Table 4.1 Data-Defining Pseudo-ops

Pseudo-op	*Stands for*
DB	define byte
DW	define word
DD	define doubleword (two consecutive words)
DQ	define quadword (four consecutive words)
DT	define tenbytes (ten consecutive bytes)

4.3 Variables

Variables play the same role in assembly language that they do in high-level languages. Each variable has a data type and is assigned a memory address by the program. The data-defining pseudo-ops and their meanings are listed in Table 4.1. Each pseudo-op can be used to set aside one or more data items of the given type.

In this section we use DB and DW to define byte variables, word variables, and arrays of bytes and words. The other data-defining pseudo-ops are used in Chapter 18 in connection with multiple-precision and noninteger operations.

4.3.1 Byte Variables

The assembler directive that defines a byte variable takes the following form:

```
name   DB              initial_value
```

where the pseudo-op DB stands for "Define Byte".

For example,

```
ALPHA   DB             4
```

This directive causes the assembler to associate a memory byte with the name ALPHA, and initialize it to 4. A question mark ("?") used in place of an initial value sets aside an uninitialized byte; for example,

```
BYT   DB               ?
```

The decimal range of initial values that can be specified is -128 to 127 if a signed interpretation is being given, or 0 to 255 for an unsigned interpretation. These are the ranges of values that fit in a byte.

4.3.2 Word Variables

The assembler directive for defining a word variable has the following form:

```
name   DW              initial_value
```

The pseudo-op DW means "Define Word." For example,

```
WRD   DW          -2
```

as with byte variables, a question mark in place of an initial value means an uninitialized word. The decimal range of initial values that can be specified is -32768 to 32767 for a signed interpretation, or 0 to 65535 for an unsigned interpretation.

4.3.3 *Arrays*

In assembly language, an **array** is just a sequence of memory bytes or words. For example, to define a three-byte array called B_ARRAY, whose initial values are 10h, 20h, and 30h, we can write,

```
B_ARRAY        DB            10H,20H,30H
```

The name B_ARRAY is associated with the first of these bytes, B_ARRAY+1 with the second, and B_ARRAY+2 with the third. If the assembler assigns the offset address 0200h to B_ARRAY, then memory would look like this:

Symbol	*Address*	*Contents*
B_ARRAY	200h	10h
B_ARRAY+1	201h	20h
B_ARRAY+2	202h	30h

In the same way, an array of words may be defined. For example,

```
W_ARRAY        DW            1000,40,29887,329
```

sets up an array of four words, with initial values 1000, 40, 29887, and 329. The initial word is associated with the name W_ARRAY, the next one with W_ARRAY + 2, the next with W_ARRAY + 4, and so on. If the array starts at 0300h, it will look like this:

Symbol	*Address*	*Contents*
W_ARRAY	0300h	1000d
W_ARRAY+2	0302h	40d
W_ARRAY+4	0304	29887d
W_ARRAY+6	0306h	329d

High and Low Bytes of a Word

Sometimes we need to refer to the high and low bytes of a word variable. Suppose we define

```
WORD1          DW            1234H
```

The low byte of WORD1 contains 34h, and the high byte contains 12h. The low byte has symbolic address WORD1, and the high byte has symbolic address WORD1+1.

Character Strings

An array of ASCII codes can be initialized with a string of characters. For example,

```
LETTERS          DB               'ABC'
```

is equivalent to

```
LETTERS          DB               41H,42H,43H
```

Inside a string, the assembler differentiates between upper and lower case. Thus, the string "abc" is translated into three bytes with values 61h, 62h, and 63h.

It is possible to combine characters and numbers in one definition; for example,

```
MSG DB              'HELLO',0AH,0DH,'$'
```

is equivalent to

```
MSG DB              48H,45H,4CH,4CH,4FH,0AH,0DH,24H
```

4.4 Named Constants

To make assembly language code easier to understand, it is often desirable to use a symbolic name for a constant quantity.

EQU (Equates)

To assign a name to a constant, we can use the **EQU** (equates) pseudo-op. The syntax is

```
name             EQU              constant
```

For example, the statement

```
LF               EQU              0AH
```

assigns the name LF to 0Ah, the ASCII code of the line feed character. The name LF may now be used in place of 0Ah anywhere in the program. Thus, the assembler translates the instructions

```
MOV  DL,0AH
```

and

```
MOV  DL,LF
```

into the same machine instruction.

The symbol on the right of an EQU can also be a string. For example,

```
PROMPT  EQU  'TYPE  YOUR  NAME'
```

Then instead of

```
MSG  DB        'TYPE YOUR  NAME'
```

we could say

```
MSG  DB              PROMPT
```

Note: no memory is allocated for EOU names.