CHAPTER 3
SOFTWARE APPLICATIONS

3.1 INTRODUCTION

The term "software" is generally used to define computer programs and, in its broadest sense, it refers to an entire set of programs, procedures, and all related documentation associated with a system. In this manual, software refers to programs or routines. The writing of software is best learned by the experience of writing your own programs; however, a few good examples can certainly speed the learning experience. The examples provided in this chapter illustrate various M6805 HMOS/M146805 CMOS Family software features and include some commonly used routines. Included at the end of this chapter is a small debug monitor program named ASSIST05. The ASSIST05 debug monitor includes many features and routines which are useful for product evaluation and development. The routines described in the following paragraphs are not necessarily the most efficient; however, each may be used as a learning tool.

3.2 SERIAL I/O ROUTINES

Although serial I/O hardware is to be included on future M6805 HMOS/M146805 CMOS Family members, none exists on the current 14 family members. However, serial I/O can be implemented on any device, provided a small amount of software and port I/O overhead can be spared.

Three different serial I/O routine examples are discussed in this chapter. The first example generates the serial data and clock inputs for the MC145000 multiplexed LCD driver. The second example generates serial data in an NRZ format, for use with an RS-232 interface, at speeds up to 9600 baud. The third example generates serial data in an NRZ format for use in a serial loop interface.

3.2.1 MC145000 Serial I/O Software

The MC145000 (Master) LCD driver is designed to drive liquid crystal displays in a multiplexed-by-four configuration. It can drive up to 48 LCD segments or six 7-segment plus decimal point characters. The required hardware connections are shown in Figure 3-1. Data for each character must be translated into a format that produces the desired display. Table 3-1 provides a listing of the display format (hexadecimal code) for each displayed character. After the format translation is completed, data can be clocked serially into the MC145000 LCD driver. Each segment of 7-segment character plus

decimal point is represented by one bit of an 8-bit byte. As shown in Figure 3-2, a logic "1" in any bit will activate the corresponding segment of the character, plus decimal points.

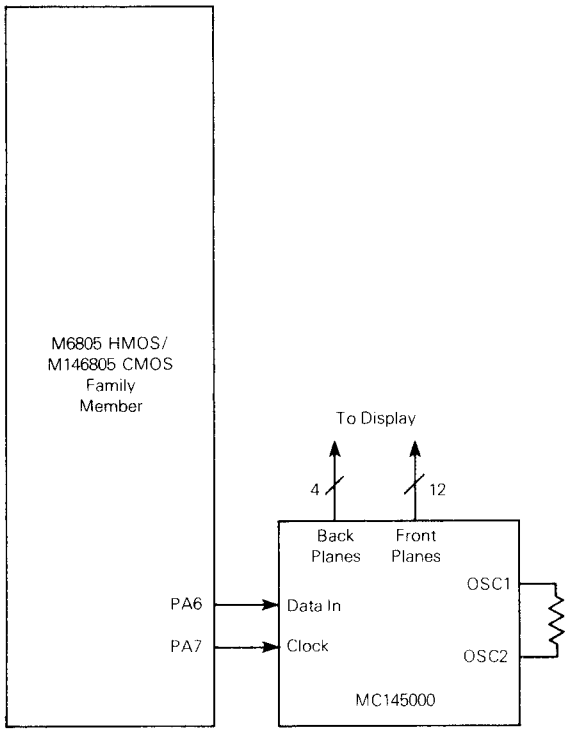


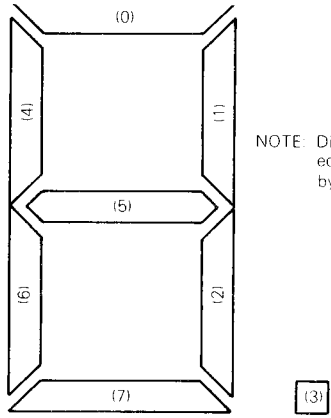
Figure 3-1. MC145000 LCD Driver Interface Schematic Diagram

Table 3-1. Display Format Conversions

Displayed Character	Display Format Hex Code
0	D7
1	06
2	E3
3	A7
4	36
5	B5
6	F5
7	07
8	F7
9	B7
A	77
b	F4
C	D1

Displayed Character	Display Format Hex Code
d	E6
E	F1
F	71
P	73
Y	B6
H	76
U	D6
L	D0
blank	00
- (dash)	20
= (equal)	A0
n	64
r	60
° (degrees)	33

NOTE: A Decimal point can be added to all but the right-most display digit by setting b3 [segment (3)] to a 1.



NOTE: Digits in parentheses represent equivalent bits of the activating byte.

Figure 3-2. 7-Segment Display Format

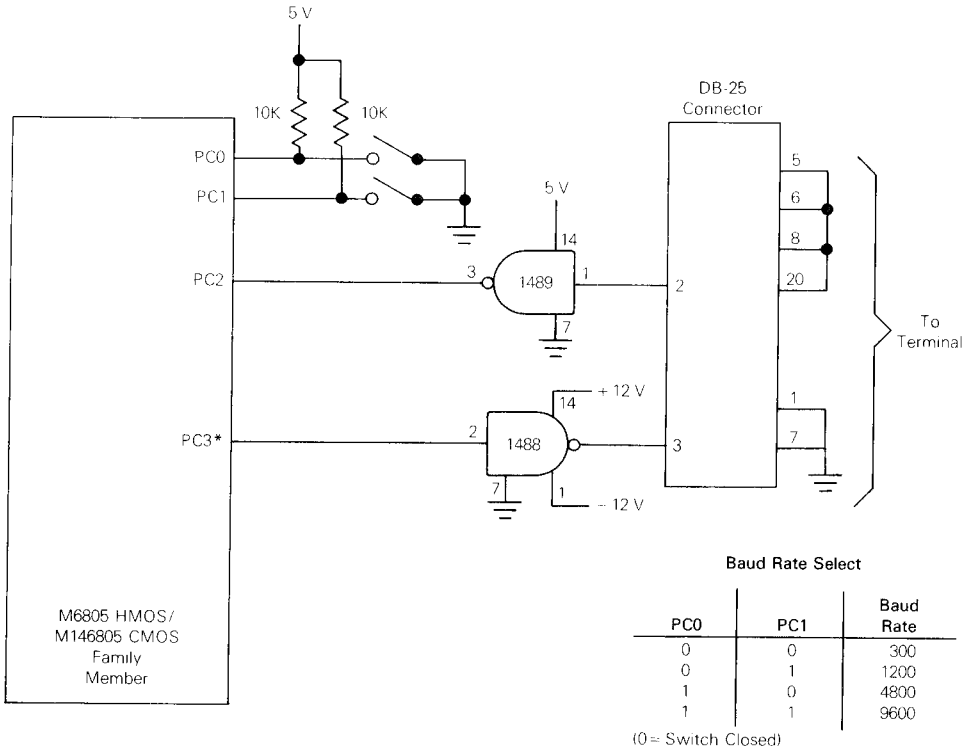
Figure 3-3 contains two software subroutine examples: DISPLY and DISTAB. The DISPLY subroutine clocks data from the accumulator into the MC145000. The DISTAB subroutine loads an eight character table into the MC145000. Note that in the DISPLY subroutine the use of bit manipulation (BSET and BCLR) helps keep the subroutine short and relatively simple. In this case, port A bits 6 and 7 are used for the data and clock lines; however, any port lines could have been stipulated in the program.

```
                  ****************************************
                  *                                      *
                  *      DISPLAY TABLE CONTENTS          *
                  *                                      *
                  *      A,X REGISTERS DESTROYED         *
                  *                                      *
                  ****************************************
                  *
AE 05          A DISTAB LDX     #5
E6 49          A DISCHR LDA     DTABL,X   LOAD DISPLAY
AD 09       1E0C        BSR     DISPLY    TABLE INTO
5A                      DECX              145000
2A F9       1DFF        BPL     DISCHR
81                      RTS
                  ****************************************
                  *                                      *
                  *      SHIFT ONE CHARACTER INTO        *
                  *             DISPLAY                  *
                  *                                      *
                  *      A REGISTER DESTROYED            *
                  *                                      *
                  ****************************************
                  *
BF 50          A DISPLY STX     WORK1     SAVE INDEX
1D 00          A        BCLR    6,PORTA   CLEAR DATA
AE 08          A        LDX     #8
48               DIS1   LSLA              SET UP
24 02       1E17        BCC     DIS2      BIT OF
1C 00          A        BSET    6,PORTA   ACCUMULATOR
1E 00          A DIS2   BSET    7,PORTA   CLOCK
1F 00          A        BCLR    7,PORTA   IT
1D 00          A        BCLR    6,PORTA   CLEAR DATA
5A                      DECX              COMPLETE?
26 F2       1E12        BNE     DIS1      NO
BE 50          A        LDX     WORK1     RESTORE INDEX
81                      RTS
```

Figure 3-3. Serial I/O Display Subroutine Examples

3.2.2 Serial I/O Software for RS-232

The example discussed here uses two I/O port lines as the serial input and output lines. Figure 3-4 contains a schematic diagram of an RS-232 interface for serial I/O. Included as part of Figure 3-4 is the baud rate selection table showing baud rates of 300, 1200, 4800, and 9600. The example subroutine is illustrated in Figure 3-5. In this example, PC2 is used as the input line and PC3 is used as the output line. Software loops are used to generate the desired baud rates; therefore, the crystal frequency (f_{osc}) is critical (3.579545 MHz). The subroutine example shown in Figure 3-5 is taken from the MC146805G2( )1 evaluation monitor. The same subroutine is essentially used in all M6805 HMOS/M146805 CMOS Family evaluation programs; however, in the example, the instructions followed by the comment "CMOS DITTO" or "CMOS EQUALIZATION" cannot be used with HMOS versions of the evaluation program. These extra instructions are necessary in the CMOS version to "make-up" for the generally fewer cycles-per-instruction of the M146805 CMOS Family members.



PC0	PC1	Baud Rate
0	0	300
0	1	1200
1	0	4800
1	1	9600

* For devices which have port C as input-only, use PB7

Figure 3-4. RS-232 Interface for Serial I/O via I/O Port Lines Schematic Diagram

```
                        *
                        *       S E R I A L   I / O   R O U T I N E S
                        *
                        *       THESE SUBROUTINES ARE MODIFICATIONS OF THE ORIGINAL NMOS
                        *       VERSION.  DIFFERENCES ARE DUE TO THE VARIATION IN CYCLE
                        *       TIME OF CMOS INSTRUCTIONS VS. NMOS.
                        *
                        *       SINCE THE INT AND TIMER INTERRUPT VECTORS ARE USED IN THE
                        *       BICYCLE ODOMETER, THE I-BIT SHOULD  ALWAYS  BE  SET  WHEN
                        *       RUNNING  THE  MONITOR.  HENCE, THE CODE THAT FIDDLES WITH
                        *       THE I-BIT HAS BEEN ELIMINATED.
                        *
                        *
                        *       DEFINITION OF SERIAL I/O LINES
                        *
                        *       NOTE: CHANGING °IN' OR °OUT' WILL NECESSITATE CHANGING THE
                        *       WAY °PUT' IS SETUP DURING RESET.
                        *
07C3 00 02              PUT     EQU     PORTC   SERIAL I/O PORT
07C3 00 02              IN      EQU     2       SERIAL INPUT LINE#
07C3 00 03              OUT     EQU     3       SERIAL OUTPUT LINE#
                        *
                        *       GETC --- GET A CHARACTER FROM THE TERMINAL
                        *
                        *       A GETS THE CHARACTER TYPED, X IS UNCHANGED.
                        *
07C3 BF 15              GETC    STX     XTEMP   SAVE X
07C5 A6 08                      LDA     #8      NUMBER OF BITS TO READ
07C7 B7 17                      STA     COUNT
07C9 04 02 FD           GETC4   BRSET   IN,PUT,GETC4 WAIT FOR HILO TRANSITION
                        *
                        *       DELAY 1/2 BIT TIME
                        *
07CC B6 02                      LDA     PUT
07CE A4 03                      AND     #%11    GET CURRENT BAUD RATE
07D0 97                         TAX
07D1 DE 08 4B                   LDX     DELAYS,X GET LOOP CONSTANT
07D4 A6 04              GETC3   LDA     #4
07D6 9D                 GETC2   NOP
07D7 4A                         DECA
07D8 26 FC                      BNE     GETC2
07DA 5D                         TSTX            LOOP PADDING
07DB 14 02                      BSET    IN,PUT  DITTO
07DD 14 02                      BSET    IN,PUT  CMOS DITTO
07DF 5A                         DECX
07E0 26 F2                      BNE     GETC3   MAJOR LOOP TEST
                        *
                        *       NOW WE SHOULD BE IN THE MIDDLE OF THE START BIT
                        *
07E2 04 02 E4                   BRSET   IN,PUT,GETC4 FALSE START BIT TEST
07E5 7D                         TST     ,X      MORE TIMING DELAYS
07E6 7D                         TST     ,X
07E7 7D                         TST     ,X
                        *
                        *       MAIN LOOP FOR GETC
                        *
07E8 AD 46              GETC7   BSR     DELAY   (6) COMMON DELAY ROUTINE
07EA 05 02 00                   BRCLR   IN,PUT,GETC6 (5)   TEST INPUT AND SET C-BIT
07ED 7D                 GETC6   TST     ,X      (4) TIMING EQUALIZER
07EE 9D                         NOP             (2) CMOS EQUALIZATION
07EF 9D                         NOP             (2) CMOS EQUALIZATION
07F0 9D                         NOP             (2) CMOS EQUALIZATION
07F1 9D                         NOP             (2) CMOS EQUALIZATION
07F2 9D                         NOP             (2) CMOS EQUALIZATION
07F3 9D                         NOP             (2) CMOS EQUALIZATION
07F4 36 16                      ROR     CHAR    (5) ADD THIS BIT TO THE BYTE
07F6 3A 17                      DEC     COUNT   (5)
07F8 26 EE                      BNE     GETC7   (3) STILL MORE BITS TO GET(SEE?)
                        *
07FA AD 34                      BSR     DELAY   WAIT OUT THE 9TH BIT
07FC B6 16                      LDA     CHAR    GET ASSEMBLED BYTE
07FE BE 15                      LDX     XTEMP   RESTORE X
                        *
```

Figure 3-5. Serial I/O Software Subroutine Example

```
0800 81                 RTS                 AND RETURN
                *
                *       PUTC --- PRINT A ON THE TERMINAL
                *
                *       X AND A UNCHANGED
                *
0801 B7 16      PUTC    STA     CHAR
0803 B7 14              STA     ATEMP   SAVE IT IN BOTH PLACES
0805 BF 15              STX     XTEMP   DON'T FORGET ABOUT X
0807 A6 09              LDA     #9      GOING TO PUT OUT
0809 B7 17              STA     COUNT   9 BITS THIS TIME
080B 5F                 CLRX            FOR VERY OBSCURE REASONS
080C 98                 CLC             THIS IS THE START BIT
080D 20 02              BRA     PUTC2   JUMP IN THE MIDDLE OF THINGS
                *
                *       MAIN LOOP FOR PUTC
                *
080F 36 16      PUTC5   ROR     CHAR    (5) GET NEXT BIT FROM MEMORY
0811 24 04      PUTC2   BCC     PUTC3   (3) NOW SET OR CLEAR PORT BIT
0813 16 02              BSET    OUT,PUT
0815 20 04              BRA     PUTC4
0817 17 02      PUTC3   BCLR    OUT,PUT (5)
0819 20 00              BRA     PUTC4   (3) EQUALIZE TIMING AGAIN
081B DD 08 30   PUTC4   JSR     DELAY,X (7) MUST BE 2-BYTE INDEXED JSR
                *                           THIS IS WHY X MUST BE ZERO
081E 43                 COMA            (3) CMOS EQUALIZATION
081F 43                 COMA            (3) CMOS EQUALIZATION
0820 43                 COMA            (3) CMOS EQUALIZATION
0821 3A 17              DEC     COUNT   (5)
0823 26 EA              BNE     PUTC5   (3) STILL MORE BITS
                *
0825 14 02              BSET    IN,PUT   7 CYCLE DELAY
0827 16 02              BSET    OUT,PUT    SEND STOP BIT
                *
0829 AD 05              BSR     DELAY   DELAY FOR THE STOP BIT
082B BE 15              LDX     XTEMP   RESTORE X AND
082D B6 14              LDA     ATEMP   OF COURSE A
082F 81                 RTS
                *
                *       DELAY --- PRECISE DELAY FOR GETC/PUTC
                *
0830 B6 02      DELAY   LDA     PUT     FIRST, FIND OUT
0832 A4 03              AND     #%11    WHAT THE BAUD RATE IS
0834 97                 TAX
0835 DE 08 4B           LDX     DELAYS,X LOOP CONSTANT FROM TABLE
0838 A6 F8              LDA     #$F8    FUNNY ADJUSTMENT FOR SUBROUTINE OVERHEAD
083A AB 09      DEL3    ADD     #$09
083C            DEL2
083C 9D                 NOP             CMOS EQUALIZATION
083D 4A                 DECA
083E 26 FC              BNE     DEL2
0840 5D                 TSTX            LOOP PADDING
0841 14 02              BSET    IN,PUT  DITTO
0843 14 02              BSET    IN,PUT  CMOS DITTO
0845 5A                 DECX
0846 26 F2              BNE     DEL3    MAIN LOOP
0848 9D                 NOP             CMOS EQUALIZATION
0849 9D                 NOP             CMOS EQUALIZATION
084A 81                 RTS             WITH X STILL EQUAL TO ZERO
                *
                *       DELAYS FOR BAUD RATE CALCULATION
                *
                *       THIS TABLE MUST NOT BE PUT ON PAGE ZERO SINCE
                *       THE ACCESSING MUST TAKE 6 CYCLES.
                *
084B 20         DELAYS  FCB     32       300 BAUD
084C 08                 FCB     8       1200 BAUD
084D 02                 FCB     2       4800 BAUD
084E 01                 FCB     1       9600 BAUD
```

Figure 3-5. Serial I/O Software Subroutine Example (Continued)

3.2.3 Software Serial Loop

The M6805 HMOS/M146805 CMOS Family may be used in either distributed or network processing. Those family devices may also be used as intelligent peripherals, and offer a variety of special features including: A/D, PLL, timer, and I/O lines. In addition, the devices could be interfaced with one of several serial interfaces; e.g., point-to-point, multidrop, or loop.

The MC6805R2( )1 is an evaluation device which contains the serial routines required to setup a serial loop. In this configuration, one device in the loop serves as the loop master and transmits user commands to the loop. Each slave device in the loop examines the transmitted message and determines if it must execute the command. The user commands enter the loop via the terminal connected to the loop master as shown in Figure 3-6. The programmed configuration of port C lines PC0-PC3 determine which device is the master, which devices are slaves, and the loop baud rate.

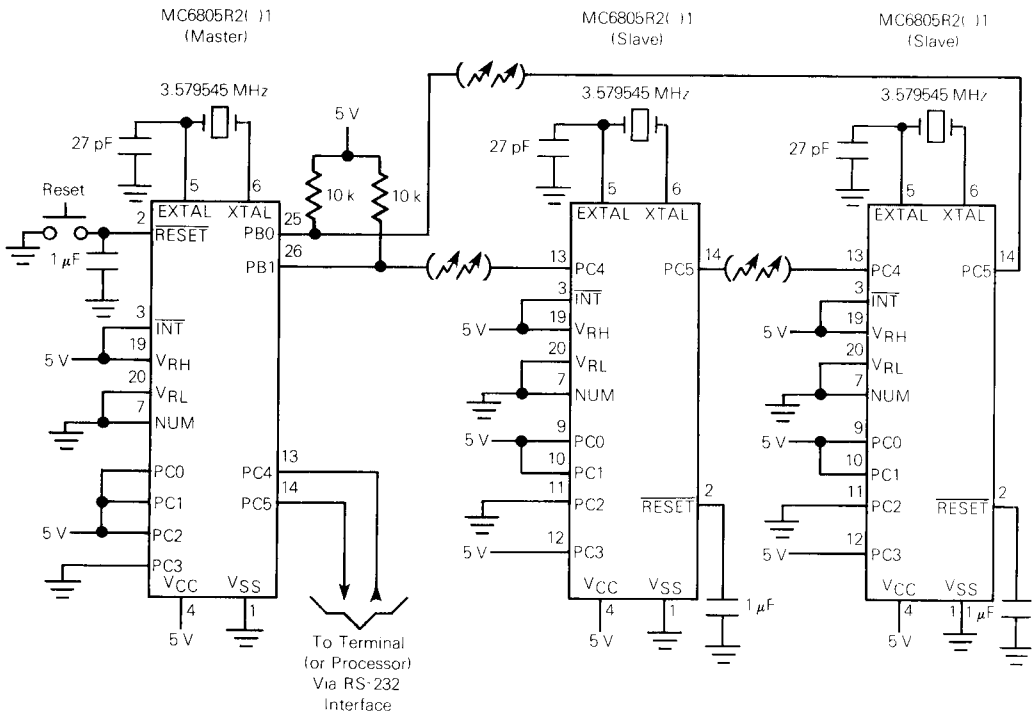


Figure 3-6. Software Serial Loop Hardware Connections Schematic Diagram

Each transmitted message contains either two or three bytes. They include:

1. destination address (node)
2. address of byte being accessed within destination plus a read/write bit
3. data, provided read/write bit in byte two indicates a write

Each device in the loop (master and slaves) examines the node address. If the node address is zero, the slave device processes the message. If the node address is non-zero, the slave device decrements the node address and passes the message to the next node in the loop.

When a slave device processes a message, data is read from or written to the address specified in the first seven bits of byte two. Only seven bits of address are necessary since the MC6805R2( )1 RAM and registers are located in the first 128 bytes. If the read/write bit in byte two indicates a write is requested, the data contained in byte three is written to the specified address; however, if the bit indicates a read is requested, the slave performs the read (from its own on-chip memory) and forms a new message that includes the maximum node address and the data just read. The maximum node address guarantees that the data is received by the loop master since it includes all devices in the loop. The loop master can then display the data in response to the user command input.

Some improvements could be made to the loop system discussed above. Improvements could include replacing the operator controlled terminal with a microprocessor (MPU), microcomputer (MCU), or an intelligent peripheral controller (IPC). The new device can submit commands to the loop in the same format as the terminal; however, when it is not providing input to the loop, it can be processing other functions not necessarily related to the loop.

The features of the slave devices could be used by the controlling MPU, MCU, or IPC. The M6805 HMOS/M146805 CMOS Family devices can be used as intelligent peripherals to provide improvements in system throughput. Serial links allow long-distance communications with minimum line costs. The example discussed above provides a simple but powerful system that can be used as a basis for a more sophisticated system.

3.3 BLOCK MOVE

One of the more commonly used routines is one in which a block of data, located in memory, is copied or moved to another memory location. The indexed addressing modes of the M6805 HMOS/M146805 CMOS Family makes the block move relatively simple.

An example of this routine is shown in Figure 3-7. In this example, the location of the first table entry is used as the offset for the indexed instruction. The index register is used to step through the table; therefore, the table may be up to 256 bytes long. This example uses a table length of 64 bytes ($40). Note that in the example of Figure 3-7, the source table and the destination are located in page zero. The difference between the two indexed instructions is the number of bytes and cycles required for execution.

```
                SOURCE   EQU   $F0
                DESTIN   EQU   $40
AE   20                  LDX   #$20        Load Index Register W/Table Length
E6   F0         REPEAT   LDA   SOURCE,X    Get Table Entry
E7   40                  STA   DESTIN,X    Store Entry Table
5A                       DECX              Next Entry
26   F8                  BNE   REPEAT      REPEAT If More
```

Figure 3-7. Block Move Routine Example

3.4 STACK EMULATION

By proper use of the stack, the versatility of a program can be increased. This can be done by allowing registers or values to be stored temporarily in RAM and then later retrieved. Variables which are stored in the stack are always positioned relative to the top of the stack. Stacks operate in a last-in-first-out (LIFO) fashion; that is, the last byte in is the first byte that can be retrieved. Because of this LIFO characteristic, the stack is useful for passing subroutine variables as well as other valuable programming tools.

The M6805 HMOS/M146805 CMOS Family stack is reserved for subroutine return addresses and for saving register contents during interrupts. This is sufficient for most control-oriented applications; however, the routine shown in Figure 3-8 can provide the MC146805E2 MPU with additional stack capability for temporary variable storage. In this routine, a temporary location called POINTER serves to hold the relative address of the next free stack location. When the routine is entered, the contents of POINTER are transferred to the index register. The two-byte indexed addressing mode is used to allow the stack to be located in any part of RAM. Since the index register is used to provide a relative address, the stack wraps around if more than 256 locations are pushed onto the stack. The stacking routine shown in Figure 3-8 uses two fixed temporary locations: one (called POINTER) is used to save the stack pointer and the other (called TEMPX) is used as a temporary storage for the index register. However, if the index register can be dedicated to the stack, both temporary locations can be deleted. In this example, two subroutines, PUSH and PULL are used to manipulate data. Subroutine PUSH is used by first loading the accumulator with the data to be saved and then performing a subroutine call to PUSH. Subroutine PULL is used by calling the subroutine PULL after which the data retrieved is contained in the accumulator.

NOTE

If a single-chip MCU is used instead of the MC146805E2, the stack must be located in RAM and a routine must check that the boundaries are not exceeded.

```
        ORG     $10
TEMP    RMB     1
POINTR  RMB     1
STACK   EQU     $3FF
        ORG     $1000
PUSH    STX     TEMPX       Save Index Reg Contents
        LDX     POINTR      Get Pointer
        STA     STACK,X     Save Byte at Stack and Pointer
        DEC     POINTR      Adjust Pointer
        LDX     TEMPX       Retrieve Index Reg Contents
        RTS

PULL    STX     TEMPX
        INC     POINTR
        LDX     POINTR
        LDA     STACK,X
        LDX     TEMPX
        RTS
```

Figure 3-8. Stack Emulation Routine

3.5 KEYPAD SCAN ROUTINE

A common task for control-oriented microprocessors is to scan a 4 × 4 keypad, such as the one illustrated in the example of Figure 3-9. The example shown uses port A lines 4-7 as scanning outputs and port A lines 0-3 as sensing inputs. The routine example shown in Figure 3-10 is intended for use with CMOS microprocessors; however, it could be modified as discussed below for HMOS microprocessors. It is often desirable to place M146805 CMOS Family members in a low-power mode; therefore, the STOP instruction is incorporated in the routine shown in Figure 3-10.

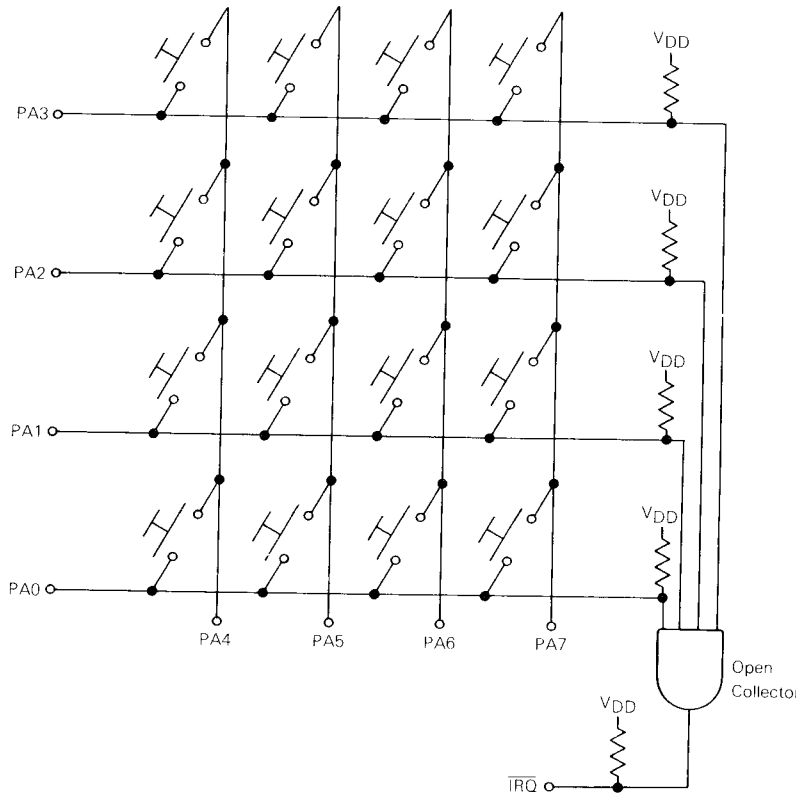


Figure 3-9. 4 x 4 Keypad and Closure Detection Circuit Schematic Diagram

The example shown in Figure 3-10 uses an interrupt driven routine and supports either the STOP or a normal wait for interrupt (see below). If one of the keypad switches is depressed while in the STOP mode, the $\overline{IRQ}$ line goes low. (This is the result of the port A scanning lines being low in the STOP mode.) When IRQ goes low the KEYSCN vector is selected and calls the KEYSCN interrupt service routine. The interrupt service routine first causes IRQ to go high and then scans each column (PA4-PA7) individually to determine which keypad switch was depressed. Once the closed keypad switch is detected, the information is stored and a debounce subroutine is called to verify the closure. The

debounce consists of checking for a keypad switch closure after a 1536 bus cycle (2040 for HMOS) delay to assure that the interrupt was not a result of noise. If a keypad switch closure still exists after the debounce is completed, the routine waits for the switch to be released before forcing all scanning lines low for detection of the next closure. (A Schmitt trigger input on the $\overline{IRQ}$ line further reduces the effects of noise.) Once the key closure is verified, a decode routine is used to determine which keypad was switch closed. If after the debounce subroutine is completed, no keypad switch is detected as being closed, the closure is considered invalid and the processor again enters the STOP mode.

```
PAGE  001  KEYSCN  .SA:0

00001                                OPT     CMOS
00002                          *
00003            0000        A PORTA  EQU     0
00004            0004        A DDRA   EQU     4
00005            0180        A DECODE EQU     $180
00006                          *
00007A 0100                          ORG     $100
00008                          *
00009A 0100 3F 00            A RESET  CLR     PORTA     PREPARE SCANNING LINES
00010A 0102 A6 F0            A        LDA     #$F0      PA4-PA7 AS OUTPUTS
00011A 0104 B7 04            A        STA     DDRA      WHICH OUTPUT LOWS
00012A 0106 8E                 STOP   STOP              ENTER LOW PWR MODE - WAIT FOR INT
00013A 0107 20 FD       0106         BRA     STOP
00014                          *
00015A 0109 A6 EF            A KEYSCN LDA     #$EF      CHECK 1ST COLUMN WITH A LOW
00016A 010B B7 00            A        STA     PORTA     AND OTHERS HIGH
00017A 010D 2E 05       0114 REPEAT BIL      GOTIT     IF IRQ LINE LOW, COLUMN FOUND
00018A 010F 38 00            A        LSL     PORTA     ELSE TRY NEXT COLUMN
00019A 0111 25 FA       010D         BCS     REPEAT    REPEAT IF MORE COLUMNS, ELSE
00020A 0113 80                 RETURN RTI               WAIT FOR VALID CLOSURE
00021                          *
00022A 0114 B6 00            A GOTIT  LDA     PORTA     SAVE KEY IN ACCA
00023A 0116 AD 0C       0124         BSR     DBOUNC    WAIT 1.5K BUS CYCLES (2K FOR HMOS)
00024A 0118 2F F9       0113         BIH     RETURN    IF IRQ LINE HIGH, INVALID CLOSURE
00025A 011A 2E FE       011A RELEAS BIL      RELEAS    WAIT FOR KEY RELEASE
00026A 011C AD 06       0124         BSR     DBOUNC    PAUSE
00027A 011E 2E FA       011A         BIL     RELEAS    IF IRQ LINE LOW, KEY NOT RELEASED
00028A 0120 3F 00            A        CLR     PORTA     PREPARE SCAN LINES FOR STOP MODE
00029A 0122 20 5C       0180         BRA     DECODE    GO TO USER KEY DECODE ROUTINE
00030                          *
00031A 0124 AE FF            A DBOUNC LDX     #$FF
00032A 0126 5A                 AGAIN DECX               LOOPS 1536 TIMES FOR CMOS
00033A 0127 26 FD       0126         BNE     AGAIN     OR 2040 FOR HMOS
00034A 0129 81                       RTS
00035                          *
00036A 012A 80                 TWIRQ RTI
00037A 012B 80                 TIRQ  RTI
00038A 012C 80                 SWI   RTI
00039                          *
00040A 07F6                          ORG     $7F6
00041                          *
00042A 07F6      012A        A       FDB     TWIRQ     TIMER WAIT VECTOR
00043A 07F8      012B        A       FDB     TIRQ      TIMER INTERNAL VECTOR
00044A 07FA      0109        A       FDB     KEYSCN    EXTERNAL INTERRUPT VECTOR
00045A 07FC      012C        A       FDB     SWI       SOFTWARE INTERRUPT VECTOR
00046A 07FE      0100        A       FDB     RESET     RESET VECTOR
00047                          *
00048                                END
TOTAL ERRORS 00000--00000
```

Figure 3-10. KEYSCN Routine Example

A value which represents the position of the closed keypad switch is passed, via the accumulator, to a routine which decodes the position either into a number or a pointer for other routines. All routines which require that the keypad be scanned, can enter the routine either by using the STOP mode (as discussed above for CMOS) or by enabling the external interrupt with a CLI instruction. The CLI instruction then requires a BRA instruction to wait for a keypad switch closure to generate an interrupt.

3.6 DAA (DECIMAL ADJUST ACCUMULATOR)

Although the M6805 HMOS/M146805 CMOS Family is primarily a controller, it is occasionally required to perform arithmetic operations on BCD numbers. Since the ADD instruction operates on binary data, the result of the ADD instruction must be adjusted in these cases. A DAA subroutine example is shown in Figure 3-11. The DAA subroutine should be called immediately after the binary ADD instruction.

```
PAGE  001  DAA      .SA:1

00001                          *
00002                          *  DAA --- DECIMAL ADJUST ACCUMULATOR
00003                          *
00004                          *  THIS INSTRUCTION SIMULATES THE DAA INSTRUCTION
00005                          *  AVAILABLE ON 6800, 6801, AND 6809 PROCESSORS.
00006                          *
00007                          *  THE SUBROUTINE SHOULD BE CALLED IMMEDIATELY AFTER
00008                          *  AN 'ADC' OR 'ADD' INSTRUCTION WHEN PERFORMING BCD
00009                          *  ARITHMETIC.
00010                          *
00011                          *  EXAMPLE:
00012                          *            LDA ARG1
00013                          *            ADD ARG2
00014                          *            BSR DAA
00015                          *
00016                          *  AT ENTRY:
00017                          *             A -- RESULT OF PREVIOUS ADD OR ADC
00018                          *            CC -- RESULT OF PREVIOUS ADD OR ADC
00019                          *
00020                          *  AT EXIT:
00021                          *             A -- CORRECTED BCD NUMBERS
00022                          *            CC -- CARRY BIT SET OR CLEARED FOR
00023                          *                  MULTI-PRECISION ARITHMETIC.
00024                          *
00025                          *  NO WORK AREA IS NEEDED; AND, THE INDEX REGISTER IS
00026                          *  UNAFFECTED.  TWO OR FOUR STACK LOCATIONS MAY BE USED
00027                          *  FOR SUBROUTINE RETURN ADDRESSES.
00028                          *
00029A 0080                          ORG    $80
00030                          *
00031A 0080 25 04      0086 DAA      BCS    DAAHAI    IF CARRY THEN ADJUST HIGH DIGIT
00032A 0082 A1 99         A          CMP    #$99      DOUBLE OVERFLOW? (>99?)
00033A 0084 23 08      008E          BLS    DAALOW    NO, CHECK LOW DIGIT
00034A 0086 40               DAAHAI NEGA              AVOID CLOBBERING H-BIT BY
00035A 0087 A0 60         A          SUB    #$60      A + $60 = - (- A - $60)
00036A 0089 40                       NEGA
00037                          *  THE ABOVE ADJUST MEANS WE MUST RETURN WITH CARRY SET
00038A 008A AD 02      008E          BSR    DAALOW    CHECK LOW DIGIT
00039A 008C 99                       SEC              SET CARRY BIT
00040A 008D 81                       RTS              RETURN WITH CARRY SET
00041                          *  CHECK LOW DIGIT FOR OVERFLOW
00042A 008E 28 03      0093 DAALOW BHCC   DAANOO     NO OVERFLOW DETECTED
00043A 0090 AB 06         A          ADD    #6        ADJUST FOR KNOWN OVERFLOW
00044A 0092 81                       RTS              RETURN WITH CARRY CLEAR
00045A 0093 AB 06         A DAANOO ADD    #6         LOW DIGIT A-F?
00046A 0095 29 02      0099          BHCS   DAARTS    BRANCH ADJUSTED IF
00047A 0097 A0 06         A          SUB    #6        CORRECT ASSUMPTION
00048A 0099 81               DAARTS RTS               RETURN WITH CARRY CLEAR
00049                                END
```

Figure 3-11. DAA Subroutine Example

3.7 MULTIPLY

Multiply subroutines for either 16-bit × 16-bit or 8-bit × 8-bit multiplications can be written using less than 30 bytes. Examples of both cases are illustrated in Figures 3-12 and 3-13. The 16-bit × 16-bit routine is from an example in the User's Group Library. The 8-bit × 8-bit routine of Figure 3-13 is also included in the MC146805G2( )1 evaluation program.

```
PAGE  001  DPMUL05 .SA:0

00001                          *
00002                          *  REF HISPDMUP PGM IN USERS GROUP LIBRARY
00003                          *  LOAD MULTIPLIER INTO (QH,QL)
00004                          *  LOAD MULTIPLICAND INTO (PH,PL)
00005                          *  (PH,PL) * (QH,QL) ---> (TEMPA,TEMPB,QH,QL)
00006                          *
00007                          *  RESULTS ARE:
00008                          *  TEMPA    MOST SIGNIFICANT BYTE
00009                          *  TEMPB    SECOND SIGNIFICANT BYTE
00010                          *  QH       THIRD SIGNIFICANT BYTE
00011                          *  QL       LEAST SIGNIFICANT BYTE
00012                          *
00013A 0064                          ORG     $64
00014                          *
00015A 0064     0001       A PH      RMB     1
00016A 0065     0001       A PL      RMB     1
00017A 0066     0001       A TEMPA   RMB     1
00018A 0067     0001       A TEMPB   RMB     1
00019A 0068     0001       A QH      RMB     1
00020A 0069     0001       A QL      RMB     1
00021                          *
00022A 0080                          ORG     $80
00023A 0080 AE 10          A STRT    LDX     #16
00024A 0082 3F 66          A         CLR     TEMPA
00025A 0084 3F 67          A         CLR     TEMPB
00026A 0086 36 68          A         ROR     QH
00027A 0088 36 69          A         ROR     QL
00028A 008A 24 0C       0098 NXT     BCC     ROTAT
00029A 008C B6 67          A         LDA     TEMPB
00030A 008E BB 65          A         ADD     PL
00031A 0090 B7 67          A         STA     TEMPB
00032A 0092 B6 66          A         LDA     TEMPA
00033A 0094 B9 64          A         ADC     PH
00034A 0096 B7 66          A         STA     TEMPA
00035A 0098 36 66          A ROTAT   ROR     TEMPA
00036A 009A 36 67          A         ROR     TEMPB
00037A 009C 36 68          A         ROR     QH
00038A 009E 36 69          A         ROR     QL
00039A 00A0 5A                       DECX
00040A 00A1 26 E7       008A         BNE     NXT
00041A 00A3 81                       RTS
00042                          *
00043                                END
```

Figure 3-12. 16-bit × 16-bit Multiplication Subroutine Example

```
                        ***********************************************************************
                        *
                        *  MULTIPLY
                        *
                        *       8 BIT BY 8 BIT UNSIGNED MULTIPLY
                        *       OPERANDS IN A AND X ON ENTRY
                        *       16 BIT RESULT IN X:A ON EXIT; X HAS MSB.
                        *
                        *       AVERAGE EXECUTION = 323 CYCLES
                        *       WORST CASE = 425 CYCLES
                        *
                        *
                        *************************
                        *
                        * MULTIPLY / DIVIDE VARIABLES
                        *
                        *************************
                        *
                        * DIVISOR, MTOTAL
                        *
                        *       DIVISOR FOR 16 BIT / 16 BIT DIVIDE ROUTINE
                        *       ALSO USED AS A TEMP IN MULTIPLY
                        *
        005D          A MTOTAL EQU    *
005D    0002          A DIVSR  RMB    2
                        *
                        * DIVIDEND
                        *
                        *       DIVIDEND FOR 16 BIT / 16 BIT DIVIDE ROUTINE
                        *
005F    0002          A DIVDND RMB    2
                        *
                        * TEMPORARY BYTE
                        *
0061    0001          A TEMP   RMB    1
                        *
                        * SAVEX, MCOUNT
                        *
                        *       TEMPORARY STORAGE FOR X REGISTER IN DIVIDE
                        *       ALSO USED FOR COUNTER IN MULTIPLY
                        *
        0062          A MCOUNT EQU    *
0062    0001          A SAVEX  RMB    1
                        *
        012C          A MULST  EQU    *
012C 3F 5D            A        CLR    MTOTAL   INITIALIZE RESULT TEMP
012E 3F 5E            A        CLR    MTOTAL+1
0130 B7 61            A        STA    TEMP     SAVE ONE ARGUMENT
0132 A6 09            A        LDA    #9
0134 B7 62            A        STA    MCOUNT   BYTE LENGTH = 8
                        *
                        * THE ALGORITHM IS A PLAIN SHIFT AND ADD
                        *
        0136          A BIGLOP EQU    *
0136 B6 61            A        LDA    TEMP     GET BACK ARGUMENT
        0138          A SMLOOP EQU    *
0138 3A 62            A        DEC    MCOUNT   WHILE COUNT IS NOT ZERO
013A 27 13        014F         BEQ    DONE
013C 38 5E            A        LSL    MTOTAL+1 SHIFT TOTAL LEFT BY 1
013E 39 5D            A        ROL    MTOTAL
0140 58                        LSLX            GET NEXT BIT FROM X
0141 24 F5        0138         BCC    SMLOOP   NO ADD IF C=0
                        *
                        * C=1; ADD A TO TOTAL
                        *
0143 B7 61            A        STA    TEMP
0145 BB 5E            A        ADD    MTOTAL+1
0147 24 02        014B         BCC    NOCARY
0149 3C 5D            A        INC    MTOTAL
014B B7 5E            A NOCARY STA    MTOTAL+1
014D 20 E7        0136         BRA    BIGLOP
                        *
                        * HERE TO EXIT
                        *
        014F          A DONE   EQU    *
014F BE 5D            A        LDX    MTOTAL
0151 B6 5E            A        LDA    MTOTAL+1 RETURN RESULT IN A:X
0153 81                        RTS
```

Figure 3-13. 8-bit × 8-bit Multiplication Subroutine Example

3.8 DIVIDE

Two examples of subroutines which can be used for performing division of two numbers are illustrated in Figures 3-14 and 3-15. One subroutine performs a 16-bit ÷ 16-bit with an 8-bit result and the other performs a 16-bit ÷ 16-bit with a 16-bit result. The subroutine of Figure 3-14 is included as part of the MC146805G2( )1 evaluation program. The subroutine of Figure 3-15 is from the User's Group Library. Notice that neither subroutine requires more than 50 bytes.

```
                        ********************************************************************
                        *
                        *       D I V I D E   R O U T I N E
                        *
                        ********************************************************************
                        *
                        *
                        *       16 BIT / 16 BIT --> 8 BIT RESULT        DIVIDE
                        *
                        *       ON ENTRY:
                        *          DIVSR CONTAINS THE DIVISOR
                        *          DIVDND CONTAINS THE DIVIDEND
                        *
                        *       ON EXIT:
                        *          A CONTAINS THE ROUNDED QUOTIENT
                        *          DIVSR AND DIVDND ARE DESTROYED
                        *          TEMP IS DESTROYED
                        *
                        *          IF DIVISION BY ZERO, 255 IS RETURNED.
                        *
                        *       ADAPTED FROM A 68Ø1 DIVIDE ALGORITHM FOUND IN THE 68Ø1
                        *       USER'S MANUAL WRITTEN BY BILL BRUCE.
                        *
                        *       AVERAGE EXECUTION SPEED = 644 CYCLES
                        *       WORST CASE SPEED = 1376 CYCLES
                        *
                        *
                        ********************************************************************

                        *************************
                        *
                        * MULTIPLY / DIVIDE VARIABLES
                        *
                        *************************
                        *
                        * DIVISOR, MTOTAL
                        *
                        *       DIVISOR FOR 16 BIT / 16 BIT DIVIDE ROUTINE
                        *       ALSO USED AS A TEMP IN MULTIPLY
                        *
        ØØ5D        A MTOTAL EQU    *
ØØ5D    ØØØ2        A DIVSR  RMB    2
                        *
                        * DIVIDEND
                        *
                        *       DIVIDEND FOR 16 BIT / 16 BIT DIVIDE ROUTINE
                        *
ØØ5F    ØØØ2        A DIVDND RMB    2
                        *
                        * TEMPORARY BYTE
                        *
ØØ61    ØØØ1        A TEMP   RMB    1
                        *
                        * SAVEX, MCOUNT
                        *
                        *       TEMPORARY STORAGE FOR X REGISTER IN DIVIDE
                        *       ALSO USED FOR COUNTER IN MULTIPLY
                        *
        ØØ62        A MCOUNT EQU    *
ØØ62    ØØØ1        A SAVEX  RMB    1
                        *
```

Figure 3-14. 16-bit ÷ 16-bit With 8-Bit Result Subroutine Example

```
        00E5      A DIV     EQU    *
00E5 A6 02        A         LDA    #2        SET SHIFT COUNT TO GENERATE 9 BITS
00E7 B7 61        A         STA    TEMP      8 FOR RESULT PLUS 1 TO ROUND
00E9 B6 5D        A         LDA    DIVSR
00EB BE 5E        A         LDX    DIVSR+1
00ED 26 07     00F6         BNE    NOZERO    CHECK FOR DIVISION BY ZERO
00EF 4D                     TSTA
00F0 26 05     00F7         BNE    DIV01     BRANCH IF NOT ZERO
00F2 A6 FF        A         LDA    #255      DIVISION BY ZERO
00F4 20 35     012B         BRA    DIVOUT    GO EXIT
                    *
        00F6      A NOZERO  EQU    *
00F6 4D                     TSTA             SHIFT DIVISOR LEFT UNTIL SIGN BIT =1
00F7 2B 06     00FF DIV01   BMI    OUTD
00F9 3C 61        A LOOP2   INC    TEMP
00FB 58                     LSLX
00FC 49                     ROLA             INCR SHIFT COUNT
00FD 2A FA     00F9         BPL    LOOP2
00FF B7 5D        A OUTD    STA    DIVSR     RESTORE DIVISOR
0101 BF 5E        A         STX    DIVSR+1
0103 5F                     CLRX             CLEAR PLACE FOR QUOTENT
                    * MAIN  LOOP
0104 B6 60        A LOOP    LDA    DIVDND+1  DIVIDEND-DIVISOR --> DIVIDEND
0106 B0 5E        A         SUB    DIVSR+1
0108 B7 60        A         STA    DIVDND+1
010A B6 5F        A         LDA    DIVDND
010C B2 5D        A         SBC    DIVSR
010E 24 09     0119         BCC    ZOT       BRANCH IF CARRY SET (BEFORE SAVE OF DIVDND)
0110 B6 60        A         LDA    DIVDND+1  ADD IT BACK
0112 BB 5E        A         ADD    DIVSR+1   NOTE, MSB WAS NEVER STORED SO WE ONLY
0114 B7 60        A         STA    DIVDND+1  HAVE TO ADD TO THE LS BYTE
0116 58                     LSLX             SHIFT IN ZERO
0117 20 04     011D         BRA    OVER
0119 B7 5F        A ZOT     STA    DIVDND    SAVE MS BYTE OF NEW DIVIDEND
011B 99                     SEC
011C 59                     ROLX             SHIFT 1 BIT INTO QUOTIENT
011D 49              OVER   ROLA             CARRY INTO QUOTIENT
011E 34 5D        A         LSR    DIVSR     SHIFT DIVISOR RIGHT BY 1
0120 36 5E        A         ROR    DIVSR+1
0122 3A 61        A         DEC    TEMP      DONE
0124 26 DE     0104         BNE    LOOP      BRANCH IF NOT
0126 44                     LSRA             GET CORRECT QUOTIENT
0127 56                     RORX             C = ROUND BIT
0128 9F                     TXA
0129 A9 00        A         ADC    #0        AND ROUND
        012B      A DIVOUT  EQU    *
012B 81                     RTS              EXIT
```

Figure 3-14. 16-bit ÷ 16-bit With 8-bit Result Subroutine Example (Continued)

```
          DIVIDE  .SA:1

00001                       ***********************************************************************
00002                       *
00003                       *     DIVIDE:  16 Bit / 16 Bit Divide Routine with 16 Bit Result
00004                       *          Reference DIV16 program in the 6800 User's Group Library.
00005                       *          16 Bit Dividend in DVDND and DVDND+1
00006                       *          16 Bit Divisor in DVSOR and DVSOR+1
00007                       *          16 Bit Result in DVDND and DVDND+1
00008                       *
00009                       *=====================================================================
00010                       *
00011A 0040                        ORG    $40
00012                       *
00013A 0040    0001      A COUNT  RMB    1
00014A 0041    0002      A DVSOR  RMB    2
00015A 0043    0002      A DVDND  RMB    2
00016A 0045    0001      A TEMPA  RMB    1
00017                       *
00018                       *
00019                       *=====================================================================
00020                       *
00021A 0100                        ORG    $100
00022                       *
00023A 0100 A6 01         A DIVIDE LDA    #1
00024A 0102 3D 41         A        TST    DVSOR
00025A 0104 2B 0B      0111        BMI    DIV153
00026A 0106 4C              DIV151 INCA
00027A 0107 38 42         A        ASL    DVSOR+1
00028A 0109 39 41         A        ROL    DVSOR
00029A 010B 2B 04      0111        BMI    DIV153
00030A 010D A1 11         A        CMP    #17
00031A 010F 26 F5      0106        BNE    DIV151
00032A 0111 B7 40         A DIV153 STA    COUNT    Save the Counter
00033A 0113 B6 43         A        LDA    DVDND
00034A 0115 BE 44         A        LDX    DVDND+1
00035A 0117 3F 43         A        CLR    DVDND
00036A 0119 3F 44         A        CLR    DVDND+1
00037A 011B B7 45         A DIV163 STA    TEMPA
00038A 011D 9F                     TXA
00039A 011E B0 42         A        SUB    DVSOR+1
00040A 0120 97                     TAX
00041A 0121 B6 45         A        LDA    TEMPA
00042A 0123 B2 41         A        SBC    DVSOR
00043A 0125 24 0D      0134        BCC    DIV165   Divisor still OK
00044A 0127 B7 45         A        STA    TEMPA    Divisor too large
00045A 0129 9F                     TXA
00046A 012A BB 42         A        ADD    DVSOR+1
00047A 012C 97                     TAX
00048A 012D B6 45         A        LDA    TEMPA
00049A 012F B9 41         A        ADC    DVSOR
00050A 0131 98                     CLC
00051A 0132 20 01      0135        BRA    DIV167
00052A 0134 99              DIV165 SEC
00053A 0135 39 44         A DIV167 ROL    DVDND+1
00054A 0137 39 43         A        ROL    DVDND
00055A 0139 34 41         A        LSR    DVSOR    Adjust Divisor
00056A 013B 36 42         A        ROR    DVSOR+1
00057A 013D 3A 40         A        DEC    COUNT
00058A 013F 26 DA      011B        BNE    DIV163
00059A 0141 81                     RTS
00060                       *
00061                       *
00062                       ***********************************************************************
00063                              END
```

Figure 3-15. 16-bit ÷ 16-Bit With 16-Bit Result Subroutine Example

3.9 ASSIST05 DEBUG MONITOR

Debug monitor ASSIST05 is a monitor which is intended for use with the MC146805E2 Microprocessor Unit (MPU). The ASSIST05 monitor uses an RS-232 interface to allow users to quickly perform hardware and software development and evaluation. Figure 3-16 contains a schematic diagram of one possible circuit that could be used to implement ASSIST05. The program listing for ASSIST05 is provided in Figure 3-17.

The serial interface shown in Figure 3-16 is provided by an MC6850 ACIA. However, this serial hardware, and the CHRIN and CHROUT subroutines of ASSIST05, could be replaced by hardware shown in Figure 3-4 and the GETC and PUTC subroutines of Figure 3-5. All M6805 HMOS/M146805 CMOS Family MCU evaluation devices include debug monitors which can be used with an RS-232 interface as discussed in the Serial I/O Software For RS-232 paragraph. If, in the case of an MC146805E2 MPU, a debug monitor that does not require an RS-232 interface is desired, Motorola Application Note AN-823 or AN-823A can be used. This application note describes a debug monitor for the MC146805E2 which uses a keypad and LCD for the user interface.

The ASSIST05 program includes commands which allow memory and register examine/change, breakpoint set/point/display, single or multiple trace, and tape punch/load. In the paragraphs which follow, each of the commands is described in greater detail, and some of the routines in ASSIST05, which might be useful in other programs, are also discussed.

3.9.1 ASSIST05 Command Description

The ASSIST05 program is initialized by either a power-on or manual reset to the MC146805E2. After a reset, "ASSIST05 1.0" is printed and the prompt character ">" is displayed to indicate that commands may be entered.

Table 3-2 summarizes the commands which may be entered. Commands are entered by typing the command, as shown in Table 3-2, followed by a carriage return.

Table 3-2. ASSIST05 Valid Display Commands

Command	Usage
R	Display all Register Contents
A	Display/Change User Accumulator Contents
X	Display/Change User Index Register Contents
C	Display/Change User Condition Code Register Contents
P XXXX	Change User Program Counter Contents
W XXXX YYYY	Write Memory to Tape
B	Display Breakpoints
B N XXXX	Set Breakpoint #N
B N O	Clear Breakpoint #N
T	Trace One Instruction
T XXXX	Trace XXXX Instruction
M XXXX	Display/Change Memory
G	Continue Program Executive at Current Program Counter
GXXXX	Execute Program at Address XXXX

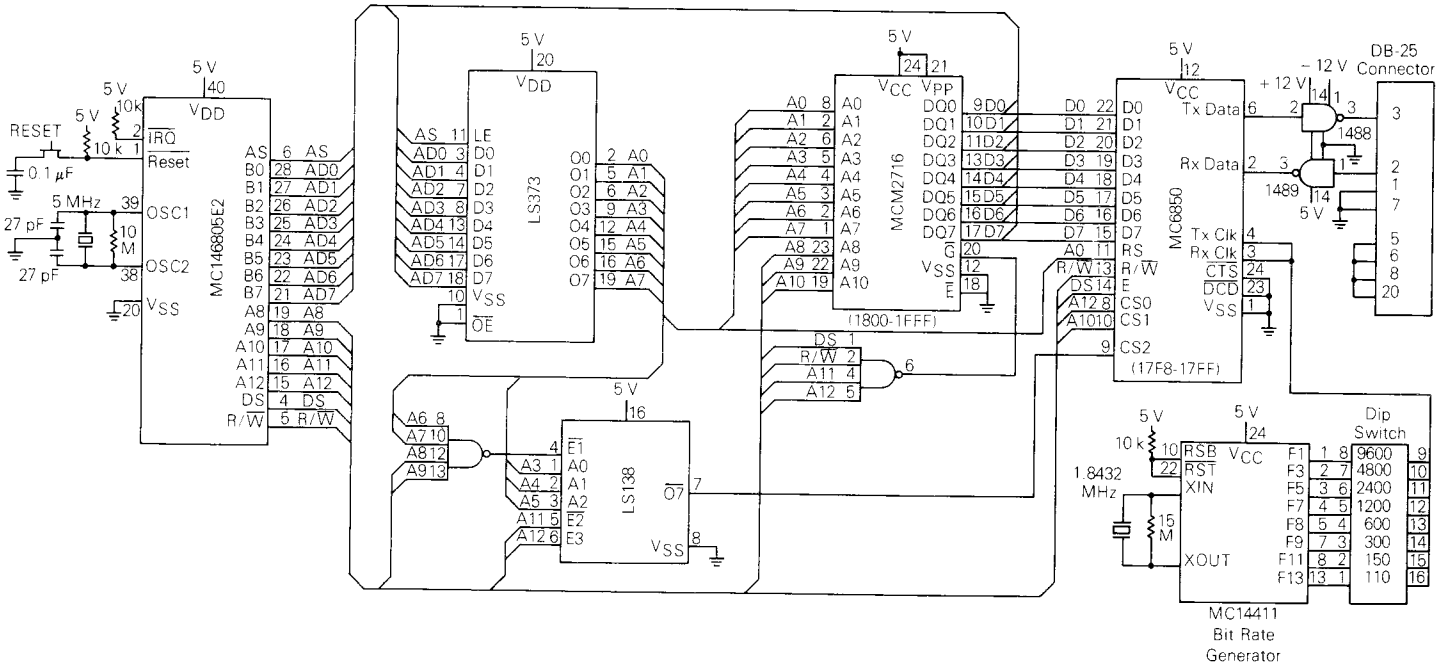


Figure 3-16. ASSIST05 Interface Schematic Diagram

3.9.2 Detailed Command Description

Register Examine/Change
The current user register contents may be displayed all at once or individually (except SP). The SP may not be directly modified by the user. The PC may be modified with either P or G commands.

R — Display Registers
Current user registers are displayed in the following format: PC A X C SP. After the registers have been displayed, the prompt character is returned.

A — Display/Change the Accumulator
This command begins by printing the current contents of the accumulator in hexadecimal. The user may then enter a new value in hexadecimal or a carriage return to terminate the command.

X — Display/Change the Index Register
This procedure is the same as the A command, but affects the index register instead.

C — Display/Change the Condition Code Register
This procedure is the same as the A command, but affects the condition code register instead.

Tape Punch/Load
This allows the user to load a tape, via the RS-232 interface, in the Motorola S1-S9 format. Memory is then loaded with the data which is contained in the file on the tape. Files in S1-S9 format have the destination addresses contained within the file. The format of this command is: W XXXX YYYY. The memory contents of addresses XXXX to YYYY are output to the RS-232 port. Data is then stored onto tape in the Motorola S1-S9 format.

Breakpoints
Up to three breakpoints may be used to allow debugging of user programs. The program execution can be halted at specified addresses so that the current user registers and memory may be examined and evaluated. Whenever program execution reaches a breakpoint address, program execution ceases, the current user registers are displayed, and the prompt character is returned. Following this, the applicable breakpoint command can then be entered. Breakpoints may only be entered from valid RAM addresses. The current program counter is not displayed; however, it may be examined by using the R command.

P — Display the Program Counter
This procedure is similar to the A command, but affects the program counter instead. Note that this is a two-byte register.

NOTE

When the user program execution is initiated via the G command, instructions up to and including the instruction at the breakpoint are executed. If the user program execution is halted with a reset, all enabled breakpoint address locations will contain $83 and should be reloaded.

B — Display Breakpoints
This command allows all breakpoint addresses to be displayed and the prompt character is returned.

B N XXXX — Set Breakpoint #N
This command enables breakpoint N, where N is a number 0-2 at address XXXX, and where XXXX is the address of the last instruction to be executed before returning to ASSIST05.

B N 0 — Clear Breakpoint #N
This command disables breakpoint N, where N is a number 0-2.

Instruction Trace
This command is used to execute one or more instructions, and is generally used after a breakpoint is reached. Tracing may also be used to step through ROM-based programs; however, unlike breakpoints, tracing is not done in real-time. To use the trace command on ROM-based programs, the user must put a jump-to-the ROM entry address in RAM. The user then sets a breakpoint at the jump instruction address. Once the breakpoint address is encountered, the jump is executed and control is returned to ASSIST05. The current user PC then points to the ROM entry address and tracing may then be used.

T — Trace One Instruction
With this command, a single instruction, located at the user PC, is executed and the registers are then displayed. Control is then returned to ASSIST05.

T XXXX — Trace XXXX Instructions
With this command, XXXX instructions are executed, beginning at the current user PC. After the specified number of instructions are executed, the registers are displayed and control is returned to ASSIST05. The instructions executed during the trace instruction are not executed in real-time. The P instruction may be used prior to tracing to point to the first instruction to be executed.

Memory Examine/Change
This command allows memory, at the specified address XXXX, to be examined. Then, if desired, the contents of that location may be changed, or the previous location or next location may be examined.

M XXXX — Display/Change Memory
With this command, memory locations XXXX, XXXX – 1, or XXXX + 1 may be acted upon. To do this or to return to ASSIST05, one of four terminal keys need be depressed. These include:

↑—to examine previous location (XXXX – 1)
LF—to examine next location (XXXX + 1)
HH—to change contents of specified location XXXX
CR—to exit memory examine/change command and return to ASSIST05.

Execute User Program

Two execute commands are used to allow real-time execution of the user program. Execution can continue either from the current user PC or begin at a specified address.

G — Continue Program Execution at Current PC

This command allows the user program to continue execution from the current user PC. This command is usually used after a breakpoint has been executed or if the user has previously altered the PC with the P instruction.

G XXXX — Execute Program at Address XXXX

This command results in the user PC being loaded with address XXXX. The user program then starts execution from the new (current) user PC.

3.9.3 ASSIST05 Routines

The ASSIST05 program contains many useful routines and subroutines which might be used in other programs. Some of the more unusual includes a routine that can find the current SP (LOCSTK) and the trace routine which allows ROM-based code to be debugged. For more information consult the complete listing which is in Figure 3-17.

```
00001                          NAM    ASSIST05


00003                 ***************************************************
00004                 *    MONITOR FOR THE AUSTIN 6805 EVALUATION MODULE*
00005                 *          (C) COPYRIGHT 1979 MOTOROLA INC.       *
00006                 ***************************************************


00008                 ***************************************************
00009                 *
00010                 *      THE MONITOR HAS THE FOLLOWING COMMANDS:
00011                 *
00012                 *      R            --  PRINT REGISTERS
00013                 *
00014                 *      A            --  DISPLAY/CHANGE A REGISTER
00015                 *
00016                 *      X            --  DISPLAY/CHANGE X REGISTER
00017                 *
00018                 *      C            --  DISPLAY/CHANGE CONDITION CODE
00019                 *
00020                 *      P            --  DISPLAY/CHANGE PROGRAM COUNTER
00021                 *
00022                 *      L            --  LOAD TAPE FILE INTO MEMORY
00023                 *
00024                 *      W XXXX YYYY --  WRITE MEMORY TO TAPE FILE
00025                 *
00026                 *      B            --  DISPLAY BREAKPOINTS
00027                 *      B N XXXX     --  SET BREAKPOINT NUMBER N
00028                 *      B N 0        --  CLEAR BREAKPOINT NUMBER N
00029                 *
00030                 *      T            --  TRACE ONE INSTRUCTION
00031                 *      T XXXX       --  TRACE XXX INSTRUCTIONS
00032                 *
00033                 *      M XXXX       --  MEMORY EXAMINE/CHANGE.
00034                 *            TYPE:   ↑ -- TO EXAMINE PREVIOUS
00035                 *                   LF -- TO EXAMINE NEXT
00036                 *                   HH -- CHANGE TO HEX DATA
00037                 *                   CR -- TERMINATE COMMAND
00038                 *
00039                 *      G            --  CONTINUE PROGRAM EXECUTION FROM
00040                 *                           CURRENT PROGRAM COUNTER.
00041                 *      G XXXX       --  GO EXECUTE PROGRAM AT SPECIFIED
00042                 *                           ADDRESS.
00043                 *
00044                 ***************************************************


00046                 ***************************************************
00047                 *      MC146805E2 GLOBAL PARAMETERS               *
00048                 ***************************************************
00049        1800     A MONSTR EQU    $1800    START OF MONITOR
```

Figure 3-17. ASSIST05 Program Listing

```
00050          001F     A PCMASK EQU    $1F       MASK OFF FOR 8K ADDRESS SPACE (E2)
00051          0003     A NUMBKP EQU    3         NUMBER OF BREAKPOINTS
00052          17F8     A ACIA   EQU    $17F8     ACIA ADDRESS
00053          003E     A PROMPT EQU    '>        PROMPT CHARACTER
00054          0008     A TIMER  EQU    8         TIMER DATA REGISTER
00055          0009     A TIMEC  EQU    9         TIMER CONTROL REGISTER


00057                     ************************************************
00058                     *       EQUATES                                *
00059                     ************************************************
00060          0004     A EOT    EQU    $04       END OF TEXT
00061          000D     A CR     EQU    $0D       CARRIAGE RETURN
00062          000A     A LF     EQU    $0A       LINE FEED
00063          0011     A DC1    EQU    $11       READER ON CONTROL FUNCTION
00064          0012     A DC2    EQU    $12       PUNCH ON CONTROL FUNCTION
00065          0013     A DC3    EQU    $13       X-OFF CONTROL FUNCTION
00066          0014     A DC4    EQU    $14       STOP CONTROL FUNCTION
00067          0020     A SP     EQU    $20       SPACE
00068          0007     A BELL   EQU    $07       CONTROL-G (BELL)
00069          0083     A SWIOP  EQU    $83       SOFTWARE INTERRUPT OPCODE
00070          00CC     A JMPOP  EQU    $CC       EXTENDED JUMP OPCODE


00072                     ************************************************
00073                     * MONITOR WORK AREA AT STACK BOTTOM
00074                     ************************************************
00075A 0041                        ORG    $41       BOTTOM OF STACK
00076          0038     A BKPTBL EQU    *-3*NUMBKP BKPT TABLE UNDER STACK BOTTOM
00077A 0041    0001     A SWIFLG RMB    1         SWI FUNCTION FLAG
00078A 0042    0001     A WORK1  RMB    1         CHRIN/LOAD/STORE/PUTBYT
00079A 0043    0001     A WORK2  RMB    1         LOAD/STORE/PUTBYT
00080A 0044    0001     A ADDRH  RMB    1         HIGH ADDRESS BYTE
00081A 0045    0001     A ADDRL  RMB    1         LOW ADDRESS BYTE
00082A 0046    0001     A WORK3  RMB    1         LOAD/STORE/PUNCH
00083A 0047    0001     A WORK4  RMB    1         STORE/PUNCH
00084A 0048    0001     A WORK5  RMB    1         TRACE
00085A 0049    0001     A WORK6  RMB    1         TRACE
00086A 004A    0001     A WORK7  RMB    1         TRACE
00087A 004B    0001     A PNCNT  RMB    1         PUNCH BREAKPOINT
00088A 004C    0002     A PNRCNT RMB    2         PUNCH
00089A 004E    0001     A CHKSUM RMB    1         PUNCH
00090A 004F    000C     A VECRAM RMB    12        VECTORS


00092A 1800                        ORG    MONSTR    START OF MONITOR
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00094                          ************************************************
00095                          * MONITOR BASE STRING/TABLE PAGE
00096                          * (MUST BE AT THE BEGINNING OF A PAGE)
00097                          ************************************************
00098            1800        A MBASE  EQU    *         START OF WORK PAGE IN ROM
00099                          * MSGUP MUST BE FIRST IN PAGE
00100A 1800      41          A MSGUP  FCC    /ASSIST05 1.1/FIREUP MESSAGE
00101A 180C      04          A MSGNUL FCB    EOT       END OF STRING
00102A 180D      3F          A MSGERR FCC    /? ERROR ?/
00103A 1816      04          A        FCB    EOT
00104A 1817      53          A MSGS1  FCB    'S,'1,EOT S1 START RECORD TEXT
00105A 181A      53          A MSGS9  FCC    /S9030000FC/
00106A 1824      0D          A        FCB    CR        S9 RECORD TEXT
00107A 1825      14          A MSGMOF FCB    DC4,DC3,EOT MOTORS OFF TEXT
00108A 1828      49          A MSGWAS FCC    /IS OPCODE/
00109A 1831      04          A        FCB    EOT
00110A 1832      CC          A VECTAB FCB    JMPOP
00111A 1833      1C85        A        FDB    TIRQ
00112A 1835      CC          A        FCB    JMPOP
00113A 1836      1C85        A        FDB    TIRQ
00114A 1838      CC          A        FCB    JMPOP
00115A 1839      1CA1        A        FDB    IRQ
00116A 183B      CC          A        FCB    JMPOP
00117A 183C      1A94        A        FDB    SWI


00119                          ************************************************
00120                          *       GO --- START EXECUTION
00121                          ************************************************
00122A 183E CD 19A9          A CMDG   JSR    GETADR    OBTAIN INPUT ADDRESS
00123A 1841 24 14       1857          BCC    NEXT      DO CONTINUE IF NONE
00124A 1843 B6 44            A        LDA    ADDRH     CHECK ADDRESS BOUNDRIES
00125A 1845 A1 20            A        CMP    #$20      FOR OVERRUN
00126A 1847 25 03       184C          BLO    GADDR
00127A 1849 CC 1A74          A        JMP    CMDERR    ERROR IF $2000 OR LARGER
00128A 184C CD 1B23          A GADDR  JSR    LOCSTK    OBTAIN CURRENT STACK ADDRESS-3
00129A 184F B6 44            A        LDA    ADDRH     LOAD PC HIGH
00130A 1851 E7 07            A        STA    7,X       INTO STACK
00131A 1853 B6 45            A        LDA    ADDRL     LOAD PC LOW
00132A 1855 E7 08            A        STA    8,X       INTO STACK
00133A 1857 0E 47 03    185D NEXT     BRSET  7,WORK4,CONT
00134A 185A CC 1A26          A        JMP    CMD
00135A 185D CD 1CC3          A CONT   JSR    SCNBKP    INIT BREAKPOINT SCAN PARMS
00136A 1860 F6                GOINSB LDA    ,X        LOAD HI BYTE
00137A 1861 2B 10       1873          BMI    GONOB     BRA EMPTY
00138A 1863 B7 44            A        STA    ADDRH     STORE HI ADDRESS
00139A 1865 E6 01            A        LDA    1,X       LOAD LOW
00140A 1867 B7 45            A        STA    ADDRL     STORE LOW
00141A 1869 CD 1943          A        JSR    LOAD      LOAD OPCODE
00142A 186C E7 02            A        STA    2,X       STORE INTO TABLE
00143A 186E A6 83            A        LDA    #SWIOP    REPLACE WITH OPCODE
00144A 1870 CD 1952          A        JSR    STORE     STORE IN PLACE
00145A 1873 5C                GONOB  INCX              TO
00146A 1874 5C                       INCX              NEXT
00147A 1875 5C                       INCX              BREAKPOINT
00148A 1876 3A 4B            A        DEC    PNCNT     COUNT DOWN
00149A 1878 26 E6       1860          BNE    GOINSB    LOOP IF MORE
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00150A 187A 33 41        A        COM    SWIFLG   FLAG BREAKPOINTS ARE IN
00151                       ****RESET USERS TIMER ENVIRONMENT******
00152A 187C 80                   RTI                RESTART PROGRAM


00154                       ********************************************
00155                       *  CLBYTE - LOAD SUBROUTINE TO READ NEXT   *
00156                       *           BYTE, ADJUST CHECKSUM,         *
00157                       *           DECREMENT COUNT.               *
00158                       * OUTPUT: A=BYTE                           *
00159                       *         CC=REFLECTS COUNT DECREMENT      *
00160                       ********************************************
00161A 187D AD 0C     188B CLBYTE BSR    GETBYT   OBTAIN NEXT BYTE
00162A 187F 24 57     18D8        BCC    CMDMIN   ERROR IF NONE
00163A 1881 B7 43        A        STA    WORK2    SAVE VALUE
00164A 1883 BB 4E        A        ADD    CHKSUM   ADD TO CHECKSUM
00165A 1885 B7 4E        A        STA    CHKSUM   REPLACE
00166A 1887 B6 43        A        LDA    WORK2    RELOAD BYTE VALUE
00167A 1889 5A                    DECX            COUNT DOWN
00168A 188A 81                    RTS             RETURN TO CALLER


00170                       ********************************************
00171                       *  GETBYT - READ BYTE IN HEX SUBROUTINE    *
00172                       * OUTPUT: C=0, Z=1 NO NUMBER               *
00173                       *         C=0, Z=0 INVALID NUMBER          *
00174                       *         C=1, Z=1, A=BINARY BYTE VALUE    *
00175                       ********************************************
00176A 188B CD 198E      A GETBYT JSR    GETNYB   GET HEX DIGIT
00177A 188E 24 0E     189E        BCC    GETBRZ   RETURN NO NUMBER
00178A 1890 48                GETBY2 ASLA         SHIFT
00179A 1891 48                       ASLA         OVER
00180A 1892 48                       ASLA         BY
00181A 1893 48                       ASLA         FOUR
00182A 1894 B7 43        A           STA    WORK2    SAVE HIGH HEX DIGIT
00183A 1896 CD 198E      A           JSR    GETNYB   GET LOW HEX
00184A 1899 4D                       TSTA            FORCE Z=0 (DELIMITER IF INVALID)
00185A 189A 24 04     18A0           BCC    GETBRT   RETURN IF INVALID NUMBER
00186A 189C BA 43        A           ORA    WORK2    COMBINE HEX DIGITS
00187A 189E 3F 43        A GETBRZ CLR    WORK2    SET Z=1
00188A 18A0 81                GETBRT RTS          RETURN TO CALLER


00190                       ********************************************
00191                       *  L -- LOAD FILE INTO MEMORY COMMAND      *
00192                       ********************************************
00193A 18A1 CD 19D0      A CMDL   JSR    CHRIN    READ CARRIAGE RETURN
00194A 18A4 A6 11        A        LDA    #DC1     TURN ON READER
00195A 18A6 CD 19EA      A        JSR    CHROUT   WITH DC1 CONTROL CODE
00196                       * SEARCH FOR AN 'S'
00197A 18A9 CD 19D0      A CMDLT  JSR    CHRIN    READ A CHARACTER
00198A 18AC A1 53        A CMDLSS CMP    #'S      ? 'S'
00199A 18AE 26 F9     18A9        BNE    CMDLT    LOOP IF NOT
00200A 18B0 CD 19D0      A        JSR    CHRIN    READ SECOND CHARACTER
00201A 18B3 A1 39        A        CMP    #'9      ? 'S9' RECORD
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00202A 18B5 27 24    18DB        BEQ    CLEOF     BRANCH END OF FILE
00203A 18B7 A1 31       A        CMP    #'1       ? 'S1' RECORD
00204A 18B9 26 F1    18AC        BNE    CMDLSS    NO, TRY 'S' AGAIN
00205                          * READ ADDRESS AND COUNT
00206A 18BB 3F 4E       A        CLR    CHKSUM    ZERO CHKSUM
00207A 18BD AD BE    187D        BSR    CLBYTE    OBTAIN SIZE OF RECORD
00208A 18BF 97                   TAX              START COUNTDOWN IN X REGISTER
00209A 18C0 AD BB    187D        BSR    CLBYTE    OBTAIN START OF ADDRESS
00210A 18C2 B7 44       A        STA    ADDRH     STORE IT
00211A 18C4 AD B7    187D        BSR    CLBYTE    OBTAIN LOW ADDRESS
00212A 18C6 B7 45       A        STA    ADDRL     STORE IT
00213                          * NOW LOAD TEXT
00214A 18C8 AD B3    187D CLLOAD BSR    CLBYTE    NEXT CHARACTER
00215A 18CA 27 08    18D4        BEQ    CLEOR     BRANCH IF COUNT DONE
00216A 18CC CD 1952     A        JSR    STORE     STORE CHARACTER
00217A 18CF CD 1962     A        JSR    PTRUP1    UP ADDRESS POINTER
00218A 18D2 20 F4    18C8        BRA    CLLOAD    LOOP UNTIL COUNT DEPLETED
00219                          * END OF RECORD
00220A 18D4 3C 4E       A CLEOR  INC    CHKSUM    TEST VALID CHECKSUM
00221A 18D6 27 C9    18A1        BEQ    CMDL      CONTINUE IF SO
00222A 18D8 CC 1A74     A CMDMIN JMP    CMDERR    ERROR IF INVALID
00223                          * END OF FILE
00224A 18DB AD A0    187D CLEOF  BSR    CLBYTE    READ S9 LENGTH
00225A 18DD 97                   TAX              PREPARE S9 FLUSH COUNT
00226A 18DE AD 9D    187D CLEOFL BSR    CLBYTE    SKIP HEX PAIR
00227A 18E0 26 FC    18DE        BNE    CLEOFL    BRANCH MORE
00228A 18E2 AE 25       A        LDX    #MSGMOF-MBASE TURN MOTORS OUT
00229A 18E4 CC 1A23     A        JMP    CMDPDT    SEND AND END COMMAND


00231                          ********************************************
00232                          *  M -- EXAMINE/CHANGE MEMORY              *
00233                          *  MCHNGE -- REGISTER CHANGE ENTRY POINT   *
00234                          ********************************************
00235A 18E7 CD 19A9     A CMDM   JSR    GETADR    OBTAIN ADDRESS VALUE
00236A 18EA 24 EC    18D8        BCC    CMDMIN    INVALID IF NO ADDRESS
00237A 18EC B6 44       A        LDA    ADDRH     CHECK ADDRESS
00238A 18EE A1 20       A        CMP    #$20      FOR OVERRUN
00239A 18F0 25 03    18F5        BLO    CMDMLP
00240A 18F2 CC 1A74     A        JMP    CMDERR    ERROR IF $2000 OF LARGER
00241A 18F5 CD 1B16     A CMDMLP JSR    PRTADR    PRINT OUT ADDRESS AND SPACE
00242A 18F8 AD 49    1943 MCHNGE BSR    LOAD      LOAD BYTE INTO A REGISTER
00243A 18FA CD 1B1D     A        JSR    CRBYTS    PRINT WITH SPACE
00244A 18FD CD 198E     A        JSR    GETNYB    SEE IF CHANGE WANTED
00245A 1900 24 0D    190F        BCC    CMDMDL    BRANCH NO
00246A 1902 AD 8C    1890        BSR    GETBY2    OBTAIN FULL BYTE
00247A 1904 26 D2    18D8        BNE    CMDMIN    TERMINATE IF INVALID HEX
00248A 1906 24 07    190F        BCC    CMDMDL    BRANCH IF OTHER DELIMITER
00249A 1908 AD 48    1952        BSR    STORE     STORE NEW VALUE
00250A 190A 25 CC    18D8        BCS    CMDMIN    BRANCH IF STORE FAILS
00251A 190C CD 19D0     A        JSR    CHRIN     OBTAIN DELIMITER
00252                          * CHECK OUT DELIMITERS
00253A 190F A1 0A       A CMDMDL CMP    #LF       ? TO NEXT BYTE
00254A 1911 27 1D    1930        BEQ    CMDMLF    BRANCH IF SO
00255A 1913 A1 5E       A        CMP    #'↑       ? TO PREVIOUS BYTE
00256A 1915 27 03    191A        BEQ    CMDMBK    BRANCH YES
00257A 1917 CC 1A29     A        JMP    CMDNNL    ENTER COMMAND HANDLER
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00258A 191A 3D 45        A CMDMBK TST    ADDRL    ? LOW BYTE ZERO
00259A 191C 26 0C     192A         BNE    CMDMB2   NO, JUST DOWN IT
00260A 191E 3A 44        A         DEC    ADDRH    DOWN HIGH FOR CARRY
00261A 1920 B6 44        A         LDA    ADDRH    CHECK ADDRESS
00262A 1922 A1 FF        A         CMP    #$FF     FOR UNDERFLOW
00263A 1924 26 04     192A         BNE    CMDMB2
00264A 1926 A6 1F        A         LDA    #$1F     CLEAR ADDRESS ON UNDERFLOW
00265A 1928 B7 44        A         STA    ADDRH
00266A 192A 3A 45        A CMDMB2 DEC    ADDRL    DOWN LOW BYTE
00267A 192C AD 51     197F         BSR    PCRLF    TO NEXT LINE
00268A 192E 20 C5     18F5         BRA    CMDMLP   TO NEXT BYTE
00269A 1930 A6 0D        A CMDMLF LDA    #CR      SEND JUST CARRIAGE RETURN
00270A 1932 CD 19F2      A         JSR    CHROU2   OUTPUT IT
00271A 1935 AD 2B     1962         BSR    PTRUP1   UP POINTER BY ONE
00272A 1937 B6 44        A         LDA    ADDRH    CHECK ADDRESS
00273A 1939 A1 20        A         CMP    #$20     FOR OVERRUN
00274A 193B 25 B8     18F5         BLO    CMDMLP
00275A 193D 3F 44        A         CLR    ADDRH    IF LARGER CLEAR
00276A 193F 3F 45        A         CLR    ADDRL    ADDRESS
00277A 1941 20 B2     18F5         BRA    CMDMLP   TO NEXT BYTE


00279                    ***********************************************
00280                    *       LOAD - LOAD INTO A FROM ADDRESS IN    *
00281                    *              POINTER ADDRH/ADDRL            *
00282                    * INPUT: ADDRH/ADDRL=ADDRESS                  *
00283                    * OUTPUT: A=BYTE FROM POINTED LOCATION        *
00284                    * X IS TRANSPARENT                            *
00285                    * WORK1,WORK2,WORK3 USED                      *
00286                    ***********************************************
00287A 1943 BF 42        A LOAD   STX    WORK1    SAVE X
00288A 1945 AE C6        A         LDX    #$C6     C6=LDA 2-BYTE EXTENDED

00290A 1947 BF 43        A LDSTCM STX    WORK2    PUT OPCODE IN PLACE
00291A 1949 AE 81        A         LDX    #$81     81=RTS
00292A 194B BF 46        A         STX    WORK3    NOW THE RETURN
00293A 194D BD 43        A         JSR    WORK2    EXECUTE BUILT ROUTINE
00294A 194F BE 42        A         LDX    WORK1    RESTORE X
00295A 1951 81                     RTS             AND EXIT


00297                    ***********************************************
00298                    *       STORE - STORE A AT ADDRESS IN POINTER*
00299                    *               ADDRH/ADDRL                  *
00300                    * INPUT: A=BYTE TO STORE                      *
00301                    *        ADDRH/ADDRL=ADDRESS                  *
00302                    * OUTPUT: C=0 STORE WENT OK                   *
00303                    *          C=1 STORE DID NOT TAKE (NOT RAM)   *
00304                    * REGISTERS TRANSPARENT                       *
00305                    * (A NOT TRANSPARENT ON INVALID STORE)        *
00306                    * WORK1,WORK2,WORK3,WORK4 USED                *
00307                    ***********************************************
00308A 1952 BF 42        A STORE  STX    WORK1    SAVE X
00309A 1954 AE C7        A         LDX    #$C7     C7=STA 2-EXTENDED
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00310A 1956 AD EF    1947          BSR    LDSTCM    CALL STORE ROUTINE
00311A 1958 B7 47       A          STA    WORK4     SAVE VALUE STORED
00312A 195A AD E7    1943          BSR    LOAD      ATTEMPT LOAD
00313A 195C B1 47       A          CMP    WORK4     ? VALID STORE
00314A 195E 27 01    1961          BEQ    STRTS     BRANCH IF VALID
00315A 1960 99                     SEC              SHOW INVALID STORE
00316A 1961 81                STRTS RTS             RETURN


00318                         ******************************************
00319                         *      PTRUP1 - INCREMENT MEMORY POINTER  *
00320                         ******************************************
00321A 1962 3C 45       A PTRUP1 INC    ADDRL     INCREMENT LOW BYTE
00322A 1964 26 02    1968          BNE    PRTRTS    NON-ZERO MEANS  NO CARRY
00323A 1966 3C 44       A          INC    ADDRH     INCREMENT HIGH BYTE
00324A 1968 81                PRTRTS RTS             RETURN TO CALLER


00326                         ******************************************
00327                         *      PUTBYT --- PRINT A IN HEX          *
00328                         *  X TRANSPARENT                          *
00329                         *  WORK1 USED                             *
00330                         ******************************************
00331A 1969 B7 42       A PUTBYT STA    WORK1     SAVE A
00332A 196B 44                     LSRA             SHIFT TO
00333A 196C 44                     LSRA             LEFT HEX
00334A 196D 44                     LSRA             DIGIT
00335A 196E 44                     LSRA             SHIFT HIGH NYBBLE DOWN
00336A 196F AD 02    1973          BSR    PUTNYB    PRINT IT
00337A 1971 B6 42       A          LDA    WORK1
00338                         * FALL INTO PUTNYB


00340                         ************************************************
00341                         *      PUTNYB --- PRINT LOWER NYBBLE OF A IN HEX*
00342                         *  A,X TRANSPARENT                             *
00343                         ************************************************
00344A 1973 A4 0F       A PUTNYB AND    #$F       MASK OFF HIGH NYBBLE
00345A 1975 AB 30       A          ADD    #'0       ADD ASCII ZERO
00346A 1977 A1 39       A          CMP    #'9       CHECK FOR A-F
00347A 1979 23 6F    19EA          BLS    CHROUT    OK, SEND OUT
00348A 197B AB 07       A          ADD    #'A-'9-1  ADJUSTMENT FOR HEX A-F
00349A 197D 20 6B    19EA          BRA    CHROUT    NOW SEND OUT


00351                         ***************************************************
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00352                          *  PDATA  - PRINT MONITOR STRING AFTER CR/LF
00353                          *  PDATA1 - PRINT MONITOR STRING
00354                          *  PCRLF  - PRINT CARRIAGE RETURN AND LINE FEED
00355                          * INPUT: X=OFFSET TO STRING IN BASE PAGE (UNUSED FOR PCRL
00356                          ************************************************
00357A 197F AE 0C          A PCRLF  LDX    #MSGNUL-MBASE LOAD NULL STRING ADDRESS
00358A 1981 A6 0D          A PDATA  LDA    #CR       PREPARE CARRIAGE RETURN
00359A 1983 AD 65       19EA PDLOOP BSR    CHROUT    SEND NEXT CHARACTER
00360A 1985 D6 1800        A PDATA1 LDA    MBASE,X   LOAD NEXT CHARACTER
00361A 1988 5C                      INCX              BUMP POINTER UP ONE
00362A 1989 A1 04          A        CMP    #EOT      ? END OF STRING
00363A 198B 26 F6       1983        BNE    PDLOOP    BRANCH NO
00364A 198D 81                      RTS              RETURN DONE


00366                          ****************************************
00367                          * GETNYB - OBTAIN NEXT HEX CHARACTER    *
00368                          * OUTPUT: C=0 NOT HEX INPUT, A=DELIMITER *
00369                          *         C=1 HEX INPUT, A=BINARY VALUE  *
00370                          *  X TRANSPARENT                         *
00371                          *  WORK1 IN USE                          *
00372                          ****************************************
00373A 198E CD 19D0        A GETNYB JSR    CHRIN     OBTAIN CHARACTER
00374A 1991 A1 30          A        CMP    #'0       ? LOWER THAN ZERO
00375A 1993 25 12       19A7        BLO    GETNCH    BRANCH NOT HEX
00376A 1995 A1 39          A        CMP    #'9       ? HIGHER THAN NINE
00377A 1997 23 0A       19A3        BLS    GETNHX    BRANCH IF 0 THRU 9
00378A 1999 A1 41          A        CMPA   #'A       ? LOWER THAN AN "A"
00379A 199B 25 0A       19A7        BLO    GETNCH    BRANCH NOT HEX
00380A 199D A1 46          A        CMPA   #'F       ? HIGHER THAN AN "F"
00381A 199F 22 06       19A7        BHI    GETNCH    BRANCH NOT HEX
00382A 19A1 A0 07          A        SUB    #7        ADJUST TO $A OFFSET
00383A 19A3 A4 0F          A GETNHX AND    #$0F      CLEAR ASCII BITS
00384A 19A5 99                      SEC              SET CARRY
00385A 19A6 81                      RTS              RETURN
00386A 19A7 98                GETNCH CLC              CLEAR CARRY FOR NO HEX
00387A 19A8 81                      RTS              RETURN


00389                          ****************************************
00390                          *     GETADR - BUILD ANY SIZE BINARY     *
00391                          *              NUMBER FROM INPUT.        *
00392                          *              LEADING BLANKS SKIPPED.   *
00393                          * OUTPUT: CC=0 NO NUMBER ENTERED         *
00394                          *         CC=1 ADDRH/ADDRL HAS NUMBER    *
00395                          *         A=DELIMITER                    *
00396                          * A,X VOLATILE                           *
00397                          * WORK1 IN USE
00398                          ****************************************
00399A 19A9 CD 19E8        A GETADR JSR    PUTSP
00400A 19AC 3F 44          A        CLR    ADDRH     CLEAR HIGH BYTE
00401A 19AE AD DE       198E        BSR    GETNYB    OBTAIN FIRST HEX VALUE
00402A 19B0 25 06       19B8        BCS    GETGTD    BRANCH IF GOT IT
00403A 19B2 A1 20          A        CMP    #'        ? SPACE
00404A 19B4 27 F3       19A9        BEQ    GETADR    LOOP IF SO
00405A 19B6 98                      CLC              RETURN NO NUMBER
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00406A 19B7 81                    RTS                 RETURN
00407A 19B8 B7 45        A GETGTD STA     ADDRL       INITIALIZE LOW VALUE
00408A 19BA AD D2     198E GETALP BSR     GETNYB      OBTAIN NEXT HEX
00409A 19BC 24 10     19CE        BCC     GETARG      BRANCH IF NONE
00410A 19BE 48                    ASLA                OVER
00411A 19BF 48                    ASLA                FOUR
00412A 19C0 48                    ASLA                BITS
00413A 19C1 48                    ASLA                FOR SHIFT
00414A 19C2 AE 04        A        LDX     #4          LOOP FOUR TIMES
00415A 19C4 48              GETASF ASLA               SHIFT NEXT BIT
00416A 19C5 39 45        A        ROL     ADDRL       INTO LOW BYTE
00417A 19C7 39 44        A        ROL     ADDRH       INTO HIGH BYTE
00418A 19C9 5A                    DECX                COUNT DOWN
00419A 19CA 26 F8     19C4        BNE     GETASF      LOOP UNTIL DONE
00420A 19CC 20 EC     19BA        BRA     GETALP      NOW DO NEXT HEX
00421A 19CE 99              GETARG SEC                SHOW NUMBER OBTAINED
00422A 19CF 81                    RTS                 RETURN TO CALLER


00424                       ******************************************
00425                       * CHRIN  -  OBTAIN NEXT INPUT CHARACTER  *
00426                       * OUTPUT: A=CHARACTER RECIEVED           *
00427                       * X IS TRANSPARENT                       *
00428                       * NULLS AND RUBOUTS IGNORED              *
00429                       * ALL CHARACTERS ECHOED OUT              *
00430                       * WORK1 USED                             *
00431                       ******************************************
00432A 19D0 C6 17F8      A CHRIN  LDA     ACIA        LOAD STATUS REGISTER
00433A 19D3 44                    LSRA                CHECK FOR INPUT
00434A 19D4 24 FA     19D0        BCC     CHRIN       LOOP UNTIL SOME
00435A 19D6 C6 17F9      A        LDA     ACIA+1      LOAD CHARACTER
00436A 19D9 A4 7F        A        AND     #$7F        AND OFF PARITY
00437A 19DB 27 F3     19D0        BEQ     CHRIN       IGNORE NULLS
00438A 19DD A1 7F        A        CMP     #$7F        ? DEL
00439A 19DF 27 EF     19D0        BEQ     CHRIN       IGNORE DELETES
00440A 19E1 B7 42        A        STA     WORK1       SAVE CHARACTER
00441A 19E3 AD 05     19EA        BSR     CHROUT      ECHO CHARACTER
00442A 19E5 B6 42        A        LDA     WORK1       RESTORE CHARACTER
00443A 19E7 81                    RTS                 RETURN TO CALLER


00445                       ***************************************************
00446                       *      PUTS --- PRINT A BLANK (SPACE)             *
00447                       *  X UNCHANGED                                    *
00448                       ***************************************************
00449A 19E8 A6 20        A PUTSP  LDA     #SP         LOAD SPACE
00450                       * FALL INTO CHROUT


00452                       ******************************************
00453                       * CHROUT  -  SEND CHARACTER TO TERMINAL. *
00454                       *            A CARRIAGE RETURN HAS AN    *
00455                       *            ADDED LINE FEED.            *
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00456                          * INPUT: A=ASCII CHARACTER TO SEND        *
00457                          * A NOT TRANSPARENT                        *
00458                          ******************************************
00459A 19EA A1 0D          A CHROUT CMP    #CR       ? CARRIAGE RETURN
00460A 19EC 26 04       19F2        BNE    CHROU2    BRANCH NOT
00461A 19EE AD 02       19F2        BSR    CHROU2    RECURSIVE CALL FOR CR
00462A 19F0 A6 0A          A        LDA    #LF       NOW SEND LINE FEED
00463A 19F2 C7 17F9        A CHROU2 STA    ACIA+1    STORE CHARACTER INTO PIC
00464A 19F5 C6 17F8        A CHROLP LDA    ACIA      LOAD STATUS REGISTER
00465A 19F8 A5 02          A        BIT    #$02      ? READY FOR NEXT
00466A 19FA 27 F9       19F5        BEQ    CHROLP    LOOP UNTIL READY
00467A 19FC 81                      RTS              AND RETURN


00469                          ******************************************
00470                          *      RESET --- POWER ON RESET ROUTINE
00471                          *
00472                          *  INITIALIZE ACIA, PUT OUT STARTUP MESSAGE
00473                          ******************************************
00474A 19FD AE 0B          A RESET  LDX    #11       MOVE VECTOR TABLE
00475A 19FF D6 1832        A RST    LDA    VECTAB,X  TO RAM USING A
00476A 1A02 E7 4F          A        STA    VECRAM,X  BLOCK MOVE ROUTINE
00477A 1A04 5A                      DECX             TO ALLOW CHANGES
00478A 1A05 2A F8       19FF        BPL    RST       ON THE FLY
00479A 1A07 A6 03          A        LDA    #3        RESET ACIA
00480A 1A09 C7 17F8        A        STA    ACIA      TO INITIALIZE
00481A 1A0C A6 51          A        LDA    #$51      8 BITS-NO PARITY-2 STOP BITS
00482A 1A0E C7 17F8        A        STA    ACIA      SETUP ACIA PARAMETERS
00483A 1A11 CD 1CC3        A        JSR    SCNBKP    CLEAR BREAKPOINTS
00484A 1A14 A6 FF          A        LDA    #$FF      TURN HIGH BIT ON
00485A 1A16 F7                REBCLR STA    ,X        SHOW SLOT EMPTY
00486A 1A17 5C                      INCX             TO
00487A 1A18 5C                      INCX             NEXT
00488A 1A19 5C                      INCX             SLOT
00489A 1A1A 3A 4B          A        DEC    PNCNT     COUNT DOWN
00490A 1A1C 26 F8       1A16        BNE    REBCLR    CLEAR NEXT
00491A 1A1E 3F 41          A RESREN CLR    SWIFLG    SETUP MONITOR ENTRANCE VALUE
00492A 1A20 83                      SWI              ENTER MONITOR
00493A 1A21 20 FB       1A1E        BRA    RESREN    REENTER IF "G"


00495                          ******************************************
00496                          * COMMAND HANDLER                          *
00497                          ******************************************
00498A 1A23 CD 1981        A CMDPDT JSR    PDATA     SEND MESSAGE OUT

00500A 1A26 CD 197F        A CMD    JSR    PCRLF     TO NEW LINE
00501A 1A29 A6 3E          A CMDNNL LDA    #PROMPT   READY PROMPT CHR
00502A 1A2B AD BD       19EA        BSR    CHROUT    SEND IT OUT
00503A 1A2D CD 1CA7        A        JSR    REMBKP    REMOVE BREAKPOINTS IF IN
00504A 1A30 AD 9E       19D0        BSR    CHRIN     GET NEXT CHARACTER
00505A 1A32 5F                      CLRX             ZERO FOR SOME COMMANDS
00506A 1A33 A1 43          A        CMPA   #'C       ? DISPLAY/CHANGE C REGISTER
00507A 1A35 27 49       1A80        BEQ    CMDC      BRANCH IF SO
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00508A 1A37 A1 58        A          CMPA   #'X       ? DISPLAY/CHANGE X REGISTER
00509A 1A39 27 43     1A7E          BEQ    CMDX      BRANCH IF SO
00510A 1A3B A1 41        A          CMPA   #'A       ? DISPLAY/CHANGE A REGISTER
00511A 1A3D 27 40     1A7F          BEQ    CMDA      BRANCH IF SO
00512A 1A3F A1 52        A          CMP    #'R       ? REGISTER DISPLAY
00513A 1A41 27 35     1A78          BEQ    REGR      BRANCH YES
00514A 1A43 A1 4C        A          CMP    #'L       ? LOAD FILE
00515A 1A45 26 03     1A4A          BNE    NOTL      NOPE
00516A 1A47 CC 18A1      A          JMP    CMDL      BRANCH YES
00517A 1A4A A1 47        A NOTL     CMPA   #'G       ? GO COMMAND
00518A 1A4C 26 05     1A53          BNE    NOTG      BRANCH NOT
00519A 1A4E 1E 47        A          BSET   7,WORK4
00520A 1A50 CC 183E      A ISP      JMP    CMDG      GO TO IT
00521A 1A53 A1 4D        A NOTG     CMP    #'M       ? MEMORY COMMAND
00522A 1A55 26 03     1A5A          BNE    NOTM      BRANCH NOT
00523A 1A57 CC 18E7      A          JMP    CMDM      GO TO MEMORY DISPLAY/CHANGE
00524A 1A5A A1 54        A NOTM     CMP    #'T       ? TRACE
00525A 1A5C 26 03     1A61          BNE    NOTT      ERROR IF NOT
00526A 1A5E CC 1C23      A          JMP    CMDT      GO TO IT
00527A 1A61 A1 57        A NOTT     CMP    #'W       ? WRITE MEMORY
00528A 1A63 26 03     1A68          BNE    NOTW      BRANCH NO
00529A 1A65 CC 1B97      A          JMP    CMDW      GO TO IT
00530A 1A68 A1 42        A NOTW     CMP    #'B       ? BREAKPOINT COMMAND
00531A 1A6A 27 0F     1A7B          BEQ    BPNT      YES
00532A 1A6C A1 50        A          CMP    #'P       ? PC COMMAND
00533A 1A6E 26 04     1A74          BNE    CMDERR
00534A 1A70 1F 47        A          BCLR   7,WORK4
00535A 1A72 20 DC     1A50          BRA    ISP
00536A 1A74 AE 0D        A CMDERR   LDX    #MSGERR-MBASE LOAD ERROR STRING
00537A 1A76 20 AB     1A23 TOCPDT   BRA    CMDPDT    AND SEND IT OUT
00538A 1A78 CC 1AEE      A REGR     JMP    CMDR
00539A 1A7B CC 1B35      A BPNT     JMP    CMDB


00541                          ***************************************************
00542                          *  X -- DISPLAY/CHANGE X REGISTER                 *
00543                          ***************************************************
00544A 1A7E 5C                 CMDX   INCX              INCREMENT INDEX
00545                          * FALL THROUGH


00547                          ***************************************************
00548                          *  A -- DISPLAY/CHANGE A REGISTER
00549                          ***************************************************
00550A 1A7F 5C                 CMDA   INCX              INCREMENT INDEX
00551                          * FALL THROUGH


00553                          ***************************************************
00554                          *  C -- DISPLAY/CHANGE CONDITION CODE REGISTER    *
00555                          ***************************************************
00556A 1A80 CD 19E8         A CMDC   JSR    PUTSP     SPACE BEFORE VALUE
00557A 1A83 BF 42           A        STX    WORK1     SAVE INDEX VALUE
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00558A 1A85 CD 1B23      A        JSR    LOCSTK   LOCATE STACK ADDRESS
00559A 1A88 9F                    TXA             STACK-2 TO A
00560A 1A89 BB 42        A        ADD    WORK1    ADD PROPER OFFSET
00561A 1A8B AB 04        A        ADD    #4       MAKE UP FOR ADDRESS RETURN DIFFERE
00562A 1A8D 3F 44        A        CLR    ADDRH    SETUP ZERO HIGH BYTE
00563A 1A8F B7 45        A        STA    ADDRL    AND SET IN LOW
00564A 1A91 CC 18F8      A TOMCHG JMP    MCHNGE   NOW ENTER MEMORY CHANGE COMMAND


00566                       ******************************************
00567                       *            S W I  HANDLER              *
00568                       * DETERMINE PROCESSING SWIFLG VALUE      *
00569                       ******************************************
00570A 1A94 5F              SWI    CLRX            DEFAULT TO STARTUP MESSAGE
00571A 1A95 3D 41        A        TST    SWIFLG   IS THIS RESET
00572A 1A97 26 04     1A9D        BNE    SWICHK   IF NOT REMOVE BREAKPOINTS
00573A 1A99 3C 41        A        INC    SWIFLG   SHOW WE ARE NOW INITIALIZED
00574A 1A9B 20 86     1A23        BRA    CMDPDT   TO COMMAND HANDLER
00575A 1A9D CD 1CC3      A SWICHK JSR    SCNBKP
00576A 1AA0 F6              SWIREP LDA    ,X       RESTORE OPCODES
00577A 1AA1 2B 0B     1AAE        BMI    SWINOB
00578A 1AA3 B7 44        A        STA    ADDRH
00579A 1AA5 E6 01        A        LDA    1,X
00580A 1AA7 B7 45        A        STA    ADDRL
00581A 1AA9 E6 02        A        LDA    2,X
00582A 1AAB CD 1952      A        JSR    STORE
00583A 1AAE 5C              SWINOB INCX
00584A 1AAF 5C                     INCX
00585A 1AB0 5C                     INCX
00586A 1AB1 3A 4B        A        DEC    PNCNT
00587A 1AB3 26 EB     1AA0        BNE    SWIREP
00588                       * TRACE ONE INSTRUCTION IF PC AT A BREAKPOINT
00589A 1AB5 CD 1B23      A        JSR    LOCSTK   FIND STACK
00590A 1AB8 E6 08        A        LDA    8,X      GET PC AND ADJUST
00591A 1ABA A0 01        A        SUB    #1
00592A 1ABC B7 47        A        STA    WORK4    SAVE PCL
00593A 1ABE E6 07        A        LDA    7,X
00594A 1AC0 A2 00        A        SBC    #0
00595A 1AC2 B7 46        A        STA    WORK3    SAVE PCH
00596A 1AC4 BF 48        A        STX    WORK5    SAVE SP
00597A 1AC6 CD 1CC3      A        JSR    SCNBKP
00598A 1AC9 F6              SWITRY LDA    0,X
00599A 1ACA 2B 1B     1AE7        BMI    SWICMP
00600A 1ACC B1 46        A        CMP    WORK3
00601A 1ACE 26 17     1AE7        BNE    SWICMP
00602A 1AD0 E6 01        A        LDA    1,X
00603A 1AD2 B1 47        A        CMP    WORK4
00604A 1AD4 26 11     1AE7        BNE    SWICMP
00605A 1AD6 BE 48        A        LDX    WORK5
00606A 1AD8 E7 08        A        STA    8,X
00607A 1ADA B6 46        A        LDA    WORK3
00608A 1ADC E7 07        A        STA    7,X
00609A 1ADE 3F 4A        A        CLR    WORK7
00610A 1AE0 A6 01        A        LDA    #1
00611A 1AE2 B7 49        A        STA    WORK6
00612A 1AE4 CC 1C32      A        JMP    TRACE
00613A 1AE7 5C              SWICMP INCX
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00614A 1AE8 5C                    INCX
00615A 1AE9 5C                    INCX
00616A 1AEA 3A 4B        A        DEC    PNCNT
00617A 1AEC 26 DB     1AC9        BNE    SWITRY
00618                            * FALL INTO REGISTER DISPLAY FOR BREAKPOINT


00620                            ***********************************************
00621                            * R -- PRINT REGISTERS                       *
00622                            ***********************************************
00623A 1AEE CD 19E8      A CMDR   JSR    PUTSP    SPACE BEFORE DISPLAY
00624A 1AF1 AD 30     1B23        BSR    LOCSTK   LOCATE STACK-4
00625A 1AF3 E6 07        A        LDA    7,X      OFFSET FOR PC HIGH
00626A 1AF5 E7 07        A        STA    7,X      RESTORE INTO STACK
00627A 1AF7 CD 1969      A        JSR    PUTBYT   PLACE BYTE OUT
00628A 1AFA E6 08        A        LDA    8,X      OFFSET TO PC LOW
00629A 1AFC AD 1F     1B1D        BSR    CRBYTS   TO HEX AND SPACE
00630A 1AFE E6 05        A        LDA    5,X      NOW TO A REGISTER
00631A 1B00 AD 1B     1B1D        BSR    CRBYTS   TO HEX AND SPACE
00632A 1B02 E6 06        A        LDA    6,X      NOW X
00633A 1B04 AD 17     1B1D        BSR    CRBYTS   HEX AND SPACE
00634A 1B06 E6 04        A        LDA    4,X      NOW CONDITION CODE
00635A 1B08 AA E0        A        ORA    #$E0     SET ON UNUSED BITS
00636A 1B0A E7 04        A        STA    4,X      RESTORE
00637A 1B0C AD 0F     1B1D        BSR    CRBYTS   HEX AND SPACE
00638A 1B0E 9F                    TXA             STACK POINTER-3
00639A 1B0F AB 08        A        ADD    #8       TO USERS STACK POINTER
00640A 1B11 AD 0A     1B1D        BSR    CRBYTS   TO HEX AND SPACE
00641A 1B13 CC 1A26      A GTOCMD JMP    CMD      BACK TO COMMAND HANDLER
00642                            * PRINT ADDRESS SUBROUTINE (X UNCHANGED)
00643A 1B16 B6 44        A PRTADR LDA    ADDRH    LOAD HIGH BYTE
00644A 1B18 CD 1969      A        JSR    PUTBYT   SEND OUT AS HEX
00645A 1B1B B6 45        A        LDA    ADDRL    LOAD LOW BYTE
00646A 1B1D CD 1969      A CRBYTS JSR    PUTBYT   PUT OUT IN HEX
00647A 1B20 CC 19E8      A        JMP    PUTSP    FOLLOW WITH A SPACE


00649                            ***********************************************
00650                            *  LOCSTK - LOCATE CALLERS STACK POINTER      *
00651                            * RETURNS X=STACK POINTER-3                   *
00652                            * A VOLATILE                                  *
00653                            ***********************************************
00654A 1B23 AD 01     1B26 LOCSTK BSR    LOCST2   LEAVE ADDRESS ON STACK
00655          001B      A STKHI  EQU    */256   HI BYTE ON STACK
00656          0025      A STKLOW EQU    *-(*/256)*256 LOW BYTE ON STACK
00657A 1B25 81                    RTS             RETURN WITH RESULT
00658A 1B26 AE 7F        A LOCST2 LDX    #$7F     LOAD HIGH STACK WORD ADDRESS
00659A 1B28 A6 1B        A LOCLOP LDA    #STKHI   HIGH BYTE FOR COMPARE
00660A 1B2A 5A                    LOCDWN DECX     TO NEXT LOWER BYTE IN STACK
00661A 1B2B F1                    CMP    ,X       ? THIS THE SAME
00662A 1B2C 26 FC     1B2A        BNE    LOCDWN   IF NOT TRY NEXT LOWER
00663A 1B2E A6 25        A        LDA    #STKLOW  COMPARE WITH LOW ADDRESS BYTE
00664A 1B30 E1 01        A        CMP    1,X      ? FOUND RETURN ADDRESS
00665A 1B32 26 F4     1B28        BNE    LOCLOP   LOOP IF NOT
00666A 1B34 81                    RTS             RETURN WITH X SET
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00668                          ***********************************************
00669                          *  B -- BREAKPOINT CLEAR, SET, OR DISPLAY     *
00670                          ***********************************************
00671A 1B35 CD 19D0      A CMDB   JSR    CHRIN     READ NEXT CHARACTER
00672A 1B38 A1 20        A        CMP    #'        ? DISPLAY ONLY
00673A 1B3A 26 38     1B74        BNE    BDSPLY    BRANCH IF SO
00674A 1B3C AD 4F     1B8D        BSR    PGTADR    OBTAIN BREAKPOINT NUMBER
00675A 1B3E 5D                    TSTX             ? ANY HIGH BYTE VALUE
00676A 1B3F 26 49     1B8A        BNE    BKERR     ERROR IF SO
00677A 1B41 4A                    DECA             DOWN COUNT BY ONE
00678A 1B42 A1 03        A        CMP    #NUMBKP   ? TO HIGH
00679A 1B44 24 44     1B8A        BHS    BKERR     ERROR IF SO
00680A 1B46 48                    ASLA             TIMES TWO
00681A 1B47 BB 45        A        ADD    ADDRL     PLUS ONE FOR THREE TIMES
00682A 1B49 AB 38        A        ADD    #BKPTBL   FIND TABLE ADDRESS
00683A 1B4B 4A                    DECA
00684A 1B4C B7 43        A        STA    WORK2     SAVE ADDRESS
00685A 1B4E AD 3D     1B8D        BSR    PGTADR    OBTAIN ADDRESS
00686A 1B50 A3 20        A        CPX    #$20
00687A 1B52 24 36     1B8A        BHS    BKERR
00688A 1B54 BE 43        A        LDX    WORK2     RELOAD ENTRY POINTER
00689A 1B56 E7 01        A        STA    1,X       SAVE LOW ADDRESS
00690A 1B58 26 08     1B62        BNE    BKNOCL    BRANCH IF NOT ZERO
00691A 1B5A B6 44        A        LDA    ADDRH     LOAD HIGH ADDRESS
00692A 1B5C 26 06     1B64        BNE    BKNCR     BRANCH NOT NULL
00693A 1B5E 4A                    DECA             CREATE NEGATIVE VALUE
00694A 1B5F F7                    STA    ,X        STORE AS HIGH BYTE
00695A 1B60 20 B1     1B13        BRA    GTOCMD    END COMMAND
00696A 1B62 B6 44        A BKNOCL LDA    ADDRH     LOAD HIGH ADDRESS
00697A 1B64 F7              BKNCR STA    ,X        STORE HIGH BYTE
00698A 1B65 CD 1943      A        JSR    LOAD      LOAD BYTE AT THE ADDRESS
00699A 1B68 43                    COMA             INVERT IT
00700A 1B69 CD 1952      A        JSR    STORE     ATTEMPT STORE
00701A 1B6C 25 1C     1B8A        BCS    BKERR     ERROR IF DID NOT STORE
00702A 1B6E 43                    COMA             RESTORE PROPER VALUE
00703A 1B6F CD 1952      A        JSR    STORE     STORE IT BACK
00704A 1B72 20 9F     1B13        BRA    GTOCMD    END COMMAND

00706                          * DISPLAY BREAKPOINTS
00707A 1B74 CD 1CC3      A BDSPLY JSR    SCNBKP    PREPARE SCAN OF TABLE
00708A 1B77 F6              BDSPLP LDA   ,X        OBTAIN HIGH BYTE
00709A 1B78 2B 07     1B81        BMI    BDSKP     SKIP IF UNUSED SLOT
00710A 1B7A CD 1969      A        JSR    PUTBYT    PRINT OUT HIGHT BYTE
00711A 1B7D E6 01        A        LDA    1,X       LOAD LOW BYTE
00712A 1B7F AD 9C     1B1D        BSR    CRBYTS    PRINT IT OUT WITH A SPACE
00713A 1B81 5C              BDSKP INCX             TO
00714A 1B82 5C                    INCX             NEXT
00715A 1B83 5C                    INCX             ENTRY
00716A 1B84 3A 4B        A        DEC    PNCNT     COUNT DOWN
00717A 1B86 26 EF     1B77        BNE    BDSPLP    LOOP IF MORE
00718A 1B88 20 89     1B13        BRA    GTOCMD    END COMMAND

00720A 1B8A CC 1A74      A BKERR  JMP    CMDERR    GIVE ERROR RESPONSE


00722                          ***********************************************
00723                          *  W -- WRITE MEMORY TO TAPE FILE S1/S9       *
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00724                          ***********************************************
00725A 1B8D CD 19A9      A PGTADR JSR     GETADR    OBTAIN INPUT ADDRESS
00726A 1B90 24 F8     1B8A        BCC     BKERR     ABORT IF NONE
00727A 1B92 BE 44        A        LDX     ADDRH     READY HIGH BYTE
00728A 1B94 B6 45        A        LDA     ADDRL     READY LOW BYTE
00729A 1B96 81                    RTS               BACK TO PUNCH COMMAND

00731A 1B97 AD F4     1B8D CMDW   BSR     PGTADR    GET STARTING ADDRESS
00732A 1B99 A3 20        A        CPX     #$20
00733A 1B9B 24 ED     1B8A        BHS     BKERR
00734A 1B9D B7 47        A        STA     WORK4     INTO WORK4
00735A 1B9F BF 46        A        STX     WORK3     AND WORK3
00736A 1BA1 AD EA     1B8D        BSR     PGTADR    GET ENDING ADDRESS
00737A 1BA3 A3 20        A        CPX     #$20
00738A 1BA5 24 E3     1B8A        BHS     BKERR
00739A 1BA7 4C                    INCA              ADD ONE TO INCLUDE TOP BYTE
00740A 1BA8 26 01     1BAB        BNE     PUPH      BRANCH NO CARRY
00741A 1BAA 5C                    INCX              UP HIGH BYTE AS WELL
00742A 1BAB B0 47        A PUPH   SUB     WORK4     COMPUTE SIZE
00743A 1BAD B7 4D        A        STA     PNRCNT+1  AND SAVE
00744A 1BAF 9F                    TXA               NOW
00745A 1BB0 B2 46        A        SBC     WORK3     SIZE HIGH BYTE
00746A 1BB2 B7 4C        A        STA     PNRCNT    AND SAVE
00747A 1BB4 B6 47        A        LDA     WORK4     MOVE
00748A 1BB6 B7 45        A        STA     ADDRL     TO
00749A 1BB8 B6 46        A        LDA     WORK3     MEMORY
00750A 1BBA B7 44        A        STA     ADDRH     POINTER
00751                       * ADDR->MEMORY START,  PNRCNT=BYTE COUNT OF AREA
00752                       * NOW TURN ON THE PUNCH
00753A 1BBC A6 12        A        LDA     #DC2      PUNCH ON CONTROL
00754A 1BBE CD 19EA      A        JSR     CHROUT    SEND OUT
00755                       * NOW SEND CR FOLLOWED BY 24 NULLS AND 'S1'
00756A 1BC1 AD 53     1C16 PREC   BSR     PNCRNL    SEND CR/LF AND NULLS
00757A 1BC3 AE 17        A        LDX     #MSGS1-MBASE POINT TO STRING
00758A 1BC5 CD 1985      A        JSR     PDATA1    SEND 'S1' OUT
00759                       * NOW SEND NEXT 24 BYTES OR LESS IF TO THE END
00760A 1BC8 B6 4D        A        LDA     PNRCNT+1  LOW COUNT LEFT
00761A 1BCA A0 18        A        SUB     #24       MINUS 24
00762A 1BCC B7 4D        A        STA     PNRCNT+1  STORE RESULT
00763A 1BCE 24 08     1BD8        BCC     PALL24    IF NO CARRY THEN OK
00764A 1BD0 3A 4C        A        DEC     PNRCNT    DOWN HIGH BYTE
00765A 1BD2 2A 04     1BD8        BPL     PALL24    ALL 24 OK
00766A 1BD4 AB 18        A        ADD     #24       WAS LESS SO BACK UP TO ORIGINAL
00767A 1BD6 20 02     1BDA        BRA     PGOTC     GO USE COUNT HERE
00768A 1BD8 A6 18        A PALL24 LDA     #24       USE ALL 24
00769A 1BDA B7 4B        A PGOTC  STA     PNCNT     COUNT FOR THIS RECORD
00770                       * SEND THE FRAME COUNT AND START CHECKSUMMING
00771A 1BDC 3F 4E        A        CLR     CHKSUM
00772A 1BDE AB 03        A        ADD     #3        ADJUST FOR COUNT AND ADDRESS
00773A 1BE0 AD 2B     1C0D        BSR     PUNBYT    SEND FRAME COUNT
00774                       * SEND ADDRESS
00775A 1BE2 B6 44        A        LDA     ADDRH     HI BYTE
00776A 1BE4 AD 27     1C0D        BSR     PUNBYT    SEND IT
00777A 1BE6 B6 45        A        LDA     ADDRL     LOW BYTE
00778A 1BE8 AD 23     1C0D        BSR     PUNBYT    SEND IT
00779                       * NOW SEND DATA
00780A 1BEA CD 1943      A PUNLOP JSR     LOAD      LOAD NEXT BYTE
00781A 1BED AD 1E     1C0D        BSR     PUNBYT    SEND IT OUT
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00782A 1BEF CD 1962      A          JSR     PTRUP1    UP ADDRESS BY ONE
00783A 1BF2 3A 4B        A          DEC     PNCNT     COUNT DOWN
00784A 1BF4 26 F4     1BEA          BNE     PUNLOP    LOOP UNTIL ZERO
00785                            * SEND OUT THE CHECKSUM
00786A 1BF6 B6 4E        A          LDA     CHKSUM    LOAD CHECKSUM
00787A 1BF8 43                      COMA              COMPLEMENT IT
00788A 1BF9 AD 12     1C0D          BSR     PUNBYT    SEND IT OUT
00789                            * LOOP OR SEND S9
00790A 1BFB B6 4C        A          LDA     PNRCNT    ? MINUS
00791A 1BFD 2B 04     1C03          BMI     PNEND     YES QUIT
00792A 1BFF BB 4D        A          ADD     PNRCNT+1  ? ZERO
00793A 1C01 26 BE     1BC1          BNE     PREC      NO, DO NEXT RECORD
00794A 1C03 AD 11     1C16 PNEND    BSR     PNCRNL    SEND CR AND NULLS
00795A 1C05 AE 1A        A          LDX     #MSGS9-MBASE LOAD S9 TEXT
00796A 1C07 CD 1985      A          JSR     PDATA1    SEND AND TO COMMAND HANDLER
00797A 1C0A CC 1A26      A          JMP     CMD       TO COMMAND HANDLER

00799                            * SUB TO SEND BYTE IN HEX AND ADJUST CHECKSUM
00800A 1C0D 97                   PUNBYT TAX            SAVE BYTE
00801A 1C0E BB 4E        A          ADD     CHKSUM    ADD TO CHECKSUM
00802A 1C10 B7 4E        A          STA     CHKSUM    STORE BACK
00803A 1C12 9F                      TXA               RESTORE BYTE
00804A 1C13 CC 1969      A          JMP     PUTBYT    SEND OUT IN HEX

00806                            * SUB TO SEND CR/LF AND 24 NULLS
00807A 1C16 CD 197F      A PNCRNL JSR     PCRLF     SEND CR/LF
00808A 1C19 AE 18        A          LDX     #24       COUNT NULLS
00809A 1C1B 4F                   PNULLS CLRA          CREATE NULL
00810A 1C1C CD 19EA      A          JSR     CHROUT    SEND OUT
00811A 1C1F 5A                      DECX              COUNT DOWN
00812A 1C20 26 F9     1C1B          BNE     PNULLS    LOOP UNTIL DONE
00813A 1C22 81                      RTS               RETURN


00815                            ********************************************
00816                            *  T -- TRACE COMMAND                      *
00817                            ********************************************
00818A 1C23 A6 01        A CMDT   LDA     #1        DEFAULT COUNT
00819A 1C25 B7 45        A        STA     ADDRL     TO ONE *GETADR CLEARS ADDRH*
00820A 1C27 CD 19A9      A        JSR     GETADR    BUILD ADDRESS IF ANY
00821A 1C2A B6 44        A        LDA     ADDRH     SAVE VALUE IN TEMPORARY
00822A 1C2C B7 4A        A        STA     WORK7     LOCATIONS FOR LATER
00823A 1C2E B6 45        A        LDA     ADDRL     USE
00824A 1C30 B7 49        A        STA     WORK6
00825                            * SETUP TIMER TO TRIGGER INTERRUPT
00826          1C32      A TRACE  EQU     *
00827A 1C32 CD 1B23      A        JSR     LOCSTK
00828A 1C35 E6 04        A        LDA     4,X       GET CURRENT USER I-MASK
00829A 1C37 A4 08        A        AND     #8
00830A 1C39 B7 48        A        STA     WORK5     SAVE IT
00831A 1C3B E6 07        A        LDA     7,X       GET CURRENT USER PC
00832A 1C3D B7 44        A        STA     ADDRH
00833A 1C3F E6 08        A        LDA     8,X
00834A 1C41 B7 45        A        STA     ADDRL
00835A 1C43 CD 1943      A        JSR     LOAD      GET OPCODE
00836A 1C46 A1 83        A        CMP     #$83      SWI?
00837A 1C48 26 0E     1C58        BNE     TRACE3    IF YES THEN
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00838A 1C4A B6 45        A         LDA    ADDRL     INC USER PC
00839A 1C4C AB 01        A         ADD    #1
00840A 1C4E E7 08        A         STA    8,X
00841A 1C50 B6 44        A         LDA    ADDRH
00842A 1C52 A9 00        A         ADC    #0
00843A 1C54 E7 07        A         STA    7,X
00844A 1C56 20 2D     1C85         BRA    TIRQ      CONTINUE TO TRACE
00845A 1C58 A1 9B        A TRACE3  CMP    #$9B      SEI?
00846A 1C5A 26 14     1C70         BNE    TRACE2    IF YES
00847A 1C5C E6 04        A         LDA    4,X       THEN SET IT IN THE STACK
00848A 1C5E AA 08        A         ORA    #8
00849A 1C60 E7 04        A         STA    4,X
00850A 1C62 B6 45        A         LDA    ADDRL     THEN INC USER PC
00851A 1C64 AB 01        A         ADD    #1
00852A 1C66 E7 08        A         STA    8,X
00853A 1C68 B6 44        A         LDA    ADDRH
00854A 1C6A A9 00        A         ADC    #0
00855A 1C6C E7 07        A         STA    7,X
00856A 1C6E 20 15     1C85         BRA    TIRQ      CONTINUE TO TRACE
00857A 1C70 A1 9A        A TRACE2  CMP    #$9A      CLI?
00858A 1C72 26 02     1C76         BNE    TRACE1    IF YES THEN
00859A 1C74 3F 48        A         CLR    WORK5
00860A 1C76 E6 04        A TRACE1  LDA    4,X
00861A 1C78 A4 F7        A         AND    #$F7
00862A 1C7A E7 04        A         STA    4,X
00863A 1C7C A6 10        A         LDA    #16       THEN SET UP TIMER
00864A 1C7E B7 08        A         STA    TIMER
00865A 1C80 A6 08        A         LDA    #8
00866A 1C82 B7 09        A         STA    TIMEC
00867A 1C84 80                     RTI              EXECUTE ONE INSTRUCTION


00869                   ********************************************
00870                   *  TIRQ -- TIMER INTERRUPT ROUTINE         *
00871                   ********************************************
00872          1C85     A TIRQ    EQU    *
00873                   *  RESTORE I-MASK TO PROPER STATE
00874A 1C85 A6 40        A         LDA    #$40
00875A 1C87 B7 09        A         STA    TIMEC
00876A 1C89 CD 1B23      A         JSR    LOCSTK
00877A 1C8C E6 04        A         LDA    4,X
00878A 1C8E BA 48        A         ORA    WORK5
00879A 1C90 E7 04        A         STA    4,X
00880                   *  SEE IF MORE TRACING IS DESIRED
00881A 1C92 3A 49        A         DEC    WORK6
00882A 1C94 26 9C     1C32         BNE    TRACE
00883A 1C96 3D 4A        A         TST    WORK7
00884A 1C98 27 04     1C9E         BEQ    DISREG
00885A 1C9A 3A 4A        A         DEC    WORK7
00886A 1C9C 20 94     1C32         BRA    TRACE
00887A 1C9E CC 1AEE      A DISREG  JMP    CMDR


00889                   ********************************************
00890                   *  INT -- INTERRUPT ROUTINE                *
00891                   ********************************************
```

Figure 3-17. ASSIST05 Program Listing (Continued)

```
00892            1CA1      A IRQ     EQU     *
00893A 1CA1 CC 1A74        A         JMP     CMDERR    HARDWARE INTERRUPT UNUSED


00895                        ***********************************************
00896                        * TWIRQ - TIMER INTERRUPT ROUTINE (WAIT MODE)*
00897                        ***********************************************
00898            1CA4      A TWIRQ   EQU     *
00899A 1CA4 CC 1A74        A         JMP     CMDERR    TIMER WAIT INTERRUPT UNUSED


00901                        ***********************************************
00902                        *  DELBKP - DELETE BREAKPOINT SUBROUTINE      *
00903                        ***********************************************
00904A 1CA7 AD 1A     1CC3 REMBKP BSR     SCNBKP    SETUP PARAMETERS
00905A 1CA9 2A 17     1CC2        BPL     REMRTS    RETURN IF NOT IN
00906A 1CAB F6               REMLOP LDA     ,X        LOAD HIGH ADDRESS
00907A 1CAC 2B 0B     1CB9        BMI     REMNOB    SKIP IF NULL
00908A 1CAE B7 44          A     STA     ADDRH     STORE HIGH ADDRESS
00909A 1CB0 E6 01          A     LDA     1,X       LOAD LOW ADDRESS
00910A 1CB2 B7 45          A     STA     ADDRL     STORE IT
00911A 1CB4 E6 02          A     LDA     2,X       LOAD OPCODE
00912A 1CB6 CD 1952        A     JSR     STORE     STORE IT BACK OVER 'SWI'
00913A 1CB9 5C               REMNOB INCX            TO
00914A 1CBA 5C                     INCX            NEXT
00915A 1CBB 5C                     INCX            ENTRY
00916A 1CBC 3A 4B          A     DEC     PNCNT     COUNT DOWN
00917A 1CBE 26 EB     1CAB        BNE     REMLOP    LOOP IF MORE
00918A 1CC0 33 41          A     COM     SWIFLG    MAKE POSITIVE TO SHOW REMOVED
00919A 1CC2 81               REMRTS RTS             RETURN

00921                        * SETUP FOR BREAKPOINT TABLE SCAN
00922A 1CC3 A6 03          A SCNBKP LDA     #NUMBKP   LOAD NUMBER OF BREAKPOINTS
00923A 1CC5 B7 4B          A     STA     PNCNT     SETUP FOR COUNTDOWN
00924A 1CC7 AE 38          A     LDX     #BKPTBL   LOAD TABLE ADDRESS
00925A 1CC9 3D 41          A     TST     SWIFLG    TEST IF BREAKPOINTS IN ALREADY
00926A 1CCB 81                   RTS               RETURN


00928                        ***********************************************
00929                        *       INTERRUPT VECTORS                     *
00930                        ***********************************************
00931A 1FF6                      ORG     MONSTR+$800-$A START OF VECTORS
00932A 1FF6      004F      A     FDB     VECRAM    TIMER INTERRUPT HANDLER (WAIT MODE
00933A 1FF8      0052      A     FDB     VECRAM+3  TIMER INTERRUPT HANDLER
00934A 1FFA      0055      A     FDB     VECRAM+6  INTERRUPT HANDLER
00935A 1FFC      0058      A     FDB     VECRAM+9  SWI HANDLER
00936A 1FFE      19FD      A     FDB     RESET     POWER ON VECTOR


00938                            END
```

Figure 3-17. ASSIST05 Program Listing (Concluded)