up: Chapter 17 -- 80386 Instruction Set
prev: 17.1 Operand Size and Address-Size Attributes
next: AAA ASCII Adjust after Addition

17.2 Instruction Format

All instruction encodings are subsets of the general instruction format shown in Figure 17-1 . Instructions consist of optional instruction prefixes, one or two primary opcode bytes, possibly an address specifier consisting of the ModR/M byte and the SIB (Scale Index Base) byte, a displacement, if required, and an immediate data field, if required.

Smaller encoding fields can be defined within the primary opcode or opcodes. These fields define the direction of the operation, the size of the displacements, the register encoding, or sign extension; encoding fields vary depending on the class of operation.

Most instructions that can refer to an operand in memory have an addressing form byte following the primary opcode byte(s). This byte, called the ModR/M byte, specifies the address form to be used. Certain encodings of the ModR/M byte indicate a second addressing byte, the SIB (Scale Index Base) byte, which follows the ModR/M byte and is required to fully specify the addressing form.

Addressing forms can include a displacement immediately following either the ModR/M or SIB byte. If a displacement is present, it can be 8-, 16- or 32-bits.

If the instruction specifies an immediate operand, the immediate operand always follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

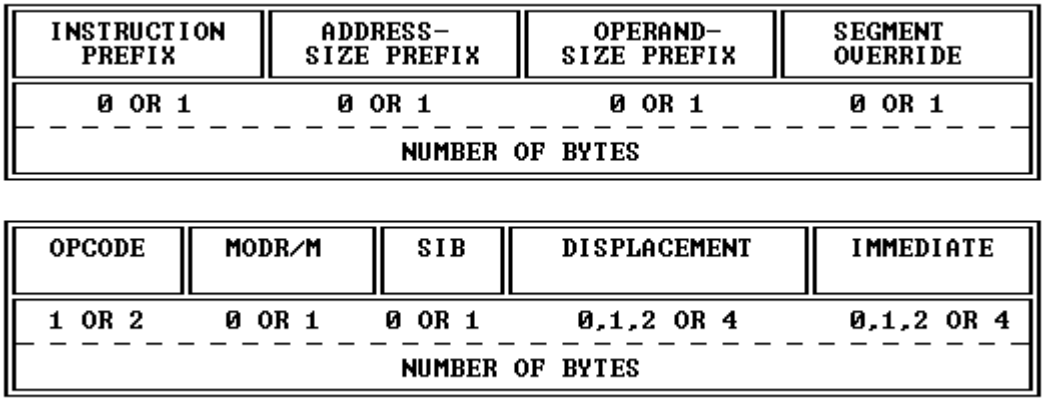
The following are the allowable instruction prefix codes:

```
F3H  REP  prefix (used only with string instructions)   F3H  REPE/REPZ  prefix (
```

The following are the segment override prefixes:

```
2EH    CS segment override prefix   36H    SS segment override prefix   3EH    D
```

Figure 17-1. 80386 Instruction Format



17.2.1 ModR/M and SIB Bytes

The ModR/M and SIB bytes follow the opcode byte(s) in many of the 80386 instructions. They contain the following information:

- The indexing type or register number to be used in the instruction
- The register to be used, or more information to select the instruction
- The base, index, and scale information

The ModR/M byte contains three fields of information:

- The mod field, which occupies the two most significant bits of the byte, combines with the r/m field to form 32 possible values: eight registers and 24 indexing modes
- The reg field, which occupies the next three bits following the mod field, specifies either a register number or three more bits of opcode information. The meaning of the reg field is determined by the first (opcode) byte of the instruction.
- The r/m field, which occupies the three least significant bits of the byte, can specify a register as the location of an operand, or can form part of the addressing-mode encoding in combination with the field as described above

The based indexed and scaled indexed forms of 32-bit addressing require the SIB byte. The presence of the SIB byte is indicated by certain encodings of the ModR/M byte. The SIB byte then includes the following fields:

- The ss field, which occupies the two most significant bits of the byte, specifies the scale factor
- The index field, which occupies the next three bits following the ss field and specifies the register number of the index register
- The base field, which occupies the three least significant bits of the byte, specifies the register number of the base register

Figure 17-2 shows the formats of the ModR/M and SIB bytes. The values and the corresponding addressing forms of the ModR/M and SIB bytes are shown in Tables 17-2, 17-3, and 17-4. The 16-bit addressing forms specified by the ModR/M byte are in Table 17-2. The 32-bit addressing forms specified by ModR/M are in Table 17-3. Table 17-4 shows the 32-bit addressing forms specified by the SIB byte

Figure 17-2. ModR/M and SIB Byte Formats

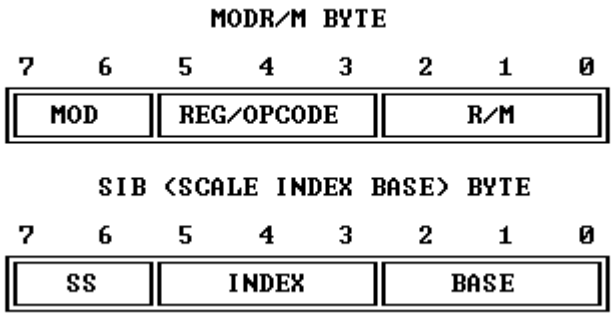


Table 17-2. 16-Bit Addressing Forms with the ModR/M Byter8(/r)

Notes

disp8 denotes an 8-bit displacement following the ModR/M byte, to be sign-extended and added to the index. disp16 denotes a 16-bit displacement following the ModR/M byte, to be added to the index. Default segment register is SS for the effective addresses containing a BP index, DS for other effective addresses.

```
Table 17-3. 32-Bit Addressing Forms with the ModR/M Byter8(/r)
```

Notes

[--] [--] means a SIB follows the ModR/M byte. disp8 denotes an 8-bit displacement following the SIB byte, to be sign-extended and added to the index. disp32 denotes a 32-bit displacement following the ModR/M byte, to be added to the index.

```
Table 17-4. 32-Bit Addressing Forms with the SIB Byte   r32
```

Notes

[*] means a disp32 with no base if MOD is 00, [ESP] otherwise. This provides the following addressing modes:

```
disp32[index]        (MOD=00)      disp8[EBP][index]    (MOD=01)      disp32[EBP
```

17.2.2 How to Read the Instruction Set Pages

The following is an example of the format used for each 80386 instruction description in this chapter:

CMC -- Complement Carry Flag

```
Opcode   Instruction         Clocks      DescriptionF5 CMC  2              Comple
```

The above table is followed by paragraphs labelled "Operation," "Description," "Flags Affected," "Protected Mode Exceptions," "Real Address Mode Exceptions," and, optionally, "Notes." The following sections explain the notational conventions and abbreviations used in these paragraphs of the instruction descriptions.

17.2.2.1 Opcode

The "Opcode" column gives the complete object code produced for each form of the instruction. When possible, the codes are given as hexadecimal bytes, in the same order in which they appear in memory. Definitions of entries other than hexadecimal bytes are as follows:

/digit:
: (digit is between 0 and 7) indicates that the ModR/M byte of the instruction uses only the r/m (register or memory) operand. The reg field contains the digit that provides an extension to the instruction's opcode.

/r:
: indicates that the ModR/M byte of the instruction contains both a register operand and an r/m operand.

cb, cw, cd, cp:
: a 1-byte (cb), 2-byte (cw), 4-byte (cd) or 6-byte (cp) value following the opcode that is used to specify a code offset and possibly a new value for the code segment register.

ib, iw, id:
: a 1-byte (ib), 2-byte (iw), or 4-byte (id) immediate operand to the instruction that follows the opcode, ModR/M bytes or scale-indexing bytes. The opcode determines if the operand is a signed value. All words and doublewords are given with the low-order byte first.

+rb, +rw, +rd:
: a register code, from 0 through 7, added to the hexadecimal byte given at the left of the plus

sign to form a single opcode byte. The codes are

```
rb          rw          rd     AL = 0      AX = 0      EAX = 0     CL = 1      CX =
```

17.2.2.2 Instruction

The "Instruction" column gives the syntax of the instruction statement as it would appear in an ASM386 program. The following is a list of the symbols used to represent operands in the instruction statements:

rel8:
: a relative address in the range from 128 bytes before the end of the instruction to 127 bytes after the end of the instruction.

rel16, rel32:
: a relative address within the same code segment as the instruction assembled. rel16 applies to instructions with an operand-size attribute of 16 bits; rel32 applies to instructions with an operand-size attribute of 32 bits.

ptr16:16, ptr16:32:
: a FAR pointer, typically in a code segment different from that of the instruction. The notation 16:16 indicates that the value of the pointer has two parts. The value to the right of the colon is a 16-bit selector or value destined for the code segment register. The value to the left corresponds to the offset within the destination segment. ptr16:16 is used when the instruction's operand-size attribute is 16 bits; ptr16:32 is used with the 32-bit attribute.

r8:
: one of the byte registers AL, CL, DL, BL, AH, CH, DH, or BH.

r16:
: one of the word registers AX, CX, DX, BX, SP, BP, SI, or DI.

r32:
: one of the doubleword registers EAX, ECX, EDX, EBX, ESP, EBP, ESI, or EDI.

imm8:
: an immediate byte value. imm8 is a signed number between -128 and +127 inclusive. For instructions in which imm8 is combined with a word or doubleword operand, the immediate value is sign-extended to form a word or doubleword. The upper byte of the word is filled with the topmost bit of the immediate value.

imm16:
: an immediate word value used for instructions whose operand-size attribute is 16 bits. This is a number between -32768 and +32767 inclusive.

imm32:
: an immediate doubleword value used for instructions whose operand-size attribute is 32-bits. It allows the use of a number between +2147483647 and -2147483648.

r/m8:
: a one-byte operand that is either the contents of a byte register (AL, BL, CL, DL, AH, BH, CH, DH), or a byte from memory.

r/m16:
: a word register or memory operand used for instructions whose operand-size attribute is 16 bits. The word registers are: AX, BX, CX, DX, SP, BP, SI, DI. The contents of memory are found at the address provided by the effective address computation.

r/m32:
: a doubleword register or memory operand used for instructions whose operand-size attribute is 32-bits. The doubleword registers are: EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI. The contents of memory are found at the address provided by the effective address computation.

m8:
: a memory byte addressed by DS:SI or ES:DI (used only by string instructions).

m16:

a memory word addressed by DS:SI or ES:DI (used only by string instructions).

m32:
a memory doubleword addressed by DS:SI or ES:DI (used only by string instructions).

m16:16, M16:32:
a memory operand containing a far pointer composed of two numbers. The number to the left of the colon corresponds to the pointer's segment selector. The number to the right corresponds to its offset.

m16 & 32, m16 & 16, m32 & 32:
a memory operand consisting of data item pairs whose sizes are indicated on the left and the right side of the ampersand. All memory addressing modes are allowed. m16 & 16 and m32 & 32 operands are used by the BOUND instruction to provide an operand containing an upper and lower bounds for array indices. m16 & 32 is used by LIDT and LGDT to provide a word with which to load the limit field, and a doubleword with which to load the base field of the corresponding Global and Interrupt Descriptor Table Registers.

moffs8, moffs16, moffs32:
(memory offset) a simple memory variable of type BYTE, WORD, or DWORD used by some variants of the MOV instruction. The actual address is given by a simple offset relative to the segment base. No ModR/M byte is used in the instruction. The number shown with moffs indicates its size, which is determined by the address-size attribute of the instruction.

Sreg:
a segment register. The segment register bit assignments are ES=0, CS=1, SS=2, DS=3, FS=4, and GS=5.

17.2.2.3 Clocks

The "Clocks" column gives the number of clock cycles the instruction takes to execute. The clock count calculations makes the following assumptions:

- The instruction has been prefetched and decoded and is ready for execution.
- Bus cycles do not require wait states.
- There are no local bus HOLD requests delaying processor access to the bus.
- No exceptions are detected during instruction execution.
- Memory operands are aligned.

Clock counts for instructions that have an r/m (register or memory) operand are separated by a slash. The count to the left is used for a register operand; the count to the right is used for a memory operand.

The following symbols are used in the clock count specifications:

- n, which represents a number of repetitions.
- m, which represents the number of components in the next instruction executed, where the entire displacement (if any) counts as one component, the entire immediate data (if any) counts as one component, and every other byte of the instruction and prefix(es) each counts as one component.
- pm=, a clock count that applies when the instruction executes in Protected Mode. pm= is not given when the clock counts are the same for Protected and Real Address Modes.

When an exception occurs during the execution of an instruction and the exception handler is in another task, the instruction execution time is increased by the number of clocks to effect a task switch. This parameter depends on several factors:

- The type of TSS used to represent the current task (386 TSS or 286 TSS).
- The type of TSS used to represent the new task.

- Whether the current task is in V86 mode.
- Whether the new task is in V86 mode.

Table 17-5 summarizes the task switch times for exceptions.

```
Table 17-5. Task Switch Times for Exceptions                    New TaskOld
```

17.2.2.4 Description

The "Description" column following the "Clocks" column briefly explains the various forms of the instruction. The "Operation" and "Description" sections contain more details of the instruction's operation.

17.2.2.5 Operation

The "Operation" section contains an algorithmic description of the instruction which uses a notation similar to the Algol or Pascal language. The algorithms are composed of the following elements:

- Comments are enclosed within the symbol pairs "(*" and "*)".
- Compound statements are enclosed between the keywords of the "if" statement (IF, THEN, ELSE, FI) or of the "do" statement (DO, OD), or of the "case" statement (CASE ... OF, ESAC).
- A register name implies the contents of the register. A register name enclosed in brackets implies the contents of the location whose address is contained in that register. For example, ES:[DI] indicates the contents of the location whose ES segment relative address is in register DI. [SI] indicates the contents of the address contained in register SI relative to SI's default segment (DS) or overridden segment.
- Brackets also used for memory operands, where they mean that the contents of the memory location is a segment-relative offset. For example, [SRC] indicates that the contents of the source operand is a segment-relative offset.
- A := B; indicates that the value of B is assigned to A.
- The symbols =, <>, >=, and <= are relational operators used to compare two values, meaning equal, not equal, greater or equal, less or equal, respectively. A relational expression such as A = B is TRUE if the value of A is equal to B; otherwise it is FALSE.

The following identifiers are used in the algorithmic descriptions:

- OperandSize represents the operand-size attribute of the instruction, which is either 16 or 32 bits. AddressSize represents the address-size attribute, which is either 16 or 32 bits. For example,

  ```
  IF instruction = CMPSW   THEN OperandSize ? 16;   ELSE      IF instruction =
  ```

  indicates that the operand-size attribute depends on the form of the CMPS instruction used. Refer to the explanation of address-size and operand-size attributes at the beginning of this chapter for general guidelines on how these attributes are determined.
- StackAddrSize represents the stack address-size attribute associated with the instruction, which has a value of 16 or 32 bits, as explained earlier in the chapter.
- SRC represents the source operand. When there are two operands, SRC is the one on the right.
- DEST represents the destination operand. When there are two operands, DEST is the one on the left.
- LeftSRC, RightSRC distinguishes between two operands when both are source operands.
- eSP represents either the SP register or the ESP register depending on the setting of the B-bit for the current stack segment.

The following functions are used in the algorithmic descriptions:

- Truncate to 16 bits(value) reduces the size of the value to fit in 16 bits by discarding the uppermost bits as needed.
- Addr(operand) returns the effective address of the operand (the result of the effective address calculation prior to adding the segment base).
- ZeroExtend(value) returns a value zero-extended to the operand-size attribute of the instruction. For example, if OperandSize = 32, ZeroExtend of a byte value of -10 converts the byte from F6H to doubleword with hexadecimal value 000000F6H. If the value passed to ZeroExtend and the operand-size attribute are the same size, ZeroExtend returns the value unaltered.
- SignExtend(value) returns a value sign-extended to the operand-size attribute of the instruction. For example, if OperandSize = 32, SignExtend of a byte containing the value -10 converts the byte from F6H to a doubleword with hexadecimal value FFFFFFF6H. If the value passed to SignExtend and the operand-size attribute are the same size, SignExtend returns the value unaltered.
- Push(value) pushes a value onto the stack. The number of bytes pushed is determined by the operand-size attribute of the instruction. The action of Push is as follows:

```
IF StackAddrSize = 16   THEN      IF OperandSize = 16      THEN         SP
```

- Pop(value) removes the value from the top of the stack and returns it. The statement EAX ? Pop( ); assigns to EAX the 32-bit value that Pop took from the top of the stack. Pop will return either a word or a doubleword depending on the operand-size attribute. The action of Pop is as follows:

```
IF StackAddrSize = 16   THEN      IF OperandSize = 16      THEN         ret
```

- Bit[BitBase, BitOffset] returns the address of a bit within a bit string, which is a sequence of bits in memory or a register. Bits are numbered from low-order to high-order within registers and within memory bytes. In memory, the two bytes of a word are stored with the low-order byte at the lower address.

  If the base operand is a register, the offset can be in the range 0..31. This offset addresses a bit within the indicated register. An example, "BIT[EAX, 21]," is illustrated in Figure 17-3 .

  If BitBase is a memory address, BitOffset can range from -2 gigabits to 2 gigabits. The addressed bit is numbered (Offset MOD 8) within the byte at address (BitBase + (BitOffset DIV 8)), where DIV is signed division with rounding towards negative infinity, and MOD returns a positive number. This is illustrated in Figure 17-4 .

- I-O-Permission(I-O-Address, width) returns TRUE or FALSE depending on the I/O permission bitmap and other factors. This function is defined as follows:

```
IF TSS type is 286 THEN RETURN FALSE; FI;   Ptr ? [TSS + 66]; (* fetch bitm
```

- Switch-Tasks is the task switching function described in Chapter 7 .

17.2.2.6 Description

The "Description" section contains further explanation of the instruction's operation.

Figure 17-3. Bit Offset for BIT[EAX, 21]

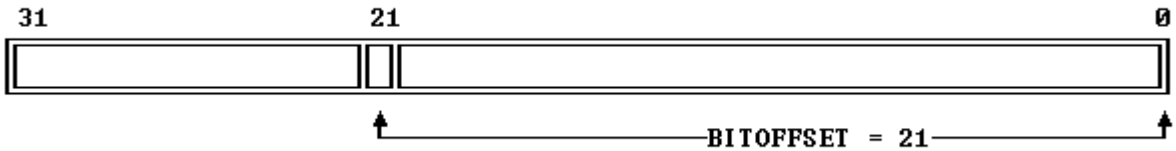
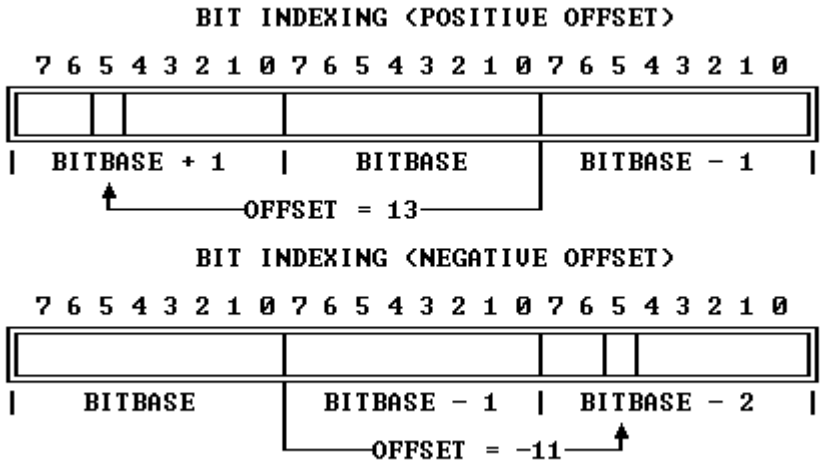


Figure 17-4. Memory Bit Indexing



17.2.2.7 Flags Affected

The "Flags Affected" section lists the flags that are affected by the instruction, as follows:

- If a flag is always cleared or always set by the instruction, the value is given (0 or 1) after the flag name. Arithmetic and logical instructions usually assign values to the status flags in the uniform manner described in Appendix C . Nonconventional assignments are described in the "Operation" section.
- The values of flags listed as "undefined" may be changed by the instruction in an indeterminate manner.

All flags not listed are unchanged by the instruction.

17.2.2.8 Protected Mode Exceptions

This section lists the exceptions that can occur when the instruction is executed in 80386 Protected Mode. The exception names are a pound sign (#) followed by two letters and an optional error code in parentheses. For example, #GP(0) denotes a general protection exception with an error code of 0. Table 17-6 associates each two-letter name with the corresponding interrupt number.

Chapter 9 describes the exceptions and the 80386 state upon entry to the exception.

Application programmers should consult the documentation provided with their operating systems to determine the actions taken when exceptions occur.

```
Table 17-6. 80386 ExceptionsMnemonic     Interrupt    Description#UD
```

17.2.2.9 Real Address Mode Exceptions

Because less error checking is performed by the 80386 in Real Address Mode, this mode has

fewer exception conditions . Refer to Chapter 14 for further information on these exceptions.

17.2.2.10 Virtual-8086 Mode Exceptions

Virtual 8086 tasks provide the ability to simulate Virtual 8086 machines. Virtual 8086 Mode exceptions are similar to those for the 8086 processor, but there are some differences . Refer to Chapter 15 for details .

---

up: Chapter 17 -- 80386 Instruction Set
prev: 17.1 Operand Size and Address-Size Attributes
next: AAA ASCII Adjust after Addition