TEORÍA

C declaration	Intel data type	Assembly-code suffix	Size (bytes)
char	Byte	**b**	1
short	Word	**w**	2
int	Double word	**l**	4
long	Quad word	**q**	8
char *	Quad word	**q**	8
float	Single precision	**s**	4
double	Double precision	**l**	8

Figure 3.1 **Sizes of C data types in x86-64.** With a 64-bit machine, pointers are 8 long.

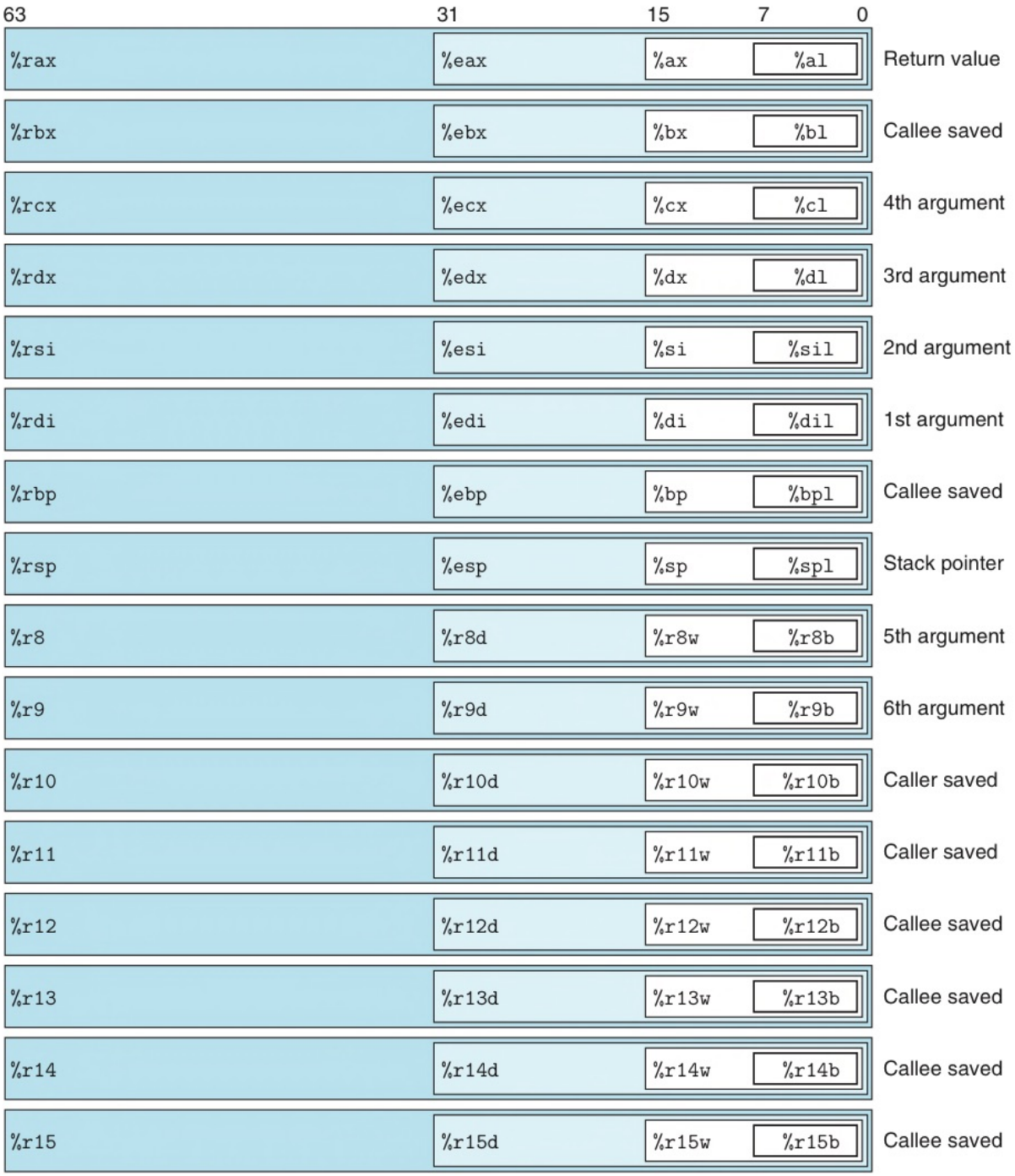

Figure 3.2 **Integer registers.** The low-order portions of all 16 registers can be accessed as byte, word (16-bit), double word (32-bit), and quad word (64-bit) quantities.

Type	Form	Operand value	Name
Immediate	$ *Imm*	*Imm*	Immediate
Register	$\mathbf{r}_a$	R[$\mathbf{r}_a$]	Register
Memory	*Imm*	M[*Imm*]	Absolute
Memory	($\mathbf{r}_a$)	M[R[$\mathbf{r}_a$]]	Indirect
Memory	*Imm*($\mathbf{r}_b$)	M[*Imm* + R[$\mathbf{r}_b$]]	Base + displacement
Memory	($\mathbf{r}_b$,$\mathbf{r}_i$)	M[R[$\mathbf{r}_b$] + R[$\mathbf{r}_i$]]	Indexed
Memory	*Imm*($\mathbf{r}_b$,$\mathbf{r}_i$)	M[*Imm* + R[$\mathbf{r}_b$] + R[$\mathbf{r}_i$]]	Indexed
Memory	(,$\mathbf{r}_i$,s)	M[R[$\mathbf{r}_i$] · s]	Scaled indexed
Memory	*Imm*(,$\mathbf{r}_i$,s)	M[*Imm* + R[$\mathbf{r}_i$] · s]	Scaled indexed
Memory	($\mathbf{r}_b$,$\mathbf{r}_i$,s)	M[R[$\mathbf{r}_b$] + R[$\mathbf{r}_i$] · s]	Scaled indexed
Memory	*Imm*($\mathbf{r}_b$,$\mathbf{r}_i$,s)	M[*Imm* + R[$\mathbf{r}_b$] + R[$\mathbf{r}_i$] · s]	Scaled indexed

Figure 3.3 **Operand forms.** Operands can denote immediate (constant) values, register values, or values from memory. The scaling factor s must be either 1, 2, 4, or 8.

Instruction	Effect	Description
pushq S	R[%**rsp**] ← R[%**rsp**] − 8; M[R[%**rsp**]] ← S	Push quad word
popq D	D ← M[R[%**rsp**]]; R[%**rsp**] ← R[%**rsp**] + 8	Pop quad word

Figure 3.8 **Push and pop instructions.**

Instruction		Effect	Description
leaq	S, D	D ← &S	Load effective address
INC	D	D ← D+1	Increment
DEC	D	D ← D−1	Decrement
NEG	D	D ← -D	Negate
NOT	D	D ← ~D	Complement
ADD	S, D	D ← D + S	Add
SUB	S, D	D ← D − S	Subtract
IMUL	S, D	D ← D * S	Multiply
XOR	S, D	D ← D ^ S	Exclusive-or
OR	S, D	D ← D | S	Or
AND	S, D	D ← D & S	And
SAL	k, D	D ← D << k	Left shift
SHL	k, D	D ← D << k	Left shift (same as SAL)
SAR	k, D	D ← $D >>_A k$	Arithmetic right shift → Fills with the sign
SHR	k, D	D ← $D >>_L k$	Logical right shift → Fills with zeros

DECIMAL	BINARIO	HEXADECIMAL
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Figure 3.10 **Integer arithmetic operations.** The load effective address (**leaq**) instruction is commonly used to perform simple arithmetic. The remaining ones are more standard unary or binary operations. We use the notation $>>_A$ and $>>_L$ to denote arithmetic and logical right shift, respectively. Note the nonintuitive ordering of the operands with ATT-format assembly code.

■ Instrucciones de Dos Operandos:

Formato		*Operación*†	
`addq`	*Src,Dest*	Dest = Dest + Src	
`subq`	*Src,Dest*	Dest = Dest – Src	
`imulq`	*Src,Dest*	Dest = Dest * Src	
`salq`	*Src,Dest*	Dest = Dest << Src	*También llamada shlq*
`sarq`	*Src,Dest*	Dest = Dest >> Src	*Aritméticas*
`shrq`	*Src,Dest*	Dest = Dest >> Src	*Lógicas*
`xorq`	*Src,Dest*	Dest = Dest ^ Src	
`andq`	*Src,Dest*	Dest = Dest & Src	
`orq`	*Src,Dest*	Dest = Dest | Src	

One - opperand

Instruction		Effect	Description
imulq	S	R[**%rdx**]:R[**%rax**] $\leftarrow$ $S \times$ R[**%rax**]	Signed full multiply
mulq	S	R[**%rdx**]:R[**%rax**] $\leftarrow$ $S \times$ R[**%rax**]	Unsigned full multiply
cqto		R[**%rdx**]:R[**%rax**] $\leftarrow$ SignExtend(R[**%rax**])	Convert to oct word
idivq	S	R[**%rdx**] $\leftarrow$ R[**%rdx**]:R[**%rax**] mod S; R[**%rax**] $\leftarrow$ R[**%rdx**]:R[**%rax**] $\div S$	Signed divide
divq	S	R[**%rdx**] $\leftarrow$ R[**%rdx**]:R[**%rax**] mod S; R[**%rax**] $\leftarrow$ R[**%rdx**]:R[**%rax**] $\div S$	Unsigned divide

Figure 3.12 Special arithmetic operations. These operations provide full 128-bit multiplication and division, for both signed and unsigned numbers. The pair of registers **%rdx** and **%rax** are viewed as forming a single 128-bit oct word.

Instruction		Based on	Description
CMP	S_1, S_2	$S_2 - S_1$	Compare
cmpb			Compare byte
cmpw			Compare word
cmpl			Compare double word
cmpq			Compare quad word
TEST	S_1, S_2	S_1 & S_2	Test
testb			Test byte
testw			Test word
testl			Test double word
testq			Test quad word

Figure 3.13 Comparison and test instructions. These instructions set the condition codes without updating any other registers.

Instruction	Synonym	Effect	Set condition
sete *D*	**setz**	*D* ← **ZF**	Equal / zero
setne *D*	**setnz**	*D* ← **~ZF**	Not equal / not zero
sets *D*		*D* ← **SF**	Negative
setns *D*		*D* ← **~SF**	Nonnegative
setg *D*	**setnle**	*D* ← **~ (SF ^ OF) & ~ZF**	Greater (signed **>**)
setge *D*	**setnl**	*D* ← **~ (SF ^ OF)**	Greater or equal (signed **>=**)
setl *D*	**setnge**	*D* ← **SF ^ OF**	Less (signed **<**)
setle *D*	**setng**	*D* ← **(SF ^ OF) | ZF**	Less or equal (signed **<=**)
seta *D*	**setnbe**	*D* ← **~ CF & ~ZF**	Above (unsigned **>**)
setae *D*	**setnb**	*D* ← **~ CF**	Above or equal (unsigned **>=**)
setb *D*	**setnae**	*D* ← **CF**	Below (unsigned **<**)
setbe *D*	**setna**	*D* ← **CF | ZF**	Below or equal (unsigned **<=**)

Figure 3.14 The SET instructions. Each instruction sets a single byte to 0 or 1 based on some combination of the condition codes. Some instructions have "synonyms," that is, alternate names for the same machine instruction.

Instruction	Synonym	Jump condition	Description
jmp *Label*		1	Direct jump
jmp ***Operand**		1	Indirect jump
je *Label*	**jz**	**ZF**	Equal / zero
jne *Label*	**jnz**	**~ZF**	Not equal / not zero
js *Label*		**SF**	Negative
jns *Label*		**~SF**	Nonnegative
jg *Label*	**jnle**	**~(SF ^ OF) & ~ZF**	Greater (signed **>**)
jge *Label*	**jnl**	**~(SF ^ OF)**	Greater or equal (signed **>=**)
jl *Label*	**jnge**	**SF ^ OF**	Less (signed **<**)
jle *Label*	**jng**	**(SF ^ OF) | ZF**	Less or equal (signed **<=**)
ja *Label*	**jnbe**	**~CF & ~ZF**	Above (unsigned **>**)
jae *Label*	**jnb**	**~CF**	Above or equal (unsigned **>=**)
jb *Label*	**jnae**	**CF**	Below (unsigned **<**)
jbe *Label*	**jna**	**CF | ZF**	Below or equal (unsigned **<=**)

Figure 3.15 The jump instructions. These instructions jump to a labeled destination when the jump condition holds. Some instructions have "synonyms," alternate names for the same machine instruction.

Instruction		Synonym	Move condition	Description
cmove	*S*, *R*	**cmovz**	**ZF**	Equal / zero
cmovne	*S*, *R*	**cmovnz**	**~ZF**	Not equal / not zero
cmovs	*S*, *R*		**SF**	Negative
cmovns	*S*, *R*		**~SF**	Nonnegative
cmovg	*S*, *R*	**cmovnle**	**~(SF ^ OF) & ~ZF**	Greater (signed **>**)
cmovge	*S*, *R*	**cmovnl**	**~(SF ^ OF)**	Greater or equal (signed **>=**)
cmovl	*S*, *R*	**cmovnge**	**SF ^ OF**	Less (signed **<**)
cmovle	*S*, *R*	**cmovng**	**(SF ^ OF) | ZF**	Less or equal (signed **<=**)
cmova	*S*, *R*	**cmovnbe**	**~CF & ~ZF**	Above (unsigned **>**)
cmovae	*S*, *R*	**cmovnb**	**~CF**	Above or equal (Unsigned **>=**)
cmovb	*S*, *R*	**cmovnae**	**CF**	Below (unsigned **<**)
cmovbe	*S*, *R*	**cmovna**	**CF | ZF**	Below or equal (unsigned **<=**)

Figure 3.18 The conditional move instructions. These instructions copy the source value *S* to its destination *R* when the move condition holds. Some instructions have "synonyms," alternate names for the same machine instruction.

SALTA - EN - MEDIO

jump-to-middle strategy yields the goto code

```
        init-expr;
        goto test;
loop:
        body-statement
        update-expr;
test:
        t = test-expr;
        if (t)
            goto loop;
```

COPIA - TEST

while the guarded-do strategy yields

```
        init-expr;
        t = test-expr;
        if (!t)
            goto done;
loop:
        body-statement
        update-expr;
        t = test-expr;
        if (t)
            goto loop;
done:
```

PRÁCTICAS

gcc			
switch	**argumento**	**significado**	**explicación**
-c		Compile	Compila o ensambla los fuentes, pero no enlaza. Se obtiene un objeto por cada fuente. Por defecto, los objetos se llaman como el fuente.o
-S		aSsembly	Compila los fuentes pero no ensambla. Se obtiene un fuente ensamblador por cada fuente C. Por defecto, se llaman como el fuente.s
-o	fichero.ext	Output	Cambiar el nombre del fichero producido. Por defecto, ejecutable a.out, objeto fuente.o, ensamblador fuente.s
-g	1…3	debuG	Genera información de depuración (por defecto, nivel 2; Eclipse usa nivel 3)
-O	0…3, s, g		Optimizar para velocidad (0 no optimización…3 agresivo) o tamaño (s, size). Poner –O = –O1. No poner nada = –O0. Para depurar usar –Og mejor.
-m32 **-m64**		Machine	Genera código para entorno de 32/64 bits, aunque no sea el configurado por defecto.
-L	dir	LibraryDir	Añadir dir a la lista de búsqueda de librerías (preferible sin espacio: -Ldir)
-l	name	LibraryName	Enlazar contra libname.so / libname.a (preferible usar sin espacio: -lname)
-print-file-name=lib			Imprime el pathname que se usaría para lib a la hora de enlazar
-v		Verbose	Imprime los comandos ejecutados para las diversas etapas de compilación
-###			Como –v, pero no ejecuta los comandos. Es más fácil de leer.
-fno-stack-protector **-fno-asynchronous-unwind-tables** **-fno-omit-frame-pointer -no-pie** **-fno-if-conversion -fno-tree-ch** **-fno-reorder-blocks -fno-tree-ter**			Versiones recientes de gcc añaden protecciones o info.debug y dificultan leer el código ASM generado. Estos switches quitan dos de esas opciones. Versiones recientes de gcc optimizan demasiado y generan código distinto al mostrado en transparencias en clase. Estos switches permiten quitar la optimización concreta y reproducir el código explicado en clase.
as			
-g		debuG	mismo sentido
-o	fichero.o	Output	mismo sentido
--32 **--64**			mismo sentido
ld			
-L	dir	LibraryDir	mismo sentido
-l	name	LibraryName	mismo sentido
-m	elf_i386 elf_x86_64	Machine	mismo sentido
-M		printImpMap	Imprimir mapa de enlazado, posición en memoria de objetos, símbolos, etc.

Tabla 1: modificadores para gcc, as, ld

objdump fich.obj.			
switch	**argumento**	**significado**	**explicación**
-d		Disassemble	Muestra los mnemotécnicos ensamblador correspondientes a las instrucciones máquina en las secciones de código del fichero objeto
-S		Source	Intercala código fuente con desensamblado. Implica –d. Requiere compilar con –g.
-h		Headers	Resumen de las cabeceras de sección presentes
-r		Reloc	Muestra las reubicaciones
-t		table	Muestra las entradas de la tabla de símbolos (similar a nm)
-T		Table	Muestra tabla de símbolos dinámicos (similar a nm -D). Para librerías compartidas.
-j / --section=	name	Just	Seleccionar información sólo de la sección mencionada
-s / --full-contents			Mostrar contenidos completos de todas las secciones (o sólo de las indicadas con -j)
nm	fich.obj.		
-D		Dynamic	Mostrar símbolos dinámicos (p.ej. en librerías compartidas)

Tabla 2: modificadores para objdump, nm

Instrucciones aritmético-lógicas							
Instrucción		**Efecto**	**Descripción**	**Instrucción**		**Efecto**	**Descripción**
INC	D	D ← D + 1	Incrementar	NEG	D	D ← -D	Negar (aritmético)
DEC	D	D ← D - 1	Decrementar	NOT	D	D ← ~D	Complementar (lógico)
ADD	D, S	D ← D + S	Sumar	XOR	D, S	D ← D ^ S	O-exclusivo
SUB	D, S	D ← D - S	Restar	OR	D, S	D ← D | S	O lógico
ADC	D, S	D ← D+S+C	Sumar con acarreo	AND	D, S	D ← D & S	Y Lógico
SBB	D, S	D ←D-(S+C)	Restar con débito	IMUL	D, S	D ← D * S	Multiplicar
SAL	D, k	D ← D << k	Desplazamiento a izq.	RCL	D, k	D ← D $\circlearrowleft_C$ k	Rotación izq. incl. acarreo
SHL	D, k	D ← D << k	Desplazamiento a izq.	ROL	D, k	D ← D $\circlearrowleft$ k	Rotación a izquierda
SAR	D, k	D ← D$>>_A$ k	Desplaz. aritm. derecha	RCR	D, k	D ← D $\circlearrowright_C$ k	Rotación der. incl. acarreo
SHR	D, k	D ← D $>>_L$ k	Desplaz. lógico derecha	ROR	D, k	D ← D $\circlearrowright$ k	Rotación a derecha

Tabla 4: Instrucciones x86 - aritmético-lógicas

Instrucciones aritméticas especiales

Instrucción		Efecto	Descripción	Operandos / Variantes	
MUL	S	$Ac_{D:A} \leftarrow Ac_A * S$	Multiplicación sin signo	AX ← AL*r/m8	EDX:EAX ← EAX*r/m32
				DX:AX ← AX*r/m16	RDX:RAX ← RAX*r/m64
DIV	S	$Ac_A \leftarrow Ac_{D:A} / S$ $Ac_D \leftarrow Ac_{D:A} \% S$	División sin signo	AL(AH)← AX / r/m8	EAX(EDX)←EDX:EAX / r/m32
				AX(DX)←DX:AX / r/m16	RAX(RDX)←RDX:RAX / r/m64
IMUL	S	$Ac_{D:A} \leftarrow Ac_A * S$	Multiplicación con signo	1 operando: Como MUL, nbit x nbit = 2nbits	
	Dr,S	D ← D * S		2 operandos: Reg * (R/M/Inmediato), n x n = nbits	
	Dr,S,I	D ← D*S*Inm		3 operandos: Reg = Reg* (R/M/Inm), n x n = nbits	
IDIV	S	$Ac_A \leftarrow Ac_{D:A} / S$ $Ac_D \leftarrow Ac_{D:A} \% S$	División con signo	Como DIV	
CBW		$Ac_{2n} \leftarrow ExtSign(Ac_n)$	Extensión de signo acum. word-to-long-to-quad	AX ← ExtSigno(AL)	cbtw
CWDE				EAX ← ExtSigno(AX)	cwtl
CDQE				RAX ← ExtSigno(EAX)	cltq
CWD		$Ac_{D:A} \leftarrow ExtSign(Ac_A)$	Extensión de signo acum. x-to-double	DX:AX ← ExtSigno(AX)	cwtd
CDQ				EDX:EAX ← ExtSigno(EAX)	cltd
CQO				RDX:RAX ← ExtSigno(RAX)	cqto

Tabla 5: Instrucciones x86 - aritmética especial

Otras instrucciones de control de flujo, y miscelánea

Instrucción		Efecto	Descripción	Instrucción		Efecto	Descripción
CALL	label	PUSH rip rip ← label	Llamada a subrutina Intel: "near relative"	INT	v	PUSH rflags CALL ISR#v	Llamada a ISR Interrupción "Interrupción software"
call	*Ptr	(AT&T)					
CALL	PtrDst	(Intel)	Intel: "near absol. indirect"				
RET		POP rip	Retorno de subrutina	IRET		RET POP rflags	Retorno de ISR
LEAVE		RSP ← RBP POP RBP	Libera marco pila Para usar con ENTER	NOP		No-op	
CLC		C ← 0	Ajustes del flag acarreo	CLI		I ← 0	Ajustes del flag Interrupt (des)habilitar IRQs
STC		C ← 1		STI		I ← 1	
CMC		C ← ~C					

Tabla 7: Instrucciones x86 - control y miscelánea