
SOURCE: http://www.csn.ul.ie/~darkstar/assembler/a86.zip

How to Read the Instruction Set Chart

The following chart summarizes the machine instructions you can program with A86. In order to use the chart, you need to learn the meanings of the specifiers (each given by 2 lower case letters) that follow most of the instruction mnemonics. Each specifier indicates the type of operand (register byte, immediate word, etc.) that follows the mnemonic to produce the given opcodes. The "v" type, for A86, is the same as "w" -- it denotes a 16-bit word. On A386, "v" denotes either a word or doubleword, depending on the presence of an operand override prefix byte.

"c" means the operand is a code label, pointing to a part of the program to be jumped to or called. A86 will also accept a constant offset in this place (or a constant segment-offset pair in the case of "cp"). "cb" is a label within about 128 bytes (in either direction) of the current location. "cv" is a label within the same code segment as this program; "cp" is a pair of constants separated by a colon-- the segment value to the left of the colon, and the offset to the right. The offset is always a word in A86; it can be either a word or a doubleword in A386. Note that in both the cb and cv cases, the object code generated is the offset from the location following the current instruction, not the absolute location of the label operand. In some assemblers (most notably for the Z-80 processor) you have to code this offset explicitly by putting "$-" before every relative jump operand in your source code. You do NOT need to, and should not do so with A86.

"e" means the operand is an Effective Address. The concept of an Effective Address is central to the 86 machine architecture, and thus to 86 assembly language programming. It is described in detail at the start of this chapter. We summarize here by saying that an Effective Address is either a general purpose register, a memory variable, or an indexed memory quantity. For example, the instruction "ADD rb,eb" includes the instructions: ADD AL,BL, and ADD CH,BYTEVAR, and ADD DL,B[BX+17].

"i" means the operand is an immediate constant, provided as part of the instruction itself. "ib" is a byte-sized constant; "iw" is a constant occupying a full 16-bit word. The operand can also be a label, defined with a colon. In that case, the immediate constant which is the location of the label is used. Examples: "MOV rw,iw" includes the instructions: MOV AX,17, or MOV SI,VAR_ARRAY, where "VAR_ARRAY:" appears somewhere in the program, defined with a colon. NOTE that if VAR_ARRAY were defined without a colon, e.g., "VAR_ARRAY DW 1,2,3", then "MOV SI,VAR_ARRAY" would be a "MOV rw,ew" NOT a "MOV rw,iw". The MOV would move the contents of memory at VAR_ARRAY (in this case 1) into SI, instead of the location of the memory. To load the location, you can code "MOV SI,OFFSET VAR_ARRAY".

"m" means a memory variable or an indexed memory quantity; i.e., any Effective Address EXCEPT a register.

"r" means the operand is a general purpose register. The 8 "rb" registers are AL,BL,CL,DL,AH,BH,CH,DH; the 8 "rw" registers are AX,BX,CX,DX,SI,DI,BP,SP.

"rv/m" is used in the Bit Test instructions to denote either a word-or-doubleword register, or an array of bits in memory that can any length.

NOTE: The following chart gives all instructions for all processors through the Pentium. You must take care to use only the instructions appropriate for the target processor of your program (the P switch will enforce this for you: see Chapter 3).

If an instruction form does not run on all processors, there is a letter or digit just before the description field. "N" means the instruction runs only on NEC processors (which are rare nowdays). A digit x means the instruction runs on the x86 or later: 1 for 186, 2 for 286, 3 for 386, 4 for 486, 5 for Pentium. Instructions with 3 or greater are recognized only by my A386 assembler, received only by those who register both A86 and D86.

```
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/

Opcodes     Instruction      Description

67 or nil   A2 (prefix)    3 Use 16-bit address (indexing) in next instruction
67 or nil   A4 (prefix)    3 Use 32-bit address (indexing) in next instruction
37          AAA              ASCII adjust AL (carry into AH) after addition
D5 0A       AAD              ASCII adjust before division (AX = 10*AH + AL)
D4 0A       AAM              ASCII adjust after multiply (AL/10: AH=Quo AL=Rem)
3F          AAS              ASCII adjust AL (borrow from AH) after subtraction

14 ib       ADC AL,ib        Add with carry immediate byte into AL
15 iv       ADC eAX,iv       Add with carry immediate vword into eAX
80 /2 ib    ADC eb,ib        Add with carry immediate byte into EA byte
10 /r       ADC eb,rb        Add with carry byte register into EA byte
83 /2 ib    ADC ev,ib        Add with carry immediate byte into EA vword
81 /2 iv    ADC ev,iv        Add with carry immediate vword into EA vword
11 /r       ADC ev,rv        Add with carry vword register into EA vword
12 /r       ADC rb,eb        Add with carry EA byte into byte register
13 /r       ADC rv,ev        Add with carry EA vword into vword register

04 ib       ADD AL,ib        Add immediate byte into AL
05 iv       ADD eAX,iv       Add immediate vword into eAX
80 /0 ib    ADD eb,ib        Add immediate byte into EA byte
00 /r       ADD eb,rb        Add byte register into EA byte
83 /0 ib    ADD ev,ib        Add immediate byte into EA vword
81 /0 iv    ADD ev,iv        Add immediate vword into EA vword
01 /r       ADD ev,rv        Add vword register into EA vword
02 /r       ADD rb,eb        Add EA byte into byte register
03 /r       ADD rv,ev        Add EA vword into vword register
0F 20       ADD4S          N Add CL nibbles BCD, DS:SI into ES:DI (CL even,NZ)

24 ib       AND AL,ib        Logical-AND immediate byte into AL
25 iv       AND eAX,iv       Logical-AND immediate vword into eAX
80 /4 ib    AND eb,ib        Logical-AND immediate byte into EA byte
20 /r       AND eb,rb        Logical-AND byte register into EA byte
83 /4 ib    AND ev,ib        Logical-AND immediate byte into EA vword
81 /4 iv    AND ev,iv        Logical-AND immediate vword into EA vword
21 /r       AND ev,rv        Logical-AND vword register into EA vword
22 /r       AND rb,eb        Logical-AND EA byte into byte register
23 /r       AND rv,ev        Logical-AND EA vword into vword register
63 /r       ARPL ew,rw     2 Adjust RPL of EA word not smaller than RPL of rw

62 /r       BOUND rv,m2v   2 INT 5 if rw not between 2 vwords at [m] inclusive
0F BC       BSF rv,ev      3 Set rv to lowest position of NZ bit in ev
0F BD       BSR rv,ev      3 Set rv to highest position of NZ bit in ev
0F C8+r     BSWAP rd       4 Swap bytes 1,4 and 2,3 of dword register

0F BA/4 ib BT rv/m,ib      3 Set Carry flag to bit # ib of array at rv/m
0F A3/r     BT rv/m,rv     3 Set Carry flag to bit # rv of array at rv/m
0F BA/7 ib BTC rv/m,ib     3 Set CF to, then compl bit ib of array at rv/m
0F BB/r     BTC rv/m,rv    3 Set CF to, then compl bit rv of array at rv/m
0F BA/6 ib BTR rv/m,       3 Set CF to, then reset bit ib of array at rv/m
0F B3/r     BTR rv/m,rv    3 Set CF to, then reset bit rv of array at rv/m
0F BA/5 ib BTS rv/m,ib     3 Set CF to, then set bit ib of array at rv/m
0F AB/r     BTS rv/m,rv    3 Set CF to, then set bit rv of array at rv/m

9A cp       CALL cp          Call far segment, immediate 4- or 6-byte address
E8 cv       CALL cv          Call near, offset relative to next instruction
FF /3       CALL ep          Call far segment, address at EA memory location
FF /2       CALL ev          Call near, offset absolute at EA vword
0F FF ib    CALL80 ib      N Call 8080-emulation code at INT number ib

98          CBW              Convert byte into word (AH = top bit of AL)
99          CDQ            3 Convert dword to qword (EDX = top bit of EAX)
```

```
F8          CLC             Clear carry flag
FC          CLD             Clear direction flag so SI and DI will increment
FA          CLI             Clear interrupt enable flag; interrupts disabled

0F 12/0     CLRBIT eb,CL  N Clear bit CL of eb
0F 13/0     CLRBIT ew,CL  N Clear bit CL of ew
0F 1A/0 ib  CLRBIT eb,ib  N Clear bit ib of eb
0F 1B/0 ib  CLRBIT ew,ib  N Clear bit ib of ew
0F 06       CLTS          2 Clear task switched flag
F5          CMC             Complement carry flag

3C ib       CMP AL,ib       Subtract immediate byte from AL for flags only
3D iv       CMP eAX,iv      Subtract immediate vword from eAX for flags only
80 /7 ib    CMP eb,ib       Subtract immediate byte from EA byte for flags only
38 /r       CMP eb,rb       Subtract byte register from EA byte for flags only
83 /7 ib    CMP ev,ib       Subtract immediate byte from EA vword for flags only
81 /7 iv    CMP ev,iv       Subtract immediate vword from EA vword, flags only
39 /r       CMP ev,rv       Subtract vword register from EA vword for flags only
3A /r       CMP rb,eb       Subtract EA byte from byte register for flags only
3B /r       CMP rv,ev       Subtract EA vword from vword register for flags only

0F 26       CMP4S         N Compare CL nibbles BCD, DS:SI - ES:DI (CL even,NZ)
A6          CMPS mb,mb      Compare bytes [SI] - ES:[DI], advance SI,DI
A7          CMPS mv,mv      Compare vwords [SI] - ES:[DI], advance SI,DI
A6          CMPSB           Compare bytes DS:[SI] - ES:[DI], advance SI,DI
A7          CMPSD           Compare dwords DS:[SI] - ES:[DI], advance SI,DI
A7          CMPSW           Compare words DS:[SI] - ES:[DI], advance SI,DI

0F C7 /1    CMPX8 mq      5 If EDXEAX=mq then mq:=ECXEBX, else EAXEDX:=mq
0F B0 /r    CMPXCHG eb,rb 4 If AL=eb then set eb to rb, else set AL to eb
0F B1 /r    CMPXCHG ev,rv 4 If eAX=ev then set ev to rv, else set eAX to ev
0F A2       CPUID         5 If EAX=1 set EDXEAX to CPU identification values
99          CWD             Convert word to doubleword (DX = top bit of AX)
98          CWDE          3 Sign-extend word AX to doubleword EAX
2E          CS (prefix)     Use CS segment for the following memory reference

27          DAA             Decimal adjust AL after addition
2F          DAS             Decimal adjust AL after subtraction
FE /1       DEC eb          Decrement EA byte by 1
FF /1       DEC ev          Decrement EA vword by 1
48+rv       DEC rv          Decrement vword register by 1

F6 /6       DIV eb          Unsigned divide AX by EA byte (AL=Quo AH=Rem)
F7 /6       DIV ev          Unsigned divide eDXeAX by EA vword (eAX=Quo eDX=Rem)
3E          DS (prefix)     Use DS segment for the following memory reference

C8 iw 00    ENTER iw,0    1 Make stack frame, iw bytes local storage, 0 levels
C8 iw 01    ENTER iw,1    1 Make stack frame, iw bytes local storage, 1 level
C8 iw ib    ENTER iw,ib   1 Make stack frame, iw bytes local storage, ib levels
26          ES (prefix)     Use ES segment for the following memory reference
            F(any)          Floating point set is in Chapter 7
F4          HLT             Halt

F6 /7       IDIV eb         Signed divide AX by EA byte (AL=Quo AH=Rem)
F7 /7       IDIV ev         Signed divide eDXeAX by EA vword (eAX=Quo eDX=Rem)
F6 /5       IMUL eb         Signed multiply (AX = AL * EA byte)
F7 /5       IMUL ev         Signed multiply (eDXeAX = eAX * EA vword)
0F AF /r    IMUL rv,ev    3 Signed multiply ev into rv
6B /r ib    IMUL rv,ib    1 Signed multiply imm byte into vword register
69 /r iv    IMUL rv,iv    1 Signed multiply imm vword into vword register
69 /r iv    IMUL rv,ev,iv 1 Signed multiply (rv = EA vword * imm vword)
6B /r ib    IMUL rv,ev,ib 1 Signed multiply (rv = EA vword * imm byte)

E4 ib       IN AL,ib        Input byte from immediate port into AL
EC          IN AL,DX        Input byte from port DX into AL
E5 ib       IN eAX,ib       Input vword from immediate port into eAX
ED          IN eAX,DX       Input vword from port DX into eAX

FE /0       INC eb          Increment EA byte by 1
FF /0       INC ev          Increment EA vword by 1
40+rv       INC rv          Increment vword register by 1

6C          INS eb,DX     1 Input byte from port DX into [DI], advance DI
6D          INS ev,DX     1 Input vword from port DX into [DI], advance DI
6C          INSB          1 Input byte from port DX into ES:[DI], advance DI
6D          INSD          3 Input dword from port DX into ES:[DI], advance DI
```

```
6D          INSW           1 Input vword from port DX into ES:[DI], advance DI

CC          INT 3            Interrupt 3 (trap to debugger) (far call, with flags
CD ib       INT ib           Interrupt numbered by immediate byte    pushed first)
CE          INTO             Interrupt 4 if overflow flag is 1
0F 08       INVD           4 Invalidate the Data Cache without writing
0F 01 /7    INVLPG m       4 Invalidate the TLB Entry that points to m
CF          IRET             Interrupt return (far return and pop flags)
CF          IRETD          3 Interrupt return (pop EIP, ECS, Eflags)

77 cb       JA cb            Jump short if above (CF=0 and ZF=0)    above=UNSIGNED
73 cb       JAE cb           Jump short if above or equal (CF=0)
72 cb       JB cb            Jump short if below (CF=1)             below=UNSIGNED
76 cb       JBE cb           Jump short if below or equal (CF=1 or ZF=1)
72 cb       JC cb            Jump short if carry (CF=1)

E3 cb       JCXZ cb          Jump short if CX register is zero
74 cb       JE cb            Jump short if equal (ZF=1)
E3 cb       JECXZ cb       3 Jump short if ECX register is zero
7F cb       JG cb            Jump short if greater (ZF=0 and SF=OF) greater=SIGNED
7D cb       JGE cb           Jump short if greater or equal (SF=OF)
7C cb       JL cb            Jump short if less (SF>OF)               less=SIGNED
7E cb       JLE cb           Jump short if less or equal (ZF=1 or SF>OF)

EB cb       JMP cb           Jump short (signed byte relative to next instruction)
EA cp       JMP cp           Jump far (4- or 6-byte immediate address)
E9 cv       JMP cv           Jump near (vword offset relative to next instruction)
0F 8n cv    Jcond LONG cv  3 Jump, if condition, to offset >127 away
FF /4       JMP ev           Jump near to EA vword (absolute offset)
FF /5       JMP md           Jump far (4-byte address in memory doubleword)

76 cb       JNA cb           Jump short if not above (CF=1 or ZF=1)
72 cb       JNAE cb          Jump short if not above or equal (CF=1)
73 cb       JNB cb           Jump short if not below (CF=0)
77 cb       JNBE cb          Jump short if not below or equal (CF=0 and ZF=0)
73 cb       JNC cb           Jump short if not carry (CF=0)

75 cb       JNE cb           Jump short if not equal (ZF=0)
7E cb       JNG cb           Jump short if not greater (ZF=1 or SF>OF)
7C cb       JNGE cb          Jump short if not greater or equal (SF>OF)
7D cb       JNL cb           Jump short if not less (SF=OF)
7F cb       JNLE cb          Jump short if not less or equal (ZF=0 and SF=OF)

71 cb       JNO cb           Jump short if not overflow (OF=0)
7B cb       JNP cb           Jump short if not parity (PF=0)
79 cb       JNS cb           Jump short if not sign (SF=0)
75 cb       JNZ cb           Jump short if not zero (ZF=0)
70 cb       JO cb            Jump short if overflow (OF=1)

7A cb       JP cb            Jump short if parity (PF=1)
7A cb       JPE cb           Jump short if parity even (PF=1)
7B cb       JPO cb           Jump short if parity odd (PF=0)
78 cb       JS cb            Jump short if sign (SF=1)
74 cb       JZ cb            Jump short if zero (ZF=1)

9F          LAHF             Load: AH = flags  SF ZF xx AF xx PF xx CF
0F 02 /r    LAR rv,ew      2 Load: high(rw) = Access Rights byte, selector ew
C5 /r       LDS rv,ep        Load EA pointer into DS and vword register
8D /r       LEA rv,m         Calculate EA offset given by m, place in rv
C9          LEAVE          1 Set SP to BP, then POP BP (reverses previous ENTER)
C4 /r       LES rv,ep        Load EA pointer into ES and vword register
0F B4 /r    LFS rv,ep      3 Load EA pointer into FS and vword register

0F 01 /2    LGDT m         2 Load 6 bytes at m into Global Descriptor Table reg
0F B5 /r    LGS rv,ep      3 Load EA pointer into GS and vword register
0F 01 /3    LIDT m         2 Load 6 bytes into Interrupt Descriptor Table reg
0F 00 /2    LLDT ew        2 Load selector ew into Local Descriptor Table reg
0F 01 /6    LMSW ew        2 Load EA word into Machine Status Word
F0          LOCK (prefix)    Assert BUSLOCK signal for the next instruction

0F 33/r     LODBITS rb,rb N Load AX with DS:SI,bit rb (incr. SI,rb), rb+1 bits
0F 3B/0 ib  LODBITS rb,ib N Load AX with DS:SI,bit rb (incr. SI,rb), ib+1 bits
AC          LODS mb          Load byte [SI] into AL, advance SI
AD          LODS mv          Load vword [SI] into eAX, advance SI
AC          LODSB            Load byte [SI] into AL, advance SI
AD          LODSD            Load dword [SI] into EAX, advance SI
```

```
AD          LODSW             Load word [SI] into AX, advance SI

E2 cb       LOOP cb           noflags DEC CX; jump short if CX>0
E1 cb       LOOPE cb          noflags DEC CX; jump short if CX>0 and equal (ZF=1)
E0 cb       LOOPNE cb         noflags DEC CX; jump short if CX>0 and not equal
E0 cb       LOOPNZ cb         noflags DEC CX; jump short if CX>0 and ZF=0
E1 cb       LOOPZ cb          noflags DEC CX; jump short if CX>0 and zero (ZF=1)

0F 03 /r    LSL rv,ev      2  Load: rv = Segment Limit, selector ew
0F B2 /r    LSS rv,ep      3  Load EA pointer into SS and vword register
0F 00 /3    LTR ew         2  Load EA word into Task Register

A0 iv       MOV AL,xb         Move byte variable (offset iv) into AL
A1 iv       MOV eAX,xv        Move vword variable (offset iv) into eAX
0F 22 /4    MOV CR4,rd     5  Move rd into control register 4
0F 22 /n    MOV CRn,rd     3  Move rd into control register n (=0,2, or 3)

0F 23 /n    MOV DRn,rd     3  Move rd into debug register n (=0,1,2,3)
0F 23 /n    MOV DRn,rd     3  Move rd into debug register n (=6,7)
0F 26 /n    MOV TRn,rd     3  Move rd into test register TRn (=6,7)

C6 /0 ib    MOV eb,ib         Move immediate byte into EA byte
88 /r       MOV eb,rb         Move byte register into EA byte
C7 /0 iv    MOV ev,iv         Move immediate vword into EA vword
89 /r       MOV ev,rv         Move vword register into EA vword

8C /r       MOV ew,segreg     Move segment register into EA word
B0+rb ib    MOV rb,ib         Move immediate byte into byte register
8A /r       MOV rb,eb         Move EA byte into byte register
0F 20 /4    MOV rd,CR4     5  Move control register 4 into rd
0F 20 /n    MOV rd,CRn     3  Move control register n (=0,2, or 3) into rd

0F 21 /n    MOV rd,DRn     3  Move debug register n (=0,1,2,3) into rd
0F 21 /n    MOV rd,DRn     3  Move debug register n (=6,7) into rd
0F 24 /n    MOV rd,TRn     3  Move test register TRn (=6,7) into rd
B8+rw iv    MOV rv,iv         Move immediate vword into vword register
8B /r       MOV rv,ev         Move EA vword into vword register

8E /r       MOV segreg,mw     Move EA word into segment register (except CS)
A2 iv       MOV xb,AL         Move AL into byte variable (offset iv)
A3 iv       MOV xv,eAX        Move eAX into vword register (offset iv)
A4          MOVS mb,mb        Move byte [SI] to ES:[DI], advance SI,DI
A5          MOVS mv,mv        Move vword [SI] to ES:[DI], advance SI,DI
A4          MOVSB             Move byte DS:[SI] to ES:[DI], advance SI,DI
A5          MOVSD          3  Move dword DS:[SI] to ES:[DI], advance SI,DI
A5          MOVSW             Move word DS:[SI] to ES:[DI], advance SI,DI

0F BF /r    MOVSX rd,ew    3  Move word to dword, with sign-extend
0F BE /r    MOVSX rv,eb    3  Move byte to vword, with sign-extend
0F B7 /r    MOVZX rd,ew    3  Move word to dword, with zero-extend
0F B6 /r    MOVZX rv,eb    3  Move byte to vword, with zero-extend
8C /r       MOVZX rw,seg   3  Move segment register into EA word

F6 /4       MUL eb            Unsigned multiply (AX = AL * EA byte)
F7 /4       MUL ev            Unsigned multiply (eDXeAX = eAX * EA vword)
F6 /3       NEG eb            Two's complement negate EA byte
F7 /3       NEG ev            Two's complement negate EA vword
            NIL (prefix)      Special "do-nothing" opcode assembles no code
90          NOP               No Operation

F6 /2       NOT eb            Reverse each bit of EA byte
F7 /2       NOT ev            Reverse each bit of EA word
0F 16/0     NOTBIT eb,CL   N   Complement bit CL of eb
0F 17/0     NOTBIT ew,CL   N   Complement bit CL of ew
0F 1E/0 ib  NOTBIT eb,ib   N  Complement bit ib of eb
0F 1F/0 ib  NOTBIT ew,ib   N  Complement bit ib of ew

66 or nil   O2 (prefix)    3  Use 16-bit data operand in the next instruction
66 or nil   O4 (prefix)    3  Use 32-bit data operand in the next instruction
0C ib       OR AL,ib          Logical-OR immediate byte into AL
0D iv       OR eAX,iv         Logical-OR immediate word into eAX
80 /1 ib    OR eb,ib          Logical-OR immediate byte into EA byte
08 /r       OR eb,rb          Logical-OR byte register into EA byte
83 /1 ib    OR ev,ib          Logical-OR immediate byte into EA word
81 /1 iv    OR ev,iv          Logical-OR immediate word into EA word
09 /r       OR ev,rv          Logical-OR word register into EA word
```

```
0A /r      OR rb,eb          Logical-OR EA byte into byte register
0B /r      OR rv,ev          Logical-OR EA word into word register

E6 ib      OUT ib,AL         Output byte AL to immediate port number ib
E7 ib      OUT ib,eAX        Output word eAX to immediate port number ib
EE         OUT DX,AL         Output byte AL to port number DX
EF         OUT DX,eAX        Output word eAX to port number DX
6E         OUTS DX,eb      1 Output byte [SI] to port number DX, advance SI
6F         OUTS DX,ev      1 Output word [SI] to port number DX, advance SI
6E         OUTSB           1 Output byte DS:[SI] to port number DX, advance SI
6F         OUTSD           3 Output dword DS:[SI] to port number DX, advance SI
6F         OUTSW           1 Output word DS:[SI] to port number DX, advance SI

1F         POP DS            Set DS to top of stack, increment SP by 2
07         POP ES            Set ES to top of stack, increment SP by 2
0F A1      POP FS          3 Set FS to top of stack, increment SP by 2
0F A9      POP GS          3 Set GS to top of stack, increment SP by 2
8F /0      POP mv            Set memory word to top of stack, increment SP by 2
58+rw      POP rv            Set word register to top of stack, increment SP by 2
17         POP SS            Set SS to top of stack, increment SP by 2

61         POPA            1 Pop DI,SI,BP,SP,BX,DX,CX,AX (SP value is ignored)
61         POPAD           3 Pop EDI,ESI,EBP,x,EBX,EDX,ECX,EAX (ESP ign.)
9D         POPF              Set flags register to top of stack, increment SP by 2
9D         POPFD           3 Set eflags reg to top of stack, incr SP by 2

0E         PUSH CS           Set [SP-2] to CS, then decrement SP by 2
1E         PUSH DS           Set [SP-2] to DS, then decrement SP by 2
06         PUSH ES           Set [SP-2] to ES, then decrement SP by 2
0F A0      PUSH FS         3 Set [SP-2] to FS, then decrement SP by 2
0F A8      PUSH GS         3 Set [SP-2] to GS, then decrement SP by 2
6A ib      PUSH ib         1 Push sign-extended immediate byte
68 iv      PUSH iv         1 Set [SP-2] to immediate word, then decrement SP by 2
FF /6      PUSH mv           Set [SP-2] to memory word, then decrement SP by 2
50+rw      PUSH rv           Set [SP-2] to word register, then decrement SP by 2
16         PUSH SS           Set [SP-2] to SS, then decrement SP by 2

60         PUSHA           1 Push AX,CX,DX,BX,original SP,BP,SI,DI
60         PUSHAD          3 Push EAX,ECX,EDX,EBX,original ESP,EBP,ESI,EDI
9C         PUSHF             Set [SP-2] to flags register, then decrement SP by 2
9C         PUSHFD          3 Set [SP-4] to eflags reg, then decr SP by 4

D0 /2      RCL eb,1          Rotate 9-bit quantity (CF, EA byte) left once
D2 /2      RCL eb,CL         Rotate 9-bit quantity (CF, EA byte) left CL times
C0 /2 ib   RCL eb,ib       1 Rotate 9-bit quantity (CF, EA byte) left ib times
D1 /2      RCL ev,1          Rotate v+1-bit quantity (CF, EA word) left once
D3 /2      RCL ev,CL         Rotate v+1-bit quantity (CF, EA word) left CL times
C1 /2 ib   RCL ev,ib       1 Rotate v+1-bit quantity (CF, EA word) left ib times

D0 /3      RCR eb,1          Rotate 9-bit quantity (CF, EA byte) right once
D2 /3      RCR eb,CL         Rotate 9-bit quantity (CF, EA byte) right CL times
C0 /3 ib   RCR eb,ib       1 Rotate 9-bit quantity (CF, EA byte) right ib times
D1 /3      RCR ev,1          Rotate v+1-bit quantity (CF, EA word) right once
D3 /3      RCR ev,CL         Rotate v+1-bit quantity (CF, EA word) right CL times
C1 /3 ib   RCR ev,ib       1 Rotate v+1-bit quantity (CF, EA word) right ib times
0F 32      RDMSR           5 Read Model Specific Reg #ECX to EDXEAX
0F 31      RDTSC           5 Read Time Stamp Counter to EDXEAX

F3         REP (prefix)      Repeat following MOVS,LODS,STOS,INS, or OUTS CX times
65         REPC (prefix)   N Repeat following CMPS or SCAS CX times or until CF=0
F3         REPE (prefix)     Repeat following CMPS or SCAS CX times or until ZF=0
64         REPNC (prfix)   N Repeat following CMPS or SCAS CX times or until CF=1
F2         REPNE (prfix)     Repeat following CMPS or SCAS CX times or until ZF=1
F2         REPNZ (prfix)     Repeat following CMPS or SCAS CX times or until ZF=1
F3         REPZ (prefix)     Repeat following CMPS or SCAS CX times or until ZF=0

CB         RETF              Return to far caller (pop offset, then seg)
C3         RET               Return to near caller (pop offset only)
CA iw      RETF iw           RET (far), pop offset, seg, iw bytes
C2 iw      RET iw            RET (near), pop offset, iw bytes pushed before Call

D0 /0      ROL eb,1          Rotate 8-bit EA byte left once
D2 /0      ROL eb,CL         Rotate 8-bit EA byte left CL times
C0 /0 ib   ROL eb,ib       1 Rotate 8-bit EA byte left ib times
D1 /0      ROL ev,1          Rotate 16- or 32-bit EA vword left once
D3 /0      ROL ev,CL         Rotate 16- or 32-bit EA vword left CL times
```

```
C1 /0 ib    ROL ev,ib      1 Rotate 16 or 32-bit EA vword left ib times
0F 28/0     ROL4 eb        N Rotate nibbles: Heb=Leb   HAL,Leb=LAL  LAL=Heb

D0 /1       ROR eb,1         Rotate 8-bit EA byte right once
D2 /1       ROR eb,CL        Rotate 8-bit EA byte right CL times
C0 /1 ib    ROR eb,ib      1 Rotate 8-bit EA byte right ib times
D1 /1       ROR ev,1         Rotate 16- or 32-bit EA vword right once
D3 /1       ROR ev,CL        Rotate 16- or 32-bit EA vword right CL times
C1 /1 ib    ROR ev,ib      1 Rotate 16- or 32-bit EA vword right ib times
0F 2A/0     ROR4 eb        N Rotate nibbles: Leb=Heb   Heb=LAL  AL=eb
0F AA       RSM            5 Resume from System Management mode

9E          SAHF             Store AH into flags  SF ZF xx AF xx PF xx CF
D0 /4       SAL eb,1         Multiply EA byte by 2, once
D2 /4       SAL eb,CL        Multiply EA byte by 2, CL times
C0 /4 ib    SAL eb,ib      1 Multiply EA byte by 2, ib times
D1 /4       SAL ev,1         Multiply EA vword by 2, once
D3 /4       SAL ev,CL        Multiply EA vword by 2, CL times
C1 /4 ib    SAL ev,ib      1 Multiply EA vword by 2, ib times

D0 /7       SAR eb,1         Signed divide EA byte by 2, once
D2 /7       SAR eb,CL        Signed divide EA byte by 2, CL times
C0 /7 ib    SAR eb,ib      1 Signed divide EA byte by 2, ib times
D1 /7       SAR ev,1         Signed divide EA vword by 2, once
D3 /7       SAR ev,CL        Signed divide EA vword by 2, CL times
C1 /7 ib    SAR ev,ib      1 Signed divide EA vword by 2, ib times

1C ib       SBB AL,ib        Subtract with borrow immediate byte from AL
1D iv       SBB eAX,iv       Subtract with borrow immediate word from eAX
80 /3 ib    SBB eb,ib        Subtract with borrow immediate byte from EA byte
18 /r       SBB eb,rb        Subtract with borrow byte register from EA byte
83 /3 ib    SBB ev,ib        Subtract with borrow immediate byte from EA word
81 /3 iv    SBB ev,iv        Subtract with borrow immediate word from EA word
19 /r       SBB ev,rv        Subtract with borrow word register from EA word
1A /r       SBB rb,eb        Subtract with borrow EA byte from byte register
1B /r       SBB rv,ev        Subtract with borrow EA word from word register

AE          SCAS mb          Compare bytes AL - ES:[DI], advance DI
AF          SCAS mv          Compare vwords eAX - ES:[DI], advance DI
AE          SCASB            Compare bytes AL - ES:[DI], advance DI
AF          SCASD            Compare dwords EAX - ES:[DI], advance DI
AF          SCASW            Compare words AX - ES:[DI], advance DI

0F 14/0     SETBIT eb,CL   N Set bit CL of eb
0F 15/0     SETBIT ew,CL   N Set bit CL of ew
0F 1C/0 ib  SETBIT eb,ib   N Set bit ib of eb
0F 1D/0 ib  SETBIT ew,ib   N Set bit ib of ew
0F 9n /r    SETcond eb     3 Set eb byte to 1 if condition, 0 if not
0F 01 /0    SGDT m         2 Store 6-byte Global Descriptor Table register to M

D0 /4       SHL eb,1         Multiply EA byte by 2, once
D2 /4       SHL eb,CL        Multiply EA byte by 2, CL times
C0 /4 ib    SHL eb,ib      1 Multiply EA byte by 2, ib times
D1 /4       SHL ev,1         Multiply EA word by 2, once
D3 /4       SHL ev,CL        Multiply EA word by 2, CL times
C1 /4 ib    SHL ev,ib      1 Multiply EA word by 2, ib times

0F A5/r     SHLD ev,rv,CL  3 Set ev to high of ((ev,rv) SHL CL)
0F A4/r ib  SHLD ev,rv,ib  3 Set ev to high of ((ev,rv) SHL ib)

D0 /5       SHR eb,1         Unsigned divide EA byte by 2, once
D2 /5       SHR eb,CL        Unsigned divide EA byte by 2, CL times
C0 /5 ib    SHR eb,ib      1 Unsigned divide EA byte by 2, ib times
D1 /5       SHR ev,1         Unsigned divide EA word by 2, once
D3 /5       SHR ev,CL        Unsigned divide EA word by 2, CL times
C1 /5 ib    SHR ev,ib      1 Unsigned divide EA word by 2, ib times
0F AD/r     SHRD ev,rv,CL  3 Set ev to low of ((rv,ev) SHR CL)
0F AC/r ib  SHRD ev,rv,ib  3 Set ev to low of ((rv,ev) SHR ib)

0F 01 /1    SIDT m         2 Store 6-byte Interrupt Descriptor Table register to M
0F 00 /0    SLDT ew        2 Store Local Descriptor Table register to EA word
0F 01 /4    SMSW ew        2 Store Machine Status Word to EA word
36          SS               Use SS segment for the following memory reference
F9          STC              Set carry flag
FD          STD              Set direction flag so SI and DI will decrement
FB          STI              Set interrupt enable flag, interrupts enabled
```

```
0F 31/r     STOBITS rb,rb N Store AX to ES:DI,bit rb (incr. DI,rb), rb+1 bits
0F 39/0 ib  STOBITS rb,ib N Store AX to ES:DI,bit rb (incr. DI,rb), ib+1 bits
AA          STOS mb         Store AL to byte [DI], advance DI
AB          STOS mv         Store eAX to word [DI], advance DI
AA          STOSB           Store AL to byte ES:[DI], advance DI
AB          STOSD           Store EAX to dword ES:[DI], advance DI
AB          STOSW           Store AX to word ES:[DI], advance DI
0F 00 /1    STR ew        2 Store Task Register to EA word

2C ib       SUB AL,ib       Subtract immediate byte from AL
2D iv       SUB eAX,iv      Subtract immediate word from eAX
80 /5 ib    SUB eb,ib       Subtract immediate byte from EA byte
28 /r       SUB eb,rb       Subtract byte register from EA byte
83 /5 ib    SUB ev,ib       Subtract immediate byte from EA word
81 /5 iv    SUB ev,iv       Subtract immediate word from EA word
29 /r       SUB ev,rv       Subtract word register from EA word
2A /r       SUB rb,eb       Subtract EA byte from byte register
2B /r       SUB rv,ev       Subtract EA word from word register
0F 22       SUB4S         N Sub CL nibbles BCD, DS:SI - ES:DI (CL even,NZ)

A8 ib       TEST AL,ib      AND immediate byte into AL for flags only
A9 iv       TEST eAX,iv     AND immediate word into eAX for flags only
F6 /0 ib    TEST eb,ib      AND immediate byte into EA byte for flags only
84 /r       TEST eb,rb      AND byte register into EA byte for flags only
F7 /0 iv    TEST ev,iv      AND immediate word into EA word for flags only
85 /r       TEST ev,rv      AND word register into EA word for flags only
84 /r       TEST rb,eb      AND EA byte into byte register for flags only
85 /r       TEST rv,ev      AND EA word into word register for flags only

0F 10/0     TESTBIT eb,CL N Test bit CL of eb, set Z flag
0F 11/0     TESTBIT ev,CL N Test bit CL of ew, set Z flag
0F 18/0 ib  TESTBIT eb,ib N Test bit ib of eb, set Z flag
0F 19/0 ib  TESTBIT ew,ib N Test bit ib of ew, set Z flag

0F 00 /4    VERR ew       2 Set ZF=1 if segment can be read, selector ew
0F 00 /5    VERW ew       2 Set ZF=1 if segment can be written to, selector ew
9B          WAIT            Wait until BUSY pin is inactive (HIGH)
0F 09       WBINVD        4 Write Back and Invalidate the Data Cache
0F 30       WRMSR         5 Write EDXEAX to Model Specific Reg #ECX
0F C0 /r    XADD eb,rb    4 Exchange eb with rb then add into new eb
0F C1 /r    XADD ev,rv    4 Exchange ev with rv then add into new ev

9r          XCHG eAX,rv     Exchange word register with eAX
86 /r       XCHG eb,rb      Exchange byte register with EA byte
87 /r       XCHG ev,rv      Exchange word register with EA word
86 /r       XCHG rb,eb      Exchange EA byte with byte register
9r          XCHG rv,eAX     Exchange  with word register
87 /r       XCHG rv,ev      Exchange EA word with word register

D7          XLAT mb         Set AL to memory byte [BX + unsigned AL]
D7          XLATB           Set AL to memory byte DS:[BX + unsigned AL]
34 ib       XOR AL,ib       Exclusive-OR immediate byte into AL
35 iv       XOR eAX,iv      Exclusive-OR immediate word into eAX
80 /6 ib    XOR eb,ib       Exclusive-OR immediate byte into EA byte
30 /r       XOR eb,rb       Exclusive-OR byte register into EA byte
83 /6 ib    XOR ev,ib       Exclusive-OR immediate byte into EA word
81 /6 iv    XOR ev,iv       Exclusive-OR immediate word into EA word
31 /r       XOR ev,rv       Exclusive-OR word register into EA word
32 /r       XOR rb,eb       Exclusive-OR EA byte into byte register
33 /r       XOR rv,ev       Exclusive-OR EA word into word register
```

"N" next to the instruction description means that instruction works only on NEC chips. A digit x means that instruction works only on the x86 or later processor. See the note just before the chart.