REVERSE ENGINEERING – CLASS 0x01

ASSEMBLY X64 CRASH COURSE

Cristian Rusu

RECAP

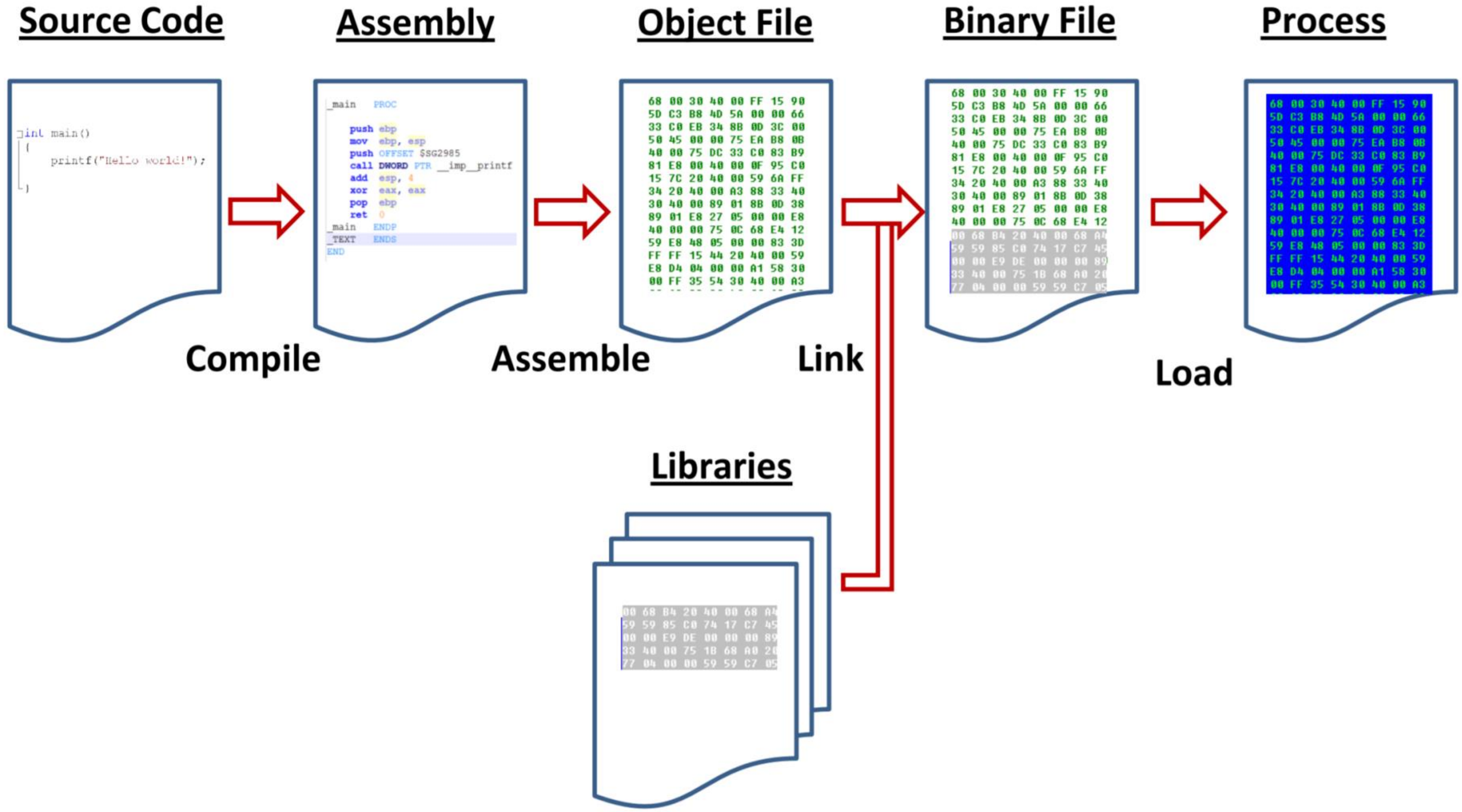
- **black-box analysis of binaries (Lab session 0x01)**
  - only interactions of the binary with other binaries, libraries, SO
- **white-box analysis of binaries**
  - assembly code analysis
- **gray-box analysis of binaries**
  - a combination of the two above

TABLE OF CONTENTS

- compilation process
- assembly x64
- machine code
- examples

FROM SOURCE CODE TO EXECUTION

- în general (nu doar pentru Assembly)



https://slideplayer.com/slide/4695781/

FROM SOURCE CODE TO EXECUTION

source code: main.c

```
#include <stdio.h>

int main()
{
        printf("hello\n");
        return 42;
}
```

gcc -S -o main.asm main.c

source code, assembly main.s

```
.LC0:
        .string "hello"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        leaq    .LC0(%rip), %rdi
        call    puts@PLT
        movl    $42, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

gcc -o main main.c

machine code, main (hexdump)

```
0000000 457f 464c 0102 0001 0000 0000 0000 0000
0000010 0003 003e 0001 0000 1060 0000 0000 0000
0000020 0040 0000 0000 0000 3978 0000 0000 0000
0000030 0000 0000 0040 0038 000d 0040 001f 001e
0000040 0006 0000 0004 0000 0040 0000 0000 0000
0000050 0040 0000 0000 0000 0040 0000 0000 0000
0000060 02d8 0000 0000 0000 02d8 0000 0000 0000
```

.

AN EXAMPLE

- **objdump –d checklicense**

```
0000000000000740 <main>:
 740:	55                   	push   %rbp
 741:	48 89 e5             	mov    %rsp,%rbp
 744:	48 83 ec 10          	sub    $0x10,%rsp
 748:	89 7d fc             	mov    %edi,-0x4(%rbp)
 74b:	48 89 75 f0          	mov    %rsi,-0x10(%rbp)
 74f:	83 7d fc 02          	cmpl   $0x2,-0x4(%rbp)
 753:	75 59                	jne    7ae <main+0x6e>
 755:	48 8b 45 f0          	mov    -0x10(%rbp),%rax
 759:	48 83 c0 08          	add    $0x8,%rax
 75d:	48 8b 00             	mov    (%rax),%rax
 760:	48 89 c6             	mov    %rax,%rsi
 763:	48 8d 3d ea 00 00 00 	lea    0xea(%rip),%rdi        # 854 <_IO_stdin_used+0x4>
 76a:	b8 00 00 00 00       	mov    $0x0,%eax
 76f:	e8 6c fe ff ff       	callq  5e0 <printf@plt>
 774:	48 8b 45 f0          	mov    -0x10(%rbp),%rax
 778:	48 83 c0 08          	add    $0x8,%rax
 77c:	48 8b 00             	mov    (%rax),%rax
 77f:	48 8d 35 ec 00 00 00 	lea    0xec(%rip),%rsi        # 872 <_IO_stdin_used+0x22>
 786:	48 89 c7             	mov    %rax,%rdi
 789:	e8 62 fe ff ff       	callq  5f0 <strcmp@plt>
 78e:	85 c0                	test   %eax,%eax
 790:	75 0e                	jne    7a0 <main+0x60>
 792:	48 8d 3d e2 00 00 00 	lea    0xe2(%rip),%rdi        # 87b <_IO_stdin_used+0x2b>
 799:	e8 32 fe ff ff       	callq  5d0 <puts@plt>
 79e:	eb 1a                	jmp    7ba <main+0x7a>
 7a0:	48 8d 3d e4 00 00 00 	lea    0xe4(%rip),%rdi        # 88b <_IO_stdin_used+0x3b>
 7a7:	e8 24 fe ff ff       	callq  5d0 <puts@plt>
 7ac:	eb 0c                	jmp    7ba <main+0x7a>
 7ae:	48 8d 3d e4 00 00 00 	lea    0xe4(%rip),%rdi        # 899 <_IO_stdin_used+0x49>
 7b5:	e8 16 fe ff ff       	callq  5d0 <puts@plt>
 7ba:	b8 00 00 00 00       	mov    $0x0,%eax
 7bf:	c9                   	leaveq
 7c0:	c3                   	retq
 7c1:	66 2e 0f 1f 84 00 00 	nopw   %cs:0x0(%rax,%rax,1)
 7c8:	00 00 00
 7cb:	0f 1f 44 00 00       	nopl   0x0(%rax,%rax,1)
```

AN EXAMPLE

- **hexeditor checklicense**

```
00000790  75 0E 48 8D  3D E2 00 00   00 E8 32 FE  FF FF EB 1A   u.H.=.....2.....
000007A0  48 8D 3D E4  00 00 00 E8   24 FE FF FF  EB 0C 48 8D   H.=.....$.....H.
000007B0  3D E4 00 00  00 E8 16 FE   FF FF B8 00  00 00 00 C9   =...............
000007C0  C3 66 2E 0F  1F 84 00 00   00 00 00 0F  1F 44 00 00   .f..........D..
000007D0  41 57 41 56  41 89 FF 41   55 41 54 4C  8D 25 F6 05   AWAVA..AUATL.%..
000007E0  20 00 55 48  8D 2D F6 05   20 00 53 49  89 F6 49 89    .UH.-.. .SI..I.
000007F0  D5 4C 29 E5  48 83 EC 08   48 C1 FD 03  E8 9F FD FF   .L).H...H.......
00000800  FF 48 85 ED  74 20 31 DB   0F 1F 84 00  00 00 00 00   .H..t 1.........
00000810  4C 89 EA 4C  89 F6 44 89   FF 41 FF 14  DC 48 83 C3   L..L..D..A...H..
00000820  01 48 39 DD  75 EA 48 83   C4 08 5B 5D  41 5C 41 5D   .H9.u.H...[]A\A]
00000830  41 5E 41 5F  C3 90 66 2E   0F 1F 84 00  00 00 00 00   A^A_..f.........
00000840  F3 C3 00 00  48 83 EC 08   48 83 C4 08  C3 00 00 00   ....H...H.......
00000850  01 00 02 00  43 68 65 63   6B 69 6E 67  20 74 68 65   ....Checking the
00000860  20 6C 69 63  65 6E 73 65   3A 20 25 73  20 2E 2E 2E    license: %s ...
00000870  0A 00 41 42  43 44 45 46   47 48 00 41  63 63 65 73   ..ABCDEFGH.Acces
00000880  73 20 67 72  61 6E 74 65   64 21 00 41  63 63 65 73   s granted!.Acces
00000890  73 20 64 65  6E 69 65 64   00 55 73 61  67 65 3A 20   s denied.Usage: 
000008A0  3C 6B 65 79  3E 00 00 00   01 1B 03 3B  3C 00 00 00   <key>......;<...
000008B0  06 00 00 00  18 FD FF FF   88 00 00 00  58 FD FF FF   ............X...
000008C0  B0 00 00 00  68 FD FF FF   58 00 00 00  98 FE FF FF   ....h...X.......
000008D0  C8 00 00 00  28 FF FF FF   E8 00 00 00  98 FF FF FF   ....(...........
000008E0  30 01 00 00  00 00 00 00   14 00 00 00  00 00 00 00   0...............
000008F0  01 7A 52 00  01 78 10 01   1B 0C 07 08  90 01 07 10   .zR..x..........
00000900  14 00 00 00  1C 00 00 00   08 FD FF FF  2B 00 00 00   ............+...
00000910  00 00 00 00  00 00 00 00   14 00 00 00  00 00 00 00   ................
00000920  01 7A 52 00  01 78 10 01   1B 0C 07 08  90 01 00 00   .zR..x..........
00000930  24 00 00 00  1C 00 00 00   88 FC FF FF  40 00 00 00   $...........@...
00000940  00 0E 10 46  0E 18 4A 0F   0B 77 08 80  00 3F 1A 3B   ...F..J..w...?.;
00000950  2A 33 24 22  00 00 00 00   14 00 00 00  44 00 00 00   *3$"........D...
00000960  A0 FC FF FF  08 00 00 00   00 00 00 00  00 00 00 00   ................
00000970  1C 00 00 00  5C 00 00 00   C8 FD FF FF  81 00 00 00   ....\...........
00000980  00 41 0E 10  86 02 43 0D   06 02 7C 0C  07 08 00 00   .A....C...|.....
00000990  44 00 00 00  7C 00 00 00   38 FE FF FF  65 00 00 00   D...|...8...e...
000009A0  00 42 0E 10  8F 02 42 0E   18 8E 03 45  0E 20 8D 04   .B....B....E. ..
000009B0  42 0E 28 8C  05 48 0E 30   86 06 48 0E  38 83 07 4D   B.(..H.0..H.8..M
000009C0  0E 40 72 0E  38 41 0E 30   41 0E 28 42  0E 20 42 0E   .@r.8A.0A.(B. B.
000009D0  18 42 0E 10  42 0E 08 00   14 00 00 00  C4 00 00 00   .B..B...........
000009E0  60 FE FF FF  02 00 00 00   00 00 00 00  00 00 00 00   `...............
000009F0  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00   ................
00000A00  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00   ................
00000A10  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00   ................
00000A20  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00   ................
00000A30  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00   ................
00000A40  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00   ................
^G Help  ^C Exit (No Save)  ^T goTo Offset  ^X Exit and Save  ^W Search  ^U Undo  ^L Redraw  ^E Text Mode
```

Intel 64 and IA-32 Architectures Software Developer Manuals,
http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

BINARY FILES

- **contain the machine code (not assembly)**
  - assembly = readable machine code
- **also headers and other information for the SO**
  - ELF
  - PE
  - WASM
- **(many) more details on binary files in the next class**

.

ASSEMBLY CRASH COURSE

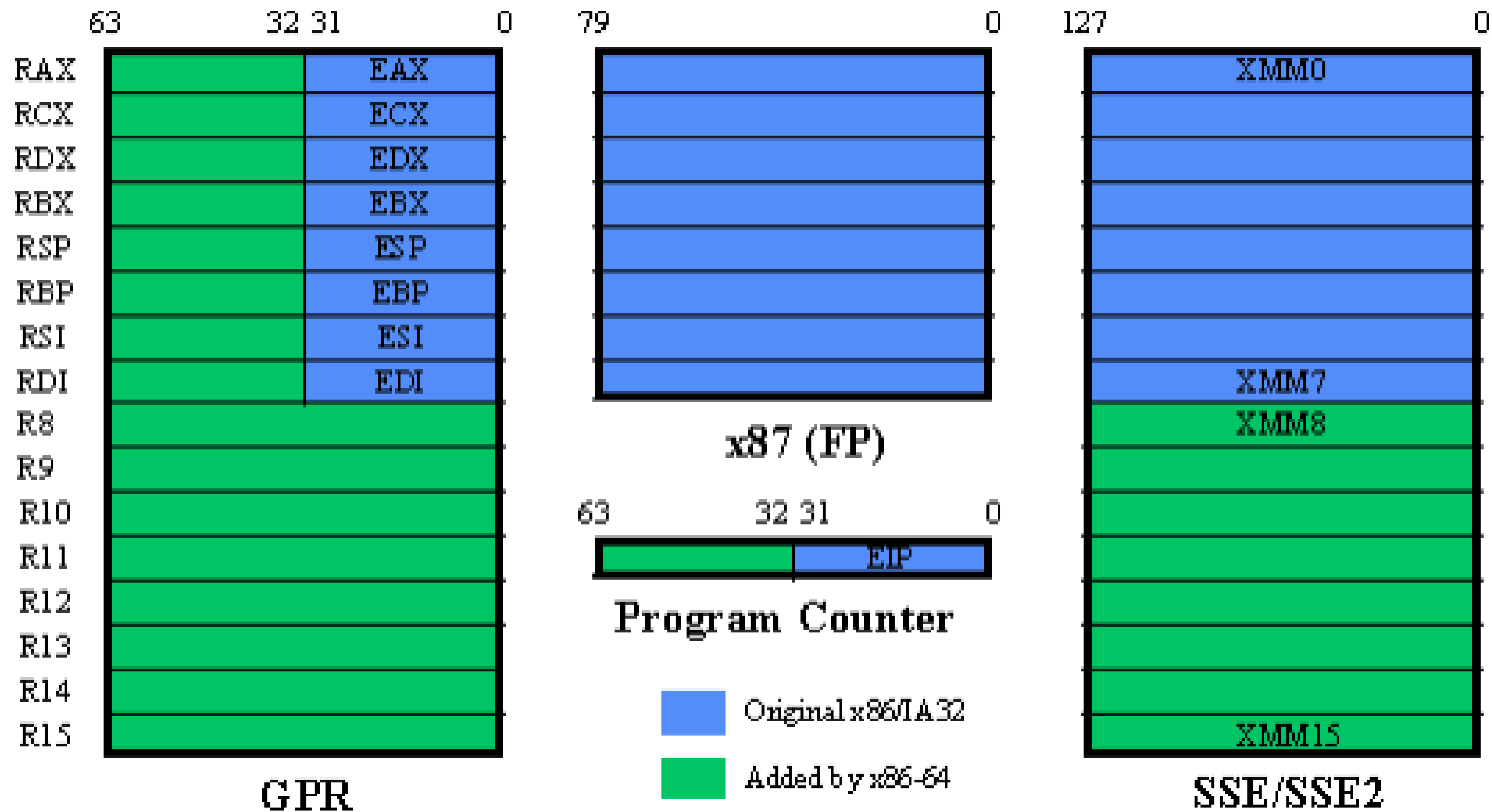
- **the CPU consumes only machine code**
- **assembly = readable machine code**
  - but this is only for the sake of humans
- **assembly advantages**
  - produces fast code (no overhead)
  - fined grained control (cannot get any finer than this)
  - understand what the CPU/compiler does
- **assembly disadvantages**
  - in the beginning, it is hell on earth to write assembly code
  - steep learning curve, low productivity
  - hard to maintain large assembly repos
  - compiler may generate „better” code (it often does)

ASSEMBLY CRASH COURSE

- **registers**
- **instructions**
  - arithmetic
  - logic
  - memory access
  - control flow
- **flags**
- **the stack**
- **interrupts**

ACC: REGISTERS

- like the „variables” in your code



- RIP: Instruction Pointer
- RSP: Stack Pointer; RBP: Base Pointer
- RDI, RSI: for arrays

https://www.realworldtech.com/battle64-2003/4/

ACC: ARITHMETIC AND LOGIC

- MOV RAX, 2021 ; rax = 2021
- SUB RAX, RDX ; rax -= rdx
- AND RCX, RBX ; rcx &= rbx
- SHL RAX, 10 ; rax <<= 10
- SHR RAX, 10 ; rax >>= 10 (sign bit not preserved)
- SAR RAX, 10 ; rax >>= 10 (sign bit preserved)
- IMUL RAX, RCX ; rax = rax * rcx
- IMUL RCX ; <rdx:rax> = rax * rcx (128 bit mul)
- XOR RAX, RAX ; rax ^= rax
- LEA RCX, [RAX * 8 + RBX] ; rcx = rax * 8 + rb

ACC: MEMORY ACCESS

- MOV RAX, QWORD PTR [0x123456] ; rax = *(int64_t*) 0x123456
- MOV QWORD PTR [0x123456], RAX ; *(int64_t*) 0x123456 = rax

- MOV EAX, DWORD PTR [0x123456] ; rax = *(int32_t*) 0x123456
- MOV AL, BYTE PTR [0x123456] ; al = *(int8_t*) 0x123456

.

ACC: CONTROL FLOW

- JMP 0x1234 ; rip = 0x1234
- JMP [RAX] ; rip = *(int64_t) rax

- JZ/JE 0xABCD ; if (zf) rip = 0xabcd
- JNZ/JNE 0xABCD ; if (!zf) rip = 0xabcd

.

ACC: EFLAGS

- **Carry : Addition, Subtraction**
- **Zero: Last operation result was 0**
- **Sign : Last operation result was < 0**
- **Over: Last operation result was > $2^{\text{bit count}} - 1$**

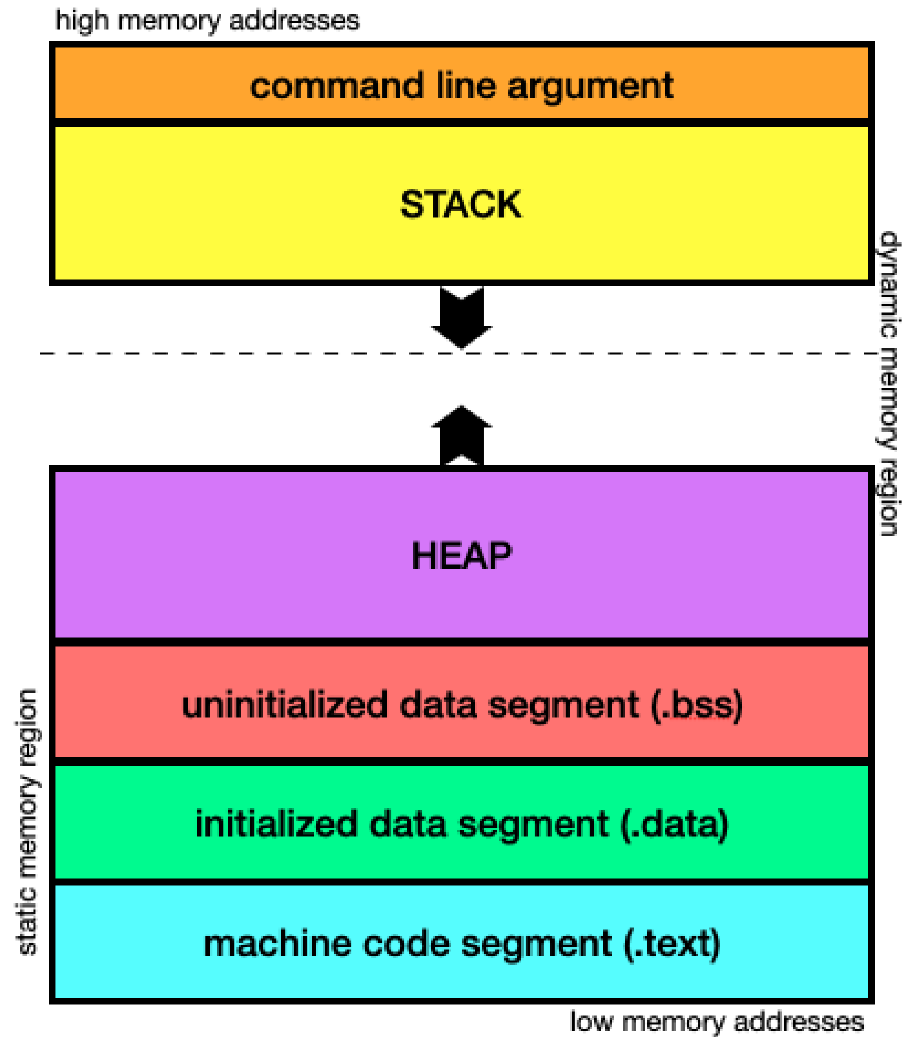
```
TEST RAX, RBX        ; _ = rax & rbx; set SF, ZF, PF
                     ; useful when checking for null vals
                     ; and bit masks

CMP RAX, RBX         ; _ = rax - rbx
                     ; arithmetic comparisons
```

.

ACC: THE STACK

- **the memory space of a program**



ACC: THE STACK

```
• PUSH RAX                 ; rsp -= 8; *(int64_t*)rsp = rax;
• POP RAX                  ; rax = *(int64_t*)rsp; rsp += 8
• CALL 0x12345             ; PUSH RIP; JMP 0x12345
• RET                      ; POP RIP, return value is in RAX


• PUSH RBP                 ; save previous frame base
• MOV RBP, RSP             ; move frame base to current top
• SUB RSP, 100             ; allocate 100 bytes on the stack
                           ; "push new stack frame"
• MOV RBX, [RBP - 0x20]    ; rbx = *(int64_t*)(rbp-0x20)
                           ; use the allocated space for storage
• LEAVE                    ; MOV RSP, RBP ; POP RBP
                           ; "pop current stack frame"
```

.

ACC: SYSCALLS

- **many syscalls**
  - execve,exit
  - file operations: open, close, read, write, delete
  - allocate/release memory (HEAP)
  - sockets
  - IPC

```
MOV RAX, 0x2                ; Choose syscall number 2 (open)
MOV RDI, [RSP + 0x10]       ; Set first argument to some stack value
SYSCALL                     ; Invoke kernel functionality
```

.

CONCLUSIONS

- assembly is hard
- you do not need to become an expert
- you need to be able to read assembly code, not write it
- absolutely crucial for RE

REFERENCES

- https://cs.unibuc.ro/~crusu/asc/labs.html
  - but this is 32 bit assembly, and it is at&t syntax

- **Binary Exploitation, https://www.youtube.com/watch?v=iyAyN3GFM7A&list=PLhixgUqwRTjxglIswKp9mpkfPNfHkzyeN , videos 0x00 - 0x04**

- **x64 Assembly and C++ Tutorial, https://www.youtube.com/playlist?list=PL0C5C980A28FEE68D**

- **Linux x64 Assembly Tutorial, https://www.youtube.com/playlist?list=PLKK11Ligqiti8g3gWRtMjMgf1KoKDOvME**