SIZE MISMATCH ERRORS

If we change the directive in the last example from DW to DB, the program will still assemble and debug without errors; however, the results will not be as expected. When the first operand is read, 0A05 will be placed in the register, which is actually two operands placed in consecutive byte memory locations. The second number will be read as 000F, which is the zero byte of num4 appended to the 15 of num3. The third number will be junk, depending on the current state of the machine.

According to our data declaration, the third number should be at 0114, but it is accessed at 011D (calculated with word offsets). This is a logical error. The declarations and their access must remain synchronized, which is the programmer's responsibility, not the assembler's.

The assembler allows the programmer to do almost anything, as long as it can run on the processor. The assembler prevents illegal instructions that the processor cannot execute, but it won't stop logical errors.

For example, the programmer must access data as a word if declared as a word, and as a byte if declared as a byte. Here's how to handle byte-sized data in assembly:

Example 2.5:

```
001 ; a program to add three numbers using byte variables
002 [org 0x0100]
003 mov al, [num1] ; load first number in AL
004 mov bl, [num1+1] ; load second number in BL
005 add al, bl ; accumulate sum in AL
```

```
006 mov bl, [num1+2] ; load third number in BL
007 add al, bl ; accumulate sum in AL
008 mov [num1+3], al ; store sum at num1+3
009 mov ax, 0x4c00 ; terminate program
010 int 0x21
011 num1: db 5, 10, 15, 0
```

In this example, we declare byte variables using "db" instead of "dw" to ensure that each data entry only occupies one byte. The AL register is used to load and process each byte separately.

The processor relies on the register sizes to determine the correct size of the data movement operation. For instance, when using "mov ax, [num1]," the AX register indicates that the memory is accessed as a word, while "mov al, [num1]" tells the processor that it should be accessed as a byte.

Similarly, you can't mix registers of different sizes, such as using "mov ax, bl." The assembler detects this as illegal since the sizes don't match.

However, using ambiguous instructions such as "mov [num1], 5" leaves the assembler unsure whether to treat the value as a byte or word. To clarify, we can specify the size using:

```
mov byte [num1], 5
mov word [num1], 5
```

Assembly language provides a lot of flexibility and power, but it also requires the programmer

to be responsible for maintaining data alignment and accessing the correct sizes.

Always ensure you access data in the way it was declared to avoid logical errors.