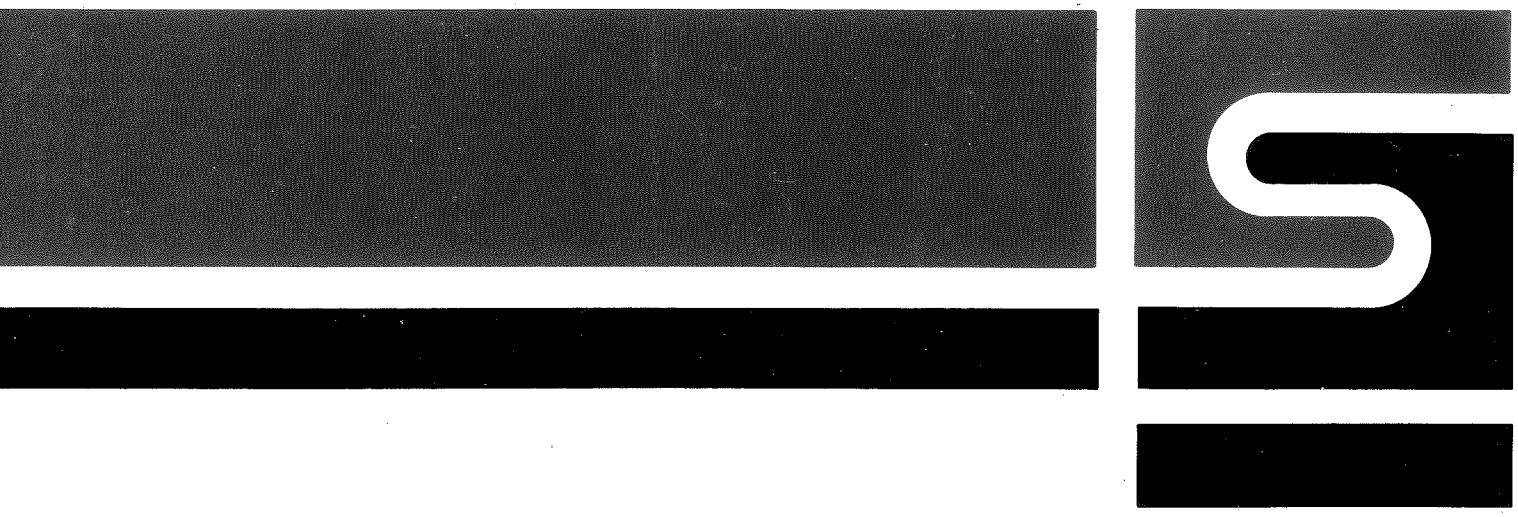


Synertek Systems

# SYM-1/69

## SYM-1 Supplement

# SYM-1/69 SUPPLEMENT



SSPub MAN-260087-A

First Printing: September 1982

Synertek
System Products

P.O. Box 552 • Santa Clara, Calif. 95052 • Telephone (408) 998-5689 • TWX: 910-338-0135

## TABLE OF CONTENTS



## LIST OF ILLUSTRATIONS

# CHAPTER 1

# INTRODUCTION

Congratulations on your purchase of the SYM-1/69! This package adds the power of the S6809 to the excellent input/output capabilities of the SYM-1. In addition, it provides all of the convenience and facilities of the SUPERMON monitor program available in your development of machine language programs for the S6809.

## 1.1 DESCRIPTION OF THE CONVERSION HARDWARE

The hardware is an adaptation to the S6809 which allows it to occupy a socket assembly originally designed for the S6509 microprocessor. It provides a relocation of the various address, data and control lines to match the appropriate locations on the board. Also included are a different microprocessor clock and a simulation of some of the signals required by the SYM-1 which normally are not provided by the S6809. The generation and usage of these signals is explained further in Chapter 9 of the SYM-1 Manual.

Follow the detailed installation instructions in the enclosed S6809 Instruction Set Summary card and you will have all of the power of the S6809 in your SYM-1 system.

## 1.2 DESCRIPTION OF THE REPLACEMENT SUPERMON

SUPERMON, the SYM-1 Monitor program, has been rewritten to provide the same facilities to the S6809 market as was provided for the SY6502 user in its original form. All of the functions this monitor program can perform are described in the SYM-1 Manual. Most of the commands function exactly as described. See the next section for more information.

## 1.3 THE MANUAL

This manual was written to supplement Chapter 9 (Advanced Monitor and Programming Techniques) of the SYM-1 Manual. Chapter 2 contains the same type of data for the S6809 as was provided in the original chapter for the SY6502.

The new chapter explains where the command or display sequences differ and provides a series of programming examples to get you started. In addition, it shows differences in the allocation of the system memory areas as described below.

### 1.3.1 The Command Sequences

Although the detailed SUPERMON keystroke combinations described in the SYM-1 Manual will perform the same function as the S6809, there will be some differences in the actual operation because the S6809 is a more complex microprocessor and has a different display sequence.

To understand the operational differences specified in Chapter 9, familiarize yourself with Chapter 5 of the SYM-1 Manual. It contains a description of the operating sequence of the original SY6502 SUPERMON and explains the basic function of each of the commands. All of these command sequences remain valid, except for addition of items displayed in the R or Register display command sequence. This is because the S6809 has more registers than the SY6502, thus having additional data to display during this command sequence.

### 1.3.2 Useful Addresses and Entry Points

As you develop more complex routines, it sometimes becomes convenient to avoid "reinventing the wheel" by incorporating previously debugged routines into your programs. This usually is done by means of a set of subroutine calls. Table 2-1 provides a list of the entry points for the various routines and affected registers so you can use them in your programs. The list corresponds to that shown for the SY6502 SUPERMON in the SYM-1 Manual as Table 9-1.

### 1.3.3 Interrupt Vectors and Memory Allocation

Data regarding system memory usage and the generation and use of the interrupt vectors can be found in Section 2.1.4. The organization of the original memory table (SYM-1 Manual, Figure 4-10) has been preserved. The notes given in Chapter 4 for this figure have not been reproduced here because they are still valid.

### 1.3.4 Other Advanced Programming Techniques

The remainder of this supplement is devoted to programming examples for the S6809, along with an explanation of the S6809 instruction set. To effectively use the S6809, obtain a copy of item 1 in the following references. It contains a description of programming techniques and an introduction to effective use of the S6809.

## 1.4 REFERENCES

1. S6809 ASSEMBLY LANGUAGE PROGRAMMING, Lance Levanthal, 1981. Osborne/McGraw-Hill Book Co.

2. SYM-1 Monitor Theory of Operations Manual (S6502 version, but can give insight into the construction and utilization of the monitor). Robert A. Peck, 1979. Available from SYM-1 Users Group, SYM-PHYSIS, Box 315, Chico, Ca. 95926.

# CHAPTER 2

# OPERATING THE SYM-1/69

This chapter is designed to instruct you to create programs for your S6809 version of the SYM-1. Also included is information about the SYM-S6809 features, instruction set and assembly language.

## 2.1 SYM-1/69 UNIQUE MONITOR IMPLEMENTATION

All SYM-1 monitor versions are by design almost identical. However, their unique registers, interrupts and instruction sets create some minor, but still visible, differences.

### 2.1.1 Register Display and Modification

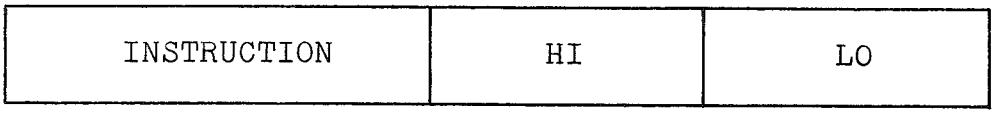
The S6809 registers consist of the program counter, flag register, stack register, two 8-bit accumulators and two 16-bit index registers. The R command described in Chapter 5 of the SYM-1 Manual is uniquely programmed to display and modify these S6809 registers. It is one of only two SYM-1 commands which is tailored to a specific CPU.

### 2.1.2 Transfer Vectors

The S6809 instruction set includes an indirect jump to subroutine command (JSR) which allows all S6809 SUPERMON transfer vectors to be 2 bytes long. The JSR to the appropriate routine is made through the address in the vector, rather than by programming a jump to the vector, which jumps to the routine.

### 2.1.3 Address Formation

All S6809 microprocessor 2-byte address constants and extended addresses within the instruction must be in the following form:




BYTE 1 BYTE 2 BYTE 3

LO = Low-order 8 bits of address

HIGH = High-order 8 bits of address

| NAME | ADDRESS | REGISTERS ALTERED | FUNCTION(S) |
|---|---|---|---|
| *MONITR | 8000 | | Cold entry to the monitor. Stack and initialize. |
| *WARM | 8004 | | Warm entry to the monitor. |
| USRENT | 802B | | User pseudo-interrupt entry saves all registers when entered with JSR. Displays PC and code 3. Control passes to the monitor. |
| SAVINT | 807A | ALL | Saves all registers when called after interrupt. Returns by using RTS. |
| DBOFF | 80BC | A,F | Simulates depressing DEBUG off key. |
| DBON | 80C5 | A,F | Simulates depressing DEBUG on key. |
| DENEW | 80D7 | A,F | Release DEBUG mode to key control. |
| GETCOM | 80E0 | A | Get the command and 0-3 parameters. System RAM is unprotected.<br>No error: **A=0D (CR).**<br>Error: **A** contains erroneous entry. |
| DISPAT | 817A | A,F | Dispatch to execute blocks. Dispatch to URCVEC if there is an error. If there is an error at return, then Carry is set and **A** contains a byte in error. |
| ERMSG | 819E | F | If Carry is set, print **(CR)ER NN**, where **NN** is contents of **A.** |
| SAVER | 8213 | NONE | Save all the registers on the stack. At return, stack looks like:<br>F<br>A<br>B<br>DP<br>X<br>Y<br>U |

*Do not enter by JSR.

**TABLE 2-1: MONITOR CALLS, ENTRIES AND TABLES**

| NAME | ADDRESS | REGISTERS ALTERED | FUNCTION(S) |
|---|---|---|---|
| *RESXAF | 822B | RESTORED | Jumped to here after SAVER to restore the registers from the stack except **A** and **F**. Perform RTS. |
| *RESXF | 8231 | RESTORED | Jumped to here after SAVER to restore the registers from the stack except **F**. erform RTS. |
| *RESALL | 8235 | RESTORED | Jumped to here after SAVER to restore all the registers from the stack. Perform RTS. |
| INBYTE | 8246 | A,F | Get two ASCII hex digits from INCHR and pack into byte in **A**. If Carry is set and **V** clear, the first digit is non-hex. If Carry is set and **V** is set, the second digit is non-hex. **N** and **Z** reflect the compare with a carriage return if Carry is set. |
| PSHOVE | 8277 | F | Shove Parms down 16 bits. Move P2 TO P1, P3 to P2 and zeros to P3. |
| PARM | 8128 | A,F | Get 0 - 3 parameters. Return by **CR** or error. **A** contains the last character entered. The flags reflect the compare with **CR**. |
| ASCNIB | 8292 | A,F | Convert an ASCII character in **A** to 4 bits in low nibble of **A**. Carry is set if non-hex. |
| OUTPC | 8304 | A,B,F | Print user PC. At return, A=PCH and B=PCL. |
| OUTABH | 8307 | F | Print **A** and **B** (four hex digits). |
| OUTBYT | 8310 | F | Print **A** (two hex digits). |
| NIBASC | 8320 | A,F | Convert low nibble of **A** to ASCII. Hex in **A**. |
| COMMA | 8342 | F | Print comma. |

*Do not enter by JSR.

**TABLE 2-1: MONITOR CALLS, ENTRIES AND TABLES (Continued)**

| NAME | ADDRESS | REGISTERS ALTERED | FUNCTION(S) |
|---|---|---|---|
| CRLF | 8355 | F | Print (CR) (LF) |
| DELAY | 8363 | F,B | Delay according to trace velocity. Relation is approximately logarithmic (base = 2). Result of INSTAT is returned in Carry. |
| INSTAT | 838A | F,A | If a key is down, wait for its release. Carry is set if the key was down (vectored thru INSVEC). |
| GETKEY | 83D1 | A,F | Get a key from the hex keyboard (more than one if SHIFT or ASCII key is used) and return with ASCII or HASH Code in **A**. Scan the display while waiting (vectored through SCNVEC). |
| HDOUT | 8418 | A,B,X,F | Transfer an ASCII character from **A** to hex display, scan display once and return with Z=1 if a key is down. |
| SCAND | 841B | A,B,X,F | Scan the LED display once from the data in DISBUF. Return with **Z** set if a key on the hex is down. |
| KEYQ | 834F | A,B,F | Determine if a key is down on the hex keyboard. If a key is down, then Z=1. |
| KYSTAT | 848B | A,B,F | Determine if a key is down. If a key is down, then Carry is set. |
| BEEP | 949A | NONE | Beep the on-board beeper. |
| HKEY | 84CF | A,F | Get a key from the hex keyboard and echo it in DISBUF. ASCII returns in **A**. Scan the display while waiting (vectored thru SCVNVEC). |
| OUTDSP | 84DE | NONE | Convert ASCII in **A** to segment code and put it in DISBUF. |

*Do not enter by JSR.

**TABLE 2-1: MONITOR CALLS, ENTRIES AND TABLES (Continued)**

| NAME | ADDRESS | REGISTERS ALTERED | FUNCTION(S) |
|---|---|---|---|
| TEXT | 8516 | F | Shove scope buffer down and push **A** into SCPBUF. |
| INCHR | 8399 | A,F | Get a character vectored thru INVEC. Drop parity and convert to upper case. If the character **CTL O (OF)**, toggle bit 6 of TECHO and get another. |
| NBASOC | 83BD | A,F | Convert the low nibble of **A** to ASCII. The output is vectored thru OUTVEC. |
| OUTCHR | 83BD | A,F | Output an ASCII character from **A** (vectored thru OUTVEC). Output is inhibited by bit 7 of TECHO. |
| INTCHR | 852C | A,F | Get a character from the serial ports. Echo is inhibited by bit 7 of TECHO. Baud rate is determined by SDBYT. Input with echo masked with TOUTFL. |
| TSTAT | 84C2 | B,F | See if a Break Key is down on the terminal. If one is down, then Carry is set. |
| *RESET | 84C2 | ALL | Initialize all registers, disable Power On Reset (POR), stop tape, initialize system RAM to default values, determine input on the keyboard or terminal, and determine baud rate and cold minitor entry. |
| *NEWDEV | 8AE5 | ALL | Determine the baud rate and cold monitor entry. |
| ACCESS | 81F6 | NONE | Un-write protect system RAM. |
| NACCESS | 820A | NONE | Write protect system RAM. |
| *TTY | 81D2 | A,X,F | Set vectors, TOUTFL and SDBYT for TTY. |

*Do not enter by JSR.

**TABLE 2-1: MONITOR CALLS, ENTRIES AND TABLES (Continued)**

| NAME | ADDRESS | REGISTERS ALTERED | FUNCTION(S) |
|---|---|---|---|
| *DFTBLK | 8FA0 | TABLE | Default block is entirely copied into System RAM (A620 - A67F) at reset. |
| *ASCII | 8B29 | TABLE | Table of ASCII codes and HASH codes. |
| *SEGS | 8B6A | TABLE | Table of segment codes correspond to ASCII codes above. |
| *Do not enter by JSR. | | | |

**TABLE 2-1: MONITOR CALLS, ENTRIES AND TABLES (Continued)**

### 2.1.4 User Interrupt Vectors

Interrupts are discussed at length in Section 7.4 of the SYM-1 Manual. Following are details on the specific interrupts available on the S6809 and their implementation on the SYM-1. There are seven types of interrupts on the S6809:

NMI - Non-maskable Interrupt
RST - Reset
IRQ - Interrupt Request
FIRQ - Fast Interrupt Request
SW1 - First Software Interrupt
SW2 - Second Software Interrupt
SW3 - Third Software Interrupt

When one of these interrupts occurs, the CPU locates its next address at one of the following locations:

FFF2, FFF3 - SW3
FFF4, FFF5 - SW2
FFF6, FFF7 - FIRQ
FFF8, FFF9 - IRQ
FFFA, FFFB - SWI
FFFC, FFFD - NMI
FFFE, FFFF - RESET

If SYM-1 interrupt routines are used, the following interrupt codes will be displayed:

0 - SW1 Instruction
1 - IRQ
2 - NMI
3 - User Entry
4 - FIRQ
5 - SW2
6 - SW3

If it is necessary to supply your own interrupt rountines, you may store their addresses at the proper places in address FFF2-FFFF. The default addresses in these interrupt vectors are for the rountines which, save all registers, display the above interrupt codes and return to the monitor.

Also, when using the vectors UIRQVC and UBRKVC for the IRQ and software (BRK) interrupts, respectively, SYM-1 determines whether the condition causing the interrupt is an IRQ or a BRK. It then saves all registers and jumps through the appropriate vector.

There are no supplementary user interrupt vector locations in the S6809 version of SUPERMON as there are in the SY6502 version. This is because the S6809 automatically vectors BRKs through separate vectors, while the SY6502 uses the same interrupt vector locations for the BRK and IRQ. The SY6502 SUPERMON then provides a routine which determines if the interrupt was caused by a BRK and jumped through the appropriate user vector. This routine is not necessary for the S6809.

The following Table 2-2 is the layout of the SY6502 system RAM for the S6809.

| SYMBOL | ADDRESS | DEFAULT | COMMENT |
|---|---|---|---|
| SCPBUF | A600 thru A61f | 00 ... 00 | 32 Bytes Oscilliscope Buffer |
| JTABLE | A620 | C0 | JUMP entry 0 - Basic |
| | A621 | 00 | Socket P1 |
| | A622 | 81 | JUMP Entry 1 - TTY |
| | A623 | D2 | |
| | A624 | 8A | JUMP Entry 2 - NEWDEV |
| | A625 | E5 | |
| | A626 | 00 | JUMP Entry 3 - Page |
| | A627 | 00 | Zero |
| | A628 | 02 | JUMP Entry 4 - 0200 |
| | A629 | 00 | |
| | A62A | 03 | JUMP Entry 4 - 0200 |
| | A62B | 00 | |
| | A62C | C8 | JUMP Entry 6 - User |
| | A62D | 00 | Socket |
| | A62E | D0 | JUMP Entry 7 - User |
| | A62F | 00 | Socket |
| TAPDEL | A630 | 04 | Tape Delay (90 seconds) |
| KMDRY | A631 | 2C | KIM Tape Boundary |
| HSBDRY | A632 | 46 | High Speed Tape Boundary |
| TAPET2 | A633 | 33 | High Speed Tape Second 1/2 Bit |

**TABLE 2-2: SYSTEM RAM LAYOUT**

| SYMBOL | ADDRESS | DEFAULT | COMMENT |
|---|---|---|---|
| TAPET1 | A634 | 59 | HIGH Speed Tape First 1/2 Bit |
| RC | A635 | 00 | |
| SCR6 | A636 | 00 | Scratch or Checksum |
| SCR7 | A637 | 00 | Accumulation |
| SCR8 | A638 | 00 | Monitor |
| SCR9 | A639 | 00 | |
| SCRA | A63B | 00 | Scratch Areas |
| SCRB | A63B | 00 | |
| DISBUF | A63C | 00 | Display Buffer |
| | A63D | 00 | |
| | A63E | 6D | |
| | A63F | 6E | |
| | A640 | 86 | |
| RDIG | A641 | 06 | Right-most Display Digit |
| PARNR | A642 | 00 | Number of Entered Parameters |
| P3H | A643 | 00 | Third Parameter |
| P3L | A644 | 00 | |
| P2H | A645 | 00 | Second Parameter |
| P2L | A646 | 00 | |
| P1H | A647 | 00 | First Parameter |
| P1L | A648 | 00 | |
| OLD | A649 | 00 | Work Area |
| | A64A | 00 | |
| PADBIT | A64B | 01 | Number of Pad Bits on Carriage Return |
| SDBYT | A64C | 4C | BAUD Rate (Note 1) |
| ERCHT | A64D | 00 | Error Count (Note 2) |
| TECHO | A64E | 80 | Terminal Echo (Note 3) |
| TOUTFL | A64F | B0 | In/Out Enable Flag (Note 4) |
| KSHFL | A650 | 00 | Hex Keyboard Shift Flat |
| TV | A651 | 00 | Trace Velocity (Note 5) |
| LSTCOM | A652 | 00 | Last Monitor Command |
| MAXRC | A653 | 10 | Maximum Number Bytes Paper Tape Received (Note 6) |
| PCHR | A654 | 8A | User PC Register |
| PCLR | A655 | C7 | Storage |
| SR | A656 | 01 | User System Stack |
| | A657 | FF | Register Storage |
| UR | A658 | 00 | User Register Storage |
| | A659 | 00 | |
| FR | A65A | 00 | User CC Register Storage |
| AR | A65B | 00 | User Register A Storage |
| BR | A65C | 00 | User Register B Storage |
| DPR | A65D | 00 | User DP Register Storage |
| XR | A65E | 00 | User Register X Storage |
| | A65F | 00 | |

**TABLE 2-2: SYSTEM RAM LAYOUT (Continued)**

| SYMBOL | ADDRESS | DEFAULT | COMMENT |
|---|---|---|---|
| YR | A660 | 00 | User Register Y Storage |
| | A661 | 00 | |
| INVEC | A662 | 84 | Address of Character |
| | A663 | CF | Input Routine |
| OUTVEC | A664 | 84 | Address of Character |
| | A665 | 18 | Output Routine |
| INSVEC | A666 | 84 | In-status Vector |
| | A667 | 8B | |
| URSCVEC | A668 | 82 | Unrecognized Syntax |
| | A669 | 3B | Vector |
| URCVEC | A66A | 82 | Unrecognized Command |
| | A66B | 3B | Vector |
| SCNVEC | A66C | 84 | Display Scan Vector |
| | A66D | 1B | |
| EXEVEC | A66E | 8A | Input Pointer for |
| | A66F | 97 | EXEC from RAM |
| TRCVEC | A670 | 80 | User Trace Vector |
| | A671 | 5D | |
| SW3VEC | A672 | 80 | Third Software |
| | A673 | 24 | Interrupt Vector |
| SW2VEC | A674 | 80 | Second Software |
| | A675 | 1D | Interrupt Vector |
| FIRQVC | A676 | 80 | Fast Interrupt Request |
| | A677 | 3E | Vector |
| IRQVEC | A678 | 80 | Interrupt Request Vector |
| | A679 | 0F | |
| SW1VEC | A67A | 80 | First Software Interrupt |
| | A67B | 16 | |
| NMIVEC | A67C | 80 | NMI Vector |
| | A67D | 53 | |
| RSTVEC | A67E | 8A | Reset Vector |
| | A67F | C7 | |

## SYSTEM RAM - NOTES

1. **BAUD RATE**

| BAUD | SDBYT |
|---|---|
| 110 | D5 |
| 300 | 4C |
| 600 | 24 |
| 1200 | 10 |
| 2400 | 06 |
| 4800 | 01 |

2. **ERCNT** Used by FILL, B M O V

Number of bytes which failed to write correctly and invalid checksums up to $FF. Used by LD P as a count of bytes which failed to write correctly (up to $F) in low nibble and a count of invalid checksums (up to $F) in high nibble.

| | | |
|---|---|---|
| 3. | **TECHO** | Bit 7 - ECHO/NO ECHO<br><br>Bit 6 - OUTPUT/NO OUTPUT<br><br>This bit is toggled every time a control **O** (ASCII OF) is encountered in the input stream. |
| 4. | **TOUTFL** | Bit 7 = Enable CRT IN<br>Bit 4 = Enable CRT OUT |
| 5. | **TV** | Trace velocity<br>00 = Single step<br><br>Non-zero - Print program counter and accumulator. Pause and resume.<br><br>Pause depends on trace velocity (9TRY TV = 09) |
| 6. | **User PC** | Default = 8AC7 = Reset |
| 7. | **NEWDEV** | Change the baud rate on RS-232C interface |

**TABLE 2-2: SYSTEM RAM LAYOUT (Continued)**

### 2.1.5 Instruction Trace Implementation

The S6809 hardware implementation provides a pulse signal each time an address is on the address bus which is not in the address ranges normally used by the monitor. Using this signal and a flag set when DEBUG is depressed, the S6809 will produce an NMI for each instruction fetch. As explained in Section 7.6 of the Sym-1 Manual, the NMI interrupt may be used to single step through a set of instructions or to trace the execution of a set of instructions. If user routines are added, the system trace routine will be used. Following is the S6809 implementation of the user trace routine (UTRC) as described in the SYM-1 Manual.

```
Page - 1 TRACE          FILE:SYM6809


0000:                   .PROC TRACE
Current memory available:    8711
0000;     ;             .NOPATCHLIST
0000;     ;
0000;     ;             UTRC - USER TRACE ROUTINE FOR S6809
                        PRINT NEXT OP CODE INSTEAD OF ACCUMULATOR
                        THIS ROUTINE IS ENTIRELY RELOCATABLE
0000;     ;
0000;     83 3D         OPCCOM    .EQU    8340H    ;PRINT PC, PRINT C
0000;     A6 54         PCHR      .ECU    0A655H
0000;     A6 55         PCLR      .EQU    0A655H   ;PRINT BYTE FROM ACC
0000;     83 50         OBCRLF    .EQU    8353H    ;PRINT BYTE FROM ACC
0000;                                                ;THEN PRINT CR,LF
0000;     83 60         DELAY     .EQU    8363H    ;DELAY BASED ON TV
```

```
0000;    80 04          WARM      .EQU   8004H    ;WARM MONITOR DELAY
0000;    80 74          TRACON    .EQU   8076H    ;TURN TRACE ON
                                                  ;THEN RESUME EXEC
0000;    A6 51          TV        .EQU   OA651H   ;TRACE VELOCITY
0000;                   ;
0000;                   ;
0000;    BD 83 40       UTRC      JSR    OPCCOM   ;PRINT PC, COMMA
0003;    A6 9F A6 54               LDA    @PCHR    ;LOAD OP CODE
0007;    BD 83 53                  JSR    OBCRLF   ;OUTPUT OP CODE, CRLF
000A;    7D A6 51                  TST    TV       ;GET TRACE VELOCITY
000D;    27 05                     BEQ    NOGO     ;NOGO IF ZERO
000F;    BD 83 63                  JSR    DELAY    ;DELAY ACCORDING TO TV
0012;    24 03                     BCC    GO       ;CARRY SET IF KEY DOWN
0014;    7E 80 04       NOGO       JMP    WARM     ;HALT
0017;    7E 80 76       GO         JMP    TRACON   ;CONTINUE
001A;                              .END
```

The above user trace example is relocatable, which means that in theory, it may be intered into memory at any user RAM location. However, in practice, any user trace routine must be fully contained in address pages 00 or 01 because of the instruction trace implementation described here. If this restriction is not observed, the trace routine will be traced and interrupts will occur.

With the S6809 CPU, only a single vector needs to be changed to use URTC. Change TRCVEC to point to UTRC where A670 is the address of UTRC. Then, enter a non-zero value in trace velocity, turn DEBUG on and you are ready to trace.

SD (UTRC), (A670) (CR)

Because the S6809 CPUs do not have SYNC outputs, hardware has been added to the adapter PCB which will generate an NMI when the following two statements are true: DEBUG is on, and an address is outputted by the CPU which is not in the monitor, I/O or address pages 00, 01 or FF.

Since there is no way of knowing if an address refers to an Op Code, operand or data, a **Monitor** command, which accesses user RAM above address page 01, will generate an NMI if DEBUG is on. DEBUG must be off if any commands such as **Memory, Verify, Deposit, Load (Store) Paper Tape** or **Load (Store) Cassette Tape** are to be used.

It is important to turn DEBUG off to examine memory during single stepping and turn it back on before resuming single stepping. Turn DEBUG off whenever program execution is complete.

Address pages 01 or 00 must be used for the system stack to debug user programs, otherwise portions of the monitor (which will use that stack) will be traced, destroying the user registers saved in RAM.

### 2.1.6 Mixed I/O Configurations

The following short routine must be entered into RAM if S6809 input is to be echoed on the terminal device. Complete the sequence discussed in Section 7.7 of the Sym-1 Manual, then, enter the following routine:

```
UIN     JSR     GETKEY
        BITA    TECHO
        BPL     UOUT
        JMP     OUTCHR
UOUT    RTS
```

Enter the UIN routine at any user RAM location, then use the **SD** command to put the UIN address into INVEC.

### 2.1.7 Monitor Extension

The following is the S6809 implementation of the monitor extension command **UO** which will "AND" together the first two parameters and display the two hex byte results.

```
LOGAND  CMPA    #14H        ;USRO
        BNE     NEXT
        CMPX    #2          ;two parms
        BNE     NEXT
DOAND   LDD     P2H         ;Load Parm 2
        ANDA    P3H         ;here's the "AND" high
        ANDB    P3L         ;"AND" Low
        JSR     CRLF        ;get new line
        JMP     OUTABH      ;print A and B, return to monitor
NEXT    ORCC    #01         ;set carry
        RTS
        .END
```

To attach LOGAND to the monitor, it must be assembled, entered into memory and URCVEC altered to contain the address of LOGAND. Note that more than one command could have been added by pointing NEXT to the next possible command instead of an RTS.

### 2.1.8 SUPERMON as an Extension to User Routines

Because SUPERMON contains a user entry, it can easily be appended to enhance your software. For instance, if a trace routine used an **M** command, the SUPERMON would be available to the user. An example of this code resembles the following:

```
USRRTN
...                         User Code
        JSR INCHR
        CMPA #'M
        BNE ELSE
        JSR USRENT
ELSE    ....                Code is executed if character
                            input is not "M".
```

In this example, the user will type an **M** to get into the monitor and a **(G) (CR)** to return to the calling portion of USSRTN. Note that the user PC and S registers should not be modified while in the monitor if a return to UTRACE is intended.

## 2.2 S6809 INSTRUCTION SET AND ASSEMBLY LANGUAGE

### 2.2.1 S6809 Microprocessor Assembly Language Syntax

The S6809 microprocessor used in your SYM-1 is an 8-bit or 16-bit CPU which is capable of processing either 8 or 16 bits of data at a time depending upon the instruction being executed. It has a 16-bit address bus, which means that up to 65,535 bytes of memory may be accessed, and it features the many interrupts described in earlier sections. It also has two 8-bit accumulators, which can be treated as a single 16-bit accumulator; two 16-bit index registers; two 16-bit stack registers, which facilitate subroutine and interrupt routine linkages; a direct (default zero) page register, which can be used to conserve memory and speed execution; an 8-bit hardware multiply command; and an almost infinite variety of instruction addressing modes.

An assembly language program consists of the following possible parts:

**LABEL** — This is optional. It is used to allow branching to the line containing the label and for certain addressing situations.

**MNEMONIC** — This is required. The mnemonic is a three- or four-character abbreviation which represents the instruction to be carried out. Thus, the mnemonic to store the contents of the accumulator in a specific memory location is STAA (Store Accumulator A).

**OPERAND(S)** — Some may be required or none may be allowed. This depends entirely upon the instruction itself and may be determined from the following discussion.

**COMMENT** — This is optional. It is separated from the last operand or from the command mnemonic where no operand is used by at least one blank. These words are ignored by the assembler program but included only to allow programmers and users to understand the program.

Note that some of the instructions make use of direct addressing, an important concept like page-zero addressing introduced briefly in Chapter 4 of the SYM-1 Manual. Page-zero addressing modes are designed to reduce memory requirements and provide faster execution. When the S6809 processor encounters an instruction using direct addressing, it assumes the high-order bytes of the address are supplied by the direct-page register, which means you do not need to define that byte in your program. This technique is particularly useful in dealing with working registers and intermediate values.

### 2.2.2 S6809 Instruction Set

The S6809 Instruction Set Summary card provides a summary of the S6809 instruction set. Each instruction is shown with its mnemonic, a brief description of the function(s) it carries out and the corresponding Op Code for its valid addressing modes. The Op Code is the hex representation of the instruction and what will appear when the instruction byte is displayed by SUPERMON.

Applications programs for the SYM-1 normally will be written in the S6809 assembly language mnemonic structure. The user then must perform a "hand assembly" to generate the Op Codes and operands. The hand-assembling of a code is explained in greater detail in Section 9.3 of the SYM-1 Manual. Refer to this table or to the S6809 Instruction Set Summary card frequently during programming.

To understand the assembly instructions, be familiar with the six condition-code register flags and their functions (listed below). These flags are set and reset depending on the program execution.

| | |
|---|---|
| **E** | Entire Flag (PC, Flags and all Register) on stack. |
| **F** | When one, FIRQ to CPU is held pending. |
| **H** | Half carry. Set to one when the low-order 4 bits of a sum exceeds 9 (used in decimal adjust). |
| **I** | When IRQ to the CPU is held pending. |
| **N** | Set to one when the result of the previous instruction is negative. |
| **Z** | Set to one when the result of the previous instruction is zero. |
| **V** | Set to one when the result of the previous instruction causes an arithmetic overflow. |
| **C** | Set to one when the previous instruction results in an arithmetic carry. Set to one when the previous instruction results in borrow (subtract). Also modified by shift, rotate and compare instructions. |

The S6809 has an extensive set of 10 addressing modes. For example, it has 59 basic instructions. However, it will accept 1,464 variations of addressing modes and instructions. Table 2-3 describes these addressing modes.

## 2.3 EXAMPLES FOR PROGRAMMING THE SYM-1/S6809

Creating a program on the SYM-1 using the S6809 involves several steps. First, the program input and desired output must carefully be defined. The flow of program logic usually is depicted graphically in the form of a flowchart. Then, the flowchart symbols are converted to assembly language instructions. These instructions are translated into machine language, which is entered into memory and executed. If the program does not run correctly the first time, it must be debugged to uncover the errors. This chapter illustrates the steps involved in creating a program that adds two 32-bit binary numbers. Two additional programming problems with suggested solutions also are provided. These three programs are designed to communicate basic programming principles and techniques, and to demonstrate a programmer's approach to simple problems.

TABLE 2-3: S6809 ADDRESSING MODES

| MODE | DESCRIPTION | # BYTES | EXAMPLE |
|---|---|---|---|
| **INHERENT** | In this addressing mode, the Op Code of the instruction contains all the address information necessary. | 1 | **CLRB:** Clear accumulator B. |
| **IMMEDIATE** | In immediate addressing, the effective address of the data is the location immediately following the Op Code (i.e., the data to be used in the instruction immediately follows the Op Code of the instruction). The S6809 uses both 8- and 16-bit immediate values depending on the size of the argument specified by the Op Code. | 2 & 3 | **LDA #20H:** Load hex 20 into accumulator A. |
| **EXTENDED** | In extended addressing, the contents of the 2 bytes immediately following the Op Code fully specify the 16-bit effective address used by the instruction. Note that the address generated by an extended instruction defines an absolute address and is not position independent. | 3 | **STX SAVX:** Store index register X into the location whose symbolic label is SAVX. |
| **EXTENDED INDIRECT** | As a special case of indexed addressing (discussion below), one level of indirection may be added to the extended addressing. In extended indirect, the 2 bytes following the post byte of an indexed instruction contains the address of the address of the data. | 4 | **LDA (SAVX):** Load accumulator A with the byte whose address is stored at location SAVX. |

TABLE 2-3: S6809 ADDRESSING MODES (Continued)

| MODE | DESCRIPTION | # BYTES | EXAMPLE |
|---|---|---|---|
| DIRECT | Direct addressing is similar to extended addressing except only 1 byte of address follows the Op Code. This byte specifies the lower 8 bits of the address to be used. The upper 8 bits are supplied by the direct page register. Since only 1 byte of the address is required, this mode requires less memory and executes faster than extended addressing. Of course, only 256 locations (one page) can be accessed without redefining the contents of the direct page register. | 2 | **LDA $20:** Load accumulator A with the value stored at the address computed as the sum of 20 (hex) and the contents of the direct page register times 100H. |
| INDEXED | In indexed addressing modes, the effective address of the operand is calculated by using one of the registers X, Y, U, S or PC. The five types of indexing are discussed below. The type of indexing and the register to be used are selected by the post byte. | | |
| | **Zero Offset Indexed** | | |
| | In this mode, the register selected by the post byte contains the effective address of the operand. | | **LDA ,X:** Load accumulator A with the value stored at the address contained in X. |

TABLE 2-3: S6809 ADDRESSING MODES (Continued)

| MODE | DESCRIPTION | # BYTES | EXAMPLE |
|---|---|---|---|
| INDEXED | **Constant Offset Indexed**<br><br>In this mode, a signed two's complement offset is added to the effective addreess of the operand. The register's content is not altered. The three sizes of offset are: -16 to +15 range using 5 bits of the post byte; -128 to +127 range using the byte following the post byte; and -32768 to +32767 using the 2 bytes following the post byte. | | **LDA - 20H,X:** Load accumulator A with the value stored at the address computed as the sum of 20 (hex) and contents of X. |
| | **Accumulator Offset Indexed**<br><br>The signed two's complement content of accumulator A, B or D is added to the register selected by the post byte to yield the effective address of the operand. Neither the content of the accumulator nor that of the register is altered. | | **LDA B,X:** Load accumulator A with the value stored at the address computed as the sum of the signed contents of B and X. |
| | **Auto Increment Indexed**<br><br>This addressing mode operates like the zero offset indexed mode. The effective address is contained in the register indicated by the post byte. After the register is used, it is incremented by one or two. The increment is selected in the post byte. | | **LDA ,X+:** Load accumulator A with the value stored at the address contained in X, then increment X by one. |

TABLE 2-3: S6809 ADDRESSING MODES (Continued)

| MODE | DESCRIPTION | # BYTES | EXAMPLE |
|---|---|---|---|
| INDEXED | **Auto Decrement Indexed**<br><br>This addressing mode is identical to auto increment indexed except the register is decremented by one or two before it is used. These predecrement and postincrement addressing modes can be used to create stacks that behave identically to the U and S stacks. | | **LDA ,Y-:** Decrement Y by one, then load accumulator A with the value stored at the address contained in Y. |
| INDEXED<br>DIRECT | For all indexed addressing modes except increment, decrement by one and constant offset with a range of -16 to +15 (offset contained in the post byte), a single additional level of indirection may be specified. In indexed indirect addressing, mode, the effective address of the operand is contained at the location specified by the contents of the the index register plus offset, if any. For example, in the code below, accumulator A is loaded indirectly through an effective address which is calculated from the index register plus an offset. | 2 - 4 | **JMP (10,X):** Jump to the location whose address is contained at the address computed as the sum of 10 (decimal) plus register X. |

TABLE 2-3: S6809 ADDRESSING MODES (Continued)

| MODE | DESCRIPTION | # BYTES | EXAMPLE |
|---|---|---|---|
| **RELATIVE ADDRESSING** | The single byte or two bytes following the Op Code are treated as a signed offset and added to the program routine. If the branch condition is satisfied, the calculated address is loaded into the program counter and execution continues at the new address. | 2 - 3 | **BEQ LABELX1:** Branch to the location LABELX1 which is expressed in the instruction as an offset from the program counter. |
| **PROGRAM COUNTER RELATIVE** | In this mode, an 8- or 16-bit signed offset is added to the PC to yield the effective address of the operand. Program counter relative addressing is useful for writing position-independent software. An additional level of indirection is permitted. | 3 - 4 | **LDA 20,PC:** Load accumulator A from the location 20 bytes past the program counter. |

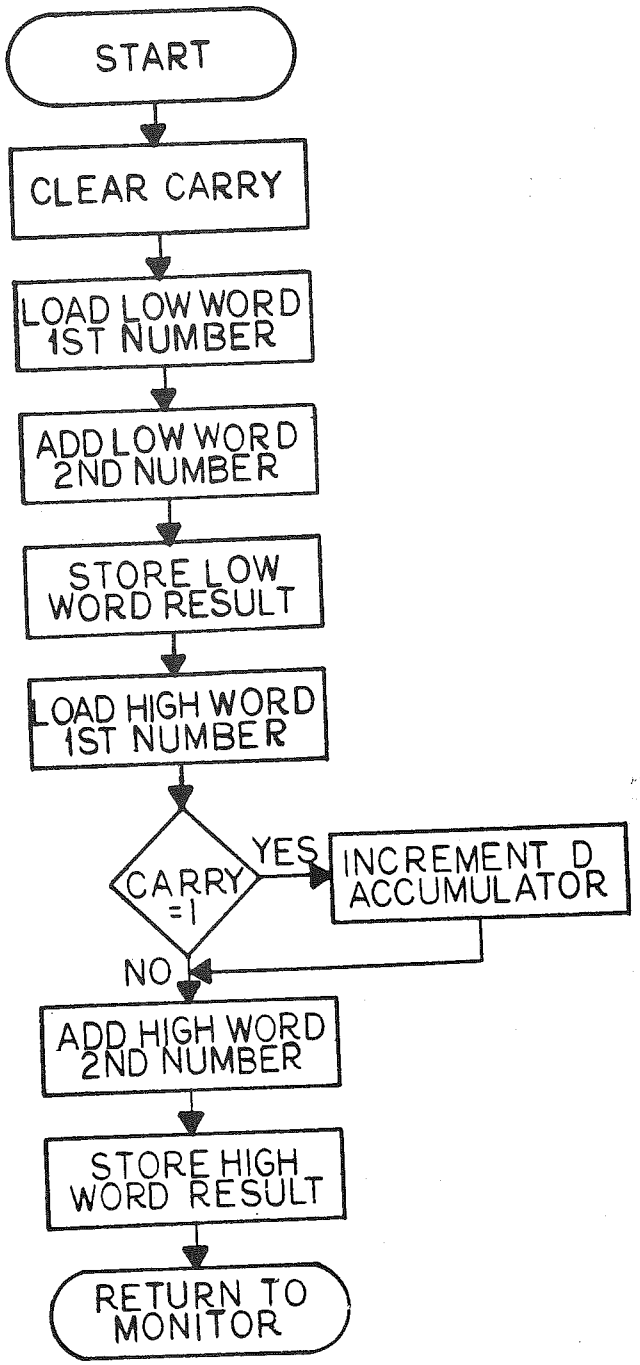



FIGURE 2-1: DOUBLE-PRECISION ADDITION FLOWCHART

### 2.3.1 Double-Precision Addition

Since the 16 bits of the accumulator can represent positive values only in the range 0-65,535, 65,535 is the largest sum that can be obtained by loading one 16-bit number into the accumulator and adding another. However, by utilizing the Carry Flag, which is set to 1 whenever the result of an addition exceeds 65,535, multiple-byte numbers may be added and the results stored in memory. A 32-bit sum can represent values up to 4,294,967,295. When 32-bit numbers are added instead of 16-bit numbers, it is "double-precision" addition.

#### 2.3.1.1 Defining Program Flow

A flowchart is easier to follow than a list of instructions because it facilitates debugging and serves as a reference when using a program written in advance. Some common flowchart symbols are shown in Chapter 6 of the SYM-1 Manual.

The object of the following program is to add two 32-bit numbers, each stored in 4 bytes of RAM, and obtain a 32-bit result. The sequence of operations the processor must perform is shown in Figure 2-1.

To accomplish double-precision addition, clear the Carry Flag. The Carry Flag is used in the program and must start at zero. Load the low-order word of the first 32-bit number into the accumulator and add the low-order word of the second number by using the add (**ADDD**) command. The contents of the accumulator makes up the low-order word of the result. The Carry Flag is set if the low-word sum is greater than FFFF (hex).

Store the accumulator contents in memory, load the high-order 2-byte word of the first number into the accumulator and add the high-order 2-byte word of the second number plus one if the carry bit is set. After the second addition, the contents of the accumulator make up the high-order 2-byte word of the result. Figure 2-2 below illustrates the addition of 98,688 and 32,896.

```
0000  0000  0000  0001  1000  0001  1000  0000   98,688 (0001 8180 hex)
0000  0000  0000  0000  1000  0000  1000  0000   32,896 (0000 8080 hex)

                Add low-order 2 bytes:  (Carry/clear)
                      1000  0001  1000  0000
                      1000  0000  1000  0000

          Carry = 1   ---------------------------
                      0000  0010  0000  0000

                      0000  0000  0000  0001
                      0000  0000  0000  0000
                            + 1  Carry

          Carry = 0   ---------------------------
                      0000  0000  0000  0010

    Result = 0000  0000  0000  0010  0000  0010  0000  0000
                           = 131,584
                       (0002, 0200 hex)
```

**FIGURE 2-2: DOUBLE-PRECISION ADDITION ARITHMETIC ILLUSTRATION**

| ADDR | INSTRUCTIONS B1 | B2 | B3 | LABEL | MNEMONIC | OPERAND | COMMENTS |
|---|---|---|---|---|---|---|---|
| 200 | 4F | | | ADD 1 | CLRA | | Clear accumulator A, which clears the Carry bit. |
| 201 | FC | 03 | 03 | | LDD | L1 | Load low-order word of first number. |
| 204 | F3 | 03 | 07 | | ADDD | L2 | Add low-order word of second number. |
| 206 | FD | 03 | 0B | | STD | L3 | Save low-order word of result. |
| 209 | FC | 03 | 01 | | LDD | H1 | Load high-order word of first number. |
| 20C | 24 | 03 | | | BCC | ADD2 | Skip to ADD2 if Carry is not set. |
| 20E | C3 | 00 | 01 | | ADDD | #2 | Add 1 for Carry. |
| 211 | F3 | 03 | 05 | ADD2 | ADDD | H2 | Add high-order word of second number. |
| 214 | FD | 03 | 09 | | STD | H3 | Store high-order word of result. |
| 217 | 7E | 80 | 00 | | JMP | MONITOR | Jump to the monitor. |
| | | | | | | | |
| | | | | | | | |
| | | | | H1 | EQU | 301H | High-order word of first number. |
| | | | | L1 | EQU | 303H | Low-order word of first number. |
| | | | | H2 | EQU | 305H | High-order word of second number. |
| | | | | L2 | EQU | 307H | Low-order word of second number. |
| | | | | H3 | EQU | 309H | High-order word of result. |
| | | | | L3 | EQU | 30BH | Low-order word of result. |
| | | | | MONITOR | EQU | 8000H | MONITOR START. |

FIGURE 2-3: DOUBLE-PRECISION ADDITION ROUTINE

### 2.3.1.2 Coding and "Hand Assembly"

Once the program has been flowcharted, you may code it onto a form like the one in Figure 2-3. The S6809 Microprocessor Machine Code is described in the S6809 Instruction Set Summary card.

The first step involves finding the S6809 commands that correspond to the operations specified in the flowchart. Arbitrary labels were assigned to represent the addresses of the monitor, the two addends and the sum entered in the operand fields. As written, the assembly language program specifies where in memory the program and data will be stored.

To store and execute the program, assemble it by translating the mnemonics into hex command codes and assign the program to a set of addresses in user RAM. When you perform this procedure with pencil and paper rather than with a special assembler program, it is called "hand assembling".

SUPERMON begins at hex location 8000, and the addends and the sum have been arbitrarily assigned to locations 0301 through 030C. Note that the high- and low-order words of a 32-bit number are not restricted to these contiguous locations.

This program will be stored beginning in locations 0200, another arbitrary choice. Data and programs may be stored anywhere in user RAM. Columns B1, B2 and B3 of Figure 2-2 represent the first 3 of the 5 possible bytes in any S6809 instruction. In this example, B1 always contains the hex Op Code, and B2 and B3 represent the operand(s). On the coding form, the LDD and BCC instructions occupy 3 bytes each. On the Instruction Set Summary card, the LDD mnemonic represents several different Op Codes depending on the addressing mode selected. FC indicates absolute addressing and specifies a 3-byte command. When all the operation codes and operands have been translated into pairs of hex digits, the program is ready to be entered into memory and executed.

### 2.3.1.3 Entering and Executing the Program

| YOU KEY IN | DISPLAY SHOWS | EXPLANATION |
|---|---|---|
| (RST) | | |
| (CR) | SY1.0.. | |
| (MEM) 200 (CR) | 0200.**. | Enter memory display and modify code |
| 4F | 0201.**. | Store FC in location 200 and advance to next location. |
| FC | 0202.**. | Store 03 in location 201 and advance to next location. |
| 03 | 0203.**. | . |
| 03 | 0204.**. | . |
| F3 | 0205.**. | . |
| 03 | 0206.**. | . |
| 07 | 0207.**. | . |
| FD | 0208.**. | . |

| | | |
|---|---|---|
| 03 | 0209.**. | . |
| 0B | 020A.**. | . |
| FC | 020B.**. | . |
| 03 | 020C.**. | . |
| 01 | 020D.**. | . |
| 24 | 020E.**. | . |
| 03 | 020F.**. | . |
| C3 | 0210.**. | |
| 00 | 0211.**. | |
| 01 | 0212.**. | |
| F3 | 0213.**. | |
| 03 | 0214.**. | |
| 05 | 0215.**. | |
| FD | 0216.**. | . |
| 03 | 0217.**. | . |
| 09 | 0218.**. | . |
| 7E | 0219.**. | . |
| 80 | 021A.**. | . |
| 00 | 021B.**. | . |
| (CR) | .**. | Exit memory display and modify. |

Once the program is entered, examine each location to make sure all values are correct. Then store the addend values in locations 301-308 using the same numbers used in Figure 2-2.

| YOU KEY IN | DISPLAY SHOWS | EXPLANATION |
|---|---|---|
| (Mem) 301 (CR) | 0301.**. | Enter high-order byte of high-order word, first addend. |
| 00 | 0302.**. | Enter low-order byte of high-order word, first addend. |
| 01 | 0303.**. | Enter high-order byte of low-order word, first addend. |
| 81 | 0304.**. | Enter low-order byte of low-order word, first addend. |
| 80 | 0305.**. | Enter high-order byte of high-order word, second addend. |
| 00 | 0306.**. | Enter low-order byte of high-order word, second addend. |
| 00 | 0307.**. | Enter high-order byte of low-order word, second addend. |
| 80 | 0308.**. | Enter low-order byte of low-order word, second addend. |
| 80 (CR) | 0309.**. | Exit memory display and modify. |

To execute the program, enter the command shown below.

| YOU KEY IN | DISPLAY SHOWS | EXPLANATION |
|---|---|---|
| (GO) 200 (CR) | g 200. | Execute program starting at location 0200. |

Now, use the **MEM** command to examine locations 309 through 30C. Verify that they are the high- and low-order words of the result, 00, 02, 02 and 00. If you find other data at these locations, the program needs to be debugged.

### 2.3.1.4 Debugging Methods

The first step in debugging is to check that the program and data have been entered correctly. Use the **MEM** command to examine the program starting address, then use the right-pointing arrow key to advance one location at a time to verify the contents of each. If you have a terminal, generate a listing by entering an **SP** command without turning on the tape punch or by using the **VER** command. Also, examine the locations that contain the initial data.

If the program and data are correct but the program still does not execute properly, you may want to use the SYM-1 DEBUG function. If DEBUG is on when the execute **(GO)** command is entered, the program will execute the first instruction, then return control to the monitor. The address on the display will be the address of the first byte of the next instruction. If you press **GO (CR)** again to execute (do not specify an address this time), the computer will execute the next instruction, then halt. Execute one step at a time in this manner.

By entering a non-zero trace velocity at location A651, execution will automatically resume after a pause, during which the accumulator is displayed. Depressing any key will halt automatic resumption.

After certain instructions, you will want to examine the contents of memory locations or registers. This can be don by using the **MEM** or **REG** commands, then resume operation by entering another **GO** command.

For example, to examine the Carry Flag after the low-order addition, use the **REG** command as shown below.

| YOU KEY IN | DISPLAY SHOWS | EXPLANATION |
|---|---|---|
| (ON) | Not important | Turn DEBUG function on. |
| (GO) 200 (CR) | 202.2 | Execute 4F instruction. |
| (GO) (CR) | 205.2 | Execute FC instruction. |
| (GO) (CR) | 208.2 | Execute F3 instruciton, low-order add with carry set or cleared as a result. |
| (REG) (CR) | PC 0208. | Program counter. |
| | S FD. | System-Stack pointer. |
| | U 00 | User-stack pointer. |
| | F 63 | Status Register. |
| (CR) | 2 63. | End register examination. |
| (GO) (CR) | 020B.2 | Execute FD instruction. |

The Carry Flag is the lowest bit of the Status Register. To determine whether the Flag was set, convert the hex digits 83 to binary. The result of this conversion is 1000 0011, and since the low bit is one, it is confirmed the sum of the 2 low-order words was greater than 65,535 (FFFF hex).

You may turn the DEBUG switch off after instruction. The next time you press **GO**, the program will finish executing.

Since reading from or writing to any I/O port is the same as reading from or writing to a memory location, the DEBUG feature also may be used to debug I/O operations. When the port address is examined with a **MEM** command, the two hex digits representing data indicate the status of each line of the port. For example, if the value C2 is displayed, pin status is:

| PIN | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 0 = LOW |
|---|---|---|---|---|---|---|---|---|---|
| STATUS | $\overline{1}$ | $\overline{1}$ | $\overline{0}$ | $\overline{0}$ | $\overline{0}$ | $\overline{0}$ | $\overline{1}$ | $\overline{0}$ | 1 = HIGH |

### 2.3.2 Conditional Testing

The most useful computer programs do not go in straight lines. That is, they do not simply execute a series of instructions in consecutive memory locations. Instead, they perform different operations by testing data words and jumping to different locations. Typical tests answer the following kinds of questions.

1. Is a selected bit of a specified data byte a one or a zero?
2. Is the content of a specified data byte equal to a selected ASCII character or numeric value?

The sample program will answer question one. It also can be patched to answer question two. Use the same principles in the first two examples to make more complicated tests.

### 2.3.3 Bit Testing

This sample program looks at the byte in hex location 31 and tests bit 3. If bit 3 is set to one, it jumps to location 8494; if bit 3 is zero, it returns to the executive. Location 8494 is a monitor subroutine that makes the SYM-1 beep.

The only problem involved in bit testing is isolating bit 3. The simplest way is to use a mask for bit 3, which is a byte in memory with bit 3 set. If you logically **AND** the mask with the sample byte, the value will be zero if bit 3 is zero and non-zero otherwise. The bit test performs the **AND** and tests the value without altering the state of the accumulator.

Figure 2-4 contains the program code for this program.

To test bit 0 or bit 7 of a byte, use a shift operation to place the selected bit in the Carry status bit and use a **BCC** or **BCS** to test Carry. This saves one or more program locations and alters the accumulator, so you may have to shift it back for later processing.

| ADDR | INSTRUCTIONS | | | LABEL | MNEMONIC | OPERAND | COMMENTS |
|---|---|---|---|---|---|---|---|
| | B1 | B2 | B3 | | | | |
| | | | | BEEP | EQU | 8494H | Beep the routine address. |
| | | | | MASK | EQU | 08H | Bit 3 mask. |
| | | | | TEST | | | Use a different value. |
| | | | | | | | |
| 200 | 96 | 30 | | | LDA | TEST | Get the test value. |
| 202 | 85 | 08 | | | BITA | MASK | Compare and set zero bit. |
| 204 | 27 | 03 | | | BEQ | HOME | If there is no bit, return. |
| 207 | 7E | 84 | 94 | | JMP | BEEP | Beep the annunciator. |
| 20A | 39 | | | | HOME | RTS | Return. |

FIGURE 2-4: BIT-3 TEST ROUTINE

### 2.3.4 Character, Value or Magnitude Testing

To test whether a byte is equal to an ASCII character or value, either use the **Compare** command or set a mask location equal to that character or value. Then use the **EOR** command to find the exclusive OR of the two values and test the result for zero. It will be zero only if the values are identical. Note that this destroys the test value, so keep an extra copy.

To test whether a byte is greater than, equal to or less than a given value, either use the **Compare** command or set a mask to the test values and subtract it from the test value. The test value will be destroyed by a **Subtract** command. Test the result to see whether it is positive, negative or zero (this takes two sequential tests) and branch accordingly. Try writing a program that makes a series of magnitude tests to determine whether a given byte is an ASCII control character (0-1F hex), punctuation mark, number or letter. The values of the ASCII character set are listed on the S6809 Instruction Set Summary card.

### 2.3.5 Multiplication

Since the S6809 contains a hardware 8-bit multiply instruction which produces a 16-bit product, a multiplication routine is not presented here.