PROGRAM DESCRIPTION

This I/O design using the 8255A in Mode 1 allows two operations: outputting to the printer and data entry through the keyboard. The printer interfacing is designed with the status check and the keyboard interfacing with the interrupt.

In the PRINT subroutine, the character is placed in the accumulator, and the status is read by the instruction IN FEH. Initially, port B is empty, bit PC_1 ($\overline{OBF_B}$) is high, and the instruction OUT FDH sends the first character to port B. The rising edge of the $\overline{WR}$ signal sets signal $\overline{OBF}$ low, indicating the presence of a data byte in port B, which is sent out to the printer (Figure 15.12). After receiving a character, the printer sends back an acknowledge signal ($\overline{ACK}$), which in turn sets $\overline{OBF_B}$ high, indicating that port B is ready for the next character, and the PRINT subroutine continues.

If a key is pressed during the PRINT, a data byte is transmitted to port A and the $\overline{STB_A}$ goes low, which sets IBF_A high. The initialization routine should set the $INTE_A$ flip-flop. When the $\overline{STB_A}$ goes high, all the conditions (i.e., $IBF_A = 1$, $INTE_A = 1$) to generate $INTR_A$ are met. This signal, which is connected to the RST 6.5, interrupts the MPU, and the program control is transferred to the service routine. This service routine would read the contents of port A. enable the interrupts, and return to the PRINT routine (the interrupt service routine is not shown here).

15.1.7 Mode 2: Bidirectional Data Transfer

This mode is used primarily in applications such as data transfer between two computers or floppy disk controller interface. In this mode, port A can be configured as the bidirectional port and port B either in Mode 0 or Mode 1. Port A uses five signals from port C as handshake signals for data transfer. The remaining three signals from port C can be used either as simple I/O or as handshake for port B. Figure 15.14 shows two configurations of Mode 2. This mode is illustrated in Section 15.3.

[illegible]

ILLUSTRATION: INTERFACING KEYBOARD AND SEVEN-SEGMENT DISPLAY 15.2

This illustration is concerned with interfacing a pushbutton keyboard and a seven-segment LED display using the 8255A. The emphasis in this illustration is not particularly on the features of 8255A but on how to integrate hardware and software. When a key is pressed, the binary reading of the key has almost no relationship to what it represents. Similarly, to display a number at a seven-segment LED, the binary value of the number needs to be converted into the seven-segment code, which is primarily decided by the hardware consideration. This illustration demonstrates how the microprocessor monitors the changes in hardware reading and converts into appropriate binary reading using its instruction set.

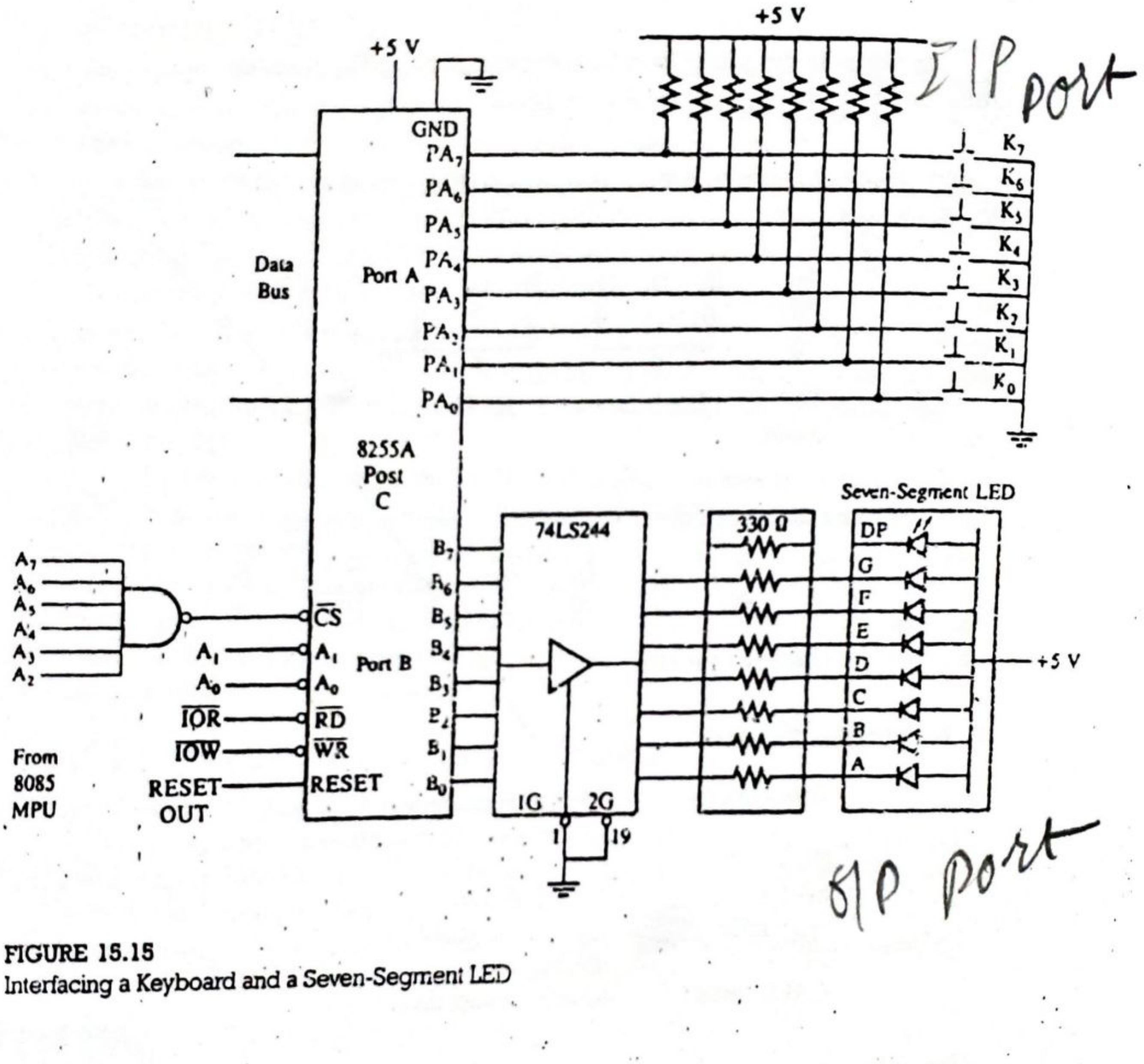

FIGURE 15.15
Interfacing a Keyboard and a Seven-Segment LED

1. Check if a key is pressed.
2. Debounce the key.
3. Identify and encode the key in appropriate binary format.
4. Obtain the seven-segment code and display it.

The instructions for these steps can be written in separate modules, as shown in the next section.

15.2.3 Keyboard

The keys K_7–K_0 are tied high through 10 k resistors, and when a key is pressed, the corresponding line is grounded. When all keys are open and if the 8085 reads port A, the reading on the data bus will be FFH. When any key is pressed, the reading will be less than FFH. For example, if K_7 is pressed, the output of port A will be 0111

1111 (7FH). This reading should be encoded into the binary equivalent of the digit 7 (0111) by using software routines. The subroutines KYCHK and KYCODE accomplish the tasks of checking a key pressed and encoding the key in appropriate binary format.

```
KYCHK:   ;This subroutine first checks whether all keys are open.
         ; Then, it checks for a key closure, debounces the key, and places
         ; the reading in the accumulator. See Figure 15.16 for flowchart.
         IN PORTA          ;Read keyboard
         CPI 0FFH          ;Are all keys open?
         JNZ KYCHK         ;If not, wait in loop
         CALL DBONCE       ;If yes, wait 20 ms
KYPUSH:  IN PORTA          ;Read keyboard
         CPI 0FFH          ;Is key pressed?
         JZ KYPUSH         ;If not, wait in loop
         CALL DBONCE       ;If yes, wait 20 ms
         CMA               ;Set 1 for key closure
         ORA A             ;Set 0 flag for an error
         JZ KYPUSH         ;It is error, check again
         RET
```

PROGRAM DESCRIPTION

This subroutine is based on hardware; when all keys are open the keyboard reading is FFH, and when a key is pressed, the reading is less than FFH. The routine begins with the loop to check whether all keys are open, and it stays in the loop until all keys are open (Figure 15.16). This prevents reading the same key repeatedly if someone were to hold the key for a long time. When the routine finds that a key has been released, it waits for 20 ms for a key debounce.

The loop starting at KYPUSH (Figure 15.16) checks whether a key is pressed. When a key is pressed, the reading is less than FFH; thus, the compare instruction does not set the Z flag and the program goes to the next instruction for a key debounce. The CMA instruction complements the accumulator reading; thus, the reading of the key pressed is set to 1, and other bits are set to 0. The next two instructions check for an error. If it is a momentary contact (false alarm), all bits will be 0s. The ORA instruction sets the Z flag, and the Jump instruction takes the program back to checking keys.

```
KYCODE: ;This routine converts (encodes) the binary hardware reading of the key
        ; pressed into appropriate binary format according to the number of the
        ; key.
        MVI C,08H         ;Set code encounter
NEXT:   DCR C             ;Adjust key code
        RAL               ;Place MSB in CY
        JNC NEXT          ;If bit = 0, go back to check next bit
        MOV A,C           ;Place key code in the accumulator
        RET
```

85

PROGRAM DESCRIPTION

Conceptually, this is an important routine; it establishes the relationship between the hardware and the number of a key. For example, if key K_7 is pressed, the reading from the routine KYCHK in the accumulator will be 1000 0000 (the reading is already complemented). The KYCODE routine sets register C for the count of eight and immediately decrements the count to seven. The instruction RAL places bit D_7 in the CY flag, and the next instruction checks for the CY flag. If it is set, the key K_7 must be pressed, and the key code (digit 7) is

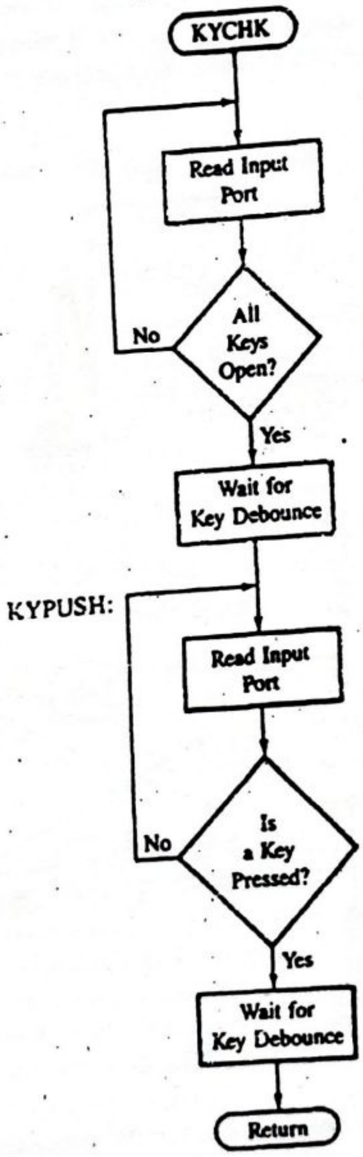

FIGURE 15.16
Flowchart: Key Check Subroutine

Debounce
press

in register C. If CY = 0, the program loops back to check the next bit (D_6). The loop is repeated until 1 is found in CY, and at every iteration of the loop the key code in register C is adjusted for the next key. If more than one key is pressed, this routine ignores the low-order key. Finally, the subroutine places the key code in the accumulator and returns.

KEY DEBOUNCE

When a mechanical pushbutton key, shown in Figure 15.17(a), is pressed or released, the metal contacts of the key momentarily bounce before giving a steady-state reading, as shown in Figure 15.17(b). Therefore, it is necessary that the bouncing of the key should not be read as an input. The key bounce can be eliminated from input data by the **key-debounce technique**, using either hardware or software.

Figure 15.17(c) shows a key debounce circuit. In this circuit, the outputs of the NAND gates do not change even if the key is released from position A_1. The outputs change when the key makes a contact with position B_1. When the key is connected to A_1,

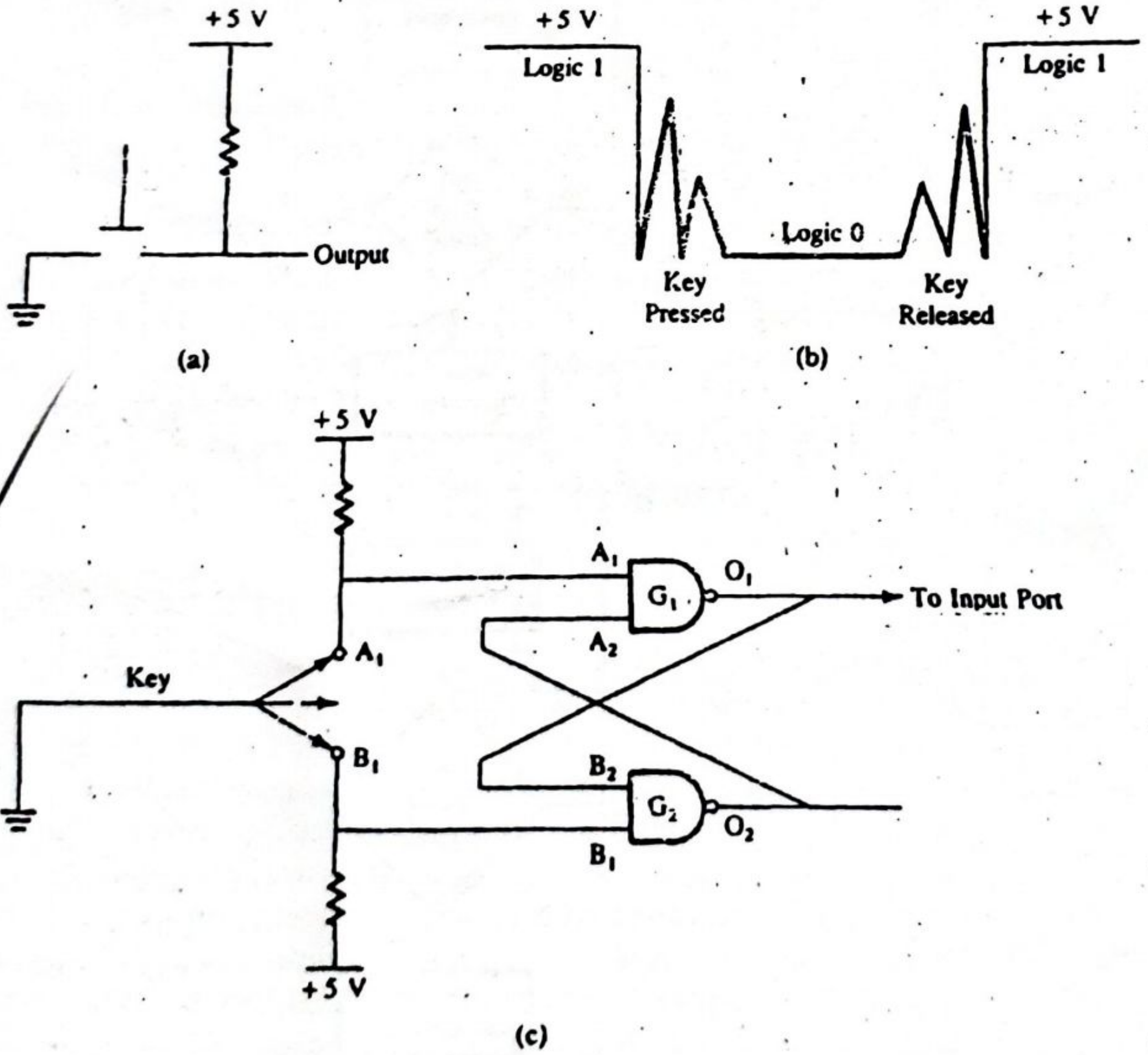

FIGURE 15.17
Pushbutton Key (a), Key Bounce (b), and Key Debounce Circuit Using NAND Gates (c)

A_1 goes low. If one of the inputs to gate G_1 is low, the output O_1 becomes 1, which makes B_2 high. Because line B_1 is already high, the output of O_2 goes low, which makes A_2 low. When the key connection is released from A_1, it goes high, but because A_2 is low the output doesn't change. When the key makes contact with B_1, the outputs change. This means when the key goes from one contact (+5 V) to another contact (ground), the output does not change during the transition period, thus eliminating multiple readings.

In the software technique, when a key closure is found, the microprocessor waits for 10 to 20 ms before it accepts the key as an input. The delay routine is as follows:

```
DBONCE: ;This is a 20 ms delay routine
        ;The delay COUNT should be calculated based on system frequency
        ;This does not destroy any register contents
        ;Input and Output = None
        PUSH B          ;Save register contents
        PUSH PSW
        LXI B,COUNT     ;Load delay count
LOOP:   DCX B           ;Next count
        MOV A,C
        ORA B           ;Set Z flag if (BC) = 0
        JNZ LOOP
        POP PSW         ;Restore register contents
        POP BC
        RET
```

PROGRAM DESCRIPTION

This is a simple delay routine similar to the delay routines discussed in Chapter 8. The first instruction loads the BC register with a 16-bit number, and the loop is repeated until BC = 0. In this routine, the 16-bit number (COUNT) should be calculated on the basis of the clock frequency of the system and the T-states in the loop (see Chapter 8 for details).

15.2.4 Seven-Segment Display

Figure 15.15 shows that a common-anode seven-segment LED is connected to port B through the driver 74LS244. The driver is necessary to increase the current capacity of port B; each LED segment requires 15–20 mA of current. The code for each Hex digit from 0 to F can be determined by examining the connections of the data lines to the segments and the logic requirements.

The driver 74LS244 (Figure 15.15) is an octal noninverting driver with tri-state output and current sinking capacity of 24 mA. It has two active low enable lines ($1\overline{G}$ and $2\overline{G}$), and the driver is permanently enabled by grounding these lines. In this circuit, this driver functions simply as a current amplifier; whatever logic is at port B will be at the output of the driver.

To display the number of the key pressed, a routine is necessary that will send an appropriate code to port B. The routine KYCODE supplies the binary number of the key pressed; however, there is no relationship between the binary value of a digit and its seven-segment code. Therefore, the table look-up technique (refer to Chapter 10, Section 10.3) will have to be used to find the code for the digit supplied by the KYCODE; this is shown in the next routine, DSPLAY.

```
DSPLAY: ;This routine takes the binary number and converts into its common-
        ;  anode seven-segment LED code. The codes are stored in memory
        ;  sequentially, starting from the address CACODE
        ;Input: Binary number in accumulator
        ;Output: None
        ;Modifies contents of HL and A
        LXI H,CACODE        ;Load starting address of code table in HL
        ADD L               ;Add digit to low-order address in L
        MOV L,A             ;Place code address in L
        MOV A,M             ;Get code from memory
        OUT PORTB           ;Send code to port B
        RET
CACODE: ;Common-anode seven-segment codes are stored sequentially in memory
        DB 40H,79H,24H,30H,19H,12H ;Codes for digits from 0 to 5
        DB 02H,78H,00H,18H,08H,03H ;Codes for digits from 6 to B
        DB 46H,21H,06H,0EH         ;Codes from digits from C to F
```

PROGRAM DESCRIPTION

In this routine the HL register is used as a memory pointer to code location. The digit to be displayed is in the accumulator, supplied by the routine KYCODE, and the seven-segment code is stored sequentially in memory, starting from location CACODE. The basic concept in this routine is to modify the memory pointer by adding the value of the digit to the base address and get the code location. For example, let us assume that the starting address of CACODE is 2050H and the digit 7 is in the accumulator. The code for digit 0 is in location 2050H; consequently, the code for digit 7 is in location 2057H. Therefore, to display digit 7, the routine adds the contents of the accumulator (7) to the low-order byte 50H in register L, resulting in the sum 57H. By transferring 57H in register L, the memory pointer in HL is modified to 2057H. Thus, the code for digit 7 is obtained by using this memory pointer.

15.2.5 Main Program

Now to monitor the keyboard and display the key pressed, we need to initialize the 8255A ports and combine the software modules discussed below:

```
KYBORD: ;This program initializes the 8255A ports; port A and port B in Mode 0
        ;  and then calls the subroutine modules discussed
        ;  previously to monitor the keyboard
PORTA   EQU FCH             ;Port A address
PORTB   EQU FDH             ;Port B address
```

```
CNTRL    EQU FFH          ;Control register
CNWORD   EQU 90H          ;Mode 0 control word, port A input and port B output
STACK    EQU 20AFH        ;Beginning stack address
         LXI SP,STACK
PPI:     MVI A,CNWORD
         OUT CNTRL        ;Set up port A in Mode 1
NEXTKY:  CALL KYCHK       ;Check if a key is pressed
         CALL KYCODE      ;Encode the key
         CALL DSPLAY      ;Display key pressed
         JMP NEXTKY       ;Check the next key pressed
```

PROGRAM DESCRIPTION

This is the main program, which involves the initialization of the 8255A and the stack pointer. The port addresses defined here are from Figure 15.13, and the address of STACK (20AFH) is shown as an illustration; it has no specific significance. Because the problem is divided into small modules, the main program consists primarily of calling these modules.

15.2.6 Comments and Alternative Approaches

The interfacing of the pushbutton keyboard and seven-segment display is a simplified illustration of industrial applications. The illustration is deliberately kept simple to emphasize the conceptual framework between hardware and software. However, as an application, it has several limitations, as follows:

1. The method of connecting the keyboard demands the number of I/O ports be in proportion to the number of keys; only eight keys can be connected to an 8-bit port. Generally, keys are connected in a matrix format (discussed in Chapter 17). For example, in the matrix format, 16 keys can be connected to one 8-bit port or 64 keys can be connected to two 8-bit ports.
2. The method of connecting a seven-segment LED needs excessive hardware, one port per seven-segment LED and a driver. Furthermore, it consumes a large amount of current (100 to 150 mA per display). To minimize hardware and power consumption, the technique of multiplexing is generally used (discussed in Chapter 17).

In this illustration, the approach is primarily software. For example, in the keyboard, the debouncing and encoding are performed by using instructions. However, nowadays, interfacing chips are available commercially that can sense a key closure, debounce the key, and encode the key. In addition, the chip can generate an interrupt signal when a key is pressed. Similarly, in the seven-segment display, the table look-up can be replaced by a decoder/driver. However, the hardware approach increases unit price. On the other hand, the software approach involves considerable labor (programming and debugging) cost. The choice is generally determined by the production volume and the total unit price.