Figure 1.2 contains examples of data movement operations for SIC and SIC/XE. There are no memory-to-memory move instructions; thus, all data movement must be done using registers. Figure 1.2(a) shows two examples of data movement. In the first, a 3-byte word is moved by loading it into register A and then storing the register at the desired destination. Exactly the same thing could be accomplished using register X (and the instructions LDX, STX) or register L (LDL, STL). In the second example, a single byte of data is moved using the instructions LDCH (Load Character) and STCH (Store Character). These instructions operate by loading or storing the rightmost 8-bit byte of register A; the other bits in register A are not affected.

Figure 1.2(a) also shows four different ways of defining storage for data items in the SIC assembler language. (These assembler directives are discussed in more detail in Section 2.1.) The statement WORD reserves one word of storage, which is initialized to a value defined in the operand field of the statement. Thus the WORD statement in Fig. 1.2(a) defines a data word labeled FIVE whose value is initialized to 5. The statement RESW reserves one or

```
        LDA     FIVE        LOAD CONSTANT 5 INTO REGISTER A
        STA     ALPHA       STORE IN ALPHA
        LDCH    CHARZ       LOAD CHARACTER 'Z' INTO REGISTER A
        STCH    C1          STORE IN CHARACTER VARIABLE C1
         .
         .
         .
ALPHA   RESW    1           ONE-WORD VARIABLE
FIVE    WORD    5           ONE-WORD CONSTANT
CHARZ   BYTE    C'Z'        ONE-BYTE CONSTANT
C1      RESB    1           ONE-BYTE VARIABLE
```

(a)

```
        LDA     #5          LOAD VALUE 5 INTO REGISTER A
        STA     ALPHA       STORE IN ALPHA
        LDA     #90         LOAD ASCII CODE FOR 'Z' INTO REG A
        STCH    C1          STORE IN CHARACTER VARIABLE C1
         .
         .
         .
ALPHA   RESW    1           ONE-WORD VARIABLE
C1      RESB    1           ONE-BYTE VARIABLE
```

(b)

Figure 1.2 Sample data movement operations for (a) SIC and (b) SIC/XE.

more words of storage for use by the program. For example, the RESW statement in Fig. 1.2(a) defines one word of storage labeled ALPHA, which will be used to hold a value generated by the program.

The statements BYTE and RESB perform similar storage-definition functions for data items that are characters instead of words. Thus in Fig. 1.2(a) CHARZ is a 1-byte data item whose value is initialized to the character "Z", and C1 is a 1-byte variable with no initial value.

The instructions shown in Fig. 1.2(a) would also work on SIC/XE; however, they would not take advantage of the more advanced hardware features available. Figure 1.2(b) shows the same two data-movement operations as they might be written for SIC/XE. In this example, the value 5 is loaded into register A using immediate addressing. The operand field for this instruction contains the flag # (which specifies immediate addressing) and the data value to be loaded. Similarly, the character "Z" is placed into register A by using immediate addressing to load the value 90, which is the decimal value of the ASCII code that is used internally to represent the character "Z".

Figure 1.3(a) shows examples of arithmetic instructions for SIC. All arithmetic operations are performed using register A, with the result being left in register A. Thus this sequence of instructions stores the value (ALPHA + INCR – 1) in BETA and the value (GAMMA + INCR – 1) in DELTA.

Figure 1.3(b) illustrates how the same calculations could be performed on SIC/XE. The value of INCR is loaded into register S initially, and the register-to-register instruction ADDR is used to add this value to register A when it is needed. This avoids having to fetch INCR from memory each time it is used in a calculation, which may make the program more efficient. Immediate addressing is used for the constant 1 in the subtraction operations.

Looping and indexing operations are illustrated in Fig. 1.4. Figure 1.4(a) shows a loop that copies one 11-byte character string to another. The index register (register X) is initialized to zero before the loop begins. Thus, during the first execution of the loop, the target address for the LDCH instruction will be the address of the first byte of STR1. Similarly, the STCH instruction will store the character being copied into the first byte of STR2. The next instruction, TIX, performs two functions. First it adds 1 to the value in register X, and then it compares the new value of register X to the value of the operand (in this case, the constant value 11). The condition code is set to indicate the result of this comparison. The JLT instruction jumps if the condition code is set to "less than." Thus, the JLT causes a jump back to the beginning of the loop if the new value in register X is less than 11.

During the second execution of the loop, register X will contain the value 1. Thus, the target address for the LDCH instruction will be the second byte of STR1, and the target address for the STCH instruction will be the second byte of STR2. The TIX instruction will again add 1 to the value in register X, and the