x86 & x86_64

Desde el principio

José María Foces Vivancos

Introducción

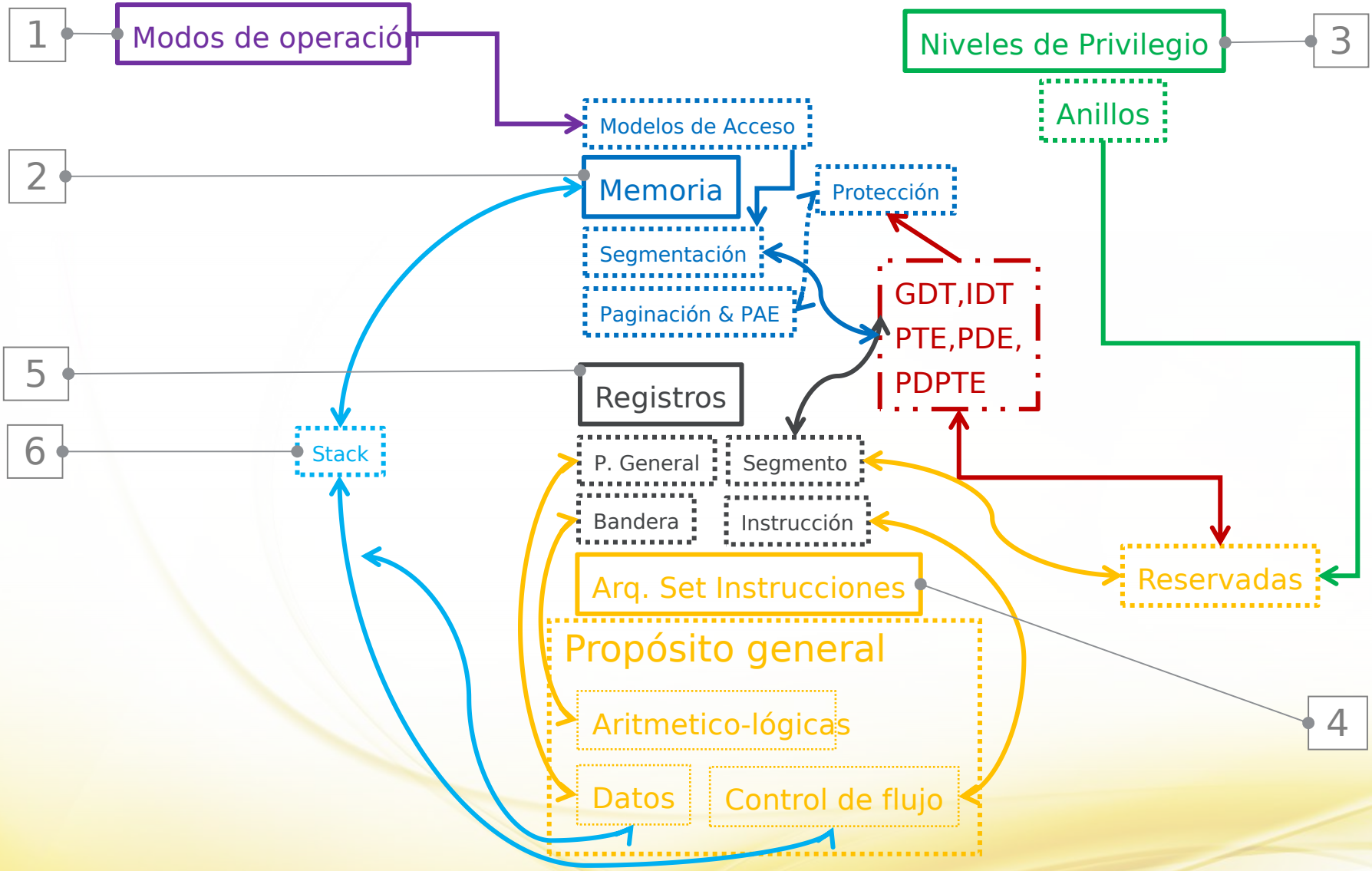
- Objetivo: refrescar conocimientos sobre x86 & x86_64 para soportar una buena comprensión de la explotación software a bajo nivel.
- Formato de referencias:
  - IDM: Intel Developer's Manual Intel 64 & IA-32.

IDM Vol.<V> ...

- Entorno:
  - Hardware: PC x86 / x86_64
  - Debian Linux (9)
  - GNU Binutils

Índice

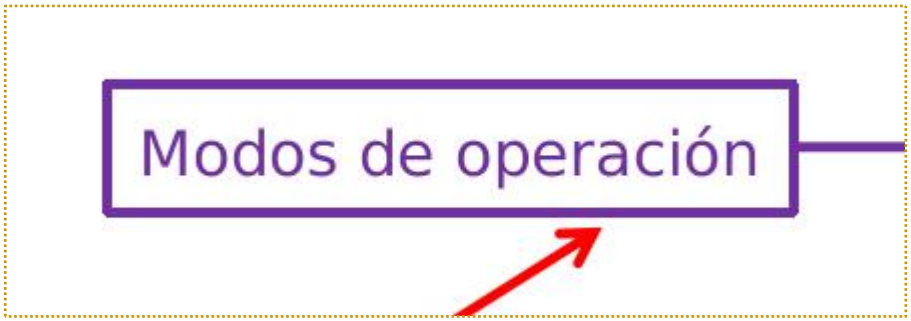
Conocimiento Base: Mapa Conceptual



Modos de operación

- Real-Address (BIOS)
- System Management
- Protegido (32-bit)
  - Virtual 8086
- IA-32e
  - Compatibility
  - 64-bit

IDM Vol. 1 3.1



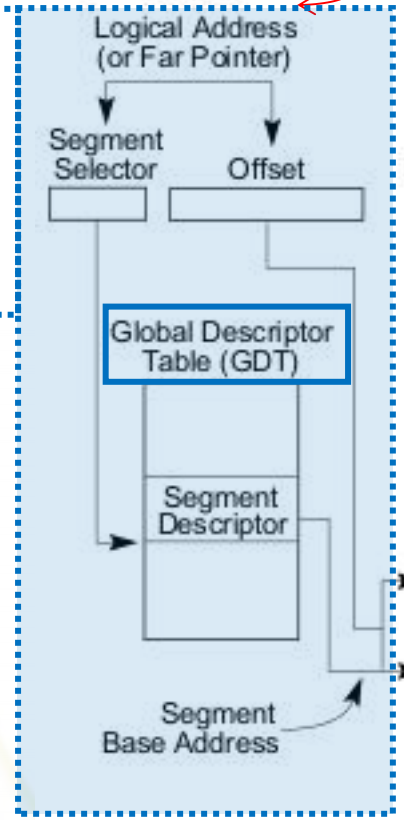
Memoria: Subsistemas (MMU)- GDT,IDT,PTE,PDE,PDPTE

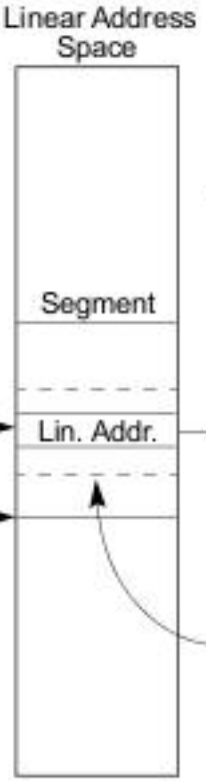
Programa: Quiero el dato 0xfffdd000

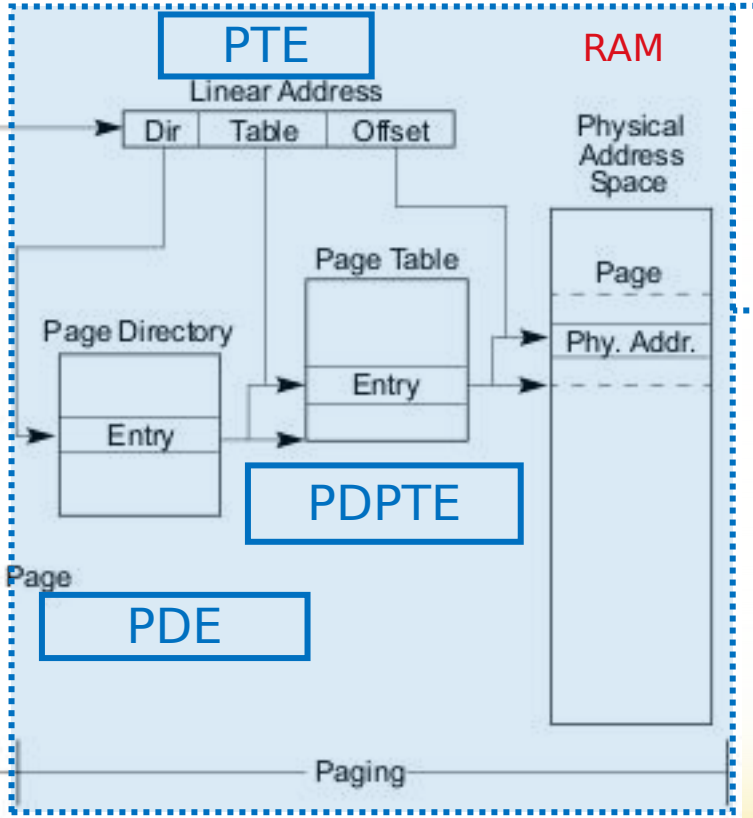
https://github.com/cirosantilli/x86-bare-metal-examples/

IDM Vol.3A Figure 3-1

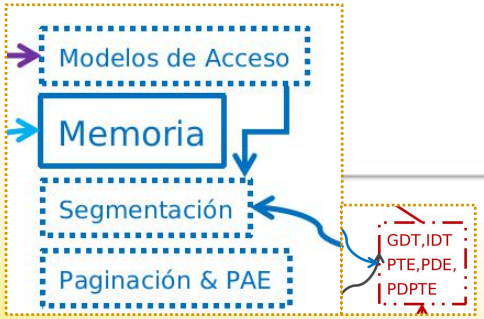
- Allocation of variable length memory regions AKA (segments)







- Allocation of constant memory regions AKA Pages (4k)
- Performance
- Management
- Virtualization



Segmentation

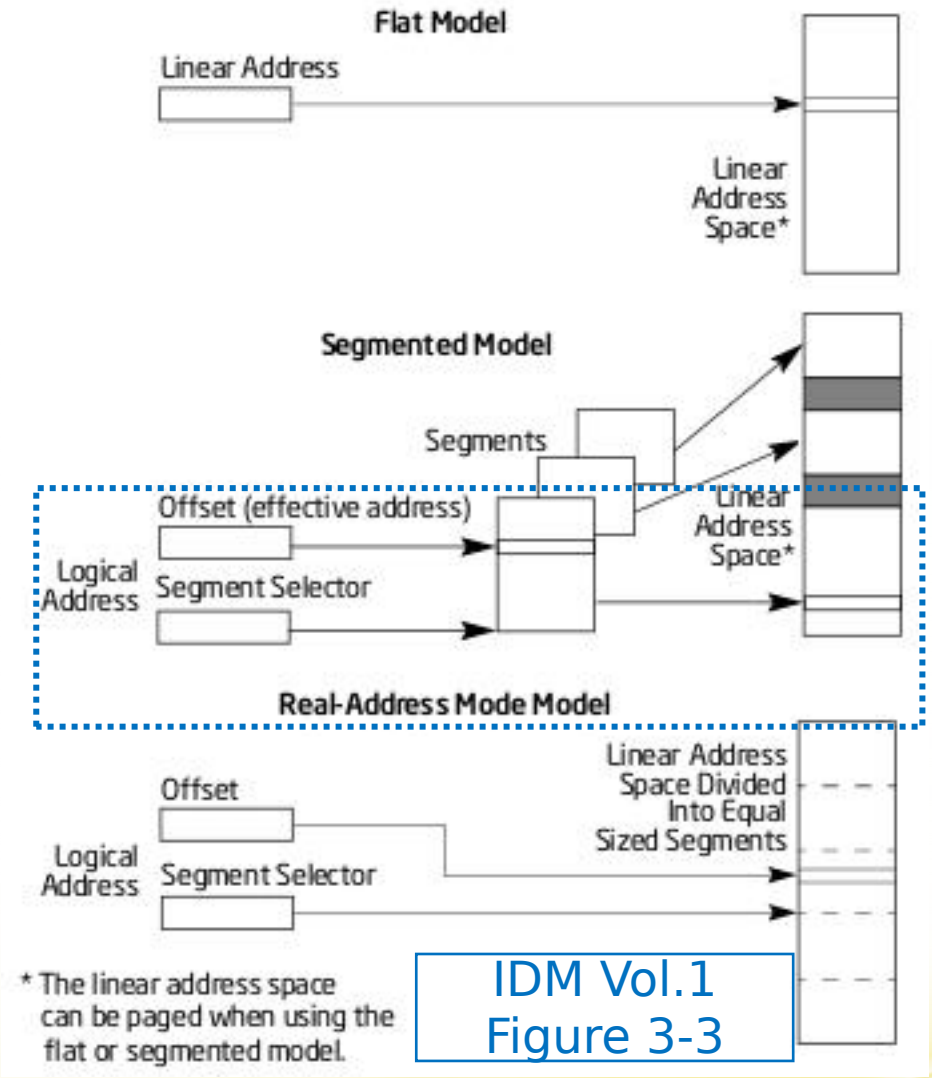
Paging

Memoria: Modelos de acceso

- Flat Model
- Segmented Model:
  - Vol. 3A 2-21
  - Vol. 3 Figure 2-1
- Real Address
  - Empleado por el kernel via Virtual 8086.
  - Puede emular "Flat Model".

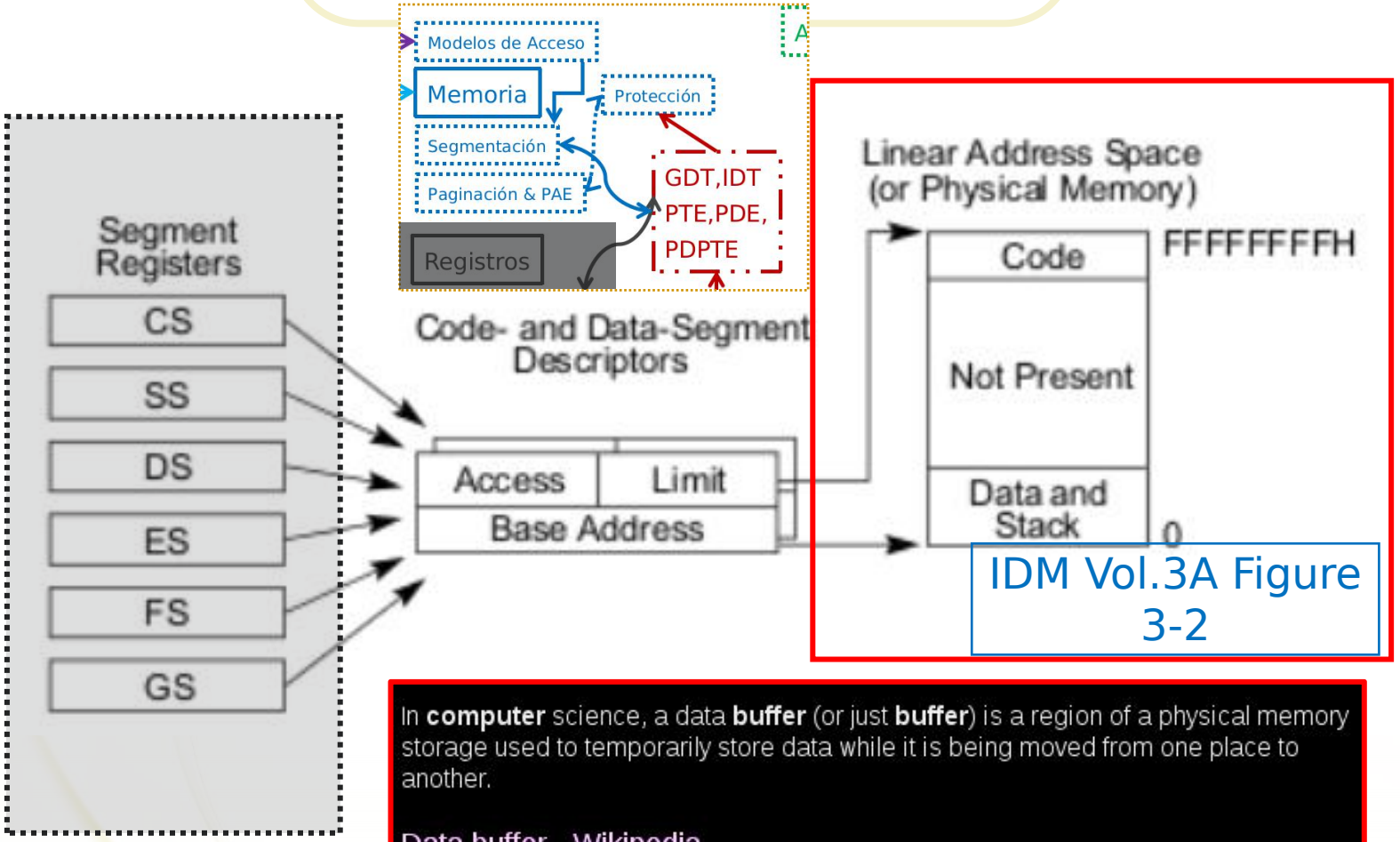
IDM Vol.3A 3.2.1

Modelos de Acceso



IDM Vol.1 Figure 3-3

Memoria: punto de vista



In **computer** science, a data **buffer** (or just **buffer**) is a region of a physical memory storage used to temporarily store data while it is being moved from one place to another.

Data buffer - Wikipedia
https://en.wikipedia.org/wiki/Data_**buffer**

Memoria: Bit & Byte ordering Little Endian

Bit & Byte Ordering

IDM Vol.1 1.3.1

- "Hello, World\n" is stored in memory as:

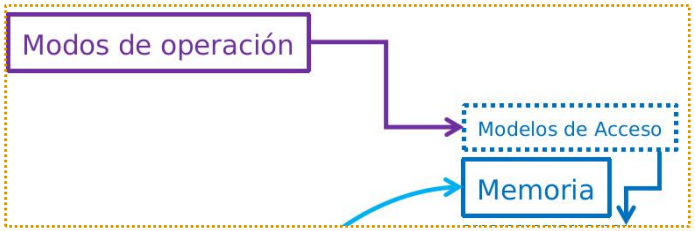

```
(gdb) shell  python -c 'import re; print re.sub("([0-9a-f]{2})",r" 0x\1 ","Hello, world\n".encode("hex"))'
 0x48  0x65  0x6c  0x6c  0x6f  0x2c  0x20  0x77  0x6f  0x72  0x6c  0x64  0x0a
```

```
(gdb) x/s $rdi
0x725de8:       "./example1_64"
(gdb) x/14bx $rdi
0x725de8:       0x2e    0x2f    0x65    0x78    0x61    0x6d    0x70    0x6c
0x725df0:       0x65    0x31    0x5f    0x36    0x34    0x00
(gdb) shell python -c "import re; print re.sub(r'([0-9a-f]{2})',r' 0x\1 ','./example1_64'.encode('hex'))"
 0x2e  0x2f  0x65  0x78  0x61  0x6d  0x70  0x6c  0x65  0x31  0x5f  0x36  0x34
(gdb)
```

Modos de Operación - Memoria (RET) - (RET)

Processor Mode	Memory Management Mode
Real-Address	Real-Address
System Management	~ Real-Address
Protected & Virtual 8086	Any
Compatibility Mode	Any
64-bit Mode	Segmentation usually Disabled CS, DS, ES & SS --> 0 Segmented & Real Address not available

IDM Vol.1 3.1 & 3.3.4



Niveles de Privilegios

"Real-Address"

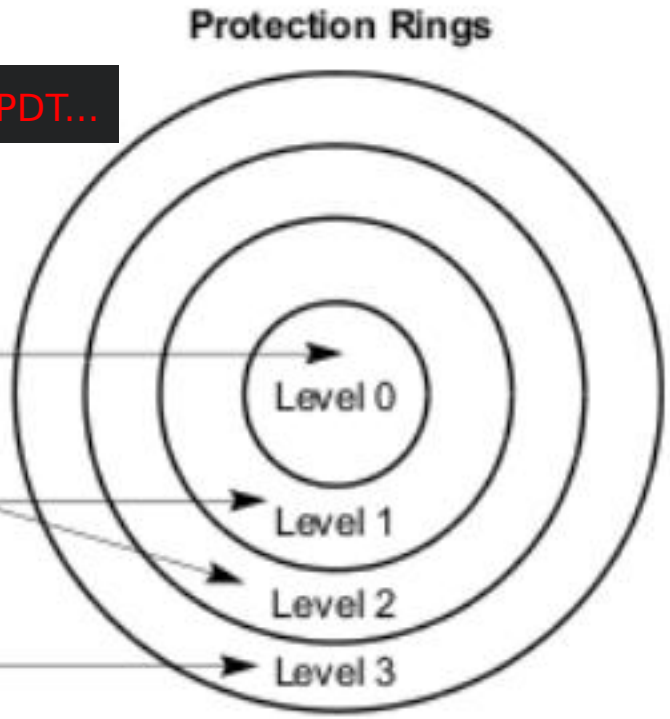
Puede cambiar GDT, IDT & PDT...

Virtual 8086

Linux kernel & Modules

Unused on Linux

Linux Apps



IDM Vol.3A Figure 5-3

Niveles de Privilegio

Anillos

Instrucciones & Operandos

- Conceptos:
  - Mnemonic
    - Un mnemónico es un nombre reservado para una clase de códigos de instrucción que tienen la misma función
  - Operandos
    - fuente -> derecha
    - destino -> izquierda
  - Sintaxsis Intel
    - *<inst> <dest>,<fuen*

For example:

LOADREG: MOV EAX, SUBTOTAL

```
(gdb) disas _start
Dump of assembler code for function _start:
   0x08048074 <+0>:     mov    $0x4,%eax
   0x08048079 <+5>:     mov    $0x1,%ebx
   0x0804807e <+10>:    mov    $0x8049096,%ecx
   0x08048083 <+15>:    mov    $0xd,%edx
   0x08048088 <+20>:    int    $0x80
   0x0804808a <+22>:    mov    $0x0,%ebx
   0x0804808f <+27>:    mov    $0x1,%eax
   0x08048094 <+32>:    int    $0x80
```

```
mov     $4, %eax
mov     $1, %ebx
mov     $hello, %ecx
mov     $s_len, %edx
int     $0x80
```

- GNU Assembler
  - Operandos
    - fuente -> izquierda
    - destino -> derecha
  - *La sintaxis puede cambiar entre ensambladores (nasm, yasm, gas)*
  - *Sintaxis AT&T*
    - *<inst> <fuente>,<destino>*

CISC-ALU

Arq. Set Instrucciones

<In>

<In> &,$r

<In> $r,$r

Arquitectura del Set de instrucciones I

General Purpose

X87 FPU

SSE (1,2,3,4,4.1,4.2)

AESNI

16-bit FP

System

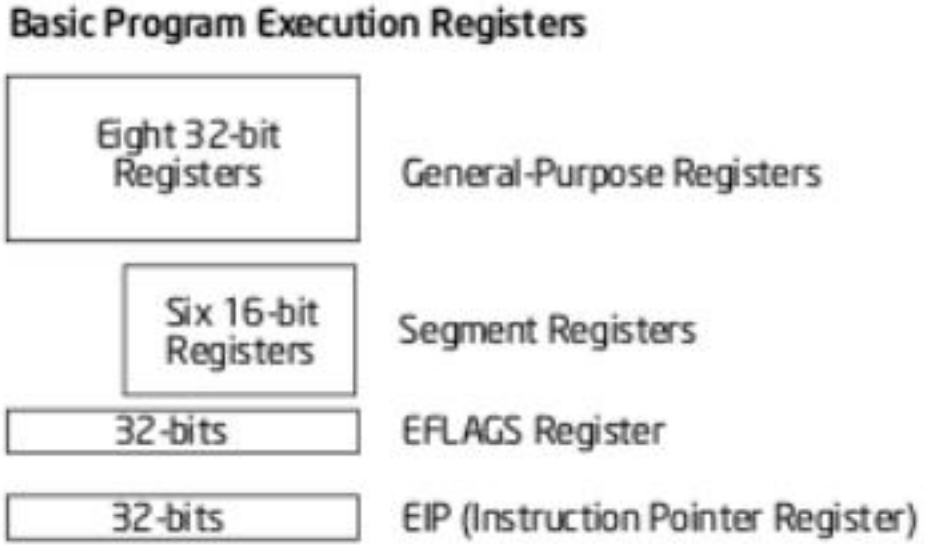
AVX

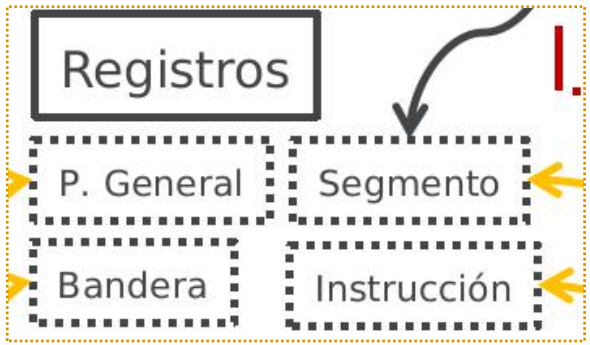
...

IDM Vol.1
Chapter 5

Registros (x86)

- Propósito general
  - EAX, EBX, ECX, EDX
  - EBP, ESP, ESI, EDI
- Segmento
  - CS: Código
  - SS: Pila
  - Datos:
    - DS, ES, FS, GS
- FLAGS (old) EFLAGS
- Instruccion (EIP)





IDM Vol.1 Figure 3-1

Registros GP (x86)

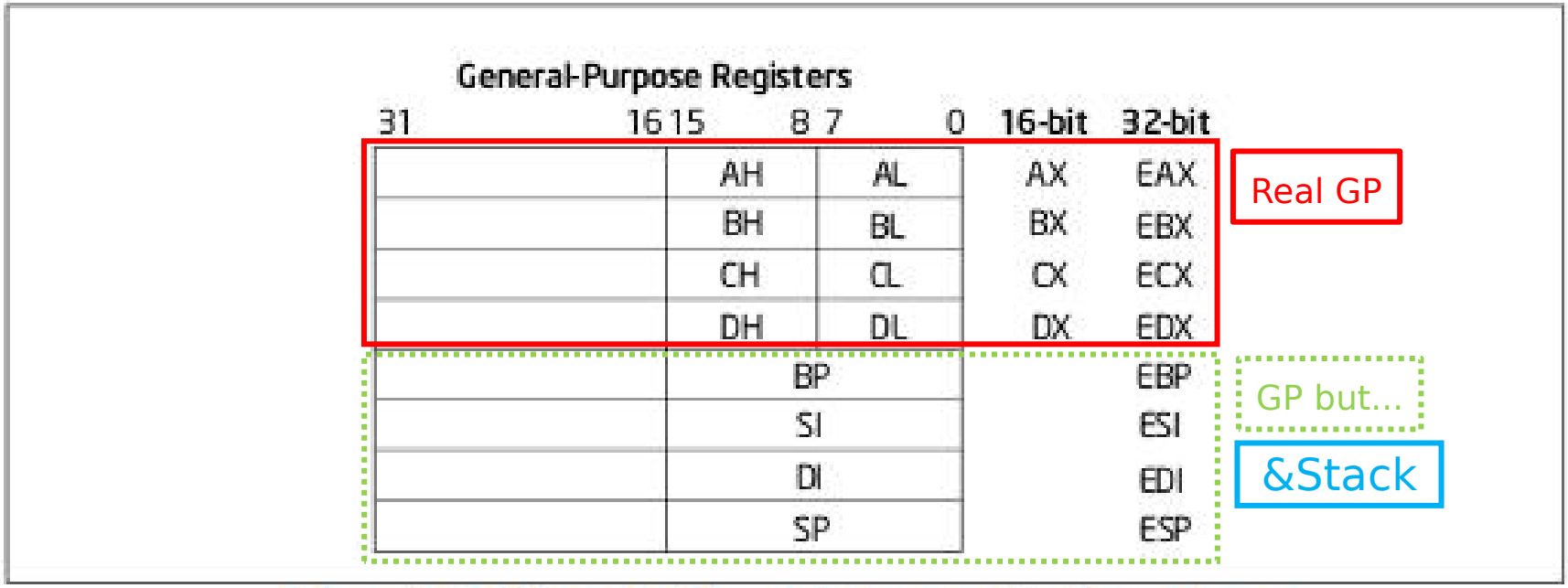
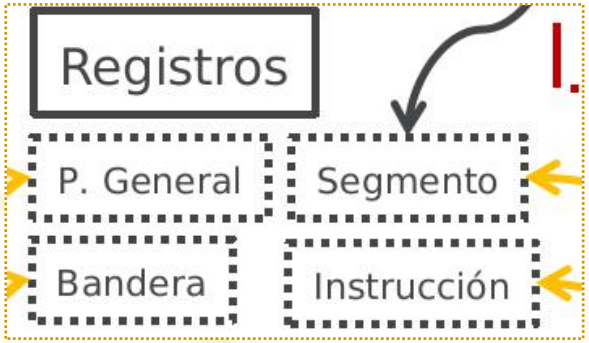
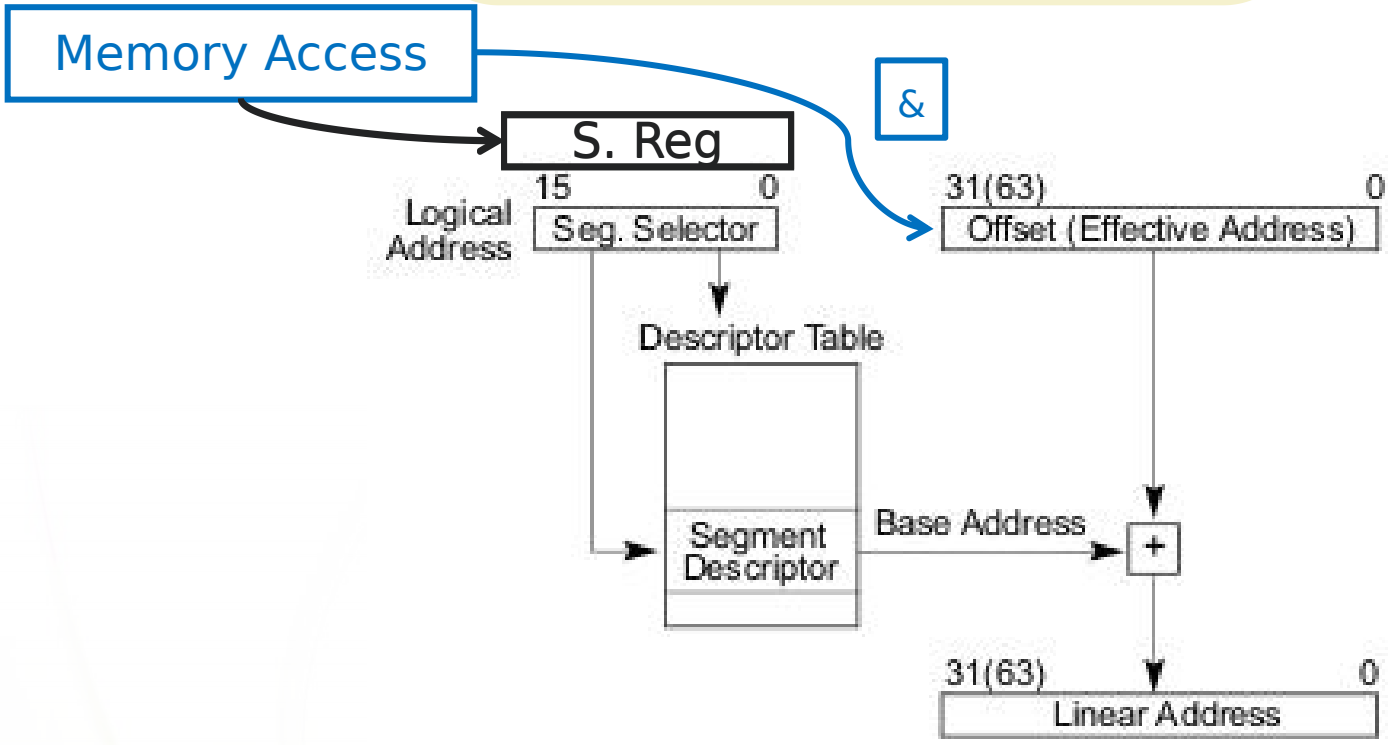


Figure 3-5. Alternate General-Purpose Register Names



IDM Vol.1 Figure 3-1

Selectores de segmento - Registros de Segmento (PTR)



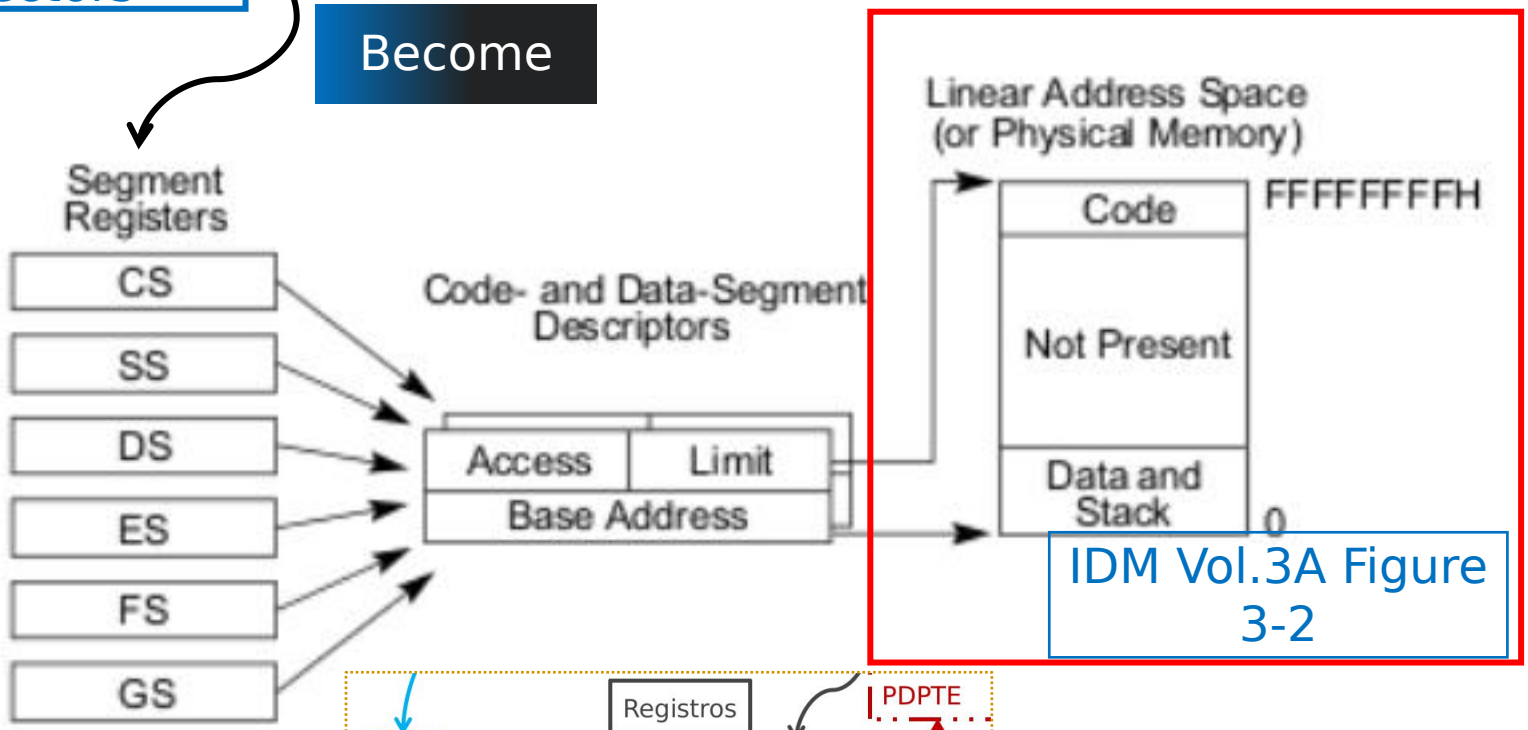
Visible Part	Hidden Part	
Segment Selector	Base Address, Limit, Access Information	CS
		SS
	~GDT CACHED	DS
		ES
		FS
		GS

IDM Vol.3A 3.4.2 & 3.4.3

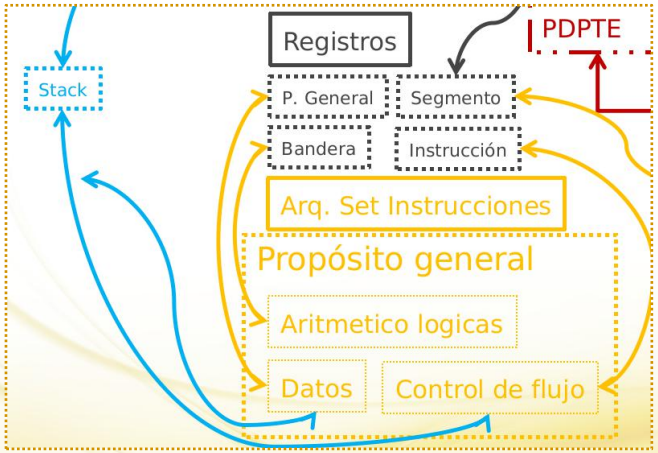
Memoria - Instrucciones - Registros (RET)

Segment Selectors

Become



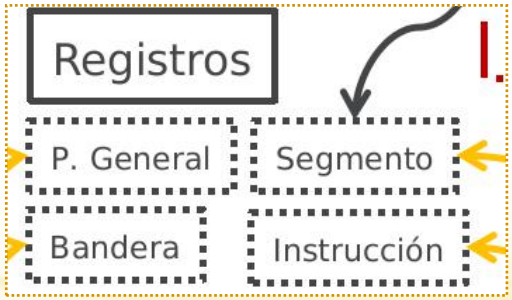
IDM Vol.3A Figure 3-2



Registro FLAGS

- Aporta información de estado sobre el resultado de las operaciones aritmeticológicas.

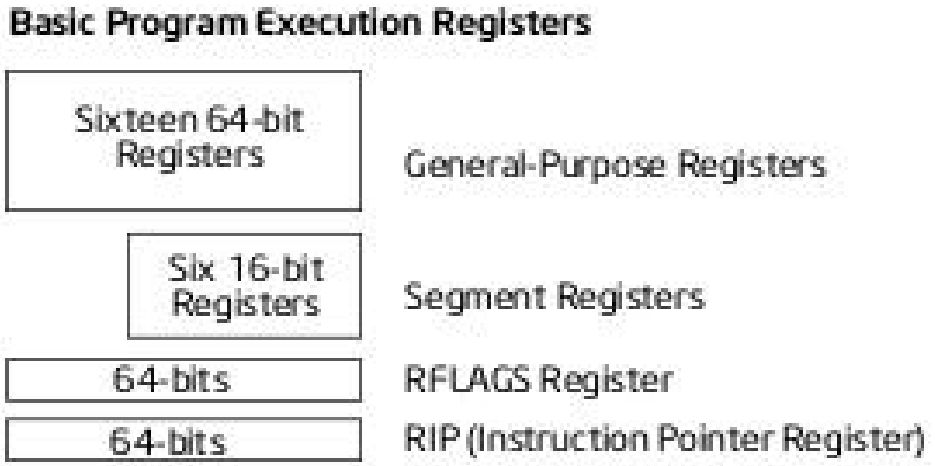
CF (bit 0)	**Carry flag** — Set if an arithmetic operation generates a carry or a borrow out of the most-significant bit of the result; cleared otherwise. This flag indicates an overflow condition for unsigned-integer arithmetic. It is also used in multiple-precision arithmetic.
PF (bit 2)	**Parity flag** — Set if the least-significant byte of the result contains an even number of 1 bits; cleared otherwise.
AF (bit 4)	**Auxiliary Carry flag** — Set if an arithmetic operation generates a carry or a borrow out of bit 3 of the result; cleared otherwise. This flag is used in binary-coded decimal (BCD) arithmetic.
ZF (bit 6)	**Zero flag** — Set if the result is zero; cleared otherwise.
SF (bit 7)	**Sign flag** — Set equal to the most-significant bit of the result, which is the sign bit of a signed integer. (0 indicates a positive value and 1 indicates a negative value.)
OF (bit 11)	**Overflow flag** — Set if the integer result is too large a positive number or too small a negative number (excluding the sign-bit) to fit in the destination operand; cleared otherwise. This flag indicates an overflow condition for signed-integer (two's complement) arithmetic.



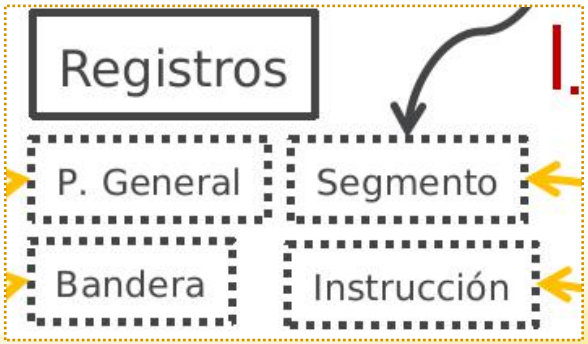
IDM Vol.1 3-16

Registros (x86_64)

- Propósito general
  - RAX, RBX, RCX, RDX
    - 64,32,16,8 bits
  - RSP, RBP, RDI, RSI
    - 64, 32
- Segmento
  - Generalmente deshabilitado…
  - Vol 3.A 3.4.3 (Virtual Exec)
- RFLAGS
- Instruccion (RIP)



IDM Vol.1 Figure 3-1



Propósito General

Datos <->

PUSH
POP
MOV

&Stack

Presentan la base de los BoFs

Control de Flujo

CALL
RET
IRET
INT
INTO
BOUND
ENTER
LEAVE

Misc

LEA
NOP
UD

Aritmetico-Lógicas

AND
OR
XOR
NOT
ADD
ADC
SUB

0x00 significa final de cadena
mov $1,%EAX mov $1,%EBX
sub $EBX, %EAX

Segmento

LDS
LES
LFS
LGS
LSS

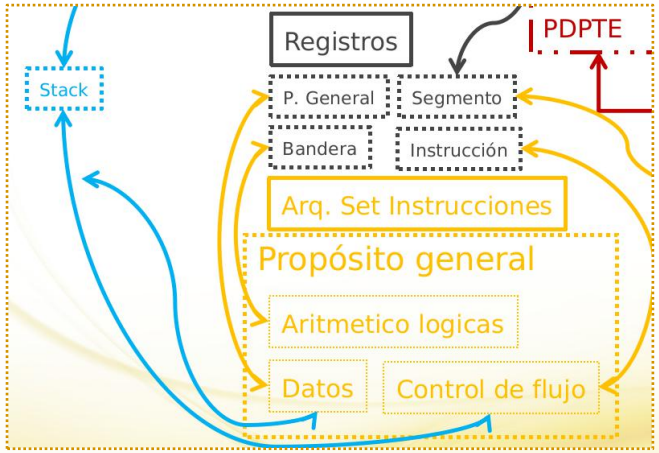
Instrucciones - Registros (RET)

- Condicionales
  - Jcc
- Instrucciónes
  - Explícita
    - MOV
  - Implícita
    - MOV
    - PUSH, PUSHA
    - POP, POPA
    - LDS/LES/LFS/LGS/LSS
    - CALL
    - JMP
    - RET
    - SYSENTER
    - SYSCALL
    - INT

IDM Vol.2B MOV

A3	MOV *moffs16**,AX
A3	MOV *moffs32**,EAX

Move AX to (*seg*:*offset*).
Move EAX to (*seg*:*offset*).



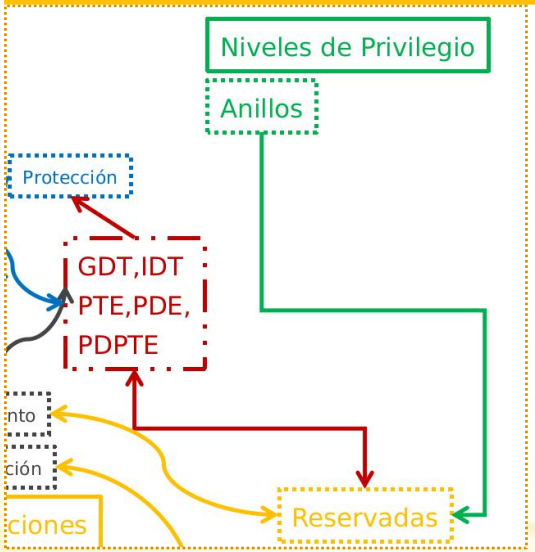
Sistema - Memoria - Niveles de Privilegio (RET)

Memory subsystem tables managment instructions

LGDT	Load global descriptor table (GDT) register.
SGDT	Store global descriptor table (GDT) register.
MOV	Load and store control registers.
CLTS	Clear the task-switched flag.
ARPL	Adjust requested privilege level.
LAR	Load access rights.
LSL	Load segment limit.
SYSENTER	Fast System Call, transfers to a flat protected mode kernel at CPL = 0.
SYSEXIT	Fast System Call, transfers to a flat protected mode kernel at CPL = 3.
SYSCALL	Fast call to privilege level 0 system procedures.
SYSRET	Return from fast systemcall.

x86_64

...

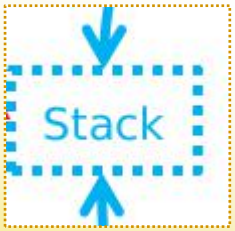


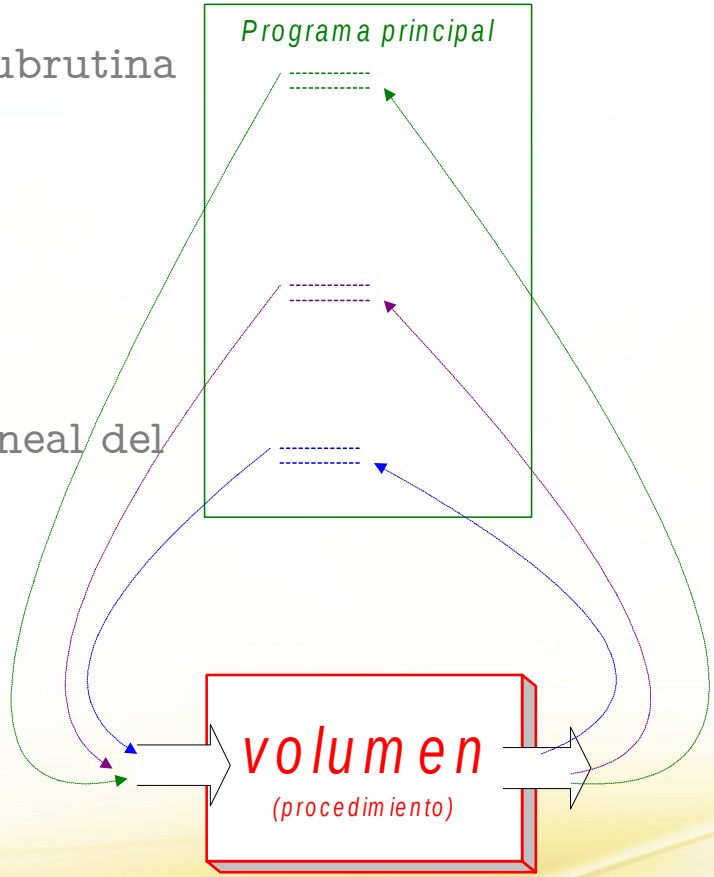
- **Descriptor privilege level (DPL) field** — (Bits 13 and 14 in the second doubleword of a segment descriptor.) Determines the privilege level of the segment.
- **Requested privilege level (RPL) field** — (Bits 0 and 1 of any segment selector.) Specifies the requested privilege level of a segment selector.
- **Current privilege level (CPL) field** — (Bits 0 and 1 of the CS segment register.) Indicates the privilege level of the currently executing program or procedure. The term current privilege level (CPL) refers to the setting of this field.
- **User/ supervisor (U/ S) flag** — (Bit 2 of paging-structure entries.) Determines the type of page: user or supervisor.
- **Read/ write (R/ W) flag** — (Bit 1 of paging-structure entries.) Determines the type of access allowed to a page: read-only or read/write.
- **Execute-disable (XD) flag** — (Bit 63 of certain paging-structure entries.) Determines the type of access allowed to a page: executable or not-executable.

IDM Vol.3A 5.2

Stack (1/3)

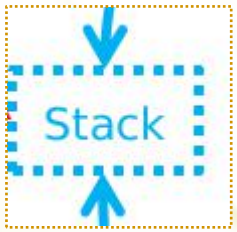
- Estructura de datos implementada por el procesador
  - Sigue una disciplina LIFO (Last-in – First-out)
  - Esto permite llamadas recursivas a subrutina
- Sobre la memoria principal
  - SS apunta al segmento de Stack
  - ESP es el offset del puntero de pila.
  - EBP es el offset base.
  - SS:ESP es la dirección de memoria lineal del puntero de pila
- Esencial para subrutinas (Procedimientos)

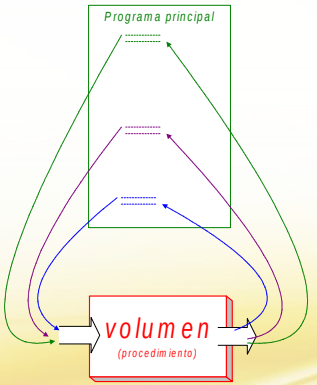




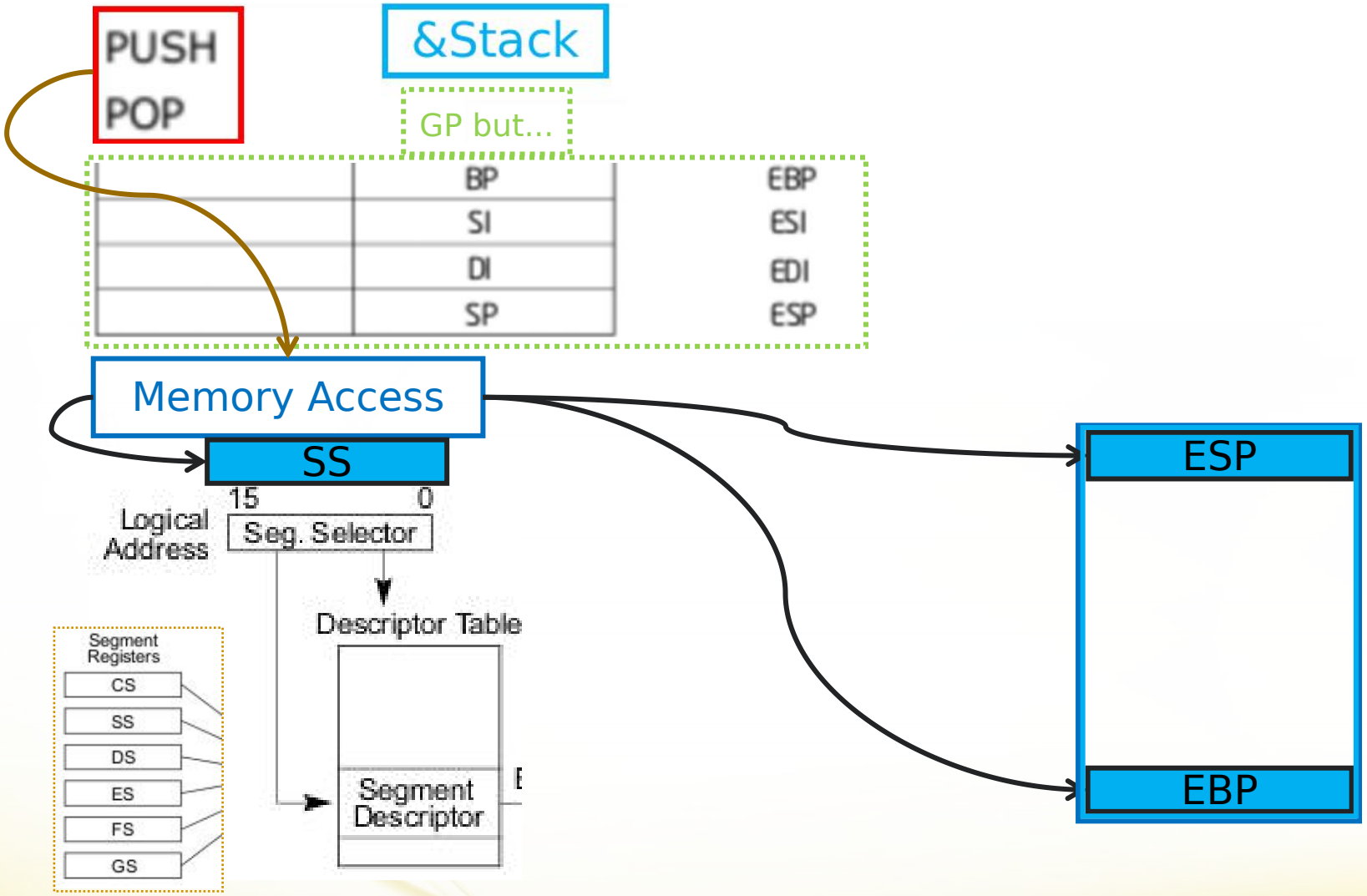
Stack (2/3)

- Estructura de datos implementada por el procesador
- Básica para implementar subrutinas (procedimientos)
  - Instrucciones call/ret y el retorno de interrupción iret
  - Paso de parámetros en stack frames
  - Las variables locales del procedimiento llamado se almacenan en la stack
  - EBP debe apuntar a la base de la pila al entrar (enter/leave) en un procedimiento





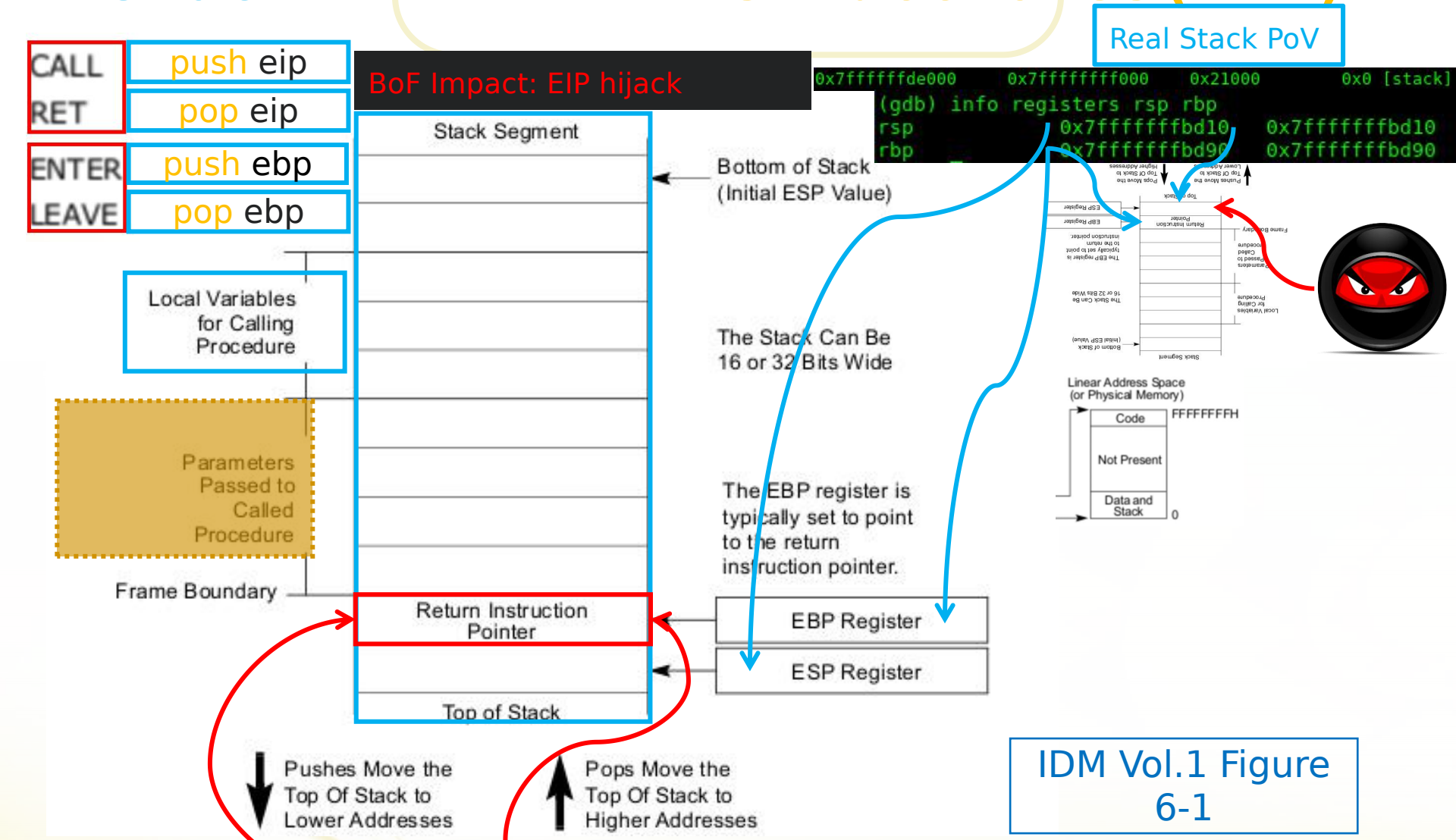
Stack - Registros "Frontera" - Instrucciones (RET)



Stack - EIP - Instrucciones(RET)

CALL	push eip
RET	pop eip
ENTER	push ebp
LEAVE	pop ebp

BoF Impact: EIP hijack



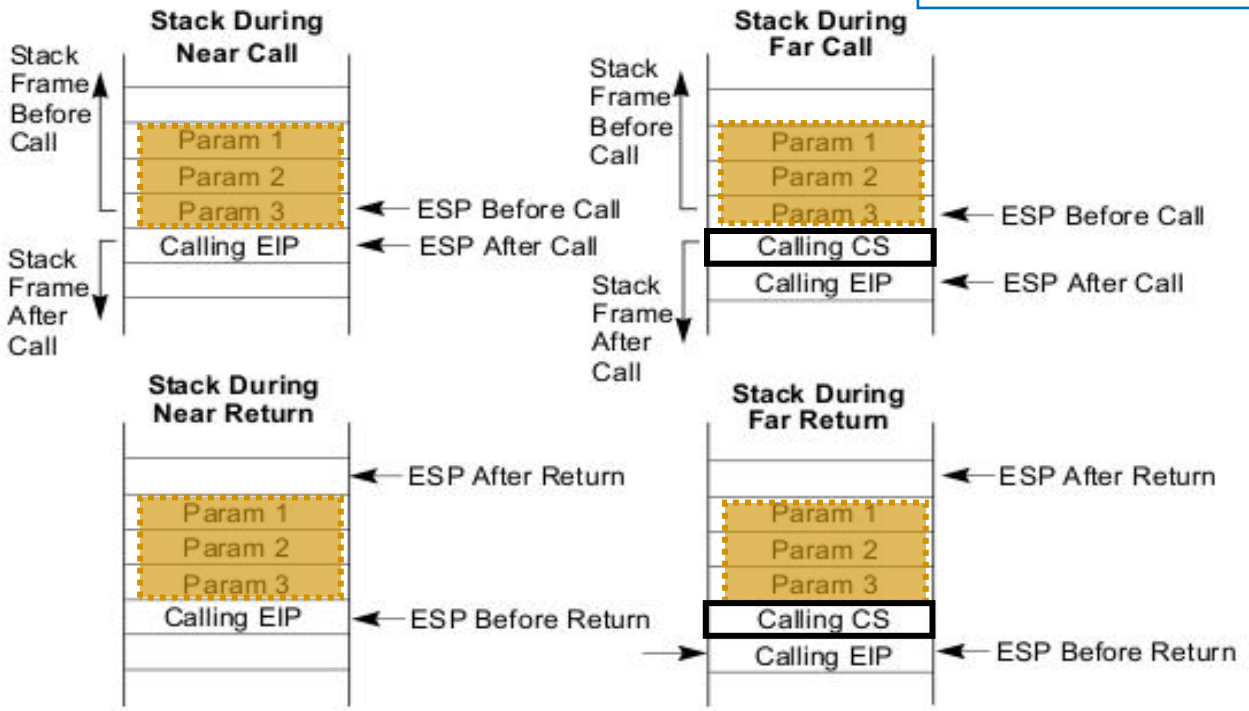
Real Stack PoV

```
0x7fffffde000    0x7fffffff000    0x21000          0x0 [stack]
(gdb) info registers rsp rbp
rsp            0x7fffffffbd10   0x7fffffffbd10
rbp            0x7fffffffbd90   0x7fffffffbd90
```

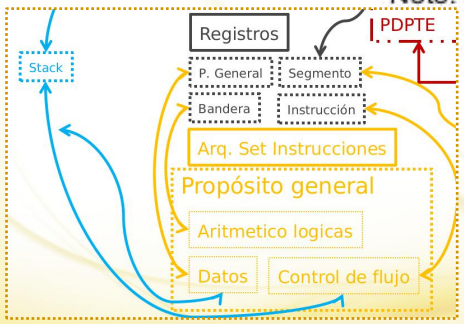

IDM Vol.1 Figure 6-1

Instrucciones-Memoria-Stack-Registros(RET)

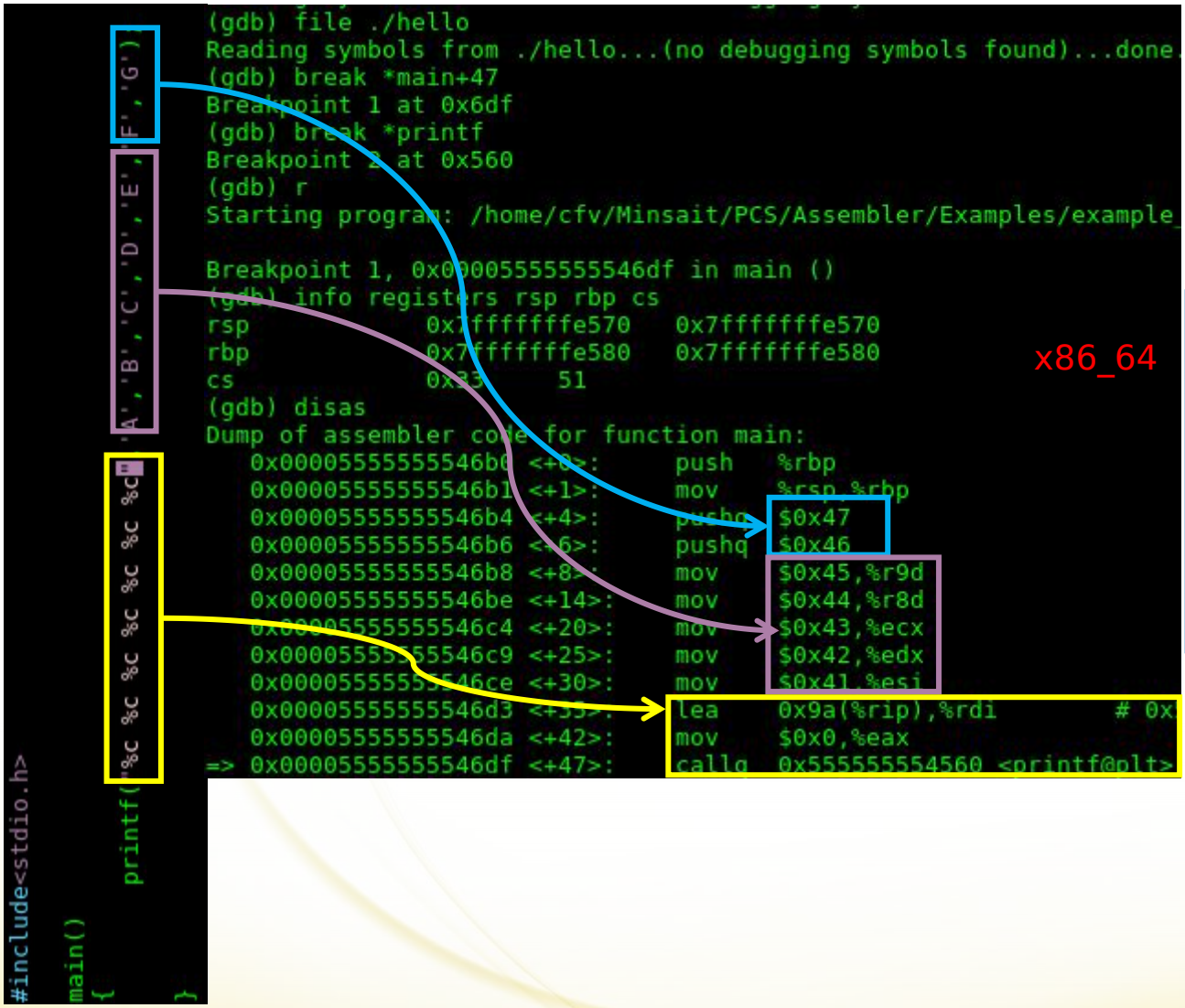
IDM Vol.1 Figure 6-2

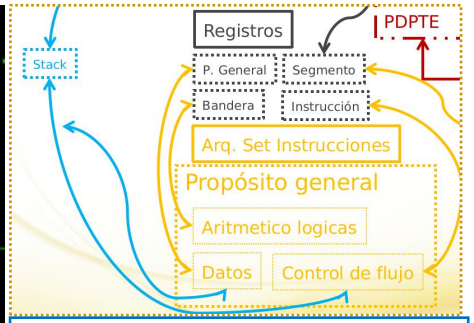


Note: On a near or far return, parameters are released from the stack based on the optional *n* operand in the RET *n* instruction.



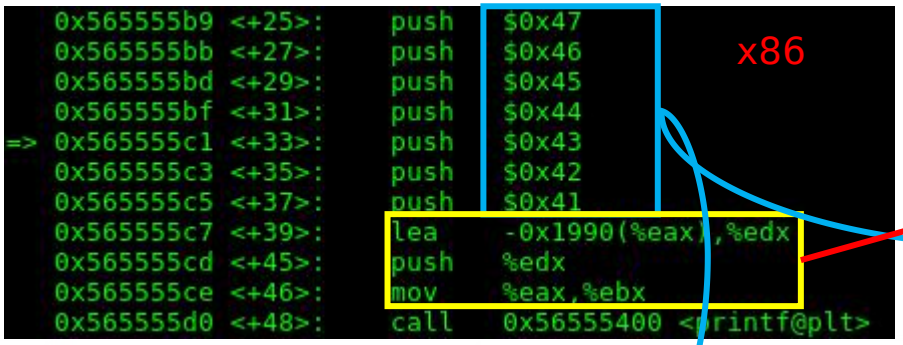
Instrucciones-Memoria-Stack-Registros-> Calling Conventions



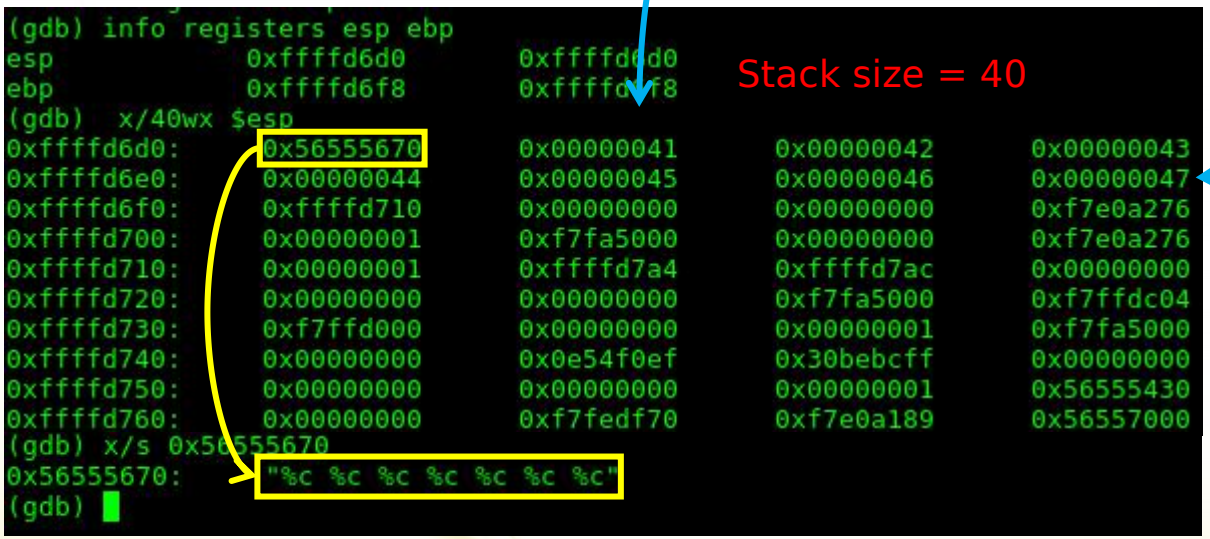


a way to make data available for other routine

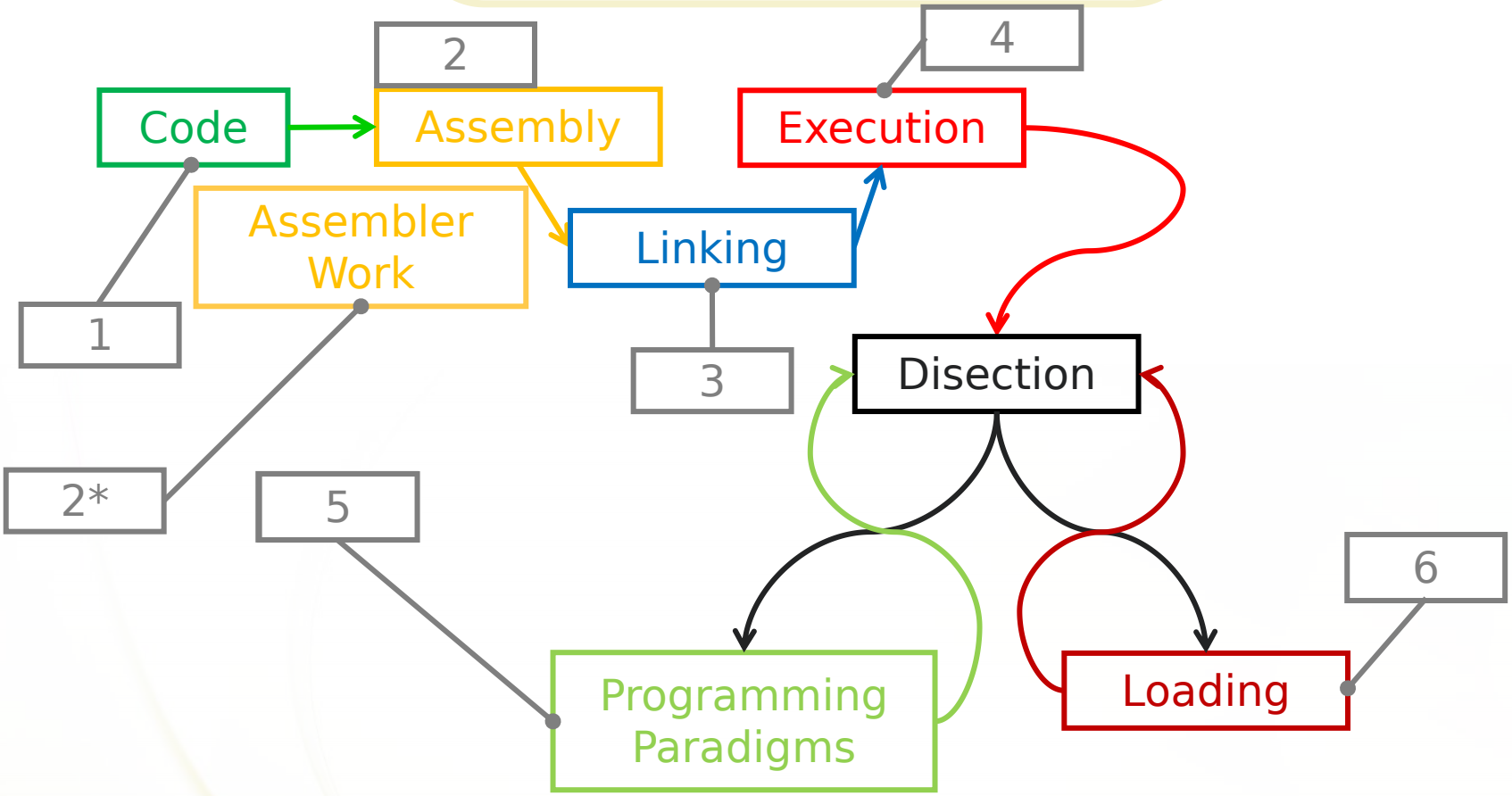
Instrucciones-Memoria-Stack-Registros-Calling Conventions



Base de las vulnerabilidades Format String. Cuando el número de parámetros a tomar de la pila viene de una fuente no confiable.



Just do It: Mapa Conceptual



Code, Assembly, Link & Run: Mapa Conceptual

Human Readable

Assembly code

```
.global      _start
.text
_start:
    nop
    nop
    nop
    nop
    nop
    call _bye
_bye:
    mov     $ret_c, %ebx
    mov     $1, %eax
    int     $0x80
.section .data
ret_c = 0x0
```

AS - the portable GNU assembler.

Machine Readable

Not linked

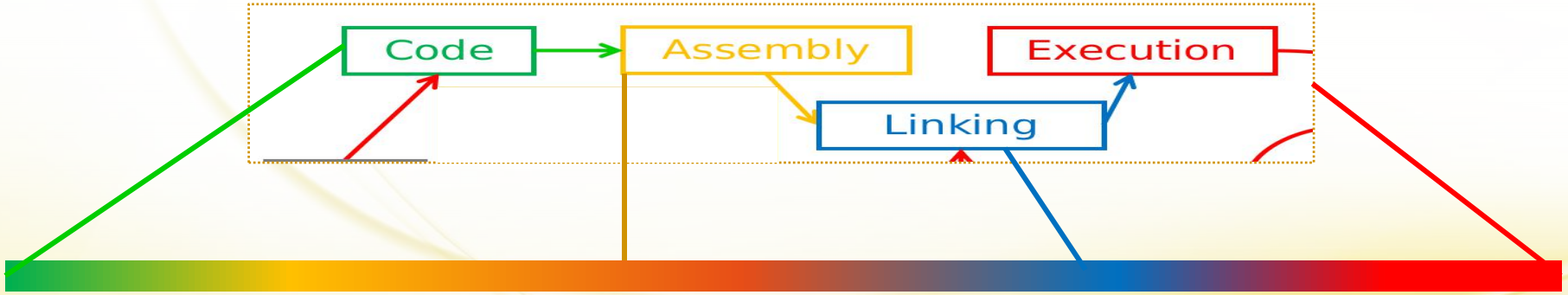
```
example1_32.o: ELF 32-bit LSB relocatable, Intel 80386,
version 1 (SYSV), not stripped
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Entry point address:                0x0
  Type:                               REL (Relocatable file)
```

ld - The GNU Linker

Machine Readable

Linked Executable

```
example1_32: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
statically linked, not stripped
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Entry point address:                0x8048074
  Type:                               EXEC (Executable file)
```



as && ld && ./<executable>

```
$ ../../tools/assembly_link_gnu.sh example1_32.s 32
Assemble: as -32 -o example1_32.o example1_32.s
Assembled
Link : ld -o example1_32 -static -m elf_i386 example1_32.o
Linked example1_32
example1_32: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, not stripped
example1_32.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
Exec: ---
FinExec: ---
```

X86

```
$ ../../tools/assembly_link_gnu.sh example1_32.s 64
Assemble: as -64 -o example1_32.o example1_32.s
Assembled
Link : ld -o example1_32 -static -m elf_x86_64 example1_32.o
Linked example1_32
example1_32: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, not stripped
example1_32.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped
Exec: ---
FinExec: ---
```

x86_64

Programming Paradigm: OS API

Wikipedia

Linux kernel-to-userspace

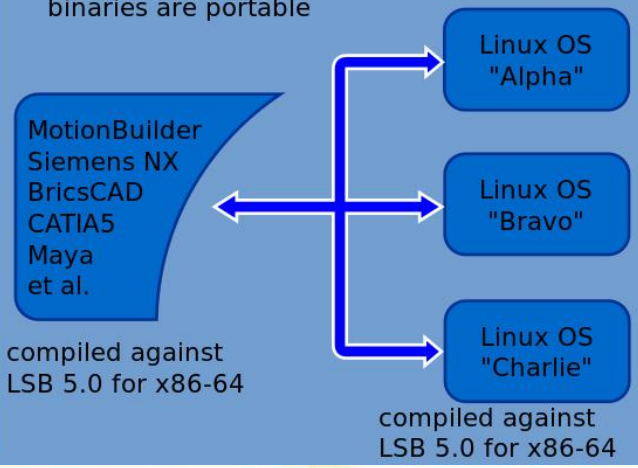
API

☑ API stability **is** guaranteed, source code is portable!

Syscall set and identifiers.
Unistd.h: defines the access to POSIX api (IDs)

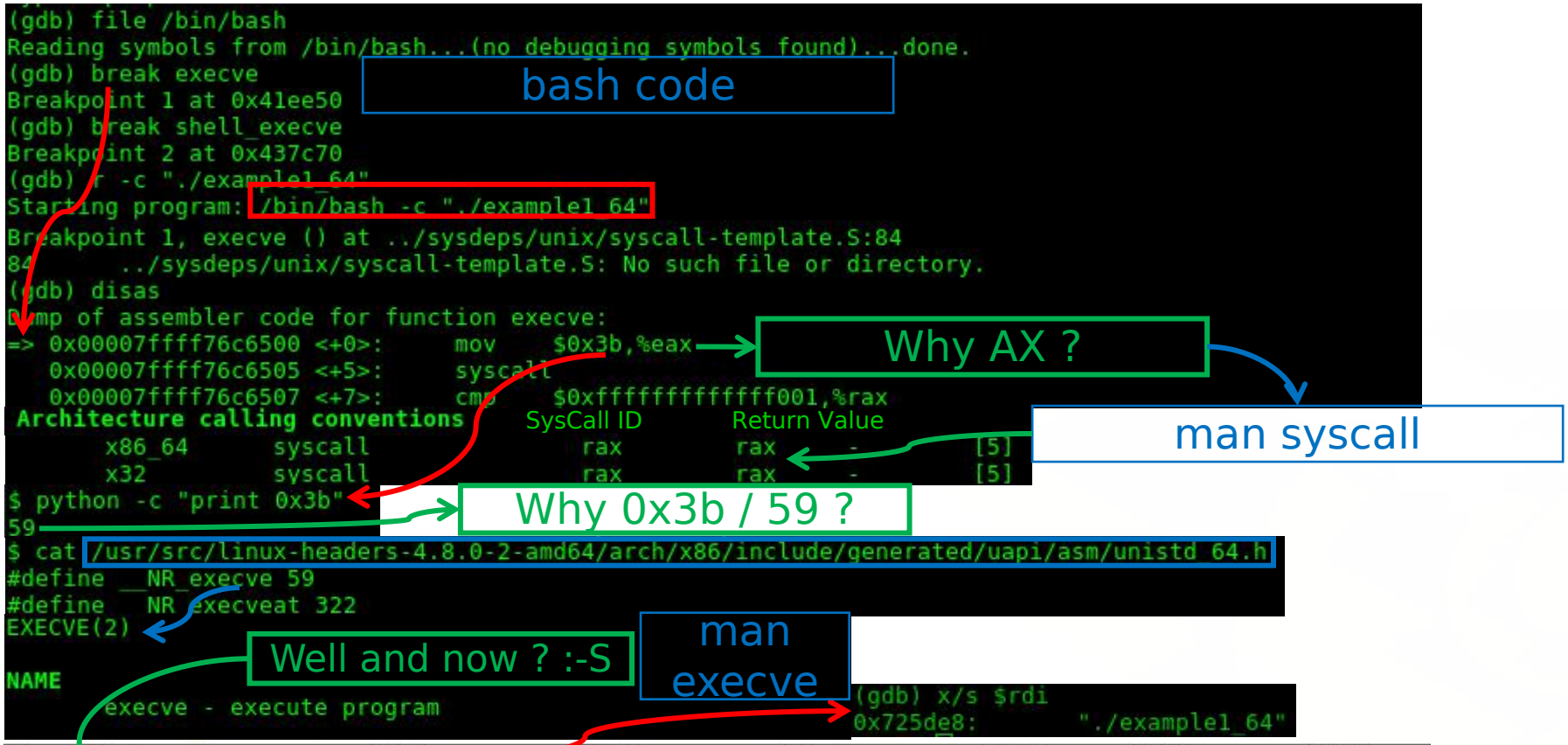
ABI

☑ compatible ABI **can be** guaranteed, binaries are portable



- Sizes
- Alignment
- Calling Conventions
- How to syscall

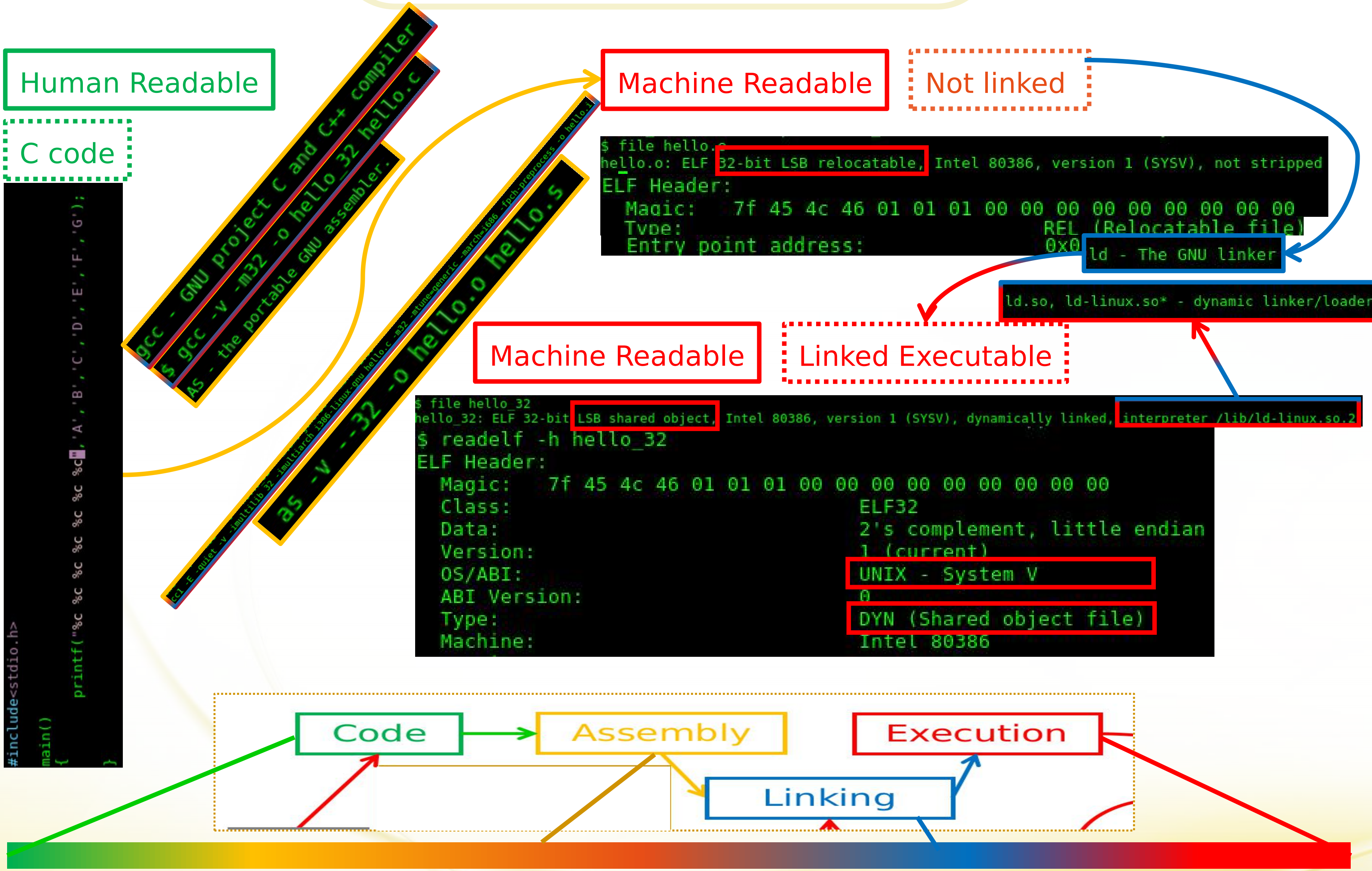
Ejecución - Programming Paradigms: OS API



%rax	System call	%rdi	%rsi	%rdx	%r10	%r8	%r9
59	sys_execve	const char *filename	const char *const argv[]	const char *const envp[]			

http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

GCC: Mapa Conceptual



Linking - Loading - Available functions

¿ De qué símbolos disponemos en ejecución ?

```
LDD(1)

NAME
       ldd - print shared object dependencies
```

```
linux-gate.so.1 (0xf77a0000)
libc.so.6 => /lib32/libc.so.6 (
/lib/ld-linux.so.2 (0x56589000)
```

```
(gdb) file hello_32
Reading symbols from hello_32..
(gdb) break main
Breakpoint 1 at 0x5af
(gdb) r
Starting program: /home/cfv/Mir

Breakpoint 1, 0x565555af in mai
(gdb) info functions socket
All functions matching regular

Non-debugging symbols:
0xf7edaee0  __socket
0xf7edaee0  socket
0xf7edaf30  socketpair
(gdb)
```

Debbuging ld.so

```
$ LD_DEBUG=help ./hello_32
Valid options for the LD_DEBUG environment variable are:

  libs        display library search paths
  reloc       display relocation processing
  files       display progress for input file
  symbols     display symbol table processing
  bindings    display information about symbol binding
  versions    display version dependencies
  scopes      display scope information
  all         all previous options combined
  statistics  display relocation statistics
  unused      determined unused DSOs
```

```
$ LD_DEBUG=all LD_DEBUG_OUTPUT=out.tmp ./hello_32
A B C D E F G$
```

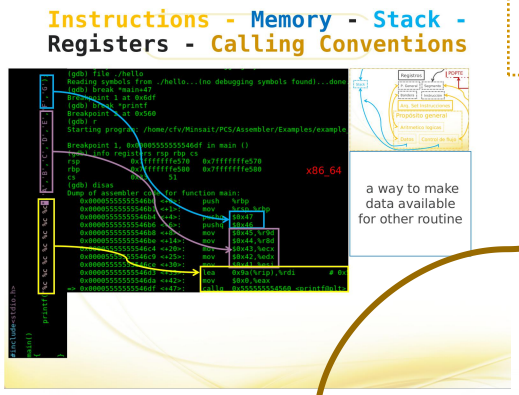
```
    19609:     file=libc.so.6 [0];  needed by ./hello_32 [0]
    19609:     find library=libc.so.6 [0]; searching
    19609:      search cache=/etc/ld.so.cache
    19609:       trying file=/lib32/libc.so.6                    ...
    19609:
    19609:     file=libc.so.6 [0];  generating link map
    19609:       dynamic: 0xf777fdb0  base: 0xf75cd000   size: 0x001b6a1c
    19609:         entry: 0xf75e53f0  phdr: 0xf75cd034  phnum:         10
```

```
(gdb) disas shell_execve
No symbol table is loaded.  Use the "file" command.
(gdb)
```

```
$ cat out.tmp.19609 |grep symbol= | awk -F" " '{print $2}'| sort |uniq | grep printf
symbol=printf;
$
```

```
$ cat out.tmp.19609 |grep symbol= | awk -F" " '{print $2}'| sort |uniq | grep execve ; echo $?
1
$
```

Programming Paradigms - Syscalls vs System Function Calls

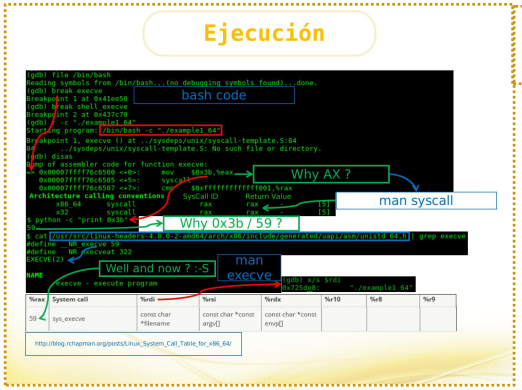


System Functions

OS stdlib

The attacker has to know where is the function he wants in memory.

POSIX (Portable OS. System interface)



Direct Syscalls

OS service request identifiers

More portable between different programs & OS With the same API/ABI (POSIX)

Instruction Encoding: MOV example (Assembler work)

IDM Vol.2A Table 2-2

Effective Address	Mod	R/M	
[EAX]	00	000	00
[ECX]		001	01
[EDX]		010	02
[EBX]		011	03
[--][--][1]		100	04
disp32[2]		101	05
[ESI]		110	06
[EDI]		111	07

- Immediate Values
  - imm8
  - imm[16,32,64] words

IDM Vol.2A 3.1.1.3

Assembler Work

- Instructions from code:

```
mov    $4, %eax
mov    $1, %ebx
```

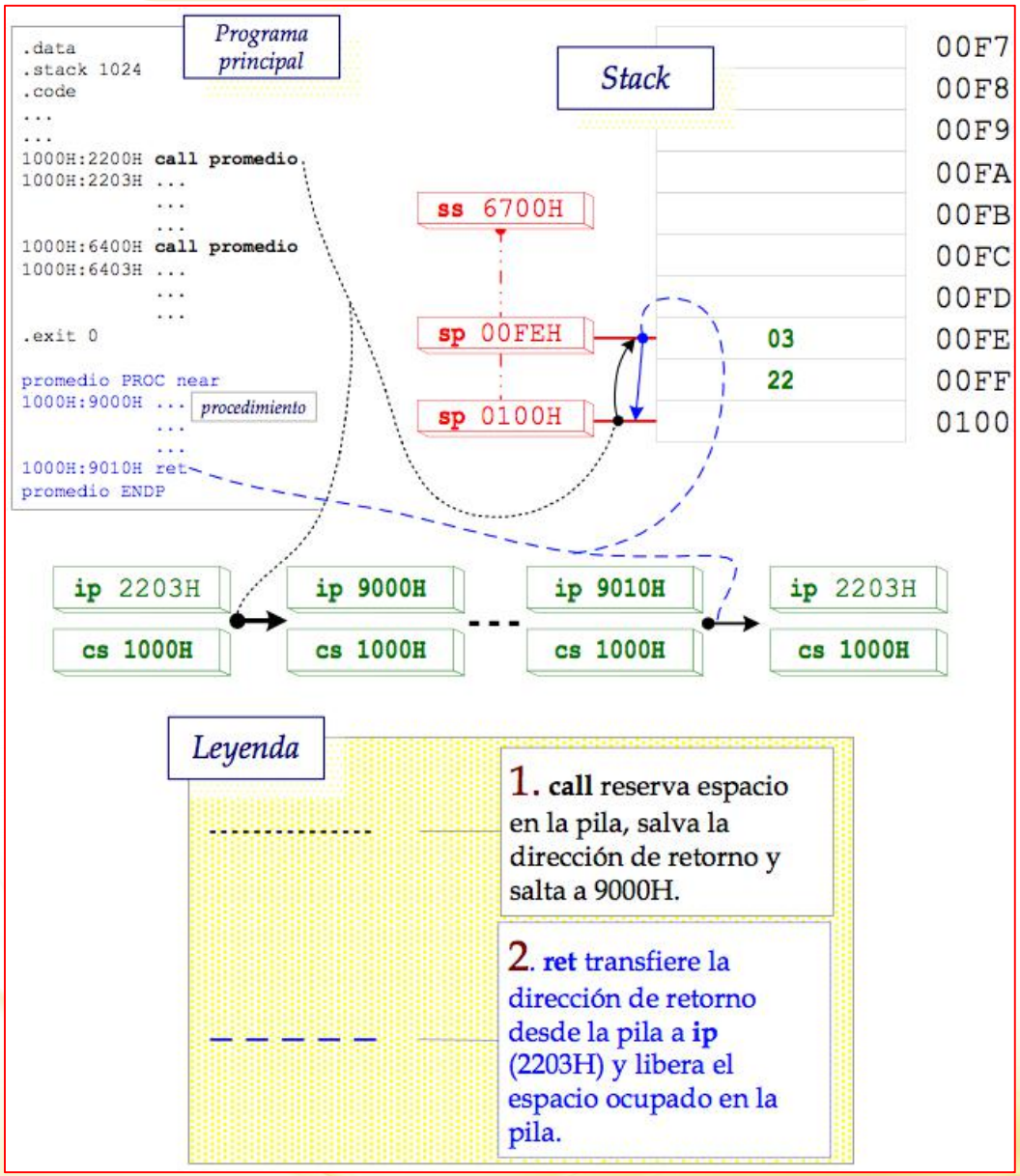
B8+ rd id	MOV r32, imm32

```
0xb8    0x04    0x00    0x00    0x00
0xbb    0x01    0x00    0x00    0x00
```

- Simple:
  - 0xb8 + 0x00
  - 0xb8 + 0x03

IDM Vol.2B MOV

Stack (3/3), ejemplo en modo real



FIN