AAA: ASCII Adjust after Addition
If the rightmost 4 bits are > 9, 1 is added to the AH.
Flags: AF, CF.
Source code: AAA (no operand)
Object code: 00110111

AAM: ASCII Adjust after Multiplication
Divides the AL by 10. Stores the quotient in AH and remainder in AL.
Flags: PF, SF, ZF.
Source code: AAM (no operand)
Object code: | 11010100 | 00001010 |

AAS: ASCII Adjust after Subtraction
If the rightmost 4 bits are > 9, 6 is subtracted from AL and 1 from AH.
Flags: AF, CF.
Source code: AAS (no operand)
Object code: 00111111

AAD: ASCII Adjust before Division
Corrects the dividend to a binary value in the AL. Multiplies AH by 10 and adds the product to the AL.
Flags: PF, SF, ZF.
Source code: AAD (no operand)
Object code: | 11010101 | 00001010 |

ADC: Add with Carry
Flags: AF, CF, OF, PF, SF, ZF.
Source code: ADC {register/memory}, {register/memory/immediate}
Object code:
- Reg/mem with register: | 000100dw | mod reg r/m | desp-low | desp-high |
- Immed to accumulator: | 0001010w | -data- | data if w=1 |
- Immed to reg/mem: | 100000sw | mod010r/m | -data- | data if sw=01

ADD: Add Binary Numbers
Flags: AF, CF, OF, PF, SF, ZF.
Source code: ADD {register/memory}, {register/memory/immediate}
Object code:
- Reg/mem with register: | 000000dw | mod reg r/m | desp-low | desp-high |
- Immed to accumulator: | 0000010w | -data- | data if w=1 |
- Immed to reg/mem: | 100000sw | mod000r/m | -data- | data if sw=01

AND: Logical AND
Flags: CF (0), OF (0), PF, SF, ZF.
Source code: AND {register/memory}, {register/memory/immediate}
Object code:
- Reg/mem with register: | 001000dw | mod reg r/m | desp-low | desp-high |
- Immed to accumulator: | 0010010w | -data- | data if w=1 |
- Immed to reg/mem: | 100000sw | mod100r/m | -data- | data if sw=1

BSF, BSR: Bit Scan
BSF scan from left-to-right; BSR from right-to-left.
Flags: ZF.
Source code: BSF/BSR register, {register/memory}
Object code:
BSF: | 00001111 | 10111100 | mod reg/m | desp-low | desp-high
BSR: | 00001111 | 10111101 | mod reg/m | desp-low | desp-high

BT/BTC/BTR/BTS: Bit Test
Flags: CF.
Source code: BT/BTC/BTR/BTS {register/memory}, {register/immediate}
Object code:
- Reg/mem to reg: | 00001111 | 10***010 | mod reg r/m | desp-low | desp-high |
- Immed to reg: | 00001111 | 10111010 |mod***r/m | desp-low | desp-high |

*** *BT=100, BTC= 111, BTR= 110, BTS= 101*

CALL: Call a Procedure
Calls a NEAR (IP) or FAR (CS) procedure.
Source code: CALL {register/memory}
Object code:
- Direct within segment: | 11101000 | desp-low | desp-high |
- Indirect within segment: | 11111111 | mod010r/m | | desp-low | desp-high |
- Indirect intersegment: | 11111111 | mod011r/m | | desp-low | desp-high |
- Direct intersegment: | 10011010 | desp-low | desp-high | seg-low | seg-high

CMC: Complement Carry Flag
If CF=1, becomes 0, and vice versa.
Flags: CF (reversed)
Source code: CMC (no operand)
Object code: 11110101

CDQ: Convert Doubleword to Quadword
32 to 64 bits.
Source code: CDQ (no operand)
Object code: 10011001

CLI: Clear Interrupt Flag
Disable external interrupts.
Flags: IF (becomes 0)
Source code: CLI (no operand)
Object code: 11111010

CWD: Convert Word to Doubleword
16 to 32 bits, using AX.
Source code: CWD (no operand)
Object code: 10011001

CLC: Clear Carry Flag
There will be no carry.
Flags: CF (becomes 0)
Source code: CLC (no operand)
Object code: 11111000

CBW: Convert Byte to Word
8 to 16 bits.
Source code: CBW (no operand)
Object code: 10011000

CMP: Compare
Subtracts the second operand from the first, but doesn't store the result.
Flags: AF, CF, OF, PF, SF, ZF.
Source code: CMP {register/memory}, {register/memory/immediate}
Object code:

- Reg/mem with register: | 001110dw | mod reg r/m | desp-low | desp-high |
- Immed to accumulator: | 0011110w | -data- | data if w=1 |
- Immed to reg/mem: | 100000sw | mod111r/m | -data- | data if sw=0 | desp-low | desp-high

CLD: Clear Direction Flag
From left-to-right.
Flags: DF (becomes 0)
Source code: CLD (no operand)
Object code: 11111100

CMPS/CMPSB/CMPSW/CMPSD: Compare Strings
CMPSB compares bytes, CMPSW word, CMPSD doublewords.
Flags: AF, CF, OF, PF, SF, ZF.
Source code: [REPnn] CMPSB/CMPSW/CMPSD (no operand)
Object code: 1010011w

CWDE: Convert Word to Doubleword to Extended Doubleword
16 to 32 bits, using EAX
Source code: CWDE (no operand)
Object code: 10011000

DAA: Decimal Adjust after Addition
If the rightmost 4 bits are > 9, 6 is added to AL. If AL's value > 99H, 60H is added to AL.
Flags: AF, CF, PF, SF, ZF.
Source code: DAA (no operand)
Object code: 00100111

DAS: Decimal Adjust after Subtraction
If the rightmost 4 bits are > 9, 60H is subtracted from AL.
Flags: AF, CF, PF, SF, ZF.
Source code: DAS (no operand)
Object code: 00101111 (no operand)

DIV: Unsigned Divide. *The leftmost bit is treated as a data bit, not sign bit.*

Size	Dividend	Divisor	Quotient	Remainder	Example
16-bit	AX	8-bit reg/memory	AL	AH	DIV BH
32-bit	DX:AX	16-bit reg/memory	AX	DX	DIV CX
64-bit	EDX:EAX	32-bit reg/memory	EAX	EDX	DIV ECX

Flags: AF, CF, OF, PF, SF, ZF.
Source code: DIV {register/memory}
Object code: | 1111011w | mod110r/m | desp-low | desp-high

ESC: Escape
Facilitates the use of coprocessors.
Source code: ESC {register/memory}
Object code: |11011xxx | modxxxr/m | desp-low | desp-high
x-bits aren't important.

DEC: Decrement by 1
Flags: AF, OF, PF, SF, ZF.
Source code: DEC {register/memory}
Object code:

- Register: | 01001reg |
- Reg/memory: | 1111111w | mod001r/m | desp-low | desp-high

HLT: Enter Halt State
While waiting for an interrupt.
Source code: HLT (no operand)
Object code: 11110100

INTO: Interrupt on Overflow. *Performs INT 04H, if OF=1.*
Flags: IF, TF.
Source code: INTO (no operand)
Object code: 11001110

INT: Interrupt
Transfers controls to one of the 256 addresses.
Flags: Clears IF and TF.
Source code: INT number
Object code: | 1100110v | -type- | if v = 0 type is 3

IMUL: Signed Multiply. *Treats the leftmost bit as the sign*

Size	Multiplicand	Multiplier	Product	Example
8-bit	AL	8-bit reg/memory	AX	IMUL BL
16-bit	AX	16-bit reg/memory	DX:AX	IMUL BX
32-bit	EAX	32-bit reg/memory	EDX:EAX	IMUL ECX

Flags: CF, OF
Source code: MUL {register/memory}
Object code: | 1111011w | mod101r/m | desp-low | desp-high

IN: Input Byte or Word
Port must be a fixed numeric operand.
Source code: IN {AL/AX}, {portno/DX}
Object code:
- Variable port: |1110110w |
- Fixed port: | 1110010w | port |

INC: Increment by 1
Flags: AF, OF, PF, SF, ZF.
Source code: INC {register/memory}
Object code:
- Register: | 01000reg |
- Reg/memory: | 1111111w | mod000r/m | desp-low | desp-high

IDIV: Signed Divide. (Same table as DIV's)
Treats the leftmost bit as the sign.
Flags: AF, CF, OF, PF, SF, ZF.
Source code: IDIV {register/memory}
Object code: | 1111011w | mod111r/m | desp-low | desp-high

IRET/IRETD: Interrupt Return
Far return from an interrupt routine.
Flags: All.
Source code: IRET
Object code: 11001111 (no operand)

JC: Jump if Carry
Identical to JB/JNAE

JBE/JNA: Jump if Below/Equal or Jump if Not Above
If CF=1, a jump is performed.
Source code: JBE/JNA label
Object code: | 01110110 | -disp- |

JAE/JNB: Jump if Above/Equal or Jump if not Below
If CF=0, a jump is performed.
Source code: JAE/JNB label
Object code: | 01110011 | -disp- |

JA/JNBE: Jump if Above or Jump if not Below/Equal
If CF=0 and ZF=0, a jump is performed
Source code: JA/JNBE label
Object code: | 01110111 | -disp- |

JB/JNAE: Jump if Below or Jump if Not Above/Equal
If CF=1, a jump is performed.
Source code: JB/JNAE label
Object code: | 01110010 | -disp- |

JCXZ/JECXZ: Jump if CX/ECX is Zero
Jump to a specified address.
Source code: JCXZ/JECXZ label
Object code: | 11100011 | -disp- |

JE/JZ: Jump if Equal or Jump if Zero
If Zf=1, a jump is performed
Source code: JE/JZ label
Object code: | 01110100 | -disp- |

JNO: Jump if No Overflow
If OF=0, a jump is performed.
Source code: JNO label
Object code: | 01110001 | -disp- |

JL/JNGE: Jump if Less or Jump if Not Greater/Equal
If SF≠OF, a jump is performed.
Source code: JL/JNGE label
Object code: | 01111100 | -disp- |

JG/JNLE: Jump if Greater or Jump if Not Less/Equal
If ZF=0 and SF=OF, a jump is performed.
Source code: JG/JNLE label
Object code: | 01111111 | -disp- |

JGE/JNL: Jump if Greater/Equal or Jump if Not Less
If SF=OF, a jump is performed.
Source code: JGE/JNL label
Object code: | 01111101 | -disp- |

JLE/JNG: Jump if Less/Equal or Jump if Not Greater
If ZF=1 or SF≠OF, a jump is performed.
Source code: JL/JNGE label
Object code: | 01111110 | -disp- |

JNC: Jump if No Carry
Identical to JAE/JNB

JMP: Unconditional Jump
Jumps to a designated address under any condition.
Source code: JMP {register/memory}
Object code:
- Direct within seg short: | 11101011 | -desp- |
- Direct within segment: | 11101001 | desp-low | desp-high |
- Indirect within segment: | 11111111 | mod100r/m | desp-low | desp-high |
- Indirect intersegment: | 11111111 | mod101r/m | desp-low | desp-high |
- Direct intersegment: | 11101010 | desp-low | desp-high | seg-low | seg-high

JNE/JNZ: Jump if Not Equal or Jump if Not Zero
If ZF=0, a jump is performed.
Source code: JNE/JNZ label
Object code: | 01110101 | -disp- |

JNP/JPO: Jump if No Parity or Jump if Parity Odd
If PF=0 (odd parity), a jump is performed.
Source code: JNP/JPO label
Object code: | 01111011 | -disp- |

JO: Jump if Overflow
If OF=1, a jump is performed.
Source code: JO label

JNS: Jump if No Sign
If SF=0 (positive), a jump is performed.
Source code: JNS label
Object code: | 01111001 | -disp- |

JS: Jump if Sign
If SF=1, a jump is performed.
Source code: JS label
Object code: | 01111000 | -disp- |

LES/LFS/LGS: Load Extra Segment Register
See LDS.

JP/JPE: Jump if Parity or Jump if Parity Even
If PF=1 (even parity), a jump is performed.
Source code: JP/JPE label
Object code: | 01111010 | -disp- |

LAHF: Load AH from Flags
Loads the rightmost 8 bits of Flags register into AH.
Source code: LAHF (no operand)
Object code: 10011111

LSS: Load Stack Segment Register
See LDS

LDS/LES/LFS/LGS/LSS: Load Segment Register
Initializes far address and offset of data so instructions can access it
Source code: LDS/LES/LFS/LGS/LSS {register, memory}
Object code:
LDS: | 11000101 | mod reg r/m | desp-low | desp-high
LES: | 11000100 | mod reg r/m | desp-low | desp-high
LFS: | 00001111 | 10110100 | mod reg r/m | desp-low | desp-high
LGS: | 00001111 | 10110101 | mod reg r/m | desp-low | desp-high
LSS: | 00001111 | 10110010 | mod reg r/m | desp-low | desp-high

LODS/LODSB/LODSW/LODSD: Load Byte, Word or Doubleword String
To the AL, AX or EAX, respectively.
Source code: LODSB/LODSW/LODSD (no operand)
Object code: 1010110w

LOOP/LOOPW/LOOPD: Load AH from Flags
Controls the execution of a routine a specified number of times.
Source code: LOOP label
Object code: | 11100010 | -disp- |

LOOPE/LOOPZ: Loop while Equal or Loop while Zero
If CX is nonzero and ZF=1, they transfer the operand address
Source code: LOOPE/LOOPZ label
Object code: | 11100001 | -disp- |

LOOPNE/LOOPNZ: Loop while Not Equal or Loop while Not Zero
If CX is nonzero and ZF=0, they transfer the operand address
Source code: LOOPNE/LOOPNZ label
Object code: | 11100000 | -disp- |

LEA: Load Effective Address
Loads a near address into a register.
Source code: LEA {register, memory}
Object code: 10001101

LOCK: Lock Bus
Prevents coprocessors from changing data at the same time as the processor
Source code: LOCK instruction
Object code: 11110000

MOV: Move Data
Transfers data between registers or between a register and a memory.
Source code: MOV {register/memory}, {register/memory/immediate}
Object code:

- Reg/mem to/from reg: | 100010dw | mod reg r/m | desp-low | desp-high
- Immed to reg/mem: | 1100011w | mod000r/m | -data- | data if w=1 | desp-low | desp-high
- Immed to register: | 1011wreg | -data- | data if w=1 |
- Mem to accumulator: | 1010000w | addr-low | addr-high |
- Accumulator to mem: | 1010001w | addr-low | addr-high |
- Reg/mem to seg reg: | 10001110 | mog0sg r/m | desp-low | desp-high
- Seg reg to reg/mem: | 10001100 | mod0sg r/m | desp-low | desp-high

(sg=seg reg)

MOVS/MOVSB/MOVSW/MOVSD: Move String
REP decrements the CX by 1 for each repetition. Terminates when CX=0.
Source code: [REP] MOVSB/MOVSW/MOVSD (no operand)
Object code: 1010010w

MOVSX/MOVZX: Move with Sign Extend or Zero Extend
MOVSX fills the sign bit into leftmost bits. MOVZX fills with zero the bits.
Source code: MOVSX/MOVZX {register/memory}, {register/memory/immediate}
Object code:
MOVSX: | 00001111 | 1011111w | mod reg r/m | desp-low | desp-high
MOVZX: | 00001111 | 1011011w | mod reg r/m | desp-low | desp-high

MUL: Unsigned Multiply. (Same table as IMUL's)
Flags: CF, OF.
Source code: MUL {register/memory}
Object code: | 1111011w | mod100 r/m | desp-low | desp-high

NEG: Negate
Reverse a binary value (two's complement)
Flags: AF, CF, OF, PF, SF, ZF.
Source code: NEG {register/memory}
Object code: | 1111011w | mod011 r/m |

NOP: No Operation
Used to delete or insert machine code or to delay execution.
Source code: NOP (no operand)
Object code: 10010000

NOT: Logical NOT
Changes 0 to 1 and vice versa.
Source code: NOT {register/memory}
Object code: | 1111011w |

OR: Logical OR
Flags: CF (0), OF (0), PF, SF, ZF.
Source code: OR {register/memory}, {register/memory/immediate}
Object code:
- Reg/mem with register: | 000010dw | mod reg r/m | desp-low | desp-high
- Immed to accumulator: | 0000110w | -data- | data if w=1 |
- Immed to reg/mem: | 100000sw | mod 001 r/m | -data- | data if w=1 | desp-low | desp-high

OUT: Output Byte or Word
Source code: OUT {AL/AX}, {portno/DX}
Object code:
- Variable port: | 1110111w |
- Fixed port: | 1110011w | -port- |

PUSHF: Push Flags onto Stack
Pushes the contents of the flags register onto the stack for later use
Source code: PUSHF (no operand)
Object code: 10011100

POPF: Pop Flags off Stack
Pops the top word from the stack to the Flags register.
Flags: All.
Source code: POPF/POPFD (no operand)
Object code: 10011101

POP: Pop Word off Stack
Pops a w or dw pushed on the stack.to a specified destination
Source code: POP {register/memory}
Object code:
- Register: | 01011reg |
- Segment reg: | 000sg111 | (sg= segment reg)
- Reg/memory: | 10001111 | mod 000 r/m | desp-low | desp-high

PUSHA/PUSHAD: Push All General Registers
Pushes the AX, CX, DX, BX, SP, BP, SI and DI, onto the stack.
Source code: PUSHA/PUSHAD (no operand)
Object code: 01100000

POPA/POPAD: Pop All General Registers
Pops the top eight words from the stack into DI, SI, BP, SP, BX, DX, CX and AX.
Source code: POPA/POPAD (no operand)
Object code: 01100001

PUSH: Push onto Stack
Pushes a w or dw onto the stack for later use.
Source code: PUSH {register/memory}
Object code:
- Register: | 01010 reg |
- Segment reg: | 000 sg 110 |
- Reg/memory: | 11111111 | mod 110 r/m | desp-low | desp-high

REP: Repeat String
Flags: Associated string instructions
Source code: REP string-instruction
Object code: 11110010

REPE/REPZ/REPNE/REPNZ: Repeat String Conditionally
REPE/REPZ: Repeat while equal/zero. REPNE/REPNZ: Repeat while not equal/zero
Flags: Associated string instructions
Source code: REPE/REPZ/REPNE/REPNZ string-instruction
Object code:
REPNE/REPNZ: 11110010
REPE/REPZ: 11110011

RCL/RCR: Rotate Left through Carry and Rotate Right through Carry
RCL: the leftmost bit enters CF, and CF bit enters bit 0 of the destination.
RCR: bit 0 enters CF, and CF bit enters the leftmost bit of the destination.
Flags: CF, OF.
Source code: RCL/RCR {CL/immediate}, {register/memory}
Object code:
RCL: | 110100cw | mod 010 r/m | if c=0; shift is 1 | desp-low | desp-high
RCR: | 110100cw | mod 011 r/m | if c=1; shift is in CL | desp-low | desp-

ROL/ROR: Rotate Left and Rotate Right
Flags: CF and OF.
Source code: ROL/ROR {CL/immediate}, {register/memory}
Object code:
RCL: | 110100cw | mod 000 r/m | if c=0; count=1 | desp-low | desp-high
RCR: | 110100cw | mod 001 r/m | if c=1; count is in CL | desp-low | desp-high

SAL/SAR: Shift Algebraic Left or Shift Algebraic Right
SAL=SHL
SAR: considers the sign of the referenced field.
Flags: CF, OF, PF, SF, ZF.
Source code: SAL/SAR {CL/immediate}, {register/memory}
Object code:
SAL: | 110100cw | mod 100 r/m | if c=0; count=1 | desp-low | desp-high
SAR: | 110100cw | mod 111 r/m | if c=1; count is in CL | desp-low | desp-high

SBB: Subtract with Borrow
Flags: AF, CF, OF, PF, SF, ZF.
Source code: SBB {register/memory}, {register/memory/immediate}
Object code:
- Reg/mem with register: | 000110dw | mod reg r/m | desp-low | desp-high
- Immed to accumulator: | 0001110w | -data- | data if w=1 |
- Immed to reg/mem: | 100000sw | mod 011 r/m | -data- | data if sw=01 | desp-low | desp-high

SUB: Subtract Binary Values
Flags: AF, CF, OF, PF, SF, ZF.
Source code: SUB {register/memory}, {register/memory/immediate}
Object code:
- Reg/mem with register: | 001010dw | mod reg r/m | desp-low | desp-high
- Immed to accumulator: | 0010110w | -data- | data if w=1 |
- Immed to reg/mem: | 100000sw | mod 101 r/m | -data- | data if sw=01 | desp-low | desp-high

SHL/SHR: Shift Logical Left or Shift Logical Right
Flags: CF, OF, PF, SF, ZF.
Source code: SHL/SHR {CL/immediate}, {register/memory}
Object code:
SHL: | 110100cw | mod 100 r/m | if c=0; count=1 | desp-low | desp-high
SHR: | 110100cw | mod 101 r/m | if c=1; count is in CL | desp-low | desp-high

RET/RETN/RETF: Return from a Procedure
RETN: NEAR. RETF: FAR
Source code: RET/RETN/RETF [pop-value]
Object code:
- Within a segment: | 11000011 |
- Within a segment with pop value: |11000010 | data-low | data-high |
- Intersegment: | 11001011 |
- Intersegment with pop value: | 11001010 | data-low | data -high |

SAHF: Store AH Contents in Flags
Flags: AF, CF, PF, SF, ZF.
Source code: SAHF (no operand)
Object code: 10011110

SHLD/SHRD: Shift Double Precision
Shifts multiple bits into an operand.
Flags: CF, OF, PF, SF, ZF.
Source code: SHLD/SHRD {CL/immediate}, {register}, {register/memory}
Object code: | 00001111 | 10100100 | mod reg r/m | desp-low | desp-high

SCAS/SCASB/SCASW/SCASD: Scan String
Scans a string in memory for a specified value.
Flags: AF, CF, OF, PF, SF, ZF.
Source code: [REPnn] SCASB/SCASW/SCASD (no operand)
Object code: 1010111w

STOS/STOSB/STOSW/STOSD: Store String
Stores the contents of the accumulator in memory.
Source code: [REP] STOSB/STOSW/STOSD (no operand)
Object code: 1010101w

SETnn: Set Byte Conditionally
If a tested condition is true, the operation sets the byte operand to 1, otherwise to 0.
Source code: SETnn {register/memory}
Object code: | 00001111 | 1001 cond | mod 000 r/m | desp-low | desp-high
cond= condition

STC: Set Carry Flag
Flags: Sets CF to 1.
Source code: STC (no operand)
Object code: 11111001

STD: Set Direction Flag
Flags: Sets DF to 1.
Source code: STD (no operand)
Object code: 11111101

STI: Set Interrupt Flag
Flags: Sets IF to 1.
Source code: STI (no operand)
Object code: 11111011

WAIT: Put Processor in Wait State
Until an external interrupt occurs.
Source code: WAIT (no operand)
Object code: 10011011

TEST: Test Bits
Tests a field for a specific bit configuration.
Flags: PF, SF, ZF. Clears CF and OF.
Source code: TEST {register/memory}, {register/memory/immediate}
Object code:

- Reg/mem with register: | 1000010dw | mod reg r/m | desp-low | desp-high
- Immed to accumulator: | 1010100w | -data- | data if w=1 |
- Immed to reg/mem: | 1111011w | mod 000 r/m | -data- | data if w=1 | desp-low | desp-high

XCHG: Exchange
Exchanges data between two registers or between a register a memory
Source code: XCHG {register/memory}, {register/memory/immediate}
Object code:

- Reg with accumulator: | 10010 reg | desp-low | desp-high
- Reg/mem with reg: | 1000011w | mod reg r/m | desp-low | desp-high

XLAT/XLATB: Translate
Translates bytes into a different format, such as ASCII to EBCDIC
Source code: XLAT [AL*]
Object code: 11010111
*optional

XOR: Exclusive OR
If the matched bits are the same, the bit in the first operand is cleared to 0.
If the matched bits are different the bit in the first operand is set to 1.
Flags: CF (0), OF (0), PF, SF, ZF.
Source code: XOR {register/memory}, {register/memory/immediate}
Object code:

- Reg/mem with register: | 001100dw | mod reg r/m | desp-low | desp-high
- Immed to accumulator: | 0011010w | -data- | data if w=1 |
- Immed to reg/mem: | 1000000w | mod 110 r/m | -data- | data if w=1 | desp-low | desp-high

Notas:

- disp=desp
- Los cuadros sin “Flags” son los que no afectan a ninguna bandera.