המרכז
לחינוך סייבר
CYBER EDUCATION CENTER

Assembly for Reverse Engineering

Stack, Procedures

Barak Gonen

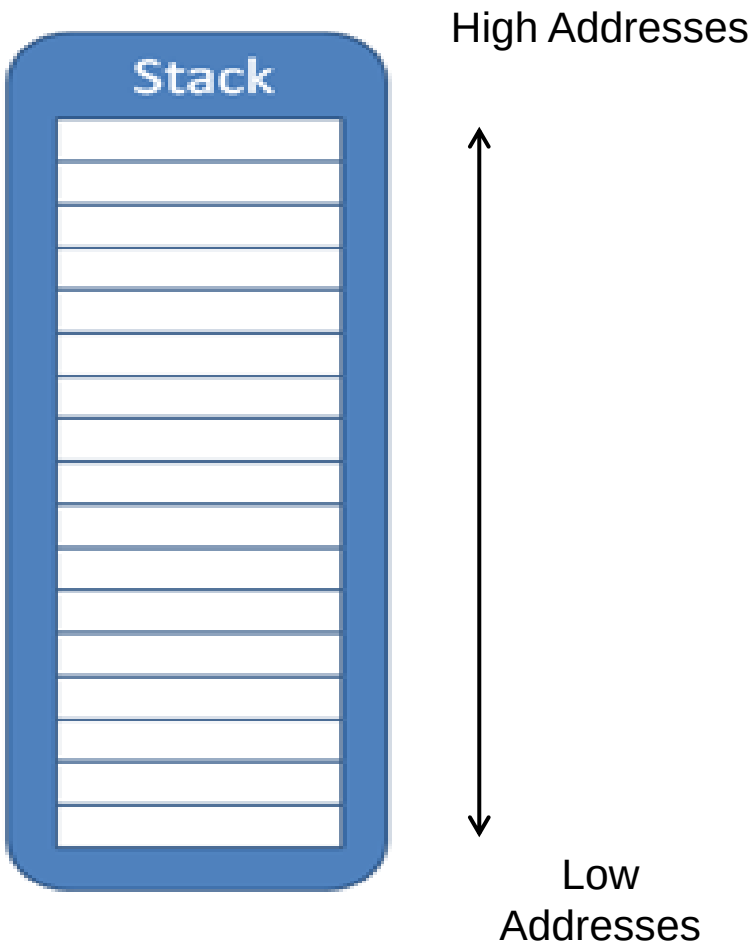
Contents

המרכז לחינוך סייבר
CYBER EDUCATION CENTER

- The STACK
  - Registers
  - Special instructions
- Procedures
  - Call / Ret
  - Passing parameters
  - EBP
  - Local variables
  - Calling conventions

The Stack

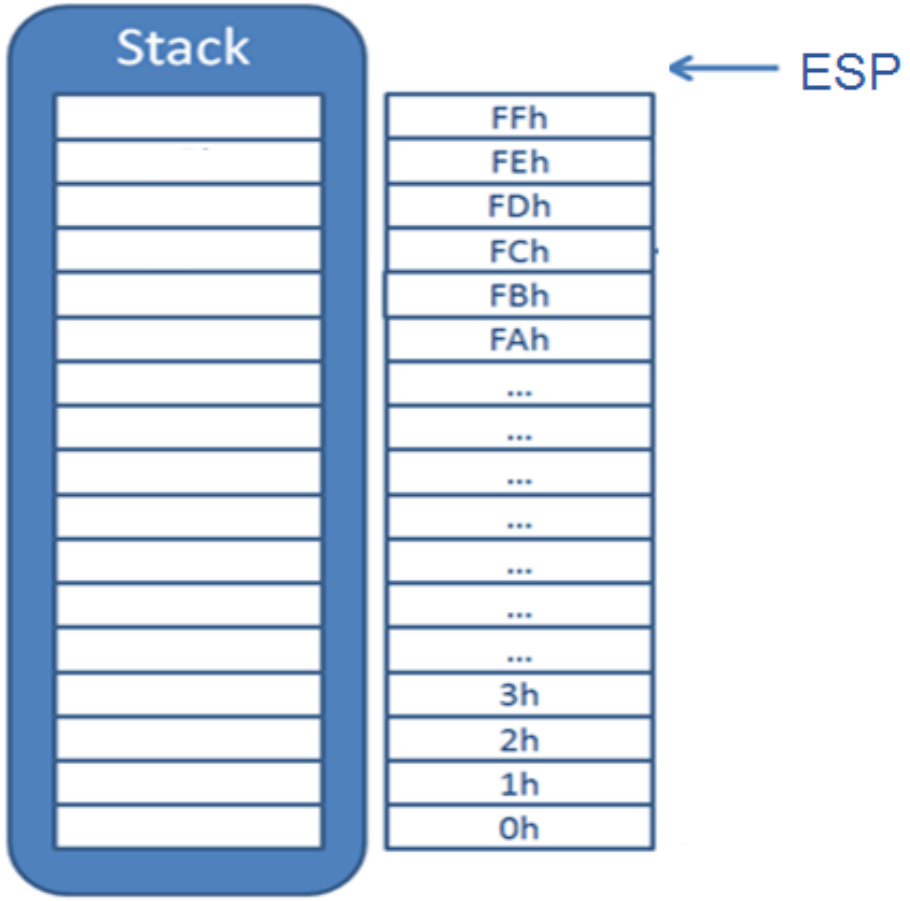
- Used for short term storage of:
  - Data
  - Addresses
- Important for understanding functions
- FASM default – 64K



Stack Register- ESP

המרכז לחינוך סייבר
CYBER EDUCATION CENTER

- ESP – (Extended) Stack Pointer.
- Initial value – stack size (yes- outside the stack)
- Example: stack size 100h



Inserting/ Extracting Stack Values

- LIFO – Last In First Out
- Before inserting value to the stack – ESP decreases
- After extracting value from the stack – ESP increases
- PUSH inserts value
- POP extracts value

PUSH

המרכז לחינוך סייבר
CYBER EDUCATION CENTER

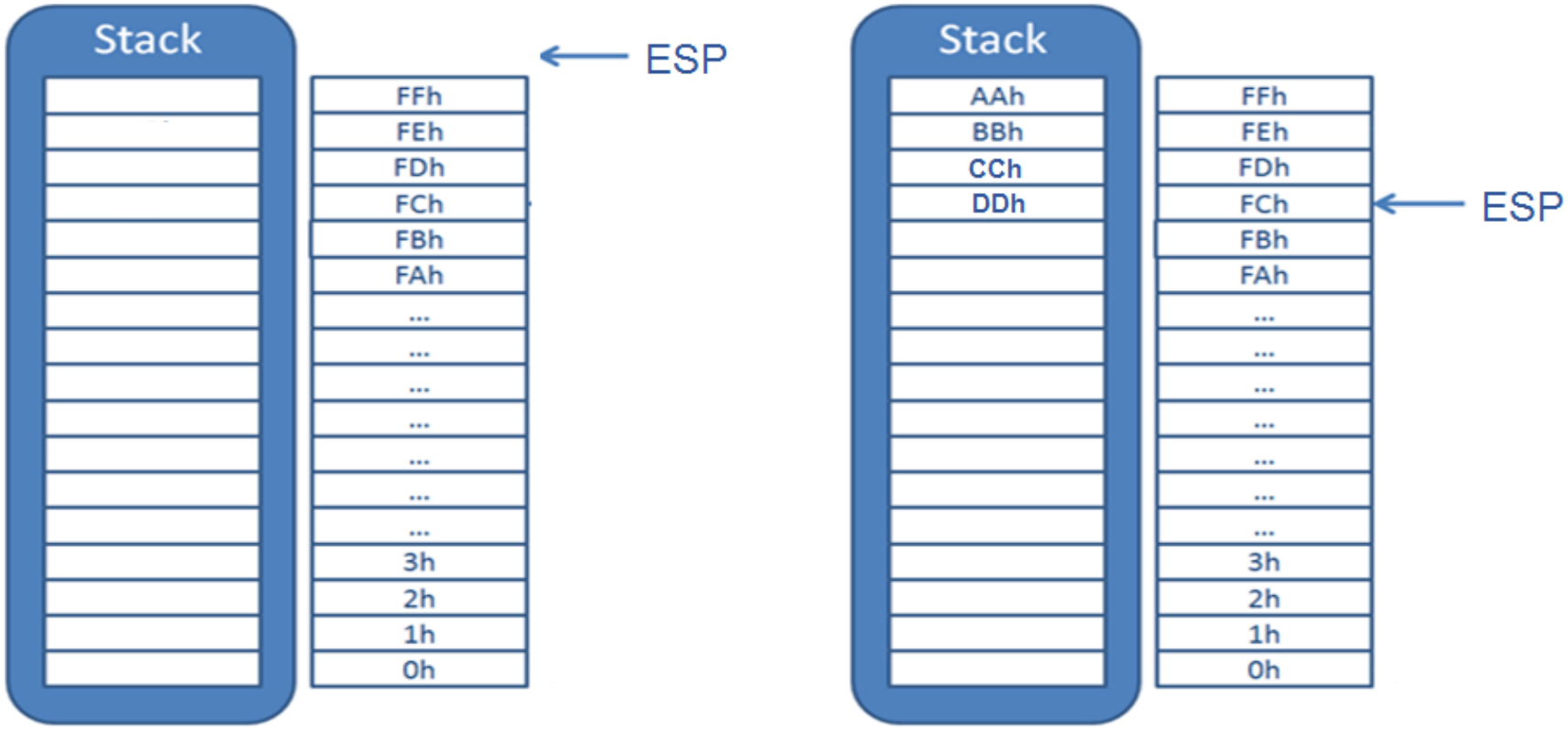
- ESP is reduced by operand size
  - 4 or 2
  - 1 – not possible
- Value is stored in [ESP]

push operand

PUSH - Example

המרכז לחינוך סייבר
CYBER EDUCATION CENTER

```
mov     eax, 0xAABBCCDD
push    eax
```



POP

- Value from [ESP] is copied to operand
- ESP is increased by operand size
  - 4 or 2

pop operand

Practice

- Copy EAX to EBX without using MOV
- Add instructions so EAX == 4, EBX == 5

```
mov     eax, 4
mov     ebx, 5
push    eax
push    ebx
xor     eax, eax
xor     ebx, ebx
; add code to restore eax, ebx
```

Procedures

המרכז לחינוך סייבר
CYBER EDUCATION CENTER

- CALL, RET- what do they do?
- Self study
  - Note- debug with “Step Into”
  - Monitor EIP, ESP

```
start:
    mov     eax, 4
    mov     ebx, 5
    mov     ecx, 6
    mov     edx, 7
    call    zero_regs
    push    0
    call    [ExitProcess]

zero_regs:
    xor     eax, eax
    xor     ebx, ebx
    xor     ecx, ecx
    xor     edx, edx
    ret
```

CALL- RET

המרכז לחינוך סייבר
CYBER EDUCATION CENTER

- CALL:
  - Decrease ESP
  - Copy the return address to the stack
  - Change EIP to start of procedure
  - Jump to EIP
- RET:
  - Read the return address, copy to EIP
  - Increase ESP
  - Jump to EIP

Class Excercise

- Mul.asm
- Fib.asm

Passing Parameters

- Suppose we have a variable:

```
section '.data' data readable writeable
    I   dd  5
```

- Memory:

Address	Hex dump
00401000	05 00 00 00

- We can pass the variable to a procedure:
  - By value
  - By reference

Passing Parameters

Pass by value

- Push value

```
push      [I]
```

- Stack:

```
00CFF88|| 00000005
```

- Procedure may not change parameter

Pass by reference

- Push address

```
push      I
```

- Stack:

```
00CFF88|| 00401000
```

- Procedure may change parameter

Passing Parameters

- The procedure ‘simple’ accepts I,J,K as params, and returns EAX = I + J - K

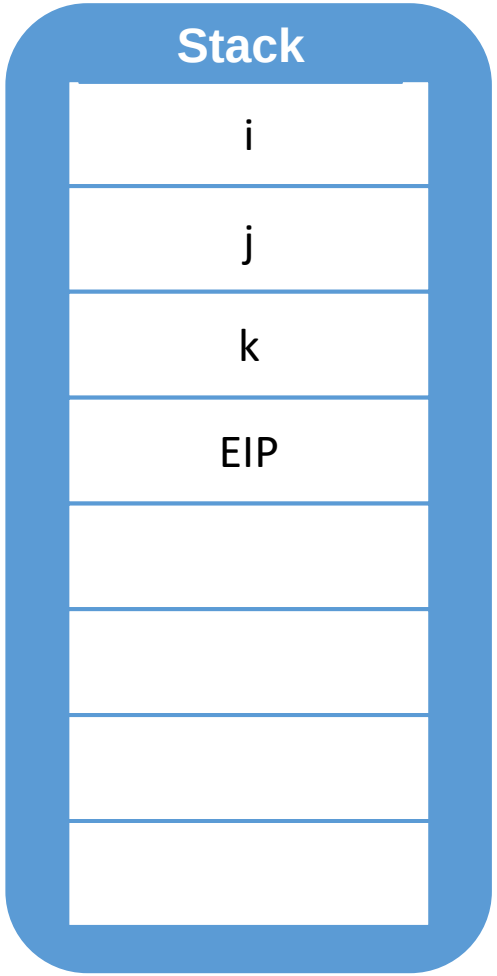
```
section '.data' data readable writeable
    I   dd  5
    J   dd  6
    K   dd  7
section '.text' code readable executable

start:
    push    [I]
    push    [J]
    push    [K]
    call    simple
    push    0
    call    [ExitProcess]
```

- Can you write ‘simple’?

Accessing Parameters

- How can we access I, J, K?
  - The first POP will extract the return address
  - RET will no longer work…

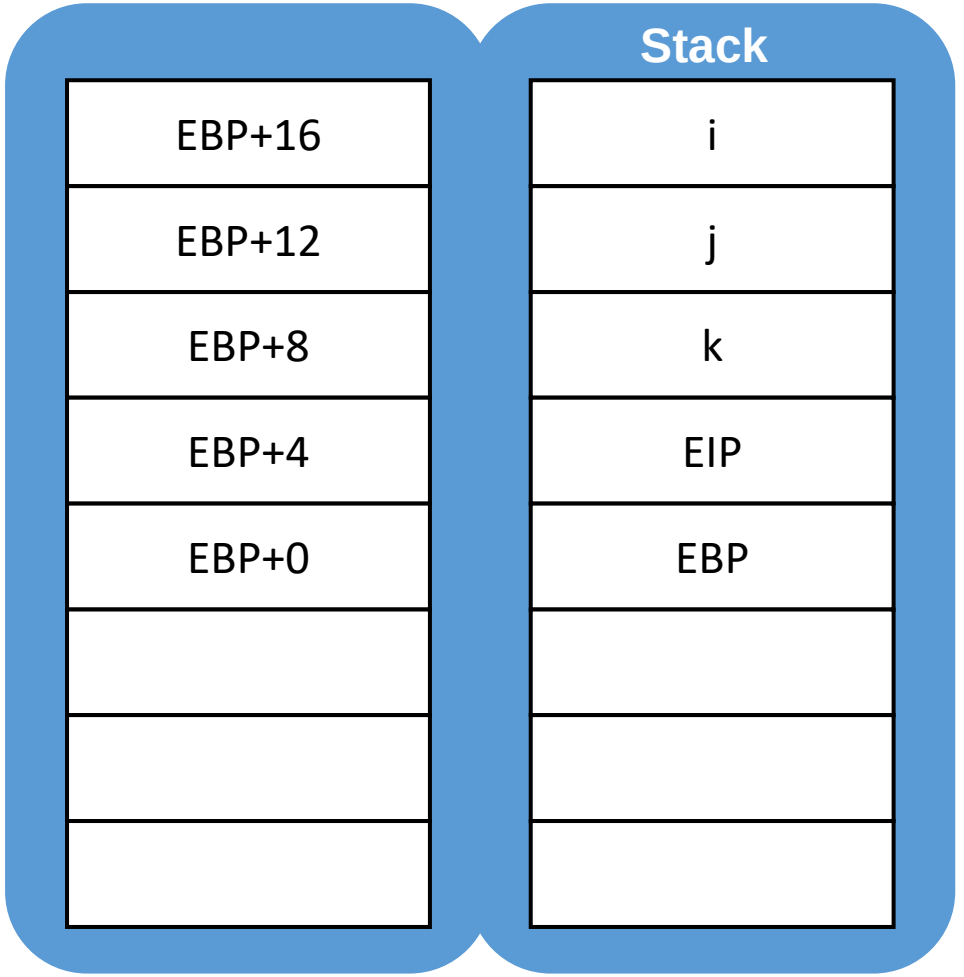


EBP

- EBP – (Extended) Base Pointer
- Assists accessing values on the stack

```
; eax = I+J-k
push    ebp
mov     ebp, esp
; ...
pop     ebp
ret     12
```

- How does it help?



EBP- cont.

- Our code can now be written with relative addresses:

```
simple:
    ; eax = I+J-k
    push    ebp
    mov     ebp, esp
    mov     eax, [ebp+16]
    add     eax, [ebp+12]
    sub     eax, [ebp+8]
    pop     ebp
    ret     12
```

RET + Const

- Think: Why have we used RET 12?
- RET + Const:
  - ESP = ESP+Const
  - RET
- Why 12?
  - We pushed 12 bytes (3 x 32 bit)

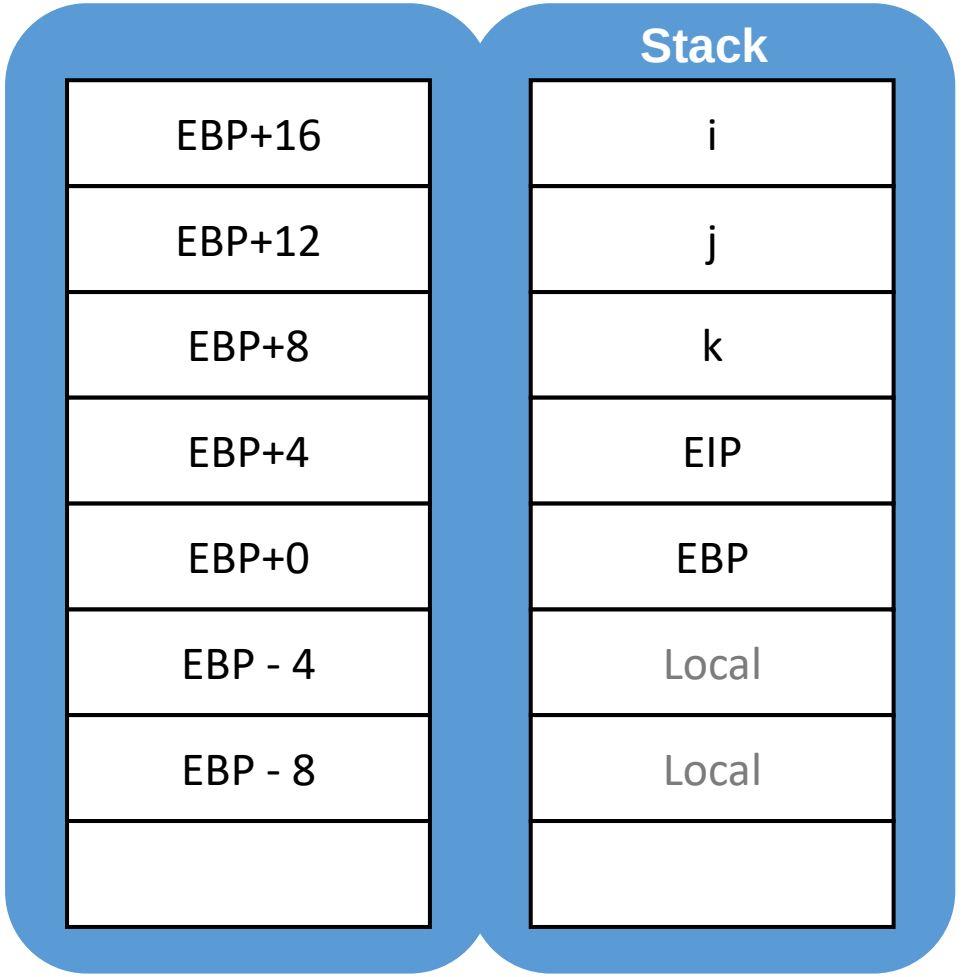
Local Variables

- A function may have local variables
- Stored on the stack
- Any memory reserved on the stack – reduces ESP
  - Reserve 8 bytes: sub ESP, 8
  - Free reserved memory: add ESP, 8

Local Variables – cont.

המרכז לחינוך סייבר
CYBER EDUCATION CENTER

- Accessing locals- easy and fun with EBP :-)
- Don’t forget to free memory before RET
  - Or else, program might crash…



Class Excercise

- Fibon.asm

LEAVE

Equivalent to:

```
mov     esp, ebp
pop     ebp
```

What is the purpose?

Clear local vars & restore ebp with one opcode

Calling Conventions

- Caller – the code that calls the function
- Callee – the function which is called
- Must coordinate between caller & callee:
  - Order of passing parameters to the stack
  - Register to store return value
  - “Cleaning” the stack

Calling Conventions- cont.

- Consider the C function declaration:

  int MyFunc(int a, int b)

- And the following code:

  int result = MyFunc(1,2)

- How will the code be assembled?

```
push    1
push    2
call    MyFunc
```

```
push    2
push    1
call    MyFunc
```

Calling Conventions –cont.

- Cleaning the stack:

```
; callee cleans stack
push    1
push    2
call    MyProc
; MyProc has RET 8
```

```
; caller cleans stack
push    1
push    2
call    MyProc
add     esp, 8
```

CDCEL / STDCALL/FASTCALL

	CDECL	STDCALL	FASTCALL
Passing Params	Right to Left	Right to Left	Left to Right*
Return reg.	EAX	EAX	EAX
Clean the stack	Caller	Callee	Callee

* First 2 arguments, the rest are right to left

Translation to Assembly

המרכז לחינוך סייבר
CYBER EDUCATION CENTER

```
; CDECL: passing params right to left
push    2
push    1
call    MyProc
; caller cleans the stack
add     esp, 8
```

```
; STDCALL: passing params right to left
push    2
push    1
call    MyProc
; callee cleans the stack
```

```
; FASTCALL: passing params left to right, ECX & EDX
mov     ecx, 1
mov     edx, 2
call    MyProc
```

Advantages

- CDECL advantage- number of params is flexible
  - Caller knows and cleans stack
  - Example- print
- STDCALL advantage – faster
  - Callee has RET N
  - Saves one instruction (ADD ESP, N)
- FASTCALL even faster
  - Registers are faster than memory (stack)

ExitProcess

- Self study ExitProcess only through MSDN
  - What is the input parameter?
  - Which DLL file should be included?

Including DLL's

המרכז לחינוך סייבר
CYBER EDUCATION CENTER

```
section '.idata' import data readable

library kernel,'kernel32.dll'

import   kernel,\
         ExitProcess,'ExitProcess'
```

Set section

DLL to be imported

Function name

Class Exercise – call PRINTF

המרכז לחינוך סייבר
CYBER EDUCATION CENTER

- Print ‘My name is %s my age is %d’

Exercise

- Add the following functionality:
  - A console message ‘Please enter text’ (printf)
  - Calculate length of user text (strlen)
  - Print the user text and it’s length (printf)

```
Please enter text
Hello Cool Cyber Class!
User entered: Hello Cool Cyber Class!
Length: 23
```

Done :-)

המרכז לחינוך סייבר
CYBER EDUCATION CENTER

```
push class
call great_work
```


QUESTIONS