Nivel ISA – Intel Core i7

Arquitectura y Organización de Computadoras II

Ticiano J. Torres Peralta

REV2021

Las 7 Dimensiones del ISA

Desde el punto de vista de las dimensiones del ISA, visto en clase, veremos los diseños de los ISA:

- IA-32, del Intel Core i7
- ARM v7, del SoC OMAP4430

Intel Core i7

A presar de ser un procesador moderno, el Core i7 todavía retiene compatibilidad para ejecutar programas escritos para la 8086 y la 8088, procesadores de 16 bits de los años 1970. La ISA de esta época fue conocida como la x86.

El procesador Intel 80386 (1985) fuel el primero de 32 bits y desde entonces prácticamente tuvo la misma arquitectura llamada la IA-32.

La mayor diferencia entre el Core i7 moderno y el 80386 en términos de su arquitectura, es que ademas de la IA-32, el Core i7 también implementa otras arquitecturas, como la MMX, SSE, SSE2 y la x86-64 (la versión de 64 bits de la IA-32).

Intel Core i7

El i7 tiene tres formas de operar:

- Real Mode: Toda característica desde el 8086 en adelante están deshabilitadas. Funciona como un 8086 muy caro.
- Virtual 8086 Mode: El sistema operativo controla la maquina entera. Habilita la forma de correr programas en una forma aislada y protegida.
- Protected Mode: Aquí, el i7 se comporta como un i7 y habilita todas las características del mismo. Permite cuatro niveles de privilegios manejados usando el PSW,
  - Level 0: Kernel Mode
  - Level 1 y 2: Device Drivers
  - Level 3: User Mode

Modelo de Memoria

El i7 tiene un espacio de direccionamiento teórico enorme, con 16,384 segmentos de 32 bits, que direccionan de 0 a 2^{32} – 1 bytes. Las palabras son de 32 bits y guardadas en formato little-endian.

Estas direcciones virtuales de 48 bits son formadas por un numero de segmento de 16 bits con un segment offset de 32 bits. Estas direcciones virtuales después son traducidas por el MMU (Memory Management System) a direcciones físicas de 36 bits, permitiendo un total de 36 GB de memoria principal.

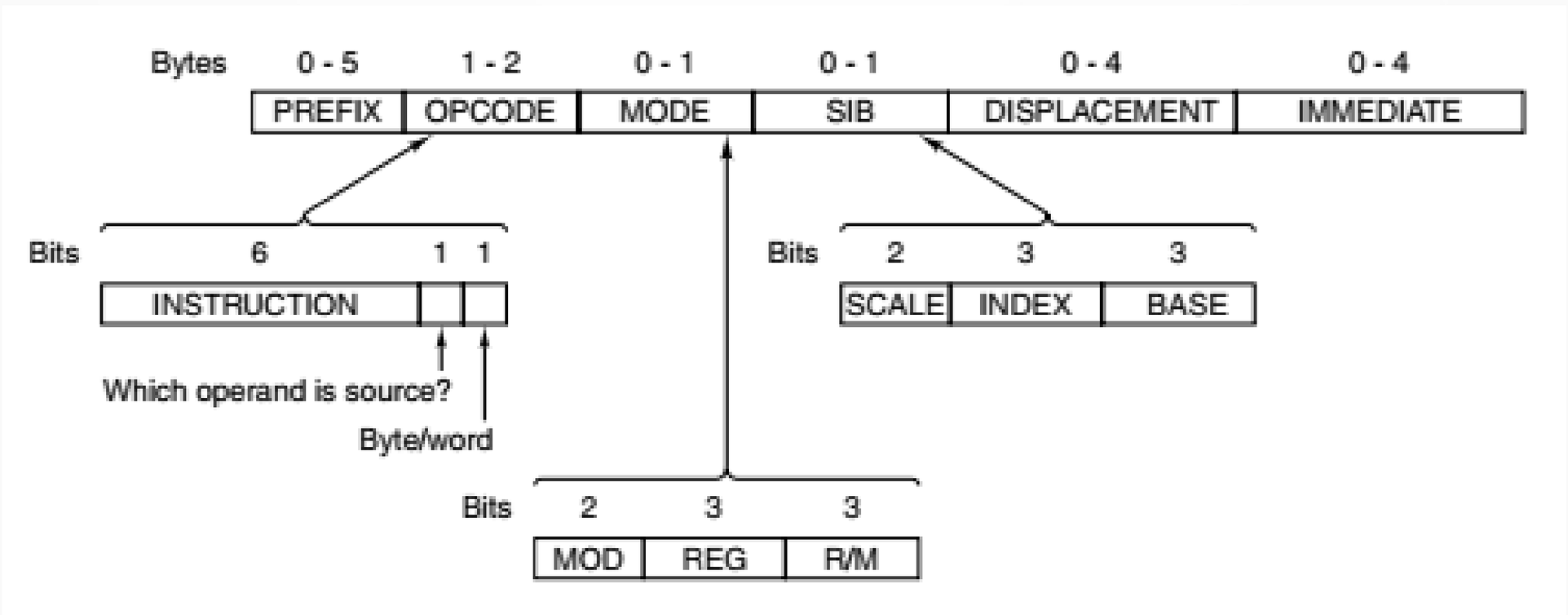
La mayoría de los programas solo ven un segmento, y pueden efectivamente direccionar un espacio de 2^{32} bytes.

Soporta hasta 3 canales de memoria, cada uno pudiendo hacer operaciones con memoria de forma paralela a los otros, con un total de alrededor de 25 GB/sec.

Recomienda que el usuario/compilador haga un esfuerzo de alinear la memoria para mejorar rendimiento. Por ejemplo, en múltiples de 4 bytes para direcciones de operandos, que un objeto no cruce lineas de Cache, o que un objeto estén alineados con los tamaños de registros para instrucciones SIMD.

Formato de Instrucción

El formato de instrucción de la i7 es irregular y complejo, soportando hasta 6 campos variables, 5 lo cuales son opcionales.



Formato de Instrucción

En arquitecturas mas antiguas, el opcode era de 1 byte con el concepto de un prefix para modificar el funcionamiento de la instrucción. Esto es parecido a la instrucción WIDE de la IJVM, que por ejemplo, indica que el operando es de 2 bytes y no de 1 byte.

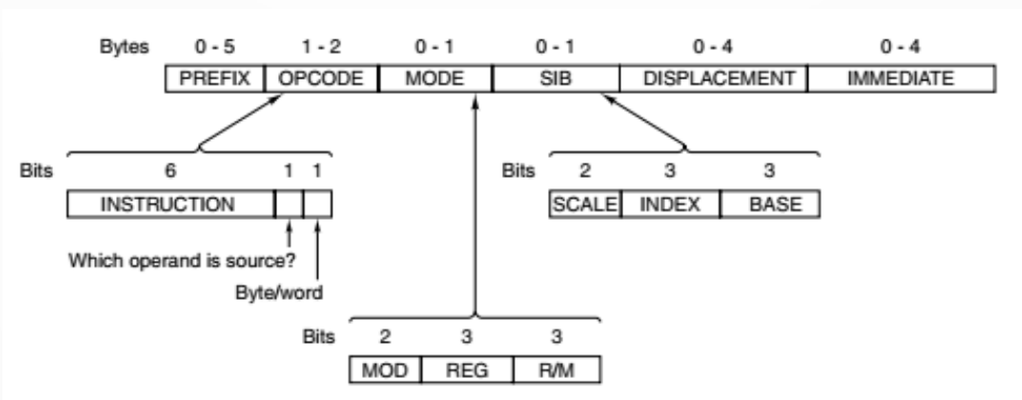
En algún momento, la arquitectura se quedó sin opcodes disponibles y se usó el opcode xFF para indicar que el siguiente byte también es parte del opcode (osea, para indicar un opcode de 2 bytes).

Los bits individuales del opcode no contienen información importante, requiriendo que todo el opcode sea decodificado para determinar que instrucción será ejecutada y que longitud tendrá la instrucción. Esto dificulta el rendimiento de la maquina, porque una decodificación completa es necesaria previo a saber que tiene que hacer.

Formato de Instrucción

El campo MODE define el modo de direccionamiento. Estos 8 bits están divididos en tres grupos. Aveces los primeros 3 bits (R/M) son utilizados como una extensión para un opcode de 1 byte, permitiendo uno de 11 bits.

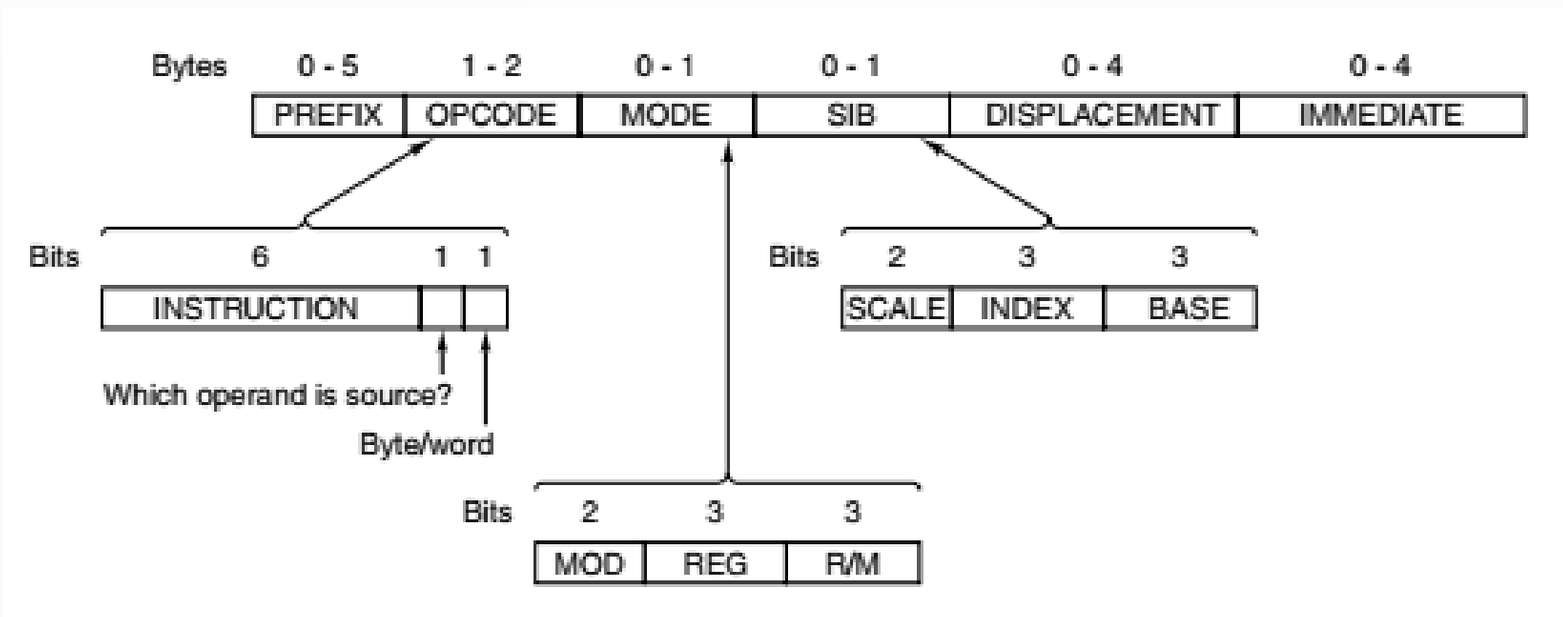
- MOD usa 2 bits define si la instrucción es R-R, R-M, o M-R.
- REG usa 3 bits
- R/M usa 3 bits



Lógicamente todos los registros generales deberían estar disponibles pero las reglas de codificación limitan algunas combinaciones. Ademas, algunos modos requieren un byte extra, el campo SIB (Scale, Index, Base), para definir información adicional.

Formato de Instrucción

Finalmente, hasta 4 bytes están disponibles para indicar una direccional de memoria (DISPLACEMENT), o un constante (IMMEDIATE).



Modos de Direccionamiento

Los modos de direccionamiento son bastante irregulares y depende si estamos usado instrucciones de 16, 32, y 64 bits. Vamos a concentrarnos en los modos de la IA-32 de 32 bits.

Los modos soportados son:

- Immediate
- Direct
- Register
- Register Indirect
- Displacement (Tanenbaum lo llama Indexed)
- Special Mode (Indexed-Base, para elementos de un arreglo)

Una observación particularmente problemática es que no todo los modos aplican a todas las instrucciones y no todos los registros están disponibles en todos los modos. Esto dificulta la tarea de los compiladores para producir buen código.

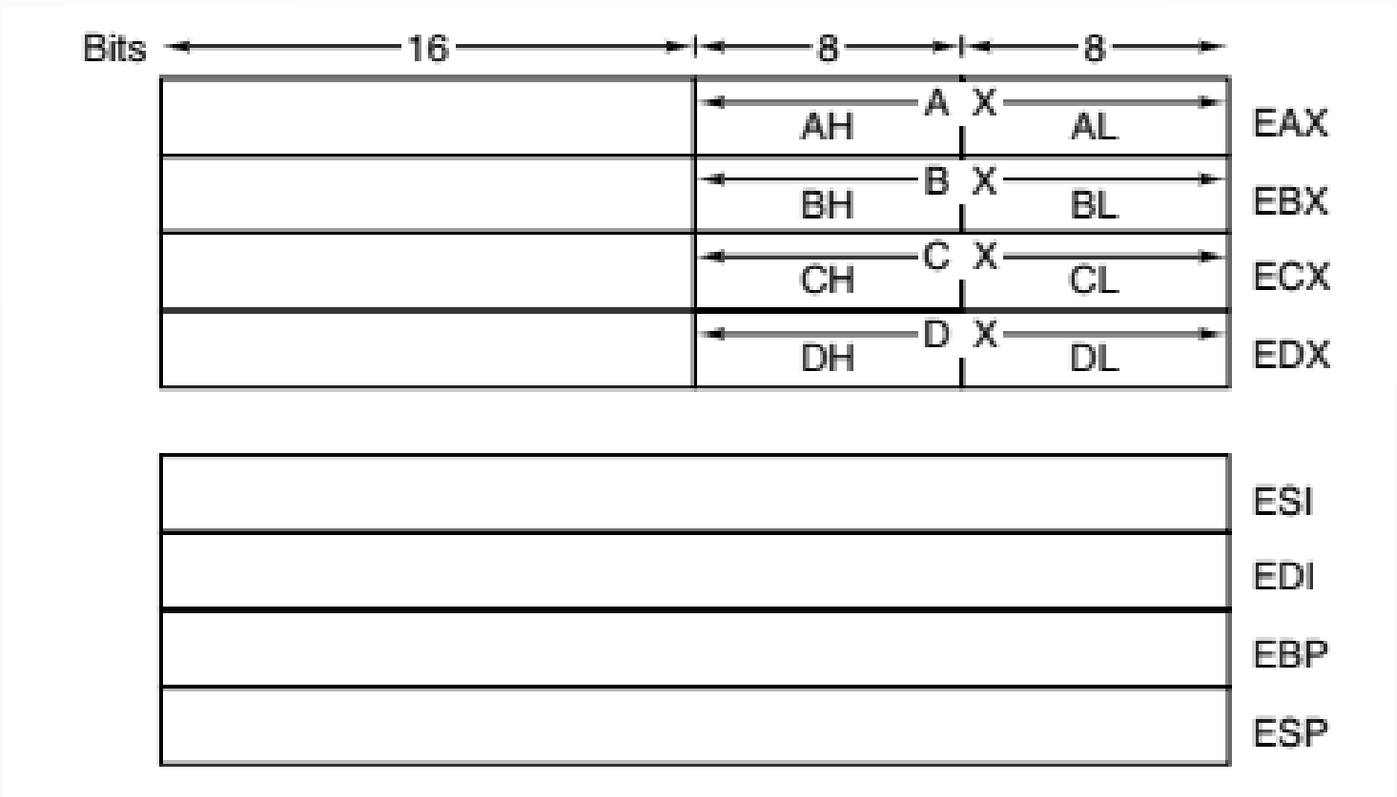
Modos de Direccionamiento

Como visto en el formato de instrucción, el campo MODE controla los modos de direccionamiento. REG siempre especifica un registro, mientras que la combinación de MOD y R/M especifica el otro operando en combinación con el modo de direccionamiento.

- MOD 00 indica Register Indirect y Direct
- MOD 01 y 10 indican Displacement.
- MOD 11 indica Register, de 32 bits para instrucciones de palabras y 8 bits para instrucciones de bytes.

Formato de Instrucción

REG puede direccionar los siguientes:



Formato de Instrucción

- 32 combinaciones de MOD y R/M:

	MOD			
R/M	**00**	**01**	**10**	**11**
000	M[EAX]	M[EAX + OFFSET8]	M[EAX + OFFSET32]	EAX or AL
001	M[ECX]	M[ECX + OFFSET8]	M[ECX + OFFSET32]	ECX or CL
010	M[EDX]	M[EDX + OFFSET8]	M[EDX + OFFSET32]	EDX or DL
011	M[EBX]	M[EBX + OFFSET8]	M[EBX + OFFSET32]	EBX or BL
100	SIB	SIB with OFFSET8	SIB with OFFSET32	ESP or AH
101	Direct	M[EBP + OFFSET8]	M[EBP + OFFSET32]	EBP or CH
110	M[ESI]	M[ESI + OFFSET8]	M[ESI + OFFSET32]	ESI or DH
111	M[EDI]	M[EDI + OFFSET8]	M[EDI + OFFSET32]	EDI or BH

Formato de Instrucción

Aveces siguiente MODE, esta SIB. Este especifica un factor de escala (S), y dos registros (I y B).

Cuando presente, la direccion effectiva es computada con lo siguiente:

- S indica un factor de 1, 2, 4 o 8
- El registro indice (I) es multiplicado por el factor de S.
- El resultado es sumado al registro base (B).
- Y finalmente sumándolo, si es necesario, a un desplazamiento de 8 bits o 32 bits.

Formato de Instrucción

A[] es un arreglo de enteros (4 Bytes cada entero).

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

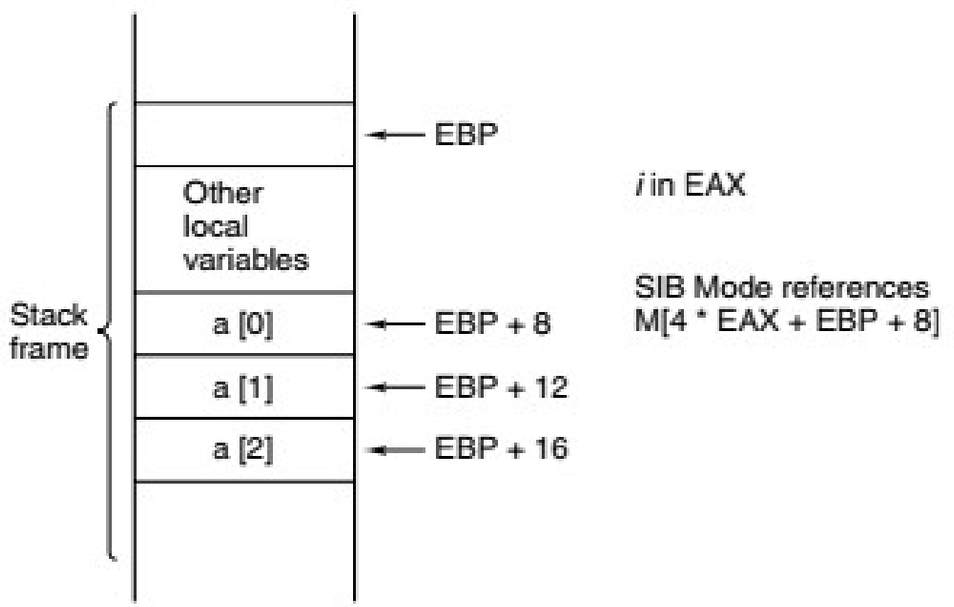


Figure 5-27. Access to *a*[*i*].

Tipos de Datos

- El IA-32 del Core i7 soporta:
  - Enteros signados (codificado en complemento-a-dos).
  - Enteros no-signados
  - BDC
  - Punto flotante (codificado en IEEE 754)

Type	8 Bits	16 Bits	32 Bits	64 Bits
Signed integer	×	×	×	× (64-bit)
Unsigned integer	×	×	×	× (64-bit)
Binary coded decimal integer	×			
Floating point			×	×

También tiene buenas capacidades para manipular caracteres ASCII de 8 bits, con instrucciones especiales para este tipo de dato. Soporta cadenas de caracteres de tamaño conocido y también de tamaño desconocido marcadas con un carácter de terminación.

Tipos de Instrucciones

Las instrucciones del IA-32 del Core i7 son una mezcla de instrucciones que tienen sentido cuando se trabaja con 32 bits y otras para retener compatibilidad en reversa (8 y 16 bits).

Las instrucciones referencian uno o dos operandos. Cuando hay dos presente, con uno siempre apunta a un registro y el otro a un registro o memoria. Como usa un máximo de dos operando, el resultado de operaciones diádicas siempre remplaza uno de los operandos (DST).

Muchas instrucciones usan un prefijo (PREFIX) para modificar su comportamiento. Por ejemplo:

- REP causa que la instrucción se repita hasta que el registro ECX llegue a 0. ECX es decrementado automáticamente en cada repetición.
- REPZ y REPNZ causa que la instrucción se repita hasta que el código de condición Z se afirme o no, respectivamente.
- LOCK causa una reserva del bus para toda la instrucción.
- Algunos prefijos fuerzan a la instrucción a funcionar en modo 8, 16, o 32 bits.
- Algunos prefijos fuerzan a la instrucción a usar de forma especifica algunos segmentos de memoria.

Siguiente muestra una pequeña selección de instrucciones.

Tipos de Instrucciones

Moves

MOV DST,SRC	Move SRC to DST
PUSH SRC	Push SRC onto the stack
POP DST	Pop a word from the stack to DST
XCHG DS1,DS2	Exchange DS1 and DS2
LEA DST,SRC	Load effective addr of SRC into DST
CMOVcc DST,SRC	Conditional move

Transfer of control

JMP ADDR	Jump to ADDR
Jxx ADDR	Conditional jumps based on flags
CALL ADDR	Call procedure at ADDR
RET	Return from procedure
IRET	Return from interrupt
LOOPxx	Loop until condition met
INT n	Initiate a software interrupt
INTO	Interrupt if overflow bit is set

SRC = source
DST = destination

# = shift/rotate count
LV = # locals

Tipos de Instrucciones

Arithmetic

ADD DST,SRC	Add SRC to DST
SUB DST,SRC	Subtract SRC from DST
MUL SRC	Multiply EAX by SRC (unsigned)
IMUL SRC	Multiply EAX by SRC (signed)
DIV SRC	Divide EDX:EAX by SRC (unsigned)
IDIV SRC	Divide EDX:EAX by SRC (signed)
ADC DST,SRC	Add SRC to DST, then add carry bit
SBB DST,SRC	Subtract SRC & carry from DST
INC DST	Add 1 to DST
DEC DST	Subtract 1 from DST
NEG DST	Negate DST (subtract it from 0)

Binary coded decimal

DAA	Decimal adjust
DAS	Decimal adjust for subtraction
AAA	ASCII adjust for addition
AAS	ASCII adjust for subtraction
AAM	ASCII adjust for multiplication
AAD	ASCII adjust for division

SRC = source
DST = destination

# = shift/rotate count
LV = # locals

Tipos de Instrucciones

Boolean

AND DST,SRC	Boolean AND SRC into DST
OR DST,SRC	Boolean OR SRC into DST
XOR DST,SRC	Boolean Exclusive OR SRC to DST
NOT DST	Replace DST with 1's complement

Shift/rotate

SAL/SAR DST,#	Shift DST left/right # bits
SHL/SHR DST,#	Logical shift DST left/right # bits
ROL/ROR DST,#	Rotate DST left/right # bits
RCL/RCR DST,#	Rotate DST through carry # bits

Test/compare

TEST SRC1,SRC2	Boolean AND operands, set flags
CMP SRC1,SRC2	Set flags based on SRC1 - SRC2

SRC = source
DST = destination
# = shift/rotate count
LV = # locals

Tipos de Instrucciones

Condition codes

STC	Set carry bit in EFLAGS register
CLC	Clear carry bit in EFLAGS register
CMC	Complement carry bit in EFLAGS
STD	Set direction bit in EFLAGS register
CLD	Clear direction bit in EFLAGS reg
STI	Set interrupt bit in EFLAGS register
CLI	Clear interrupt bit in EFLAGS reg
PUSHFD	Push EFLAGS register onto stack
POPFD	Pop EFLAGS register from stack
LAHF	Load AH from EFLAGS register
SAHF	Store AH in EFLAGS register

SRC = source
DST = destination

# = shift/rotate count
LV = # locals

Tipos de Instrucciones

Strings

LODS	Load string
STOS	Store string
MOVS	Move string
CMPS	Compare two strings
SCAS	Scan Strings

SRC = source
DST = destination

# = shift/rotate count
LV = # locals

Tipos de Instrucciones

Miscellaneous

SWAP DST	Change endianness of DST
CWQ	Extend EAX to EDX:EAX for division
CWDE	Extend 16-bit number in AX to EAX
ENTER SIZE,LV	Create stack frame with SIZE bytes
LEAVE	Undo stack frame built by ENTER
NOP	No operation
HLT	Halt
IN AL,PORT	Input a byte from PORT to AL
OUT PORT,AL	Output a byte from AL to PORT
WAIT	Wait for an interrupt

SRC = source
DST = destination

# = shift/rotate count
LV = # locals