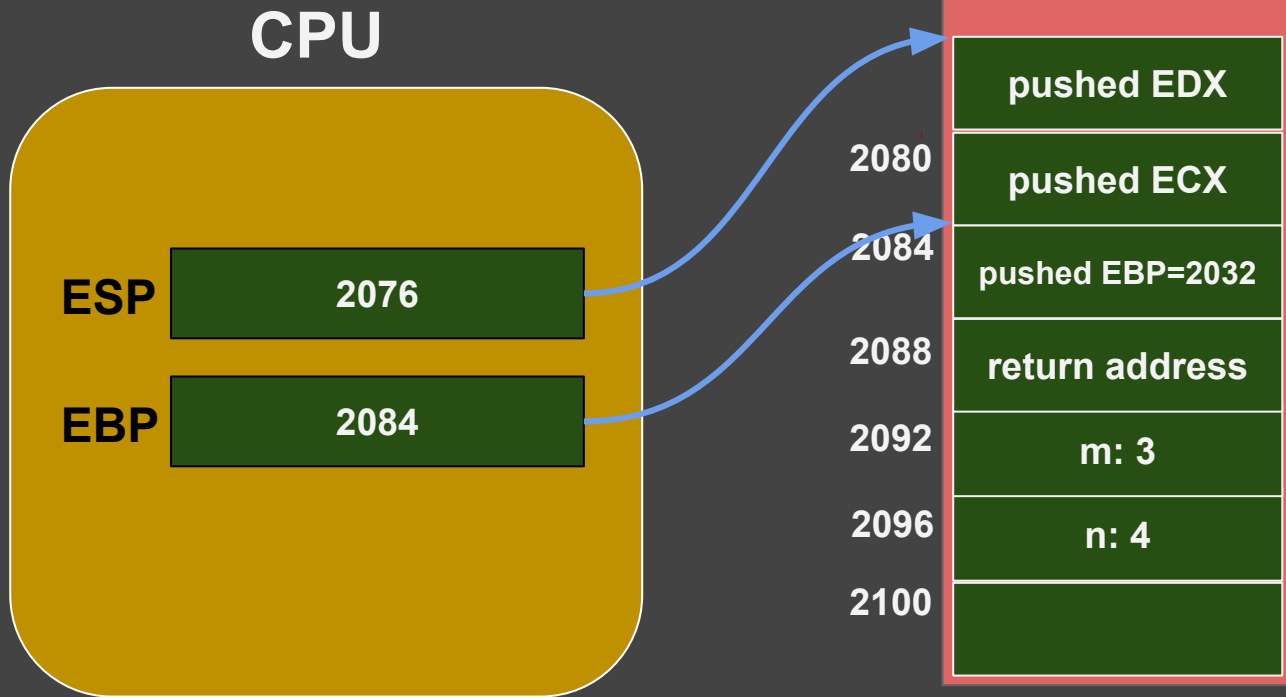
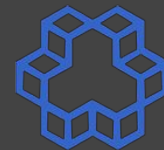
    push ecx
    push edx

    mov ecx, [ebp+12]
    mov eax, 1
loop1:
    imul dword [ebp+8]
    loop loop1

    pop edx
    pop ecx

    pop ebp
    ret
```

# Accessing Parameters via EBP



pow: powfunc7.asm

```
push ebp
mov ebp, esp

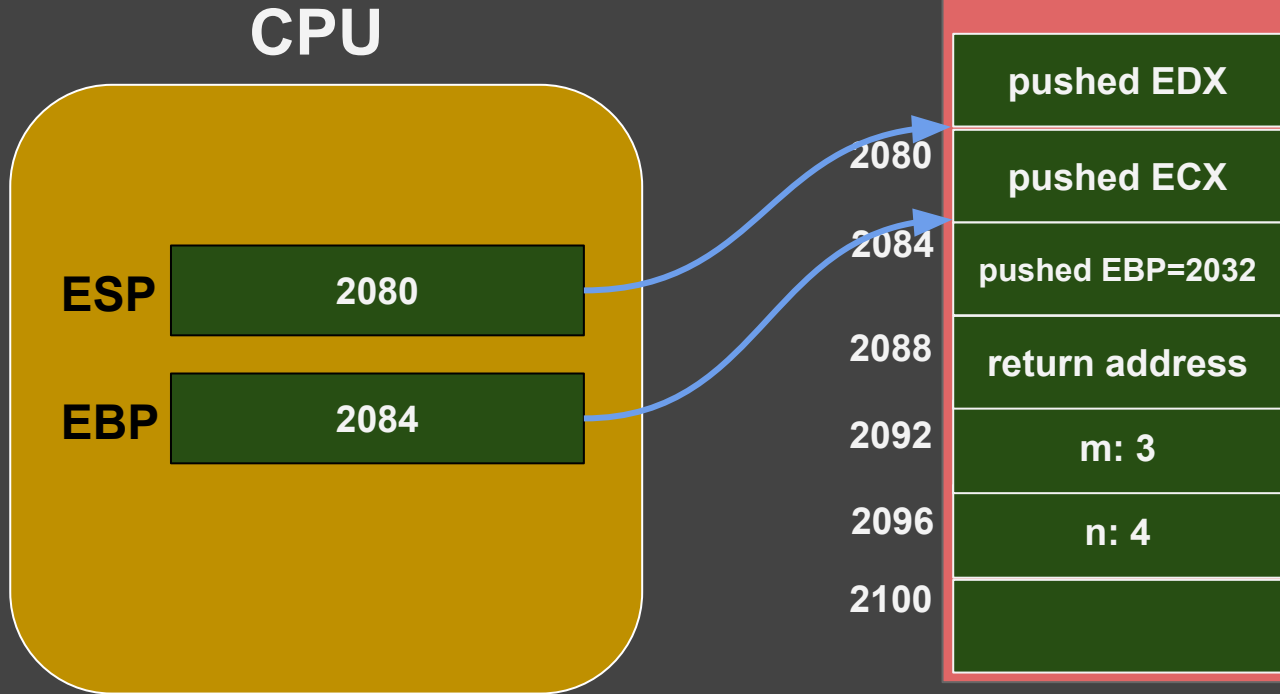
push ecx
push edx

mov ecx, [ebp+12]
mov eax, 1
loop1:
imul dword [ebp+8]
loop loop1

pop edx
pop ecx

pop ebp
ret
```

# Accessing Parameters via EBP



```
pow: powfunc7.asm
    push ebp
    mov ebp, esp

    push ecx
    push edx

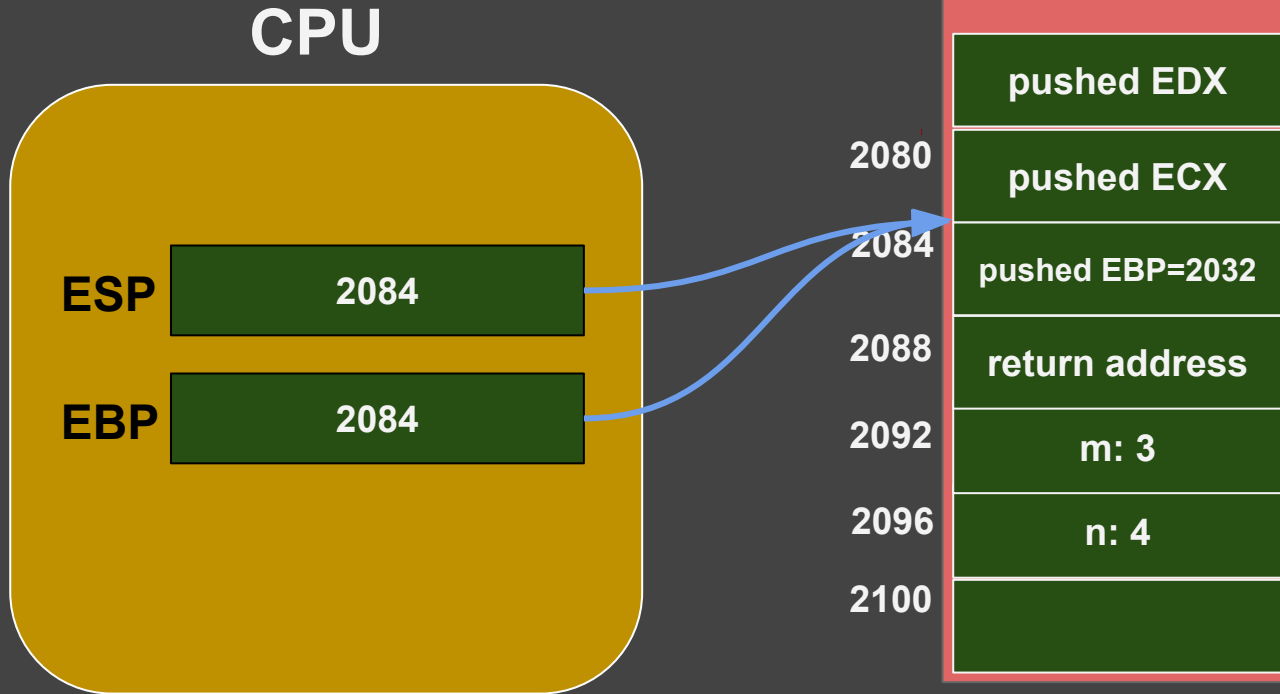
    mov ecx, [ebp+12]
    mov eax, 1
loop1:
    imul dword [ebp+8]
    loop loop1

    pop edx
    pop ecx

    pop ebp
    ret
```



# Accessing Parameters via EBP



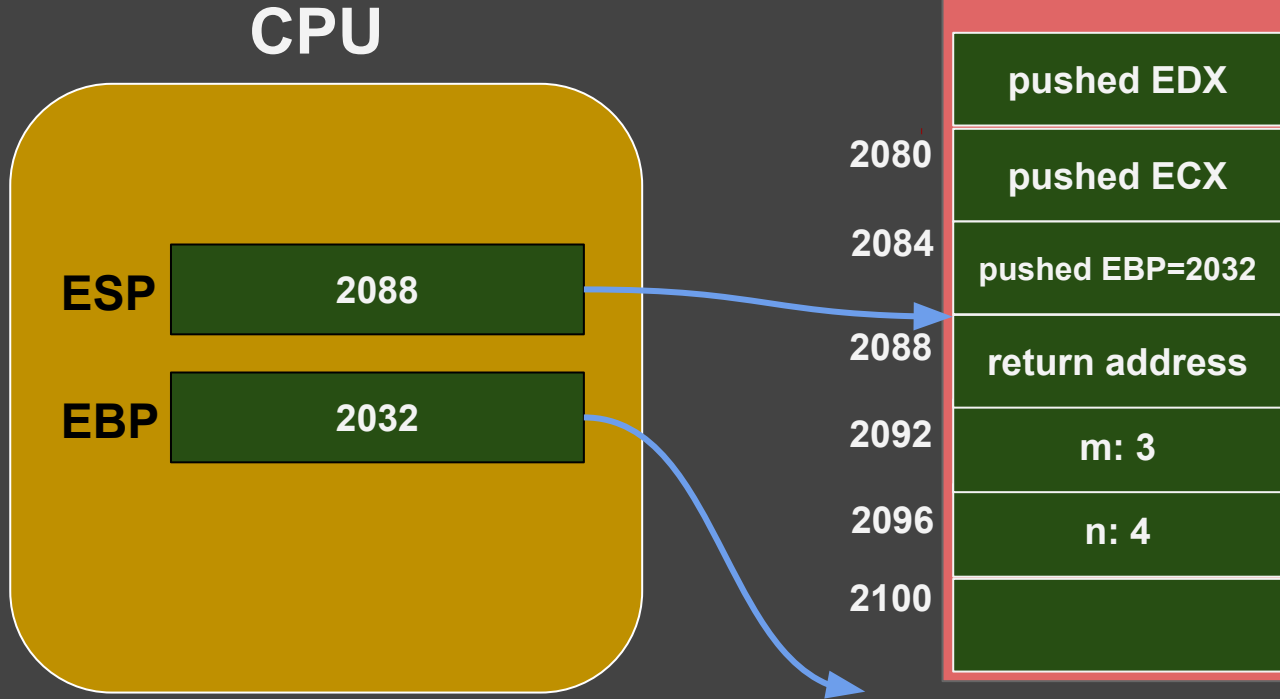
```
pow: powfunc7.asm
    push ebp
    mov ebp, esp

    push ecx
    push edx

    mov ecx, [ebp+12]
    mov eax, 1
loop1:
    imul dword [ebp+8]
    loop loop1

    pop edx
    pop ecx
    pop ebp
    ret
```

# Accessing Parameters via EBP



```
pow: powfunc7.asm
    push ebp
    mov ebp, esp

    push ecx
    push edx

    mov ecx, [ebp+12]
    mov eax, 1
loop1:
    imul dword [ebp+8]
    loop loop1

    pop edx
    pop ecx

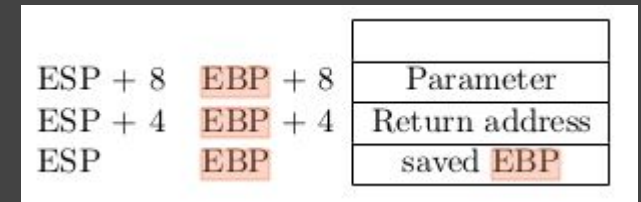
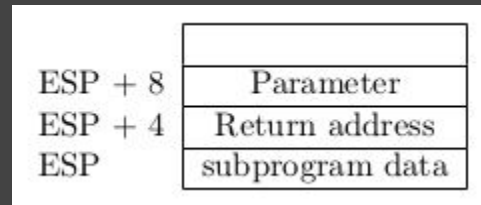
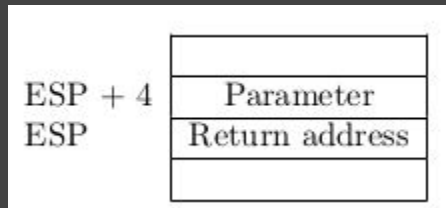
    pop ebp
    ret
```

# Figures from the book



K. N. Toosi  
University of Technology

memory drawn in reverse order:



Carter, *PC Assembly Language*, 2007.

# Calling Conventions



K. N. Toosi  
University of Technology

- Parameters/return value can get passed in different ways
- A Calling Convention specifies standards about how a subprogram is implemented, such as
  - how the subprogram receive parameters,
  - how it returns a value (or multiple values),
  - what registers need to be unaltered,
  - etc.
- Varies among different programming languages (sometimes even different compilers)
- Here, we mainly discuss calling convention of the **C** programming language

# C Calling Conventions



K. N. Toosi  
University of Technology

```
callfunc.c
#include <stdio.h>

int sum(int,int,int,int);

int main() {
    int c;

    c = sum(2,4,8,10);

    return 0;
}

int sum(int a, int b, int c, int d) {
    return a+b+c+d;
}
```

# C Calling Conventions



K. N. Toosi  
University of Technology

```
#include <stdio.h>
int sum(int,int,int,int);

int main() {
    int c;

    c = sum(2,4,8,10);

    return 0;
}

int sum(int a, int b, int c, int d) {
    return a+b+c+d;
}
```

callfunc.c

create 32-bit  
assembly

intel assembly  
syntax

compile to  
assembly

```
gcc -m32 -S -masm=intel -o callfunc.asm callfunc.c
```

# C Calling Conventions



```
#include <stdio.h>
callfunc.c

int sum(int,int,int,int);

int main() {
    int c;

    c = sum(2,4,8,10);

    return 0;
}

int sum(int a, int b, int c, int d) {
    return a+b+c+d;
}
```

```
callfunc.asm

.file "callfunc.c"
.intel_syntax noprefix
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
lea ecx, [esp+4]
.cfi_def_cfa 1, 0
and esp, -16
push DWORD PTR [ecx-4]
push ebp
.cfi_escape 0x10,0x5,0x2,0x75,0
mov ebp, esp
push ecx
.cfi_escape 0xf,0x3,0x75,0x7c,0x6
sub esp, 20
```

```
callfunc.asm (cont.)

push 10
push 8
push 4
push 2
call sum
add esp, 16
mov DWORD PTR [ebp-12], eax
mov eax, 0
mov ecx, DWORD PTR [ebp-4]
.cfi_def_cfa 1, 0
leave
.cfi_restore 5
lea esp, [ecx-4]
.cfi_def_cfa 4, 4
ret
.cfi_endproc
:
:
```

```
gcc -m32 -S -masm=intel -o callfunc.asm callfunc.c
```

# C Calling Conventions



K. N. Toosi  
University of Technology

```
#include <stdio.h> callfunc.c

int sum(int,int,int,int);

int main() {
    int c;

    c = sum(2,4,8,10);

    return 0;
}

int sum(int a, int b, int c, int d) {
    return a+b+c+d;
}
```

no cfi directives  
(to get rid of the clutter)



```
gcc -m32 -S -masm=intel -fno-asynchronous-unwind-tables -o callfunc.asm callfunc.c
```



# C Calling Conventions



```
#include <stdio.h>
```

```
callfunc.c
```

```
int sum(int,int,int,int);
```

```
int main() {  
    int c;
```

```
    c = sum(2,4,8,10);
```

```
    return 0;
```

```
}
```

```
int sum(int a, int b, int c, int d) {  
    return a+b+c+d;
```

```
}
```

```
callfunc.asm
```

```
.file "callfunc.c"  
.intel_syntax noprefix  
.text  
.globl main  
.type main, @function  
main:  
    lea ecx, [esp+4]  
    and esp, -16  
    push DWORD PTR [ecx-4]  
    push ebp  
    mov ebp, esp  
    push ecx  
    sub esp, 20  
    push 10  
    push 8  
    push 4  
    push 2  
    call sum  
    add esp, 16  
    mov DWORD PTR [ebp-12], eax
```

```
callfunc.asm (cont.)
```

```
mov eax, 0  
mov ecx, DWORD PTR [ebp-4]  
leave  
lea esp, [ecx-4]  
ret  
.size main, .-main  
.globl sum  
.type sum, @function  
sum:  
    push ebp  
    mov ebp, esp  
    mov edx, DWORD PTR [ebp+8]  
    mov eax, DWORD PTR [ebp+12]  
    add edx, eax  
    mov eax, DWORD PTR [ebp+16]  
    add edx, eax  
    mov eax, DWORD PTR [ebp+20]  
    add eax, edx  
    pop ebp  
    ret
```

```
gcc -m32 -S -masm=intel -fno-asynchronous-unwind-tables -o callfunc.asm callfunc.c
```

# C Calling Conventions



```
#include <stdio.h>
```

callfunc.c

```
int sum(int,int,int,int);
```

```
int main() {  
    int c;
```

```
    c = sum(2,4,8,10);
```

```
    return 0;  
}
```

```
int sum(int a, int b, int c, int d) {  
    return a+b+c+d;  
}
```

callfunc.asm

```
.file "callfunc.c"  
.intel_syntax noprefix  
.text  
.globl main  
.type main, @function
```

main:

```
    lea ecx, [esp+4]  
    and esp, -16  
    push DWORD PTR [ecx-4]  
    push ebp  
    mov ebp, esp  
    push ecx  
    sub esp, 20
```

```
    push 10  
    push 8  
    push 4  
    push 2  
    call sum  
    add esp, 16
```

```
    mov DWORD PTR [ebp-12], eax  
    mov eax, 0  
    mov ecx, DWORD PTR [ebp-4]  
    leave
```

callfunc.asm (cont.)

```
    lea esp, [ecx-4]  
    ret  
.size main, .-main  
.globl sum  
.type sum, @function
```

sum:

```
    push ebp  
    mov ebp, esp  
    mov edx, DWORD PTR [ebp+8]  
    mov eax, DWORD PTR [ebp+12]  
    add edx, eax  
    mov eax, DWORD PTR [ebp+16]  
    add edx, eax  
    mov eax, DWORD PTR [ebp+20]  
    add eax, edx  
    pop ebp  
    ret
```

```
.size sum, .-sum  
.ident "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0  
20160609"  
.section .note.GNU-stack,"",@progbits
```

# C Calling Conventions



```
#include <stdio.h>
```

callfunc.c

```
int sum(int,int,int,int);
```

```
int main() {  
    int c;
```

```
    c = sum(2,4,8,10);
```

```
    return 0;  
}
```

```
int sum(int a, int b, int c, int d) {  
    return a+b+c+d;  
}
```

callfunc.asm

```
.file "callfunc.c"  
.intel_syntax noprefix  
.text  
.globl main  
.type main, @function
```

main:

```
    lea ecx, [esp+4]  
    and esp, -16  
    push DWORD PTR [ecx-4]  
    push ebp  
    mov ebp, esp  
    push ecx  
    sub esp, 20
```

```
    push 10  
    push 8  
    push 4  
    push 2  
    call sum  
    add esp, 16
```

→ last parameter pushed first

```
    mov DWORD PTR [ebp-12], eax  
    mov eax, 0  
    mov ecx, DWORD PTR [ebp-4]  
    leave
```

callfunc.asm (cont.)

```
    lea esp, [ecx-4]  
    ret  
    .size main, .-main  
    .globl sum  
    .type sum, @function
```

sum:

```
    push ebp  
    mov ebp, esp  
    mov edx, DWORD PTR [ebp+8]  
    mov eax, DWORD PTR [ebp+12]  
    add edx, eax  
    mov eax, DWORD PTR [ebp+16]  
    add edx, eax  
    mov eax, DWORD PTR [ebp+20]  
    add eax, edx  
    pop ebp  
    ret
```

```
    .size sum, .-sum  
    .ident "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0  
20160609"  
    .section .note.GNU-stack,"",@progbits
```

# C Calling Conventions



```
#include <stdio.h>
```

callfunc.c

```
int sum(int,int,int,int);
```

```
int main() {  
    int c;
```

```
    c = sum(2,4,8,10);
```

```
    return 0;  
}
```

```
int sum(int a, int b, int c, int d) {  
    return a+b+c+d;  
}
```

callfunc.asm

```
.file "callfunc.c"  
.intel_syntax noprefix  
.text  
.globl main  
.type main, @function
```

main: → caller

```
    lea ecx, [esp+4]  
    and esp, -16  
    push DWORD PTR [ecx-4]  
    push ebp  
    mov ebp, esp  
    push ecx  
    sub esp, 20
```

```
    push 10  
    push 8  
    push 4  
    push 2
```

```
    call sum  
    add esp, 16
```

Caller clears  
the parameters  
from stack

```
    mov DWORD PTR [ebp-12], eax  
    mov eax, 0  
    mov ecx, DWORD PTR [ebp-4]  
    leave
```

callfunc.asm (cont.)

```
    lea esp, [ecx-4]  
    ret  
.size main, -main  
.globl sum  
.type sum, @function
```

sum: → callee

```
    push ebp  
    mov ebp, esp  
    mov edx, DWORD PTR [ebp+8]  
    mov eax, DWORD PTR [ebp+12]  
    add edx, eax  
    mov eax, DWORD PTR [ebp+16]  
    add edx, eax  
    mov eax, DWORD PTR [ebp+20]  
    add eax, edx  
    pop ebp  
    ret
```

```
.size sum, -sum  
.ident "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0  
20160609"  
.section .note.GNU-stack,"",@progbits
```

# C Calling Conventions



```
#include <stdio.h>
int sum(int,int,int,int);

int main() {
    int c;

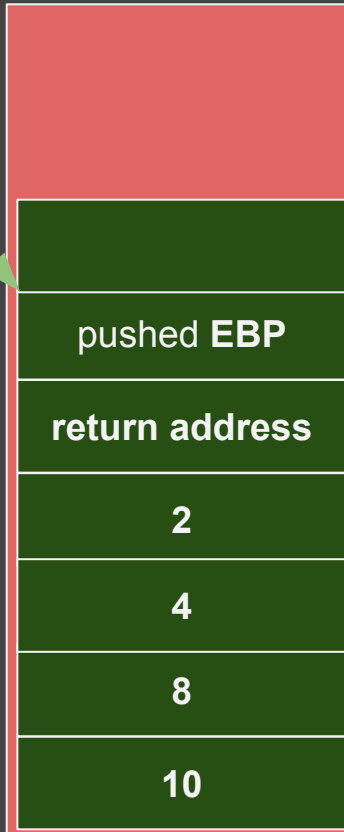
    c = sum(2,4,8,10);

    return 0;
}

int sum(int a, int b, int c, int d) {
    return a+b+c+d;
}
```

callfunc.c

ESP



```
lea esp, [ecx-4]
ret
.size main,.-main
.globl sum
.type sum, @function
```

callfunc.asm (cont.)

```
sum:      → callee
    push ebp
    mov  ebp, esp
    mov  edx, DWORD PTR [ebp+8]
    mov  eax, DWORD PTR [ebp+12]
    add  edx, eax
    mov  eax, DWORD PTR [ebp+16]
    add  edx, eax
    mov  eax, DWORD PTR [ebp+20]
    add  eax, edx
    pop  ebp
    ret

.size sum,.-sum
.ident "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0
20160609"
.section .note.GNU-stack,"",@progbits
```

# C Calling Conventions



```
#include <stdio.h>
int sum(int,int,int,int);

int main() {
    int c;

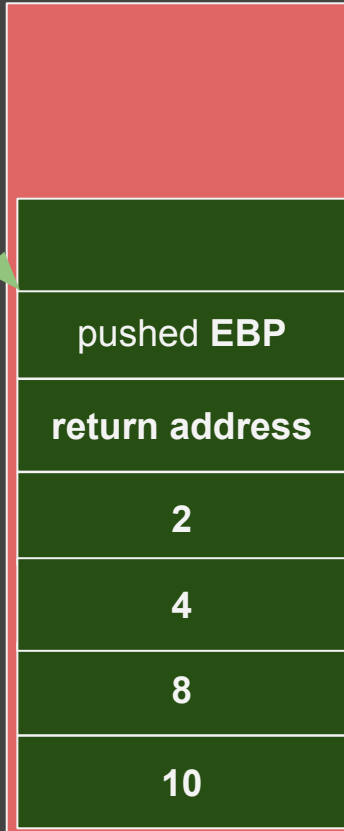
    c = sum(2,4,8,10);

    return 0;
}

int sum(int a, int b, int c, int d) {
    return a+b+c+d;
}
```

callfunc.c

ESP



```
lea esp, [ecx-4]
ret
.size main, .-main
.globl sum
.type sum, @function
```

callfunc.asm (cont.)

```
sum:      → callee
    push ebp
    mov  ebp, esp
    mov  edx, DWORD PTR [ebp+8]
    mov  eax, DWORD PTR [ebp+12]
    add  edx, eax
    mov  eax, DWORD PTR [ebp+16]
    add  edx, eax
    mov  eax, DWORD PTR [ebp+20]
    add  eax, edx
    pop  ebp
    ret

.size sum, .-sum
.ident "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0
20160609"
.section .note.GNU-stack,"",@progbits
```

# C Calling Conventions



N. Toosi  
University of Technology

```
#include <stdio.h> callfunc.c
```

```
int sum(int,int,int,int);
```

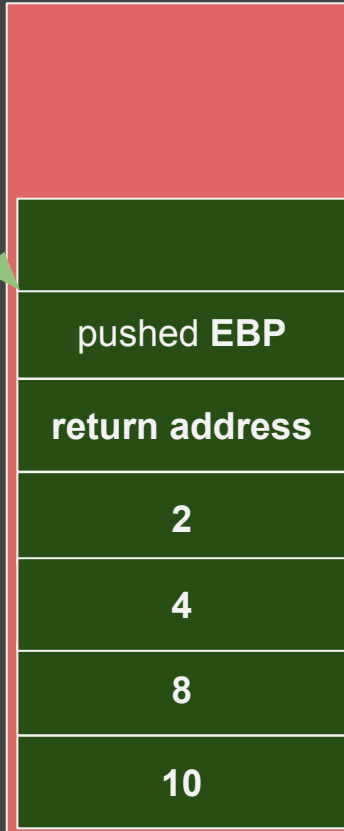
```
int main() {  
    int c;
```

```
    c = sum(2,4,8,10);
```

```
    return 0;  
}
```

```
int sum(int a, int b, int c, int d) {  
    return a+b+c+d;  
}
```

ESP



```
lea esp, [ecx-4]  
ret  
.size main, -main  
.globl sum  
.type sum, @function
```

callfunc.asm (cont.)

sum: → callee

```
push ebp
```

```
mov ebp, esp
```

```
mov edx, DWORD PTR [ebp+8] → a
```

```
mov eax, DWORD PTR [ebp+12] → b
```

```
add edx, eax
```

```
mov eax, DWORD PTR [ebp+16] → c
```

```
add edx, eax
```

```
mov eax, DWORD PTR [ebp+20] → d
```

```
add eax, edx
```

```
pop ebp
```

```
ret
```

```
.size sum, -sum
```

```
.ident "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0
```

```
20160609"
```

```
.section .note.GNU-stack,"",@progbits
```

# C Calling Conventions



N. Toosi  
University of Technology

```
#include <stdio.h> callfunc.c
```

```
int sum(int,int,int,int);
```

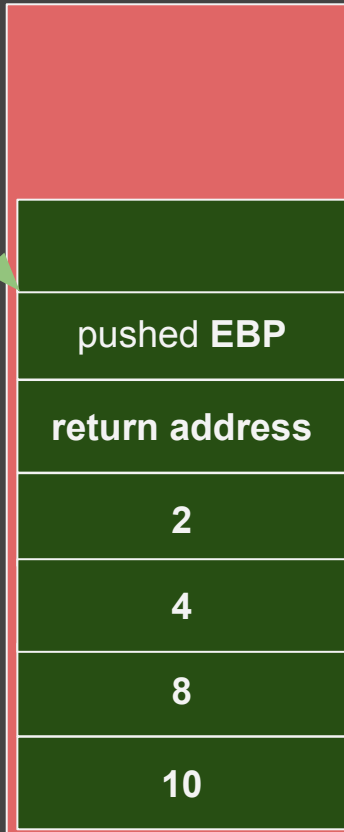
```
int main() {  
    int c;
```

```
    c = sum(2,4,8,10);
```

```
    return 0;  
}
```

```
int sum(int a, int b, int c, int d) {  
    return a+b+c+d;  
}
```

ESP



```
lea esp, [ecx-4]  
ret  
.size main, -main  
.globl sum  
.type sum, @function
```

**callfunc.asm (cont.)**

**sum:** → **callee**

```
push ebp
```

```
mov ebp, esp
```

```
mov edx, DWORD PTR [ebp+8] → a
```

```
mov eax, DWORD PTR [ebp+12] → b
```

```
add edx, eax
```

```
mov eax, DWORD PTR [ebp+16] → c
```

```
add edx, eax
```

```
mov eax, DWORD PTR [ebp+20] → d
```

```
add eax, edx
```

```
pop ebp → return value stored in EAX
```

```
ret
```

```
.size sum, -sum
```

```
.ident "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0
```

```
20160609"
```

```
.section .note.gnu-stack,"",@progbits
```





# C Calling Conventions

- CDECL (C Declaration): **default C convention**
  - STDCALL
  - FASTCALL
  - etc.
- 
- For gcc-supported calling conventions look at
    - <https://gcc.gnu.org/onlinedocs/gcc/x86-Function-Attributes.html>
  - For x86 calling conventions look at
    - [https://en.wikipedia.org/wiki/X86\\_calling\\_conventions](https://en.wikipedia.org/wiki/X86_calling_conventions)
    - [https://en.wikibooks.org/wiki/X86\\_Disassembly/Calling\\_Conventions](https://en.wikibooks.org/wiki/X86_Disassembly/Calling_Conventions)

# Second form of RET



K. N. Toosi  
University of Technology

`ret immed`

- returns to the caller and pops **immed** bytes off the stack.

# C Calling Conventions: Example



K. N. Toosi  
University of Technology

```
#include <stdio.h>
int sum(int,int,int,int);

int main() {
    int c;

    c = sum(2,4,8,10);

    return 0;
}

int sum(int a, int b, int c, int d) {
    return a+b+c+d;
}
```

cdecl (default C convention)

```
#include <stdio.h>
int __attribute ((stdcall)) sum(int,int,int,int);

int main() {
    int c;

    c = sum(2,4,8,10);

    return 0;
}

int __attribute ((stdcall)) sum(int a, int b, int c, int d) {
    return a+b+c+d;
}
```

stdcall convention

```

:
main:
:
push 10
push 8
push 4
push 2
call sum
add esp, 16

mov DWORD PTR [ebp-12], eax
:
sum:
push ebp
mov ebp, esp
mov edx, DWORD PTR [ebp+8]
mov eax, DWORD PTR [ebp+12]
add edx, eax
mov eax, DWORD PTR [ebp+16]
add edx, eax
mov eax, DWORD PTR [ebp+20]
add eax, edx
pop ebp
ret

```

cdecl (default C convention)

```

:
main:
:
push 10
push 8
push 4
push 2
call sum

mov DWORD PTR [ebp-12], eax
:
sum:
push ebp
mov ebp, esp
mov edx, DWORD PTR [ebp+8]
mov eax, DWORD PTR [ebp+12]
add edx, eax
mov eax, DWORD PTR [ebp+16]
add edx, eax
mov eax, DWORD PTR [ebp+20]
add eax, edx
pop ebp
ret 16

```

stdcall convention





```
callfunc.asm
:
main:
:
push 10
push 8
push 4
push 2
call sum
add esp, 16
mov DWORD PTR [ebp-12], eax
:
sum:
push ebp
mov ebp, esp
mov edx, DWORD PTR [ebp+8]
mov eax, DWORD PTR [ebp+12]
add edx, eax
mov eax, DWORD PTR [ebp+16]
add edx, eax
mov eax, DWORD PTR [ebp+20]
add eax, edx
pop ebp
ret
```

cdecl (default C convention)

```
callfunc2.asm
:
main:
:
push 10
push 8
push 4
push 2
call sum
mov DWORD PTR [ebp-12], eax
:
sum:
push ebp
mov ebp, esp
mov edx, DWORD PTR [ebp+8]
mov eax, DWORD PTR [ebp+12]
add edx, eax
mov eax, DWORD PTR [ebp+16]
add edx, eax
mov eax, DWORD PTR [ebp+20]
add eax, edx
pop ebp
ret 16
```

stdcall convention



```
callfunc.asm
:
main:
:
push 10
push 8
push 4
push 2
call sum
add esp, 16
mov DWORD PTR [ebp-12], eax
:
sum:
push ebp
mov ebp, esp
mov edx, DWORD PTR [ebp+8]
mov eax, DWORD PTR [ebp+12]
add edx, eax
mov eax, DWORD PTR [ebp+16]
add edx, eax
mov eax, DWORD PTR [ebp+20]
add eax, edx
pop ebp
ret
```

Caller clears  
the stack

cdecl (default C convention)

```
callfunc2.asm
:
main:
:
push 10
push 8
push 4
push 2
call sum
mov DWORD PTR [ebp-12], eax
:
sum:
push ebp
mov ebp, esp
mov edx, DWORD PTR [ebp+8]
mov eax, DWORD PTR [ebp+12]
add edx, eax
mov eax, DWORD PTR [ebp+16]
add edx, eax
mov eax, DWORD PTR [ebp+20]
add eax, edx
pop ebp
ret 16
```

Callee clears  
the stack

stdcall convention

# x86-64 C Calling Conventions



K. N. Toosi  
University of Technology

- Very different from 32-bit conventions
- Look at
  - [https://en.wikipedia.org/wiki/X86\\_calling\\_conventions#x86-64\\_calling\\_conventions](https://en.wikipedia.org/wiki/X86_calling_conventions#x86-64_calling_conventions)
  - <https://aaronbloomfield.github.io/pdr/book/x86-64bit-ccc-chapter.pdf>

# Implementing Local Variables



K. N. Toosi  
University of Technology

- Use registers

```
void subprogram(int n, int p) {  
    int sum, i;  
    :  
    :  
}
```

C

```
subprogram:
```

Assembly



# Implementing Local Variables



K. N. Toosi  
University of Technology

- Use registers
- Use data segment

```
void subprogram(int n, int p) {  
    int sum, i;  
    :  
    :  
}
```

C

```
segment .data  
subprogram_sum: dd 0  
subprogram_i:   dd 0  
  
segment .text  
subprogram:
```

Assembly

# Implementing Local Variables



K. N. Toosi  
University of Technology

- Use registers
- Use data segment
  - Global Variables

```
void subprogram(int n, int p) {  
    int sum, i;  
    :  
    :  
}
```

C

```
segment .data  
subprogram_sum: dd 0  
subprogram_i:   dd 0  
  
segment .text  
subprogram:
```

Assembly

# Implementing Local Variables



K. N. Toosi  
University of Technology

- Use registers
- Use data segment
  - Global Variables
  - Static Variables

```
void subprogram(int n, int p) {  
    int sum, i;  
    :  
    :  
}
```

C

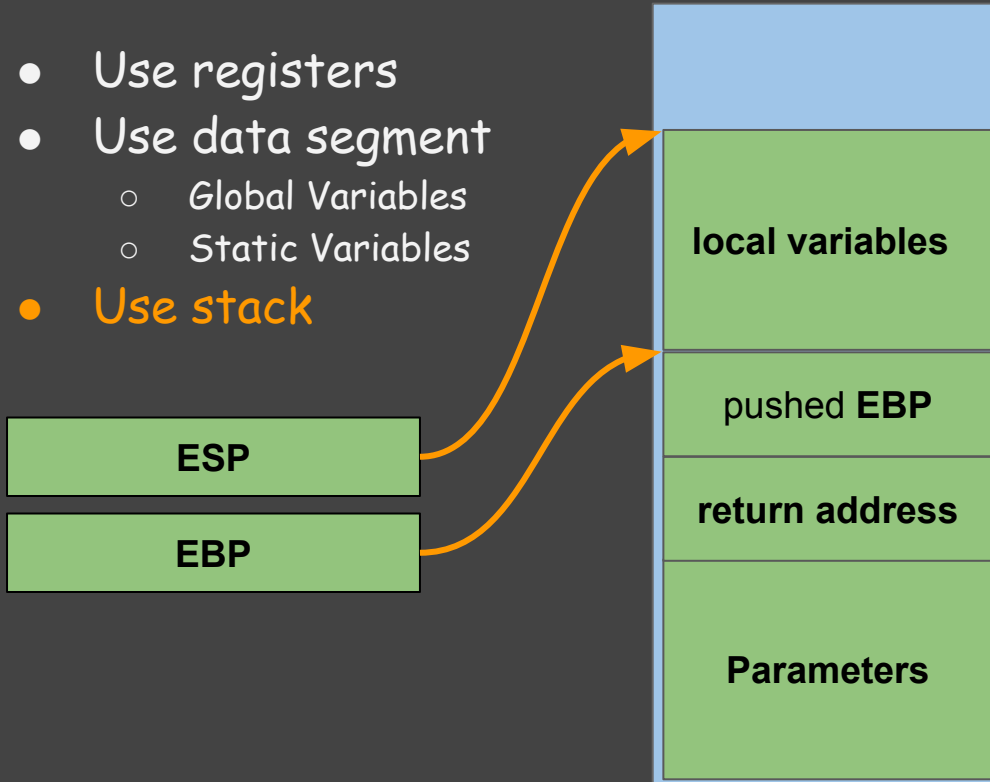
```
segment .data  
subprogram_sum: dd 0  
subprogram_i:   dd 0  
  
segment .text  
subprogram:
```

Assembly

# Implementing Local Variables



- Use registers
- Use data segment
  - Global Variables
  - Static Variables
- Use stack



**subprogram:**

```
push ebp
```

```
mov ebp, esp
```

```
sub esp, NO_OF_BYTES
```



```
;; subprogram body
```

```
mov esp, ebp ; release locals
```

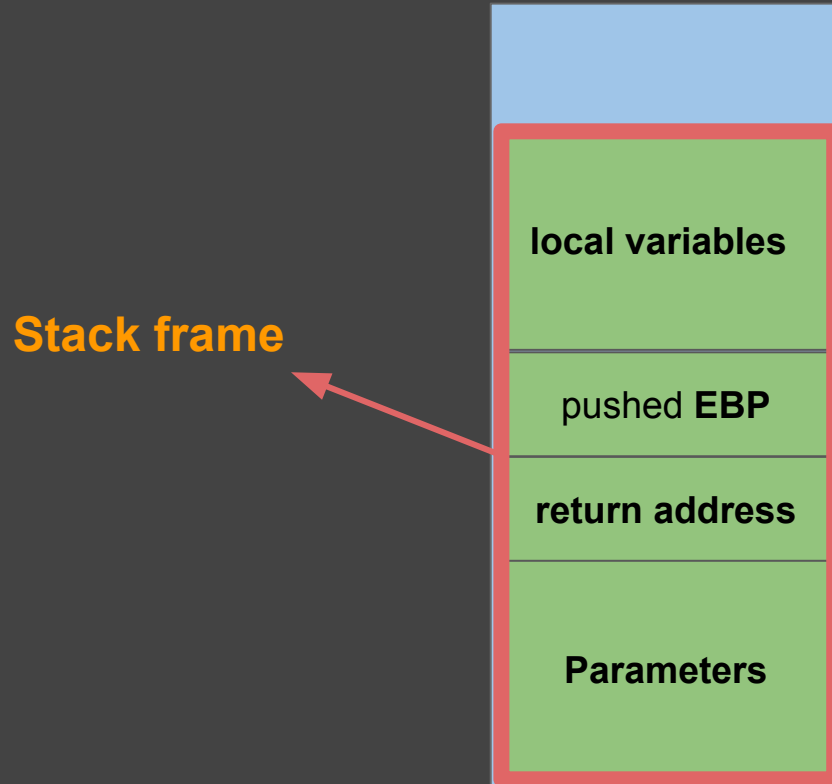
```
pop ebp
```

```
ret
```

# Implementing Local Variables



K. N. Toosi  
University of Technology



# Practice

```
#include <stdio.h>
void store_sum(int, int*);

int a;

int main() {
    store_sum(10, &a);
    printf("%d\n", a);
    return 0;
}

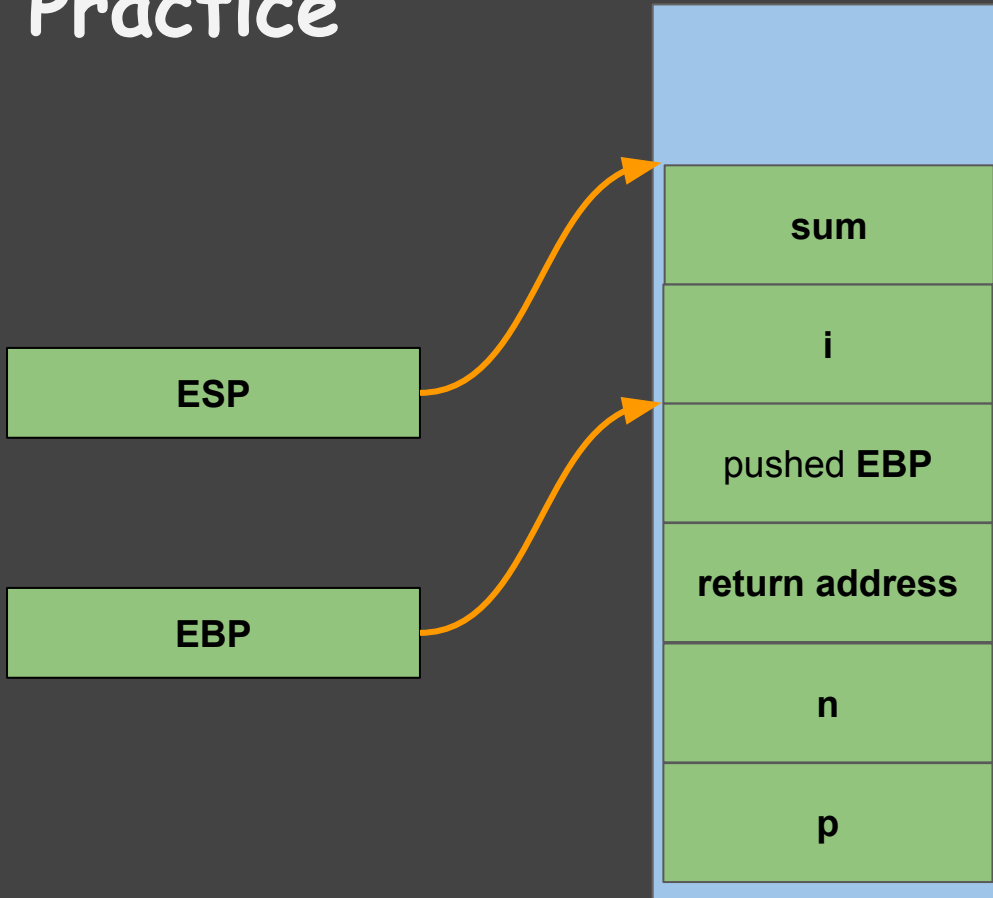
void store_sum(int n, int *p) {
    int sum, i;

    sum = 0;
    for (i = 1; i <= n; i++)
        sum += i;

    *p = sum;
}
```



# Practice



```
store_sum.c
#include <stdio.h>
void store_sum(int, int*);
int a;
int main() {
    store_sum(10, &a);
    printf("%d\n", a);
    return 0;
}

void store_sum(int n, int *p) {
    int sum, i;
    sum = 0;
    for (i = 1; i <= n; i++)
        sum += i;
    *p = sum;
}
```

# Practice



```
#include <stdio.h>
void store_sum(int, int*);

int a;

int main() {
    store_sum(10, &a);
    printf("%d\n", a);
    return 0;
}

void store_sum(int n, int *p) {
    int sum, i;

    sum = 0;
    for (i = 1; i <= n; i++)
        sum += i;

    *p = sum;
}
```

```
segment .data
a:  resd 1 ; reserve a dword

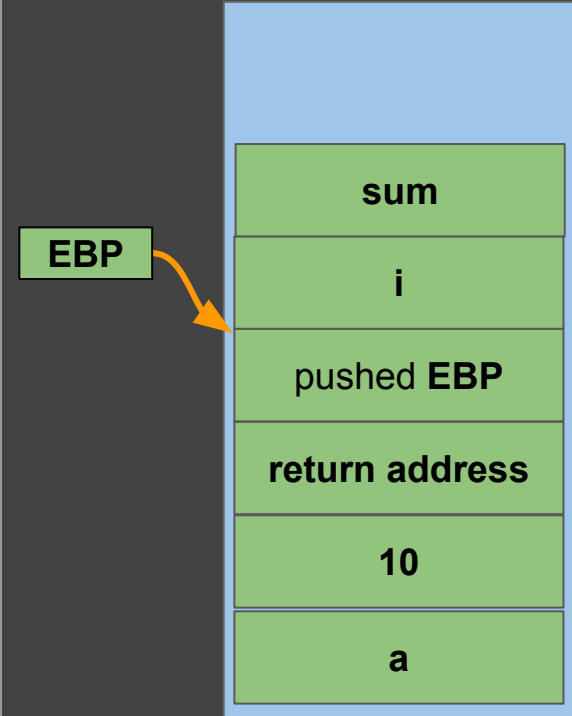
segment .text
global asm_main

asm_main:
    ;
    push a ; push the address of a
    push 10

    call store_sum

    add esp, 8

    mov eax, [a]
    call print_int
    call print_nl
    ;
    ;
```





# Practice



```
#include <stdio.h>
void store_sum(int, int*);

int a;

int main() {
    store_sum(10, &a);
    printf("%d\n", a);
    return 0;
}

void store_sum(int n, int *p) {
    int sum, i;

    sum = 0;
    for (i = 1; i <= n; i++)
        sum += i;

    *p = sum;
}
```

```
segment .bss
a:  resd 1 ; reserve a dword

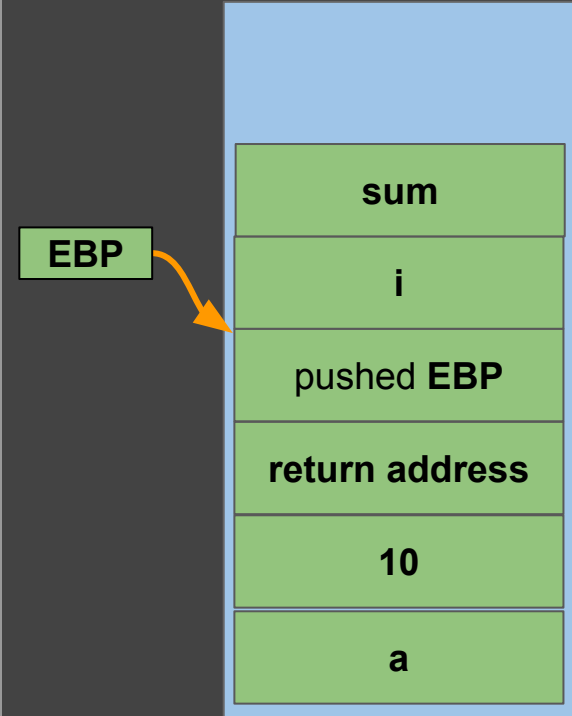
segment .text
global asm_main

asm_main:
    :
    push a ; push the address of a
    push 10

    call store_sum

    add esp, 8

    mov eax, [a]
    call print_int
    call print_nl
    :
    :
```





```
store_sum.c
#include <stdio.h>
void store_sum(int, int*);

int a;

int main() {
    store_sum(10, &a);
    printf("%d\n", a);

    return 0;
}

void store_sum(int n, int *p) {
    int sum, i;

    sum = 0;
    for (i = 1; i <= n; i++)
        sum += i;

    *p = sum;
}
```

```
store_sum.asm
store_sum:
    push ebp
    mov  ebp, esp
    sub  esp, 8      ; local variables

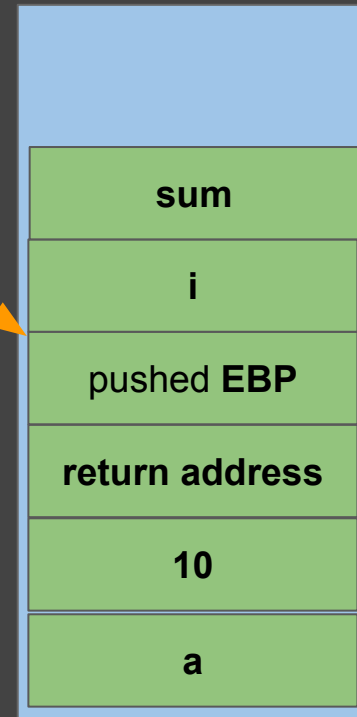
    mov  dword [ebp-8], 0      ; sum = 0
    mov  dword [ebp-4], 1      ; i = 1

forloop:
    ; if (i > n) goto endloop
    mov  eax, [ebp-4]      ; eax = i
    cmp  eax, [ebp+8]
    jg   endloop

    ;; sum = sum + i
    add  [ebp-8], eax      ; NOTE: eax == i
    inc  dword [ebp-4]     ; i++
    jmp  forloop

endloop:
    mov  ecx, [ebp+12]
    mov  eax, [ebp-8]
    mov  [ecx], eax
    mov  esp, ebp      ; release local vars
    pop  ebp
    ret
```

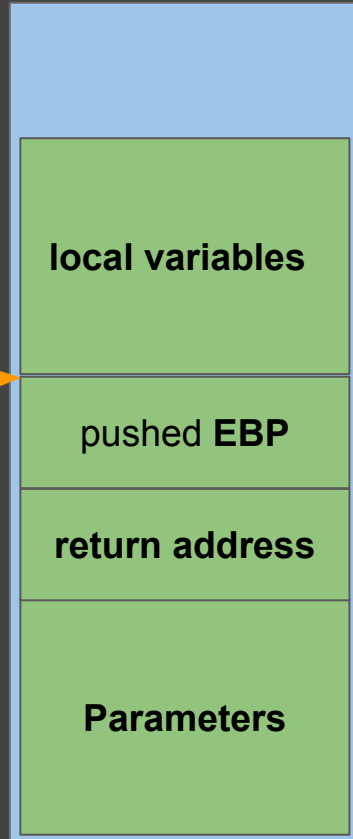
EBP



# ENTER and LEAVE instructions



K. N. Toosi  
University of Technology



```
subprogram:  
  push ebp  
  mov ebp, esp  
  sub esp, NO_OF_BYTES  
  
  ;; subprogram body  
  
  mov esp, ebp  
  pop ebp  
  ret
```

```
subprogram:  
  
  enter NO_OF_BYTES, 0  
  
  ;; subprogram body  
  
  leave  
  ret
```



## store\_sum: store\_sum.asm

```
push ebp
mov ebp, esp
sub esp, 8      ; local variables

mov dword [ebp-8], 0      ; sum = 0
mov dword [ebp-4], 1      ; i = 1

forloop:
; if (i > n) goto endloop
mov eax, [ebp-4]      ; eax = i
cmp eax, [ebp+8]
jg endloop

;; sum = sum + i
add [ebp-8], eax      ; NOTE: eax == i
inc dword [ebp-4]      ; i++
jmp forloop

endloop:
mov ecx, [ebp+12]
mov eax, [ebp-8]
mov [ecx], eax
mov esp, ebp      ; release local vars
pop ebp
ret
```

## store\_sum: store\_sum2.asm

```
enter 8,0

mov dword [ebp-8], 0      ; sum = 0
mov dword [ebp-4], 1      ; i = 1

forloop:
; if (i > n) goto endloop
mov eax, [ebp-4]      ; eax = i
cmp eax, [ebp+8]
jg endloop

;; sum = sum + i
add [ebp-8], eax      ; NOTE: eax == i
inc dword [ebp-4]      ; i++
jmp forloop

endloop:
mov ecx, [ebp+12]
mov eax, [ebp-8]
mov [ecx], eax

leave
ret
```

EBP

