- Pushes Instruction Pointer (containing the address of the next instruction) onto the stack.
- Performs the required operation.

To return from the interrupt, the operation issues an IRET (Interrupt Return), which pops the registers off the stack. The restored CS.IP causes a return to the instruction immediately following the INT. Note that in 80386 + processors the CS:EIP is pushed into the stack SS:ESP.

Because the preceding process is entirely automatic, your only concerns are to define a stack large enough for the necessary pushing and popping and to use the appropriate INT operations.

ADDRESSING MODES

An operand address provides a source of data for an instruction to process. Some instructions, such as CLC and RET, do not require an operand, whereas other instructions may have one, two, or three operands. Where there are two operands, the first operand is the *destination,* which contains data in a register or in memory, and which is to be processed. The second operand is the *source,* which contains either the data to be delivered (immediate) or the address (in memory or of a register) of the data. The source data for most instructions is unchanged by the operation. The three basic modes of addressing are register, immediate, and memory; memory addressing consists of six types, for eight modes in all.

1. Register Addressing

For this mode, a register provides the name of any of the 8-, 16-, or 32-bit registers. Depending on the instruction, the register may appear in the first operand, the second operand, or both, as the following examples illustrate:

```
MOV DX,WORD_MEM     ;Register in first operand
MOV WORD_MEM,CX     ;Register in second operand
MOV EDX,EBX         ;Registers in both operands
```

Because processing data between registers involves no reference to memory, it is the fastest type of operation.

2. Immediate Addressing

An immediate operand contains a constant value or an expression. Here are some examples of valid immediate constants:

```
Hexadecimal:      0148H
Decimal:          328 (which the assembler converts to 0148H)
Binary:           101001000B (which converts to 0148H)
```

For many instructions with two operands, the first operand may be a register or memory location, and the second may be an immediate constant. The destination field (first operand) defines the length of the data. Here are some examples:

```
BYTE_VAL DB 150              ;Define byte
WORD_VAL DW 300              ; word
DBWD_VAL DD 0                ; doubleword
```

```
...
SUB BYTE_VAL, 50     ;Immediate to memory (byte)
MOV WORD_VAL, 40H    ;Immediate to memory (word)
MOV DBWD_VAL, 0      ;Immediate to memory (doubleword)
MOV AX, 0245H        ;Immediate to register (word)
```

The instruction in the last example moves the immediate constant 0245H to AX. The 3-byte object code is B84502, where B8 means "move an immediate value to AX" and the following two bytes contain the value itself (4502H, in reverse-byte sequence).

The use of an immediate operand provides faster processing than defining a numeric constant in the data segment and referencing it in an operand. The data is embedded with the code in this and thus available in the code segment than in the data segment.

The length of an immediate constant cannot exceed the length defined by the first operand. In the following invalid example, the immediate operand is two bytes, but AL is only one byte:

```
MOV AL,0245H     ;Invalid immediate length
```

However, if an immediate operand is shorter than a receiving operand, as in

```
ADD AX,48H     ;Valid immediate length
```

the assembler expands the immediate operand to two bytes, 0048H, and stores it in object code as 4800H.

3. Direct Memory Addressing

In this format, one of the operands references a memory location and the other operand references a register. (The only instructions that allow both operands to address memory directly are MOVS and CMPS.) DS is the default segment register for addressing data in memory, as DS:offset. Here are some examples:

```
ADD BYTE_VAL, DL   ;Add register to memory (byte)
MOV BX,WORD_VAL    ;Move memory to register (word)
```

4. Direct-Offset Addressing

This addressing mode, a variation of direct addressing, uses arithmetic operators to modify an address. The following examples use these definitions of tables:

```
BYTE_TBL DB 12, 15, 16, 22,  ...  ;Table of bytes
WORD_TBL DB 163, 227, 485,   ...  ;Table of words
DBWD_TBL DB 465, 563, 897,   ...  ;Table of doublewords
```

Byte Operations. These instructions access bytes from BYTE_TBL:

```
MOV CL,BYTE_TBL[2]    ;Get byte from BYTE_TBL
MOV CL,BYTE_TBL+2     ;Same operation
```

The first MOV uses an arithmetic operator to access the third byte (16) from BYTE__TBL. (BYTE_TBL[0] is the first byte, BYTE_TBL[1] the second, and BYTE_TBL[2] the third.) The second MOV uses a plus (+) operator for exactly the same effect.

Word Operations. These instructions access words from WORD_TBL:

```
MOV CX,WORD_TBL[4]      ;Get word from WORD_TBL
MOV CX,W0RD_TBL+4       ;Same operation
```

The MOVs access the third word of WORD_TBL. (WORD_TBL[0] is the first word, WORD_TBL[2] the second, and WORD_TBL[4] the third.)

Doubleword Operations. These instructions access doublewords from DBWD_TBL:

```
MOV CX,DBWD_TBL[8]      ;Get doubleword from DBWD_TBL
MOV CX,DBWD_TBL+8       ;Same operation
```

The MOVs access the third doubleword of DBWD_TBL. (DBWD_TBL[0] is the first doubleword, DBWD_TBL[4] the second, and DBWD_TBL[8] the third.)

5. Indirect Memory Addressing

Indirect addressing takes advantage of the computer's capability for segment:offset addressing. The registers used for this purpose are base registers (BX and BP) and index registers (DI and SI), coded within square brackets, which indicate a reference to memory. If you code the .386, .486, or .586 directive, you can also use any of the general purpose registers (EAX, EBX, ECX, and EDX) for indirect addressing.

An indirect address such as [DI] tells the assembler that the memory address to use will be in DI when the program subsequently executes. BX, DI, and SI are associated with DS as DS:BX, DS:DI, and DS:SI, for processing data in the data segment. BP is associated with SS as SS:BP, for handling data in the stack.

When the first operand contains an indirect address, the second operand references a register or immediate value; when the second operand contains an indirect address, the first operand references a register. Note that a reference in square brackets to BP, BX, DI, or SI implies an indirect operand, and the processor treats the contents of the register as an offset address when the program is executing.

In the following example, LEA first initializes BX with the offset address of DATA_VAL. MOV then uses the address now in BX to store CL in the memory location to which it points, in this case, DATA_VAL:

```
DATA_VAL DB 50              ;Define byte
         ...
         LEA BX,DATA_VAL    ;Load BX with offset
         MOV [BX],CL        ;Move CL to DATA_VAL
```

The effect of the two MOVs is the same as coding MOV DATA_VAL,25, although the uses for indexed addressing are usually not so trivial. Here are a few more examples of indirect operands:

```
         ADD CL, [BX]           ;2nd operand = DS:BX
         MOV BYTE PTR [DI],25   ;1st operand = DS:DI
         ADD [BP],CL            ;1st operand = SS:BP
.386
         MOV DX,[EAX]           ;2nd operand = DS:EAX
```

The next example uses an absolute value for an offset:

```
MOV CX,DS:[38B0H]  ;Word in memory at offset 38B0H
```

6. Base Displacement Addressing

This addressing mode also uses base registers (BX and BP) and index registers (DI and SI), but combined with a displacement (a number or offset value) to form an effective address. The following MOV instruction moves zero to a location two bytes immediately following the start of DATA_TBL:

```
DATA_TBL DB 365 DUP(?)          ;Define bytes
         ...
         LEA BX,DATA_TBL        ;Load BX with offset
         MOV BYTE PTR [BX+2],0  ;Move 0 to DATA_TBL+2
```

And here are some additional examples:

```
         ADD CL,[DI+12]          ;DI offset plus 12 (or 12[DI])
         SUB DATA_TBL[SI],25     ;SI contains offset (0-364)
         MOV DATA_TBL[DI],DL     ;DI contains offset (0-364)
.386
         MOV DX,[EAX+4]          ;EAX offset plus 4
         ADD DATA_TBL[EDX],CL    ;EDX + offset DATA_TBL
```

7. Base-Index Addressing

This addressing mode combines a base register (BX or BP) with an index register (DI or SI) to form an effective address: for example, [BX+DI] means the address in BX plus the address in DI. A common use for this mode is in addressing a 2-dimensional array, where, say, BX references the row and SI the column. Here are some examples:

```
MOV AX, [BX+SI]          ;Move word from memory
ADD [BX+DI],CL           ;Add byte to memory
```

8. Base-Index with Displacement Addressing

This addressing mode, a variation on base-index, combines a base register, an index register, and a displacement to form an effective address. Here are some examples:

```
MOV AX, [BX+DI+10]       ;or 10[BX+DI]
MOV CL,DATA_TBL[BX+DI]   ;or [BX+DI+DATA_TBL]
```

9. Scale-Index-Base Addressing

In this addressing mode the *scale* field specifies the scale factor (2, 4,8) , *index* specifies the index register (EAX, EBX ,DI/SI etc) and the *base* field specifies the base register (BX, BP).

```
.386
MOV EBX, [ECX*2 +ESP+4]
```

This example moves into EBX the contents of (ECX x 2) plus the contents of (ESP + 4).