

Programación a Nivel-Máquina I: Conceptos Básicos

Estructura de Computadores
Semana 2

Bibliografía:

[BRY11] Cap.3

Computer Systems: A Programmer's Perspective. Bryant, O'Hallaron. Pearson, 2011

Signatura ESIIT/[C.1 BRY com](#)

Transparencias del libro CS:APP, Cap.3

Introduction to Computer Systems: a Programmer's Perspective

Autores: Randal E. Bryant y David R. O'Hallaron

Guía de trabajo autónomo (4h/s)

■ **Lectura:** del Cap.3 CS:APP (Bryant/O'Hallaron)

- Historical perspective, Program Encodings, Data Formats, Accessing Info.
 - 3.1 - 3.4 pp.187-211
- Procedures, Stack frame, Transferring control (opcional por ahora)
 - 3.7.1 - 3.7.2 pp.253-257
- x86-64, History, Overview, Accessing Info., hasta Arithmetic
 - 3.13 - 3.13.3 pp.301-311

■ **Ejercicios:** del Cap.3 CS:APP (Bryant/O'Hallaron)

- Probl. 3.1 - 3.5 pp.204, 208, 210-211
- Probl. 3.30 p.257
- Probl. 3.46 - 3.47 pp.305, 310

Bibliografía:

[BRY11] Cap.3

Computer Systems: A Programmer's Perspective. Bryant, O'Hallaron. Pearson, 2011

Signatura ESIIT/[C.1 BRY com](#)

Programación Máquina I: Conceptos Básicos

- Historia de los procesadores y arquitecturas de Intel
- Lenguaje C, ensamblador, código máquina
- Conceptos básicos asm: Registros, operandos, move
- Intro a x86-64

3

Procesadores Intel x86

- Dominan totalmente mercado portátil/sobremesa/servidor
- Diseño evolutivo
 - Compatible ascendentemente hasta el 8086, introducido en 1978
 - Va añadiendo características conforme pasa el tiempo
- Computador con repertorio instrucciones complejo (CISC)
 - Muchas instrucciones diferentes, con muchos formatos distintos
 - Pero sólo un pequeño subconjunto aparece en programas Linux
 - Difícil igualar prestaciones Computadores Repertorio Instr. Reducido (RISC)
 - Sin embargo, Intel lo ha conseguido
 - En lo que a velocidad se refiere. No tanto en (bajo) consumo.

4

Grado Informática, 2º Curso

Estructura de Computadores

Evolución Intel x86: Hitos significativos

Nombre	Fecha	Transistores	MHz
■ 8086	1978	29K	5-10
<ul style="list-style-type: none">Primer procesador Intel 16-bit. Base para el IBM PC & MS-DOSEspacio direccionamiento 1MB			
■ 386	1985	275K	16-33
<ul style="list-style-type: none">Primer procesador Intel 32-bit de la familia IA32 (x86)Agregó "direccionamiento plano" ("flat addressing")Capaz de arrancar UnixLinux/gcc 32-bit no usa instrucciones introducidas en modelos posteriores			
■ Pentium 4F	2004	125M	2800-3800
<ul style="list-style-type: none">Primer procesador 64-bit Intel, familia Intel 64 (EM64t, x86-64)			
■ Core i7	2008	731M	2667-3333
<ul style="list-style-type: none">4 cores, hyperthreading (2 vías)			

5

Grado Informática, 2º Curso

Estructura de Computadores

Procesadores Intel x86: Visión General

Arquitecturas

"X86-16"

X86 / IA32

MMX

SSE

SSE2

SSE3

X86-64 / EM64t

SSE4

Procesadores

8086

286

386

486

Pentium

Pentium MMX

Pentium III

Pentium 4

Pentium 4E

Pentium 4F

Core 2 Duo

Core i7

tiempo

IA: siempre redefinido como "última Arquitectura Intel"

6

Grado Informática, 2º Curso

Estructura de Computadores

Procesadores Intel x86, cont.

■ Evolución de las máquinas

■ 386	1985	0.3M
■ Pentium	1993	3.1M
■ Pentium/MMX	1997	4.5M
■ PentiumPro	1995	6.5M
■ Pentium III	1999	8.2M
■ Pentium 4	2001	42M
■ Core 2 Duo	2006	291M
■ Core i7	2008	731M

■ Características añadidas

- Instrucciones de soporte para operación multimedia
 - Operaciones paralelas con datos 1B, 2B y 4-byte, tanto enteros & p. flot.
- Instrucciones para posibilitar operaciones condicionales más eficientes

■ Evolución Linux/GCC

- 2 etapas principales: 1) soporte 32-bit 386. 2) soporte 64-bit x86-64

Microfotografía de un dado Core i7

7

Grado Informática, 2º Curso

Estructura de Computadores

Más Información

■ Procesadores Intel ([Wikipedia](#))

■ [Microarquitecturas](#) Intel

List_of_Intel_microprocessors

ark.intel.com

List_of_Intel_CPU_microarchitectures

Wikipedia EMT64 / EM64T / x86_64

8

Grado Informática, 2º Curso

4

Grado Informática, 2º Curso

Estructura de Computadores

Nueva especie: ia64, IPF*, Itanium,...

Nombre	Fecha	Transistores
■ Itanium	2001	10M
<ul style="list-style-type: none">▪ Primer intento de arquitectura 64-bit: inicialmente denominada IA64▪ Repertorio instr. radicalmente nuevo, diseñado para altas prestaciones▪ Puede ejecutar programas IA32 anteriores<ul style="list-style-type: none">▪ “motor x86 a bordo”▪ Proyecto conjunto con Hewlett-Packard		
■ Itanium 2	2002	221M
<ul style="list-style-type: none">▪ Gran aumento de prestaciones		
■ Itanium 2 Dual-Core	2006	1.7B
<ul style="list-style-type: none">■ Itanium no ha cuajado en el mercado<ul style="list-style-type: none">▪ Sin compatibilidad ascendente, sin apoyo buen compilador▪ Pentium 4 salió “demasiado bueno”		

* “IPF” = Itanium Processor Family 9

Grado Informática, 2º Curso

Estructura de Computadores

Clones x86: Advanced Micro Devices (AMD)

- Históricamente
 - AMD ha ido siguiendo a Intel en todo
 - CPUs un poco más lentas, mucho más baratas
- Y entonces
 - Reclutaron los mejores diseñadores de circuitos de Digital Equipment Corp. y otras compañías con tendencia descendente
 - Construyeron el Opteron: duro competidor para el Pentium 4
 - Desarrollaron x86-64, su propia extensión a 64 bits

10

Los 64-Bit de Intel

- **Intel intentó un cambio radical de IA32 a IA64**
 - Arquitectura totalmente diferente (Itanium)
 - Ejecuta código IA32 sólo como herencia*
 - Prestaciones decepcionantes
- **AMD intervino con una Solución Evolutiva**
 - x86-64 (ahora llamado “AMD64”)
- **Intel se sintió obligado a concentrarse en IA64**
 - Difícil admitir error, o admitir que AMD es mejor
- **2004: Intel anuncia extensión EM64T de la IA32**
 - Extended Memory 64-bit Technology
 - Casi idéntica a x86-64
- **Todos los procesadores x86 salvo gama baja soportan x86-64**
 - Pero gran cantidad de código se ejecuta aún en modo 32-bits

* “legacy” = herencia de características 11

Nosotros veremos:

- **IA32**
 - El x86 tradicional
- **x86-64/EM64T**
 - El “nuevo” estándar
 - Comentaremos brevemente las diferencias
- **Presentación**
 - El libro presenta IA32 en las Secciones 3.1—3.12
 - Cubre x86-64 en 3.13
 - Nosotros los veremos a la vez
 - Las prácticas se concentran en IA32

12

Programación Máquina I: Conceptos Básicos

- Historia de los procesadores y arquitecturas de Intel
- Lenguaje C, ensamblador, código máquina
- Conceptos básicos asm: Registros, operandos, move
- Intro a x86-64

13

Definiciones

- **Arquitectura:** (también arquitectura del repertorio de instrucciones: ISA) Las partes del diseño de un procesador que se necesitan entender para escribir código ensamblador.
 - Ejemplos: especificación del repertorio de instrucciones, registros.
- **Microarquitectura:** Implementación de la arquitectura.
 - Ejemplos: tamaño de las caches y frecuencia de los cores.
- Ejemplo de ISAs (Intel): x86, IA, IPF

14

Grado Informática, 2º Curso

Estructura de Computadores

Perspectiva del Programador en Ensamblador

CPU

PC

Registros

Códigos Condición

Direcciones

Datos

Instrucciones

Memoria

Código Objeto

Datos Programa

Datos SO

Pila

■ Estado visible al programador

■ PC: Contador de programa

- Dirección de la próxima instrucción
- Llamado "EIP" (IA32) ó "RIP" (x86-64)

■ Archivo de **registros**

- Datos del programa muy utilizados

■ Códigos de condición / **flags** de estado

- Almacenan información estado sobre la operación aritmética más reciente
- Usados para bifurcación condicional

■ Memoria

- Array direccionable por **bytes**
- Código, datos usuario, (algo) datos SO
- Incluye **pila**, usada para llamadas a procedimientos

15

Grado Informática, 2º Curso

Estructura de Computadores

Convertir C en Código Objeto

■ Código en ficheros `p1.c` `p2.c`

■ Compilar con el comando: `gcc -O1 p1.c p2.c -o p`

- Usar optimizaciones básicas (`-O1`)
- Poner binario resultante en fichero `p`

texto

programa C (`p1.c` `p2.c`)

↓

Compilador (`gcc -S`)

texto

programa asm (`p1.s` `p2.s`)

↓

Ensamblador (`gcc` ó `as`)

binario

programa objeto (`p1.o` `p2.o`)

↓

Enlazador (`gcc` ó `ld`)

binario

programa ejecutable (`p`)

librerías estáticas (`.a`)

↙

16

Grado Informática, 2º Curso

8

Compilar a ensamblador

Código C

```
int sum(int x, int y)
{
    int t = x+y;
    return t;
}
```

Ensamblador IA32 generado

```
sum:
    pushl %ebp
    movl %esp,%ebp
    movl 12(%ebp),%eax
    addl 8(%ebp),%eax
    popl %ebp
    ret
```

Algunos compiladores usan la instrucción "leave" *

Obtenerlo con el comando

```
/usr/local/bin/gcc -O1 -S code.c
```

Produce el fichero code.s

* LEAVE deshace ENTER previo: ESP←EBP, POP EBP, 17

Características Ensamblador: Tipos de Datos

- Datos "**enteros**" de 1, 2, ó 4 bytes
 - Valores de datos
 - Direcciones (**punteros** sin tipo)
- Datos en **punto flotante** de 4, 8, ó 10 bytes
- No hay tipos compuestos como arrays o estructuras
 - Tan sólo bytes ubicados contiguamente (uno tras otro) en memoria

Grado Informática, 2º Curso

Estructura de Computadores

Características ensamblador: Operaciones

- Realizan función **aritmética** sobre datos en registros o memoria
- **Transfieren** datos entre **memoria y registros**
 - Cargar datos de memoria a un registro
 - Almacenar datos de un registro en memoria
- **Transferencia de control**
 - Llamadas incondicionales a procedimientos / retornos desde ellos
 - Saltos condicionales

19

Grado Informática, 2º Curso

Estructura de Computadores

Código Objeto

Código para `sum`

```
0x401040 <sum>:
0x55
0x89
0xe5
0x8b
0x45
0x0c
0x03
0x45
0x08
0x5d
0xc3
```

- **11 bytes total**
- **Cada instrucción 1, 2, ó 3 bytes**
- **Empieza en direcc. 0x401040**

- **Ensamblador**
 - Traduce `.s` pasándolo a `.o`
 - Instrucciones codificadas en binario
 - Imagen casi completa del código ejecutable
 - Le faltan enlaces entre código de ficheros diferentes
- **Enlazador**
 - Resuelve referencias entre ficheros
 - Combina con libs. de tiempo ejec. estáticas*
 - P.ej., código para `malloc()`, `printf()`
 - Algunas libs. son *dinámicamente enlazadas***
 - En enlace ocurre cuando el programa empieza a ejecutarse

** "dynamically linked libraries", o también "shared libs"

* "static run-time libraries" = bibliotecas estáticas para soporte en tiempo de ejecución

20

Grado Informática, 2º Curso

Estructura de Computadores

Ejemplo de Instrucción Máquina

```
int t = x+y;
```

```
addl 8(%ebp),%eax
```

Similar a la expresión:

`x += y`

Más precisamente:

```
int eax;
int *ebp;
eax += ebp[2]
```

```
0x80483ca: 03 45 08
```

- **Código C**
 - Suma dos enteros con signo
- **Ensamblador**
 - Suma 2 enteros de 4-byte
 - Palabras “long” en jerga GCC
 - Misma instrucción, ya sea con o sin signo
 - Operandos:
 - `x`: Registro `%eax`
 - `y`: Memoria `M[%ebp+8]`
 - `t`: Registro `%eax`
 - Devolver valor func. en `%eax`
- **Código Objeto**
 - Instrucción de 3-byte
 - Almacenada en dir. `0x80483ca`

21

Grado Informática, 2º Curso

Estructura de Computadores

Desensamblando Código Objeto

Desensamblado

```
080483c4 <sum>:
80483c4: 55          push    %ebp
80483c5: 89 e5       mov     %esp,%ebp
80483c7: 8b 45 0c    mov     0xc(%ebp),%eax
80483ca: 03 45 08    add     0x8(%ebp),%eax
80483cd: 5d          pop     %ebp
80483ce: c3         ret
```

- **Desensamblador**
`objdump -d p`
 - Herramienta útil para examinar código objeto
 - Analiza el patrón de bits de series de instrucciones
 - Produce versión aproximada del código ensamblador (correspondiente)
 - Puede ejecutarse sobre el fich. a.out (ejecutable completo) ó el .o

22

Grado Informática, 2º Curso

11

Grado Informática, 2º Curso

Estructura de Computadores

Desensamblado Alternativo

Objeto

Desensamblado

0x401040:
0x55
0x89
0xe5
0x8b
0x45
0x0c
0x03
0x45
0x08
0x5d
0xc3

Dump of assembler code for function sum:

0x080483c4 <sum+0>: push %ebp
0x080483c5 <sum+1>: mov %esp,%ebp
0x080483c7 <sum+3>: mov 0xc(%ebp),%eax
0x080483ca <sum+6>: add 0x8(%ebp),%eax
0x080483cd <sum+9>: pop %ebp
0x080483ce <sum+10>: ret

■ Desde el Depurador gdb

gdb p
disassemble sum

■ Desensamblar procedimiento

x/11xb sum

■ Examinar 11 bytes a partir de sum

23

Grado Informática, 2º Curso

Estructura de Computadores

¿Qué se puede Desensamblar?

% objdump -d WINWORD.EXE

WINWORD.EXE: file format pei-i386

No symbols in "WINWORD.EXE".

Disassembly of section .text:

30001000 <.text>:
30001000: 55 push %ebp
30001001: 8b ec mov %esp,%ebp
30001003: 6a ff push \$0xffffffff
30001005: 68 90 10 00 30 push \$0x30001090
3000100a: 68 91 dc 4c 30 push \$0x304cdc91

■ Cualquier cosa que se pueda interpretar como código ejecutable

■ El desensamblador examina bytes y reconstruye el fuente asm.

24

Grado Informática, 2º Curso

12

Grado Informática, 2º Curso

Estructura de Computadores

Programación Máquina I: Conceptos Básicos

- Historia de los procesadores y arquitecturas de Intel
- Lenguaje C, ensamblador, código máquina
- Conceptos básicos asm: Registros, operandos, move
- Intro a x86-64

25

Grado Informática, 2º Curso

Estructura de Computadores

Registros Enteros (IA32)

propósito general

%eax	%ax	%ah	%al	acumulador
%ecx	%cx	%ch	%cl	contador
%edx	%dx	%dh	%dl	datos
%ebx	%bx	%bh	%bl	base
%esi	%si			índice fuente
%edi	%di			índice destino
%esp	%sp			puntero de pila
%ebp	%bp			puntero base

registros virtuales 16-bit (compatibilidad ascendente)

Motivos nombre
(mayoría obsoletos)

26

Grado Informática, 2º Curso

Estructura de Computadores

Mover Datos: IA32

■ Mover Datos

movl Source, Dest *

■ Tipo de Operandos

■ **Inmediato**: Datos enteros constantes

- Ejemplo: \$0x400, \$-533
- Como constante C, pero con prefijo '\$'
- Codificado mediante 1, 2, ó 4 bytes

■ **Registro**: Alguno de los 8 registros enteros

- Ejemplo: %eax, %edx
- Pero %esp y %ebp reservados para usos especiales
- Otros tienen usos especiales con instrucciones particulares

■ **Memoria**: 4 bytes consecutivos mem. en dirección dada por un registro

- Ejemplo más sencillo: (%eax)
- Hay otros diversos "modos de direccionamiento"

%eax

%ecx

%edx

%ebx

%esi

%edi

%esp

%ebp

* "source/destination" = fuente/destino 27

Grado Informática, 2º Curso

Estructura de Computadores

Combinaciones de Operandos movl

	Source	Dest	Src, Dest	Análogo C
movl	Imm*	Reg	movl \$0x4,%eax	temp = 0x4;
		Mem	movl \$-147,(%eax)	*p = -147;
	Reg	Reg	movl %eax,%edx	temp2 = temp1;
		Mem	movl %eax,(%edx)	*p = temp;
	Mem	Reg	movl (%eax),%edx	temp = *p;

Ver resto instrucciones transferencia (incluyendo pila) en el libro

No se puede transferir Mem-Mem con sólo una instrucción

* "Immediate" = Inmediato 28

Grado Informática, 2º Curso

14

Grado Informática, 2º Curso

Estructura de Computadores

Modos Direccionamiento a memoria sencillos

■ Indirecto reg. (R)

Mem[Reg[R]]

■ El registro R indica la dirección de memoria

```
movl (%ecx), %eax
```

■ Desplazamiento D(R)

Mem[Reg[R]+D]

■ El registro R indica el inicio de una región de memoria

■ La constante de desplazamiento D indica el *offset**

```
movl 8(%ebp), %edx
```

* "offset" = compensación, desbalance, para nosotros "desplazamiento"

29

Grado Informática, 2º Curso

Estructura de Computadores

Usando los Modos Direccionamiento sencillos

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

```
swap:
    pushl %ebp
    movl  %esp,%ebp
    pushl %ebx
    } Ajuste Inicial

    movl  8(%ebp), %edx
    movl  12(%ebp), %ecx
    movl  (%edx), %ebx
    movl  (%ecx), %eax
    movl  %eax, (%edx)
    movl  %ebx, (%ecx)
    } Cuerpo

    popl  %ebx
    popl  %ebp
    ret
    } Fin
```

30

Grado Informática, 2º Curso

Estructura de Computadores

Usando los Modos Direccionamiento sencillos

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

```
swap:
    pushl %ebp
    movl  %esp,%ebp
    pushl %ebx

    movl  8(%ebp), %edx
    movl  12(%ebp), %ecx
    movl  (%edx), %ebx
    movl  (%ecx), %eax
    movl  %eax, (%edx)
    movl  %ebx, (%ecx)

    popl  %ebx
    popl  %ebp
    ret
```

Ajuste Inicial

Cuerpo

Fin

31

Grado Informática, 2º Curso

Estructura de Computadores

Comprendiendo swap*

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

Registro	Valor
%edx	xp
%ecx	yp
%ebx	t0
%eax	t1

```
movl  8(%ebp), %edx  # edx = xp
movl  12(%ebp), %ecx # ecx = yp
movl  (%edx), %ebx   # ebx = *xp (t0)
movl  (%ecx), %eax   # eax = *yp (t1)
movl  %eax, (%edx)   # *xp = t1
movl  %ebx, (%ecx)   # *yp = t0
```

Offset

•

•

•

Pila (en memoria)

12

yp

8

xp

4

Dir.retorno

0

Ant.%ebp

← %ebp

-4

Ant.%ebx

← %esp

* "swap" = intercambio 32

Grado Informática, 2º Curso

Estructura de Computadores

Comprendiendo swap

%eax	
%edx	
%ecx	
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104

Offset

yp12

xp8

4

%ebp → 0

-4

123

456

0x120

0x124

Dir.retorno

Dirección

0x124

0x120

0x11c

0x118

0x114

0x110

0x10c

0x108

0x104

0x100

movl8(%ebp), %edx# edx = xp

movl12(%ebp), %ecx# ecx = yp

movl(%edx), %ebx# ebx = *xp (t0)

movl(%ecx), %eax# eax = *yp (t1)

movl%eax, (%edx)# *xp = t1

movl%ebx, (%ecx)# *yp = t0

33

Grado Informática, 2º Curso

Estructura de Computadores

Comprendiendo swap

%eax	
%edx	0x124
%ecx	
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104

Offset

yp12

xp8

4

%ebp → 0

-4

123

456

0x120

0x124

Dir.retorno

Dirección

0x124

0x120

0x11c

0x118

0x114

0x110

0x10c

0x108

0x104

0x100

movl8(%ebp), %edx# edx = xp

movl12(%ebp), %ecx# ecx = yp

movl(%edx), %ebx# ebx = *xp (t0)

movl(%ecx), %eax# eax = *yp (t1)

movl%eax, (%edx)# *xp = t1

movl%ebx, (%ecx)# *yp = t0

34

Grado Informática, 2º Curso

Estructura de Computadores

Comprendiendo swap

%eax	
%edx	0x124
%ecx	0x120
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104

Offset

yp12

xp8

4

%ebp → 0

-4

123

456

0x120

0x124

Dir.retorno

Dirección

0x124

0x120

0x11c

0x118

0x114

0x110

0x10c

0x108

0x104

0x100

movl 8(%ebp), %edx # edx = xp

movl 12(%ebp), %ecx # ecx = yp

movl (%edx), %ebx # ebx = *xp (t0)

movl (%ecx), %eax # eax = *yp (t1)

movl %eax, (%edx) # *xp = t1

movl %ebx, (%ecx) # *yp = t0

35

Grado Informática, 2º Curso

Estructura de Computadores

Comprendiendo swap

%eax	
%edx	0x124
%ecx	0x120
%ebx	123
%esi	
%edi	
%esp	
%ebp	0x104

Offset

yp12

xp8

4

%ebp → 0

-4

123

456

0x120

0x124

Dir.retorno

Dirección

0x124

0x120

0x11c

0x118

0x114

0x110

0x10c

0x108

0x104

0x100

movl 8(%ebp), %edx # edx = xp

movl 12(%ebp), %ecx # ecx = yp

movl (%edx), %ebx # ebx = *xp (t0)

movl (%ecx), %eax # eax = *yp (t1)

movl %eax, (%edx) # *xp = t1

movl %ebx, (%ecx) # *yp = t0

36

Grado Informática, 2º Curso

Estructura de Computadores

Comprendiendo swap

%eax	456
%edx	0x124
%ecx	0x120
%ebx	123
%esi	
%edi	
%esp	
%ebp	0x104

Offset

yp12

xp8

4

%ebp → 0

-4

123

456

0x120

0x124

Dir.retorno

Dirección

0x124

0x120

0x11c

0x118

0x114

0x110

0x10c

0x108

0x104

0x100

movl 8(%ebp), %edx # edx = xp

movl 12(%ebp), %ecx # ecx = yp

movl (%edx), %ebx # ebx = *xp (t0)

movl (%ecx), %eax # eax = *yp (t1)

movl %eax, (%edx) # *xp = t1

movl %ebx, (%ecx) # *yp = t0

37

Grado Informática, 2º Curso

Estructura de Computadores

Comprendiendo swap

%eax	456
%edx	0x124
%ecx	0x120
%ebx	123
%esi	
%edi	
%esp	
%ebp	0x104

Offset

yp12

xp8

4

%ebp → 0

-4

456

456

0x120

0x124

Dir.retorno

Dirección

0x124

0x120

0x11c

0x118

0x114

0x110

0x10c

0x108

0x104

0x100

movl 8(%ebp), %edx # edx = xp

movl 12(%ebp), %ecx # ecx = yp

movl (%edx), %ebx # ebx = *xp (t0)

movl (%ecx), %eax # eax = *yp (t1)

movl %eax, (%edx) # *xp = t1

movl %ebx, (%ecx) # *yp = t0

38

Grado Informática, 2º Curso

Estructura de Computadores

Comprendiendo swap

%eax	456
%edx	0x124
%ecx	0x120
%ebx	123
%esi	
%edi	
%esp	
%ebp	0x104

Offset

yp12

xp8

4

%ebp → 0

-4

456

123

0x120

0x124

Dir.retorno

Dirección

0x124

0x120

0x11c

0x118

0x114

0x110

0x10c

0x108

0x104

0x100

movl8(%ebp), %edx# edx = xp

movl12(%ebp), %ecx# ecx = yp

movl(%edx), %ebx# ebx = *xp (t0)

movl(%ecx), %eax# eax = *yp (t1)

movl%eax, (%edx)# *xp = t1

movl%ebx, (%ecx)# *yp = t0

39

Grado Informática, 2º Curso

Estructura de Computadores

Modos Direccinamiento a memoria completos

■ Forma más general

D(Rb,Ri,S)

Mem[Reg[Rb]+S*Reg[Ri]+ D]

▪ D: “Desplazamiento” constante 1, 2, ó 4 bytes

▪ Rb: Registro base: Cualquiera de los 8 registros enteros

▪ Ri: Registro índice: Cualquiera, excepto %esp

- Tampoco es probable que se use %ebp

▪ S: Factor de escala: 1, 2, 4, ú 8 (¿por qué esos números?)

■ Casos Especiales

(Rb,Ri)

Mem[Reg[Rb]+Reg[Ri]]

D(Rb,Ri)

Mem[Reg[Rb]+Reg[Ri]+D]

(Rb,Ri,S)

Mem[Reg[Rb]+S*Reg[Ri]]

40

Grado Informática, 2º Curso

Estructura de Computadores

Programación Máquina I: Conceptos Básicos

- Historia de los procesadores y arquitecturas de Intel
- Lenguaje C, ensamblador, código máquina
- Conceptos básicos asm: Registros, operandos, move
- Intro a x86-64

41

Grado Informática, 2º Curso

Estructura de Computadores

Representación Datos: IA32 + x86-64

- Tamaño de Objetos C (en Bytes)

<i>Tipo de Datos C</i>	<i>Normal 32-bit</i>	<i>Intel IA32</i>	<i>x86-64</i>
▪ unsigned	4	4	4
▪ int	4	4	4
▪ long int	4	4	8
▪ char	1	1	1
▪ short	2	2	2
▪ float	4	4	4
▪ double	8	8	8
▪ long double	8	10/12	16
▪ char *	4	4	8

– o cualquier otro puntero

42

Grado Informática, 2º Curso

Estructura de Computadores

Registros Enteros x86-64

%rax	%eax	%r8	%r8d
%rbx	%ebx	%r9	%r9d
%rcx	%ecx	%r10	%r10d
%rdx	%edx	%r11	%r11d
%rsi	%esi	%r12	%r12d
%rdi	%edi	%r13	%r13d
%rsp	%esp	%r14	%r14d
%rbp	%ebp	%r15	%r15d

- Extiende los registros anteriores. Añade 8 nuevos.
- Hace %ebp / %rbp de propósito general

43

Grado Informática, 2º Curso

Estructura de Computadores

Instrucciones

- palabra Long l (4 Bytes) ↔ palabra Quad q (8 Bytes)
- Nuevas instrucciones:
 - movl → movq
 - addl → addq
 - sall → salq
 - etc.
- instrucciones 32-bit que generan resultados 32-bit
 - Ajustan los bits más significativos del registro destino a 0
 - Ejemplo: addl

44

Grado Informática, 2º Curso

Estructura de Computadores

Código de 32-bit para swap

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

swap:

pushl %ebp
movl %esp,%ebp
pushl %ebx

Ajuste Inicial

movl 8(%ebp), %edx
movl 12(%ebp), %ecx
movl (%edx), %ebx
movl (%ecx), %eax
movl %eax, (%edx)
movl %ebx, (%ecx)

Cuerpo

popl %ebx
popl %ebp
ret

Fin

45

Grado Informática, 2º Curso

Estructura de Computadores

Código de 64-bit para swap

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

swap:

movl (%rdi), %edx
movl (%rsi), %eax
movl %eax, (%rdi)
movl %edx, (%rsi)

Ajuste Inicial

Cuerpo

ret

Fin

- Los operandos se pasan en registros (*¿por qué ventajoso?*)
 - Primero (xp) en %rdi, segundo (yp) en %rsi
 - Punteros 64-bit
- No requiere operaciones en pila
- Datos 32-bit
 - Datos mantenidos en registros %eax y %edx
 - operación movl (no q)

46

Grado Informática, 2º Curso

23

Código de 64-bit para swap long int

swap_1:

```
void swap(long *xp, long *yp)
{
    long t0 = *xp;
    long t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

```
movq    (%rdi), %rdx
movq    (%rsi), %rax
movq    %rax, (%rdi)
movq    %rdx, (%rsi)
```

```
ret
```

} Ajuste Inicial

} Cuerpo

} Fin

■ Datos 64-bit

- Datos mantenidos en registros %rax y %rdx
- operación movq (no l)
 - “q” viene de quad-word

47

Programación Máquina I: Resumen

- **Historia de los procesadores y arquitecturas de Intel**
 - Diseño evolutivo lleva a muchas peculiaridades y artefactos
- **Lenguaje C, ensamblador, código máquina**
 - El compilador debe transformar sentencias, expresiones, procedimientos, en secuencias de instrucciones de bajo nivel
- **Conceptos básicos asm: Registros, operandos, move**
 - Las instrucciones move x86 cubren amplia gama de formas de movimiento de datos (transferencia)
- **Intro a x86-64**
 - Un cambio importante respecto al estilo de código visto en IA32

48

Guía de trabajo autónomo (4h/s)

- **Estudio:** del Cap.3 CS:APP (Bryant/O'Hallaron)
 - Historical perspective, Program Encodings, Data Formats, Accessing Info.
 - 3.1 - 3.4 pp.187-211
 - Probl. 3.1 - 3.5 pp.204, 208, 210-211
 - Procedures, Stack frame, Transferring control (opcional por ahora)
 - 3.7.1 - 3.7.2 pp.253-257
 - Probl. 3.30 p.257
 - x86-64, History, Overview, Accessing Info., hasta Arithmetic
 - 3.13 - 3.13.3 pp.301-311
 - Probl. 3.46 - 3.47 pp.305, 310

Bibliografía:

[BRY11] Cap.3

Computer Systems: A Programmer's Perspective. Bryant, O'Hallaron. Pearson, 2011

Signatura ESIIT/[C.1 BRY com](#)