Ingeniería Inversa para Principiantes

Dennis Yurichev

Ingeniería Inversa para Principiantes

Dennis Yurichev
<dennis(a)yurichev.com>

Versión del texto (13 de abril de 2016).

La última versión (así como las versiones en inglés y ruso) de este texto está disponible en beginners.re. Una versión para lector de libros electrónicos también está disponible.

Además puedes seguirme en twitter para obtener información sobre actualizaciones de este texto: @yurichev[1], o subscribirte a la lista de correo[2].

La portada fue hecha por Andy Nechaevsky: facebook.

[1] twitter.com/yurichev
[2] yurichev.com

Spanish text placeholder

¡Atención: ésta es una versión LITE resumida!

Es apróximadamente 6 veces más corta que la versión completa (~150 páginas) y está dirigida a aquellos que deseen una introducción breve a la esencia de la ingeniería inversa. No incluye nada sobre MIPS, ARM, OllyDBG, GCC, GDB, IDA, no contiene ejercicios, ejemplos, etc.

Si aún estás interesado en la ingeniería inversa, la versión completa del libro siempre está disponible en mi sitio: beginners.re.

Índice general

Prólogo

Existen muchos significados populares para el término «reverse engineering»: 1) La ingeniería inversa de software: la investigación de programas compilados; 2) El escaneo de estructuras 3D y la manipulación digital subsecuente requerida para duplicarlas; 3) La recreación de la estructura de un DBMS[3]. Este libro es acerca del primer significado.

Ejercicios y tareas

...fueron movidos al sitio web: `http://challenges.re`.

Sobre el autor

Dennis Yurichev es un reverser y programador experimentado. Puede ser contactado por email: **dennis(a)yurichev.com**, o en Skype: **dennis.yurichev**.

Elogios para *Ingeniería Inversa para Principiantes*

- «It's very well done .. and for free .. amazing.»[4] Daniel Bilar, Siege Technologies, LLC.
- «... excellent and free»[5] Pete Finnigan,gurú de seguridad en Oracle RDBMS.
- «... book is interesting, great job!» Michael Sikorski, autor de *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*.
- «... my compliments for the very nice tutorial!» Herbert Bos, catedrático de tiempo completo en la Vrije Universiteit Amsterdam, coautor de *Modern Operating Systems (4th Edition)*.
- «... It is amazing and unbelievable.» Luis Rocha, CISSP / ISSAP, Technical Manager, Network & Information Security at Verizon Business.
- «Thanks for the great work and your book.» Joris van de Vis, especialista en SAP Netweaver & Security .
- «... reasonable intro to some of the techniques.»[6] Mike Stay, profesor en el Federal Law Enforcement Training Center, Georgia, US.
- «I love this book! I have several students reading it at the moment, plan to use it in graduate course.»[7] Sergey Bratus, Research Assistant Professor en el Departamento de Ciencias de la Computación en Dartmouth College
- «Dennis @Yurichev has published an impressive (and free!) book on reverse engineering»[8] Tanel Poder, experto en afinación de rendimiento de Oracle RDBMS .
- «This book is some kind of Wikipedia to beginners...» Archer, Chinese Translator, IT Security Researcher.

Agradecimientos

Por contestar pacientemente a todas mis preguntas: Andrey «herm1t» Baranovich, Slava «Avid» Kazakov.

Por enviarme notas acerca de errores e inexactitudes: Stanislav «Beaver» Bobrytskyy, Alexander Lysenko, Shell Rocket, Zhu Ruijin, Changmin Heo.

[3] Database management systems
[4] twitter.com/daniel_bilar/status/436578617221742593
[5] twitter.com/petefinnigan/status/400551705797869568
[6] reddit
[7] twitter.com/sergeybratus/status/505590326560833536
[8] twitter.com/TanelPoder/status/524668104065159169

Por ayudarme de otras formas: Andrew Zubinski, Arnaud Patard (rtp en #debian-arm IRC), .

Por traducir el libro a Chino Simplificado: Antiy Labs (antiy.cn) y Archer.

Por traducir el libro a Coreano : Byungho Min.

Spanish text placeholder: Diego Boy, Luis Alberto Espinosa Calvo.

Por corrección de pruebas: Alexander «Lstar» Chernenkiy, Vladimir Botov, Andrei Brazhuk, Mark “Logxen” Cooper, Yuan Jochen Kang, Mal Malakov, Lewis Porter, Jarle Thorsen.

Vasil Kolev realizó una gran cantidad de trabajo en corrección de pruebas y corrección de muchos errores.

Por las ilustraciones y el arte de la portada: Andy Nechaevsky.

Gracias a toda la gente en github.com que ha contribuido con notas y correcciones.

Muchos paquetes de LaTeX fueron utiliados: quiero agradecer también a sus autores.

Donadores

Aquellos que me apoyaron durante el tiempo que escribí una parte significativa del libro:

2 * Oleg Vygovsky (50+100 UAH), Daniel Bilar ($50), James Truscott ($4.5), Luis Rocha ($63), Joris van de Vis ($127), Richard S Shultz ($20), Jang Minchang ($20), Shade Atlas (5 AUD), Yao Xiao ($10), Pawel Szczur (40 CHF), Justin Simms ($20), Shawn the R0ck ($27), Ki Chan Ahn ($50), Triop AB (100 SEK), Ange Albertini (€10+50), Sergey Lukianov (300 RUR), Ludvig Gislason (200 SEK), Gérard Labadie (€40), Sergey Volchkov (10 AUD), Vankayala Vigneswararao ($50), Philippe Teuwen ($4), Martin Haeberli ($10), Victor Cazacov (€5), Tobias Sturzenegger (10 CHF), Sonny Thai ($15), Bayna AlZaabi ($75), Redfive B.V. (€25), Joona Oskari Heikkilä (€5), Marshall Bishop ($50), Nicolas Werner (€12), Jeremy Brown ($100), Alexandre Borges ($25), Vladimir Dikovski (€50), Jiarui Hong (100.00 SEK), Jim Di (500 RUR), Tan Vincent ($30), Sri Harsha Kandrakota (10 AUD), Pillay Harish (10 SGD), Timur Valiev (230 RUR), Carlos Garcia Prado (€10), Salikov Alexander (500 RUR), Oliver Whitehouse (30 GBP), Katy Moe ($14), Maxim Dyakonov ($3), Sebastian Aguilera (€20), Hans-Martin Münch (€15), Jarle Thorsen (100 NOK), Vitaly Osipov ($100), Yuri Romanov (1000 RUR), Aliaksandr Autayeu (€10), Tudor Azoitei ($40), Z0vsky (€10), Yu Dai ($10).

¡Gracias a cada donante!

mini-FAQ

Q: ¿Por qué debería aprender lenguaje ensamblador hoy en día?

A: A menos que seas un desarrollador de SO[9], probablemente no necesitas programar en ensamblador—los compiladores modernos son mucho mejores generando optimizaciones que los humanos [10]. Además, los CPU[11]s modernos son dispositivos muy complejos y el conocimiento de ensamblador en realidad no ayuda a comprender su funcionamiento interno.

Una vez dicho eso, hay al menos dos áreas donde un buen entendimiento de ensamblador puede ser útil: Antes que nada, la investigación de seguridad/malware. También es una buena manera de obtener un mejor entendimiento de tu código compilado mientras es depurado.

Por lo tanto, este libro está dirigido a aquellos que desean comprender el lenguaje ensamblador en vez de codificar en él, razón por la cual contiene tantos ejemplos de código generado por compilador.

Q: Di click en un link dentro del documento PDF, ¿cómo regreso?

A: En Acrobat Reader, presiona Alt+FlechaIzquierda.

Q: No estoy seguro de si debería tratar de aprender ingeniería inversa o no.

A: Quizá, el tiempo promedio para familiarizarse con los contenidos de la versión LITE es de 1-2 meses.

Q: ¿Puedo imprimir este libro / usarlo para enseñanza?

A: ¡Por supuesto! Por eso es que el libro está registrado bajo Creative Commons. Puede que alguien quiera generar su propia versión del libro—lee here para más información al respecto.

Q: ¿Cómo se consigue un trabajo en ingeniería inversa?

A: Existen threads de contratación que aparecen de vez en cuando en reddit, dedicados a reversing[12] (2013 Q3, 2014). Intenta buscando ahí.

Un thread en ocasiones relacionado con contrataciones puede ser encontrado en el subreddit «netsec»: 2014 Q2.

Q: Tengo una pregunta...

[9] Sistema Operativo
[10] Un buen texto acerca de este tema: [Fog13]
[11] Central processing unit
[12] reddit.com/r/ReverseEngineering/

A: Envíamela por email (dennis(a)yurichev.com).

Acerca de la traducción al Coreano

En enero del 2015, la editorial Acorn (www.acornpub.co.kr) en Corea del Sur realizó una enorme cantidad de trabajo traduciendo y publicando mi libro (como era en agosto del 2014) en Coreano.

Ahora se encuentra disponible en su sitio web.

El traductor es Byungho Min (twitter/tais9).

El arte de la portada fue hecho por mi artístico amigo, Andy Nechaevsky: facebook/andydinka.

Ellos también poseen los derechos de autor de la traducción al coreano.

Así que, si quieren tener un libro *real* en coreano en su estante y quieren apoyar mi trabajo, ya se encuentra disponible a la venta.

Parte I

Patrones de código

Todo es relativo

Autor desconocido

Cuando el autor de este libro comenzó a aprender C y, más tarde, C++, él solía escribir pequeños trozos de código, compilarlos, y luego ver los resultados en lenguaje assembly. Esto lo hizo muy fácil para él entender lo que estaba pasando en el código que había escrito. [13]. Él lo hizo tantas veces que la relación entre el código C/C++ y lo que el compilador producido se imprimió profundamente en su mente. És fácil imaginar al instante un esbozo de la aparencia y función del código C. Quizás esta técnica podría ser útil para otra persona.

En ciertas partes, se han empleado aquí compiladores muy antiguas, con el fin de obtener lo mas corta (o simple) posible snippet.

Níveles de optimización y la información de depuración

El código fuente puede ser compilado por diferentes compiladores com varios niveles de optimización. Un compilador típico tiene alredor de tres de esos niveles, donde el nivel cero significa desactivar la optimización. La optimización también puede dirigirse hacia el tamaño del código o la velocidad de código. Un compilador sin optimización es más rápido y produce código más inteligible (aunque más grande), mientras un compilador con optimización es más lento y trata de producir un código que corre más rápido (pero no necesariamente más compacto). Además de los niveles y dirección de la otimización, el compilador puede incluir informaciones de depuración en el archivo resultante, produciendo así código para fácil depuración. Una de los características importantes del código de ´debug' és que puede contener enlaces entre cada línea del código fuente y las direcciones de código de máquina respectivos. Compiladores con optimización, por otro lado, tienden a producir una salida donde líneas enteras de código fuente pueden ser optimizados al punto de ser eliminados y por consiguiente no estar presentes en el código de máquina resultante. Ingenieros Inversos pueden encontrar ambas versiones, simplesmente porque alguns desarrolladores activan los flags de optimización del compilador, y otros no activan. Debido a esto, vamos a tratar de trabajar con ejemplos de ambas versiones de debug y release del código resaltado en este libro, cuando sea posible.

[13] De hecho, todavia lo hace cuando no puede entender lo que hace una determinada pieza de código.

CAPÍTULO 1. UNA BREVE INTRODUCCIÓN A LA CPU

Capítulo 1

Una breve introducción a la CPU

La CPU es el dispositivo que ejecuta el código de máquina que constituye un programa.

Un breve glosario:

Instrucción : Una primitiva CPU comando. Los ejemplos más simples incluyen: mover datos entre registros, trabajar con la memoria, operaciones aritméticas primitivas. Como regla general, cada CPU tiene su proprio conjunto de instrucciones (ISA[1]).

Spanish text placeholder : Código que la CPU procesa directamente. Cada instrucción generalmente se codifica por vários bytes.

Lenguaje assembly : Código mnemónico y algunas extensiones como macros que destinados a hacer la vida del programador más fácil.

Registros de la CPU : Cada CPU tiene un conjunto fijo de registros de propósito general (GPR[2]). ≈ 8 Spanish text placeholder x86, ≈ 16 Spanish text placeholder x86-64, ≈ 16 Spanish text placeholder ARM. La forma más fácil de entender un registro es pensar en ello como una variable temporal sin tipo. Imagine si estuviera trabajando con una LP[3] de alto nivel y sólo podría utilizar ocho variables de 32-bit (o de 64-bit). Sin embargo mucho se puede hacer usando sólo estos!

Uno podría perguntarse por qué es necessário que haya diferencia entre el código de la máquina y una lenguaje de programación de alto nivel. La respuesta está en el hecho de que los seres humanos y CPUs no son iguales—És mucho más fácil para los humanos utilizar un LP de alto nivel como C/C++, Java, Python, etc., pero és más fácil para una CPU utilizar un nivel mucho más bajo de abstración. Tal vez sería posible inventar una CPU que podría ejecutar código de LP de alto nivel, pero sería muchas veces más compleja que las CPUs que conocemos hoy. En uma manera similar, es muy incómodo para los seres humanos escribir en lenguaje assembly, debido a que es tan bajo nivel y difícil escribir sin hacer una gran cantidade de errores molestos. El programa que convierte el código de LP de alto nivel en assembly se llama *compiler*.

[1] Instruction Set Architecture
[2] General Purpose Registers
[3] Lenguaje de Programación

Capítulo 2

La función más simple

La función más simple posible es sin duda uno que simplemente devuelve un valor constante:

Ejemplo:

Listing 2.1:

```
int f()
{
        return 123;
};
```

Vamos a compilar!

2.1 x86

Esto es lo que optimiza el GCC y los compiladores de MSVC los cuales se producen en la plataforma x86:

Listing 2.2: Con optimización GCC/MSVC (salida del ensamblador)

```
f:
        mov     eax, 123
        ret
```

Hay solo dos instrucciones: los primeros lugares el valor 123 en el registro EAX, que se utiliza por convención para almacenar el valor de retorno y el segundo es RET, que devuelve la ejecución al llamado.

La persona que llama tomar el resultado del registro EAX.

Vale la pena señalar que MOV es un nombre engañoso para la instrucción tanto en x86 y ARM ISAs.

Los datos de hecho, no se *mueven*, sino que se *copian*.

Capítulo 3

¡Hola, mundo!

"The C programming Language"[Ker88]:

```
#include <stdio.h>

int main()
{
    printf("hello, world\n");
    return 0;
}
```

3.1 x86

3.1.1 MSVC

MSVC 2010:

```
cl 1.cpp /Fa1.asm
```

Listing 3.1: MSVC 2010

```
CONST   SEGMENT
$SG3830 DB      'hello, world', 0AH, 00H
CONST   ENDS
PUBLIC  _main
EXTRN   _printf:PROC
; Function compile flags: /Odtp
_TEXT   SEGMENT
_main   PROC
        push    ebp
        mov     ebp, esp
        push    OFFSET $SG3830
        call    _printf
        add     esp, 4
        xor     eax, eax
        pop     ebp
        ret     0
_main   ENDP
_TEXT   ENDS
```

:

```
#include <stdio.h>

const char $SG3830[]="hello, world\n";

int main()
{
    printf($SG3830);
    return 0;
}
```

. : 25.1.1 on page 103.

:main(). [1].

:CALL _printf.

ADD ESP, 4 ADD ESP, 4

:

Listing 3.2: Oracle RDBMS 10.2 Linux (app.o)

```
.text:0800029A                push    ebx
.text:0800029B                call    qksfroChild
.text:080002A0                pop     ecx
```

XOR EAX, EAX. [2] MOV EAX, 0—

SUB EAX, EAX,

3.2 x86-64

3.2.1 MSVC—x86-64

:

Listing 3.3: MSVC 2012 x64

```
$SG2989 DB      'hello, world', 0AH, 00H

main    PROC
        sub     rsp, 40
        lea     rcx, OFFSET FLAT:$SG2989
        call    printf
        xor     eax, eax
        add     rsp, 40
        ret     0
main    ENDP
```

. (fastcall). —. RCX, RDX, R8, R9. .

. .

RAX/EAX/AX/AL x86-64:

7mo (número de byte)	6to	5to	4to	3ro	2do	1ro	0
RAX^{x64}							
				EAX			
						AX	
						AH	AL

. «shadow space», : 8.2.1 on page 25.

3.3 Conclusión

[1] (4 on the next page).
[2] wikipedia

Capítulo 4

```
    push    ebp
    mov     ebp, esp
    sub     esp, X
```

```
    mov    esp, ebp
    pop    ebp
    ret    0
```

Capítulo 5

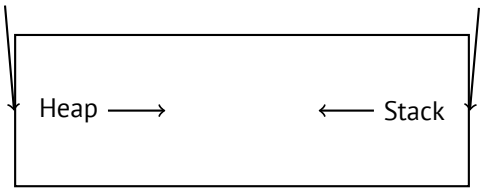
Pila

[1].

5.1

.

. . .



[RT74] :

> The user-core part of an image is divided into three logical segments. The program text segment begins at location 0 in the virtual address space. During execution, this segment is write-protected and a single copy of it is shared among all processes executing the same program. At the first 8K byte boundary above the program text segment in the virtual address space begins a nonshared, writable data segment, the size of which may be extended by a system call. Starting at the highest address in the virtual address space is a stack segment, which automatically grows downward as the hardware's stack pointer fluctuates.

5.2

5.2.1

x86

```
void f()
{
        f();
};
```

```
c:\tmp6>cl ss.cpp /Fass.asm
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.21022.08 for 80x86
Copyright (C) Microsoft Corporation.  All rights reserved.

ss.cpp
c:\tmp6\ss.cpp(4) : warning C4717: 'f' : recursive on all control paths, function will cause ↙
    ↳ runtime stack overflow
```

[1] wikipedia.org/wiki/Call_stack

...:

```
?f@@YAXXZ PROC                                          ; f
; File c:\tmp6\ss.cpp
; Line 2
        push    ebp
        mov     ebp, esp
; Line 3
        call    ?f@@YAXXZ                               ; f
; Line 4
        pop     ebp
        ret     0
?f@@YAXXZ ENDP                                          ; f
```

...

```
?f@@YAXXZ PROC                                          ; f
; File c:\tmp6\ss.cpp
; Line 2
$LL3@f:
; Line 3
        jmp     SHORT $LL3@f
?f@@YAXXZ ENDP                                          ; f
```

5.2.2

«cdecl»:

```
push arg3
push arg2
push arg1
call f
add esp, 12 ; 4*3=12
```

ESP	
ESP+4	argumento#1, Marcado en IDA[2] como arg_0
ESP+8	argumento#2, Marcado en IDA como arg_4
ESP+0xC	argumento#3, Marcado en IDA como arg_8
...	...

[3].

```
printf("%d %d %d", 1234);
```

printf()

: main(), main(int argc, char *argv[]) main(int argc, char *argv[], char *envp[]).

```
push envp
push argv
push argc
call main
...
```

main(int argc, char *argv[]), main(int argc),.

5.2.3

5.2.4 x86:

[4].

[3]

[4] alloca16.asm y chkstk.asm en C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\crt\src\intel

```
#ifdef __GNUC__
#include <alloca.h> // GCC
#else
#include <malloc.h> // MSVC
#endif
#include <stdio.h>

void f()
{
    char *buf=(char*)alloca (600);
#ifdef __GNUC__
    snprintf (buf, 600, "hi! %d, %d, %d\n", 1, 2, 3); // GCC
#else
    _snprintf (buf, 600, "hi! %d, %d, %d\n", 1, 2, 3); // MSVC
#endif

    puts (buf);
};
```

MSVC

(MSVC 2010):

Listing 5.1: MSVC 2010

```
...

    mov    eax, 600         ; 00000258H
    call   __alloca_probe_16
    mov    esi, esp

    push   3
    push   2
    push   1
    push   OFFSET $SG2672
    push   600              ; 00000258H
    push   esi
    call   __snprintf

    push   esi
    call   _puts
    add    esp, 28          ; 0000001cH

...
```

[5].

5.2.5 (Windows) SEH

.

5.2.6

(16.2 on page 58).

[5].

5.2.7

5.3

...	...
ESP-0xC	#2, Marcado en IDA como `var_8`
ESP-8	#1, Marcado en IDA como `var_4`
ESP-4	EBP
ESP	
ESP+4	argumento#1, Marcado en IDA como `arg_0`
ESP+8	argumento#2, Marcado en IDA como `arg_4`
ESP+0xC	argumento#3, Marcado en IDA como `arg_8`
...	...

Capítulo 6

printf() con varios argumentos

```
#include <stdio.h>

int main()
{
        printf("a=%d; b=%d; c=%d", 1, 2, 3);
        return 0;
};
```

6.1 x86

6.1.1 x86:

MSVC

```
$SG3830 DB      'a=%d; b=%d; c=%d', 00H

...

        push    3
        push    2
        push    1
        push    OFFSET $SG3830
        call    _printf
        add     esp, 16                                 ; 00000010H
```

.

```
push a1
push a2
call ...
...
push a1
call ...
...
push a1
push a2
push a3
call ...
add esp, 24
```

Listing 6.1: x86

```
.text:100113E7                push    3
.text:100113E9                call    sub_100018B0 ;  (3)
.text:100113EE                call    sub_100019D0 ;
.text:100113F3                call    sub_10006A90 ;
.text:100113F8                push    1
.text:100113FA                call    sub_100018B0 ;  (1)
```

```
.text:100113FF                 add     esp, 8          ;
```

6.1.2 x64:

:

```
#include <stdio.h>

int main()
{
        printf("a=%d; b=%d; c=%d; d=%d; e=%d; f=%d; g=%d; h=%d\n", 1, 2, 3, 4, 5, 6, 7, 8);
        return 0;
};
```

MSVC

RCX, RDX, R8, R9 . . .

Listing 6.2: MSVC 2012 x64

```
$SG2923 DB      'a=%d; b=%d; c=%d; d=%d; e=%d; f=%d; g=%d; h=%d', 0aH, 00H

main    PROC
        sub     rsp, 88

        mov     DWORD PTR [rsp+64], 8
        mov     DWORD PTR [rsp+56], 7
        mov     DWORD PTR [rsp+48], 6
        mov     DWORD PTR [rsp+40], 5
        mov     DWORD PTR [rsp+32], 4
        mov     r9d, 3
        mov     r8d, 2
        mov     edx, 1
        lea     rcx, OFFSET FLAT:$SG2923
        call    printf

        ;  0
        xor     eax, eax

        add     rsp, 88
        ret     0
main    ENDP
_TEXT   ENDS
END
```

6.2 Conclusión

:

Listing 6.3: x86

```
...
PUSH
PUSH
PUSH
CALL
;
```

Listing 6.4: x64 (MSVC)

```
MOV RCX,
```

```
MOV RDX,
MOV R8,
MOV R9,
...
PUSH
CALL
;
```

6.3

CPU .

Capítulo 7

scanf()

7.1

```
#include <stdio.h>

int main()
{
        int x;
        printf ("Enter X:\n");

        scanf ("%d", &x);

        printf ("You entered %d...\n", x);

        return 0;
};
```

7.1.1

C/C++

; memcpy(), , void*,

( ). *scanf()*.

C/C++

7.1.2 x86

MSVC

```
CONST    SEGMENT
$SG3831    DB    'Enter X:', 0aH, 00H
$SG3832    DB    '%d', 00H
$SG3833    DB    'You entered %d...', 0aH, 00H
CONST    ENDS
PUBLIC    _main
EXTRN    _scanf:PROC
EXTRN    _printf:PROC
; Function compile flags: /Odtp
_TEXT    SEGMENT
_x$ = -4                        ; size = 4
_main    PROC
    push   ebp
    mov    ebp, esp
    push   ecx
    push   OFFSET $SG3831 ; 'Enter X:'
    call   _printf
    add    esp, 4
```

```
    lea     eax, DWORD PTR _x$[ebp]
    push    eax
    push    OFFSET $SG3832 ; '%d'
    call    _scanf
    add     esp, 8
    mov     ecx, DWORD PTR _x$[ebp]
    push    ecx
    push    OFFSET $SG3833 ; 'You entered %d...'
    call    _printf
    add     esp, 8

    ;  0
    xor     eax, eax
    mov     esp, ebp
    pop     ebp
    ret     0
_main    ENDP
_TEXT    ENDS
```

:

...	...
EBP-8	#2, Marcado en IDA como var_8
EBP-4	#1, Marcado en IDA como var_4
EBP	EBP
EBP+4	
EBP+8	argumento#1, Marcado en IDA como arg_0
EBP+0xC	argumento#2, Marcado en IDA como arg_4
EBP+0x10	argumento#3, Marcado en IDA como arg_8
...	...

`lea eax, [ebp-4]`.

`You entered %d...\n`.

7.1.3 x64

.

MSVC

Listing 7.1: MSVC 2012 x64

```
_DATA   SEGMENT
$SG1289 DB      'Enter X:', 0aH, 00H
$SG1291 DB      '%d', 00H
$SG1292 DB      'You entered %d...', 0aH, 00H
_DATA   ENDS

_TEXT   SEGMENT
x$ = 32
main    PROC
$LN3:
        sub     rsp, 56
        lea     rcx, OFFSET FLAT:$SG1289 ; 'Enter X:'
        call    printf
        lea     rdx, QWORD PTR x$[rsp]
        lea     rcx, OFFSET FLAT:$SG1291 ; '%d'
        call    scanf
        mov     edx, DWORD PTR x$[rsp]
        lea     rcx, OFFSET FLAT:$SG1292 ; 'You entered %d...'
        call    printf

        ;  0
```

```
        xor     eax, eax
        add     rsp, 56
        ret     0
main    ENDP
_TEXT   ENDS
```

7.2

```
#include <stdio.h>

//
int x;

int main()
{
        printf ("Enter X:\n");

        scanf ("%d", &x);

        printf ("You entered %d...\n", x);

        return 0;
};
```

7.2.1 MSVC: x86

```
_DATA    SEGMENT
COMM    _x:DWORD
$SG2456    DB    'Enter X:', 0aH, 00H
$SG2457    DB    '%d', 00H
$SG2458    DB    'You entered %d...', 0aH, 00H
_DATA    ENDS
PUBLIC    _main
EXTRN    _scanf:PROC
EXTRN    _printf:PROC
; Function compile flags: /Odtp
_TEXT    SEGMENT
_main    PROC
    push   ebp
    mov    ebp, esp
    push   OFFSET $SG2456
    call   _printf
    add    esp, 4
    push   OFFSET _x
    push   OFFSET $SG2457
    call   _scanf
    add    esp, 8
    mov    eax, DWORD PTR _x
    push   eax
    push   OFFSET $SG2458
    call   _printf
    add    esp, 8
    xor    eax, eax
    pop    ebp
    ret    0
_main    ENDP
_TEXT    ENDS
```

```
int x=10; //
```

```
_DATA   SEGMENT
_x      DD      0aH

...
```

```
.data:0040FA80 _x              dd ?                    ; DATA XREF: _main+10
.data:0040FA80                                         ; _main+22
.data:0040FA84 dword_40FA84    dd ?                    ; DATA XREF: _memset+1E
.data:0040FA84                                         ; unknown_libname_1+28
.data:0040FA88 dword_40FA88    dd ?                    ; DATA XREF: ___sbh_find_block+5
.data:0040FA88                                         ; ___sbh_free_block+2BC
.data:0040FA8C ; LPVOID lpMem
.data:0040FA8C lpMem           dd ?                    ; DATA XREF: ___sbh_find_block+B
.data:0040FA8C                                         ; ___sbh_free_block+2CA
.data:0040FA90 dword_40FA90    dd ?                    ; DATA XREF: _V6_HeapAlloc+13
.data:0040FA90                                         ; __calloc_impl+72
.data:0040FA94 dword_40FA94    dd ?                    ; DATA XREF: ___sbh_free_block+2FE
```

7.2.2 MSVC: x64

Listing 7.2: MSVC 2012 x64

```
_DATA   SEGMENT
COMM    x:DWORD
$SG2924 DB      'Enter X:', 0aH, 00H
$SG2925 DB      '%d', 00H
$SG2926 DB      'You entered %d...', 0aH, 00H
_DATA   ENDS

_TEXT   SEGMENT
main    PROC
$LN3:
        sub     rsp, 40

        lea     rcx, OFFSET FLAT:$SG2924 ; 'Enter X:'
        call    printf
        lea     rdx, OFFSET FLAT:x
        lea     rcx, OFFSET FLAT:$SG2925 ; '%d'
        call    scanf
        mov     edx, DWORD PTR x
        lea     rcx, OFFSET FLAT:$SG2926 ; 'You entered %d...'
        call    printf

        ;  0
        xor     eax, eax

        add     rsp, 40
        ret     0
main    ENDP
_TEXT   ENDS
```

x86. . DWORD PTR.

7.3

```
#include <stdio.h>

int main()
{
        int x;
        printf ("Enter X:\n");

        if (scanf ("%d", &x)==1)
```

```
                printf ("You entered %d...\n", x);
        else
                printf ("What you entered? Huh?\n");

        return 0;
};
```

scanf()[1]

:

```
C:\...>ex3.exe
Enter X:
123
You entered 123...

C:\...>ex3.exe
Enter X:
ouch
What you entered? Huh?
```

7.3.1 MSVC: x86

(MSVC 2010):

```
        lea     eax, DWORD PTR _x$[ebp]
        push    eax
        push    OFFSET $SG3833 ; '%d', 00H
        call    _scanf
        add     esp, 8
        cmp     eax, 1
        jne     SHORT $LN2@main
        mov     ecx, DWORD PTR _x$[ebp]
        push    ecx
        push    OFFSET $SG3834 ; 'You entered %d...', 0aH, 00H
        call    _printf
        add     esp, 8
        jmp     SHORT $LN1@main
$LN2@main:
        push    OFFSET $SG3836 ; 'What you entered? Huh?', 0aH, 00H
        call    _printf
        add     esp, 4
$LN1@main:
        xor     eax, eax
```

[1] scanf, wscanf: MSDN

7.3.2 MSVC: x86 + Hiew

. .

MSVCR*.DLL (/MD)[2], main() .text. .text (Enter, F8, F6, Enter, Enter).

:

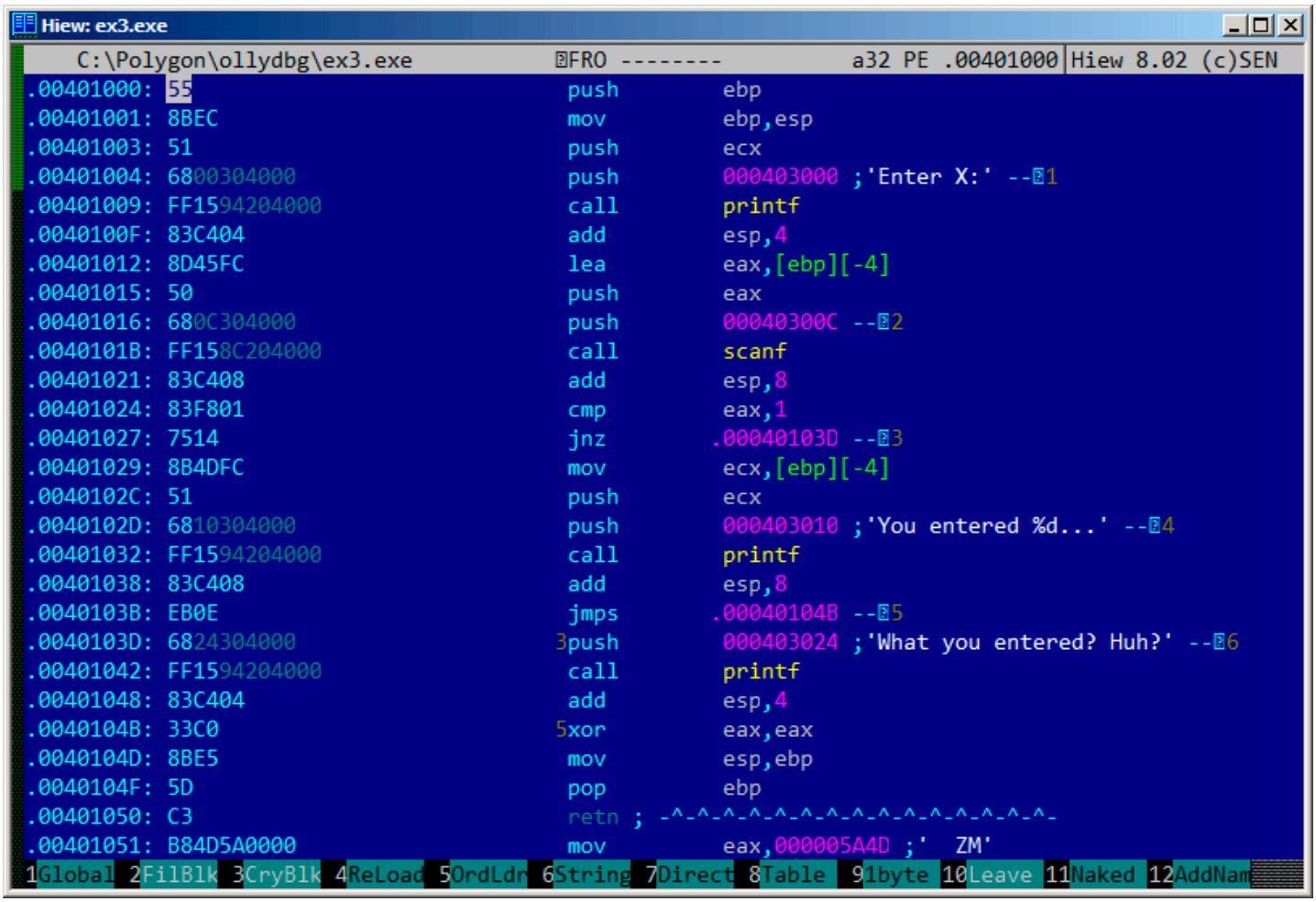


Figura 7.1: Hiew: main()

Hiew ASCIIZ[3].

[2] «dynamic linking»
[3] ASCII Zero ()

.00401027 (JNZ), F3, «9090»(NOP[4]):

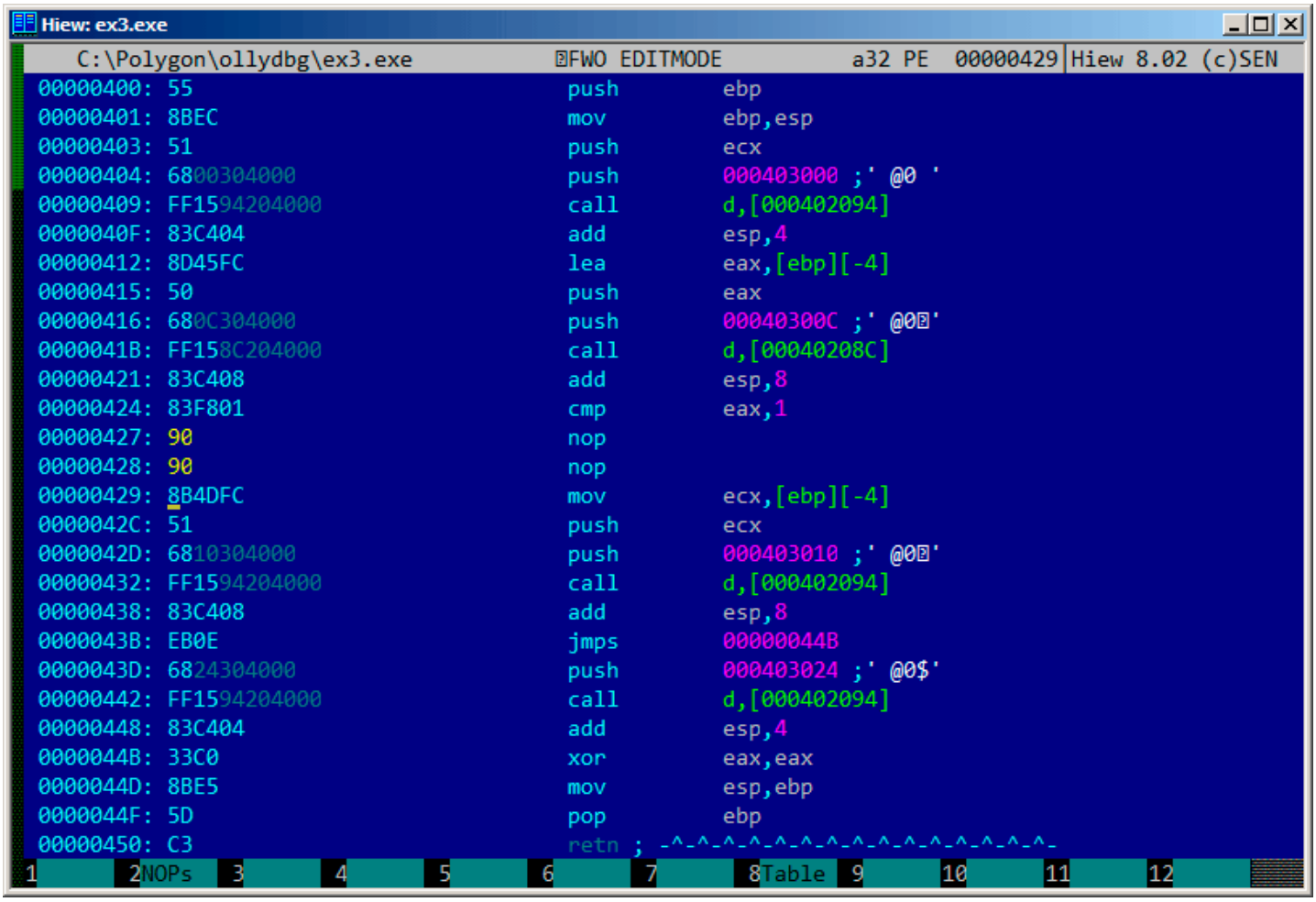


Figura 7.2: Hiew: JNZ NOP

F9 (update).

NOP . JNZ .

.

7.3.3 MSVC: x64

Listing 7.3: MSVC 2012 x64

```
_DATA   SEGMENT
$SG2924 DB      'Enter X:', 0aH, 00H
$SG2926 DB      '%d', 00H
$SG2927 DB      'You entered %d...', 0aH, 00H
$SG2929 DB      'What you entered? Huh?', 0aH, 00H
_DATA   ENDS

_TEXT   SEGMENT
x$ = 32
main    PROC
$LN5:
        sub     rsp, 56
        lea     rcx, OFFSET FLAT:$SG2924 ; 'Enter X:'
        call    printf
        lea     rdx, QWORD PTR x$[rsp]
        lea     rcx, OFFSET FLAT:$SG2926 ; '%d'
        call    scanf
```

[4]No OPeration

```
        cmp     eax, 1
        jne     SHORT $LN2@main
        mov     edx, DWORD PTR x$[rsp]
        lea     rcx, OFFSET FLAT:$SG2927 ; 'You entered %d...'
        call    printf
        jmp     SHORT $LN1@main
$LN2@main:
        lea     rcx, OFFSET FLAT:$SG2929 ; 'What you entered? Huh?'
        call    printf
$LN1@main:
        ;  0
        xor     eax, eax
        add     rsp, 56
        ret     0
main    ENDP
_TEXT   ENDS
END
```

7.4 Ejercicio

- http://challenges.re/53

Capítulo 8

Listing 8.1:

```
#include <stdio.h>

int f (int a, int b, int c)
{
        return a*b+c;
};

int main()
{
        printf ("%d\n", f(1, 2, 3));
        return 0;
};
```

8.1 x86

8.1.1 MSVC

(MSVC 2010 Express):

Listing 8.2: MSVC 2010 Express

```
_TEXT   SEGMENT
_a$ = 8                                                 ; size = 4
_b$ = 12                                                ; size = 4
_c$ = 16                                                ; size = 4
_f      PROC
        push    ebp
        mov     ebp, esp
        mov     eax, DWORD PTR _a$[ebp]
        imul    eax, DWORD PTR _b$[ebp]
        add     eax, DWORD PTR _c$[ebp]
        pop     ebp
        ret     0
_f      ENDP

_main   PROC
        push    ebp
        mov     ebp, esp
        push    3 ;
        push    2 ;
        push    1 ;
        call    _f
        add     esp, 12
        push    eax
        push    OFFSET $SG2463 ; '%d', 0aH, 00H
        call    _printf
        add     esp, 8
        ;  0
        xor     eax, eax
        pop     ebp
```

```
        ret     0
_main   ENDP
```

8.2 x64

8.2.1 MSVC

Con optimización MSVC:

Listing 8.3: Con optimización MSVC 2012 x64

```
$SG2997 DB      '%d', 0aH, 00H

main    PROC
        sub     rsp, 40
        mov     edx, 2
        lea     r8d, QWORD PTR [rdx+1] ; R8D=3
        lea     ecx, QWORD PTR [rdx-1] ; ECX=1
        call    f
        lea     rcx, OFFSET FLAT:$SG2997 ; '%d'
        mov     edx, eax
        call    printf
        xor     eax, eax
        add     rsp, 40
        ret     0
main    ENDP

f       PROC
        ; ECX -
        ; EDX -
        ; R8D -
        imul    ecx, edx
        lea     eax, DWORD PTR [r8+rcx]
        ret     0
f       ENDP
```

:

Listing 8.4: MSVC 2012 x64

```
f               proc near

; shadow space:
arg_0           = dword ptr  8
arg_8           = dword ptr  10h
arg_10          = dword ptr  18h

                ; ECX -
                ; EDX -
                ; R8D -
                mov     [rsp+arg_10], r8d
                mov     [rsp+arg_8], edx
                mov     [rsp+arg_0], ecx
                mov     eax, [rsp+arg_0]
                imul    eax, [rsp+arg_8]
                add     eax, [rsp+arg_10]
                retn
f               endp

main            proc near
                sub     rsp, 28h
                mov     r8d, 3 ;
                mov     edx, 2 ;
                mov     ecx, 1 ;
                call    f
                mov     edx, eax
```

```
                lea     rcx, $SG2931    ; "%d\n"
                call    printf

                ;  0
                xor     eax, eax
                add     rsp, 28h
                retn
main            endp
```

«shadow space» [1]: : 1) 2) [2].

[1] MSDN
[2] MSDN

Capítulo 9

[1]

9.1

```
push envp
push argv
push argc
call main
push eax
call exit
```

```
exit(main(argc,argv,envp));
```

```
#include <stdio.h>

void main()
{
        printf ("Hello, world!\n");
};
```

Linux.

GCC 4.8.1 `printf()` `puts()`, `puts()` `printf()`. EAX `main()`.

Listing 9.1: GCC 4.8.1

```
.LC0:
        .string "Hello, world!"
main:
        push    ebp
        mov     ebp, esp
        and     esp, -16
        sub     esp, 16
        mov     DWORD PTR [esp], OFFSET FLAT:.LC0
        call    puts
        leave
        ret
```

:

Listing 9.2: tst.sh

```
#!/bin/sh
./hello_world
echo $?
```

:

```
$ tst.sh
Hello, world!
14
```

14 .

[1] Véase también: MSDN: Return Values (C++): MSDN

9.2

```
int f()
{
    // skip first 3 random values
    rand();
    rand();
    rand();
    // and use 4th
    return rand();
};
```

Capítulo 10

[Dij68], [Knu74], [Yur13, pág. 1.3.2].

:

```
#include <stdio.h>

int main()
{
        printf ("begin\n");
        goto exit;
        printf ("skip me!\n");
exit:
        printf ("end\n");
};
```

MSVC 2012:

Listing 10.1: MSVC 2012

```
$SG2934 DB      'begin', 0aH, 00H
$SG2936 DB      'skip me!', 0aH, 00H
$SG2937 DB      'end', 0aH, 00H

_main   PROC
        push    ebp
        mov     ebp, esp
        push    OFFSET $SG2934 ; 'begin'
        call    _printf
        add     esp, 4
        jmp     SHORT $exit$3
        push    OFFSET $SG2936 ; 'skip me!'
        call    _printf
        add     esp, 4
$exit$3:
        push    OFFSET $SG2937 ; 'end'
        call    _printf
        add     esp, 4
        xor     eax, eax
        pop     ebp
        ret     0
_main   ENDP
```

10.1

Listing 10.2: Con optimización MSVC 2012

```
$SG2981 DB      'begin', 0aH, 00H
$SG2983 DB      'skip me!', 0aH, 00H
$SG2984 DB      'end', 0aH, 00H

_main   PROC
        push    OFFSET $SG2981 ; 'begin'
        call    _printf
        push    OFFSET $SG2984 ; 'end'
```

```
$exit$4:
        call    _printf
        add     esp, 8
        xor     eax, eax
        ret     0
_main   ENDP
```

«skip me!» .

Capítulo 11

11.1

```
#include <stdio.h>

void f_signed (int a, int b)
{
    if (a>b)
        printf ("a>b\n");
    if (a==b)
        printf ("a==b\n");
    if (a<b)
        printf ("a<b\n");
};

void f_unsigned (unsigned int a, unsigned int b)
{
    if (a>b)
        printf ("a>b\n");
    if (a==b)
        printf ("a==b\n");
    if (a<b)
        printf ("a<b\n");
};

int main()
{
    f_signed(1, 2);
    f_unsigned(1, 2);
    return 0;
};
```

11.1.1 x86

x86 + MSVC

Listing 11.1: Sin optimización MSVC 2010

```
_a$ = 8
_b$ = 12
_f_signed PROC
    push   ebp
    mov    ebp, esp
    mov    eax, DWORD PTR _a$[ebp]
    cmp    eax, DWORD PTR _b$[ebp]
    jle    SHORT $LN3@f_signed
    push   OFFSET $SG737        ; 'a>b'
    call   _printf
    add    esp, 4
$LN3@f_signed:
    mov    ecx, DWORD PTR _a$[ebp]
    cmp    ecx, DWORD PTR _b$[ebp]
    jne    SHORT $LN2@f_signed
```

```
    push   OFFSET $SG739        ; 'a==b'
    call   _printf
    add    esp, 4
$LN2@f_signed:
    mov    edx, DWORD PTR _a$[ebp]
    cmp    edx, DWORD PTR _b$[ebp]
    jge    SHORT $LN4@f_signed
    push   OFFSET $SG741        ; 'a<b'
    call   _printf
    add    esp, 4
$LN4@f_signed:
    pop    ebp
    ret    0
_f_signed ENDP
```

Jump if Less or Equal. JNE: *Jump if Not Equal. .*

JGE: *Jump if Greater or Equal—.*

`f_unsigned()`

Listing 11.2: GCC

```
_a$ = 8   ; size = 4
_b$ = 12  ; size = 4
_f_unsigned PROC
    push   ebp
    mov    ebp, esp
    mov    eax, DWORD PTR _a$[ebp]
    cmp    eax, DWORD PTR _b$[ebp]
    jbe    SHORT $LN3@f_unsigned
    push   OFFSET $SG2761    ; 'a>b'
    call   _printf
    add    esp, 4
$LN3@f_unsigned:
    mov    ecx, DWORD PTR _a$[ebp]
    cmp    ecx, DWORD PTR _b$[ebp]
    jne    SHORT $LN2@f_unsigned
    push   OFFSET $SG2763    ; 'a==b'
    call   _printf
    add    esp, 4
$LN2@f_unsigned:
    mov    edx, DWORD PTR _a$[ebp]
    cmp    edx, DWORD PTR _b$[ebp]
    jae    SHORT $LN4@f_unsigned
    push   OFFSET $SG2765    ; 'a<b'
    call   _printf
    add    esp, 4
$LN4@f_unsigned:
    pop    ebp
    ret    0
_f_unsigned ENDP
```

JBE—*Jump if Below or Equal* y JAE—*Jump if Above or Equal.* (JA/JAE/JB/JBE) JG/JGE/JL/JLE

(22 on page 96).

Listing 11.3: `main()`

```
_main   PROC
        push    ebp
        mov     ebp, esp
        push    2
        push    1
        call    _f_signed
        add     esp, 8
        push    2
        push    1
        call    _f_unsigned
        add     esp, 8
        xor     eax, eax
        pop     ebp
```

```
        ret     0
_main   ENDP
```

x86 + MSVC + Hiew

`f_unsigned()` «a==b», . Hiew:

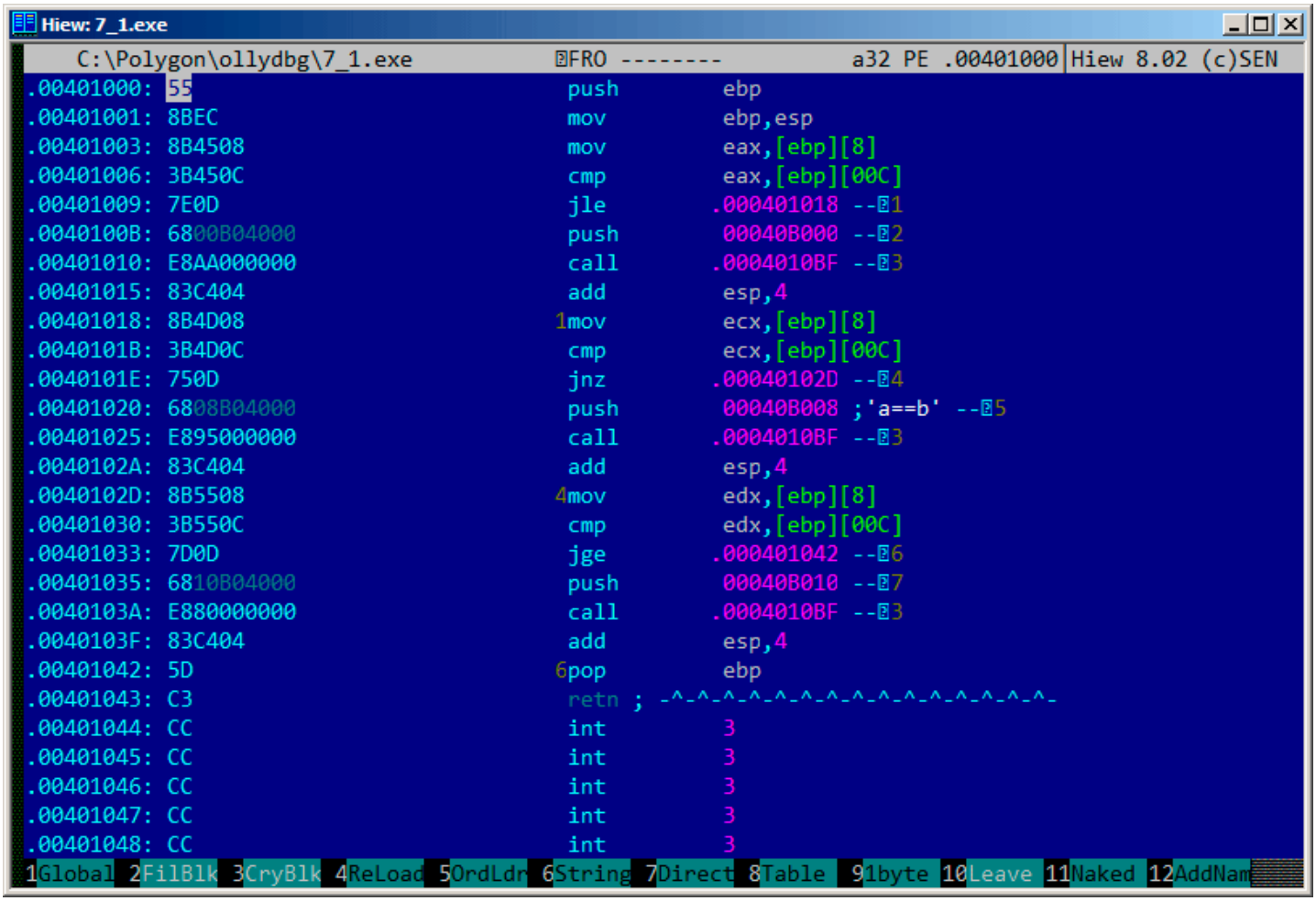


Figura 11.1: Hiew: `f_unsigned()`

:

- ;
- ;
- .

`printf()`, «a==b».

:

- JMP, .
-
-

:

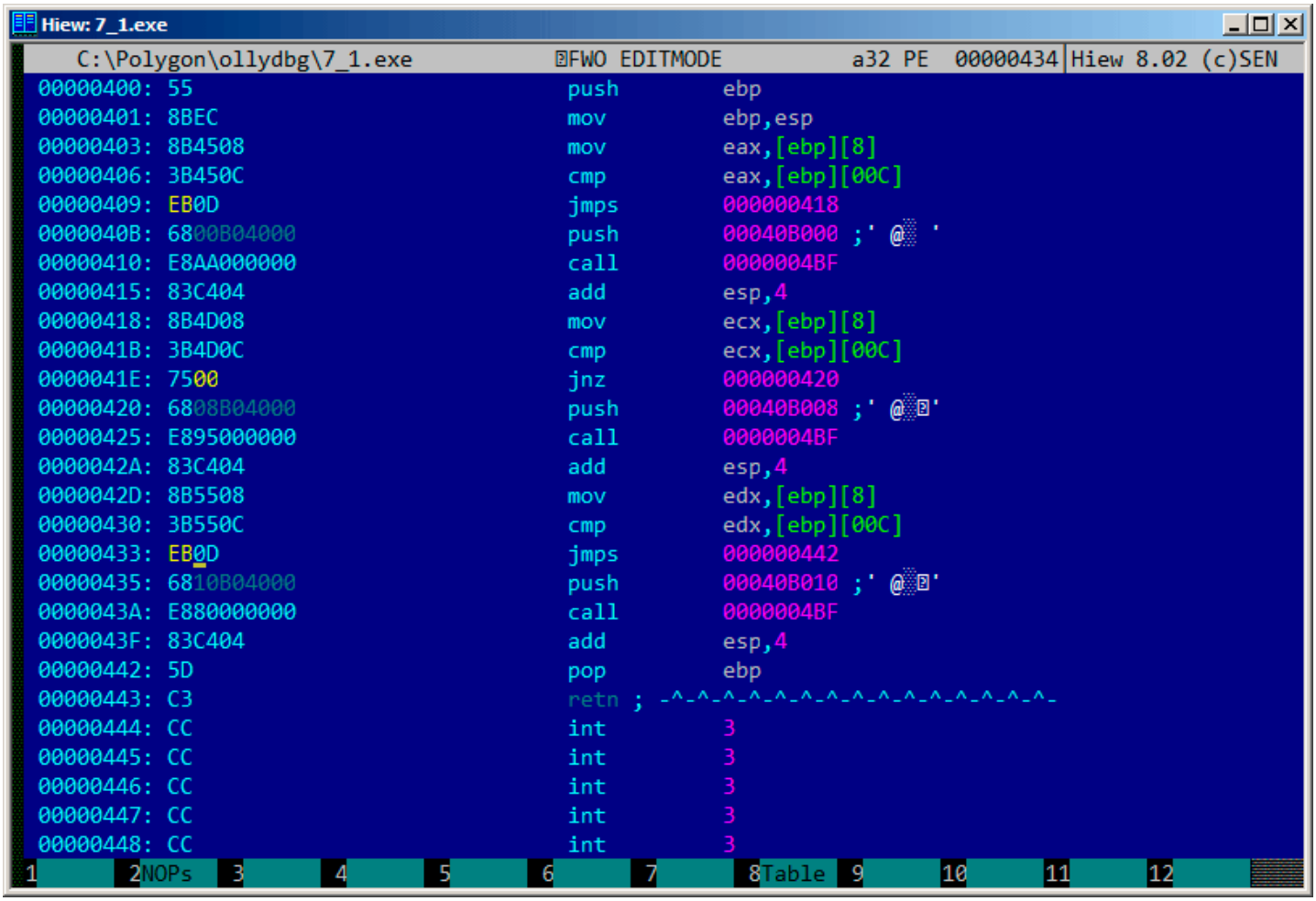


Figura 11.2: Hiew: `f_unsigned()`

11.2

:

```
int my_abs (int i)
{
        if (i<0)
                return -i;
        else
                return i;
};
```

11.2.1 Con optimización MSVC

Listing 11.4: Con optimización MSVC 2012 x64

```
i$ = 8
my_abs  PROC
; ECX = input
        test    ecx, ecx
;
;
        jns     SHORT $LN2@my_abs
;
        neg     ecx
```

```
$LN2@my_abs:
;  EAX:
        mov     eax, ecx
        ret     0
my_abs  ENDP
```

11.3

C/C++:

```
expression ? expression : expression
```

:

```
const char* f (int a)
{
        return a==10 ? "it is ten" : "it is not ten";
};
```

11.3.1 x86

Listing 11.5: Sin optimización MSVC 2008

```
$SG746  DB      'it is ten', 00H
$SG747  DB      'it is not ten', 00H

tv65 = -4 ;
_a$ = 8
_f      PROC
        push    ebp
        mov     ebp, esp
        push    ecx
;  10
        cmp     DWORD PTR _a$[ebp], 10
;  $LN3@f
        jne     SHORT $LN3@f
; :
        mov     DWORD PTR tv65[ebp], OFFSET $SG746 ; 'it is ten'
;
        jmp     SHORT $LN4@f
$LN3@f:
; :
        mov     DWORD PTR tv65[ebp], OFFSET $SG747 ; 'it is not ten'
$LN4@f:
;
        mov     eax, DWORD PTR tv65[ebp]
        mov     esp, ebp
        pop     ebp
        ret     0
_f      ENDP
```

Listing 11.6: Con optimización MSVC 2008

```
$SG792  DB      'it is ten', 00H
$SG793  DB      'it is not ten', 00H

_a$ = 8 ; size = 4
_f      PROC
;  10
        cmp     DWORD PTR _a$[esp-4], 10
        mov     eax, OFFSET $SG792 ; 'it is ten'
;  $LN4@f
        je      SHORT $LN4@f
```

```
        mov     eax, OFFSET $SG793 ; 'it is not ten'
$LN4@f:
        ret     0
_f      ENDP
```

:

Listing 11.7: Con optimización MSVC 2012 x64

```
$SG1355 DB      'it is ten', 00H
$SG1356 DB      'it is not ten', 00H

a$ = 8
f       PROC
;
        lea     rdx, OFFSET FLAT:$SG1355 ; 'it is ten'
        lea     rax, OFFSET FLAT:$SG1356 ; 'it is not ten'
;  10
        cmp     ecx, 10
;  ("it is ten")
;
        cmove   rax, rdx
        ret     0
f       ENDP
```

Con optimización GCC 4.8 para x86 CMOVcc.

11.3.2

```
const char* f (int a)
{
        if (a==10)
                return "it is ten";
        else
                return "it is not ten";
};
```

Listing 11.8: Con optimización GCC 4.8

```
.LC0:
        .string "it is ten"
.LC1:
        .string "it is not ten"
f:
.LFB0:
;  10
        cmp     DWORD PTR [esp+4], 10
        mov     edx, OFFSET FLAT:.LC1 ; "it is not ten"
        mov     eax, OFFSET FLAT:.LC0 ; "it is ten"
;
;
        cmovne  eax, edx
        ret
```

MSVC 2012 .

11.4

11.4.1 32-bit

```
int my_max(int a, int b)
{
        if (a>b)
```

```
                return a;
        else
                return b;
};

int my_min(int a, int b)
{
        if (a<b)
                return a;
        else
                return b;
};
```

Listing 11.9: Sin optimización MSVC 2013

```
_a$ = 8
_b$ = 12
_my_min PROC
        push    ebp
        mov     ebp, esp
        mov     eax, DWORD PTR _a$[ebp]
; :
        cmp     eax, DWORD PTR _b$[ebp]
; :
        jge     SHORT $LN2@my_min
;
        mov     eax, DWORD PTR _a$[ebp]
        jmp     SHORT $LN3@my_min
        jmp     SHORT $LN3@my_min ;  JMP
$LN2@my_min:
;  B
        mov     eax, DWORD PTR _b$[ebp]
$LN3@my_min:
        pop     ebp
        ret     0
_my_min ENDP

_a$ = 8
_b$ = 12
_my_max PROC
        push    ebp
        mov     ebp, esp
        mov     eax, DWORD PTR _a$[ebp]
; :
        cmp     eax, DWORD PTR _b$[ebp]
; :
        jle     SHORT $LN2@my_max
;
        mov     eax, DWORD PTR _a$[ebp]
        jmp     SHORT $LN3@my_max
        jmp     SHORT $LN3@my_max ;  JMP
$LN2@my_max:
;  B
        mov     eax, DWORD PTR _b$[ebp]
$LN3@my_max:
        pop     ebp
        ret     0
_my_max ENDP
```

11.5 Conclusión

11.5.1 x86

:

Listing 11.10: x86

```
CMP register, register/value
Jcc true ; cc=
false:
...  ...
JMP exit
true:
...  ...
exit:
```

11.5.2

Capítulo 12

switch()/case/default

12.1

```
#include <stdio.h>

void f (int a)
{
    switch (a)
    {
    case 0: printf ("zero\n"); break;
    case 1: printf ("one\n"); break;
    case 2: printf ("two\n"); break;
    default: printf ("something unknown\n"); break;
    };
};

int main()
{
    f (2); // test
};
```

12.1.1 x86

Sin optimización MSVC

(MSVC 2010):

Listing 12.1: MSVC 2010

```
tv64 = -4 ; size = 4
_a$ = 8   ; size = 4
_f    PROC
    push   ebp
    mov    ebp, esp
    push   ecx
    mov    eax, DWORD PTR _a$[ebp]
    mov    DWORD PTR tv64[ebp], eax
    cmp    DWORD PTR tv64[ebp], 0
    je     SHORT $LN4@f
    cmp    DWORD PTR tv64[ebp], 1
    je     SHORT $LN3@f
    cmp    DWORD PTR tv64[ebp], 2
    je     SHORT $LN2@f
    jmp    SHORT $LN1@f
$LN4@f:
    push   OFFSET $SG739 ; 'zero', 0aH, 00H
    call   _printf
    add    esp, 4
    jmp    SHORT $LN7@f
$LN3@f:
    push   OFFSET $SG741 ; 'one', 0aH, 00H
```

```
    call   _printf
    add    esp, 4
    jmp    SHORT $LN7@f
$LN2@f:
    push   OFFSET $SG743 ; 'two', 0aH, 00H
    call   _printf
    add    esp, 4
    jmp    SHORT $LN7@f
$LN1@f:
    push   OFFSET $SG745 ; 'something unknown', 0aH, 00H
    call   _printf
    add    esp, 4
$LN7@f:
    mov    esp, ebp
    pop    ebp
    ret    0
_f    ENDP
```

```
void f (int a)
{
    if (a==0)
        printf ("zero\n");
    else if (a==1)
        printf ("one\n");
    else if (a==2)
        printf ("two\n");
    else
        printf ("something unknown\n");
};
```

Con optimización MSVC

MSVC (/Ox): `cl 1.c /Fa1.asm /Ox`

Listing 12.2: MSVC

```
_a$ = 8 ; size = 4
_f    PROC
    mov    eax, DWORD PTR _a$[esp-4]
    sub    eax, 0
    je     SHORT $LN4@f
    sub    eax, 1
    je     SHORT $LN3@f
    sub    eax, 1
    je     SHORT $LN2@f
    mov    DWORD PTR _a$[esp-4], OFFSET $SG791 ; 'something unknown', 0aH, 00H
    jmp    _printf
$LN2@f:
    mov    DWORD PTR _a$[esp-4], OFFSET $SG789 ; 'two', 0aH, 00H
    jmp    _printf
$LN3@f:
    mov    DWORD PTR _a$[esp-4], OFFSET $SG787 ; 'one', 0aH, 00H
    jmp    _printf
$LN4@f:
    mov    DWORD PTR _a$[esp-4], OFFSET $SG785 ; 'zero', 0aH, 00H
    jmp    _printf
_f    ENDP
```

- ESP— RA[1]
- ESP+4—

12.1.2 Conclusión

listado.12.1.1.

[1] Dirección de Retorno

12.2

```
#include <stdio.h>

void f (int a)
{
    switch (a)
    {
    case 0: printf ("zero\n"); break;
    case 1: printf ("one\n"); break;
    case 2: printf ("two\n"); break;
    case 3: printf ("three\n"); break;
    case 4: printf ("four\n"); break;
    default: printf ("something unknown\n"); break;
    };
};

int main()
{
    f (2); // test
};
```

12.2.1 x86

Sin optimización MSVC

(MSVC 2010):

Listing 12.3: MSVC 2010

```
tv64 = -4  ; size = 4
_a$ = 8    ; size = 4
_f    PROC
    push   ebp
    mov    ebp, esp
    push   ecx
    mov    eax, DWORD PTR _a$[ebp]
    mov    DWORD PTR tv64[ebp], eax
    cmp    DWORD PTR tv64[ebp], 4
    ja     SHORT $LN1@f
    mov    ecx, DWORD PTR tv64[ebp]
    jmp    DWORD PTR $LN11@f[ecx*4]
$LN6@f:
    push   OFFSET $SG739 ; 'zero', 0aH, 00H
    call   _printf
    add    esp, 4
    jmp    SHORT $LN9@f
$LN5@f:
    push   OFFSET $SG741 ; 'one', 0aH, 00H
    call   _printf
    add    esp, 4
    jmp    SHORT $LN9@f
$LN4@f:
    push   OFFSET $SG743 ; 'two', 0aH, 00H
    call   _printf
    add    esp, 4
    jmp    SHORT $LN9@f
$LN3@f:
    push   OFFSET $SG745 ; 'three', 0aH, 00H
    call   _printf
    add    esp, 4
    jmp    SHORT $LN9@f
$LN2@f:
    push   OFFSET $SG747 ; 'four', 0aH, 00H
    call   _printf
    add    esp, 4
    jmp    SHORT $LN9@f
```

```
$LN1@f:
    push   OFFSET $SG749 ; 'something unknown', 0aH, 00H
    call   _printf
    add    esp, 4
$LN9@f:
    mov    esp, ebp
    pop    ebp
    ret    0
    npad    2 ;
$LN11@f:
    DD    $LN6@f ; 0
    DD    $LN5@f ; 1
    DD    $LN4@f ; 2
    DD    $LN3@f ; 3
    DD    $LN2@f ; 4
_f    ENDP
```

jumptable o *branch table*[2].

Sin optimización GCC

:

Listing 12.4: GCC 4.4.1

```
        public f
f       proc near ; CODE XREF: main+10

var_18 = dword ptr -18h
arg_0  = dword ptr  8

        push    ebp
        mov     ebp, esp
        sub     esp, 18h
        cmp     [ebp+arg_0], 4
        ja      short loc_8048444
        mov     eax, [ebp+arg_0]
        shl     eax, 2
        mov     eax, ds:off_804855C[eax]
        jmp     eax

loc_80483FE: ; DATA XREF: .rodata:off_804855C
        mov     [esp+18h+var_18], offset aZero ; "zero"
        call    _puts
        jmp     short locret_8048450

loc_804840C: ; DATA XREF: .rodata:08048560
        mov     [esp+18h+var_18], offset aOne ; "one"
        call    _puts
        jmp     short locret_8048450

loc_804841A: ; DATA XREF: .rodata:08048564
        mov     [esp+18h+var_18], offset aTwo ; "two"
        call    _puts
        jmp     short locret_8048450

loc_8048428: ; DATA XREF: .rodata:08048568
        mov     [esp+18h+var_18], offset aThree ; "three"
        call    _puts
        jmp     short locret_8048450

loc_8048436: ; DATA XREF: .rodata:0804856C
        mov     [esp+18h+var_18], offset aFour ; "four"
        call    _puts
        jmp     short locret_8048450

loc_8048444: ; CODE XREF: f+A
```

[2] *computed GOTO* : wikipedia. !

```
        mov     [esp+18h+var_18], offset aSomethingUnkno ; "something unknown"
        call    _puts

locret_8048450: ; CODE XREF: f+26
                ; f+34...
        leave
        retn
f       endp

off_804855C dd offset loc_80483FE   ; DATA XREF: f+12
            dd offset loc_804840C
            dd offset loc_804841A
            dd offset loc_8048428
            dd offset loc_8048436
```

12.2.2 Conclusión

switch():

Listing 12.5: x86

```
MOV REG, input
CMP REG, 4 ;
JA default
SHL REG, 2 ;
MOV REG, jump_table[REG]
JMP REG

case1:
    ;
    JMP exit
case2:
    ;
    JMP exit
case3:
    ;
    JMP exit
case4:
    ;
    JMP exit
case5:
    ;
    JMP exit

default:

    ...

exit:

    ....

jump_table dd case1
           dd case2
           dd case3
           dd case4
           dd case5
```

`JMP jump_table[REG*4]`. `JMP jump_table[REG*8]` en x64.

16.4 on page 62.

12.3

```
#include <stdio.h>

void f(int a)
{
        switch (a)
        {
        case 1:
        case 2:
        case 7:
        case 10:
                printf ("1, 2, 7, 10\n");
                break;
        case 3:
        case 4:
        case 5:
        case 6:
                printf ("3, 4, 5\n");
                break;
        case 8:
        case 9:
        case 20:
        case 21:
                printf ("8, 9, 21\n");
                break;
        case 22:
                printf ("22\n");
                break;
        default:
                printf ("default\n");
                break;
        };
};

int main()
{
        f(4);
};
```

12.3.1 MSVC

Listing 12.6: Con optimización MSVC 2010

```
1
2  $SG2798 DB      '1, 2, 7, 10', 0aH, 00H
3  $SG2800 DB      '3, 4, 5', 0aH, 00H
4  $SG2802 DB      '8, 9, 21', 0aH, 00H
5  $SG2804 DB      '22', 0aH, 00H
6  $SG2806 DB      'default', 0aH, 00H
7
8  _a$ = 8
9  _f      PROC
10         mov     eax, DWORD PTR _a$[esp-4]
11         dec     eax
12         cmp     eax, 21
13         ja      SHORT $LN1@f
14         movzx   eax, BYTE PTR $LN10@f[eax]
15         jmp     DWORD PTR $LN11@f[eax*4]
16 $LN5@f:
17         mov     DWORD PTR _a$[esp-4], OFFSET $SG2798 ; '1, 2, 7, 10'
18         jmp     DWORD PTR __imp__printf
19 $LN4@f:
20         mov     DWORD PTR _a$[esp-4], OFFSET $SG2800 ; '3, 4, 5'
21         jmp     DWORD PTR __imp__printf
22 $LN3@f:
23         mov     DWORD PTR _a$[esp-4], OFFSET $SG2802 ; '8, 9, 21'
24         jmp     DWORD PTR __imp__printf
25 $LN2@f:
```

```
26          mov     DWORD PTR _a$[esp-4], OFFSET $SG2804 ; '22'
27          jmp     DWORD PTR __imp__printf
28  $LN1@f:
29          mov     DWORD PTR _a$[esp-4], OFFSET $SG2806 ; 'default'
30          jmp     DWORD PTR __imp__printf
31          npad    2 ;  $LN11@f
32  $LN11@f:
33          DD      $LN5@f ;  '1, 2, 7, 10'
34          DD      $LN4@f ;  '3, 4, 5'
35          DD      $LN3@f ;  '8, 9, 21'
36          DD      $LN2@f ;  '22'
37          DD      $LN1@f ;  'default'
38  $LN10@f:
39          DB      0 ; a=1
40          DB      0 ; a=2
41          DB      1 ; a=3
42          DB      1 ; a=4
43          DB      1 ; a=5
44          DB      1 ; a=6
45          DB      0 ; a=7
46          DB      2 ; a=8
47          DB      2 ; a=9
48          DB      0 ; a=10
49          DB      4 ; a=11
50          DB      4 ; a=12
51          DB      4 ; a=13
52          DB      4 ; a=14
53          DB      4 ; a=15
54          DB      4 ; a=16
55          DB      4 ; a=17
56          DB      4 ; a=18
57          DB      4 ; a=19
58          DB      2 ; a=20
59          DB      2 ; a=21
60          DB      3 ; a=22
61  _f      ENDP
```

: ($LN10@f) , ($LN11@f) .

(línea 13).

: 0 (1, 2, 7, 10), 1 (3, 4, 5), 2 (8, 9, 21), 3 (22), 4 .

(línea 14).

.

? (12.2.1 on page 42), ? .

12.4 Fall-through

switch() .:

```
1   #define R 1
2   #define W 2
3   #define RW 3
4
5   void f(int type)
6   {
7           int read=0, write=0;
8
9           switch (type)
10          {
11          case RW:
12                  read=1;
13          case W:
14                  write=1;
15                  break;
16          case R:
17                  read=1;
```

```
18                 break;
19         default:
20                 break;
21         };
22         printf ("read=%d, write=%d\n", read, write);
23 };
```

$type = 1$ (R), $read$ 1, $type = 2$ (W), $write$ 2. $type = 3$ (RW), $read$ y $write$ 1.

$type = RW$ $type = W$. .

12.4.1 MSVC x86

Listing 12.7: MSVC 2012

```
$SG1305 DB      'read=%d, write=%d', 0aH, 00H

_write$ = -12   ; size = 4
_read$ = -8     ; size = 4
tv64 = -4       ; size = 4
_type$ = 8      ; size = 4
_f      PROC
        push    ebp
        mov     ebp, esp
        sub     esp, 12
        mov     DWORD PTR _read$[ebp], 0
        mov     DWORD PTR _write$[ebp], 0
        mov     eax, DWORD PTR _type$[ebp]
        mov     DWORD PTR tv64[ebp], eax
        cmp     DWORD PTR tv64[ebp], 1 ; R
        je      SHORT $LN2@f
        cmp     DWORD PTR tv64[ebp], 2 ; W
        je      SHORT $LN3@f
        cmp     DWORD PTR tv64[ebp], 3 ; RW
        je      SHORT $LN4@f
        jmp     SHORT $LN5@f
$LN4@f: ; case RW:
        mov     DWORD PTR _read$[ebp], 1
$LN3@f: ; case W:
        mov     DWORD PTR _write$[ebp], 1
        jmp     SHORT $LN5@f
$LN2@f: ; case R:
        mov     DWORD PTR _read$[ebp], 1
$LN5@f: ; default
        mov     ecx, DWORD PTR _write$[ebp]
        push    ecx
        mov     edx, DWORD PTR _read$[ebp]
        push    edx
        push    OFFSET $SG1305 ; 'read=%d, write=%d'
        call    _printf
        add     esp, 12
        mov     esp, ebp
        pop     ebp
        ret     0
_f      ENDP
```

$LN4@f y $LN3@f: $LN4@f, $read$ $write$. $type = W$, $LN3@f, .

Capítulo 13

Bucles

13.1

13.1.1 x86

for().

```
{
    ;
}
```

:

```
#include <stdio.h>

void printing_function(int i)
{
        printf ("f(%d)\n", i);
};

int main()
{
        int i;

        for (i=2; i<10; i++)
                printing_function(i);

        return 0;
};
```

(MSVC 2010):

Listing 13.1: MSVC 2010

```
_i$ = -4
_main    PROC
    push   ebp
    mov    ebp, esp
    push   ecx
    mov    DWORD PTR _i$[ebp], 2   ;
    jmp    SHORT $LN3@main
$LN2@main:
    mov    eax, DWORD PTR _i$[ebp] ; :
    add    eax, 1                  ;
    mov    DWORD PTR _i$[ebp], eax
$LN3@main:
    cmp    DWORD PTR _i$[ebp], 10  ;
    jge    SHORT $LN1@main         ;
    mov    ecx, DWORD PTR _i$[ebp] ;  printing_function(i)
    push   ecx
```

```
    call   _printing_function
    add    esp, 4
    jmp    SHORT $LN2@main         ;
$LN1@main:                         ;
    xor    eax, eax
    mov    esp, ebp
    pop    ebp
    ret    0
_main    ENDP
```

Listing 13.2: Con optimización MSVC

```
_main    PROC
    push   esi
    mov    esi, 2
$LL3@main:
    push   esi
    call   _printing_function
    inc    esi
    add    esp, 4
    cmp    esi, 10     ; 0000000aH
    jl     SHORT $LL3@main
    xor    eax, eax
    pop    esi
    ret    0
_main    ENDP
```

13.1.2

: i

```
for (i=0; i<total_entries_to_process; i++)
    ;
```

total_entries_to_process 0, .

13.2

```
#include <stdio.h>

void my_memcpy (unsigned char* dst, unsigned char* src, size_t cnt)
{
        size_t i;
        for (i=0; i<cnt; i++)
                dst[i]=src[i];
};
```

13.2.1

Listing 13.3: GCC 4.9 x64 (-Os)

```
my_memcpy:
; RDI =
; RSI =
; RDX =

;
        xor     eax, eax
.L2:
;
        cmp     rax, rdx
```

```
        je      .L5
;  RSI+i:
        mov     cl, BYTE PTR [rsi+rax]
;  RDI+i:
        mov     BYTE PTR [rdi+rax], cl
        inc     rax ; i++
        jmp     .L2
.L5:
        ret
```

13.3 Conclusión

:

Listing 13.4: x86

```
    mov [counter], 2 ;
    jmp check
body:
    ;
    ;
    ;
    add [counter], 1 ;
check:
    cmp [counter], 9
    jle body
```

Listing 13.5: x86

```
    MOV [counter], 2 ;
    JMP check
body:
    ;
    ;
    ;
    MOV REG, [counter] ;
    INC REG
    MOV [counter], REG
check:
    CMP [counter], 9
    JLE body
```

Listing 13.6: x86

```
    MOV EBX, 2 ;
    JMP check
body:
    ;
    ;
    ;
    INC EBX ;
check:
    CMP EBX, 9
    JLE body
```

Listing 13.7: x86

```
    MOV [counter], 2 ;
    JMP label_check
label_increment:
    ADD [counter], 1 ;
label_check:
    CMP [counter], 10
    JGE exit
```

```
    ;
    ;
    ;
    JMP label_increment
exit:
```

Listing 13.8: x86

```
    MOV REG, 2 ;
body:
    ;
    ;
    ;
    INC REG ;
    CMP REG, 10
    JL body
```

Listing 13.9: x86

```
    ;
    MOV ECX, 10
body:
    ;
    ;
    ;
    LOOP body
```

Capítulo 14

Procesamiento simple de cadenas en C

14.1 strlen()

```
int my_strlen (const char * str)
{
        const char *eos = str;

        while( *eos++ ) ;

        return( eos - str - 1 );
}

int main()
{
        // test
        return my_strlen("hello!");
};
```

14.1.1 x86

Sin optimización MSVC

```
_eos$ = -4                      ; size = 4
_str$ = 8                       ; size = 4
_strlen PROC
    push    ebp
    mov     ebp, esp
    push    ecx
    mov     eax, DWORD PTR _str$[ebp]  ;  "str"
    mov     DWORD PTR _eos$[ebp], eax  ;  "eos"
$LN2@strlen_:
    mov     ecx, DWORD PTR _eos$[ebp]  ; ECX=eos

    ;

    movsx   edx, BYTE PTR [ecx]
    mov     eax, DWORD PTR _eos$[ebp]  ; EAX=eos
    add     eax, 1                     ;  EAX
    mov     DWORD PTR _eos$[ebp], eax  ;  "eos"
    test    edx, edx                   ; EDX ?
    je      SHORT $LN1@strlen_         ;
    jmp     SHORT $LN2@strlen_         ;
$LN1@strlen_:

    ;

    mov    eax, DWORD PTR _eos$[ebp]
    sub    eax, DWORD PTR _str$[ebp]
    sub    eax, 1                      ;
```

```
    mov    esp, ebp
    pop    ebp
    ret    0
_strlen_ ENDP
```

«*Representación de números con signo*» (22 on page 96).

Con optimización MSVC

:

Listing 14.1: Con optimización MSVC 2012 /Ob0

```
_str$ = 8                       ; size = 4
_strlen PROC
        mov     edx, DWORD PTR _str$[esp-4] ; EDX ->
        mov     eax, edx                    ;  EAX
$LL2@strlen:
        mov     cl, BYTE PTR [eax]          ; CL = *EAX
        inc     eax                         ; EAX++
        test    cl, cl                      ; CL==0?
        jne     SHORT $LL2@strlen           ;
        sub     eax, edx                    ;
        dec     eax                         ;  EAX
        ret     0
_strlen ENDP
```

INC/DEC—

Capítulo 15

Substituición de instrucciones aritméticas por otras

15.1

15.1.1

:

Listing 15.1: Con optimización MSVC 2010

```
unsigned int f(unsigned int a)
{
        return a*8;
};
```

```
_TEXT   SEGMENT
_a$ = 8                                                 ; size = 4
_f      PROC
; File c:\polygon\c\2.c
        mov     eax, DWORD PTR _a$[esp-4]
        add     eax, eax
        add     eax, eax
        add     eax, eax
        ret     0
_f      ENDP
_TEXT   ENDS
END
```

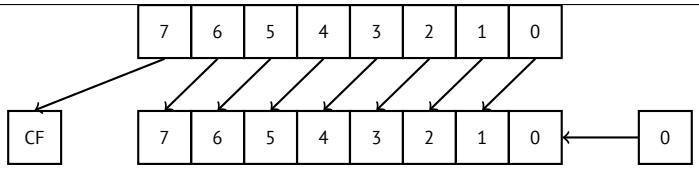
15.1.2

```
unsigned int f(unsigned int a)
{
        return a*4;
};
```

Listing 15.2: Sin optimización MSVC 2010

```
_a$ = 8         ; size = 4
_f      PROC
        push    ebp
        mov     ebp, esp
        mov     eax, DWORD PTR _a$[ebp]
        shl     eax, 2
        pop     ebp
        ret     0
_f      ENDP
```

:



.

15.1.3

.

32-

```
#include <stdint.h>

int f1(int a)
{
        return a*7;
};

int f2(int a)
{
        return a*28;
};

int f3(int a)
{
        return a*17;
};
```

x86

Listing 15.3: Con optimización MSVC 2012

```
; a*7
_a$ = 8
_f1     PROC
        mov     ecx, DWORD PTR _a$[esp-4]
; ECX=a
        lea     eax, DWORD PTR [ecx*8]
; EAX=ECX*8
        sub     eax, ecx
; EAX=EAX-ECX=ECX*8-ECX=ECX*7=a*7
        ret     0
_f1     ENDP

; a*28
_a$ = 8
_f2     PROC
        mov     ecx, DWORD PTR _a$[esp-4]
; ECX=a
        lea     eax, DWORD PTR [ecx*8]
; EAX=ECX*8
        sub     eax, ecx
; EAX=EAX-ECX=ECX*8-ECX=ECX*7=a*7
        shl     eax, 2
; EAX=EAX<<2=(a*7)*4=a*28
        ret     0
_f2     ENDP

; a*17
_a$ = 8
_f3     PROC
        mov     eax, DWORD PTR _a$[esp-4]
```

```
; EAX=a
        shl     eax, 4
; EAX=EAX<<4=EAX*16=a*16
        add     eax, DWORD PTR _a$[esp-4]
; EAX=EAX+a=a*16+a=a*17
        ret     0
_f3     ENDP
```

64-

```
#include <stdint.h>

int64_t f1(int64_t a)
{
        return a*7;
};

int64_t f2(int64_t a)
{
        return a*28;
};

int64_t f3(int64_t a)
{
        return a*17;
};
```

x64

Listing 15.4: Con optimización MSVC 2012

```
; a*7
f1:
        lea     rax, [0+rdi*8]
; RAX=RDI*8=a*8
        sub     rax, rdi
; RAX=RAX-RDI=a*8-a=a*7
        ret

; a*28
f2:
        lea     rax, [0+rdi*4]
; RAX=RDI*4=a*4
        sal     rdi, 5
; RDI=RDI<<5=RDI*32=a*32
        sub     rdi, rax
; RDI=RDI-RAX=a*32-a*4=a*28
        mov     rax, rdi
        ret

; a*17
f3:
        mov     rax, rdi
        sal     rax, 4
; RAX=RAX<<4=a*16
        add     rax, rdi
; RAX=a*16+a=a*17
        ret
```

15.2

15.2.1

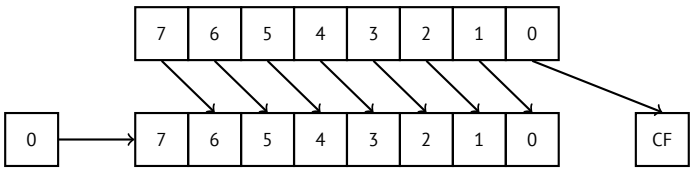
:

```
unsigned int f(unsigned int a)
{
        return a/4;
};
```

(MSVC 2010):

Listing 15.5: MSVC 2010

```
_a$ = 8                                                 ; size = 4
_f      PROC
        mov     eax, DWORD PTR _a$[esp-4]
        shr     eax, 2
        ret     0
_f      ENDP
```



Capítulo 16

Matriz

16.1

```
#include <stdio.h>

int main()
{
    int a[20];
    int i;

    for (i=0; i<20; i++)
        a[i]=i*2;

    for (i=0; i<20; i++)
        printf ("a[%d]=%d\n", i, a[i]);

    return 0;
};
```

16.1.1 x86

MSVC

:

Listing 16.1: MSVC 2008

```
_TEXT    SEGMENT
_i$ = -84                       ; size = 4
_a$ = -80                       ; size = 80
_main      PROC
    push   ebp
    mov    ebp, esp
    sub    esp, 84        ; 00000054H
    mov    DWORD PTR _i$[ebp], 0
    jmp    SHORT $LN6@main
$LN5@main:
    mov    eax, DWORD PTR _i$[ebp]
    add    eax, 1
    mov    DWORD PTR _i$[ebp], eax
$LN6@main:
    cmp    DWORD PTR _i$[ebp], 20   ; 00000014H
    jge    SHORT $LN4@main
    mov    ecx, DWORD PTR _i$[ebp]
    shl    ecx, 1
    mov    edx, DWORD PTR _i$[ebp]
    mov    DWORD PTR _a$[ebp+edx*4], ecx
    jmp    SHORT $LN5@main
$LN4@main:
    mov    DWORD PTR _i$[ebp], 0
    jmp    SHORT $LN3@main
```

```
$LN2@main:
    mov    eax, DWORD PTR _i$[ebp]
    add    eax, 1
    mov    DWORD PTR _i$[ebp], eax
$LN3@main:
    cmp    DWORD PTR _i$[ebp], 20    ; 00000014H
    jge    SHORT $LN1@main
    mov    ecx, DWORD PTR _i$[ebp]
    mov    edx, DWORD PTR _a$[ebp+ecx*4]
    push   edx
    mov    eax, DWORD PTR _i$[ebp]
    push   eax
    push   OFFSET $SG2463
    call   _printf
    add    esp, 12        ; 0000000cH
    jmp    SHORT $LN2@main
$LN1@main:
    xor    eax, eax
    mov    esp, ebp
    pop    ebp
    ret    0
_main      ENDP
```

16.2

16.2.1

```
#include <stdio.h>

int main()
{
        int a[20];
        int i;

        for (i=0; i<20; i++)
                a[i]=i*2;

        printf ("a[20]=%d\n", a[20]);

        return 0;
};
```

(MSVC 2008):

Listing 16.2: Sin optimización MSVC 2008

```
$SG2474 DB    'a[20]=%d', 0aH, 00H

_i$ = -84 ; size = 4
_a$ = -80 ; size = 80
_main    PROC
    push   ebp
    mov    ebp, esp
    sub    esp, 84
    mov    DWORD PTR _i$[ebp], 0
    jmp    SHORT $LN3@main
$LN2@main:
    mov    eax, DWORD PTR _i$[ebp]
    add    eax, 1
    mov    DWORD PTR _i$[ebp], eax
$LN3@main:
    cmp    DWORD PTR _i$[ebp], 20
    jge    SHORT $LN1@main
    mov    ecx, DWORD PTR _i$[ebp]
    shl    ecx, 1
    mov    edx, DWORD PTR _i$[ebp]
    mov    DWORD PTR _a$[ebp+edx*4], ecx
```

```
    jmp     SHORT $LN2@main
$LN1@main:
    mov     eax, DWORD PTR _a$[ebp+80]
    push    eax
    push    OFFSET $SG2474 ; 'a[20]=%d'
    call    DWORD PTR __imp__printf
    add     esp, 8
    xor     eax, eax
    mov     esp, ebp
    pop     ebp
    ret     0
_main    ENDP
_TEXT    ENDS
END
```

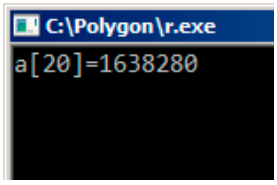


Figura 16.1: OllyDbg:

16.2.2

```
#include <stdio.h>

int main()
{
        int a[20];
        int i;

        for (i=0; i<30; i++)
                a[i]=i;

        return 0;
};
```

MSVC

Listing 16.3: Sin optimización MSVC 2008

```
_TEXT    SEGMENT
_i$ = -84 ; size = 4
_a$ = -80 ; size = 80
_main   PROC
 push   ebp
 mov    ebp, esp
 sub    esp, 84
 mov    DWORD PTR _i$[ebp], 0
 jmp    SHORT $LN3@main
$LN2@main:
 mov    eax, DWORD PTR _i$[ebp]
 add    eax, 1
 mov    DWORD PTR _i$[ebp], eax
$LN3@main:
 cmp    DWORD PTR _i$[ebp], 30 ; 0000001eH
 jge    SHORT $LN1@main
 mov    ecx, DWORD PTR _i$[ebp]
 mov    edx, DWORD PTR _i$[ebp]      ;
 mov    DWORD PTR _a$[ebp+ecx*4], edx ;
 jmp    SHORT $LN2@main
$LN1@main:
```

```
 xor    eax, eax
 mov    esp, ebp
 pop    ebp
 ret    0
_main   ENDP
```

OllyDbg,

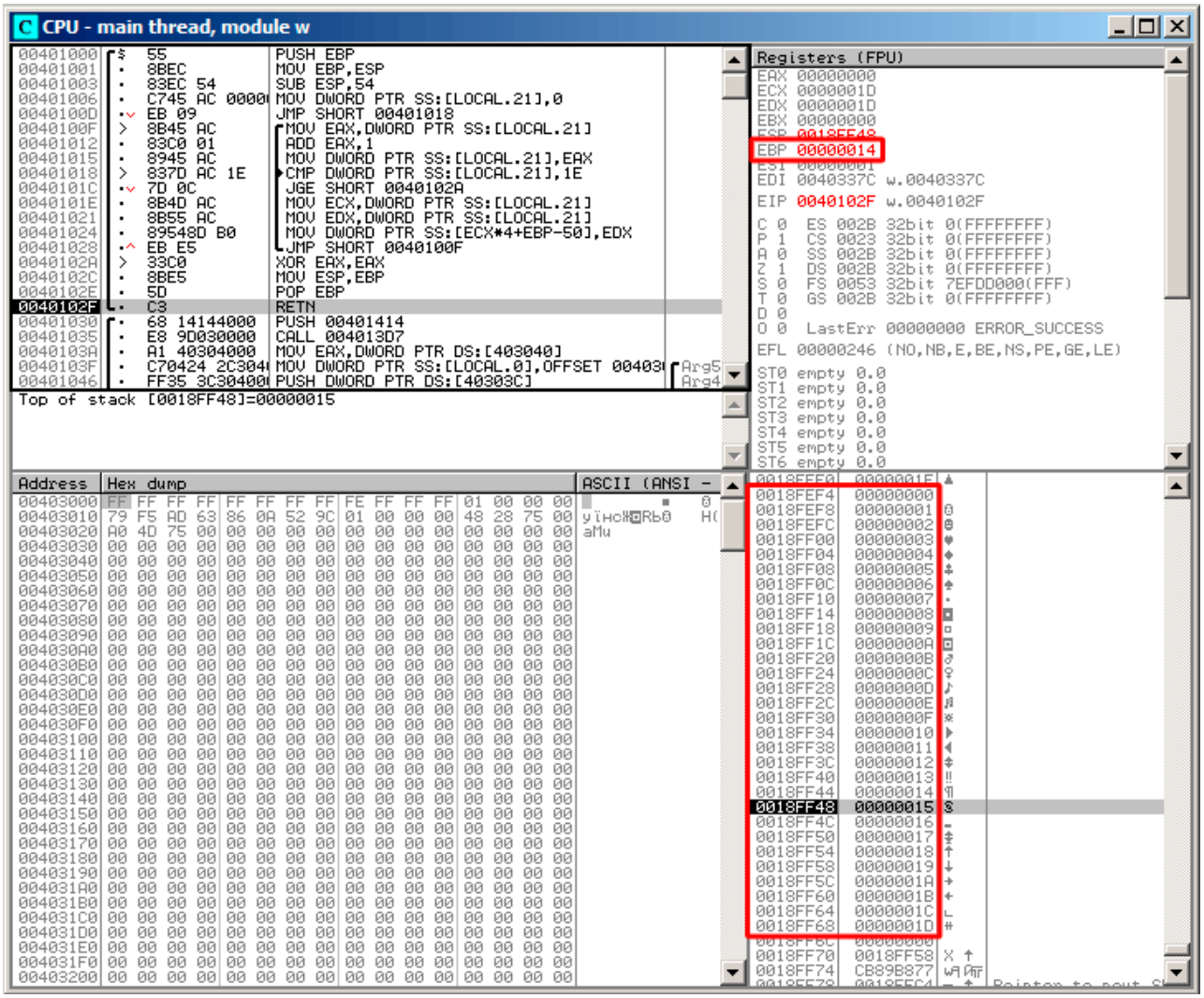


Figura 16.2: OllyDbg:

:

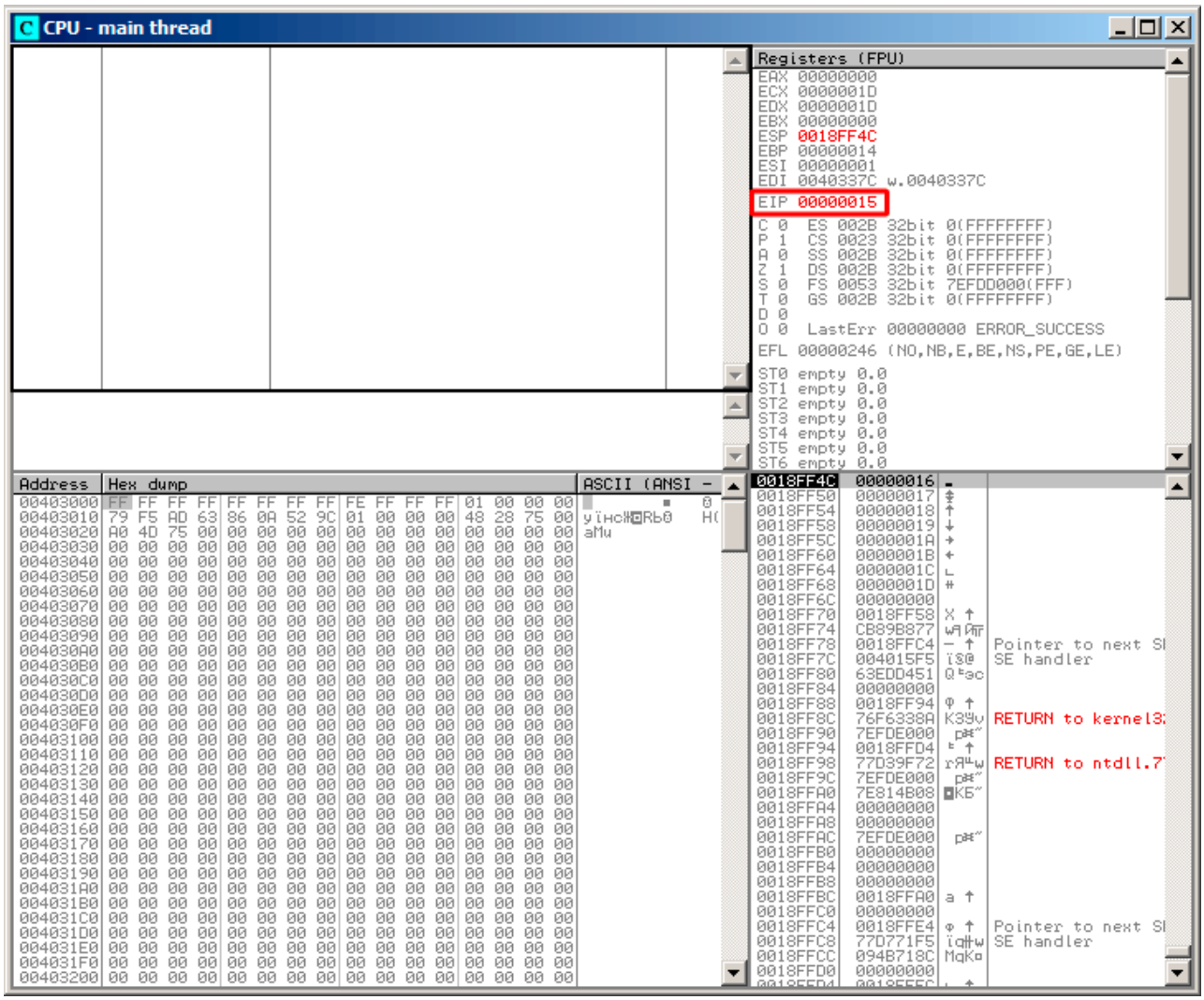


Figura 16.3: OllyDbg:

`main()`:

ESP	
ESP+4	
ESP+84	
ESP+88	

buffer overflow[1].

16.3

```
void f(int size)
{
    int a[size];
...
};
```

16.4

Listing 16.4:

[1] wikipedia

```
#include <stdio.h>

const char* month1[]=
{
        "January",
        "February",
        "March",
        "April",
        "May",
        "June",
        "July",
        "August",
        "September",
        "October",
        "November",
        "December"
};

//
const char* get_month1 (int month)
{
        return month1[month];
};
```

16.4.1 x64

Listing 16.5: Con optimización MSVC 2013 x64

```
_DATA   SEGMENT
month1  DQ      FLAT:$SG3122
        DQ      FLAT:$SG3123
        DQ      FLAT:$SG3124
        DQ      FLAT:$SG3125
        DQ      FLAT:$SG3126
        DQ      FLAT:$SG3127
        DQ      FLAT:$SG3128
        DQ      FLAT:$SG3129
        DQ      FLAT:$SG3130
        DQ      FLAT:$SG3131
        DQ      FLAT:$SG3132
        DQ      FLAT:$SG3133
$SG3122 DB      'January', 00H
$SG3123 DB      'February', 00H
$SG3124 DB      'March', 00H
$SG3125 DB      'April', 00H
$SG3126 DB      'May', 00H
$SG3127 DB      'June', 00H
$SG3128 DB      'July', 00H
$SG3129 DB      'August', 00H
$SG3130 DB      'September', 00H
$SG3156 DB      '%s', 0aH, 00H
$SG3131 DB      'October', 00H
$SG3132 DB      'November', 00H
$SG3133 DB      'December', 00H
_DATA   ENDS

month$ = 8
get_month1 PROC
        movsxd  rax, ecx
        lea     rcx, OFFSET FLAT:month1
        mov     rax, QWORD PTR [rcx+rax*8]
        ret     0
get_month1 ENDP
```

:

- [2].
-
- .

Con optimización GCC 4.9 [3]:

Listing 16.6: Con optimización GCC 4.9 x64

```
        movsx   rdi, edi
        mov     rax, QWORD PTR month1[0+rdi*8]
        ret
```

32-bit MSVC

Listing 16.7: Con optimización MSVC 2013 x86

```
_month$ = 8
_get_month1 PROC
        mov     eax, DWORD PTR _month$[esp-4]
        mov     eax, DWORD PTR _month1[eax*4]
        ret     0
_get_month1 ENDP
```

16.5

0	[0][0]
1	[0][1]
2	[0][2]
3	[0][3]
4	[1][0]
5	[1][1]
6	[1][2]
7	[1][3]
8	[2][0]
9	[2][1]
10	[2][2]
11	[2][3]

Cuadro 16.1:

0	1	2	3
4	5	6	7
8	9	10	11

Cuadro 16.2:

row-major order, C/C++ y Python. *row-major order*

column-major order () FORTRAN, MATLAB y R. *column-major order*

?

16.5.1

0..3:

2

3

Listing 16.8:

```
#include <stdio.h>

char a[3][4];

int main()
{
        int x, y;

        //
        for (x=0; x<3; x++)
                for (y=0; y<4; y++)
                        a[x][y]=0;

        //  0..3:
        for (y=0; y<4; y++)
                a[1][y]=y;
};
```

. 0, 1, 2 y 3:

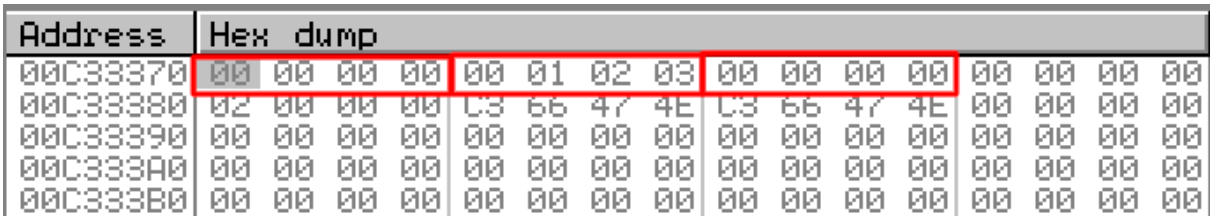


Figura 16.4: OllyDbg:

0..2:

Listing 16.9:

```
#include <stdio.h>

char a[3][4];

int main()
{
        int x, y;

        //
        for (x=0; x<3; x++)
                for (y=0; y<4; y++)
                        a[x][y]=0;

        //  0..2:
        for (x=0; x<3; x++)
                a[x][2]=x;
};
```

. 0, 1 y 2.

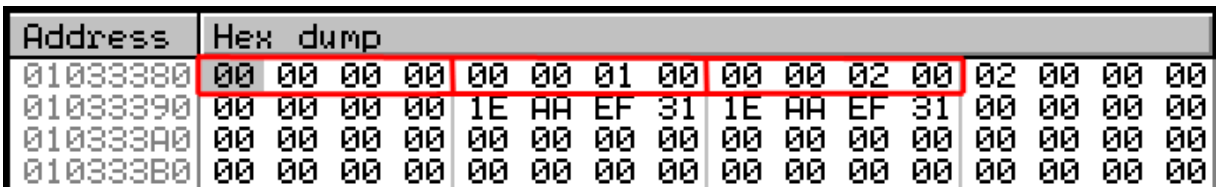


Figura 16.5: OllyDbg:

16.5.2

```
#include <stdio.h>

char a[3][4];

char get_by_coordinates1 (char array[3][4], int a, int b)
{
        return array[a][b];
};

char get_by_coordinates2 (char *array, int a, int b)
{
        //
        // 4
        return array[a*4+b];
};

char get_by_coordinates3 (char *array, int a, int b)
{
        // ,
        //
        // 4
        return *(array+a*4+b);
};

int main()
{
        a[2][3]=123;
        printf ("%d\n", get_by_coordinates1(a, 2, 3));
        printf ("%d\n", get_by_coordinates2(a, 2, 3));
        printf ("%d\n", get_by_coordinates3(a, 2, 3));
};
```

Listing 16.10: Con optimización MSVC 2013 x64

```
array$ = 8
a$ = 16
b$ = 24
get_by_coordinates3 PROC
; RCX=
; RDX=a
; R8=b
        movsxd  rax, r8d
; EAX=b
        movsxd  r9, edx
; R9=a
        add     rax, rcx
; RAX=b+
        movzx   eax, BYTE PTR [rax+r9*4]
; AL= RAX+R9*4=b++a*4=+a*4+b
        ret     0
get_by_coordinates3 ENDP

array$ = 8
a$ = 16
b$ = 24
get_by_coordinates2 PROC
        movsxd  rax, r8d
        movsxd  r9, edx
        add     rax, rcx
        movzx   eax, BYTE PTR [rax+r9*4]
        ret     0
get_by_coordinates2 ENDP

array$ = 8
a$ = 16
b$ = 24
```

```
get_by_coordinates1 PROC
        movsxd  rax, r8d
        movsxd  r9, edx
        add     rax, rcx
        movzx   eax, BYTE PTR [rax+r9*4]
        ret     0
get_by_coordinates1 ENDP
```

16.5.3

:

Listing 16.11:

```
#include <stdio.h>

int a[10][20][30];

void insert(int x, int y, int z, int value)
{
        a[x][y][z]=value;
};
```

x86

(MSVC 2010):

Listing 16.12: MSVC 2010

```
_DATA    SEGMENT
COMM     _a:DWORD:01770H
_DATA    ENDS
PUBLIC   _insert
_TEXT    SEGMENT
_x$ = 8                 ; size = 4
_y$ = 12                ; size = 4
_z$ = 16                ; size = 4
_value$ = 20            ; size = 4
_insert    PROC
    push   ebp
    mov    ebp, esp
    mov    eax, DWORD PTR _x$[ebp]
    imul   eax, 2400                    ; eax=600*4*x
    mov    ecx, DWORD PTR _y$[ebp]
    imul   ecx, 120                     ; ecx=30*4*y
    lea    edx, DWORD PTR _a[eax+ecx]   ; edx=a + 600*4*x + 30*4*y
    mov    eax, DWORD PTR _z$[ebp]
    mov    ecx, DWORD PTR _value$[ebp]
    mov    DWORD PTR [edx+eax*4], ecx   ; *(edx+z*4)=
    pop    ebp
    ret    0
_insert    ENDP
_TEXT      ENDS
```

Listing 16.13: GCC 4.4.1

```
                public insert
insert          proc near

x               = dword ptr  8
y               = dword ptr  0Ch
z               = dword ptr  10h
value           = dword ptr  14h

                push    ebp
```

```
                mov     ebp, esp
                push    ebx
                mov     ebx, [ebp+x]
                mov     eax, [ebp+y]
                mov     ecx, [ebp+z]
                lea     edx, [eax+eax]              ; edx=y*2
                mov     eax, edx                    ; eax=y*2
                shl     eax, 4                      ; eax=(y*2)<<4 = y*2*16 = y*32
                sub     eax, edx                    ; eax=y*32 - y*2=y*30
                imul    edx, ebx, 600               ; edx=x*600
                add     eax, edx                    ; eax=eax+edx=y*30 + x*600
                lea     edx, [eax+ecx]              ; edx=y*30 + x*600 + z
                mov     eax, [ebp+value]
                mov     dword ptr ds:a[edx*4], eax  ; *(a+edx*4)=
                pop     ebx
                pop     ebp
                retn
insert          endp
```

. $:(y+y) \ll 4-(y+y)=(2y) \ll 4-2y=2\cdot 16\cdot y-2y=32y-2y=30y$. .

16.6 Conclusión

Capítulo 17

Manipulando bit(s) específicos

17.1

17.1.1 x86

```
        HANDLE fh;

        fh=CreateFile ("file", GENERIC_WRITE | GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_ALWAYS↙
    ↳ , FILE_ATTRIBUTE_NORMAL, NULL);
```

(MSVC 2010):

Listing 17.1: MSVC 2010

```
        push    0
        push    128                                     ; 00000080H
        push    4
        push    0
        push    1
        push    -1073741824                             ; c0000000H
        push    OFFSET $SG78813
        call    DWORD PTR __imp__CreateFileA@28
        mov     DWORD PTR _fh$[ebp], eax
```

WinNT.h:

Listing 17.2: WinNT.h

```
#define GENERIC_READ                     (0x80000000L)
#define GENERIC_WRITE                    (0x40000000L)
#define GENERIC_EXECUTE                  (0x20000000L)
#define GENERIC_ALL                      (0x10000000L)
```

, `GENERIC_READ | GENERIC_WRITE = 0x80000000 | 0x40000000 = 0xC0000000`, `CreateFile()`[1].

Listing 17.3: KERNEL32.DLL (Windows XP SP3 x86)

```
.text:7C83D429                test    byte ptr [ebp+dwDesiredAccess+3], 40h
.text:7C83D42D                mov     [ebp+var_8], 1
.text:7C83D434                jz      short loc_7C83D417
.text:7C83D436                jmp     loc_7C810817
```

(7.3.1 on page 19)).

```
if ((dwDesiredAccess&0x40000000) == 0) goto loc_7C83D417
```

[1] msdn.microsoft.com/en-us/library/aa363858(VS.85).aspx

17.2

:

```
#include <stdio.h>

#define IS_SET(flag, bit)       ((flag) & (bit))
#define SET_BIT(var, bit)       ((var) |= (bit))
#define REMOVE_BIT(var, bit)    ((var) &= ~(bit))

int f(int a)
{
    int rt=a;

    SET_BIT (rt, 0x4000);
    REMOVE_BIT (rt, 0x200);

    return rt;
};

int main()
{
    f(0x12340678);
};
```

17.2.1 x86

Sin optimización MSVC

(MSVC 2010):

Listing 17.4: MSVC 2010

```
_rt$ = -4         ; size = 4
_a$ = 8           ; size = 4
_f  PROC
    push   ebp
    mov    ebp, esp
    push   ecx
    mov    eax, DWORD PTR _a$[ebp]
    mov    DWORD PTR _rt$[ebp], eax
    mov    ecx, DWORD PTR _rt$[ebp]
    or     ecx, 16384                ; 00004000H
    mov    DWORD PTR _rt$[ebp], ecx
    mov    edx, DWORD PTR _rt$[ebp]
    and    edx, -513                 ; fffffdffH
    mov    DWORD PTR _rt$[ebp], edx
    mov    eax, DWORD PTR _rt$[ebp]
    mov    esp, ebp
    pop    ebp
    ret    0
_f  ENDP
```

Con optimización MSVC

Listing 17.5: Con optimización MSVC

```
_a$ = 8                       ; size = 4
_f    PROC
    mov    eax, DWORD PTR _a$[esp-4]
    and    eax, -513          ; fffffdffH
    or     eax, 16384         ; 00004000H
    ret    0
_f    ENDP
```

CAPÍTULO 17. MANIPULANDO BIT(S) ESPECÍFICOS

17.3 Desplazamientos

SHL (SHift Left) y SHR (SHift Right) .

15.1.2 on page 53, 15.2.1 on page 56.

17.4

«population count»[2].

```
#include <stdio.h>

#define IS_SET(flag, bit)       ((flag) & (bit))

int f(unsigned int a)
{
    int i;
    int rt=0;

    for (i=0; i<32; i++)
        if (IS_SET (a, 1<<i))
            rt++;

    return rt;
};

int main()
{
    f(0x12345678); // test
};
```

$1 \ll i$ $i = 0 \ldots 31$:

[2]

C/C++			
$1 \ll 0$	1	1	1
$1 \ll 1$	2^1	2	2
$1 \ll 2$	2^2	4	4
$1 \ll 3$	2^3	8	8
$1 \ll 4$	2^4	16	0x10
$1 \ll 5$	2^5	32	0x20
$1 \ll 6$	2^6	64	0x40
$1 \ll 7$	2^7	128	0x80
$1 \ll 8$	2^8	256	0x100
$1 \ll 9$	2^9	512	0x200
$1 \ll 10$	2^{10}	1024	0x400
$1 \ll 11$	2^{11}	2048	0x800
$1 \ll 12$	2^{12}	4096	0x1000
$1 \ll 13$	2^{13}	8192	0x2000
$1 \ll 14$	2^{14}	16384	0x4000
$1 \ll 15$	2^{15}	32768	0x8000
$1 \ll 16$	2^{16}	65536	0x10000
$1 \ll 17$	2^{17}	131072	0x20000
$1 \ll 18$	2^{18}	262144	0x40000
$1 \ll 19$	2^{19}	524288	0x80000
$1 \ll 20$	2^{20}	1048576	0x100000
$1 \ll 21$	2^{21}	2097152	0x200000
$1 \ll 22$	2^{22}	4194304	0x400000
$1 \ll 23$	2^{23}	8388608	0x800000
$1 \ll 24$	2^{24}	16777216	0x1000000
$1 \ll 25$	2^{25}	33554432	0x2000000
$1 \ll 26$	2^{26}	67108864	0x4000000
$1 \ll 27$	2^{27}	134217728	0x8000000
$1 \ll 28$	2^{28}	268435456	0x10000000
$1 \ll 29$	2^{29}	536870912	0x20000000
$1 \ll 30$	2^{30}	1073741824	0x40000000
$1 \ll 31$	2^{31}	2147483648	0x80000000

`ssl_private.h` Apache 2.4.6:

```
/**
 * Define the SSL options
 */
#define SSL_OPT_NONE           (0)
#define SSL_OPT_RELSET         (1<<0)
#define SSL_OPT_STDENVVARS     (1<<1)
#define SSL_OPT_EXPORTCERTDATA (1<<3)
#define SSL_OPT_FAKEBASICAUTH  (1<<4)
#define SSL_OPT_STRICTREQUIRE  (1<<5)
#define SSL_OPT_OPTRENEGOTIATE (1<<6)
#define SSL_OPT_LEGACYDNFORMAT (1<<7)
```

.

17.4.1 x86

MSVC

(MSVC 2010):

Listing 17.6: MSVC 2010

```
_rt$ = -8               ; size = 4
_i$ = -4                ; size = 4
_a$ = 8                 ; size = 4
_f  PROC
    push   ebp
    mov    ebp, esp
    sub    esp, 8
```

```
    mov    DWORD PTR _rt$[ebp], 0
    mov    DWORD PTR _i$[ebp], 0
    jmp    SHORT $LN4@f
$LN3@f:
    mov    eax, DWORD PTR _i$[ebp]   ;  i
    add    eax, 1
    mov    DWORD PTR _i$[ebp], eax
$LN4@f:
    cmp    DWORD PTR _i$[ebp], 32    ; 00000020H
    jge    SHORT $LN2@f              ; ?
    mov    edx, 1
    mov    ecx, DWORD PTR _i$[ebp]
    shl    edx, cl                   ; EDX=EDX<<CL
    and    edx, DWORD PTR _a$[ebp]
    je     SHORT $LN1@f              ; ?
                                     ;
    mov    eax, DWORD PTR _rt$[ebp]  ;
    add    eax, 1                    ;  rt
    mov    DWORD PTR _rt$[ebp], eax
$LN1@f:
    jmp    SHORT $LN3@f
$LN2@f:
    mov    eax, DWORD PTR _rt$[ebp]
    mov    esp, ebp
    pop    ebp
    ret    0
_f    ENDP
```

17.4.2 x64

:

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

#define IS_SET(flag, bit)       ((flag) & (bit))

int f(uint64_t a)
{
    uint64_t i;
    int rt=0;

    for (i=0; i<64; i++)
        if (IS_SET (a, 1ULL<<i))
            rt++;

    return rt;
};
```

Con optimización MSVC 2010

Listing 17.7: MSVC 2010

```
a$ = 8
f       PROC
; RCX =
        xor     eax, eax
        mov     edx, 1
        lea     r8d, QWORD PTR [rax+64]
; R8D=64
        npad    5
$LL4@f:
        test    rdx, rcx
; ?
; .
```

```
        je      SHORT $LN3@f
        inc     eax     ; rt++
$LN3@f:
        rol     rdx, 1  ; RDX=RDX<<1
        dec     r8      ; R8--
        jne     SHORT $LL4@f
        fatret  0
f       ENDP
```

ROL SHL, «rotate left» «shift left», SHL.

R8 . .

:

RDX	R8
0x0000000000000001	64
0x0000000000000002	63
0x0000000000000004	62
0x0000000000000008	61
...	...
0x4000000000000000	2
0x8000000000000000	1

Con optimización MSVC 2012

Listing 17.8: MSVC 2012

```
a$ = 8
f       PROC
; RCX =
        xor     eax, eax
        mov     edx, 1
        lea     r8d, QWORD PTR [rax+32]
; EDX = 1, R8D = 32
        npad    5
$LL4@f:
;
        test    rdx, rcx
        je      SHORT $LN3@f
        inc     eax     ; rt++
$LN3@f:
        rol     rdx, 1  ; RDX=RDX<<1
; -----------------------------------------
;
        test    rdx, rcx
        je      SHORT $LN11@f
        inc     eax     ; rt++
$LN11@f:
        rol     rdx, 1  ; RDX=RDX<<1
; -----------------------------------------
        dec     r8      ; R8--
        jne     SHORT $LL4@f
        fatret  0
f       ENDP
```

Con optimización MSVC 2012 MSVC 2010, .

17.5 Conclusión

17.5.1

Listing 17.9: C/C++

```
if (input&0x40)
    ...
```

Listing 17.10: x86

```
TEST REG, 40h
JNZ is_set
;
```

Listing 17.11: x86

```
TEST REG, 40h
JZ is_cleared
;
```

17.5.2

Listing 17.12: C/C++

```
if ((value>>n)&1)
    ....
```

Listing 17.13: x86

```
; REG=input_value
; CL=n
SHR REG, CL
AND REG, 1
```

Listing 17.14: C/C++

```
if (value & (1<<n))
    ....
```

Listing 17.15: x86

```
; CL=n
MOV REG, 1
SHL REG, CL
AND input_value, REG
```

17.5.3

Listing 17.16: C/C++

```
value=value|0x40;
```

Listing 17.17: x86

```
OR REG, 40h
```

17.5.4

Listing 17.18: C/C++

```
value=value|(1<<n);
```

Listing 17.19: x86

```
; CL=n
MOV REG, 1
SHL REG, CL
OR input_value, REG
```

CAPÍTULO 17. MANIPULANDO BIT(S) ESPECÍFICOS

17.5.5

Listing 17.20: C/C++

```
value=value&(~0x40);
```

Listing 17.21: x86

```
AND REG, 0FFFFFFBFh
```

Listing 17.22: x64

```
AND REG, 0FFFFFFFFFFFFFFBFh
```

17.5.6

Listing 17.23: C/C++

```
value=value&(~(1<<n));
```

Listing 17.24: x86

```
; CL=n
MOV REG, 1
SHL REG, CL
NOT REG
AND input_value, REG
```

Capítulo 18

```
#include <stdint.h>

//
#define RNG_a 1664525
#define RNG_c 1013904223

static uint32_t rand_state;

void my_srand (uint32_t init)
{
        rand_state=init;
}

int my_rand ()
{
        rand_state=rand_state*RNG_a;
        rand_state=rand_state+RNG_c;
        return rand_state & 0x7fff;
}
```

. [Pre+07].

18.1 x86

Listing 18.1: Con optimización MSVC 2013

```
_BSS    SEGMENT
_rand_state DD  01H DUP (?)
_BSS    ENDS

_init$ = 8
_srand  PROC
        mov     eax, DWORD PTR _init$[esp-4]
        mov     DWORD PTR _rand_state, eax
        ret     0
_srand  ENDP

_TEXT   SEGMENT
_rand   PROC
        imul    eax, DWORD PTR _rand_state, 1664525
        add     eax, 1013904223         ; 3c6ef35fH
        mov     DWORD PTR _rand_state, eax
        and     eax, 32767              ; 00007fffH
        ret     0
_rand   ENDP

_TEXT   ENDS
```

:

Listing 18.2: Sin optimización MSVC 2013

```
_BSS    SEGMENT
```

```
_rand_state DD  01H DUP (?)
_BSS    ENDS

_init$ = 8
_srand  PROC
        push    ebp
        mov     ebp, esp
        mov     eax, DWORD PTR _init$[ebp]
        mov     DWORD PTR _rand_state, eax
        pop     ebp
        ret     0
_srand  ENDP

_TEXT   SEGMENT
_rand   PROC
        push    ebp
        mov     ebp, esp
        imul    eax, DWORD PTR _rand_state, 1664525
        mov     DWORD PTR _rand_state, eax
        mov     ecx, DWORD PTR _rand_state
        add     ecx, 1013904223         ; 3c6ef35fH
        mov     DWORD PTR _rand_state, ecx
        mov     eax, DWORD PTR _rand_state
        and     eax, 32767              ; 00007fffH
        pop     ebp
        ret     0
_rand   ENDP

_TEXT   ENDS
```

18.2 x64

Listing 18.3: Con optimización MSVC 2013 x64

```
_BSS    SEGMENT
rand_state DD   01H DUP (?)
_BSS    ENDS

init$ = 8
my_srand PROC
; ECX =
        mov     DWORD PTR rand_state, ecx
        ret     0
my_srand ENDP

_TEXT   SEGMENT
my_rand PROC
        imul    eax, DWORD PTR rand_state, 1664525      ; 0019660dH
        add     eax, 1013904223                         ; 3c6ef35fH
        mov     DWORD PTR rand_state, eax
        and     eax, 32767                              ; 00007fffH
        ret     0
my_rand ENDP

_TEXT   ENDS
```

Capítulo 19

Estructuras

19.1 MSVC:

Listing 19.1: WinBase.h

```
typedef struct _SYSTEMTIME {
  WORD wYear;
  WORD wMonth;
  WORD wDayOfWeek;
  WORD wDay;
  WORD wHour;
  WORD wMinute;
  WORD wSecond;
  WORD wMilliseconds;
} SYSTEMTIME, *PSYSTEMTIME;
```

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

void main()
{
    SYSTEMTIME t;
    GetSystemTime (&t);

    printf ("%04d-%02d-%02d %02d:%02d:%02d\n",
        t.wYear, t.wMonth, t.wDay,
        t.wHour, t.wMinute, t.wSecond);

    return;
};
```

(MSVC 2010):

Listing 19.2: MSVC 2010 /GS-

```
_t$ = -16 ; size = 16
_main      PROC
    push   ebp
    mov    ebp, esp
    sub    esp, 16
    lea    eax, DWORD PTR _t$[ebp]
    push   eax
    call   DWORD PTR __imp__GetSystemTime@4
    movzx  ecx, WORD PTR _t$[ebp+12] ; wSecond
    push   ecx
    movzx  edx, WORD PTR _t$[ebp+10] ; wMinute
    push   edx
    movzx  eax, WORD PTR _t$[ebp+8] ; wHour
    push   eax
    movzx  ecx, WORD PTR _t$[ebp+6] ; wDay
    push   ecx
    movzx  edx, WORD PTR _t$[ebp+2] ; wMonth
    push   edx
```

```
    movzx  eax, WORD PTR _t$[ebp] ; wYear
    push   eax
    push   OFFSET $SG78811 ; '%04d-%02d-%02d %02d:%02d:%02d', 0aH, 00H
    call   _printf
    add    esp, 28
    xor    eax, eax
    mov    esp, ebp
    pop    ebp
    ret    0
_main      ENDP
```

19.1.1

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

void main()
{
    WORD array[8];
    GetSystemTime (array);

    printf ("%04d-%02d-%02d %02d:%02d:%02d\n",
        array[0] /* wYear */, array[1] /* wMonth */, array[3] /* wDay */,
        array[4] /* wHour */, array[5] /* wMinute */, array[6] /* wSecond */);

    return;
};
```

```
systemtime2.c(7) : warning C4133: 'function' : incompatible types - from 'WORD [8]' to '↙
    ↳ LPSYSTEMTIME'
```

:

Listing 19.3: Sin optimización MSVC 2010

```
$SG78573 DB     '%04d-%02d-%02d %02d:%02d:%02d', 0aH, 00H

_array$ = -16   ; size = 16
_main   PROC
        push    ebp
        mov     ebp, esp
        sub     esp, 16
        lea     eax, DWORD PTR _array$[ebp]
        push    eax
        call    DWORD PTR __imp__GetSystemTime@4
        movzx   ecx, WORD PTR _array$[ebp+12] ; wSecond
        push    ecx
        movzx   edx, WORD PTR _array$[ebp+10] ; wMinute
        push    edx
        movzx   eax, WORD PTR _array$[ebp+8] ; wHoure
        push    eax
        movzx   ecx, WORD PTR _array$[ebp+6] ; wDay
        push    ecx
        movzx   edx, WORD PTR _array$[ebp+2] ; wMonth
        push    edx
        movzx   eax, WORD PTR _array$[ebp] ; wYear
        push    eax
        push    OFFSET $SG78573
        call    _printf
        add     esp, 28
        xor     eax, eax
        mov     esp, ebp
        pop     ebp
        ret     0
_main   ENDP
```

!

.

.

19.2

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

void main()
{
    SYSTEMTIME *t;

    t=(SYSTEMTIME *)malloc (sizeof (SYSTEMTIME));

    GetSystemTime (t);

    printf ("%04d-%02d-%02d %02d:%02d:%02d\n",
        t->wYear, t->wMonth, t->wDay,
        t->wHour, t->wMinute, t->wSecond);

    free (t);

    return;
};
```

Listing 19.4: Con optimización MSVC

```
_main      PROC
    push   esi
    push   16
    call   _malloc
    add    esp, 4
    mov    esi, eax
    push   esi
    call   DWORD PTR __imp__GetSystemTime@4
    movzx  eax, WORD PTR [esi+12] ; wSecond
    movzx  ecx, WORD PTR [esi+10] ; wMinute
    movzx  edx, WORD PTR [esi+8] ; wHour
    push   eax
    movzx  eax, WORD PTR [esi+6] ; wDay
    push   ecx
    movzx  ecx, WORD PTR [esi+2] ; wMonth
    push   edx
    movzx  edx, WORD PTR [esi] ; wYear
    push   eax
    push   ecx
    push   edx
    push   OFFSET $SG78833
    call   _printf
    push   esi
    call   _free
    add    esp, 32
    xor    eax, eax
    pop    esi
    ret    0
_main      ENDP
```

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

void main()
{
    WORD *t;

    t=(WORD *)malloc (16);

    GetSystemTime (t);
```

```
    printf ("%04d-%02d-%02d %02d:%02d:%02d\n",
        t[0] /* wYear */, t[1] /* wMonth */, t[3] /* wDay */,
        t[4] /* wHour */, t[5] /* wMinute */, t[6] /* wSecond */);

    free (t);

    return;
};
```

:

Listing 19.5: Con optimización MSVC

```
$SG78594 DB     '%04d-%02d-%02d %02d:%02d:%02d', 0aH, 00H

_main   PROC
        push    esi
        push    16
        call    _malloc
        add     esp, 4
        mov     esi, eax
        push    esi
        call    DWORD PTR __imp__GetSystemTime@4
        movzx   eax, WORD PTR [esi+12]
        movzx   ecx, WORD PTR [esi+10]
        movzx   edx, WORD PTR [esi+8]
        push    eax
        movzx   eax, WORD PTR [esi+6]
        push    ecx
        movzx   ecx, WORD PTR [esi+2]
        push    edx
        movzx   edx, WORD PTR [esi]
        push    eax
        push    ecx
        push    edx
        push    OFFSET $SG78594
        call    _printf
        push    esi
        call    _free
        add     esp, 32
        xor     eax, eax
        pop     esi
        ret     0
_main   ENDP
```

19.3 Organización de campos en la estructura

```
#include <stdio.h>

struct s
{
    char a;
    int b;
    char c;
    int d;
};

void f(struct s s)
{
    printf ("a=%d; b=%d; c=%d; d=%d\n", s.a, s.b, s.c, s.d);
};

int main()
{
    struct s tmp;
    tmp.a=1;
```

```
    tmp.b=2;
    tmp.c=3;
    tmp.d=4;
    f(tmp);
};
```

19.3.1 x86

Listing 19.6: MSVC 2012 /GS- /Ob0

```
1
2  _tmp$ = -16
3  _main    PROC
4      push   ebp
5      mov    ebp, esp
6      sub    esp, 16
7      mov    BYTE PTR _tmp$[ebp], 1     ;  a
8      mov    DWORD PTR _tmp$[ebp+4], 2  ;  b
9      mov    BYTE PTR _tmp$[ebp+8], 3   ;  c
10     mov    DWORD PTR _tmp$[ebp+12], 4 ;  d
11     sub    esp, 16                    ;
12     mov    eax, esp
13     mov    ecx, DWORD PTR _tmp$[ebp]  ;
14     mov    DWORD PTR [eax], ecx
15     mov    edx, DWORD PTR _tmp$[ebp+4]
16     mov    DWORD PTR [eax+4], edx
17     mov    ecx, DWORD PTR _tmp$[ebp+8]
18     mov    DWORD PTR [eax+8], ecx
19     mov    edx, DWORD PTR _tmp$[ebp+12]
20     mov    DWORD PTR [eax+12], edx
21     call    _f
22     add    esp, 16
23     xor    eax, eax
24     mov    esp, ebp
25     pop    ebp
26     ret    0
27 _main    ENDP
28
29 _s$ = 8 ; size = 16
30 ?f@@YAXUs@@@Z PROC ; f
31     push   ebp
32     mov    ebp, esp
33     mov    eax, DWORD PTR _s$[ebp+12]
34     push   eax
35     movsx  ecx, BYTE PTR _s$[ebp+8]
36     push   ecx
37     mov    edx, DWORD PTR _s$[ebp+4]
38     push   edx
39     movsx  eax, BYTE PTR _s$[ebp]
40     push   eax
41     push   OFFSET $SG3842
42     call   _printf
43     add    esp, 20
44     pop    ebp
45     ret    0
46 ?f@@YAXUs@@@Z ENDP ; f
47 _TEXT    ENDS
```

(/Zp1) (*/Zp[n] pack structures on n-byte boundary*).

Listing 19.7: MSVC 2012 /GS- /Zp1

```
1
2  _main    PROC
3      push   ebp
4      mov    ebp, esp
5      sub    esp, 12
6      mov    BYTE PTR _tmp$[ebp], 1     ;  a
7      mov    DWORD PTR _tmp$[ebp+1], 2  ;  b
```

```
8     mov    BYTE PTR _tmp$[ebp+5], 3   ;  c
9     mov    DWORD PTR _tmp$[ebp+6], 4  ;  d
10    sub    esp, 12                    ;
11    mov    eax, esp
12    mov    ecx, DWORD PTR _tmp$[ebp]  ;
13    mov    DWORD PTR [eax], ecx
14    mov    edx, DWORD PTR _tmp$[ebp+4]
15    mov    DWORD PTR [eax+4], edx
16    mov    cx, WORD PTR _tmp$[ebp+8]
17    mov    WORD PTR [eax+8], cx
18    call    _f
19    add    esp, 12
20    xor    eax, eax
21    mov    esp, ebp
22    pop    ebp
23    ret    0
24 _main    ENDP
25
26 _TEXT    SEGMENT
27 _s$ = 8 ; size = 10
28 ?f@@YAXUs@@@Z PROC    ; f
29    push   ebp
30    mov    ebp, esp
31    mov    eax, DWORD PTR _s$[ebp+6]
32    push   eax
33    movsx  ecx, BYTE PTR _s$[ebp+5]
34    push   ecx
35    mov    edx, DWORD PTR _s$[ebp+1]
36    push   edx
37    movsx  eax, BYTE PTR _s$[ebp]
38    push   eax
39    push   OFFSET $SG3842
40    call   _printf
41    add    esp, 20
42    pop    ebp
43    ret    0
44 ?f@@YAXUs@@@Z ENDP    ; f
```

Listing 19.8: WinNT.h

```
#include "pshpack1.h"
```

Listing 19.9: WinNT.h

```
#include "pshpack4.h"                    // 4 byte packing is the default
```

Listing 19.10: PshPack1.h

```
#if ! (defined(lint) || defined(RC_INVOKED))
#if ( _MSC_VER >= 800 && !defined(_M_I86)) || defined(_PUSHPOP_SUPPORTED)
#pragma warning(disable:4103)
#if !(defined( MIDL_PASS )) || defined( __midl )
#pragma pack(push,1)
#else
#pragma pack(1)
#endif
#else
#pragma pack(1)
#endif
#endif /* ! (defined(lint) || defined(RC_INVOKED)) */
```

19.3.2

```
void f(char a, int b, char c, int d)
{
    printf ("a=%d; b=%d; c=%d; d=%d\n", a, b, c, d);
};
```

.

19.4

```
#include <stdio.h>

struct inner_struct
{
    int a;
    int b;
};

struct outer_struct
{
    char a;
    int b;
    struct inner_struct c;
    char d;
    int e;
};

void f(struct outer_struct s)
{
    printf ("a=%d; b=%d; c.a=%d; c.b=%d; d=%d; e=%d\n",
        s.a, s.b, s.c.a, s.c.b, s.d, s.e);
};

int main()
{
    struct outer_struct s;
    s.a=1;
    s.b=2;
    s.c.a=100;
    s.c.b=101;
    s.d=3;
    s.e=4;
    f(s);
};
```

...

(MSVC 2010):

Listing 19.11: Con optimización MSVC 2010 /Ob0

```
$SG2802 DB    'a=%d; b=%d; c.a=%d; c.b=%d; d=%d; e=%d', 0aH, 00H

_TEXT    SEGMENT
_s$ = 8
_f    PROC
    mov    eax, DWORD PTR _s$[esp+16]
    movsx  ecx, BYTE PTR _s$[esp+12]
    mov    edx, DWORD PTR _s$[esp+8]
    push   eax
    mov    eax, DWORD PTR _s$[esp+8]
    push   ecx
    mov    ecx, DWORD PTR _s$[esp+8]
    push   edx
    movsx  edx, BYTE PTR _s$[esp+8]
    push   eax
    push   ecx
    push   edx
    push   OFFSET $SG2802 ; 'a=%d; b=%d; c.a=%d; c.b=%d; d=%d; e=%d'
    call   _printf
    add    esp, 28
    ret    0
_f    ENDP

_s$ = -24
_main    PROC
    sub    esp, 24
    push   ebx
```

```
    push   esi
    push   edi
    mov    ecx, 2
    sub    esp, 24
    mov    eax, esp
    mov    BYTE PTR _s$[esp+60], 1
    mov    ebx, DWORD PTR _s$[esp+60]
    mov    DWORD PTR [eax], ebx
    mov    DWORD PTR [eax+4], ecx
    lea    edx, DWORD PTR [ecx+98]
    lea    esi, DWORD PTR [ecx+99]
    lea    edi, DWORD PTR [ecx+2]
    mov    DWORD PTR [eax+8], edx
    mov    BYTE PTR _s$[esp+76], 3
    mov    ecx, DWORD PTR _s$[esp+76]
    mov    DWORD PTR [eax+12], esi
    mov    DWORD PTR [eax+16], ecx
    mov    DWORD PTR [eax+20], edi
    call   _f
    add    esp, 24
    pop    edi
    pop    esi
    xor    eax, eax
    pop    ebx
    add    esp, 24
    ret    0
_main    ENDP
```

19.5

19.5.1

3:0 (4 bits)	Stepping
7:4 (4 bits)	Model
11:8 (4 bits)	Family
13:12 (2 bits)	Processor Type
19:16 (4 bits)	Extended Model
27:20 (8 bits)	Extended Family

```
#include <stdio.h>

#ifdef __GNUC__
static inline void cpuid(int code, int *a, int *b, int *c, int *d) {
  asm volatile("cpuid":"=a"(*a),"=b"(*b),"=c"(*c),"=d"(*d):"a"(code));
}
#endif

#ifdef _MSC_VER
#include <intrin.h>
#endif

struct CPUID_1_EAX
{
    unsigned int stepping:4;
    unsigned int model:4;
    unsigned int family_id:4;
    unsigned int processor_type:2;
    unsigned int reserved1:2;
    unsigned int extended_model_id:4;
    unsigned int extended_family_id:8;
    unsigned int reserved2:4;
};

int main()
{
    struct CPUID_1_EAX *tmp;
    int b[4];
```

```
#ifdef _MSC_VER
    __cpuid(b,1);
#endif

#ifdef __GNUC__
    cpuid (1, &b[0], &b[1], &b[2], &b[3]);
#endif

    tmp=(struct CPUID_1_EAX *)&b[0];

    printf ("stepping=%d\n", tmp->stepping);
    printf ("model=%d\n", tmp->model);
    printf ("family_id=%d\n", tmp->family_id);
    printf ("processor_type=%d\n", tmp->processor_type);
    printf ("extended_model_id=%d\n", tmp->extended_model_id);
    printf ("extended_family_id=%d\n", tmp->extended_family_id);

    return 0;
};
```

MSVC

:

Listing 19.12: Con optimización MSVC 2008

```
_b$ = -16  ; size = 16
_main    PROC
    sub    esp, 16
    push   ebx

    xor    ecx, ecx
    mov    eax, 1
    cpuid
    push   esi
    lea    esi, DWORD PTR _b$[esp+24]
    mov    DWORD PTR [esi], eax
    mov    DWORD PTR [esi+4], ebx
    mov    DWORD PTR [esi+8], ecx
    mov    DWORD PTR [esi+12], edx

    mov    esi, DWORD PTR _b$[esp+24]
    mov    eax, esi
    and    eax, 15
    push   eax
    push   OFFSET $SG15435 ; 'stepping=%d', 0aH, 00H
    call   _printf

    mov    ecx, esi
    shr    ecx, 4
    and    ecx, 15
    push   ecx
    push   OFFSET $SG15436 ; 'model=%d', 0aH, 00H
    call   _printf

    mov    edx, esi
    shr    edx, 8
    and    edx, 15
    push   edx
    push   OFFSET $SG15437 ; 'family_id=%d', 0aH, 00H
    call   _printf

    mov    eax, esi
    shr    eax, 12
    and    eax, 3
    push   eax
    push   OFFSET $SG15438 ; 'processor_type=%d', 0aH, 00H
    call   _printf
```

```
    mov    ecx, esi
    shr    ecx, 16
    and    ecx, 15
    push   ecx
    push   OFFSET $SG15439 ; 'extended_model_id=%d', 0aH, 00H
    call   _printf

    shr    esi, 20
    and    esi, 255
    push   esi
    push   OFFSET $SG15440 ; 'extended_family_id=%d', 0aH, 00H
    call   _printf
    add    esp, 48
    pop    esi

    xor    eax, eax
    pop    ebx

    add    esp, 16
    ret    0
_main    ENDP
```

Capítulo 20

20.1

```
#include <stdint.h>

uint64_t f ()
{
        return 0x1234567890ABCDEF;
};
```

20.1.1 x86

Listing 20.1: Con optimización MSVC 2010

```
_f      PROC
        mov     eax, -1867788817        ; 90abcdefH
        mov     edx, 305419896          ; 12345678H
        ret     0
_f      ENDP
```

20.2

```
#include <stdint.h>

uint64_t f_add (uint64_t a, uint64_t b)
{
        return a+b;
};

void f_add_test ()
{
#ifdef __GNUC__
        printf ("%lld\n", f_add(12345678901234, 23456789012345));
#else
        printf ("%I64d\n", f_add(12345678901234, 23456789012345));
#endif
};

uint64_t f_sub (uint64_t a, uint64_t b)
{
        return a-b;
};
```

20.2.1 x86

Listing 20.2: Con optimización MSVC 2012 /Ob1

```
_a$ = 8         ; size = 8
```

```
_b$ = 16        ; size = 8
_f_add  PROC
        mov     eax, DWORD PTR _a$[esp-4]
        add     eax, DWORD PTR _b$[esp-4]
        mov     edx, DWORD PTR _a$[esp]
        adc     edx, DWORD PTR _b$[esp]
        ret     0
_f_add  ENDP

_f_add_test PROC
        push    5461            ; 00001555H
        push    1972608889      ; 75939f79H
        push    2874            ; 00000b3aH
        push    1942892530      ; 73ce2ff_subH
        call    _f_add
        push    edx
        push    eax
        push    OFFSET $SG1436 ; '%I64d', 0aH, 00H
        call    _printf
        add     esp, 28
        ret     0
_f_add_test ENDP

_f_sub  PROC
        mov     eax, DWORD PTR _a$[esp-4]
        sub     eax, DWORD PTR _b$[esp-4]
        mov     edx, DWORD PTR _a$[esp]
        sbb     edx, DWORD PTR _b$[esp]
        ret     0
_f_sub  ENDP
```

f_add_test().

.

...

. SUB : .

.

20.3

```
#include <stdint.h>

uint64_t f_mul (uint64_t a, uint64_t b)
{
        return a*b;
};

uint64_t f_div (uint64_t a, uint64_t b)
{
        return a/b;
};

uint64_t f_rem (uint64_t a, uint64_t b)
{
        return a % b;
};
```

20.3.1 x86

Listing 20.3: Con optimización MSVC 2013 /Ob1

```
```

```
_a$ = 8 ; size = 8
_b$ = 16 ; size = 8
_f_mul  PROC
        push    ebp
        mov     ebp, esp
        mov     eax, DWORD PTR _b$[ebp+4]
        push    eax
        mov     ecx, DWORD PTR _b$[ebp]
        push    ecx
        mov     edx, DWORD PTR _a$[ebp+4]
        push    edx
        mov     eax, DWORD PTR _a$[ebp]
        push    eax
        call    __allmul ; long long multiplication
        pop     ebp
        ret     0
_f_mul  ENDP

_a$ = 8 ; size = 8
_b$ = 16 ; size = 8
_f_div  PROC
        push    ebp
        mov     ebp, esp
        mov     eax, DWORD PTR _b$[ebp+4]
        push    eax
        mov     ecx, DWORD PTR _b$[ebp]
        push    ecx
        mov     edx, DWORD PTR _a$[ebp+4]
        push    edx
        mov     eax, DWORD PTR _a$[ebp]
        push    eax
        call    __aulldiv ; unsigned long long division
        pop     ebp
        ret     0
_f_div  ENDP

_a$ = 8 ; size = 8
_b$ = 16 ; size = 8
_f_rem  PROC
        push    ebp
        mov     ebp, esp
        mov     eax, DWORD PTR _b$[ebp+4]
        push    eax
        mov     ecx, DWORD PTR _b$[ebp]
        push    ecx
        mov     edx, DWORD PTR _a$[ebp+4]
        push    edx
        mov     eax, DWORD PTR _a$[ebp]
        push    eax
        call    __aullrem ; unsigned long long remainder
        pop     ebp
        ret     0
_f_rem  ENDP
```

.

20.4

```
#include <stdint.h>

uint64_t f (uint64_t a)
{
        return a>>7;
};
```

20.4.1 x86

Listing 20.4: Con optimización MSVC 2012 /Ob1

```
_a$ = 8         ; size = 8
_f      PROC
        mov     eax, DWORD PTR _a$[esp-4]
        mov     edx, DWORD PTR _a$[esp]
        shrd    eax, edx, 7
        shr     edx, 7
        ret     0
_f      ENDP
```

. .

20.5

```
#include <stdint.h>

int64_t f (int32_t a)
{
        return a;
};
```

20.5.1 x86

Listing 20.5: Con optimización MSVC 2012

```
_a$ = 8
_f      PROC
        mov     eax, DWORD PTR _a$[esp-4]
        cdq
        ret     0
_f      ENDP
```

. . . . .

Capítulo 21

21.1 x86-64

- RAX, RBX, RCX, RDX, RBP, RSP, RSI, RDI, R8, R9, R10, R11, R12, R13, R14, R15.

7mo (número de byte)	6to	5to	4to	3ro	2do	1ro	0
RAX^{x64}							
				EAX			
						AX	
						AH	AL

7mo (número de byte)	6to	5to	4to	3ro	2do	1ro	0
R8							
				R8D			
						R8W	
							R8L

  XMM0-XMM15.
- System V AMD64 ABI (Linux, *BSD, Mac OS X)[Mit13] fastcall, RDI, RSI, RDX, RCX, R8, R9 . .
-
- .

Parte II

Fundamentos importantes

CAPÍTULO 22. REPRESENTACIÓN DE NÚMEROS CON SIGNO

Capítulo 22

Representación de números con signo

Hay distintos métodos para representar números con signo[1], pero el «complemento a dos» es el más popular en las computadoras.

Aquí hay una tabla con los valores de algunos bytes:

binario	hexadecimal	sin signo	con signo (complemento a dos)
01111111	0x7f	127	127
01111110	0x7e	126	126
...			
00000110	0x6	6	6
00000101	0x5	5	5
00000100	0x4	4	4
00000011	0x3	3	3
00000010	0x2	2	2
00000001	0x1	1	1
00000000	0x0	0	0
11111111	0xff	255	-1
11111110	0xfe	254	-2
11111101	0xfd	253	-3
11111100	0xfc	252	-4
11111011	0xfb	251	-5
11111010	0xfa	250	-6
...			
10000010	0x82	130	-126
10000001	0x81	129	-127
10000000	0x80	128	-128

La diferencia entre números con signo y sin signo está en que si representamos 0xFFFFFFFE y 0x00000002 sin signo, el primero (4294967294) es mayor que el segundo (2). Pero si representamos ambos con signo, el primero se vuelve -2, y es menor que el segundo (2). Esa es la razón por la que se tienen saltos condicionales (11 on page 30) tanto para operaciones con signo (e.g. JG, JL) como sin signo (JA, JB).

Con el fin de la simplicidad, esto es lo que uno debe de saber:

- Los números pueden ser con signo y sin signo.
- Tipos con signo en C/C++:
  - `int64_t` (-9,223,372,036,854,775,808..9,223,372,036,854,775,807) (- 9.2.. 9.2 billones) o 0x8000000000000000..0x7FFFFFFFFFFFFFFF),
  - *int* (-2,147,483,648..2,147,483,647 (- 2.15.. 2.15Gb) o 0x80000000..0x7FFFFFFF),
  - *char* (-128..127 o 0x80..0x7F),
  - `ssize_t`.

  Sin signo:
  - `uint64_t` (0..18,446,744,073,709,551,615 (18 billones) o 0..0xFFFFFFFFFFFFFFFF),
  - `unsigned int` (0..4,294,967,295 (4.3Gb) o 0..0xFFFFFFFF),
  - `unsigned char` (0..255 o 0..0xFF),

[1] wikipedia

  - `size_t`.

- En los tipos con signo, el signo se encuentra en el bit más significativo: 1 significa «menos», 0 significa «más».
- Promover a un tipo de dato más grande es sencillo: 20.5 on page 92.
- La negación es simple: sólo invierte todos los bits y suma 1. Podemos recordar que un número con signo contrario se localiza en el lado opuesto a la misma distancia de cero. La suma de 1 es necesaria porque el cero se localiza en medio.
- Las operaciones de suma y resta funcionan bien para valores tanto con signo como sin signo. Sin embargo, para las operaciones de multiplicación y división x86 tiene instrucciones distintas: `IDIV/IMUL` para números con signo y `DIV/MUL` para números sin signo.

Capítulo 23

Memoria

Existen 3 tipos principales de memoria:

- Memoria global AKA[1] «asignación estática de memoria». No hay necesidad de asignarla explícitamente, la asignación es realizada al declarar variables/arreglos globales. Estas variables globales residen en los segmentos de datos o de constantes. Están disponibles globalmente (por lo tanto, se consideran un anti-patrón). No son convenientes para buffers/arreglos porque deben tener un tamaño fijo. Los desbordamientos de buffer que occurren aquí usualmente sobreescriben variables o buffers que residen junto a ellos en memoria. En este libro hay un ejemplo: 7.2 on page 17.
- Pila AKA «asignación en pila». La asignación se realiza al declarar variables/arreglos dentro de una función. Son usualmente variables locales a la función. Algunas veces estas variables locales también estan disponibles para funciones descendientes (funciones llamadas, si aquel que la llama le pasa un apuntador a una de sus variables). La asignación y desasignación son muy rápidas, sólo necesita que SP[2] sea ajustado. Los desbordamientos de buffer suelen reescribir estructuras importantes en la pila: 16.2 on page 58.
- Heap AKA «asignación dinámica de memoria». La asignación/desasignación es realizada llamando a `malloc()/free()` o `new/delete` en C++. Éste es el método más conveniente: el tamaño del bloque puede establecerse en tiempo de ejecución. Cambiar el tamaño es posible (usando `realloc()`), pero puede ser lento. Ésta es la forma más lenta de asignar memoria: el asignador de memoria debe suportar y actualizar todas las estructuras de control mientras se asigna y desasigna. Los desbordamientos de buffer suelen sobreescribir estas estructuras. Las asiganciones en el heap también son el origen de problemas de fuga de memoria: cada bloque de memoria tiene que ser desasgnado explícitamente, pero uno puede olvidarse de ello, o hacerlo de manera incorrecta. Otro problema es el «uso después de la liberación»—usar un bloque de memoria después de que `free()` ha sido llamado en él, lo cual es muy peligroso. Un ejemplo en este libro: 19.2 on page 81.

[1] Also Known As - (También Conocido Como)
[2] stack pointer. SP/ESP/RSP en x86/x64. SP en ARM.

Parte III

http://www.google.com/search?q=CsDecomprLZC

Capítulo 24

(win32)

.

: Process Monitor[1] SysInternals.

Wireshark[2].

.

blog.yurichev.com.

24.1 Windows API

.. CRT[3].

- (advapi32.dll): RegEnumKeyEx[4] [5], RegEnumValue[6] [5], RegGetValue[7] [5], RegOpenKeyEx[8] [5], RegQueryValueEx[9] [5].
- (kernel32.dll): GetPrivateProfileString [10] [5].
- (user32.dll): MessageBox [11] [5], MessageBoxEx [12] [5], SetDlgItemText [13] [5], GetDlgItemText [14] [5].
- : (user32.dll): LoadMenu [15] [5].
- (ws2_32.dll): WSARecv [16], WSASend [17].
- (kernel32.dll): CreateFile [18] [5], ReadFile [19], ReadFileEx [20], WriteFile [21], WriteFileEx [22].
- Internet (wininet.dll): WinHttpOpen [23].
- (wintrust.dll): WinVerifyTrust [24].

[1] http://go.yurichev.com/17301
[2] http://go.yurichev.com/17303
[3] C runtime library
[4] MSDN
[5]
[6] MSDN
[7] MSDN
[8] MSDN
[9] MSDN
[10] MSDN
[11] MSDN
[12] MSDN
[13] MSDN
[14] MSDN
[15] MSDN
[16] MSDN
[17] MSDN
[18] MSDN
[19] MSDN
[20] MSDN
[21] MSDN
[22] MSDN
[23] MSDN
[24] MSDN

- (msvcr*.dll): assert, itoa, ltoa, open, printf, read, strcmp, atol, atoi, fopen, fread, fwrite, memcmp, rand, strlen, strstr, strchr.

24.2 tracer:

```
--one-time-INT3-bp:somedll.dll!.*
```

```
--one-time-INT3-bp:somedll.dll!xml.*
```

```
tracer -l:uptime.exe --one-time-INT3-bp:cygwin1.dll!.*
```

:

```
One-time INT3 breakpoint: cygwin1.dll!__main (called from uptime.exe!OEP+0x6d (0x40106d))
One-time INT3 breakpoint: cygwin1.dll!_geteuid32 (called from uptime.exe!OEP+0xba3 (0x401ba3))
One-time INT3 breakpoint: cygwin1.dll!_getuid32 (called from uptime.exe!OEP+0xbaa (0x401baa))
One-time INT3 breakpoint: cygwin1.dll!_getegid32 (called from uptime.exe!OEP+0xcb7 (0x401cb7))
One-time INT3 breakpoint: cygwin1.dll!_getgid32 (called from uptime.exe!OEP+0xcbe (0x401cbe))
One-time INT3 breakpoint: cygwin1.dll!sysconf (called from uptime.exe!OEP+0x735 (0x401735))
One-time INT3 breakpoint: cygwin1.dll!setlocale (called from uptime.exe!OEP+0x7b2 (0x4017b2))
One-time INT3 breakpoint: cygwin1.dll!_open64 (called from uptime.exe!OEP+0x994 (0x401994))
One-time INT3 breakpoint: cygwin1.dll!_lseek64 (called from uptime.exe!OEP+0x7ea (0x4017ea))
One-time INT3 breakpoint: cygwin1.dll!read (called from uptime.exe!OEP+0x809 (0x401809))
One-time INT3 breakpoint: cygwin1.dll!sscanf (called from uptime.exe!OEP+0x839 (0x401839))
One-time INT3 breakpoint: cygwin1.dll!uname (called from uptime.exe!OEP+0x139 (0x401139))
One-time INT3 breakpoint: cygwin1.dll!time (called from uptime.exe!OEP+0x22e (0x40122e))
One-time INT3 breakpoint: cygwin1.dll!localtime (called from uptime.exe!OEP+0x236 (0x401236))
One-time INT3 breakpoint: cygwin1.dll!sprintf (called from uptime.exe!OEP+0x25a (0x40125a))
One-time INT3 breakpoint: cygwin1.dll!setutent (called from uptime.exe!OEP+0x3b1 (0x4013b1))
One-time INT3 breakpoint: cygwin1.dll!getutent (called from uptime.exe!OEP+0x3c5 (0x4013c5))
One-time INT3 breakpoint: cygwin1.dll!endutent (called from uptime.exe!OEP+0x3e6 (0x4013e6))
One-time INT3 breakpoint: cygwin1.dll!puts (called from uptime.exe!OEP+0x4c3 (0x4014c3))
```

Capítulo 25

25.1

25.1.1 C/C++

(ASCIIZ-).

. [Rit79] :

> A minor difference was that the unit of I/O was the word, not the byte, because the PDP-7 was a word-addressed machine. In practice this meant merely that all programs dealing with character streams ignored null characters, because null was used to pad a file to an even number of characters.

:

```
int main()
{
        printf ("Hello, world!\n");
};
```

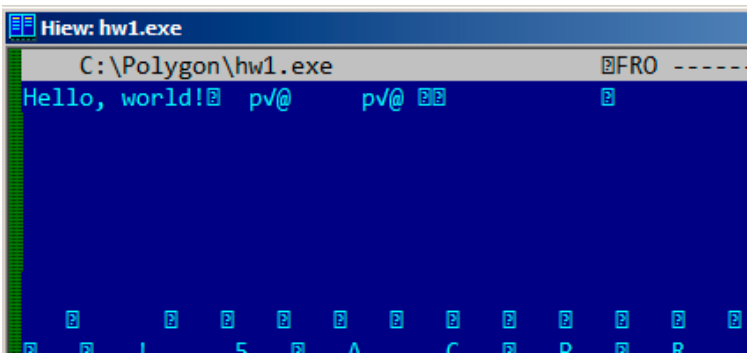


Figura 25.1: Hiew

25.1.2 Borland Delphi

:

Listing 25.1: Delphi

```
CODE:00518AC8                 dd 19h
CODE:00518ACC aLoading___Plea db 'Loading... , please wait.',0

...

CODE:00518AFC                 dd 10h
CODE:00518B00 aPreparingRun__ db 'Preparing run...',0
```

25.1.3 Unicode

. . .

: UTF-8 () y UTF-16LE (Windows).

UTF-8

UTF-8 . . .

[1]:

```
How much? 100€?

(English) I can eat glass and it doesn't hurt me.
(Greek) Μπορώ να φάω σπασμένα γυαλιά χωρίς να πάθω τίποτα.
(Hungarian) Meg tudom enni az üveget, nem lesz tőle bajom.
(Icelandic) Ég get etið gler án þess að meiða mig.
(Polish) Mogę jeść szkło i mi nie szkodzi.
(Russian) Я могу есть стекло, оно мне не вредит.
(Arabic): أنا قادر على أكل الزجاج و هذا لا يؤلمني.
(Hebrew): אני יכול לאכול זכוכית וזה לא מזיק לי.
(Chinese) 我能吞下玻璃而不伤身体。
(Japanese) 私はガラスを食べられます。それは私を傷つけません。
(Hindi) मैं काँच खा सकता हूँ और मुझे उससे कोई चोट नहीं पहुंचती.
```

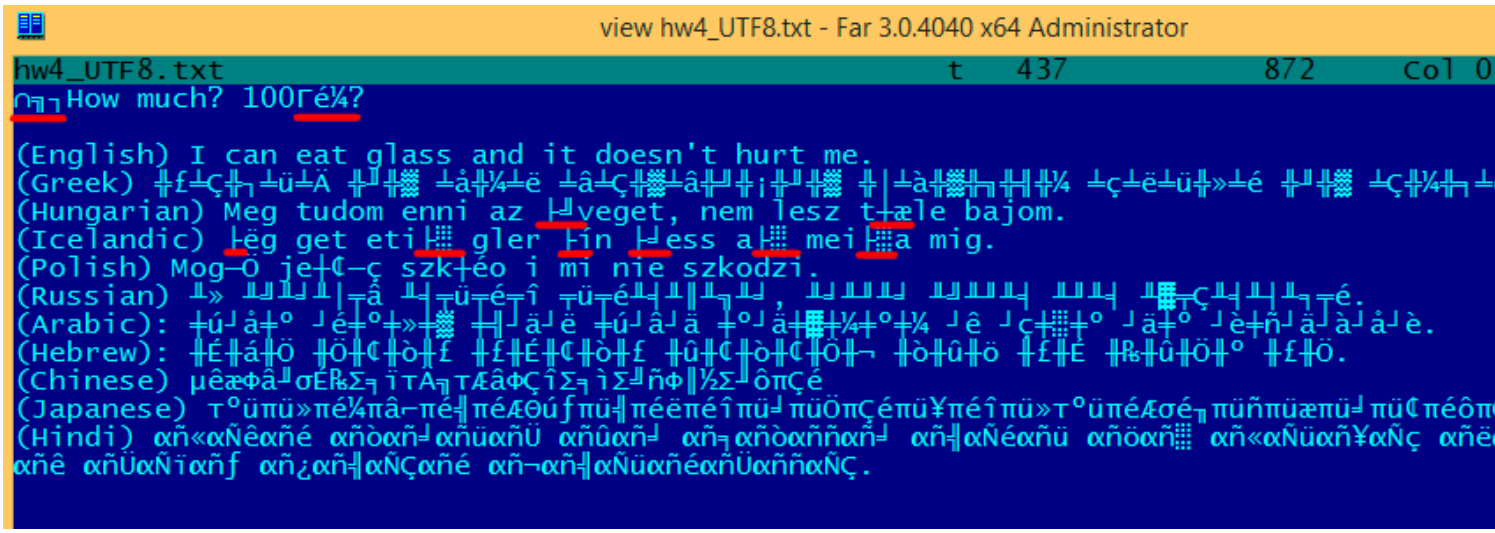


Figura 25.2: FAR: UTF-8

. . . . . .

. BOM[2] .

UTF-16LE

-A y -W. (*wide*). .

:

```
int wmain()
{
        wprintf (L"Hello, world!\n");
};
```

[1]: http://go.yurichev.com/17304
[2]Byte order mark

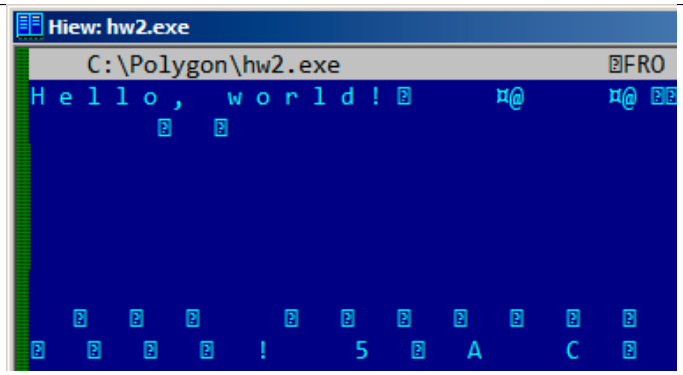


Figura 25.3: Hiew

:

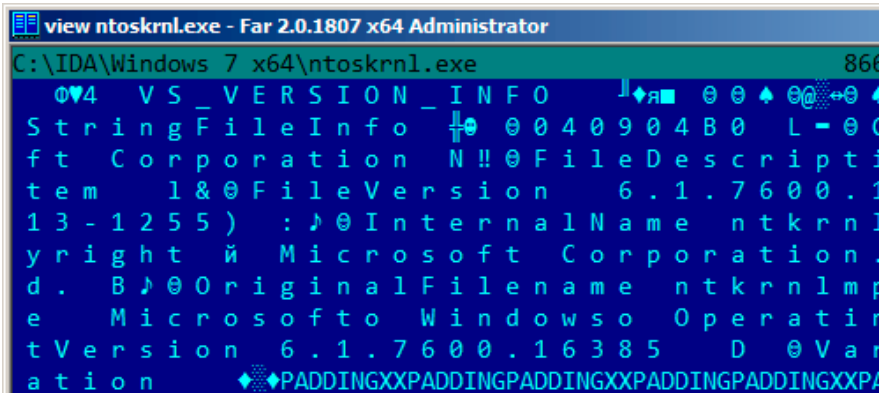


Figura 25.4: Hiew

:

```
.data:0040E000 aHelloWorld:
.data:0040E000                 unicode 0, <Hello, world!>
.data:0040E000                 dw 0Ah, 0
```

:

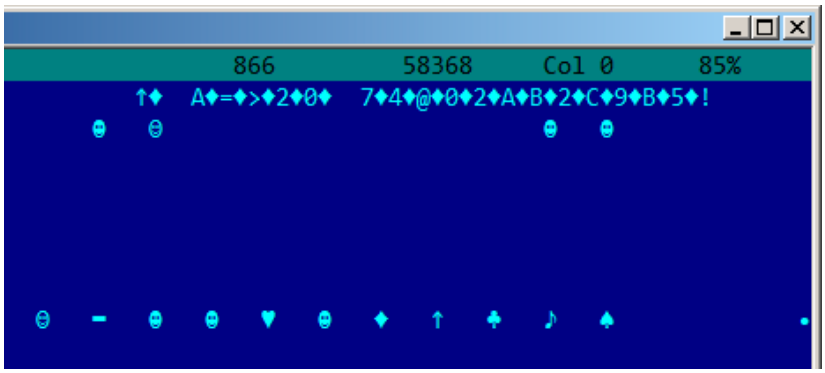


Figura 25.5: Hiew: UTF-16LE

. [3]. 0x400-0x4FF.

..

[3] wikipedia

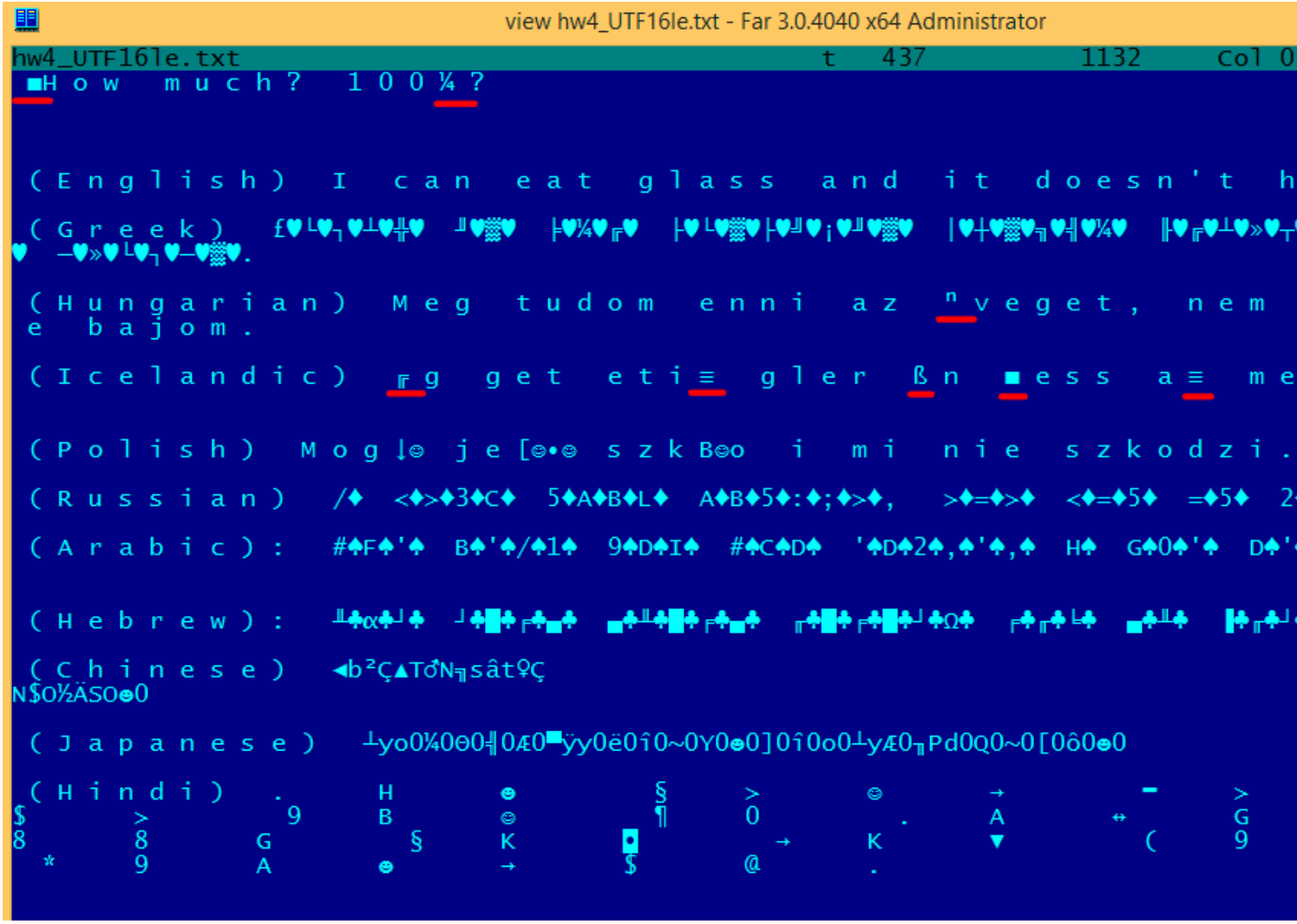


Figura 25.6: FAR: UTF-16LE

. . .

25.1.4 Base64

.

```
AVjbbVSVfcUMu1xvjaMgjNtueRwBbxnyJw8dpGnLW8ZW8aKG3v4YOicuQT+qEJAp9lAOuWs=
```

```
WVjbbVSVfcUMu1xvjaMgjNtueRwBbxnyJw8dpGnLW8ZW8aKG3v4YOicuQT+qEJAp9lAOuQ==
```

25.2

[4].

: blog.yurichev.com.

25.3

[5].

http://192.168.0.1/userRpmNatDebugRpm26525557/start_art.html.

[4] blog.yurichev.com
[5] http://sekurak.pl/tp-link-httptftp-backdoor/

base64.

Capítulo 26

Listing 26.1:

```
.text:107D4B29 mov  dx, [ecx+42h]
.text:107D4B2D cmp  edx, 1
.text:107D4B30 jz   short loc_107D4B4A
.text:107D4B32 push 1ECh
.text:107D4B37 push offset aWrite_c ; "write.c"
.text:107D4B3C push offset aTdTd_planarcon ; "td->td_planarconfig == PLANARCONFIG_CON"...
.text:107D4B41 call ds:_assert

...

.text:107D52CA mov  edx, [ebp-4]
.text:107D52CD and  edx, 3
.text:107D52D0 test edx, edx
.text:107D52D2 jz   short loc_107D52E9
.text:107D52D4 push 58h
.text:107D52D6 push offset aDumpmode_c ; "dumpmode.c"
.text:107D52DB push offset aN30     ; "(n & 3) == 0"
.text:107D52E0 call ds:_assert

...

.text:107D6759 mov  cx, [eax+6]
.text:107D675D cmp  ecx, 0Ch
.text:107D6760 jle  short loc_107D677A
.text:107D6762 push 2D8h
.text:107D6767 push offset aLzw_c   ; "lzw.c"
.text:107D676C push offset aSpLzw_nbitsBit ; "sp->lzw_nbits <= BITS_MAX"
.text:107D6771 call ds:_assert
```

Capítulo 27

10, 100, 1000, .

: 10=0xA, 100=0x64, 1000=0x3E8, 10000=0x2710.

0xAAAAAAAA (10101010101010101010101010101010) y
0x55555555 (01010101010101010101010101010101) . 0x55AA , MBR[1], y en ROM[2] .

```
var int h0 := 0x67452301
var int h1 := 0xEFCDAB89
var int h2 := 0x98BADCFE
var int h3 := 0x10325476
```

:

Listing 27.1: linux/lib/crc16.c

```
/** CRC table for the CRC-16. The poly is 0x8005 (x^16 + x^15 + x^2 + 1) */
u16 const crc16_table[256] = {
        0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
        0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
        0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
        ...
```

27.1

«MZ»[3].

```
cmp [buf], 0x6468544D ; "MThd"
jnz _error_not_a_MIDI_file
```

27.1.1

```
void xor_pwd(void)
{
        int i;

        pwd^=0x09071966;
        for(i=0;i<8;i++)
        {
                al_buf[i]= pwd & 7; pwd = pwd >> 3;
        }
};

void emulate_func2(unsigned short seed)
{
        int i, j;
        for(i=0;i<8;i++)
        {
                ch[i] = 0;
```

[1] Master Boot Record
[2] Memoria de Solo Lectura
[3] wikipedia

```
                for(j=0;j<8;j++)
                {
                        seed *= 0x1989;
                        seed += 5;
                        ch[i] |= (tab[(seed>>9)&0x3f]) << (7-j);
                }
        }
}
```

27.1.2 DHCP

`DhcpExtractOptionsForValidation()` y `DhcpExtractFullOptions()`:

Listing 27.2: dhcpcore.dll (Windows 7 x64)

```
.rdata:000007FF6483CBE8 dword_7FF6483CBE8 dd 63538263h          ; DATA XREF: ⇙
    ↳ DhcpExtractOptionsForValidation+79
.rdata:000007FF6483CBEC dword_7FF6483CBEC dd 63538263h          ; DATA XREF: ⇙
    ↳ DhcpExtractFullOptions+97
```

Listing 27.3: dhcpcore.dll (Windows 7 x64)

```
.text:000007FF6480875F  mov     eax, [rsi]
.text:000007FF64808761  cmp     eax, cs:dword_7FF6483CBE8
.text:000007FF64808767  jnz     loc_7FF64817179
```

Listing 27.4: dhcpcore.dll (Windows 7 x64)

```
.text:000007FF648082C7  mov     eax, [r12]
.text:000007FF648082CB  cmp     eax, cs:dword_7FF6483CBEC
.text:000007FF648082D1  jnz     loc_7FF648173AF
```

27.2

binary grep[4].

[4] GitHub

Capítulo 28

```
cat EXCEL.lst | grep fdiv | grep -v dbl_ > EXCEL.fdiv
```

```
.text:3011E919 DC 33                                    fdiv    qword ptr [ebx]
```

```
PID=13944|TID=28744|(0) 0x2f64e919 (Excel.exe!BASE+0x11e919)
EAX=0x02088006 EBX=0x02088018 ECX=0x00000001 EDX=0x00000001
ESI=0x02088000 EDI=0x00544804 EBP=0x0274FA3C ESP=0x0274F9F8
EIP=0x2F64E919
FLAGS=PF IF
FPU ControlWord=IC RC=NEAR PC=64bits PM UM OM ZM DM IM
FPU StatusWord=
FPU ST(0): 1.000000
```

[EBX].

```
.text:3011E91B DD 1E                                    fstp    qword ptr [esi]
```

```
PID=32852|TID=36488|(0) 0x2f40e91b (Excel.exe!BASE+0x11e91b)
EAX=0x00598006 EBX=0x00598018 ECX=0x00000001 EDX=0x00000001
ESI=0x00598000 EDI=0x00294804 EBP=0x026CF93C ESP=0x026CF8F8
EIP=0x2F40E91B
FLAGS=PF IF
FPU ControlWord=IC RC=NEAR PC=64bits PM UM OM ZM DM IM
FPU StatusWord=C1 P
FPU ST(0): 0.333333
```

```
tracer -l:excel.exe bpx=excel.exe!BASE+0x11E91B,set(st0,666)
```

```
PID=36540|TID=24056|(0) 0x2f40e91b (Excel.exe!BASE+0x11e91b)
EAX=0x00680006 EBX=0x00680018 ECX=0x00000001 EDX=0x00000001
ESI=0x00680000 EDI=0x00395404 EBP=0x0290FD9C ESP=0x0290FD58
EIP=0x2F40E91B
FLAGS=PF IF
FPU ControlWord=IC RC=NEAR PC=64bits PM UM OM ZM DM IM
FPU StatusWord=C1 P
FPU ST(0): 0.333333
Set ST0 register to 666.000000
```

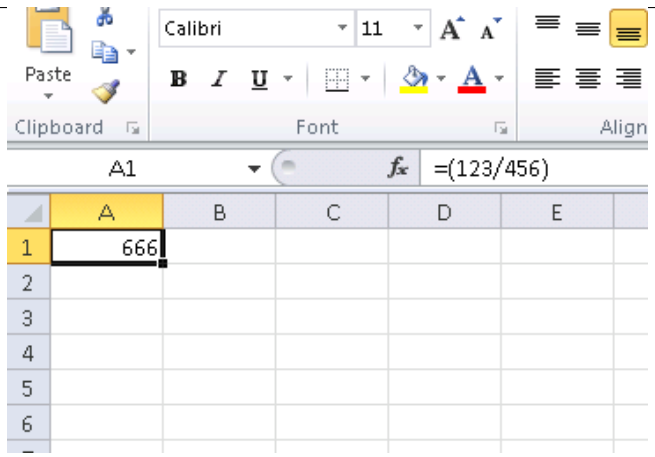


Figura 28.1:

```
tracer.exe -l:excel.exe bpx=excel.exe!BASE+0x1B7FCC,set(st0,666)
```

Capítulo 29

29.1

XOR op, op (, XOR EAX, EAX) «». . etc.

:

```
gawk -e '$2=="xor" { tmp=substr($3, 0, length($3)-1); if (tmp!=$4) if($4!="esp") if ($4!="ebp")↙
    ↳  { print $1, $2, tmp, ",", $4 } }' filename.lst
```

29.2

.

Windows 2003 (ntoskrnl.exe):

```
MultiplyTest proc near               ; CODE XREF: Get386Stepping
             xor     cx, cx
loc_620555:                          ; CODE XREF: MultiplyTest+E
             push    cx
             call    Multiply
             pop     cx
             jb      short locret_620563
             loop    loc_620555
             clc
locret_620563:                       ; CODE XREF: MultiplyTest+C
             retn
MultiplyTest endp

Multiply     proc near               ; CODE XREF: MultiplyTest+5
             mov     ecx, 81h
             mov     eax, 417A000h
             mul     ecx
             cmp     edx, 2
             stc
             jnz     short locret_62057F
             cmp     eax, 0FE7A000h
             stc
             jnz     short locret_62057F
             clc
locret_62057F:                       ; CODE XREF: Multiply+10
                                     ; Multiply+18
             retn
Multiply     endp
```

WRK[1] v1.2 *WRK-v1.2\base\ntos\ke\i386\cpu.asm*.

[1]Windows Research Kernel

Capítulo 30

:

```
0x150bf66 (_kziaia+0x14), e=       1 [MOV EBX, [EBP+8]] [EBP+8]=0xf59c934
0x150bf69 (_kziaia+0x17), e=       1 [MOV EDX, [69AEB08h]] [69AEB08h]=0
0x150bf6f (_kziaia+0x1d), e=       1 [FS: MOV EAX, [2Ch]]
0x150bf75 (_kziaia+0x23), e=       1 [MOV ECX, [EAX+EDX*4]] [EAX+EDX*4]=0xf1ac360
0x150bf78 (_kziaia+0x26), e=       1 [MOV [EBP-4], ECX] ECX=0xf1ac360
```

Capítulo 31

31.1

31.2

:

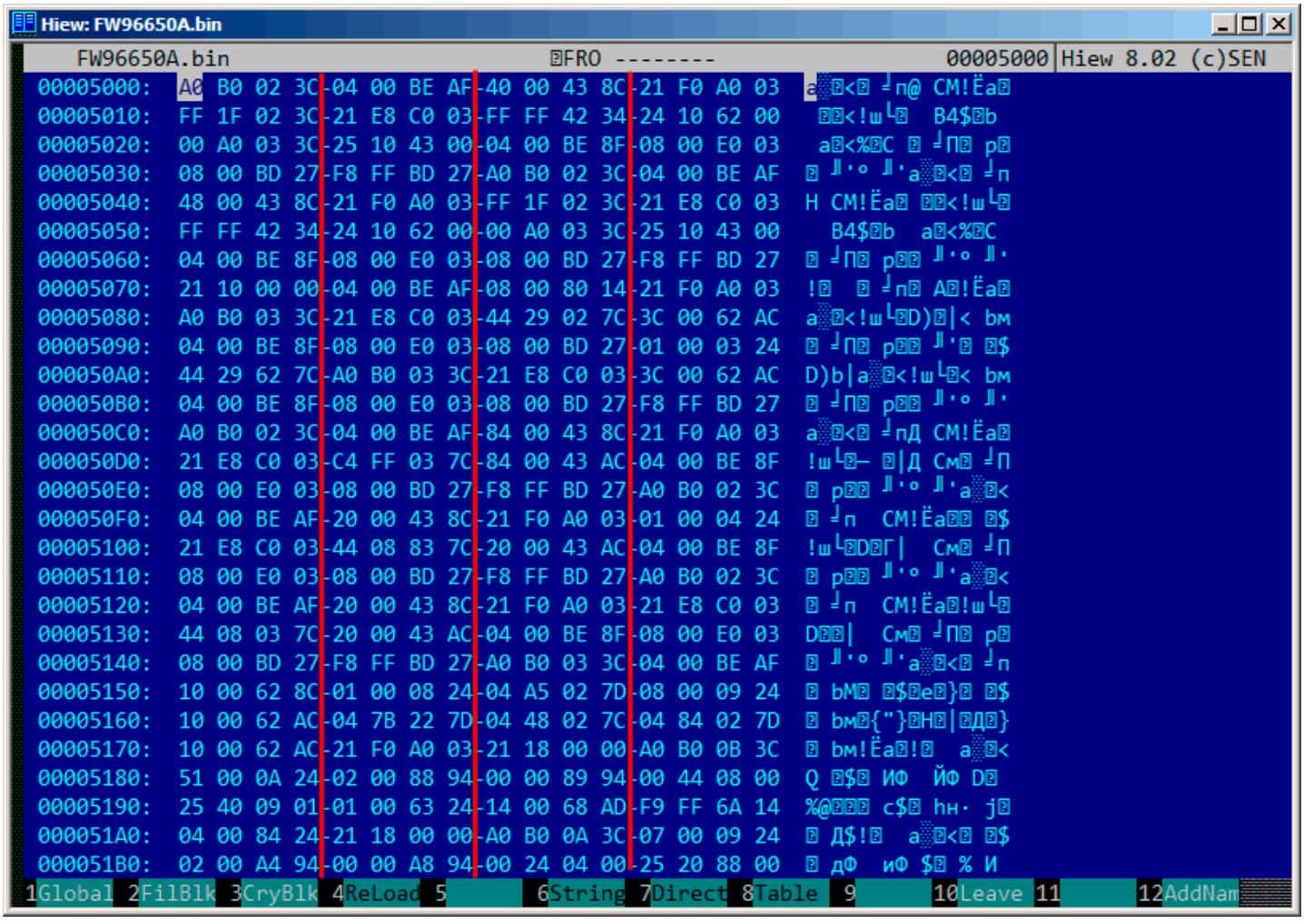


Figura 31.1: Hiew:

31.3

wikipedia.

31.3.1

.

31.3.2

Parte IV

Herramientas

Capítulo 32

Desensamblador

32.1 IDA

Una versión freeware anterior está disponible para descargar [1].

[1] hex-rays.com/products/ida/support/download_freeware.shtml

CAPÍTULO 33. DEPURADOR

Capítulo 33

Depurador

33.1 tracer

El autor suele utilizar *tracer* [1] (recurso en inglés) en vez de un depurador.

El autor de estas líneas eventualmente dejó de utilizar un depurador, ya que lo único que necesita es hallar los argumentos de las funciones durante la ejecución, o el estado de los registros en algún punto. Cargar un depurador en cada ocasión es demasiado, y fue así como nació una utilería llamada *tracer*. Funciona a través de la línea de comandos, permitiendo interceptar la ejecucón de una función, colocar breakpoints en lugares arbitrarios, leer y cambiar el estado de los registros, etc.

Sin embargo, con fines de aprendizaje es altamente recomendable trazar el código manualmente en un depurador, observar cómo cambia el estado de los registros (e.g. los clásicos SoftICE, OllyDbg, WinDbg subrayan los registros modificado), de las banderas, de los datos, modificarlos, observar la reacción, etc.

[1] yurichev.com

CAPÍTULO 34. DECOMPILADORES

Capítulo 34

Decompiladores

Existe un solo decompilador a código C de alta calidad y disponible públicamente: Hex-Rays: hex-rays.com/products/decompiler/

Capítulo 35

Otras herramientas

- Microsoft Visual Studio Express[1]: La versión mínima y gratuita de Visual Studio, conveniente para experimentos sencillos.
- Hiew[2]: para realizar modificaciones de código pequeñas en archivos binarios.
- binary grep: una pequeña utilería para buscar cualquier secuencia de bytes en un montón de archivos, incluyendo archivos no ejecutables: GitHub.

[1] visualstudio.com/en-US/products/visual-studio-express-vs
[2] hiew.ru

Parte V

Libros/blogs que merecen lectura

Capítulo 36

Libros

36.1 Windows

[RA09].

36.2 C/C++

[ISO13].

36.3 x86 / x86-64

[Int13], [AMD13]

36.4 ARM

Manuales de ARM: http://go.yurichev.com/17024

36.5 Criptografía

[Sch94]

CAPÍTULO 37. BLOGS

Capítulo 37

Blogs

37.1 Windows

- Microsoft: Raymond Chen
- nynaeve.net

CAPÍTULO 38. OTROS

Capítulo 38

Otros

Existen dos excelentes subreddits relacionados con RE[1] en reddit.com: reddit.com/r/ReverseEngineering/ y reddit.com/r/remath (en los tópicos de la intersección de RE y matemáticas).

También hay una sección sobre RE en el sitio web de Stack Exchange:

reverseengineering.stackexchange.com.

En IRC hay un canal ##re enFreeNode[2].

[1] Reverse Engineering
[2] freenode.net

Capítulo 39

No dudes en enviar cualquier pregunta por email al autor:<dennis(a)yurichev.com>

¿Tienes alguna sugerencia sobre nuevas cosas que puedan agregarse al libro?

Por favor, no dudes en enviar correcciones (incluyendo gramática),etc.

El autor está trabajando arduamente en el libro, así que los números de página, de listado, etc. cambian con frecuencia.

Por favor, no utilices como referencia números de página o de listado en tus emails.

Existe un método mucho más simple: toma una impresión de pantalla de la página, subraya el lugar donde se encuentra el error en un editor de gráficos, y envíalo. De este modo será arreglado más rápido.

Y si estás fmiliarizado con git y LaTeX puedes arreglar el error directo en el código fuente

GitHub.

No te preocupes por escribirme acerca de errores insignificantes que encuentras, incluso si no te sientes seguro. Estoy escribiendo para principiantes, después de todo, por lo tanto la opinión de los principiantes y sus comentarios son cruciales para mi trabajo.

¡Atención: ésta es una versión LITE resumida!

Es apróximadamente 6 veces más corta que la versión completa (~150 páginas) y está dirigida a aquellos que deseen una introducción breve a la esencia de la ingeniería inversa. No incluye nada sobre MIPS, ARM, OllyDBG, GCC, GDB, IDA, no contiene ejercicios, ejemplos, etc.

Si aún estás interesado en la ingeniería inversa, la versión completa del libro siempre está disponible en mi sitio: beginners.re.

Acrónimos utilizados

SO Sistema Operativo . . . x

LP Lenguaje de Programación . . . 3

ROM Memoria de Solo Lectura . . . 109

RA Dirección de Retorno . . . 40

SP stack pointer. SP/ESP/RSP en x86/x64. SP en ARM. . . . 98

IDA Desensamblador Interactivo y depurador desarrollado por Hex-Rays

AKA Also Known As - (También Conocido Como) . . . 98

CRT C runtime library . . . 101

CPU Central processing unit . . . x

DBMS Database management systems . . . ix

ISA Instruction Set Architecture . . . 3

NOP No OPeration . . . 21

ASCIIZ ASCII Zero () . . . 20

WRK Windows Research Kernel . . . 113

GPR General Purpose Registers . . . 3

RE Reverse Engineering . . . 125

BOM Byte order mark . . . 104

MBR Master Boot Record . . . 109

Glosario

Spanish text placeholder Spanish text placeholder. 130

reverse engineering Spanish text placeholder. v

Índice alfabético

Bibliografía

[AMD13] AMD. AMD64 Architecture Programmer's Manual. También disponible como http://go.yurichev.com/17284. 2013.

[Dij68] Edsger W. Dijkstra. «Letters to the editor: go to statement considered harmful». En: Commun. ACM 11.3 (mar. de 1968), págs. 147-148. ISSN: 0001-0782. DOI: 10.1145/362929.362947. URL: http://go.yurichev.com/17299.

[Fog13] Agner Fog. The microarchitecture of Intel, AMD and VIA CPUs / An optimization guide for assembly programmers and com http://go.yurichev.com/17278. 2013.

[Int13] Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes:1, 2A, 2B, 2C, 3A, 3B, and 3C. También disponible como http://go.yurichev.com/17283. 2013.

[ISO13] ISO. ISO/IEC 14882:2011 (C++ 11 standard). También disponible como http://go.yurichev.com/17275. 2013.

[Ker88] Brian W. Kernighan. The C Programming Language. Ed. por Dennis M. Ritchie. 2nd. Prentice Hall Professional Technical Reference, 1988. ISBN: 0131103709.

[Knu74] Donald E. Knuth. «Structured Programming with go to Statements». En: ACM Comput. Surv. 6.4 (dic. de 1974). Also available as http://go.yurichev.com/17271, págs. 261-301. ISSN: 0360-0300. DOI: 10.1145/356635.356640. URL: http://go.yurichev.com/17300.

[Mit13] Michael Matz / Jan Hubicka / Andreas Jaeger / Mark Mitchell. System V Application Binary Interface. AMD64 Architecture Pr También disponible como http://go.yurichev.com/17295. 2013.

[Pre+07] William H. Press y col. Numerical Recipes. 2007.

[RA09] Mark E. Russinovich y David A. Solomon with Alex Ionescu. Windows® Internals: Including Windows Server 2008 and Wind 2009.

[Rit79] Dennis M. Ritchie. «The Evolution of the Unix Time-sharing System». En: (1979).

[RT74] D. M. Ritchie y K. Thompson. «The UNIX Time Sharing System». En: (1974). También disponible como http://go.yurichev.com/17270.

[Sch94] Bruce Schneier. Applied Cryptography: Protocols, Algorithms, and Source Code in C. 1994.

[Yur13] Dennis Yurichev. C/C++ programming language notes. También disponible como http://go.yurichev.com/17289. 2013.