

Innovative systems  
through silicon.

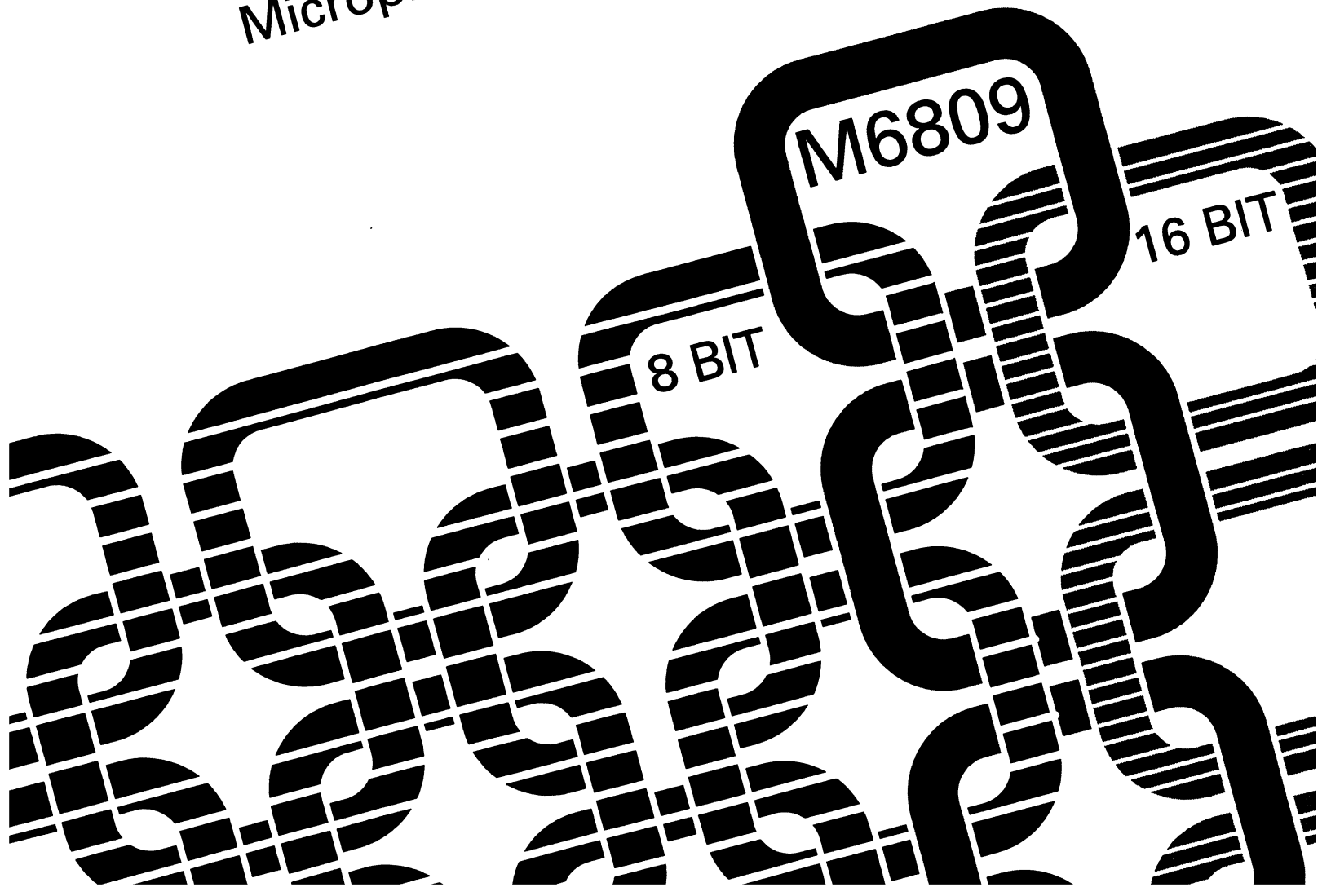


**MOTOROLA**

M6809PM/AD

# MC6809 — MC6809E

Microprocessor Programming Manual





**M6809PM (AD)**

**MC6809-MC6809E**  
**8-BIT MICROPROCESSOR**  
**PROGRAMMING MANUAL**

**Original Issue: March 1, 1981**

**Reprinted: May 1983**





# TABLE OF CONTENTS

*Paragraph No.* *Title* *Page No.*

## SECTION 1 GENERAL DESCRIPTION

|          |  |     |
|----------|--|-----|
| 1.1      | Introduction.....                                | 1-1 |
| 1.2      | Features.....                                    | 1-1 |
| 1.3      | Software Features.....                           | 1-2 |
| 1.4      | Programming Model.....                           | 1-3 |
| 1.5      | Index Registers (X, Y) .....                     | 1-3 |
| 1.6      | Stack Pointer Registers (U, S) .....             | 1-3 |
| 1.7      | Program Counter (PC) .....                       | 1-4 |
| 1.8      | Accumulator Registers (A, B, D) .....            | 1-4 |
| 1.9      | Direct Page Register (DP) .....                  | 1-4 |
| 1.10     | Condition Code Register (CC) .....               | 1-4 |
| 1.10.1   | Condition Code Bits .....                        | 1-5 |
| 1.10.1.1 | Half Carry (H), Bit 5.....                       | 1-5 |
| 1.10.1.2 | Negative (N), Bit 3.....                         | 1-5 |
| 1.10.1.3 | Zero (Z), Bit 2.....                             | 1-5 |
| 1.10.1.4 | Overflow (V), Bit 1 .....                        | 1-5 |
| 1.10.1.5 | Carry (C), Bit 0 .....                           | 1-5 |
| 1.10.2   | Interrupt Mask Bits and Stacking Indicator ..... | 1-5 |
| 1.10.2.1 | Fast Interrupt Request Mask (F), Bit 6 .....     | 1-5 |
| 1.10.2.2 | Interrupt Request Mask (I), Bit 4 .....          | 1-5 |
| 1.10.2.3 | Entire Flag (E), Bit 7 .....                     | 1-6 |
| 1.11     | Pin Assignments and Signal Description.....      | 1-6 |
| 1.11.1   | MC6809 Clocks .....                              | 1-6 |
| 1.11.1.1 | Oscillator (EXTAL, XTAL).....                    | 1-6 |
| 1.11.1.2 | Enable (E).....                                  | 1-7 |
| 1.11.1.3 | Quadrature (Q) .....                             | 1-7 |
| 1.11.2   | MC6809E Clocks (E and Q) .....                   | 1-7 |
| 1.11.3   | Three State Control (TSC) (MC6809E) .....        | 1-7 |
| 1.11.4   | Last Instruction Cycle (LIC) (MC6809E) .....     | 1-7 |
| 1.11.5   | Address Bus (A0-A15).....                        | 1-7 |
| 1.11.6   | Data Bus (D0-D7).....                            | 1-7 |
| 1.11.7   | Read/Write (R/W).....                            | 1-8 |
| 1.11.8   | Processor State Indicators (BA, BS) .....        | 1-8 |
| 1.11.8.1 | Normal .....                                     | 1-8 |
| 1.11.8.2 | Interrupt or Reset Acknowledge.....              | 1-8 |
| 1.11.8.3 | Sync Acknowledge .....                           | 1-8 |

## TABLE OF CONTENTS (CONTINUED)

| <i>Paragraph No.</i> | <i>Title</i>   | <i>Page No.</i> |
|----------------------|--|-----------------|
| 1.11.8.4             | Halt/Bus Grant .....                                       | 1-8             |
| 1.11.9               | Reset (RESET) .....  | 1-9             |
| 1.11.10              | Interrupts .....   | 1-9             |
| 1.11.10.1            | Non-Maskable Interrupt (NMI) .....                         | 1-9             |
| 1.11.10.2            | Fast Interrupt Request (FIRQ).....                         | 1-9             |
| 1.11.10.3            | Interrupt Request (IRQ).....                               | 1-9             |
| 1.11.11              | Memory Ready (MRDY) (MC6809) .....                         | 1-9             |
| 1.11.12              | Advanced Valid Memory Address (AVMA) (MC6809E) .....       | 1-10            |
| 1.11.13              | Halt (HALT).....   | 1-10            |
| 1.11.14              | Direct Memory Access/Bus Request (DMA/BREQ) (MC6809) ..... | 1-10            |
| 1.11.15              | Busy (MC6809E).....  | 1-10            |
| 1.11.16              | Power .....  | 1-11            |

## SECTION 2 ADDRESSING MODES

|         |  |     |
|---------|--|-----|
| 2.1     | Introduction.....                          | 2-1 |
| 2.2     | Addressing Modes.....                      | 2-1 |
| 2.2.1   | Inherent.....                              | 2-1 |
| 2.2.2   | Immediate .....                            | 2-1 |
| 2.2.3   | Extended.....                              | 2-2 |
| 2.2.4   | Direct.....                                | 2-2 |
| 2.2.5   | Indexed .....                              | 2-2 |
| 2.2.5.1 | Constant Offset from Register .....        | 2-2 |
| 2.2.5.2 | Accumulator Offset from Register.....      | 2-3 |
| 2.2.5.3 | Autoincrement/Decrement from Register..... | 2-3 |
| 2.2.5.4 | Indirection .....                          | 2-4 |
| 2.2.5.5 | Extended Indirect .....                    | 2-4 |
| 2.2.5.6 | Program Counter Relative.....              | 2-4 |
| 2.2.6   | Branch Relative .....                      | 2-4 |

## SECTION 3 INTERRUPT CAPABILITIES

|     |  |     |
|-----|--|-----|
| 3.1 | Introduction.....                            | 3-1 |
| 3.2 | Non-Maskable Interrupt (NMI) .....           | 3-1 |
| 3.3 | Fast Maskable Interrupt Request (FIRQ).....  | 3-2 |
| 3.4 | Normal Maskable Interrupt Request (IRQ)..... | 3-2 |
| 3.5 | Software Interrupts (SWI, SWI2, SWI3).....   | 3-2 |

## TABLE OF CONTENTS (CONCLUDED)

| <i>Paragraph No.</i>             | <i>Title</i>                   | <i>Page No.</i> |
|----------------------------------|--------------------------------|-----------------|
| <b>SECTION 4<br/>PROGRAMMING</b> |                                |                 |
| 4.1                              | Introduction.....              | 4-1             |
| 4.1.1                            | Position-Independence.....     | 4-1             |
| 4.1.2                            | Modular Programming.....       | 4-1             |
| 4.1.2.1                          | Local Storage.....             | 4-1             |
| 4.1.2.2                          | Global Storage.....            | 4-2             |
| 4.1.3                            | Reentrancy/Recursion.....      | 4-2             |
| 4.2                              | M6809 Capabilities.....        | 4-2             |
| 4.2.1                            | Module Construction.....       | 4-2             |
| 4.2.1.1                          | Parameters.....                | 4-3             |
| 4.2.1.2                          | Local Storage.....             | 4-3             |
| 4.2.1.3                          | Global Storage.....            | 4-3             |
| 4.2.2                            | Position-Independent Code..... | 4-4             |
| 4.2.3                            | Reentrant Programs.....        | 4-5             |
| 4.2.4                            | Recursive Programs.....        | 4-5             |
| 4.2.5                            | Loops.....                     | 4-5             |
| 4.2.6                            | Stack Programming.....         | 4-6             |
| 4.2.6.1                          | M6809 Stacking Operations..... | 4-6             |
| 4.2.6.2                          | Subroutine Linkage.....        | 4-7             |
| 4.2.6.3                          | Software Stacks.....           | 4-8             |
| 4.2.7                            | Real Time Programming.....     | 4-8             |
| 4.3                              | Program Documentation.....     | 4-8             |
| 4.4                              | Instruction Set.....           | 4-9             |

## APPENDIX A INSTRUCTION SET DETAILS

|     |  |     |
|-----|--|-----|
| A.1 | Introduction.....                                | A-1 |
| A.2 | Notation.....                                    | A-1 |
|     | Instructions (listed in alphabetical order)..... | A-3 |

## APPENDIX B ASSIST09 MONITOR PROGRAM

|     |                                  |     |
|-----|----------------------------------|-----|
| B.1 | General Description.....         | B-1 |
| B.2 | Implementation Requirements..... | B-1 |
| B.3 | Interrupt Control.....           | B-2 |
| B.4 | Initialization.....              | B-3 |

## TABLE OF CONTENTS (CONTINUED)

| <i>Paragraph No.</i> | <i>Title</i>               | <i>Page No.</i> |
|----------------------|----------------------------|-----------------|
| B.5                  | Input/Output Control ..... | B-4             |
| B.6                  | Command Format .....       | B-4             |
| B.7                  | Command List .....         | B-5             |
| B.8                  | Commands.....              | B-5             |
|                      | Breakpoint .....           | B-6             |
|                      | Call.....                  | B-6             |
|                      | Display.....               | B-7             |
|                      | Encode .....               | B-7             |
|                      | Go.....                    | B-8             |
|                      | Load .....                 | B-8             |
|                      | Memory .....               | B-9             |
|                      | Null .....                 | B-10            |
|                      | Offset.....                | B-10            |
|                      | Punch.....                 | B-11            |
|                      | Register.....              | B-11            |
|                      | Stlevel.....               | B-12            |
|                      | Trace.....                 | B-12            |
|                      | Verify .....               | B-13            |
|                      | Window .....               | B-13            |
| B.9                  | Services.....              | B-14            |
|                      | BKPT.....                  | B-15            |
|                      | INCHP.....                 | B-15            |
|                      | MONITR.....                | B-16            |
|                      | OUTCH .....                | B-17            |
|                      | OUT2HS.....                | B-17            |
|                      | OUT4HS.....                | B-18            |
|                      | PAUSE .....                | B-18            |
|                      | PCRLF .....                | B-19            |
|                      | PDATA .....                | B-19            |
|                      | PDATA1 .....               | B-20            |
|                      | SPACE .....                | B-21            |
|                      | VTRSW .....                | B-21            |
| B.10                 | Vector Swap Service.....   | B-22            |
|                      | .ACIA.....                 | B-23            |
|                      | .AVTBL.....                | B-23            |
|                      | .BSDTA .....               | B-24            |
|                      | .BSOFF .....               | B-24            |
|                      | .BSON .....                | B-25            |
|                      | .CIDTA .....               | B-25            |
|                      | .CIOFF .....               | B-26            |
|                      | .CION .....                | B-26            |
|                      | .CMDL1.....                | B-27            |
|                      | .CMDL2.....                | B-28            |

## TABLE OF CONTENTS (CONTINUED)

| <i>Paragraph No.</i> | <i>Title</i>         | <i>Page No.</i> |
|----------------------|----------------------|-----------------|
|                      | .CODTA.....          | B-28            |
|                      | .COOFF .....         | B-29            |
|                      | .COON .....          | B-29            |
|                      | .ECHO.....           | B-30            |
|                      | .FIRQ.....           | B-30            |
|                      | .HSDATA .....        | B-31            |
|                      | .IRQ .....           | B-31            |
|                      | .NMI .....           | B-32            |
|                      | .PAD.....            | B-32            |
|                      | .PAUSE .....         | B-33            |
|                      | .PTM.....            | B-33            |
|                      | .RESET.....          | B-34            |
|                      | .RSVD .....          | B-34            |
|                      | .SWI.....            | B-35            |
|                      | .SWI2.....           | B-35            |
|                      | .SWI3.....           | B-36            |
| B.11                 | Monitor Listing..... | B-37            |

### APPENDIX C MACHINE CODE TO INSTRUCTION CROSS REFERENCE

|     |                    |     |
|-----|--------------------|-----|
| C.1 | Introduction ..... | C-1 |
|-----|--------------------|-----|

### APPENDIX D PROGRAMMING AID

|     |                    |     |
|-----|--------------------|-----|
| D.1 | Introduction ..... | D-1 |
|-----|--------------------|-----|

### APPENDIX E ASCII CHARACTER SET

|     |  |     |
|-----|--|-----|
| E.1 | Introduction .....                                     | E-1 |
| E.2 | Character Representation and Code Identification ..... | E-1 |
| E.3 | Control Characters .....                               | E-2 |
| E.4 | Graphic Characters .....                               | E-2 |

## TABLE OF CONTENTS (CONTINUED)

*Paragraph No.* *Title* *Page No.*

### APPENDIX F OPCODE MAP

|     |                    |     |
|-----|--------------------|-----|
| F.1 | Introduction ..... | F-1 |
| F.2 | Opcode Map.....    | F-1 |

### APPENDIX G PIN ASSIGNMENTS

|     |                   |     |
|-----|-------------------|-----|
| G.1 | Introduction..... | G-1 |
|-----|-------------------|-----|

### APPENDIX H CONVERSION TABLES

|       |   |     |
|-------|---|-----|
| H.1   | Introduction.....                       | H-1 |
| H.2   | Powers of 2; Powers of 16.....          | H-1 |
| H.3   | Hexadecimal and Decimal Conversion..... | H-2 |
| H.3.1 | Converting Hexadecimal to Decimal ..... | H-2 |
| H.3.2 | Converting Decimal to Hexadecimal ..... | H-2 |

### LIST OF ILLUSTRATIONS

| <i>Figure No.</i> | <i>Title</i>  | <i>Page No.</i> |
|-------------------|---|-----------------|
| 1-1               | Programming Model.....                                | 1-3             |
| 1-2               | Condition Code Register .....                         | 1-4             |
| 1-3               | Processor Pin Assignments.....                        | 1-6             |
| 2-1               | Postbyte Usage for EXG/TFR, PSH/PUL Instructions..... | 2-2             |
| 3-1               | Interrupt Processing Flowchart.....                   | 3-5             |
| 4-1               | Stacking Order .....                                  | 4-7             |
| B-1               | Memory Map .....                                      | B-2             |
| E-1               | ASCII Character Set.....                              | E-1             |
| G-1               | Pin Assignments.....                                  | G-1             |

# LIST OF TABLES

| <i>Table No.</i> | <i>Title</i>                                      | <i>Page No.</i> |
|------------------|---|-----------------|
| 1-1              | BA/BS Signal Encoding.....                        | 1-8             |
| 2-1              | Postbyte Usage for Indexed Addressing Modes ..... | 2-3             |
| 3-1              | Interrupt Vector Locations .....                  | 3-1             |
| 4-1              | Instruction Set .....                             | 4-9             |
| 4-2              | 8-Bit Accumulator and Memory Instructions .....   | 4-11            |
| 4-3              | 16-Bit Accumulator and Memory Instructions .....  | 4-12            |
| 4-4              | Index/Stack Pointer Instructions.....             | 4-12            |
| 4-5              | Branch Instructions .....                         | 4-13            |
| 4-6              | Miscellaneous Instructions .....                  | 4-13            |
| A-1              | Operation Notation.....                           | A-1             |
| A-2              | Register Notation.....                            | A-2             |
| B-1              | Command List .....                                | B-5             |
| B-2              | Services.....                                     | B-14            |
| B-3              | Vector Table Entries .....                        | B-22            |
| C-1              | Machine Code to Instruction Cross Reference.....  | C-2             |
| D-1              | Programming Aid .....                             | D-1             |
| E-1              | Control Characters .....                          | E-2             |
| E-2              | Graphic Characters .....                          | E-3             |
| F-1              | Opcode Map.....                                   | F-2             |
| F-2              | Indexed Addressing Mode Data .....                | F-3             |
| H-1              | Powers of 2; Powers of 16.....                    | H-1             |
| H-2              | Hexadecimal and Decimal Conversion Chart .....    | H-2             |





# SECTION 1 GENERAL DESCRIPTION

## 1.1 INTRODUCTION

This section contains a general description of the Motorola MC6809 and MC6809E Microprocessor Units (MPU). Pin assignments and a brief description of each input/output signal are also given. The term MPU, processor, or M6809 will be used throughout this manual to refer to both the MC6809 and MC6809E processors. When a topic relates to only one of the processors, that specific designator (MC6809 or MC6809E) will be used.

## 1.2 FEATURES

The MC6809 and MC6809E microprocessors are greatly enhanced, upward compatible, computationally faster extensions of the MC6800 microprocessor.

Enhancements such as additional registers (a Y index register, a U stack pointer, and a direct page register) and instructions (such as MUL) simplify software design. Improved addressing modes have also been implemented.

Upward compatibility is guaranteed as MC6800 assembly language programs may be assembled using the Motorola MC6809 Macro Assembler. This code, while not as compact as native M6809 code, is, in most cases, 100% functional.

Both address and data are available from the processor earlier in an instruction cycle than from the MC6800 which simplifies hardware design. Two clock signals, E (the MC6800  $\phi 2$ ) and a new quadrature clock Q (which leads E by one-quarter cycle) also simplify hardware design.

A memory ready (MRDY) input is provided on the MC6809 for working with slow memories. This input stretches both the processor internal cycle and direct memory access bus cycle times but allows internal operations to continue at full speed. A direct memory access request ( $\overline{\text{DMA/BREQ}}$ ) input is provided for immediate memory access or dynamic memory refresh operations; this input halts the internal MC6809 clocks. Because the processor's registers are dynamic, an internal counter periodically recovers the bus from direct memory access operations and performs a true processor refresh cycle to allow unlimited length direct memory access operation. An interrupt acknowledge signal is available to allow development of vectoring by interrupt device hardware or detection of operating system calls.

Three prioritized, vectored, hardware interrupt levels are available: non-maskable, fast, and normal. The highest and lowest priority interrupts, non-maskable and interrupt request respectively, are the normal interrupts used in the M6800 family. A new interrupt on this processor is the fast interrupt request which provides faster service to its interrupt input by only stacking the program counter and condition code register and then servicing the interrupt.

Modern programming techniques such as position-independent, system independent, and reentrant programming are readily supported by these processors.

A Memory Management Unit (MMU), the MC6829, allows a M6809 based system to address a two megabyte memory space. Note: An arbitrary number of tasks may be supported — slower — with software.

This advanced family of processors is compatible with all M6800 peripheral parts.

### **1.3 SOFTWARE FEATURES**

Some of the software features of these processors are itemized in the following paragraphs. Programs developed for the MC6800 can be easily converted for use with the MC6809 or MC6809E by running the source code through a M6809 Macro Assembler or any one of the many cross assemblers that are available.

The addressing modes of any microprocessor provide it with the capability to efficiently address memory to obtain data and instructions. The MC6809 and MC6809E have a versatile set of addressing modes which allow them to function using modern programming techniques.

The addressing modes and instructions of the MC6809 and MC6809E are upward compatible with the MC6800. The old addressing modes have been retained and many new ones have been added.

A direct page register has been added which allows a 256 byte “direct” page anywhere in the 64K logical address space. The direct page register is used to hold the most-significant byte of the address used in direct addressing and decrease the time required for address calculation.

Branch relative addressing to anywhere in the memory map (– 32768 to + 32767) is available.

Program counter relative addressing is also available for data access as well as branch instructions.

The indexed addressing modes have been expanded to include:

- 0-, 5-, 8-, 16-bit constant offsets,
- 8- or 16-bit accumulator offsets,
- autoincrement/decrement (stack operation).

In addition, most indexed addressing modes may have an additional level of indirection added.

Any or all registers may be pushed on to or pulled from either stack with a single instruction.

A multiply instruction is included which multiplies unsigned binary numbers in accumulators A and B and places the unsigned result in the 16-bit accumulator D. This unsigned multiply instruction also allows signed or unsigned multiple precision multiplication.

### 1.4 PROGRAMMING MODEL

The programming model (Figure 1-1) for these processors contains five 16-bit and four 8-bit registers that are available to the programmer.

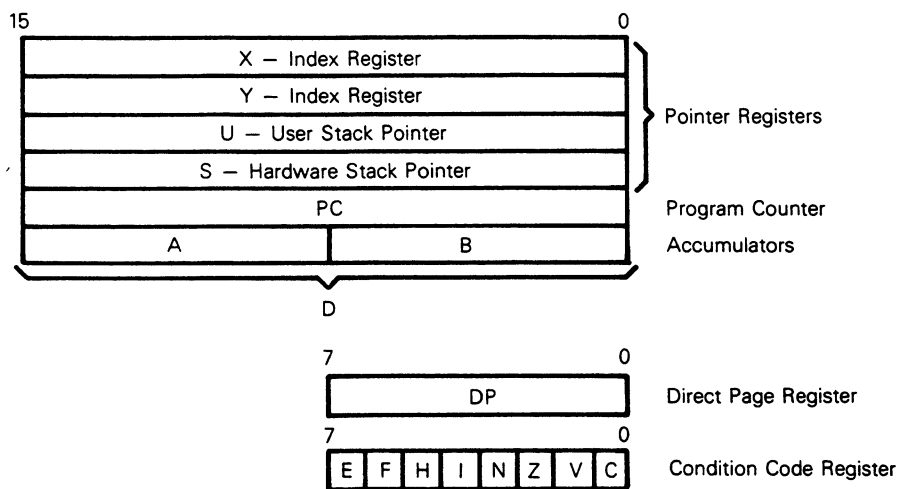


Figure 1-1. Programming Model

### 1.5 INDEX REGISTERS (X, Y)

The index registers are used during the indexed addressing modes. The address information in an index register is used in the calculation of an effective address. This address may be used to point directly to data or may be modified by an optional constant or register offset to produce the effective address.

### 1.6 STACK POINTER REGISTERS (U, S)

Two stack pointer registers are available in these processors. They are: a user stack pointer register (U) controlled exclusively by the programmer, and a hardware stack pointer register (S) which is used automatically by the processor during subroutine calls

and interrupts, but may also be used by the programmer. Both stack pointers always point to the top of the stack.

These registers have the same indexed addressing mode capabilities as the index registers, and also support push and pull instructions. All four indexable registers (X, Y, U, S) are referred to as pointer registers.

### 1.7 PROGRAM COUNTER (PC)

The program counter register is used by these processors to store the address of the next instruction to be executed. It may also be used as an index register in certain addressing modes.

### 1.8 ACCUMULATOR REGISTERS (A, B, D)

The accumulator registers (A, B) are general-purpose 8-bit registers used for arithmetic calculations and data manipulation.

Certain instructions concatenate these registers into one 16-bit accumulator with register A positioned as the most-significant byte. When concatenated, this register is referred to as accumulator D.

### 1.9 DIRECT PAGE REGISTER (DP)

This 8-bit register contains the most-significant byte of the address to be used in the direct addressing mode. The contents of this register are concatenated with the byte following the direct addressing mode operation code to form the 16-bit effective address. The direct page register contents appear as bits A15 through A8 of the address. This register is automatically cleared by a hardware reset to ensure M6800 compatibility.

### 1.10 CONDITION CODE REGISTER (CC)

The condition code register contains the condition codes and the interrupt masks as shown in Figure 1-2.

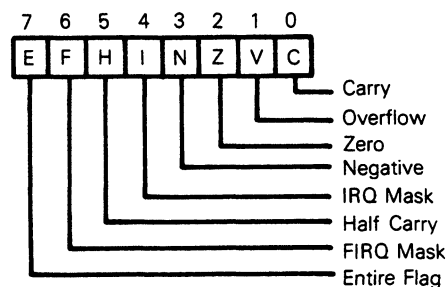


Figure 1-2. Condition Code Register

**1.10.1 CONDITION CODE BITS.** Five bits in the condition code register are used to indicate the results of instructions that manipulate data. They are: half carry (H), negative (N), zero (Z), overflow (V), and carry (C). The effect each instruction has on these bits is given in the detail information for each instruction (see Appendix A).

**1.10.1.1 Half Carry (H), Bit 5.** This bit is used to indicate that a carry was generated from bit three in the arithmetic logic unit as a result of an 8-bit addition. This bit is undefined in all subtract-like instructions. The decimal addition adjust (DAA) instruction uses the state of this bit to perform the adjust operation.

**1.10.1.2 Negative (N), Bit 3.** This bit contains the value of the most-significant bit of the result of the previous data operation.

**1.10.1.3 Zero (Z), Bit 2.** This bit is used to indicate that the result of the previous operation was zero.

**1.10.1.4 Overflow (V), Bit 1.** This bit is used to indicate that the previous operation caused a signed arithmetic overflow.

**1.10.1.5 Carry (C), Bit 0.** This bit is used to indicate that a carry or a borrow was generated from bit seven in the arithmetic logic unit as a result of an 8-bit mathematical operation.

**1.10.2 INTERRUPT MASK BITS AND STACKING INDICATOR.** Two bits (I and F) are used as mask bits for the interrupt request and the fast interrupt request inputs. When either or both of these bits are set, their associated input will not be recognized.

One bit (E) is used to indicate how many registers (all, or only the program counter and condition code) were stacked during the last interrupt.

**1.10.2.1 Fast Interrupt Request Mask (F), Bit 6.** This bit is used to mask (disable) any fast interrupt request line ( $\overline{\text{FIRQ}}$ ). This bit is set automatically by a hardware reset or after recognition of another interrupt. Execution of certain instructions such as SWI will also inhibit recognition of a  $\overline{\text{FIRQ}}$  input.

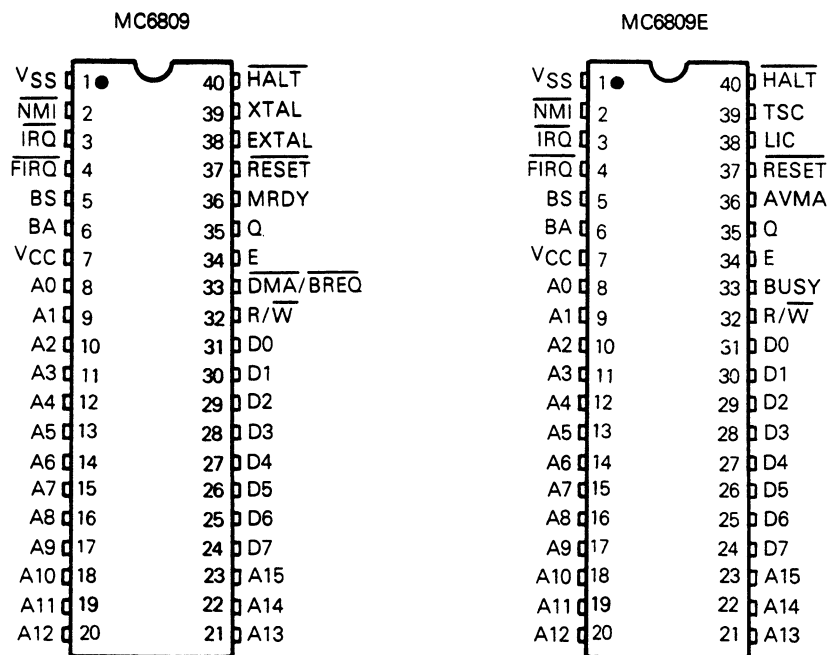
**1.10.2.2 Interrupt Request Mask (I), Bit 4.** This bit is used to mask (disable) any interrupt request input ( $\overline{\text{IRQ}}$ ). This bit is set automatically by a hardware reset or after recognition of another interrupt. Execution of certain instructions such as SWI will also inhibit recognition of an  $\overline{\text{IRQ}}$  input.

**1.10.2.3 Entire Flag (E), Bit 7.** This bit is used to indicate how many registers were stacked. When set, all the registers were stacked during the last interrupt stacking operation. When clear, only the program counter and condition code registers were stacked during the last interrupt.

The state of the E bit in the stacked condition code register is used by the return from interrupt (RTI) instruction to determine the number of registers to be unstacked.

### 1.11 PIN ASSIGNMENTS AND SIGNAL DESCRIPTION

Figure 1-3 shows the pin assignments for the processors. The following paragraphs provide a short description of each of the input and output signals.



**Figure 1-3. Processor Pin Assignments**

**1.11.1 MC6809 CLOCKS.** The MC6809 has four pins committed to developing the clock signals needed for internal and system operation. They are: the oscillator pins EXTAL and XTAL; the standard M6800 enable (E) clock; and a new, quadrature (Q) clock.

**1.11.1.1 Oscillator (EXTAL, XTAL).** These pins are used to connect the processor's internal oscillator to an external, parallel-resonant crystal. These pins can also be used for input of an external TTL timing signal by grounding the XTAL pin and applying the input to the EXTAL pin. The crystal or the external timing source is four times the resulting bus frequency.

**1.11.1.2 Enable (E).** The E clock is similar to the phase 2 ( $\phi_2$ ) MC6800 bus timing clock. The leading edge indicates to memory and peripherals that the data is stable and to begin write operations. Data movement occurs after the Q clock is high and is latched on the trailing edge of E. Data is valid from the processor (during a write operation) by the rising edge of E.

**1.11.1.3 Quadrature (Q).** The Q clock leads the E clock by approximately one half of the E clock time. Address information from the processor is valid with the leading edge of the Q clock. The Q clock is a new signal in these processors and does not have an equivalent clock within the MC6800 bus timing.

**1.11.2 MC6809E CLOCKS (E and Q).** The MC6809E has two pins provided for the TTL clock signal inputs required for internal operation. They are the standard M6800 enable (E) clock and the quadrature (Q) clock. The Q input must lead the E input.

Addresses will be valid from the processor (on address delay time after the falling edge of E) and data will be latched from the bus by the falling edge of E. The Q input is fully TTL compatible. The E input is used to drive the internal MOS circuitry directly and therefore requires input levels above the normal TTL levels.

**1.11.3 THREE STATE CONTROLS (TSC) (MC6809E).** This input is used to place the address and data lines and the R/W line in the high-impedance state and allows the address bus to be shared with other bus masters.

**1.11.4 LAST INSTRUCTION CYCLE (LIC) (MC6809E).** This output goes high during the last cycle of every instruction and its high-to-low transition indicates that the first byte of an opcode will be latched at the end of the present bus cycle.

**1.11.5 ADDRESS BUS (A0-A15).** This 16-bit, unidirectional, three-state bus is used by the processor to provide address information to the address bus. Address information is valid on the rising edge of the Q clock. All 16 outputs are in the high-impedance state when the bus available (BA) signal is high, and for one bus cycle thereafter.

When the processor does not require the address bus for a data transfer, it outputs address FFFF<sub>16</sub>, and read/write (R/W) high. This is a “dummy access” of the least-significant byte of the reset vector which replaces the valid memory address (VMA) functions of the MC6800. For the MC6809, the memory read signal internal circuitry inhibits stretching of the clocks during non-access cycles.

**1.11.6 DATA BUS (D0-D7).** This 8-bit, bidirectional, three-state bus is the general purpose data path. All eight outputs are in the high-impedance state when the bus available (BA) output is high.

**1.11.7 READ/WRITE ( $\overline{R/W}$ ).** This output indicates the direction of data transfer on the data bus. A low indicates that the processor is writing onto the data bus; a high indicates that the processor is reading data from the data bus. The signal at the  $\overline{R/W}$  output is valid at the leading edge of the Q clock. The  $\overline{R/W}$  output is in the high-impedance state when the bus available (BA) output is high.

**1.11.8 PROCESSOR STATE INDICATORS (BA, BS).** The processor uses these two output lines to indicate the present processor state. These pins are valid with the leading edge of the Q clock.

The bus available (BA) output is used to indicate that the buses (address and data) and the read/write output are in the high-impedance state. This signal can be used to indicate to bus-sharing or direct memory access systems that the buses are available. When BA goes low, an additional dead cycle will elapse before the processor regains control of the buses.

The bus status (BS) output is used in conjunction with the BA output to indicate the present state of the processor. Table 1-1 is a listing of the BA and BS outputs and the processor states that they indicate. The following paragraphs briefly explain each processor state.

**Table 1-1. BA/BS Signal Encoding**

| <u>BA</u> | <u>BS</u> | <u>Processor State</u>         |
|-----------|-----------|--------------------------------|
| 0         | 0         | Normal (Running)               |
| 0         | 1         | Interrupt or Reset Acknowledge |
| 1         | 0         | Sync Acknowledge               |
| 1         | 1         | Halt/Bus Grant Acknowledged    |

**1.11.8.1 Normal.** The processor is running and executing instructions.

**1.11.8.2 Interrupt or Reset Acknowledge.** This processor state is indicated during both cycles of a hardware vector fetch which occurs when any of the following interrupts have occurred:  $\overline{RESET}$ ,  $\overline{NMI}$ ,  $\overline{FIRQ}$ ,  $\overline{IRQ}$ , SWI, SWI2, and SWI3.

This output, plus decoding of address lines A3 through A1 provides the user with an indication of which interrupt is being serviced.

**1.11.8.3 Sync Acknowledge.** The processor is waiting for an external synchronization input on an interrupt line. See SYNC instruction in Appendix A.

**1.11.8.4 Halt/Bus Grant.** The processor is halted or bus control has been granted to some other device.



**1.11.9 RESET ( $\overline{\text{RESET}}$ ).** This input is used to reset the processor. A low input lasting longer than one bus cycle will reset the processor.

The reset vector is fetched from locations \$FFFE and \$FFFF when the processor enters the reset acknowledge state as indicated by the BA output being low and the BS output being high.

During initial power-on, the reset input should be held low until the clock oscillator is fully operational.

**1.11.10 INTERRUPTS.** The processor has three separate interrupt input pins: non-maskable interrupt ( $\overline{\text{NMI}}$ ), fast interrupt request ( $\overline{\text{FIRQ}}$ ), and interrupt request ( $\overline{\text{IRQ}}$ ). These interrupt inputs are latched by the falling edge of every Q clock except during cycle stealing operations where only the  $\overline{\text{NMI}}$  input is latched. Using this point as a reference, a delay of at least one bus cycle will occur before the interrupt is recognized by the processor.

**1.11.10.1 Non-Maskable Interrupt ( $\overline{\text{NMI}}$ ).** A negative edge on this input requests that a non-maskable interrupt sequence be generated. This input, as the name indicates, cannot be masked by software and has the highest priority of the three interrupt inputs. After a reset has occurred, a  $\overline{\text{NMI}}$  input will not be recognized by the processor until the first program load of the hardware stack pointer. The entire machine state is saved on the hardware stack during the processing of a non-maskable interrupt. This interrupt is internally blocked after a hardware reset until the stack pointer is initialized.

**1.11.10.2 Fast Interrupt Request ( $\overline{\text{FIRQ}}$ ).** This input is used to initiate a fast interrupt request sequence. Initiation depends on the F (fast interrupt request mask) bit in the condition code register being clear. This bit is set during reset. During the interrupt, only the contents of the condition code register and the program counter are stacked resulting in a short amount of time required to service this interrupt. This interrupt has a higher priority than the normal interrupt request ( $\overline{\text{IRQ}}$ ).

**1.11.10.3 Interrupt Request ( $\overline{\text{IRQ}}$ ).** This input is used to initiate what might be considered the "normal" interrupt request sequence. Initiation depends on the I (interrupt mask) bit in the condition code register being clear. This bit is set during reset. The entire machine state is saved on the hardware stack during processing of an  $\overline{\text{IRQ}}$  input. This input has the lowest priority of the three hardware interrupts.

**1.11.11 MEMORY READ (MRDY) (MC6809).** This input allows extension of the E and Q clocks to allow a longer data access time. A low on this input allows extension of the E and Q clocks (E high and Q low) in integral multiples of quarter bus cycles (up to 10 cycles) to allow interface with slow memory devices.

Memory ready does not extend the E and Q clocks during non-valid memory access cycles and therefore the processor does not slow down for “don’t care” bus accesses. Memory ready may also be used to extend the E and Q clocks when an external device is using the halt and direct memory access/bus request inputs.

**1.11.12 ADVANCED VALID MEMORY ADDRESS (AVMA) (MC6809E).** This output signal indicates that the MC6809E will use the bus in the following bus cycle. This output is low when the MC6809E is in either a halt or sync state.

**1.11.13 HALT.** This input is used to halt the processor. A low input halts the processor at the end of the present instruction execution cycle and the processor remains halted indefinitely without loss of data.

When the processor is halted, the BA output is high to indicate that the buses are in the high-impedance state and the BS output is also high to indicate that the processor is in the halt/bus grant state.

During the halt/bus grant state, the processor will not respond to external real-time requests such as  $\overline{\text{FIRQ}}$  or  $\overline{\text{IRQ}}$ . However, a direct memory access/bus request input will be accepted. A non-maskable interrupt or a reset input will be latched for processing later. The E and Q clocks continue to run during the halt/bus grant state.

**1.11.14 DIRECT MEMORY ACCESS/BUS REQUEST ( $\overline{\text{DMA/BREQ}}$ ) (MC6809).** This input is used to suspend program execution and make the buses available for another use such as a direct memory access or a dynamic memory refresh.

A low level on this input occurring during the Q clock high time suspends instruction execution at the end of the current cycle. The processor acknowledges acceptance of this input by setting the BA and BS outputs high to signify the bus grant state. The requesting device now has up to 15 bus cycles before the processor retrieves the bus for self-refresh.

Typically, a direct memory access controller will request to use the bus by setting the  $\overline{\text{DMA/BREQ}}$  input low when E goes high. When the processor acknowledges this input by setting the BA and BS outputs high, that cycle will be a dead cycle used to transfer bus mastership to the direct memory access controller. False memory access during any dead cycle should be prevented by externally developing a system DMAVMA signal which is low in any cycle when the BA output changes.

When the BA output goes low, either as a result of a direct memory access/bus request or a processor self-refresh, the direct memory access device should be removed from the bus. Another dead cycle will elapse before the processor accesses memory, to allow transfer of bus mastership without contention.

**1.11.15 BUSY (MC6809E).** This output indicates that bus re-arbitration should be deferred and provides the indivisible memory operation required for a “test-and-set” primitive.

This output will be high for the first two cycles of any Read-Modify-Write instruction, high during the first byte of a double-byte access, and high during the first byte of any indirect access or vector-fetch operation.

**1.11.16 POWER.** Two inputs are used to supply power to the processor:  $V_{CC}$  is +5.0  $\pm 5\%$ , while  $V_{SS}$  is ground or 0 volts.



## SECTION 2 ADDRESSING MODES

### 2.1 INTRODUCTION

This section contains a description of each of the addressing modes available on these processors.

### 2.2 ADDRESSING MODES

The addressing modes available on the MC6809 and MC6809E are: Inherent, Immediate, Extended, Direct, Indexed (with various offsets and autoincrementing/decrementing), and Branch Relative. Some of these addressing modes require an additional byte after the opcode to provide additional addressing interpretation. This byte is called a postbyte.

The following paragraphs provide a description of each addressing mode. In these descriptions the term effective address is used to indicate the address in memory from which the argument for an instruction is fetched or stored, or from which instruction processing is to proceed.

**2.2.1 INHERENT.** The information necessary to execute the instruction is contained in the opcode. Some operations specifying only the index registers or the accumulators, and no other arguments, are also included in this addressing mode.

Example:     **MUL**

**2.2.2 IMMEDIATE.** The operand is contained in one or two bytes immediately following the opcode. This addressing mode is used to provide constant data values that do not change during program execution. Both 8-bit and 16-bit operands are used depending on the size of the argument specified in the opcode.

Example:     **LDA #CR**  
              **LDB #7**  
              **LDA #\$F0**  
              **LDB #%1110000**  
              **LDX #\$8004**

Another form of immediate addressing uses a postbyte to determine the registers to be manipulated. The exchange (EXG) and transfer (TFR) instructions use the postbyte as shown in Figure 2-1(A). The push and pull instructions use the postbyte to designate the registers to be pushed or pulled as shown in Figure 2-1(B).

| b7          | b6              | b5 | b4 | b3               | b2              | b1 | b0 |
|-------------|-----------------|----|----|------------------|-----------------|----|----|
| SOURCE (R1) |                 |    |    | DESTINATION (R2) |                 |    |    |
| Code*       | Register        |    |    | Code*            | Register        |    |    |
| 0000        | D (A:B)         |    |    | 0101             | Program Counter |    |    |
| 0001        | X Index         |    |    | 1000             | A Accumulator   |    |    |
| 0010        | Y Index         |    |    | 1001             | B Accumulator   |    |    |
| 0011        | U Stack Pointer |    |    | 1010             | Condition Code  |    |    |
| 0100        | S Stack Pointer |    |    | 1011             | Direct Page     |    |    |

\*All other combinations of bits produce undefined results.

(A) Exchange (EXG) or Transfer (TFR) Instruction Postbyte

| b7 | b6  | b5 | b4 | b3 | b2 | b1 | b0 |
|----|-----|----|----|----|----|----|----|
| PC | S/U | Y  | X  | DP | B  | A  | CC |

PC = Program Counter  
S/U = Hardware/User Stack Pointer  
Y = Y Index Register  
X = U Index Register  
DP = Direct Page Register  
B = B Accumulator  
A = A Accumulator  
CC = Condition Code Register

(B) Push (PSH) or Pull (PUL) Instruction Postbyte

**Figure 2-1. Postbyte Usage for EXG/TFR, PSH/PUL Instructions**

**2.2.3 EXTENDED.** The effective address of the argument is contained in the two bytes following the opcode. Instructions using the extended addressing mode can reference arguments anywhere in the 64K addressing space. Extended addressing is generally not used in position independent programs because it supplies an absolute address.

Example: LDA >CAT

**2.2.4 DIRECT.** The effective address is developed by concatenation of the contents of the direct page register with the byte immediately following the opcode. The direct page register contents are the most-significant byte of the address. This allows accessing 256 locations within any one of 256 pages. Therefore, the entire addressing range is available for access using a single two-byte instruction.

Example: LDA >CAT

**2.2.5 INDEXED.** In these addressing modes, one of the pointer registers (X, Y, U, or S), and sometimes the program counter (PC) is used in the calculation of the effective address of the instruction operand. The basic types (and their variations) of indexed addressing available are shown in Table 2-1 along with the postbyte configuration used.

**2.2.5.1 Constant Offset from Register.** The contents of the register designated in the postbyte are added to a twos complement offset value to form the effective address of

the instruction operand. The contents of the designated register are not affected by this addition. The offset sizes available are:

- No offset — designated register contains the effective address
- 5-bit — 16 to +15
- 8-bit — 128 to +127
- 16-bit — 32768 to +32767

**Table 2-1. Postbyte Usage for Indexed Addressing Modes**

| Mode Type  | Variation            | Direct   | Indirect          |
|--|----------------------|----------|-------------------|
| Constant Offset from Register<br>(twos Complement Offset)    | No Offset            | 1RR00100 | 1RR10100          |
|  | 5-Bit Offset         | 0RRnnnnn | Defaults to 8-bit |
|  | 8-Bit Offset         | 1RR01000 | 1RR11000          |
|  | 16-Bit Offset        | 1RR01001 | 1RR11001          |
| Accumulator Offset from Register<br>(twos Complement Offset) | A Accumulator Offset | 1RR00110 | 1RR10110          |
|  | B Accumulator Offset | 1RR00101 | 1RR10101          |
|  | D Accumulator Offset | 1RR01011 | 1RR11011          |
| Auto Increment/Decrement from<br>Register                    | Increment by 1       | 1RR00000 | Not Allowed       |
|  | Increment by 2       | 1RR00001 | 1RR10001          |
|  | Decrement by 1       | 1RR00010 | Not Allowed       |
|  | Decrement by 2       | 1RR00011 | 1RR10011          |
| Constant Offset from Program<br>Counter                      | 8-Bit Offset         | 1XX01100 | 1XX11100          |
|  | 16-Bit Offset        | 1XX01101 | 1XX11101          |
| Extended Indirect  | 16-Bit Address       | -----    | 10011111          |

The 5-bit offset value is contained in the postbyte. The 8- and 16-bit offset values are contained in the byte or bytes immediately following the postbyte. If the Motorola assembler is used, it will automatically determine the most efficient offset; thus, the programmer need not be concerned about the offset size.

Examples: LDA ,X            LDY -64000,U  
           LDB 0,Y            LDA 17,PC  
           LDX 64,000,S    LDA There,PCR

**2.2.5.2 Accumulator Offset from Register.** The contents of the index or pointer register designed in the postbyte are temporarily added to the twos complement offset value contained in an accumulator (A, B, or D) also designated in the postbyte. Neither the designated register nor the accumulator contents are affected by this addition.

Example: LDA A,X            LDA D,U  
           LDA B,Y

**2.2.5.3 Autoincrement/Decrement from Register.** This addressing mode works in a postincrementing or predecrementing manner. The amount of increment or decrement, one or two positions, is designated in the postbyte.

In the autoincrement mode, the contents of the effective address contained in the pointer register, designated in the postbyte, and then the pointer register is automatically incremented; thus, the pointer register is postincremented.

In the autodecrement mode, the pointer register, designated in the postbyte, is automatically decremented first and then the contents of the new address are used; thus, the pointer register is predecremented.

|           |                      |                        |
|-----------|----------------------|------------------------|
| Examples: | <b>Autoincrement</b> | <b>Autodecrement</b>   |
|           | LDA ,X+    LDY ,X++  | LDA , -X    LDY , - -X |
|           | LDA ,Y+    LDX ,Y++  | LDA , -Y    LDX , - -Y |
|           | LDA ,S+    LDX ,U++  | LDA , -S    LDX , - -U |
|           | LDA ,U+    LDX ,S++  | LDA , -U    LDX , - -S |

**2.2.5.4 Indirection.** When using indirection, the effective address of the base indexed addressing mode is used to fetch two bytes which contain the final effective address of the operand. It can be used with all the indexed addressing modes and the program counter relative addressing mode.

**2.2.5.5 Extended Indirect.** The effective address of the argument is located at the address specified by the two bytes following the postbyte. The postbyte is used to indicate indirection.

Example:    LDA [\$F000]

**2.2.5.6 Program Counter Relative.** The program counter can also be used as a pointer with either an 8- or 16-bit signed constant offset. The offset value is added to the program counter to develop an effective address. Part of the postbyte is used to indicate whether the offset is 8 or 16 bits.

**2.2.6 BRANCH RELATIVE.** This addressing mode is used when branches from the current instruction location to some other location relative to the current program counter are desired. If the test condition of the branch instruction is true, then the effective address is calculated (program counter plus twos complement offset) and the branch is taken. If the test condition is false, the processor proceeds to the next in-line instruction. Note that the program counter is always pointing to the next instruction when the offset is added. Branch relative addressing is always used in position independent programs for all control transfers.

For short branches, the byte following the branch instruction opcode is treated as an 8-bit signed offset to be used to calculate the effective address of the next instruction if the branch is taken. This is called a short relative branch and the range is limited to plus 127 or minus 128 bytes from the following opcode.

For long branches, the two bytes after the opcode are used to calculate the effective address. This is called a long relative branch and the range is plus 32,767 or minus 32,768



bytes from the following opcode or the full 64K address space of memory that the processor can address at one time.

Examples:    Short Branch    Long Branch  
                  BRA POLE        LBRA CAT



## SECTION 3 INTERRUPT CAPABILITIES

### 3.1 INTRODUCTION

The MC6809 and MC6809E microprocessors have six vectored interrupts (three hardware and three software). The hardware interrupts are the non-maskable interrupt ( $\overline{\text{NMI}}$ ), the fast maskable interrupt request ( $\overline{\text{FIRQ}}$ ), and the normal maskable interrupt request ( $\overline{\text{IRQ}}$ ). The software interrupts consist of SWI, SWI2, and SWI3. When an interrupt request is acknowledged, all the processor registers are pushed onto the hardware stack, except in the case of  $\overline{\text{FIRQ}}$  where only the program counter and the condition code register is saved, and control is transferred to the address in the interrupt vector. The priority of these interrupts is, highest to lowest,  $\overline{\text{NMI}}$ , SWI,  $\overline{\text{FIRQ}}$ ,  $\overline{\text{IRQ}}$ , SWI2, and SWI3. Figure 3-1 is a detailed flowchart of interrupt processing in these processors. The interrupt vector locations are given in Table 3-1. The vector locations contain the address for the interrupt routine.

Additional information on the SWI, SWI2, and SWI3 interrupts is given in Appendix A. The hardware interrupts,  $\overline{\text{NMI}}$ ,  $\overline{\text{FIRQ}}$ , and  $\overline{\text{IRQ}}$  are listed alphabetically at the end of Appendix A.

**Table 3-1. Interrupt Vector Locations**

| Interrupt Description                               | Vector Location |         |
|---|-----------------|---------|
|   | MS Byte         | LS Byte |
| Reset (RESET)                                       | FFFE            | FFFF    |
| Non-Maskable Interrupt ( $\overline{\text{NMI}}$ )  | FFFC            | FFFD    |
| Software Interrupt (SWI)                            | FFFA            | FFFB    |
| Interrupt Request ( $\overline{\text{IRQ}}$ )       | FFF8            | FFF9    |
| Fast Interrupt Request ( $\overline{\text{FIRQ}}$ ) | FFF6            | FFF7    |
| Software Interrupt 2 (SWI2)                         | FFF4            | FFF5    |
| Software Interrupt 3 (SWI3)                         | FFF2            | FFF3    |
| Reserved  | FFF0            | FFF1    |

### 3.2 NON-MASKABLE INTERRUPT ( $\overline{\text{NMI}}$ )

The non-maskable interrupt is edge-sensitive in the sense that if it is sampled low one cycle after it has been sampled high, a non-maskable interrupt will be triggered. Because the non-maskable interrupt cannot be masked by execution of the non-maskable interrupt handler routine, it is possible to accept another non-maskable interrupt before executing the first instruction of the interrupt routine. A fatal error will exist if a non-maskable interrupt is repeatedly allowed to occur before completing the return from interrupt ( $\overline{\text{RTI}}$ ) instruction of the previous non-maskable interrupt request, since the stack

will eventually overflow. This interrupt is especially applicable to gaining immediate processor response for powerfail, software dynamic memory refresh, or other non-delayable events.

### **3.3 FAST MASKABLE INTERRUPT REQUEST ( $\overline{\text{FIRQ}}$ )**

A low level on the  $\overline{\text{FIRQ}}$  input with the F (fast interrupt request mask) bit in the condition code register clear triggers this interrupt sequence. The fast interrupt request provides fast interrupt response by stacking only the program counter and condition code register. This allows fast context switching with minimal overhead. If any registers are used by the interrupt routine then they can be saved by a single push instruction.

After accepting a fast interrupt request, the processor clears the E flag, saves the program counter and condition code register, and then sets both the I and F bits to mask any further IRQ and  $\overline{\text{FIRQ}}$  interrupts. After servicing the original interrupt, the user may selectively clear the I and F bits to allow multiple-level interrupts if so desired.

### **3.4 NORMAL MASKABLE INTERRUPT REQUEST ( $\overline{\text{IRQ}}$ )**

A low level on the  $\overline{\text{IRQ}}$  input with the I (interrupt request mask) bit in the condition code register clear triggers this interrupt sequence. The normal maskable interrupt request provides a slower hardware response to interrupts because it causes the entire machine state to be stacked. However, this means that interrupting software routines can use all processor resources without fear of damaging the interrupted routine. A normal interrupt request, having lower priority than the fast interrupt request, is prevented from interrupting the fast interrupt handler by the automatic setting of the I bit by the fast interrupt request handler.

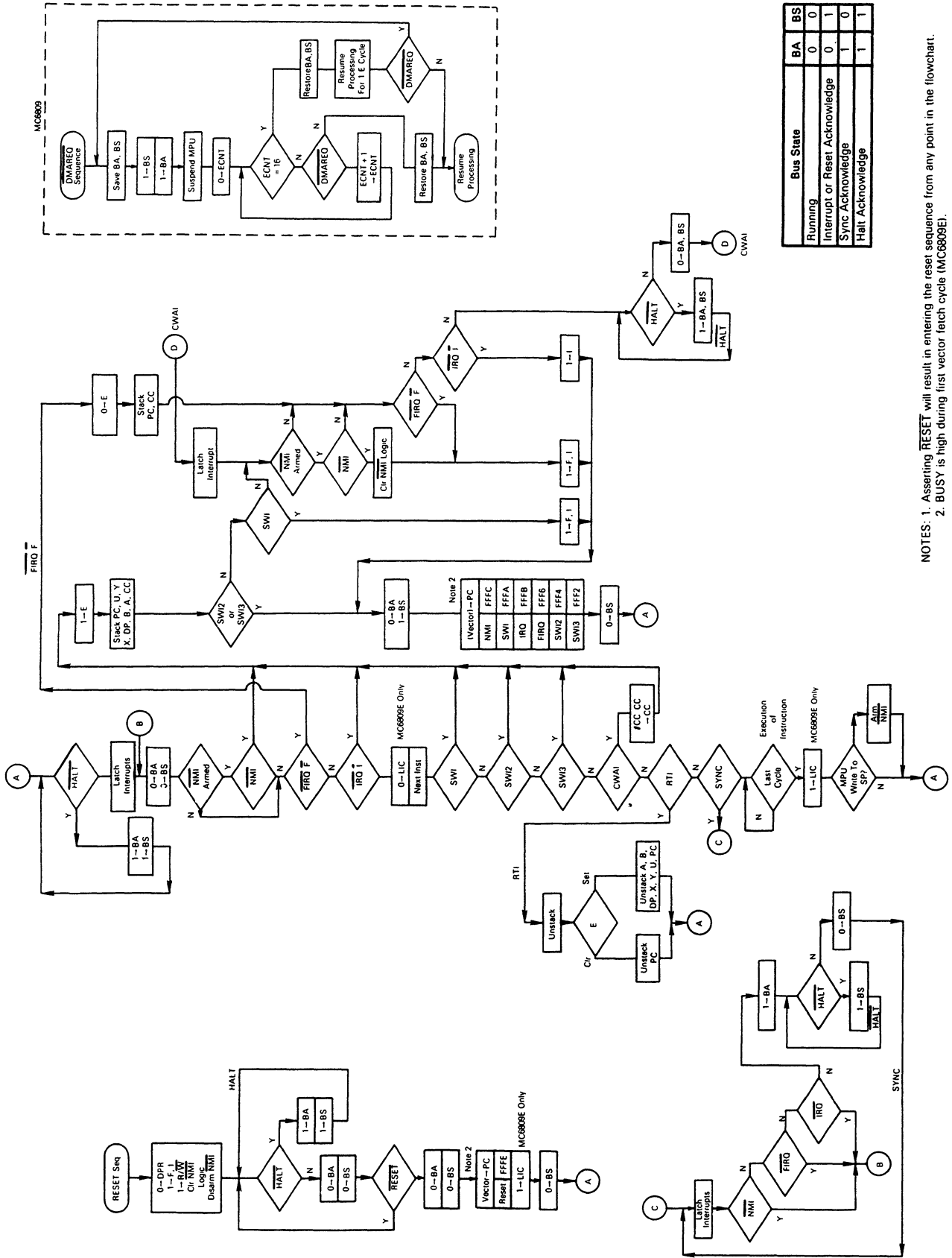
After accepting a normal interrupt request, the processor sets the E flag, saves the entire machine state, and then sets the I bit to mask any further interrupt request inputs. After servicing the original interrupt, the user may clear the I bit to allow multiple-level normal interrupts.

All interrupt handling routines should return to the formerly executing tasks using a return from interrupt ( $\overline{\text{RTI}}$ ) instruction. This instruction recovers the saved machine state from the hardware stack and control is returned to the interrupted program. If the recovered E bit is clear, it indicates that a fast interrupt request occurred and only the program counter address and condition code register are to be recovered.

### **3.5 SOFTWARE INTERRUPTS (SWI, SWI2, SWI3)**

The software interrupts cause the processor to go through the normal interrupt request sequence of stacking the complete machine state even though the interrupting source is the processor itself. These interrupts are commonly used for program debugging and for calls to an operating system.

Normal processing of the SWI input sets the I and F bits to prevent either of these interrupt requests from affecting the completion of a software interrupt request. The remaining software interrupt request inputs (SWI2 and SWI3) do not have the priority of the SWI input and therefore do not mask the two hardware interrupt request inputs ( $\overline{\text{FIRQ}}$  and  $\overline{\text{IRQ}}$ ).



NOTES: 1. Asserting **RESET** will result in entering the reset sequence from any point in the flowchart.  
 2. **BUSY** is high during first vector fetch cycle (MC68009E).

**Figure 3-1. Interrupt Processing Flowchart**

## SECTION 4 PROGRAMMING

### 4.1 INTRODUCTION

These processors are designed to be source-code compatible with the M6800 to make use of the substantial existing base of M6800 software and training. However, this asset should not overshadow the capabilities built into these processors that allow more modern programming techniques such as position-independence, modular programming, and reentrancy/recursion to be used on a microprocessor-based system. A brief review of these methods is given in the following paragraphs.

**4.1.1 POSITION INDEPENDENCE.** A program is said to be “position-independent” if it will run correctly when the same machine code is positioned arbitrarily in memory. Such a program is useful in many different hardware configurations, and might be copied from a disk into RAM when the operating system first sees a request to use a system utility. Position-independent programs never use absolute (extended or direct) addressing; instead, inherent immediate, register, indexed and relative modes are used. In particular, there should be no jump (absolute) or jump to subroutine instructions nor should absolute addresses be used. A position-independent program is almost always preferable to a position-dependent program (although position-independent code is usually 5 to 10% slower than normal code).

**4.1.2 MODULAR PROGRAMMING.** Modular programming is another indication of quality code. A module is a program element which can be easily disconnected from the rest of the program either for re-use in a new environment or for replacement. A module is usually a subroutine (although a subroutine is not necessarily a module); frequently, the programmer isolates register changes internal to the module by pushing these registers onto the stack upon entry, and pulling them off the stack before the return. Isolating register changes in the called module, to that module alone, allows the code in the calling program to be more easily analyzed since it can be assumed that all registers (except those specifically used for parameter transfer are unchanged by each called module. This leaves the processor’s registers free at each level for loop counts, address comparisons, etc.

**4.1.2.1 Local Storage.** A clean method for allocating “local” storage is required both by position-independent programs as well as modular programs. Local or temporary storage is used to hold values only during execution of a module (or called modules) and is released upon return. One way to allocate local storage is to decrement the hardware stack

pointer(s) by the number of bytes needed. Interrupts will then leave this area intact and it can be de-allocated on exiting the module. A module will almost always need more temporary storage than just the MPU registers.

**4.1.2.2 Global Storage.** Even in a modular environment there may be a need for “global” values which are accessible by many modules within a given system. These provide a convenient means for storing values from one invocation to another invocation of the same routine. Global storage may be created as local storage at some level, and a pointer register (usually U) used to point at this area. This register is passed unchanged in all subroutines, and may be used to index into the global area.

**4.1.3 REENTRANCY/RECURSION.** Many programs will eventually involve execution in an interrupt-driven environment. If the interrupt handlers are complex, they might well call the same routine which has just been interrupted. Therefore, to protect present programs against certain obsolescence, all programs should be written to be reentrant. A reentrant routine allocates different local variable storage upon each entry. Thus, a later entry does not destroy the processing associated with an earlier entry.

The same technique which was implemented to allow reentrancy also allows recursion. A recursive routine is defined as a routine that calls itself. A recursive routine might be written to simplify the solution of certain types of problems, especially those which have a data structure whose elements may themselves be a structure. For example, a parenthetical equation represents a case where the expression in parenthesis may be considered to be a value which is operated on by the rest of the equation. A programmer might choose to write an expression evaluator passing the parenthetical expression (which might also contain parenthetical expressions) in the call, and receive back the returned value of the expression within the parenthesis.

## **4.2 M6809 CAPABILITIES**

The following paragraphs briefly explain how the MC6809 is used with the programming techniques mentioned earlier.

**4.2.1 MODULE CONSTRUCTION.** A module can be defined as a logically self-contained and discrete part of a larger program. A properly constructed module accepts well defined inputs, carries out a set of processing actions, and produces a specified output. The use of parameters, local storage, and global storage by a program module is given in the following paragraphs. Since registers will be used inside the module (essentially a form of local storage), the first thing that is usually done at entry to a module is to push (save) them on to the stack. This can be done with one instruction (e.g., PSHS Y, X, B, A). After the body of the module is executed, the saved registers are collected, and a subroutine return is performed, at one time, by pulling the program counter from the stack (e.g., PULS A,B,X,Y,PC).



**4.2.1.1 Parameters.** Parameters may be passed to or from modules either in registers, if they will provide sufficient storage for parameter passage, or on the stack. If parameters are passed on the stack, they are placed there before calling the lower level module. The called module is then written to use local storage inside the stack as needed (e.g., ADDA offset,S). Notice that the required offset consists of the number of bytes pushed (upon entry), plus two from the stacked return address, plus the data offset at the time of the call. This value may be calculated, by hand, by drawing a “stack picture” diagram representing module entry, and assigning convenient mnemonics to these offsets with the assembler. Returned parameters replace those sent to the routine. If more parameters are to be returned on the stack than would normally be sent, space for their return is allocated by the calling routine before the actual call (if four additional bytes are to be returned, the caller would execute LEAS – 4,S to acquire the additional storage).

**4.2.1.2 Local Storage.** Local storage space is acquired from the stack while the present routine is executing and then returned to the stack prior to exit. The act of pushing registers which will be used in later calculations essentially saves those registers in temporary local storage. Additional local storage can easily be acquired from the stack e.g., executing LEAS – 2048,S acquires a buffer area running from the 0,S to 2047,S inclusive. Any byte in this area may be accessed directly by any instruction which has an indexed addressing mode. At the end of the routine, the area acquired for local storage is released (e.g., LEAS 2048,S) prior to the final pull. For cleaner programs, local storage should be allocated at entry to the module and released at the exit of the module.

**4.2.1.3 Global Storage.** The area required for global storage is also most effectively acquired from the stack, probably by the highest level routine in the standard package. Although this is local storage to the highest level routine, it becomes “global” by positioning a register to point at this storage, (sometimes referred to as a stack mark) then establishing the convention that all modules pass that same pointer value when calling lower level modules. In practice, it is convenient to leave this stack mark register unchanged in all modules, especially if global accesses are common. The highest level routine in the standard package would execute the following sequence upon entry (to initialize the global area):

```

PSHS   U       higher level mark, if any
TFR    S,U     new stack mark
LEAS   – 17,U  allocate global storage

```

Note that the U register now defines 17-bytes of locally allocated (permanent) globals (which are – 1,U through – 17,U) as well as other external globals (2,U and above) which have been passed on the stack by the routine which called the standard package. Any global may be accessed by any module using exactly the same offset value at any level (e.g., ROL, RAT,U; where RAT EQU – 11 has been defined). Furthermore, the values stacked prior to invoking the standard package may include pointers to data or I/O peripherals. Any indexed operation may be performed indexed indirect through those pointers, which means, for example, that the module need know nothing about the actual hardware configuration, except that (upon entry) the pointer to an I/O register has been placed at a given location on the stack.



Here we wish to point at a message to be printed from the body of the program. By writing "MSG1, PCR" we signal the assembler to compute the distance between the present address (the address of the LBSR) and MSG1. This result is inserted as a constant into the LEA instruction which will be indexed from the program counter value at the time of execution. Now, no matter where the code is located, when it is executed the computer offset from the program counter will point at MSG1. This code is position-independent.

It is common to use space in the hardware stack for temporary storage. Space is made for temporary variables from 0,S through TEMP-1, S by decrementing the stack pointer equal to the length of required storage. We could use:

LEAS            - TEMP,S.

Not only does this facilitate position-independent code but it is structured and helps reentrancy and recursion.

**4.2.3 REENTRANT PROGRAMS.** A program that can be executed by several different users sharing the same copy of it in memory is called reentrant. This is important for interrupt driven systems. This method saves considerable memory space, especially with large interrupt routines. Stacks are required for reentrant programs, and the M6809 can support up to four stacks by using the X and Y index registers as stack pointers.

Stacks are simple and convenient mechanisms for generating reentrant programs. Subroutines which use stacks for passing parameters and results can be easily made to be reentrant. Stack accesses use the indexed addressing mode for fast, efficient execution. Stack addressing is quick.

Pure code, or code that is not self-modifying, is mandatory to produce reentrant code. No internal information within the code is subject to modification. Reentrant code never has internal temporary storage, is simpler to debug, can be placed in ROM, and must be interruptable.

**4.2.4 RECURSIVE PROGRAMS.** A recursive program is one that can call itself. They are quite convenient for parsing mechanisms and certain arithmetic functions such as computing factorials. As with reentrant programming, stacks are very useful for this technique.

**4.2.5 LOOPS.** The usual structured loops (i.e., REPEAT...UNTIL, WHILE...DO, FOR..., etc.) are available in assembly language in exactly the same way a high-level language compiler could translate the construct for execution on the target machine. Using a FOR...NEXT loop as an example, it is possible to push the loop count, increment value, and termination value on the stack as variables local to that loop. On each pass through the loop, the working register is saved, the loop count picked up, the increment added in, and the result compared to the termination value. Based on this comparison, the loop counter might be updated, the working register recovered and the loop resumed, or the working register recovered and the loop variables de-allocated. Reasonable macros

could make the source form for loop trivial, even in assembly language. Such macros might reduce errors resulting from the use of multiple instructions simply to implement a standard control structure.

**4.2.6 STACK PROGRAMMING.** Many microprocessor applications require data stored as contiguous pieces of information in memory. The data may be temporary, that is, subject to change or it may be permanent. Temporary data will most likely be stored in RAM. Permanent data will most likely be stored in ROM.

It is important to allow the main program as well as subroutines access to this block of data, especially if arguments are to be passed from the main program to the subroutines and vice versa.

**4.2.6.1 M6809 Stacking Operations.** Stack pointers are markers which point to the stack and its internal contents. Although all four index registers may be used as stack registers, the S (hardware stack pointer) and the U (user stack pointer) are generally preferred because the push and pull instructions apply to these registers. Both are 16-bit indexable registers. The processor uses the S register automatically during interrupts and subroutine calls. The U register is free for any purpose needed. It is not affected by interrupts or subroutine calls implemented by the hardware.

Either stack pointer can be specified as the base address in indexed addressing. One use of the indirect addressing mode uses stack pointers to allow addresses of data to be passed to a subroutine on a stack as arguments to a subroutine. The subroutine can now reference the data with one instruction. High-level language calls that pass arguments by reference are now more efficiently coded. Also, each stack push or pull operation in a program uses a postbyte which specifies any register or set of registers to be pushed or pulled from either stack. With this option, the overhead associated with subroutine calls in both assembly and high-level language programs is greatly decreased. In fact, with the large number of instructions that use autoincrement and autodecrement, the M6809 can emulate a true stack computer architecture.

Using the S or U stack pointer, the order in which the registers are pushed or pulled is shown in Figure 4-1. Notice that we push "onto" the stack towards decreasing memory locations. The program counter is pushed first. Then the stack pointer is decremented and the "other" stack pointer is pushed onto the stack. Decrementing and storing continues until all the registers requested by the postbyte are pushed onto the stack. The stack pointer points to the top of the stack after the push operation.

The stacking order is specified by the processor. The stacking order is identical to the order used for all hardware and software interrupts. The same order is used even if a subset of the registers is pushed.

Without stacks, most modern block-structured high-level languages would be cumbersome to implement. Subroutine linkage is very important in high-level language generation. Paragraph 4.2.6.2 describes how to use a stack mark pointer for this important task.

Good programming practice dictates the use of the hardware stack for temporary storage. To reserve space, decrement the stack pointer by the amount of storage required with the instruction LEAS - TEMPS, S. This instruction makes space for temporary variables from 0,S through TEMPS - 1,S.

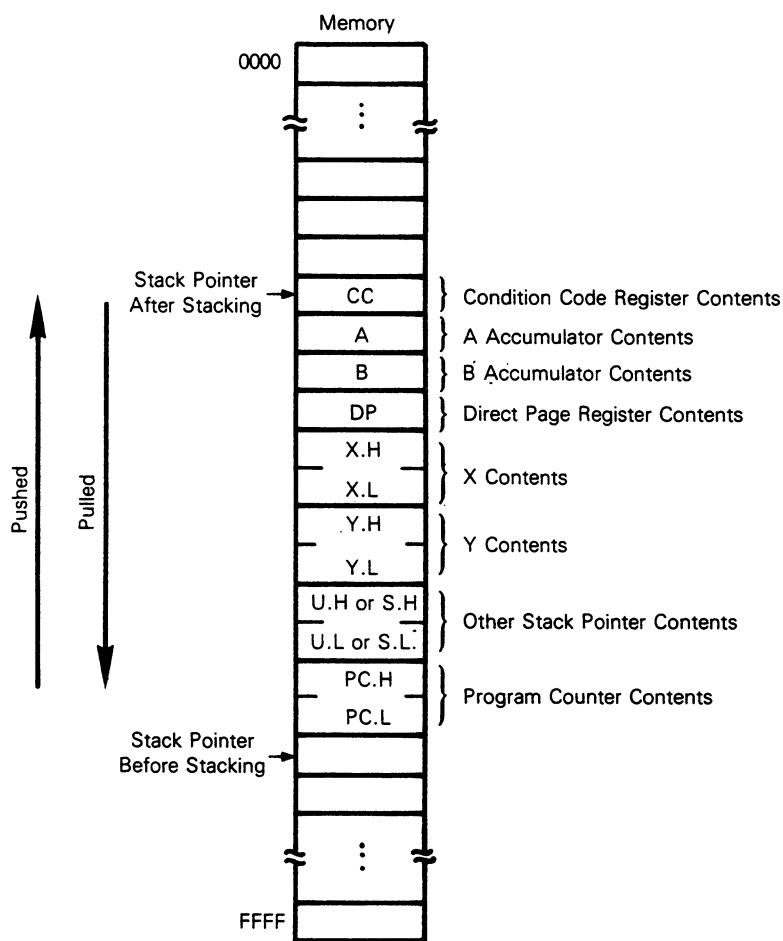


Figure 4-1. Stacking Order

**4.2.6.2 Subroutine Linkage.** In the highest level routine, global variables are sometimes considered to be local. Therefore, global storage is allocated at this point, but access to these same variables requires different offset values depending on subroutine depth. Because subroutine depth changes dynamically, the length may not be known beforehand. This problem is solved by assigning one pointer (U will be used in the following description, but X or Y could also be used) to "mark" a location on the hardware stack by using the instruction TFR S,U. If the programmer does this immediately prior to allocating global storage, then all variables will then be available at a constant negative offset location from this stack mark. If the stack is marked after the global variables are

allocated, then the global variables are available at a constant positive offset from U. Register U is then called the stack mark pointer. Recall that the hardware stack pointer may be modified by hardware interrupts. For this reason, it is fatal to use data referred to by a negative offset with respect to the hardware stack pointer, S.

**4.2.6.3 Software Stacks.** If more than two stacks are needed, autoincrement and autodecrement mode of addressing can be used to generate additional software stack pointers.

The X, Y, and U index registers are quite useful in loops for incrementing and decrementing purposes. The pointer is used for searching tables and also to move data from one area of memory to another (block moves). This autoincrement and autodecrement feature is available in the indexed addressing mode of the M6809 to facilitate such operations.

In autoincrement, the value contained in the index register (X or Y, U or S) is used as the effective address and then the register is incremented (postincremented). In autodecrement, the index register is first decremented and then used to obtain the effective address (predecremented). Postincrement or predecrement is always performed in this addressing mode. This is equivalent in operation to the push and pull from a stack. This equivalence allows the X and Y index registers to be used as software stack pointers. The indexed addressing mode can also implement an extra level of post indirection. This feature supports parameter and pointer operations.

**4.2.7 REAL TIME PROGRAMMING.** Real time programming requires special care. Sometimes a peripheral or task demands an immediate response from the processor, other times it can wait. Most real time applications are demanding in terms of processor response.

A common solution is to use the interrupt capability of the processor in solving real time problems. Interrupts mean just that; they request a break in the current sequence of events to solve an asynchronous service request. The system designer must consider all variations of the conditions to be encountered by the system including software interaction with interrupts. As a result, problems due to software design are more common in interrupt implementation code for real time programming than most other situations. Software timeouts, hardware interrupts, and program control interrupts are typically used in solving real time programming problems.

### **4.3 PROGRAM DOCUMENTATION**

Common sense dictates that a well documented program is mandatory. Comments are needed to explain each group of instructions since their use is not always obvious from looking at the code. Program boundaries and branch instructions need full clarification. Consider the following points when writing comments: up-to-date, accuracy, completeness, conciseness, and understandability.

Accurate documentation enables you and others to maintain and adapt programs for updating and/or additional use with other programs.

The following program documentation standards are suggested.

- A) Each subroutine should have an associated header block containing at least the following elements:
  - 1) A full specification for this subroutine — including associated data structures — such that replacement code could be generated from this description alone.
  - 2) All usage of memory resources must be defined, including:
    - a) All RAM needed from temporary (local) storage used during execution of this subroutine or called subroutines.
    - b) All RAM needed for permanent storage (used to transfer values from one execution of the subroutine to future executions).
    - c) All RAM accessed as global storage (used to transfer values from or to higher-level subroutines).
    - d) All possible exit-state conditions, if these are to be used by calling routines to test occurrences internal to the subroutine.
- B) Code internal to each subroutine should have sufficient associated line comments to help in understanding the code.
- C) All code must be non-self-modifying and position-independent.
- D) Each subroutine which includes a loop must be separately documented by a flowchart or pseudo high-level language algorithm.
- E) Any module or subroutine should be executable starting at the first location and exit at the last location.

#### 4.4 INSTRUCTION SET

The complete instruction set for the M6809 is given in Table 4-1.

**Table 4-1. Instruction Set**

| <b>Instruction</b> | <b>Description</b>                      |
|--------------------|---|
| ABX                | Add Accumulator B into Index Register X |
| ADC                | Add with Carry into Register            |
| ADD                | Add Memory into Register                |
| AND                | Logical AND Memory into Register        |
| ASL                | Arithmetic Shift Left                   |
| ASR                | Arithmetic Shift Right                  |
| BCC                | Branch on Carry Clear                   |
| BCS                | Branch on Carry Set                     |
| BEQ                | Branch on Equal                         |
| BGE                | Branch on Greater Than or Equal to Zero |
| BGT                | Branch on Greater                       |
| BHI                | Branch if Higher                        |
| BHS                | Branch if Higher or Same                |
| BIT                | Bit Test                                |
| BLE                | Branch if Less than or Equal to Zero    |

**Table 4-1. Instruction Set (Continued)**

| <b>Instruction</b> | <b>Description</b>                   |
|--------------------|--------------------------------------|
| BLO                | Branch on Lower                      |
| BLS                | Branch on Lower or Same              |
| BLT                | Branch on Less than Zero             |
| BMI                | Branch on Minus                      |
| BNE                | Branch Not Equal                     |
| BPL                | Branch on Plus                       |
| BRA                | Branch Always                        |
| BRN                | Branch Never                         |
| BSR                | Branch to Subroutine                 |
| BVC                | Branch on Overflow Clear             |
| BVS                | Branch on Overflow Set               |
| CLR                | Clear                                |
| CMP                | Compare Memory from a Register       |
| COM                | Complement                           |
| CWAI               | Clear CC bits and Wait for Interrupt |
| DAA                | Decimal Addition Adjust              |
| DEC                | Decrement                            |
| EOR                | Exclusive OR                         |
| EXG                | Exchange Registers                   |
| INC                | Increment                            |
| JMP                | Jump                                 |
| JSR                | Jump to Subroutine                   |
| LD                 | Load Register from Memory            |
| LEA                | Load Effective Address               |
| LSL                | Logical Shift Left                   |
| LSR                | Logical Shift Right                  |
| MUL                | Multiply                             |
| NEG                | Negate                               |
| NOP                | No Operation                         |
| OR                 | Inclusive OR Memory into Register    |
| PSH                | Push Registers                       |
| PUL                | Pull Registers                       |
| ROL                | Rotate Left                          |
| ROR                | Rotate Right                         |
| RTI                | Return from Interrupt                |
| RTS                | Return from Subroutine               |
| SBC                | Subtract with Borrow                 |
| SEX                | Sign Extend                          |
| ST                 | Store Register into Memory           |
| SUB                | Subtract Memory from Register        |
| SWI                | Software Interrupt                   |
| SYNC               | Synchronize to External Event        |
| TFR                | Transfer Register to Register        |
| TST                | Test                                 |



The instruction set can be functionally divided into five categories. They are:

8-Bit Accumulator and Memory Instructions

16-Bit Accumulator and Memory Instructions

Index Register/Stack Pointer Instructions

Branch Instructions

Miscellaneous Instructions

Tables 4-2 through 4-6 are listings of the M6809 instructions and their variations grouped into the five categories listed.

**Table 4-2. 8-Bit Accumulator and Memory Instructions**

| Instruction     | Description  |
|-----------------|--|
| ADCA, ADCB      | Add memory to accumulator with carry               |
| ADDA, ADDB      | Add memory to accumulator                          |
| ANDA, ANDB      | And memory with accumulator                        |
| ASL, ASLA, ASLB | Arithmetic shift of accumulator or memory left     |
| ASR, ASRA, ASRB | Arithmetic shift of accumulator or memory right    |
| BITA, BITB      | Bit test memory with accumulator                   |
| CLR, CLRA, CLRB | Clear accumulator or memory location               |
| CMPA, CMPB      | Compare memory from accumulator                    |
| COM, COMA, COMB | Complement accumulator or memory location          |
| DAA             | Decimal adjust A accumulator                       |
| DEC, DECA, DECB | Decrement accumulator or memory location           |
| EORA, EORB      | Exclusive or memory with accumulator               |
| EXG R1, R2      | Exchange R1 with R2 (R1, R2 = A, B, CC, DP)        |
| INC, INCA, INCB | Increment accumulator or memory location           |
| LDA, LDB        | Load accumulator from memory                       |
| LSL, LSLA, LSLB | Logical shift left accumulator or memory location  |
| LSR, LSRA, LSRB | Logical shift right accumulator or memory location |
| MUL             | Unsigned multiply ( $A \times B \rightarrow D$ )   |
| NEG, NEGA, NEGB | Negate accumulator or memory                       |
| ORA, ORB        | Or memory with accumulator                         |
| ROL, ROLA, ROLB | Rotate accumulator or memory left                  |
| ROR, RORA, RORB | Rotate accumulator or memory right                 |
| SBCA, SBCB      | Subtract memory from accumulator with borrow       |
| STA, STB        | Store accumulator to memroy                        |
| SUBA, SUBB      | Subtract memory from accumulator                   |
| TST, TSTA, TSTB | Test accumulator or memory location                |
| TFR R1, R2      | Transfer R1 to R2 (R1, R2 = A, B, CC, DP)          |

NOTE: A, B, CC, or DP may be pushed to (pulled from) either stack with PSHS, PSHU (PULS, PULU) instructions.

**Table 4-3. 16-Bit Accumulator and Memory Instructions**

| Instruction | Description                                  |
|-------------|--|
| ADDD        | Add memory to D accumulator                  |
| CMPD        | Compare memory from D accumulator            |
| EXG D, R    | Exchange D with X, Y, S, U, or PC            |
| LDD         | Load D accumulator from memory               |
| SEX         | Sign Extend B accumulator into A accumulator |
| STD         | Store D accumulator to memory                |
| SUBD        | Subtract memory from D accumulator           |
| TFR D, R    | Transfer D to X, Y, S, U, or PC              |
| TFR R, D    | Transfer X, Y, S, U, or PC to D              |

NOTE: D may be pushed (pulled) to either stack with PSHS, PSHU (PULS, PULU) instructions.

**Table 4-4. Index/Stack Pointer Instructions**

| Instruction | Description  |
|-------------|--|
| CMPS, CMPU  | Compare memory from stack pointer                        |
| CMPX, CMPY  | Compare memory from index register                       |
| EXG R1, R2  | Exchange D, X, Y, S, U or PC with D, X, Y, S, U or PC    |
| LEAS, LEAU  | Load effective address into stack pointer                |
| LEAX, LEAY  | Load effective address into index register               |
| LDS, LDU    | Load stack pointer from memory                           |
| LDX, LDY    | Load index register from memory                          |
| PSHS        | Push A, B, CC, DP, D, X, Y, U, or PC onto hardware stack |
| PSHU        | Push A, B, CC, DP, D, X, Y, X, or PC onto user stack     |
| PULS        | Pull A, B, CC, DP, D, X, Y, U, or PC from hardware stack |
| PULU        | Pull A, B, CC, DP, D, X, Y, S, or PG from hardware stack |
| STS, STU    | Store stack pointer to memory                            |
| STX, STY    | Store index register to memory                           |
| TFR R1, R2  | Transfer D, X, Y, S, U, or PC to D, X, Y, S, U, or PC    |
| ABX         | Add B accumulator to X (unsigned)                        |

**Table 4-5. Branch Instructions**

| Instruction              | Description                              |
|--------------------------|--|
| <b>SIMPLE BRANCHES</b>   |  |
| BEQ, LBEQ                | Branch if equal                          |
| BNE, LBNE                | Branch if not equal                      |
| BMI, LBMI                | Branch if minus                          |
| BPL, LBPL                | Branch if plus                           |
| BCS, LBCS                | Branch if carry set                      |
| BCC, LBCC                | Branch if carry clear                    |
| BVS, LBVS                | Branch if overflow set                   |
| BVC, LBVC                | Branch if overflow clear                 |
| <b>SIGNED BRANCHES</b>   |  |
| BGT, LBGT                | Branch if greater (signed)               |
| BVS, LBVS                | Branch if invalid twos complement result |
| BGE, LBGE                | Branch if greater than or equal (signed) |
| BEQ, LBEQ                | Branch if equal                          |
| BNE, LBNE                | Branch if not equal                      |
| BLE, LBLE                | Branch if less than or equal (signed)    |
| BVC, LBVC                | Branch if valid twos complement result   |
| BLT, LBLT                | Branch if less than (signed)             |
| <b>UNSIGNED BRANCHES</b> |  |
| BHI, LBHI                | Branch if higher (unsigned)              |
| BCC, LBCC                | Branch if higher or same (unsigned)      |
| BHS, LBHS                | Branch if higher or same (unsigned)      |
| BEQ, LBEQ                | Branch if equal                          |
| BNE, LBNE                | Branch if not equal                      |
| BLS, LBLS                | Branch if lower or same (unsigned)       |
| BCS, LBCS                | Branch if lower (unsigned)               |
| BLO, LBLO                | Branch if lower (unsigned)               |
| <b>OTHER BRANCHES</b>    |  |
| BSR, LBSR                | Branch to subroutine                     |
| BRA, LBRA                | Branch always                            |
| BRN, LBRN                | Branch never                             |

**Table 4-6. Miscellaneous Instructions**

| Instruction     | Description  |
|-----------------|--|
| ANDCC           | AND condition code register                          |
| CWAI            | AND condition code register, then wait for interrupt |
| NOP             | No operation   |
| ORCC            | OR condition code register                           |
| JMP             | Jump   |
| JSR             | Jump to subroutine                                   |
| RTI             | Return from interrupt                                |
| RTS             | Return from subroutine                               |
| SWI, SWI2, SWI3 | Software interrupt (absolute indirect)               |
| SYNC            | Synchronize with interrupt line                      |



# APPENDIX A

## INSTRUCTION SET DETAILS

### A.1 INTRODUCTION

This appendix contains detailed information about each instruction in the MC6809 instruction set. They are arranged in an alphabetical order with the mnemonic heading set in larger type for easy reference.

### A.2 NOTATION

In the operation description for each instruction, symbols are used to indicate the operation. Table A-1 lists these symbols and their meanings. Abbreviations for the various registers, bits, and bytes are also used. Table A-2 lists these abbreviations and their meanings.

**Table A-1. Operation Notation**

| <u>Symbol</u> | <u>Meaning</u>       |
|---------------|----------------------|
| ←             | Is transferred to    |
| ∧             | Boolean AND          |
| ∨             | Boolean OR           |
| ⊕             | Boolean exclusive OR |
| ¯ (Overline)  | Boolean NOT          |
| :             | Concatenation        |
| +             | Arithmetic plus      |
| -             | Arithmetic minus     |
| X             | Arithmetic multiply  |

**Table A-2. Register Notation**

| <u>Abbreviation</u> | <u>Meaning</u>   |
|---------------------|--|
| ACCA or A           | Accumulator A  |
| ACCB or B           | Accumulator B  |
| ACCA:ACCB or D      | Double accumulator D   |
| ACCX                | Either accumulator A or B  |
| CCR or CC           | Condition code register  |
| DPR or DP           | Direct page register   |
| EA                  | Effective address  |
| IFF                 | If and only if   |
| IX or X             | Index register X   |
| IY or Y             | Index register Y   |
| LSN                 | Least significant nibble   |
| M                   | Memory location  |
| MI                  | Memory immediate   |
| MSN                 | Most significant nibble  |
| PC                  | Program counter  |
| R                   | A register before the operation  |
| R'                  | A register after the operation   |
| TEMP                | Temporary storage location   |
| xxH                 | Most significant byte of any 16-bit register                                     |
| xxL                 | Least significant byte of any 16-bit register                                    |
| Sp or S             | Hardware Stack pointer   |
| Us or U             | User Stack pointer   |
| P                   | A memory argument with Immediate, Direct, Extended, and Indexed addressing modes |
| Q                   | A read-modify-write argument with Direct, Indexed, and Extended addressing modes |
| ( )                 | The data pointed to by the enclosed (16-bit address)                             |
| dd                  | 8-bit branch offset  |
| DDDD                | 16-bit branch offset   |
| #                   | Immediate value follows  |
| \$                  | Hexadecimal value follows  |
| [ ]                 | Indirection  |
| '                   | Indicates indexed addressing   |

# ABX

**Add Accumulator B into Index Register X**

# ABX

**Source Form:** ABX

**Operation:**  $IX' \leftarrow IX + ACCB$

**Condition Codes:** Not affected.

**Description:** Add the 8-bit unsigned value in accumulator B into index register X.

**Addressing Mode:** Inherent

# ADC

## Add with Carry into Register

# ADC

**Source Forms:** ADCA P; ADCB P

**Operation:**  $R' \leftarrow R + M + C$

**Condition Codes:** H — Set if a half-carry is generated; cleared otherwise.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Set if an overflow is generated; cleared otherwise.  
C — Set if a carry is generated; cleared otherwise.

**Description:** Adds the contents of the C (carry) bit and the memory byte into an 8-bit accumulator.

**Addressing Modes:** Immediate  
Extended  
Direct  
Indexed



# ADD (8-Bit)

Add Memory into Register

# ADD (8-Bit)

**Source Forms:** ADDA P; ADDB P

**Operation:**  $R' \leftarrow R + M$

**Condition Codes:** H — Set if a half-carry is generated; cleared otherwise.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Set if an overflow is generated; cleared otherwise.  
C — Set if a carry is generated; cleared otherwise.

**Description:** Adds the memory byte into an 8-bit accumulator.

**Addressing Modes:** Immediate  
Extended  
Direct  
Indexed

# **ADD (16-Bit)**      Add Memory into Register      **ADD (16-Bit)**

**Source Forms:**      ADDD P

**Operation:**       $R' \leftarrow R + M:M + 1$

**Condition Codes:**    H — Not affected.  
                              N — Set if the result is negative; cleared otherwise.  
                              Z — Set if the result is zero; cleared otherwise.  
                              V — Set if an overflow is generated; cleared otherwise.  
                              C — Set if a carry is generated; cleared otherwise.

**Description:**      Adds the 16-bit memory value into the 16-bit accumulator

**Addressing Modes:** Immediate  
                                  Extended  
                                  Direct  
                                  Indexed

# AND

## Logical AND Memory into Register

# AND

**Source Forms:** ANDA P; ANDB P

**Operation:**  $R' \leftarrow R \Delta M$

**Condition Codes:** H — Not affected.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Always cleared.  
C — Not affected.

**Description:** Performs the logical AND operation between the contents of an accumulator and the contents of memory location M and the result is stored in the accumulator.

**Addressing Modes:** Immediate  
Extended  
Direct  
Indexed

# **AND** Logical AND Immediate Memory into Condition Code Register **AND**

**Source Form:** ANDCC #xx

**Operation:**  $R' \leftarrow R \wedge MI$

**Condition Codes:** Affected according to the operation.

**Description:** Performs a logical AND between the condition code register and the immediate byte specified in the instruction and places the result in the condition code register.

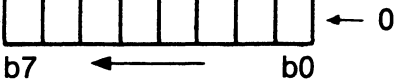
**Addressing Mode:** Immediate

# ASL

## Arithmetic Shift Left

# ASL

**Source Forms:** ASL Q; ASLA; ASLB

**Operation:** C ←  ← 0

**Condition Codes:**

- H — Undefined
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Loaded with the result of the exclusive OR of bits six and seven of the original operand.
- C — Loaded with bit seven of the original operand.

**Description:** Shifts all bits of the operand one place to the left. Bit zero is loaded with a zero. Bit seven is shifted into the C (carry) bit.

**Addressing Modes:** Inherent  
Extended  
Direct  
Indexed

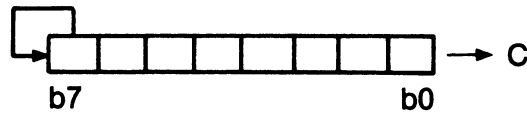
# ASR

## Arithmetic Shift Right

# ASR

**Source Forms:** ASR Q; ASRA; ASRB

**Operation:**



**Condition Codes:** H — Undefined.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Not affected.  
C — Loaded with bit zero of the original operand.

**Description:** Shifts all bits of the operand one place to the right. Bit seven is held constant. Bit zero is shifted into the C (carry) bit.

**Addressing Modes:** Inherent  
Extended  
Direct  
Indexed

# BCC

## Branch on Carry Clear

# BCC

**Source Forms:** BCC dd; LBCC DDDD

**Operation:** TEMP ← MI  
IFF C = 0 then PC' ← PC + TEMP

**Condition Codes:** Not affected.

**Description:** Tests the state of the C (carry) bit and causes a branch if it is clear.

**Addressing Mode:** Relative

**Comments:** Equivalent to BHS dd; LBHS DDDD

# BCS

## Branch on Carry Set

# BCS

**Source Forms:** BCS dd; LBCS DDDD

**Operation:** TEMP ← MI  
IFF C = 1 then PC' ← PC + TEMP

**Condition Codes:** Not affected.

**Description:** Tests the state of the C (carry) bit and causes a branch if it is set.

**Addressing Mode:** Relative

**Comments:** Equivalent to BLO dd; LBLO DDDD



# BEQ

## Branch on Equal

# BEQ

**Source Forms:** BEQ dd; LBEQ DDDD

**Operation:** TEMP ← MI  
IFF Z = 1 then PC' ← PC + TEMP

**Condition Codes:** Not affected.

**Description:** Tests the state of the Z (zero) bit and causes a branch if it is set. When used after a subtract or compare operation, this instruction will branch if the compared values, signed or unsigned, were exactly the same.

**Addressing Mode:** Relative

# BGE

## Branch on Greater than or Equal to Zero

# BGE

**Source Forms:** BGE dd; LBGE DDDD

**Operation:** TEMP ← MI  
IFF [N ⊕ V] = 0 then PC' ← PC + TEMP

**Condition Codes:** Not affected.

**Description:** Causes a branch if the N (negative) bit and the V (overflow) bit are either both set or both clear. That is, branch if the sign of a valid twos complement result is, or would be, positive. When used after a subtract or compare operation on twos complement values, this instruction will branch if the register was greater than or equal to the memory operand.

**Addressing Mode:** Relative

# BGT

## Branch on Greater

# BGT

**Source Forms:** BGT dd; LBGT DDDD

**Operation:**  $TEMP \leftarrow MI$   
IFF  $Z \wedge [N \oplus V] = 0$  then  $PC' \leftarrow PC + TEMP$

**Condition Codes:** Not affected.

**Description:** Causes a branch if the N (negative) bit and V (overflow) bit are either both set or both clear and the Z (zero) bit is clear. In other words, branch if the sign of a valid twos complement result is, or would be, positive and not zero. When used after a subtract or compare operation on twos complement values, this instruction will branch if the register was greater than the memory operand.

**Addressing Mode:** Relative

# BHI

## Branch if Higher

# BHI

**Source Forms:** BHI dd; LBHI DDDD

**Operation:** TEMP ← MI  
IFF [C v Z] = 0 then PC' ← PC + TEMP

**Condition Codes:** Not affected.

**Description:** Causes a branch if the previous operation caused neither a carry nor a zero result. When used after a subtract or compare operation on unsigned binary values, this instruction will branch if the register was higher than the memory operand.

**Addressing Mode:** Relative

**Comments:** Generally not useful after INC/DEC, LD/TST, and TST/CLR/COM instructions.

# BHS

## Branch if Higher or Same

# BHS

**Source Forms:** BHS dd; LBHS DDDD

**Operation:** TEMP ← MI  
IFF C = 0 then PC' ← PC + MI

**Condition Codes:** Not affected.

**Description:** Tests the state of the C (carry) bit and causes a branch if it is clear. When used after a subtract or compare on unsigned binary values, this instruction will branch if the register was higher than or the same as the memory operand.

**Addressing Mode:** Relative

**Comments:** This is a duplicate assembly-language mnemonic for the single machine instruction BCC. Generally not useful after INC/DEC, LD/ST, and TST/CLR/COM instructions.

# BIT

## Bit Test

# BIT

**Source Form:** Bit P

**Operation:**  $TEMP \leftarrow R \wedge M$

**Condition Codes:** H — Not affected.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Always cleared.  
C — Not affected.

**Description:** Performs the logical AND of the contents of accumulator A or B and the contents of memory location M and modifies the condition codes accordingly. The contents of accumulator A or B and memory location M are not affected.

**Addressing Modes:** Immediate  
Extended  
Direct  
Indexed

# BLE

## Branch on Less than or Equal to Zero

# BLE

**Source Forms:** BLE dd; LBLE DDDD

**Operation:** TEMP ← MI  
IFF  $Z \vee [N \oplus V] = 1$  then  $PC' ← PC + TEMP$

**Condition Codes:** Not affected.

**Description:** Causes a branch if the exclusive OR of the N (negative) and V (overflow) bits is 1 or if the Z (zero) bit is set. That is, branch if the sign of a valid twos complement result is, or would be, negative. When used after a subtract or compare operation on twos complement values, this instruction will branch if the register was less than or equal to the memory operand.

**Addressing Mode:** Relative

# BLO

## Branch on Lower

# BLO

**Source Forms:** BLO dd; LBLO DDDD

**Operation:** TEMP ← MI  
IFF C = 1 then PC' ← PC + TEMP

**Condition Codes:** Not affected.

**Description:** Tests the state of the C (carry) bit and causes a branch if it is set. When used after a subtract or compare on unsigned binary values, this instruction will branch if the register was lower than the memory operand.

**Addressing Mode:** Relative

**Comments:** This is a duplicate assembly-language mnemonic for the single machine instruction BCS. Generally not useful after INC/DEC, LD/ST, and TST/CLR/COM instructions.



# BLS

## Branch on Lower or Same

# BLS

**Source Forms:** BLS dd; LBLS DDDD

**Operation:** TEMP ← MI  
IFF (C v Z) = 1 then PC' ← PC + TEMP

**Condition Codes:** Not affected.

**Description:** Causes a branch if the previous operation caused either a carry or a zero result. When used after a subtract or compare operation on unsigned binary values, this instruction will branch if the register was lower than or the same as the memory operand.

**Addressing Mode:** Relative

**Comments:** Generally not useful after INC/DEC, LD/ST, and TST/CLR/COM instructions.

# BLT

Branch on Less than Zero

# BLT

**Source Forms:** BLT dd; LBLT DDDD

**Operation:** TEMP ← MI  
IFF [N ⊕ V] = 1 then PC' ← PC + TEMP

**Condition Codes:** Not affected.

**Description:** Causes a branch if either, but not both, of the N (negative) or V (overflow) bits is set. That is, branch if the sign of a valid twos complement result is, or would be, negative. When used after a subtract or compare operation on twos complement binary values, this instruction will branch if the register was less than the memory operand.

**Addressing Mode:** Relative

# BMI

## Branch on Minus

# BMI

**Source Forms:** BMI dd; LBMI DDDD

**Operation:** TEMP ← MI  
IFF N = 1 then PC' ← PC + TEMP

**Condition Codes:** Not affected.

**Description:** Tests the state of the N (negative) bit and causes a branch if set. That is, branch if the sign of the twos complement result is negative.

**Addressing Mode:** Relative

**Comments:** When used after an operation on signed binary values, this instruction will branch if the result is minus. It is generally preferred to use the LBLT instruction after signed operations.

# BNE

## Branch Not Equal

# BNE

**Source Forms:** BNE dd; LBNE DDDD

**Operation:** TEMP ← MI  
IFF Z = 0 then PC' ← PC + TEMP

**Condition Codes:** Not affected.

**Description:** Tests the state of the Z (zero) bit and causes a branch if it is clear. When used after a subtract or compare operation on any binary values, this instruction will branch if the register is, or would be, not equal to the memory operand.

**Addressing Mode:** Relative

# BPL

## Branch on Plus

# BPL

**Source Forms:** BPL dd; LBPL DDDD

**Operation:** TEMP ← Ml  
IFF N = 0 then PC' ← PC + TEMP

**Condition Codes:** Not affected.

**Description:** Tests the state of the N (negative) bit and causes a branch if it is clear. That is, branch if the sign of the two's complement result is positive.

**Addressing Mode:** Relative

**Comments:** When used after an operation on signed binary values, this instruction will branch if the result (possibly invalid) is positive. It is generally preferred to use the BGE instruction after signed operations.

# BRA

## Branch Always

# BRA

**Source Forms:** BRA dd; LBRA DDDD

**Operation:** TEMP ← MI  
PC' ← PC + TEMP

**Condition Codes:** Not affected.

**Description:** Causes an unconditional branch.

**Addressing Mode:** Relative

# BRN

## Branch Never

# BRN

**Source Forms:** BRN dd; LBRN DDDD

**Operation:** TEMP ← MI

**Condition Codes:** Not affected.

**Description:** Does not cause a branch. This instruction is essentially a no operation, but has a bit pattern logically related to branch always.

**Addressing Mode:** Relative

# BSR

## Branch to Subroutine

# BSR

**Source Forms:** BSR dd; LBSR DDDD

**Operation:**  $TEMP \leftarrow MI$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow PCL$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow PCH$   
 $PC' \leftarrow PC + TEMP$

**Condition Codes:** Not affected.

**Description:** The program counter is pushed onto the stack. The program counter is then loaded with the sum of the program counter and the offset.

**Addressing Mode:** Relative

**Comments:** A return from subroutine (RTS) instruction is used to reverse this process and must be the last instruction executed in a subroutine.



# BVC

## Branch on Overflow Clear

# BVC

**Source Forms:** BVC dd; LBVC DDDD

**Operation:** TEMP ← MI  
IFF V = 0 then PC' ← PC + TEMP

**Condition Codes:** Not affected.

**Description:** Tests the state of the V (overflow) bit and causes a branch if it is clear. That is, branch if the two's complement result was valid. When used after an operation on two's complement binary values, this instruction will branch if there was no overflow.

**Addressing Mode:** Relative

# BVS

## Branch on Overflow Set

# BVS

**Source Forms:** BVS dd; LBVS DDDD

**Operation:** TEMP ← MI  
IFF V = 1 then PC' ← PC + TEMP

**Condition Codes:** Not affected.

**Description:** Tests the state of the V (overflow) bit and causes a branch if it is set. That is, branch if the two's complement result was invalid. When used after an operation on two's complement binary values, this instruction will branch if there was an overflow.

**Addressing Mode:** Relative

# CLR

Clear

# CLR

**Source Form:** CLR Q

**Operation:** TEMP ← M  
M ← 0016

**Condition Codes:** H — Not affected.  
N — Always cleared.  
Z — Always set.  
V — Always cleared.  
C — Always cleared.

**Description:** Accumulator A or B or memory location M is loaded with 00000000.  
Note that the EA is read during this operation.

**Addressing Modes:** Inherent  
Extended  
Direct  
Indexed

# **CMP (8-Bit)    Compare Memory from Register    CMP (8-Bit)**

**Source Forms:**    CMPA P; CMPB P

**Operation:**     $TEMP \leftarrow R - M$

**Condition Codes:**    H — Undefined.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Set if an overflow is generated; cleared otherwise.  
C — Set if a borrow is generated; cleared otherwise.

**Description:**    Compares the contents of memory location to the contents of the specified register and sets the appropriate condition codes. Neither memory location M nor the specified register is modified. The carry flag represents a borrow and is set to the inverse of the resulting binary carry.

**Addressing Modes:** Immediate  
Extended  
Direct  
Indexed

# **CMP (16-Bit)** Compare Memory from Register **CMP (16-Bit)**

**Source Forms:** CMPD P; CMPX P; CMPY P; CMPU P; CMPS P

**Operation:** TEMP ← R - M:M + 1

**Condition Codes:** H — Not affected.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Set if an overflow is generated; cleared otherwise.  
C — Set if a borrow is generated; cleared otherwise.

**Description:** Compares the 16-bit contents of the concatenated memory locations M:M + 1 to the contents of the specified register and sets the appropriate condition codes. Neither the memory locations nor the specified register is modified unless autoincrement or autodecrement are used. The carry flag represents a borrow and is set to the inverse of the resulting binary carry.

**Addressing Modes:** Immediate  
Extended  
Direct  
Indexed

# COM

## Complement

# COM

**Source Forms:** COM Q; COMA; COMB

**Operation:**  $M' \leftarrow O + \bar{M}$

**Condition Codes:** H — Not affected.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Always cleared.  
C — Always set.

**Description:** Replaces the contents of memory location M or accumulator A or B with its logical complement. When operating on unsigned values, only BEQ and BNE branches can be expected to behave properly following a COM instruction. When operating on twos complement values, all signed branches are available.

**Addressing Modes:** Inherent  
Extended  
Direct  
Indexed

# CWAI

Clear CC bits and Wait for Interrupt

# CWAI

**Source Form:**

CWAI #\$XX

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| E | F | H | I | N | Z | V | C |
|---|---|---|---|---|---|---|---|

**Operation:**

CCR ← CCR  $\Delta$  MI (Possibly clear masks)  
Set E (entire state saved)  
SP' ← SP - 1, (SP) ← PCL  
SP' ← SP - 1, (SP) ← PCH  
SP' ← SP - 1, (SP) ← USL  
SP' ← SP - 1, (SP) ← USH  
SP' ← SP - 1, (SP) ← IYL  
SP' ← SP - 1, (SP) ← IYH  
SP' ← SP - 1, (SP) ← IXL  
SP' ← SP - 1, (SP) ← IXH  
SP' ← SP - 1, (SP) ← DPR  
SP' ← SP - 1, (SP) ← ACCB  
SP' ← SP - 1, (SP) ← ACCA  
SP' ← SP - 1, (SP) ← CCR

**Condition Codes:**

Affected according to the operation.

**Description:**

This instruction ANDs an immediate byte with the condition code register which may clear the interrupt mask bits I and F, stacks the entire machine state on the hardware stack and then looks for an interrupt. When a non-masked interrupt occurs, no further machine state information need be saved before vectoring to the interrupt handling routine. This instruction replaced the MC6800 CLI WAI sequence, but does not place the buses in a high-impedance state. A  $\overline{\text{FIRQ}}$  (fast interrupt request) may enter its interrupt handler with its entire machine state saved. The RTI (return from interrupt) instruction will automatically return the entire machine state after testing the E (entire) bit of the recovered condition code register.

**Addressing Mode:**

Immediate

**Comments:**

The following immediate values will have the following results:  
FF = enable neither  
EF = enable  $\overline{\text{IRQ}}$   
BF = enable  $\overline{\text{FIRQ}}$   
AF = enable both

# DAA

## Decimal Addition Adjust

# DAA

**Source Form:** DAA

**Operation:**  $ACCA' \leftarrow ACCA + CF(MSN):CF(LSN)$   
where CF is a Correction Factor, as follows: the CF for each nibble (BCD) digit is determined separately, and is either 6 or 0.

**Least Significant Nibble**

$CF(LSN) = 6$  IFF 1)  $C = 1$   
or 2)  $LSN > 9$

**Most Significant Nibble**

$CF(MSN) = 6$  IFF 1)  $C = 1$   
or 2)  $MSN > 9$   
or 3)  $MSN > 8$  and  $LSN > 9$

**Condition Codes:** H — Not affected.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Undefined.  
C — Set if a carry is generated or if the carry bit was set before the operation; cleared otherwise.

**Description:** The sequence of a single-byte add instruction on accumulator A (either ADDA or ADCA) and a following decimal addition adjust instruction results in a BCD addition with an appropriate carry bit. Both values to be added must be in proper BCD form (each nibble such that:  $0 \leq \text{nibble} \leq 9$ ). Multiple-precision addition must add the carry generated by this decimal addition adjust into the next higher digit during the add operation (ADCA) immediately prior to the next decimal addition adjust.

**Addressing Mode:** Inherent



# DEC

## Decrement

# DEC

**Source Forms:** DEC Q; DECA; DECB

**Operation:**  $M' \leftarrow M - 1$

**Condition Codes:** H — Not affected.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Set if the original operand was 10000000; cleared otherwise.  
C — Not affected.

**Description:** Subtract one from the operand. The carry bit is not affected, thus allowing this instruction to be used as a loop counter in multiple-precision computations. When operating on unsigned values, only BEQ and BNE branches can be expected to behave consistently. When operating on twos complement values, all signed branches are available.

**Addressing Modes:** Inherent  
Extended  
Direct  
Indexed

# EOR

## Exclusive OR

# EOR

**Source Forms:** EORA P; EORB P

**Operation:**  $R' \leftarrow R \oplus M$

**Condition Codes:** H — Not affected.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Always cleared.  
C — Not affected.

**Description:** The contents of memory location M is exclusive ORed into an 8-bit register.

**Addressing Modes:** Immediate  
Extended  
Direct  
Indexed

# EXG

## Exchange Registers

# EXG

**Source Form:** EXG R1,R2

**Operation:** R1 ↔ R2

**Condition Codes:** Not affected (unless one of the registers is the condition code register).

**Description:** Exchanges data between two designated registers. Bits 3-0 of the postbyte define one register, while bits 7-4 define the other, as follows:

|                  |                  |
|------------------|------------------|
| 0000 = A:B       | 1000 = A         |
| 0001 = X         | 1001 = B         |
| 0010 = Y         | 1010 = CCR       |
| 0011 = US        | 1011 = DPR       |
| 0100 = SP        | 1100 = Undefined |
| 0101 = PC        | 1101 = Undefined |
| 0110 = Undefined | 1110 = Undefined |
| 0111 = Undefined | 1111 = Undefined |

Only like size registers may be exchanged. (8-bit with 8-bit or 16-bit with 16-bit.)

**Addressing Mode:** Immediate

# INC

## Increment

# INC

**Source Forms:** INC Q; INCA; INCB

**Operation:**  $M' \leftarrow M + 1$

**Condition Codes:** H — Not affected.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Set if the original operand was 01111111; cleared otherwise.  
C — Not affected.

**Description:** Adds to the operand. The carry bit is not affected, thus allowing this instruction to be used as a loop counter in multiple-precision computations. When operating on unsigned values, only the BEQ and BNE branches can be expected to behave consistently. When operating on twos complement values, all signed branches are correctly available.

**Addressing Modes:** Inherent  
Extended  
Direct  
Indexed

# JMP

Jump

# JMP

**Source Form:** JMP EA

**Operation:**  $PC' \leftarrow EA$

**Condition Codes:** Not affected.

**Description:** Program control is transferred to the effective address.

**Addressing Modes:** Extended  
Direct  
Indexed

# JSR

## Jump to Subroutine

# JSR

**Source Form:** JSR EA

**Operation:**  $SP' \leftarrow SP - 1, (SP) \leftarrow PCL$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow PCH$   
 $PC' \leftarrow EA$

**Condition Codes:** Not affected.

**Description:** Program control is transferred to the effective address after storing the return address on the hardware stack. A RTS instruction should be the last executed instruction of the subroutine.

**Addressing Modes:** Extended  
Direct  
Indexed

# LD (8-Bit)

Load Register from Memory

# LD (8-Bit)

**Source Forms:** LDA P; LDB P

**Operation:**  $R' \leftarrow M$

**Condition Codes:** H — Not affected.  
N — Set if the loaded data is negative; cleared otherwise.  
Z — Set if the loaded data is zero; cleared otherwise.  
V — Always cleared.  
C — Not affected.

**Description:** Loads the contents of memory location M into the designated register.

**Addressing Modes:** Immediate  
Extended  
Direct  
Indexed

# LD (16-Bit)

Load Register from Memory

# LD (16-Bit)

**Source Forms:** LDD P; LDX P; LDY P; LDS P; LDU P

**Operation:**  $R' \leftarrow M:M + 1$

**Condition Codes:** H — Not affected.  
N — Set if the loaded data is negative; cleared otherwise.  
Z — Set if the loaded data is zero; cleared otherwise.  
V — Always cleared.  
C — Not affected.

**Description:** Load the contents of the memory location M:M+1 into the designated 16-bit register.

**Addressing Modes:** Immediate  
Extended  
Direct  
Indexed



# LEA

## Load Effective Address

# LEA

**Source Forms:** LEAX, LEAY, LEAS, LEAU

**Operation:**  $R' \leftarrow EA$

**Condition Codes:** H — Not affected.  
N — Not affected.  
Z — LEAX, LEAY: Set if the result is zero; cleared otherwise.  
LEAS, LEAU: Not affected.  
V — Not affected.  
C — Not affected.

**Description:** Calculates the effective address from the indexed addressing mode and places the address in an indexable register.

LEAX and LEAY affect the Z (zero) bit to allow use of these registers as counters and for MC6800 INX/DEX compatibility.

LEAU and LEAS do not affect the Z bit to allow cleaning up the stack while returning the Z bit as a parameter to a calling routine, and also for MC6800 INS/DES compatibility.

**Addressing Mode:** Indexed

**Comments:** Due to the order in which effective addresses are calculated internally, the LEAX, X + + and LEAX, X + do not add 2 and 1 (respectively) to the X register; but instead leave the X register unchanged. This also applies to the Y, U, and S registers. For the expected results, use the faster instruction LEAX 2, X and LEAX 1, X.

Some examples of LEA instruction uses are given in the following table.


| Instruction | Operation     | Comment                        |
|-------------|---------------|--------------------------------|
| LEAX 10, X  | $X + 10 - X$  | Adds 5-bit constant 10 to X    |
| LEAX 500, X | $X + 500 - X$ | Adds 16-bit constant 500 to X  |
| LEAY A, Y   | $Y + A - Y$   | Adds 8-bit accumulator to Y    |
| LEAY D, Y   | $Y + D - Y$   | Adds 16-bit D accumulator to Y |
| LEAU -10, U | $U - 10 - U$  | Subtracts 10 from U            |
| LEAS -10, S | $S - 10 - S$  | Used to reserve area on stack  |
| LEAS 10, S  | $S + 10 - S$  | Used to 'clean up' stack       |
| LEAX 5, S   | $S + 5 - X$   | Transfers as well as adds      |

# LSL

## Logical Shift Left

# LSL

**Source Forms:** LSL Q; LSLA; LSLB

**Operation:** 

**Condition Codes:** H — Undefined.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Loaded with the result of the exclusive OR of bits six and seven of the original operand.  
C — Loaded with bit seven of the original operand.

**Description:** Shifts all bits of accumulator A or B or memory location M one place to the left. Bit zero is loaded with a zero. Bit seven of accumulator A or B or memory location M is shifted into the C (carry) bit.

**Addressing Modes:** Inherent  
Extended  
Direct  
Indexed

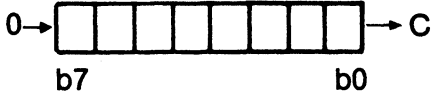
**Comments:** This is a duplicate assembly-language mnemonic for the single machine instruction ASL.

# LSR

## Logical Shift Right

# LSR

**Source Forms:** LSR Q; LSRA; LSRB

**Operation:** 

**Condition Codes:** H — Not affected.  
N — Always cleared.  
Z — Set if the result is zero; cleared otherwise.  
V — Not affected.  
C — Loaded with bit zero of the original operand.

**Description:** Performs a logical shift right on the operand. Shifts a zero into bit seven and bit zero into the C (carry) bit.

**Addressing Modes:** Inherent  
Extended  
Direct  
Indexed

# MUL

## Multiply

# MUL

**Source Form:** MUL

**Operation:**  $ACCA':ACCB' \leftarrow ACCA \times ACCB$

**Condition Codes:** H — Not affected.  
N — Not affected.  
Z — Set if the result is zero; cleared otherwise.  
V — Not affected.  
C — Set if ACCB bit 7 of result is set; cleared otherwise.

**Description:** Multiply the unsigned binary numbers in the accumulators and place the result in both accumulators (ACCA contains the most-significant byte of the result). Unsigned multiply allows multiple-precision operations.

**Addressing Mode:** Inherent

**Comments:** The C (carry) bit allows rounding the most-significant byte through the sequence: MUL, ADCA #0.

# NEG

Negate

# NEG

**Source Forms:** NEG Q; NEGA; NEGB

**Operation:**  $M' \leftarrow 0 - M$

**Condition Codes:** H — Undefined.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Set if the original operand was 10000000.  
C — Set if a borrow is generated; cleared otherwise.

**Description:** Replaces the operand with its twos complement. The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry. Note that  $80_{16}$  is replaced by itself and only in this case is the V (overflow) bit set. The value  $00_{16}$  is also replaced by itself, and only in this case is the C (carry) bit cleared.

**Addressing Modes:** Inherent  
Extended  
Direct

# NOP

No Operation

# NOP

**Source Form:** NOP

**Operation:** Not affected.

**Condition Codes:** This instruction causes only the program counter to be incremented. No other registers or memory locations are affected.

**Addressing Mode:** Inherent

# OR

## Inclusive OR Memory into Register

# OR

**Source Forms:** ORA P; ORB P

**Operation:**  $R' \leftarrow R \vee M$

**Condition Codes:** H — Not affected.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Always cleared.  
C — Not affected.

**Description:** Performs an inclusive OR operation between the contents of accumulator A or B and the contents of memory location M and the result is stored in accumulator A or B.

**Addressing Modes:** Immediate  
Extended  
Direct  
Indexed

**OR****Inclusive OR Memory Immediate into Condition Code Register****OR****Source Form:** ORCC #XX**Operation:**  $R' \leftarrow R \vee MI$ **Condition Codes:** Affected according to the operation.**Description:** Performs an inclusive OR operation between the contents of the condition code registers and the immediate value, and the result is placed in the condition code register. This instruction may be used to set interrupt masks (disable interrupts) or any other bit(s).**Addressing Mode:** Immediate

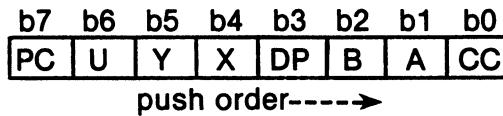


# PSHS

## Push Registers on the Hardware Stack

# PSHS

**Source Form:** PSHS register list  
PSHS #LABEL  
Postbyte:



**Operation:**

- IFF b7 of postbyte set, then:  $SP' \leftarrow SP - 1, (SP) \leftarrow PCL$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow PCH$
- IFF b6 of postbyte set, then:  $SP' \leftarrow SP - 1, (SP) \leftarrow USL$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow USH$
- IFF b5 of postbyte set, then:  $SP' \leftarrow SP - 1, (SP) \leftarrow IYL$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow IYH$
- IFF b4 of postbyte set, then:  $SP' \leftarrow SP - 1, (SP) \leftarrow IXL$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow IXH$
- IFF b3 of postbyte set, then:  $SP' \leftarrow SP - 1, (SP) \leftarrow DPR$
- IFF b2 of postbyte set, then:  $SP' \leftarrow SP - 1, (SP) \leftarrow ACCB$
- IFF b1 of postbyte set, then:  $SP' \leftarrow SP - 1, (SP) \leftarrow ACCA$
- IFF b0 of postbyte set, then:  $SP' \leftarrow SP - 1, (SP) \leftarrow CCR$

**Condition Codes:** Not affected.

**Description:** All, some, or none of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself).

**Addressing Mode:** Immediate

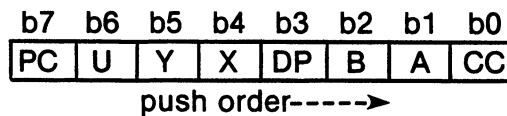
**Comments:** A single register may be placed on the stack with the condition codes set by doing an autodecrement store onto the stack (example: STX, --S).

# PSHU

## Push Registers on the User Stack

# PSHU

**Source Form:** PSHU register list  
PSHU #LABEL  
Postbyte:



**Operation:**

- IFF b7 of postbyte set, then:  $US' \leftarrow US - 1, (US) \leftarrow PCL$   
 $US' \leftarrow US - 1, (US) \leftarrow PCH$
- IFF b6 of postbyte set, then:  $US' \leftarrow US - 1, (US) \leftarrow SPL$   
 $US' \leftarrow US - 1, (US) \leftarrow SPH$
- IFF b5 of postbyte set, then:  $US' \leftarrow US - 1, (US) \leftarrow IYL$   
 $US' \leftarrow US - 1, (US) \leftarrow IYH$
- IFF b4 of postbyte set, then:  $US' \leftarrow US - 1, (US) \leftarrow IXL$   
 $US' \leftarrow US - 1, (US) \leftarrow IXH$
- IFF b3 of postbyte set, then:  $US' \leftarrow US - 1, (US) \leftarrow DPR$
- IFF b2 of postbyte set, then:  $US' \leftarrow US - 1, (US) \leftarrow ACCB$
- IFF b1 of postbyte set, then:  $US' \leftarrow US - 1, (US) \leftarrow ACCA$
- IFF b0 of postbyte set, then:  $US' \leftarrow US - 1, (US) \leftarrow CCR$

**Condition Codes:** Not affected.

**Description:** All, some, or none of the processor registers are pushed onto the user stack (with the exception of the user stack pointer itself).

**Addressing Mode:** Immediate

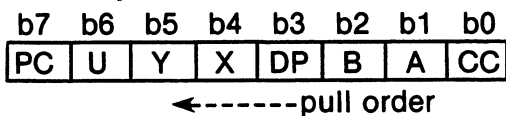
**Comments:** A single register may be placed on the stack with the condition codes set by doing an autodecrement store onto the stack (example: STX , - - U).

# PULS

## Pull Registers from the Hardware Stack

# PULS

**Source Form:** PULS register list  
 PULS #LABEL  
 Postbyte:



**Operation:**

- IFF b0 of postbyte set, then: CCR' ← (SP), SP' ← SP + 1
- IFF b1 of postbyte set, then: ACCA' ← (SP), SP' ← SP + 1
- IFF b2 of postbyte set, then: ACCB' ← (SP), SP' ← SP + 1
- IFF b3 of postbyte set, then: DPR' ← (SP), SP' ← SP + 1
- IFF b4 of postbyte set, then: IXH' ← (SP), SP' ← SP + 1
- IXL' ← (SP), SP' ← SP + 1
- IFF b5 of postbyte set, then: IYH' ← (SP), SP' ← SP + 1
- IYL' ← (SP), SP' ← SP + 1
- IFF b6 of postbyte set, then: USH' ← (SP), SP' ← SP + 1
- USL' ← (SP), SP' ← SP + 1
- IFF b7 of postbyte set, then: PCH' ← (SP), SP' ← SP + 1
- PCL' ← (SP), SP' ← SP + 1

**Condition Codes:** May be pulled from stack; not affected otherwise.

**Description:** All, some, or none of the processor registers are pulled from the hardware stack (with the exception of the hardware stack pointer itself).

**Addressing Mode:** Immediate

**Comments:** A single register may be pulled from the stack with condition codes set by doing an autoincrement load from the stack (example: LDX ,S + +).

# PULU

## Pull Registers from the User Stack

# PULU

**Source Form:** PULU register list  
 PULU #LABEL

Postbyte:

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| PC | U  | Y  | X  | DP | B  | A  | CC |

←----- pull order

**Operation:**

- IFF b0 of postbyte set, then: CCR' ←(US), US' ← US + 1
- IFF b1 of postbyte set, then: ACCA' ←(US), US' ← US + 1
- IFF b2 of postbyte set, then: ACCB' ←(US), US' ← US + 1
- IFF b3 of postbyte set, then: DPR' ←(US), US' ← US + 1
- IFF b4 of postbyte set, then: IXH' ←(US), US' ← US + 1
- IXL' ←(US), US' ← US + 1
- IFF b5 of postbyte set, then: IYH' ←(US), US' ← US + 1
- IYL' ←(US), US' ← US + 1
- IFF b6 of postbyte set, then: SPH' ←(US), US' ← US + 1
- SPL' ←(US), US' ← US + 1
- IFF b7 of postbyte set, then: PCH ←(US), US' ← US + 1
- PCL' ←(US), US' ← US + 1

**Condition Codes:** May be pulled from stack; not affected otherwise.

**Description:** All, some, or none of the processor registers are pulled from the user stack (with the exception of the user stack pointer itself).

**Addressing Mode:** Immediate

**Comments:** A single register may be pulled from the stack with condition codes set by doing an autoincrement load from the stack (example: LDX,U++).

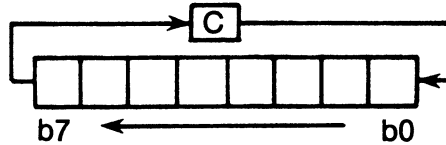
# ROL

## Rotate Left

# ROL

**Source Forms:** ROL Q; ROLA; ROLB

**Operation:**



**Condition Codes:**

- H — Not affected.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Loaded with the result of the exclusive OR of bits six and seven of the original operand.
- C — Loaded with bit seven of the original operand.

**Description:** Rotates all bits of the operand one place left through the C (carry) bit. This is a 9-bit rotation.

**Addressing Mode:** Inherent  
Extended  
Direct  
Indexed

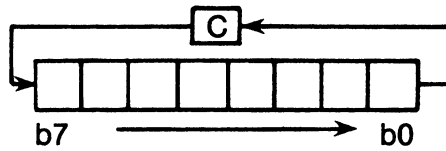
# ROR

Rotate Right

# ROR

**Source Forms:** ROR Q; RORA; RORB

**Operation:**



**Condition Codes:**

- H — Not affected.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Not affected.
- C — Loaded with bit zero of the previous operand.

**Description:** Rotates all bits of the operand one place right through the C (carry) bit. This is a 9-bit rotation.

**Addressing Modes:** Inherent  
Extended  
Direct  
Indexed

# RTI

## Return from Interrupt

# RTI

**Source Form:** RTI

**Operation:**  $CCR' \leftarrow (SP), SP' \leftarrow SP + 1$ , then

IFF CCR bit E is set, then:

|       |  |
|-------|--|
| ACCA' | $\leftarrow (SP), SP' \leftarrow SP + 1$ |
| ACCB' | $\leftarrow (SP), SP' \leftarrow SP + 1$ |
| DPR'  | $\leftarrow (SP), SP' \leftarrow SP + 1$ |
| IXH'  | $\leftarrow (SP), SP' \leftarrow SP + 1$ |
| IXL'  | $\leftarrow (SP), SP' \leftarrow SP + 1$ |
| IYH'  | $\leftarrow (SP), SP' \leftarrow SP + 1$ |
| IYL'  | $\leftarrow (SP), SP' \leftarrow SP + 1$ |
| USH'  | $\leftarrow (SP), SP' \leftarrow SP + 1$ |
| USL'  | $\leftarrow (SP), SP' \leftarrow SP + 1$ |
| PCH'  | $\leftarrow (SP), SP' \leftarrow SP + 1$ |
| PCL'  | $\leftarrow (SP), SP' \leftarrow SP + 1$ |

IFF CCR bit E is clear, then:

|      |  |
|------|--|
| PCH' | $\leftarrow (SP), SP' \leftarrow SP + 1$ |
| PCL' | $\leftarrow (SP), SP' \leftarrow SP + 1$ |

**Condition Codes:** Recovered from the stack.

**Description:** The saved machine state is recovered from the hardware stack and control is returned to the interrupted program. If the recovered E (entire) bit is clear, it indicates that only a subset of the machine state was saved (return address and condition codes) and only that subset is recovered.

**Addressing Mode:** Inherent

# RTS

## Return from Subroutine

# RTS

**Source Form:** RTS

**Operation:** PCH' ← (SP), SP' ← SP + 1  
PCL' ← (SP), SP' ← SP + 1

**Condition Codes:** Not affected.

**Description:** Program control is returned from the subroutine to the calling program. The return address is pulled from the stack.

**Addressing Mode:** Inherent



# SBC

## Subtract with Borrow

# SBC

**Source Forms:** SBCA P; SBCB P

**Operation:**  $R' \leftarrow R - M - C$

**Condition Codes:** H — Undefined.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Set if an overflow is generated; cleared otherwise.  
C — Set if a borrow is generated; cleared otherwise.

**Description:** Subtracts the contents of memory location M and the borrow (in the C (carry) bit) from the contents of the designated 8-bit register, and places the result in that register. The C bit represents a borrow and is set to the inverse of the resulting binary carry.

**Addressing Modes:** Immediate  
Extended  
Direct  
Indexed

# SEX

Sign Extended

# SEX

**Source Form:** SEX

**Operation:** If bit seven of ACCB is set then  $ACCA' \leftarrow FF_{16}$   
else  $ACCA' \leftarrow 00_{16}$

**Condition Codes:** H — Not affected.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Not affected.  
C — Not affected.

**Description:** This instruction transforms a two's complement 8-bit value in accumulator B into a two's complement 16-bit value in the D accumulator.

**Addressing Mode:** Inherent

# ST (8-Bit)

Store Register into Memory

# ST (8-Bit)

**Source Forms:** STA P; STB P

**Operation:**  $M' \leftarrow R$

**Condition Codes:** H — Not affected.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Always cleared.  
C — Not affected.

**Description:** Writes the contents of an 8-bit register into a memory location.

**Addressing Modes:** Extended  
Direct  
Indexed

# ST (16-Bit)

Store Register into Memory

# ST (16-Bit)

**Source Forms:** STD P; STX P; STY P; STS P; STU P

**Operation:**  $M':M + 1' \leftarrow R$

**Condition Codes:** H — Not affected.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Always cleared.  
C — Not affected.

**Description:** Writes the contents of a 16-bit register into two consecutive memory locations.

**Addressing Modes:** Extended  
Direct  
Indexed

# **SUB (8-Bit)**      **Subtract Memory from Register**      **SUB (8-Bit)**

**Source Forms:**      SUBA P; SUBB P

**Operation:**       $R' \leftarrow R - M$

**Condition Codes:**    H — Undefined.  
                          N — Set if the result is negative; cleared otherwise.  
                          Z — Set if the result is zero; cleared otherwise.  
                          V — Set if the overflow is generated; cleared otherwise.  
                          C — Set if a borrow is generated; cleared otherwise.

**Description:**      Subtracts the value in memory location M from the contents of a designated 8-bit register. The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry.

**Addressing Modes:** Immediate  
                          Extended  
                          Direct  
                          Indexed

# **SUB (16-Bit)** Subtract Memory from Register **SUB (16-Bit)**

**Source Forms:** SUBD P

**Operation:**  $R' \leftarrow R - M:M + 1$

**Condition Codes:** H — Not affected.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Set if the overflow is generated; cleared otherwise.  
C — Set if a borrow is generated; cleared otherwise.

**Description:** Subtracts the value in memory location M:M + 1 from the contents of a designated 16-bit register. The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry.

**Addressing Modes:** Immediate  
Extended  
Direct  
Indexed

# SWI

## Software Interrupt

# SWI

**Source Form:** SWI

**Operation:** Set E (entire state will be saved)  
SP' ← SP - 1, (SP) ← PCL  
SP' ← SP - 1, (SP) ← PCH  
SP' ← SP - 1, (SP) ← USL  
SP' ← SP - 1, (SP) ← USH  
SP' ← SP - 1, (SP) ← IYL  
SP' ← SP - 1, (SP) ← IYH  
SP' ← SP - 1, (SP) ← IXL  
SP' ← SP - 1, (SP) ← IXH  
SP' ← SP - 1, (SP) ← DPR  
SP' ← SP - 1, (SP) ← ACCB  
SP' ← SP - 1, (SP) ← ACCA  
SP' ← SP - 1, (SP) ← CCR  
Set I, F (mask interrupts)  
PC' ← (FFFA):(FFFB)

**Condition Codes:** Not affected.

**Description:** All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself), and control is transferred through the software interrupt vector. Both the normal and fast interrupts are masked (disabled).

**Addressing Mode:** Inherent

# SWI2

## Software Interrupt 2

# SWI2

**Source Form:** SWI2

**Operation:** Set E (entire state saved)  
SP' ← SP - 1, (SP) ← PCL  
SP' ← SP - 1, (SP) ← PCH  
SP' ← SP - 1, (SP) ← USL  
SP' ← SP - 1, (SP) ← USH  
SP' ← SP - 1, (SP) ← IYL  
SP' ← SP - 1, (SP) ← IYH  
SP' ← SP - 1, (SP) ← IXL  
SP' ← SP - 1, (SP) ← IXH  
SP' ← SP - 1, (SP) ← DPR  
SP' ← SP - 1, (SP) ← ACCB  
SP' ← SP - 1, (SP) ← ACCA  
SP' ← SP - 1, (SP) ← CCR  
PC' ← (FFF4):(FFF5)

**Condition Codes:** Not affected.

**Description:** All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself), and control is transferred through the software interrupt 2 vector. This interrupt is available to the end user and must not be used in packaged software. This interrupt does not mask (disable) the normal and fast interrupts.

**Addressing Mode:** Inherent



# SWI3

## Software Interrupt 3

# SWI3

**Source Form:** SWI 3

**Operation:** Set E (entire state will be saved)  
 $SP' \leftarrow SP - 1, (SP) \leftarrow PCL$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow PCH$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow USL$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow USH$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow IYL$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow IYH$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow IXL$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow IXH$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow DPR$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow ACCB$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow ACCA$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow CCR$   
 $PC' \leftarrow (FFF2):(FFF3)$

**Condition Codes:** Not affected.

**Description:** All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself), and control is transferred through the software interrupt 3 vector. This interrupt does not mask (disable) the normal and fast interrupts.

**Addressing Mode:** Inherent

# SYNC

## Synchronize to External Event

# SYNC

**Source Form:** SYNC

**Operation:** Stop processing instructions

**Condition Codes:** Not affected.

**Description:** When a SYNC instruction is executed, the processor enters a synchronizing state, stops processing instructions, and waits for an interrupt. When an interrupt occurs, the synchronizing state is cleared and processing continues. If the interrupt is enabled, and it lasts three cycles or more, the processor will perform the interrupt routine. If the interrupt is masked or is shorter than three cycles, the processor simply continues to the next instruction. While in the synchronizing state, the address and data buses are in the high-impedance state.

This instruction provides software synchronization with a hardware process. Consider the following example for high-speed acquisition of data:

|      |            |   |
|------|------------|---|
| FAST | SYNC       | WAIT FOR DATA                           |
|      | Interrupt! |   |
|      | LDA        | DISC DATA FROM DISC AND CLEAR INTERRUPT |
|      | STA        | ,X+ PUT IN BUFFER                       |
|      | DECB       | COUNT IT, DONE?                         |
|      | BNE        | FAST GO AGAIN IF NOT.                   |

The synchronizing state is cleared by any interrupt. Of course, enabled interrupts at this point may destroy the data transfer and, as such, should represent only emergency conditions.

The same connection used for interrupt-driven I/O service may also be used for high-speed data transfers by setting the interrupt mask and using the SYNC instruction as the above example demonstrates.

**Addressing Mode:** Inherent

# TFR

## Transfer Register to Register

# TFR

**Source Form:** TFR R1, R2

**Operation:** R1 → R2

**Condition Code:** Not affected unless R2 is the condition code register.

**Description:** Transfers data between two designated registers. Bits 7-4 of the postbyte define the source register, while bits 3-0 define the destination register, as follows:

|                  |                  |
|------------------|------------------|
| 0000 = A:B       | 1000 = A         |
| 0001 = X         | 1001 = B         |
| 0010 = Y         | 1010 = CCR       |
| 0011 = US        | 1011 = DPR       |
| 0100 = SP        | 1100 = Undefined |
| 0101 = PC        | 1101 = Undefined |
| 0110 = Undefined | 1110 = Undefined |
| 0111 = Undefined | 1111 = Undefined |

Only like size registers may be transferred. (8-bit to 8-bit, or 16-bit to 16-bit.)

**Addressing Mode:** Immediate

# TST

## Test

# TST

**Source Forms:** TST Q; TSTA; TSTB

**Operation:** TEMP ← M - 0

**Condition Codes:** H — Not affected.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Always cleared.  
C — Not affected.

**Description:** Set the N (negative) and Z (zero) bits according to the contents of memory location M, and clear the V (overflow) bit. The TST instruction provides only minimum information when testing unsigned values; since no unsigned value is less than zero, BLO and BLS have no utility. While BHI could be used after TST, it provides exactly the same control as BNE, which is preferred. The signed branches are available.

**Addressing Modes:** Inherent  
Extended  
Direct  
Indexed

**Comments:** The MC6800 processor clears the C (carry) bit.

# FIRQ

## Fast Interrupt Request (Hardware Interrupt)

# FIRQ

**Operation:** IFF F bit clear, then:  $SP' \leftarrow SP - 1, (SP) \leftarrow PCL$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow PCH$   
Clear E (subset state is saved)  
 $SP' \leftarrow SP - 1, (SP) \leftarrow CCR$   
Set F, I (mask further interrupts)  
 $PC' \leftarrow (FFF6):(FFF7)$

**Condition Codes:** Not affected.

**Description:** A  $\overline{FIRQ}$  (fast interrupt request) with the F (fast interrupt request mask) bit clear causes this interrupt sequence to occur at the end of the current instruction. The program counter and condition code register are pushed onto the hardware stack. Program control is transferred through the fast interrupt request vector. An RTI (return from interrupt) instruction returns the processor to the original task. It is possible to enter the fast interrupt request routine with the entire machine state saved if the fast interrupt request occurs after a clear and wait for interrupt instruction. A normal interrupt request has lower priority than the fast interrupt request and is prevented from interrupting the fast interrupt request routine by automatic setting of the I (interrupt request mask) bit. This mask bit could then be reset during the interrupt routine if priority was not desired. The fast interrupt request allows operations on memory, TST, INC, DEC, etc. instructions without the overhead of saving the entire machine state on the stack.

**Addressing Mode:** Inherent

**$\overline{\text{IRQ}}$** **Interrupt Request (Hardware Interrupt)** **$\overline{\text{IRQ}}$** 

**Operation:** IFF I bit clear, then:

- $\text{SP}' \leftarrow \text{SP} - 1, (\text{SP}) \leftarrow \text{PCL}$
- $\text{SP}' \leftarrow \text{SP} - 1, (\text{SP}) \leftarrow \text{PCH}$
- $\text{SP}' \leftarrow \text{SP} - 1, (\text{SP}) \leftarrow \text{USL}$
- $\text{SP}' \leftarrow \text{SP} - 1, (\text{SP}) \leftarrow \text{USH}$
- $\text{SP}' \leftarrow \text{SP} - 1, (\text{SP}) \leftarrow \text{IYL}$
- $\text{SP}' \leftarrow \text{SP} - 1, (\text{SP}) \leftarrow \text{IYH}$
- $\text{SP}' \leftarrow \text{SP} - 1, (\text{SP}) \leftarrow \text{IXL}$
- $\text{SP}' \leftarrow \text{SP} - 1, (\text{SP}) \leftarrow \text{IXH}$
- $\text{SP}' \leftarrow \text{SP} - 1, (\text{SP}) \leftarrow \text{DPR}$
- $\text{SP}' \leftarrow \text{SP} - 1, (\text{SP}) \leftarrow \text{ACCB}$
- $\text{SP}' \leftarrow \text{SP} - 1, (\text{SP}) \leftarrow \text{ACCA}$
- Set E (entire state saved)
- $\text{SP}' \leftarrow \text{SP} - 1, (\text{SP}) \leftarrow \text{CCR}$
- Set I (mask further  $\overline{\text{IRQ}}$  interrupts)
- $\text{PC}' \leftarrow (\text{FFF8}):(\text{FFF9})$

**Condition Codes:** Not affected.

**Description:** If the I (interrupt request mask) bit is clear, a low level on the  $\overline{\text{IRQ}}$  input causes this interrupt sequence to occur at the end of the current instruction. Control is returned to the interrupted program using a RTI (return from interrupt) instruction. A  $\overline{\text{FIRQ}}$  (fast interrupt request) may interrupt a normal  $\overline{\text{IRQ}}$  (interrupt request) routine and be recognized anytime after the interrupt vector is taken.

**Addressing Mode:** Inherent

# NMI

## Non-Maskable Interrupt (Hardware Interrupt)

# NMI

**Operation:**

$SP' \leftarrow SP - 1, (SP) \leftarrow PCL$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow PCH$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow USL$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow USH$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow IYL$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow IYH$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow IXL$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow IXH$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow DPR$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow ACCB$   
 $SP' \leftarrow SP - 1, (SP) \leftarrow ACCA$   
Set E (entire state save)  
 $SP' \leftarrow SP - 1, (SP) \leftarrow CCR$   
Set I, F (mask interrupts)  
 $PC' \leftarrow (FFFC):(FFFD)$

**Condition Codes:** Not affected.

**Description:** A negative edge on the  $\overline{NMI}$  (non-maskable interrupt) input causes all of the processor's registers (except the hardware stack pointer) to be pushed onto the hardware stack, starting at the end of the current instruction. Program control is transferred through the NMI vector. Successive negative edges on the  $\overline{NMI}$  input will cause successive  $\overline{NMI}$  operations. Non-maskable interrupt operation can be internally blocked by a  $\overline{RESET}$  operation and any non-maskable interrupt that occurs will be latched. If this happens, the non-maskable interrupt operation will occur after the first load into the stack pointer (LDS; TFR r,s; EXG r,s; etc.) after  $\overline{RESET}$ .

**Addressing Mode:** Inherent

# RESTART

Restart (Hardware Interrupt)

# RESTART

**Operation:** CCR'  $\leftarrow$  X1X1XXXX  
DPR'  $\leftarrow$  0016  
PC'  $\leftarrow$  (FFFE):(FFFF)

**Condition Codes:** Not affected.

**Description:** The processor is initialized (required after power-on) to start program execution. The starting address is fetched from the restart vector.

**Addressing Mode:** Extended Indirect



## **APPENDIX B ASSIST09 MONITOR PROGRAM**

### **B.1 GENERAL DESCRIPTION**

The M6809 is a high-performance microprocessor which supports modern programming techniques such as position-independent, reentrancy, and modular programming. For a software monitor to take advantage of such capabilities demands a more refined and sophisticated user interface than that provided by previous monitors. ASSIST09 is a monitor which supports the advanced features that the M6809 makes possible. ASSIST09 features include the following:

- Coded in a position (address) independent manner. Will execute anywhere in the 64K address space.
- Multiple means available for installing user modifications and extensions.
- Full complement of commands for program development including breakpoint and trace.
- Sophisticated monitor calls for completely address-independent user program services.
- RAM work area is located relative to the ASSIST09 ROM, not at a fixed address as with other monitors.
- Easily adapted to real-time environments.
- Hooks for user command tables, I/O handlers, and default specifications.
- A complete user interface with services normally only seen in full disk operating systems.

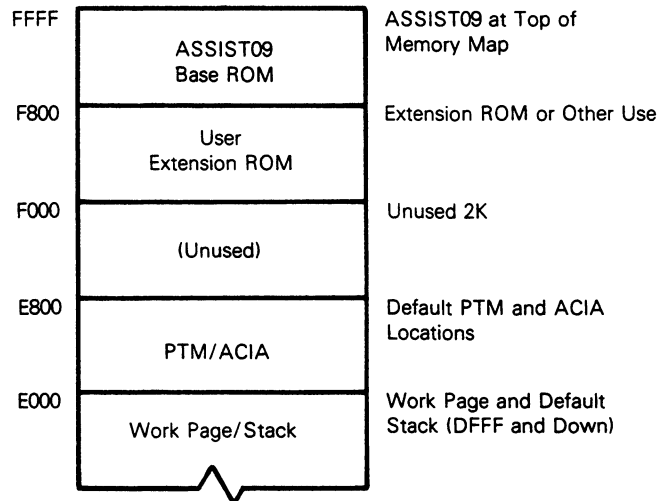
The concise instruction set of the M6809 allows all of these functions and more to be contained in only 2048 bytes.

The ASSIST09 monitor is easily adapted to run under control of a real-time operating system. A special function is available which allows voluntary time-slicing, as well as forced time-slicing upon the use of several service routines by a user program.

### **B.2 IMPLEMENTATION REQUIREMENTS**

Since ASSIST09 was coded in an address-independent manner, it will properly execute anywhere in the 64K address space of the M6809. However, an assumption must be made regarding the location of a work area needed to hold miscellaneous variables and the default stack location. This work area is called the page work area and it is addressed within ASSIST09 by use of the direct page register. It is located relative to the start of the

ASSIST09 ROM by an offset of –1900 hexadecimal. Assuming ASSIST09 resides at the top of the memory address space for direct control of the hardware interrupt vectors, the memory map would appear as shown in Figure B-1.



**Figure B-1. Memory Map**

If F800 is not the start of the monitor ROM the addresses would change, but the relative locations would remain the same except for the programmable timer module (PTM) and asynchronous communications interface adapter (ACIA) default addresses which are fixed.

The default console input/output handlers access an ACIA located at E008. For trace commands, a PTM with default address E000 is used to force an  $\overline{\text{NMI}}$  so that single instructions may be executed. These default addresses may easily be changed using one of several methods. The console I/O handlers may also be replaced by user routines. The PTM is initialized during the MONITR service call (see Paragraph B.9 SERVICES) to fireup the monitor unless its default address has been changed to zero, in which case no PTM references will occur.

### B.3 INTERRUPT CONTROL

Upon reset, a vector table is created which contains, among other things, default interrupt vector handler appendage addresses. These routines may easily be replaced by user appendages with the vector swap service described later. The default actions taken by the appendages are as follows:

$\overline{\text{RESET}}$  — Build the ASSIST09 vector table and setup monitor defaults, then invoke the monitor startup routine.

SWI — Request a service from ASSIST09.

$\overline{\text{FIRQ}}$  — An immediate RTI is done.

SWI2, SWI3,  $\overline{\text{IRQ}}$ , Reserved,  $\overline{\text{NMI}}$  — Force a breakpoint and enter the command processor.

The use of  $\overline{\text{IRQ}}$  is recommended as an abort function during program debugging sessions, as breakpoints and other ASSIST09 defaults are reinitialized upon  $\overline{\text{RESET}}$ . Only the primary software interrupt instruction (SWI) is used, not the SWI2 or SWI3. This avoids page fault problems which would otherwise occur with a memory management unit as the SWI2 and SWI3 instructions do not disable interrupts.

Counter number one of the PTM is used to cause an  $\overline{\text{NMI}}$  interrupt for the trace and breakpoint commands. At  $\overline{\text{RESET}}$  the control register for timer one is initialized for tracing purposes. If no tracing or breakpointing is done then the entire PTM is available to the user. Otherwise, only counters two and three are available. Although control register two must be used to initialize control register one, ASSIST09 returns control register two to the same value it has after a  $\overline{\text{RESET}}$  occurs. Therefore, the only condition imposed on a user program is that if the "operate/preset" bit in control register one must be turned on, \$A7 should be stored, \$A6 should be stored if it must be turned off.

## B.4 INITIALIZATION

During ASSIST09 execution, a vector table is used to address certain service routines and default values. This table is generated to provide easily changed control information for user modifications. The first byte of the ASSIST09 ROM contains the start of a subroutine which initializes the vector table along with setting up certain default values before returning to the caller.

If the ASSIST09  $\overline{\text{RESET}}$  vector receives control, it does three things:

1. Assigns a default stack in the work space,
2. Calls the aforementioned subroutine to initialize the vector table, and
3. Fires up the ASSIST09 monitor proper with a MONITR SWI service request.

However, a user routine can perform the same functions with a bonus. After calling the vector initialization subroutine, it may examine or alter any of the vector table values before starting normal ASSIST09 processing. Thus, a user routine may "bootstrap" ASSIST09 and alter the default standard values.

Another method of inserting user modifications is to have a user routine reside at an extension ROM location 2K below the start of the ASSIST09 ROM. The vector table initialization routine mentioned above, looks for a "BRA\*" flag (\$20FE) at this address, and if found calls the location following the flag as a subroutine with the U register pointing to the vector table. Since this is done after vector table initialization, any or all defaults may be altered at this time. A big advantage to using this method is that the modifications are "automatic" in that upon a  $\overline{\text{RESET}}$  condition the changes are made without overt action required such as the execution of a memory change command.

No special stack is used during ASSIST09 processing. This means that the stack pointer must be valid at all interruptable times and should contain enough room for the stacking of at least 21 bytes of information. The stack in use during the initial MONITR service call to start up ASSIST09 processing becomes the "official" stack. If any later stack validity checks occur, this same stack will be re-based before entering the command handler.

ASSIST09 uses a work area which is addressed at an offset from the start of the ASSIST09 ROM. The offset value is –1900 hexadecimal. This points to the base page used during monitor execution and contains the vector table as well as the start of the default stack. If the default stack is used and it exceeds 81 bytes in size, then contiguous RAM must exist below this base work page for proper extension of the stack.

## **B5. INPUT/OUTPUT CONTROL**

Output generated by use of the ASSIST09 services may be halted by pressing any key, causing a 'FREEZE' mode to be entered. The next keyboard entry will release this condition allowing normal output to continue. Commands which generate large amounts of output may be aborted by entering CANCEL (CONTROL-X). User programs may also monitor for CANCEL along with the 'FREEZE' condition even when not performing console I/O (PAUSE service).

## **B.6 COMMAND FORMAT**

There are three possible formats for a command:

< Command > CR

< Command > < Expression1 > CR

< Command > < Expression1 > < Expression2 > CR

The space character is used as the delimiter between the command and all arguments. Two special quick commands need no carriage return, "." and "/". To re-enter a command once a mistake is made, type the CANCEL (CONTROL-X) key.

Each "expression" above consists of one or more values separated by an operator. Values can be hex strings, the letters "P", "M", and "W", or the result of a function. Each hexadecimal string is converted internally to a 16-bit binary number. The letter "P" stands for the current program counter, "M" for the last memory examine/change address, and "W" for the window value. The window value is set by using the WINDOW command.

One function exists and it is the INDIRECT function. The character "@" following a value replaces that value with the 16-bit number obtained by using that value as an address.

Two operators are allowed, "+" and "-" which cause addition and subtraction. Values are operated on in a left-to-right order.

Examples:

480 — hexadecimal 480

W + 3 — value of window plus three

P-200 — current program counter minus 200 hexadecimal

M – W — current memory pointer minus window value

100@ — value of word addressed by the two bytes at 100 hexadecimal

P + 1@ — value addressed by the word located one byte up from the current program counter

## B.7 COMMAND LIST

Table B-1 lists the commands available in the ASSIST09 monitor.

**Table B-1. Command List**

| Command Name | Description                                | Command Entry |
|--------------|--|---------------|
| Breakpoint   | Set, clear, display, or delete breakpoints | B             |
| Call         | Call program as subroutine                 | C             |
| Display      | Display memory block in hex and ASCII      | D             |
| Encode       | Return indexed postbyte value              | E             |
| Go           | Start or resume program execution          | G             |
| Load         | Load memory from tape                      | L             |
| Memory       | Examine or alter memory                    | M             |
|              | Memory change or examine last referenced   | /             |
|              | Memory change or examine                   | hex/          |
| Null         | Set new character and new line padding     | N             |
| Offset       | Compute branch offsets                     | O             |
| Punch        | Punch memory on tape                       | P             |
| Registers    | Display or alter registers                 | R             |
| Stlevel      | Alter stack trace level value              | S             |
| Trace        | Trace number of instructions               | T             |
|              | Trace one instruction                      | .             |
| Verify       | Verify tape to memory load                 | V             |
| Window       | Set a window value                         | W             |

## B.8 COMMANDS

Each of the commands are explained on the following pages. They are arranged in alphabetical order by the command name used in the command list. The command name appears at each margin and in slightly larger type for easy reference.

# BREAKPOINT

# BREAKPOINT

**Format:** Breakpoint  
Breakpoint –  
Breakpoint < Address >  
Breakpoint – < Address >

**Operation:** Set or change the breakpoint table. The first format displays all breakpoints. The second clears the breakpoint table. The third enters an address into the table. The fourth deletes an address from the table. At reset, all breakpoints are deleted. Only instructions in RAM may be breakpointed.

# CALL

# CALL

**Format:** Call  
Call < Address >

**Operation:** Call and execute a user routine as a subroutine. The current program counter will be used unless the address is specified. The user routine should eventually terminate with a “RTS” instruction. When this occurs, a breakpoint will ensue and the program counter will point into the monitor.

# DISPLAY

# DISPLAY

**Format:** Display <From>  
Display <From> <Length>  
Display <From> <To>

**Operation:** Display contents of memory in hexadecimal and ASCII characters. The second argument, when entered, is taken to be a length if it is less than the first, otherwise it is the ending address. A default length of 16 decimal is assumed for the first format. The addresses are adjusted to include all bytes within the surrounding modulo 16 address byte boundary. The CANCEL (CONTROL-X) key may be entered to abort the display. Care must be exercised when the last 15 bytes of memory are to be displayed. The <Length> option should always be used in this case to assure proper termination: D FFE0 40

**Examples:**

- D M 10 — Display 16 bytes surrounding the last memory location examined.
- D E000 F000 — Display memory from E000 to F000 hex.

# ENCODE

# ENCODE

**Format:** Encode <Indexed operand>

**Operation:** The encode command will return the indexing instruction mode postbyte value from the entered assembler-like syntax operand. This is useful when hand coding instructions. The letter "H" is used to indicate the number of hex digits needed in the expression as shown in the following examples:

- E ,Y — Return zero offset to Y register postbyte.
- E [HHHH,PCR] — Return two byte PCR offset using indirection.
- E [,S + +] — Return autoincrement S by two indirect.
- E H,X — Return 5-bit offset from X.

Note that one "H" specifies a 5-bit offset, and that the result given will have zeros in the offset value position. This command does not detect all incorrectly specified syntax or illegal indexing modes.

# GO

# GO

**Format:** Go  
Go < Address >

**Operation:** Execute starting from the address given. The first format will continue from the current program counter setting. If it is a breakpoint no break will be taken. This allows continuation from a breakpoint. The second format will breakpoint if the address specified is in the breakpoint list.

# LOAD

# LOAD

**Format:** Load  
Load < Offset >

**Operation:** Load a tape file created using the S1-S9 format. The offset option, if used, is added to the address on the tape to specify the actual load address. All offsets are positive, but wrap around memory modulo 64K. Depending on the equipment involved, after the load is complete a few spurious characters may still be sent by the input device and interpreted as command characters. If this happens, a CANCEL (CONTROL-X) should be entered to cause such characters to be ignored. If the load was not successful a "?" is displayed.



# MEMORY

# MEMORY

**Format:** MEMORY <Address> /  
<Address> /  
/

**Operation:** Initiate the memory examine/change function. The second format will not accept an expression for the address, only a hex string. The third format defaults to the address displayed during the last memory change/examine function. (The same value is obtained in expressions by use of the letter "M".) After activation, the following actions may be taken until a carriage return is entered:

|          |   |
|----------|---|
| < Expr > | Replaces the byte with the specified value. The value may be an expression.                               |
| SPACE    | Go to next address and print the byte value.  |
| ,        | (Comma) Go to next address without printing the byte value.   |
| LF       | (Line feed) Go to next address and print it along with the byte value on the next line.                   |
| ^        | (Circumflex or Up arrow) Go the previous address and print it along with the byte value on the next line. |
| /        | Print the current address with the byte value on the next line.   |
| CR       | (Carriage return) Terminate the command.  |
| '<Text>' | Replace succeeding bytes with ASCII characters until the second apostrophe is entered.                    |

If a change attempt fails (i.e., the location is not valid RAM) then a question mark will appear and the next location displayed.

# NULL

# NULL

**Format:** Null <Specification>

**Operation:** Set the new line and character padding count values. The expression value is treated as two values. The upper two hex represent the character pad count, and the lower two the new line pad count (triggered by a carriage return). An expression of less than three hex digits will set the character pad count to zero. The values must range from zero to 7F hexadecimal (127 decimal).

Example:

- N 3 — Set the character count to zero and new line count to three.
- N 207 — Set character padding count to two and new line count to seven.

Settings for TI Silent 700 terminals are:

| Baud | Setting |
|------|---------|
| 100  | 0       |
| 300  | 4       |
| 1200 | 317     |
| 2400 | 72F     |

# OFFSET

# OFFSET

**Format:** Offset <Offset addr> <To instruction> .

**Operation:** Print the one and two byte offsets needed to perform a branch from the first expression to the instruction. Thus, offsets for branches as well as indexed mode instructions which use offsets may be obtained. If only a four byte value is printed, then a short branch count cannot be done between the two addresses.

Example:

- 0 P+2 A000 — Compute offsets needed from the current program counter plus two to A000.

# PUNCH

# PUNCH

**Format:** Punch <From> <To>

**Operation:** Punch or record formatted binary object tape in S1-S9 (MIKBUG) format.

# REGISTER

# REGISTER

**Format:** Register

**Operation:** Print the register set and prompt for a change. At each prompt the following may be entered.

|               |   |
|---------------|---|
| SPACE         | Skip to the next register prompt  |
| < Expr> SPACE | Replace with the specified value and prompt for the next register.            |
| < Expr> CR    | (carriage return) Replace with the specified value and terminate the command. |
| CR            | Terminate the command.  |

# STLEVEL

# STLEVEL

**Format:** Stlevel  
Stlevel <Address>

**Operation:** Set the stack trace level for inhibiting tracing information. As long as the stack is at or above the stack level address, the trace display will continue. However, when lower than the address it is inhibited. This allows tracing of a routine without including all subroutine and lower level calls in the trace information. Note that tracing through a ASSIST09 "SWI" service request may also temporarily suppress trace output as explained in the description of the trace command. The first format sets the stack trace level to the current program stack value.

# TRACE

# TRACE

**Format:** Trace <Count>  
. (period)

**Operation:** Trace the specified number of instructions. At each trace, the opcode just executed will be shown along with the register set. The program counter in the register display points to the NEXT instruction to be executed. A CANCEL (CONTROL-X) will prematurely halt tracing. The second format (period) will cause a single trace to occur. Breakpoints have no effect during the trace. Selected portions of a trace may be disabled using the STLEVEL command. Instructions in ROM and RAM may be traced, whereas breakpoints may be done only in RAM. When tracing through a ASSIST09 service request, the trace display will be suppressed starting two instructions into the monitor until shortly before control is returned to the user program. This is done to avoid an inordinate amount of displaying because ASSIST09, at times, performs a sizeable amount of processing to provide the requested services.

# VERIFY

# VERIFY

**Format:** Verify  
Verify <Offset>

**Operation:** Verify or compare the contents of memory to the tape file. This command has the same format and operation as a LOAD command except the file is compared to memory. If the verify fails for any reason a “?” is displayed.

# WINDOW

# WINDOW

**Format:** Window <Value>

**Operation:** Set the window to a value. This value may be referred to when entering expressions by use of the letter “W”. The window may be set to any 16-bit value.

## B.9 SERVICES

The following describes services provided by the ASSIST09 monitor. These services are invoked by using the "SWI" instruction followed by a one byte function code. All services are designed to allow complete address independence both in invocation and operation. Unless specified otherwise, all registers are transparent over the "SWI" call. In the following descriptions, the terms "input handler" and "output handler" are used to refer to appendage routines which may be replaced by the user. The default routines perform standard I/O through an ACIA for console operations to a terminal. The ASCII CANCEL code can be entered on most terminals by depressing the CONTROL and X keys simultaneously. A list of services is given in Table B-2.

**Table B-2. Services**

| Service                  | Entry  | Code | Description   |
|--------------------------|--------|------|---|
| Obtain input character   | INCHP  | 0    | Obtain the input character in register A from the input handler                   |
| Output a character       | OUTCH  | 1    | Send the character in the register A to the output handler                        |
| Send string              | PDATA1 | 2    | Send a string of characters to the output handler                                 |
| Send new line and string | PDATA  | 3    | Send a carriage return, line feed, and string of characters to the output handler |
| Convert byte to hex      | OUT2HS | 4    | Display the byte pointed to by the X register in hex                              |
| Convert word to hex      | OUT4HS | 5    | Display the word pointed to by the X register in hex                              |
| Output to next line      | PCRLF  | 6    | Send a carriage return and line feed to the output handler                        |
| Send space               | SPACE  | 7    | Send a blank to the output handler  |
| Fireup ASSIST09          | MONITR | 8    | Enter the ASSIST09 monitor  |
| Vector swap              | VCTRSW | 9    | Examine or exchange a vector table entry  |
| User breakpoint          | BRKPT  | 10   | Display registers and enter the command handler                                   |
| Program break and check  | PAUSE  | 11   | Stop processing and check for a freeze or cancel condition                        |

# BRKPT

## User Breakpoint

# BRKPT

**Code:** 10

**Arguments:** None

**Result:** A disabled breakpoint is taken. The registers are displayed and the command handler of ASSIST09 is entered.

**Description:** Establishes user breakpoints. Both SWI2 and SWI3 default appendages cause a breakpoint as well, but do not set the I and F mask bits. However, since they may both be replaced by user routines the breakpoint service always ensures breakpoint availability. These user breakpoints have nothing to do with system breakpoints which are handled differently by the ASSIST09 monitor.

**Example:** BRKPT EQU 10 INPUT CODE FOR BRKPT  
SWI REQUEST SERVICE  
FCB BRKPT FUNCTION CODE BYTE

# INCHP

## Obtain Input Character

# INCHP

**Code:** 0

**Arguments:** None

**Result:** Register A contains a character obtained from the input handler.

**Description:** Control is not returned until a valid input character is received from the input handler. The input character will have its parity bit (bit 7) stripped and forced to a zero. All NULL (\$00) and RUBOUT (\$7F) characters are ignored and not returned to the caller. The ECHO flag, which may be changed by the vector SWAP service, determines whether or not the input character is echoed to the output handler (full duplex operation). The default at reset is to echo input. When a carriage return (\$0D) is received, line feed (\$A0) is automatically sent back to the output handler.

**Example:** INCHNP EQU 0 INPUT CODE FOR INCHP  
SWI PERFORM SERVICE CALL  
FCB INCHNP FUNCTION FOR INCHNP

A REGISTER NOW CONTAINS NEXT CHARACTER

# MONITR

## Startup ASSIST09

# MONITR

**Code:** 8

**Arguments:** S → Stack to become the “official” stack  
DP → Direct page default for executed user programs  
A = 0 Call input and output console initialization handlers and give the  
“ASSIST09” startup message  
A#0 Go directly to the command handler

**Result:** ASSIST09 is entered and the comand handler given control

**Description:** The purpose for this function is to enter ASSIST09, either after a system reset, or when a user program desires to terminate. Control is not returned unless a “GO” or “CALL” command is done without altering the program counter. ASSIST09 runs on the passed stack, and if a stack error is detected during user program execution this is the stack that is rebased. The direct page register value in use remains the default for user program execution.

The ASSIST09 restart vector routine uses this function to startup monitor processing after calling the vector build subroutine as explained in INITIALIZATION.

If indicated by the A register, the input and output initialization handlers are called followed by the sending of the string “ASSIST09” to the output handler. The programmable timer (PTM) is initialized, if its address is not zero, such that register 1 can be used for causing an NMI during trace commands. The command handler is then entered to perform the command request prompt.

|                 |                 |  |
|-----------------|-----------------|--|
| <b>Example:</b> | MONITR EQU 8    | INPUT CODE FOR MONITR                                      |
|                 | LOOP CLRA       | PREPARE ZERO PAGE REGISTER AND<br>INITIALIZATION PARAMETER |
|                 | *               |  |
|                 | TFR A,DP        | SET DEFAULT PAGE VALUE                                     |
|                 | LEAS STACK, PCR | SETUP DEFAULT STACK VALUE                                  |
|                 | SWI             | REQUEST SERVICE  |
|                 | FCB MONITR      | FUNCTION CODE BYTE   |
|                 | BRA LOOP        | REENTER IF FALLOUT OCCURS                                  |



# OUTCH

Output a Character

# OUTCH

**Code:** 1

**Arguments:** Register A contains the byte to transmit.

**Result:** The character is sent to the output handler  
The character is set as follows ONLY if a LINEFEED was the character to transmit:  
CC = 0 if normal output occurred.  
CC = 1 if CANCEL was entered during output.

**Description:** If a FREEZE Occurs (any input character is received) then control is not returned to the user routine until the condition is released. The FREEZE condition is checked for only when a linefeed is being sent. Padding null characters (\$00) may be sent following the outputted character depending on the current setting of the NULLS command. For DLE (Data Link Escape), character nulls are never sent. Otherwise, carriage returns (\$00) receive the new line count of nulls, all other characters the character count of nulls.

**Example:**

|       |     |       |                            |
|-------|-----|-------|----------------------------|
| OUTCH | EQU | 1     | INPUT CODE FOR OUTCH       |
|       | LDA | #'0   | LOAD CHARACTER "0"         |
|       | SWI |       | SEND OUT WITH MONITOR CODE |
|       | FCB | OUTCH | SERVICE CODE BYTE          |

# OUT2HS

Convert Byte to Hex

# OUT2HS

**Code:** 4

**Arguments:** Register X points to a byte to display in hex.

**Result:** The byte is converted to two hex digits and sent to the output handler followed by a blank.

**Example:**

|        |      |           |                           |
|--------|------|-----------|---------------------------|
| OUT2HS | EQU  | 4         | INPUT CODE FOR OUT2HS     |
|        | LEAX | DATA, PCR | POINT TO 'DATA' TO DECODE |
|        | SWI  |           | REQUEST SERVICE           |
|        | FCB  | OUT2HS    | SERVICE CODE BYTE         |

# OUT4HS

Convert Word to Hex

# OUT4HS

**Code:** 5

**Arguments:** Register X points to a word (two bytes) to display in hex.

**Result:** The word is converted to four hex digits and sent to the output handler followed by a blank.

**Example:** OUT4HS EQU 5 INPUT CODE FOR OUT4HS  
LEAX DATA, PCR LOAD 'DATA' ADDRESS TO DECODE  
SWI REQUEST ASSIST09 SERVICE  
FCB OUT4HS SERVICE CODE BYTE

# PAUSE

Program Break and Check

# PAUSE

**Code:** 11

**Arguments:** None

**Result:** CC = 0 For a normal return.  
CC = 1 If a CANCEL was entered during the interim.

**Description:** The PAUSE service should be used whenever a significant amount of processing is done by a program without any external interaction (such as console I/O). Another use of the PAUSE service is for the monitoring of FREEZE or CANCEL requests from the input handler. This allows multi-tasking operating systems to receive control and possibly re-dispatch other programs in a timeslice-like fashion. Testing for FREEZE and CANCEL conditions is performed before return. Return may be after other tasks have had a chance to execute, or after a FREEZE condition is lifted. In a one task system, return is always immediate unless a FREEZE occurs.

# PCRLF

Output to Next Line

# PCRLF

**Code:** 6

**Arguments:** None

**Result:** A carriage return and line feed are sent to the output handler.  
C = 1 if normal output occurred.  
C = 1 if CONTROL-X was entered during output.

**Description:** If a FREEZE occurs (any input character is received), then control is not returned to the user routine until the condition is released. The string is completely sent regardless of any FREEZE or CANCEL events occurring. Padding characters may be sent as described under the OUTCH service.

**Example:** PCRLF EQU 6 INPUT CODE PCRLF  
SWI REQUEST SERVICE  
FCB PCRLF SERVICE CODE BYTE

# PDATA

Send New Line and String

# PDATA

**Code:** 3

**Arguments:** Register X points to an output string terminated with an ASCII EOT (\$04).

**Result:** The string is sent to the output handler following a carriage return and line feed.  
CC = 0 if normal output occurred.  
CC = 1 if CONTROL-X was entered during output.

**Description:** The output string may contain embedded carriage returns and line feeds thus allowing several lines of data to be sent with one function call. If a FREEZE occurs (any input character is received), then control is not returned to the user routine until the condition is released. The string is completely sent regardless of any FREEZE or CANCEL events occurring. Padding characters may be sent as described by the OUTCH function.

# PDATA

## Send New Line and String (Continued)

# PDATA

**Example:** PDATA EQU 3 INPUT CODE FOR PDATA

MSGOUT FCC 'THIS IS A MULTIPLE LINE MESSAGE.'  
FCB \$0A, \$0D LINE FEED, CARRIAGE RETURN  
FCC 'THIS IS THE SECOND LINE.'  
FCB \$04 STRING TERMINATOR

LEAX MSGOUT, PCR LOAD MESSAGE ADDRESS  
SWI REQUEST A SERVICE  
FCB PDATA SERVICE CODE BYTE

# PDATA1

## Send String

# PDATA1

**Code:** 2

**Arguments:** Register X points to an output string terminated with an ASCII EOT (\$04).

**Result:** The string is sent to the output handler.  
CC = 0 if normal output occurred.  
CC = 1 if CONTROL-X was entered during output.

**Description:** The output string may contain embedded carriage returns and line feeds thus allowing several lines of data to be sent with one function call. If a FREEZE occurs (any input character is received), then control is not returned to the user routine until the condition is released. The string is completely sent regardless of any FREEZE or CANCEL events occurring. Padding characters may be sent as described by the OUTCH function.

**Example:** PDATA EQU 2 INPUT CODE FOR PDATA1

MSG FCC 'THIS IS AN OUTPUT STRING'  
FCB \$04 STRING TERMINATOR

LEAX MSG, PCR LOAD 'MSG' STRING ADDRESS  
SWI REQUEST A SERVICE  
FCB PDATA1 SERVICE CODE BYTE

# SPACE

## Single Space Output

# SPACE

**Code:** 7

**Arguments:** None

**Result:** A space is sent to the output handler.

**Description:** Padding characters may be sent as described under the OUTCH service.

**Example:** SPACE EQU 7 INPUT CODE SPACE  
 SWI REQUEST ASSIST09 SERVICE  
 FCB SPACE SERVICE CODE BYTE

# VCTRSW

## Vector Swap

# VCTRSW

**Code:** 9

**Arguments:** Register A contains the vector swap input code.  
Register X contains zero or a replacement value.

**Result:** Register X contains the previous value for the vector.

**Description:** The vector swap service examines/alters a word entry in the ASSIST09 vector table. This table contains pointers and default values used during monitor processing. The entry is replaced with the value contained in the X register unless it is zero. The codes available are listed in Table B-3.

**Example:** VCTRSW EQU 9 INPUT CODE VCTRSW  
 .IRQ EQU 12 IRQ APPENDAGE SWAP FUNCTION  
 CODE  
 LEAX MYIRQH,PCR LOAD NEW IRQ HANDLER ADDRESS  
 LDA #.IRQ LOAD SUBCODE FOR VECTOR SWAP  
 SWI REQUEST SERVICE  
 FCB VCTRSW SERVICE CODE BYTE  
 X NOW HAS THE PREVIOUS APPENDAGE ADDRESS

## B.10 VECTOR SWAP SERVICE

The vector swap service allows user modifications of the vector table to be easily installed. Each vector handler, including the one for SWI, performs a validity check on the stack before any other processing. If the stack is not pointing to valid RAM, it is reset to the initial value passed to the MONITR request which fired-up ASSIST09 after RESET. Also, the current register set is printed following a “?” (question mark) and then the command handler is entered. A list of each entry in the vector table is given in Table B-3.

**Table B-3. Vector Table Entries**

| Entry  | Code | Description                                     |
|--------|------|---|
| .AVTBL | 0    | Returns address of vector table                 |
| .CMDL1 | 2    | Primary command list                            |
| .RSVD  | 4    | Reserved MC6809 interrupt vector appendage      |
| .SWI3  | 6    | Software interrupt 3 interrupt vector appendage |
| .SWI2  | 8    | Software interrupt 2 interrupt vector appendage |
| .FIRQ  | 10   | Fast interrupt request vector appendage         |
| .IRQ   | 12   | Interrupt request vector appendage              |
| .SWI   | 14   | Software interrupt vector appendage             |
| .NMI   | 16   | Non-maskable interrupt vector appendage         |
| .RESET | 18   | Reset interrupt vector appendage                |
| .CION  | 20   | Input console initialization routine            |
| .CIDTA | 22   | Input data byte from console routine            |
| .CIOFF | 24   | Input console shutdown routine                  |
| .COON  | 26   | Output console initialization routine           |
| .CODTA | 28   | Output/data byte to console routine             |
| .COOFF | 30   | Output console shutdown routine                 |
| .HSDTA | 32   | High speed display handler routine              |
| .BSON  | 34   | Punch/load initialization routine               |
| .BSDTA | 36   | Punch/load handler routine                      |
| .BSOFF | 38   | Punch/load shutdown routine                     |
| .PAUSE | 40   | Processing pause routine                        |
| .CMDL2 | 44   | Secondary command list                          |
| .ACIA  | 46   | Address of ACIA                                 |
| .PAD   | 48   | Character and new line pad counts               |
| .ECHO  | 50   | Echo flag                                       |
| .PTM   | 52   | Programmable timer module address               |

The following pages describe the purpose of each entry and the requirements which must be met for a user replaceable value or routine to be successfully substituted.

# **.ACIA**

ACIA Address

# **.ACIA**

**Code:** 46

**Description:** This entry contains the address of the ACIA used by the default console input and output device handlers. Standard ASSIST09 initialization sets this value to hexadecimal E008. If this must be altered, then it must be done before the MONITR startup service is invoked, since that service calls the .COON and .COIN input and output device initialization routines which initialize the ACIA pointed to by this vector slot.

# **.AVTBL**

Return Address of Vector Table

# **.AVTBL**

**Code:** 0

**Description:** The address of the vector table is returned with this code. This allows mass changes to the table without individual calls to the vector swap service. The code values are identical to the offsets in the vector table. This entry should never be changed, only examined.

# **.BSDTA**

## **Punch/Load Handler Routine**

# **.BSDTA**

**Code:** 36

**Description:** This entry contains the address of a routine which performs punch, load, and verify operations. The .BSON routine is always executed before the routine is given control. This routine is given the same parameter list documented for .BSON. The default handler uses the .CODTA routine to punch or the .CIDTA routine to read data in S1/S9 (MIKBUG) format. The function code byte must be examined to determine the type request being handled.

A return code must be given which reflects the final processing disposition:

Z = 1 Successful completion

or

Z = 0 Unsuccessful completion.

The .BSOFF routine will be called after this routine is completed.

# **.BSOFF**

## **Punch/Load Shutdown Routine**

# **.BSOFF**

**Code:** 38

**Description:** This entry points to a subroutine which is designated to terminate device processing for the punch, load, and verify handler .BSDTA. The stack contains a parameter list as documented for the .BSON entry. The default ASSIST09 routine issues DC4 (\$14 or stop) and DC3 (\$13 or x-off) followed by a one second delay to give the reader/punch time to stop. Also, an internally used flag by the INCHP service routine is cleared to reverse the effect caused by its setting in the .BSON handler. See that description for an explanation of the proper use of this flag.



# .BSON

## Punch/Load Initialization Routine

# .BSON

**Code:** 34

**Description:** This entry points to a subroutine with the assigned task of turning on the device used for punch, load, and verify processing. The stack contains a parameter list describing which function is requested. The default routine sends an ASCII "reader on" or "punch on" code of DC1 (\$11) or DC2 (\$12) respectively to the output handler (.CODTA). A flag is also set which disables test for FREEZE conditions during INCHNP processing. This is done so characters are not lost by being interpreted as FREEZE mode indicators. If a user replacement routine also uses the INCHNP service, then it also should set this same byte non-zero and clear it in the .BSOFF routine. The ASSIST09 source listing should be consulted for the location of this byte.

The stack is setup as follows:

S + 6 = Code byte, VERIFY (- 1), PUNCH (0), LOAD (1)

S + 4 = Start address for punch only

S + 2 = End address for punch, or offset for READ/LOAD

S + 0 = Return address

# .CIDTA

## Input Data Byte from Console Routine

# .CIDTA

**Code:** 22

**Description:** This entry determines the console input handler appendage. The responsibility of this routine is to furnish the requested next input character in the A register, if available, and return with a condition code. The INCHP service routine calls this appendage to supply the next character. Also, a "FREEZE" mode routine calls at various times to test for a FREEZE condition or determine if the CANCEL key has been entered. Processing for this appendage must abide by the following conventions:

**Input:** PC → ASSIST09 work page

S → Return address

**Output:** C = 0, A = input character

C = 1 if no input character is yet available

**Volatile Registers:** U, B

The handler should always pass control back immediately even if no character is yet available. This enables other tasks to do productive work while input is unavailable. The default routine reads an ACIA as explained in Paragraph B.2 Implementation Requirements.

# **.CIOFF**

## **Input Console Shutdown Routine**

# **.CIOFF**

**Code:** 24

**Description:** This entry points to a routine which is called to terminate input processing. It is not called by ASSIST09 at any time, but is included for consistency. The default routine merely does an "RTS". The environment is as follows:

**Input:** None  
**Output:** Input device terminated  
**Volatile Registers:** None

# **.CION**

## **Input Console Initialization Routine**

# **.CION**

**Code:** 20

**Description:** This entry is called to initiate the input device. It is called once during the MONITR service which initializes the monitor so the command processor may obtain commands to process. The default handler resets the ACIA used for standard input and output and sets up the following default conditions: 8-bit word length, no parity checking, 2 stop bits, divide-by-16 counter ratio. The effect of an 8-bit word with no parity checking is to accept 7-bit ASCII and ignore the parity bit.

**Input:** .ACIA Memory address of the ACIA  
**Output:** The output device is initialized  
**Volatile Registers:** A, X

# .CMDL1

## Primary Command List

# .CMDL1

**Code:** 2

**Description:** User supplied command tables may either substitute or replace the ASSIST09 standard tables. The command handler scans two lists, the primary table first followed by the secondary table. The primary table is pointed to by this entry and contains, as a default, the ASSIST09 command table. The secondary table defaults to a null list. A user may insert their own table into either position. If a user list is installed in the secondary table position, then the ASSIST09 list will be searched first. The default ASSIST09 list contains all one character command names. Thus, a user command "PRINT" would be matched if the letters "PR" are typed, but not just a "P" since the system command list would match first. A user may replace the primary system list if desired. A command is chosen on a first match basis comparing only the character(s) entered. This means that two or more commands may have the same initial characters and that if only that much is entered then the first one in the list(s) is chosen.

Each entry in the users command list must have the following format:

|     |     |            |  |
|-----|-----|------------|--|
| + 0 | FCB | L          | Where "L" is the size of the entry including this byte                     |
| + 1 | FCC | '<string>' | Where "<string>" is the command name                                       |
| + N | FDB | EP - *     | Where "EP" represents the symbol defining the start of the command routine |

The first byte is an entry length byte and is always three more than the length of the command string (one for the length itself plus two for the routine offset). The command string must contain only ASCII alphanumeric characters, no special characters. An offset to the start of the command routine is used instead of an absolute address so that position-independent programs may contain command tables. The end of the command table is a one byte flag. A -1 (\$FF) specifies that the secondary table is to be searched, or a -2 (\$FE) that command list searching is to be terminated. The table represented as the secondary command list must end with -2. The first list must end with a -1 if both lists are to be searched, or a -2 if only one list is to be used.

A command routine is entered with the following registers set:

|      |  |
|------|--|
| DPR→ | ASSIST09 page work area.                       |
| S→   | A return address to the command processor.     |
| Z=1  | A carriage return terminated the command name. |
| Z=0  | A space delimiter followed the command name.   |

# .CMDL1

## Primary Command List (Continued)

# .CMDL1

A command routine is entered after the delimiter following the command name is typed in. This means that a carriage return may be the delimiter entered with the input device resting on the next line. For this reason the Z bit in the condition code is set so the command routine may determine the current position of the input device. The command routine should ensure that the console device is left on a new line before returning to the command handler.

# .CMDL2

## Secondary Command List

# .CMDL2

**Code:** 44

**Description:** This entry points to the second list table. The default is a null list followed by a byte of – 2. A complete explanation of the use for this entry is provided under the description of the .CMDL1 entry.

# .CODTA

## Output Data Byte to Console Routine

# .CODTA

**Code:** 28

**Description:** The responsibility of this handler is to send the character in the A register to the output device. The default routine also follows with padding characters as explained in the description of the OUTCH service. If the output device is not ready to accept a character, then the “pause” subroutine should be called repeatedly while this condition lasts. The address of the pause routine is obtained from the .PAUSE entry in the vector table. The character counts for padding are obtained from the .PAD entry in the table. All ASSIST09 output is done with a call to this appendage. This includes punch processing as well. The default routine sends the character to an ACIA as explained in Paragraph B.2 Implementation Requirements. The operating environment is as follows:

|                            |   |
|----------------------------|---|
| <b>Input:</b>              | A = Character to send<br>DP = ASSIST09 work page<br>.PAD = Character and new line padding counts<br>(in vector table)<br>.PAUSE = Pause routine (in vector table) |
| <b>Output:</b>             | Character sent to the output device   |
| <b>Volatile Registers:</b> | None. All work registers must be restored   |

## **.COOFF**

Output Console Shutdown Routine

## **.COOFF**

**Code:** 30

**Description:** This entry addresses the routine to terminate output device processing. ASSIST09 does not call this routine. It is included for completeness. The default routine is an “RTS”.

**Input:** DP→ ASSIST09 work page  
**Output:** The output device is terminated  
**Volatile Registers:** None

## **.COON**

Output Console Initialization Routine

## **.COON**

**Code:** 26

**Description:** This entry points to a routine to initialize the standard output device. The default routine initializes an ACIA and is the very same one described under the .CION vector swap definition.

**Input:** .ACIA vector entry for the ACIA address  
**Output:** The output device is initialized  
**Volatile Registers:** A, X

## **.ECHO**

Echo Flag

## **.ECHO**

**Code:** 50

**Description:** The first byte of this word is used as a flag for the INCHP service routine to determine the requirement of echoing input received from the input handler. A non-zero value means to echo the input; zero not to echo. The echoing will take place even if user handlers are substituted for the default .CIDTA handler as the INCHP service routine performs the echo.

## **.FIRQ**

Fast Interrupt Request Vector Appendage

## **.FIRQ**

**Code:** 10

**Description:** The fast interrupt request routine is located via this pointer. The MC6809 addresses hexadecimal FFF6 to locate the handler when processing a  $\overline{\text{FIRQ}}$ . The stack and machine status is as defined for the  $\overline{\text{FIRQ}}$  interrupt upon entry to this appendage. It should be noted that this routine is "jumped" to with an indirect jump instruction which adds eleven cycles to the interrupt time before the handler actually receives control. The default handler does an immediate "RTI" which, in essence, ignores the interrupt.

# .HSDTA

## High Speed Display Handler Routine

# .HSDTA

**Code:** 32

**Description:** This entry is invoked as a subroutine by the DISPLAY command and passed a parameter list containing the "TO" and "FROM" addresses. The from value is rounded down to a 16 byte address boundary. The default routine displays memory in both hexadecimal and ASCII representations, with a title produced on every 128 byte boundary. The purpose for this vector table entry is for easy implementation of a user routine for special purpose handling of a block of data. (The data could, for example, be sent to a high speed printer for later analysis.) The parameters are all passed on the stack. The environment is as follows:

**Input:** S + 4 = Start address  
S + 2 = Stop address  
S + 0 = Return Address  
DP → ASSIST09 work page

**Output:** Any purpose desired

**Volatile Registers:** X, D

# .IRQ

## Interrupt Request Vector Appendage

# .IRQ

**Code:** 12

**Description:** All interrupt requests are passed to the routine pointed to by this vector. Hexadecimal FFF8 is the MC6809 location where this interrupt vector is fetched. The stack and processor status is that defined for the  $\overline{\text{IRQ}}$  interrupt upon entry to the handler. Since the routine's address is in the vector table, an indirect jump must be done to invoke it. This adds eleven cycles to the interrupt time before the  $\overline{\text{IRQ}}$  handler receives control. The default  $\overline{\text{IRQ}}$  handler prints the registers and enters the ASSIST09 command handler.

## **.NMI**

### **Non-Maskable Interrupt Vector Appendage**

## **.NMI**

**Code:** 16

**Description:** This entry points to the non-maskable interrupt handler to receive control whenever the processor branches to the address at hexadecimal FFFC. Since ASSIST09 uses the  $\overline{\text{NMI}}$  interrupt during trace and breakpoint processing, such commands should not be used if a user handler is in control. This is true unless the user handler has the intelligence to forward control to the default handler if the  $\overline{\text{NMI}}$  interrupt has not been generated due to user facilities. The  $\overline{\text{NMI}}$  handler given control will have an eleven cycle overhead as its address must be fetched from the vector table.

## **.PAD**

### **Character and New Line Pad Count**

## **.PAD**

**Code:** 48

**Description:** This entry contains the pad count for characters and new lines. The first of the two bytes is the count of nulls for other characters, and the second is the number of nulls (\$00) to send out after any line feed is transmitted. The ASCII Escape character (\$10) never has nulls sent following it. The default .CODTA handler is responsible for transmitting these nulls. A user handler may or may not use these counts as required.

The "NULLS" command also sets these two bytes with user specified values.



# .PAUSE

## Processing Pause Routine

# .PAUSE

**Code:** 40

**Description:** In order to support real-time (also known as multi-tasking) environments ASSIST09 calls a dead-time routine whenever processing must wait for some external change of state. An example would be when the OUTCH service routine attempts the sending of a character to the ACIA through the default .CODTA handler and the ACIA status registers shows that it cannot yet be accepted. The default dead-time routine resides in a reserved four byte area which contains the single instruction, "RTS". The .PAUSE vector entry points to this routine after standard initialization. This pointer may be changed to point to a user routine which dispatches other programs so that the MC6809 may be utilized more efficiently. Another example of use would be to increment a counter so that dead-time cycle counts may be accumulated for statistical or debugging purposes. The reason for the four byte reserved area (which exists in the ASSIST09 work page) is so other code may be overlaid without the need for another space in the address map to be assigned. For example, a master monitor may be using a memory management unit to assign a complete 64K block of memory to ASSIST09 and the programs being executed/tested under ASSIST09 control. The master monitor wishes, or course, to be reentered when any "dead time" occurs, so it overlays the default routine ("RTS") with its own "SWI". Since the master monitor would be "front ending" all "SWI's" anyway, it knows when a "pause" call is being performed and can redispach other systems on a time-slice basis.

All registers must be transparent across the pause handler. Along with selected points in ASSIST09 user service processing, there is a special service call specifically for user programs to invoke the pause routine. It may be suggested that if no services are being requested for a given time period (say 10 ms) user programs should call the .PAUSE service routine so that fair-task dispatching can be guaranteed.

# .PTM

## Programmable Timer Module Address

# .PTM

**Code:** 53

**Description:** This entry contains the address of the MC6840 programmable timer module (PTM). Alteration of this slot should occur before the MONITR startup service is called as explained in Paragraph B.4 Initialization. If no PTM is available, then the address should be changed to a zero so that no initialization attempt will take place. Note that if a zero is supplied, ASSIST09 Breakpoint and Trace commands should not be issued.

## **.RESET**

### **Reset Interrupt Vector Appendage**

## **.RESET**

**Code:** 18

**Description:** This entry returns the address of the RESET routine which initializes ASSIST09. Changing it has no effect, but it is included in the vector table in case a user program wishes to determine where the ASSIST09 restart code resides. For example, if ASSIST09 resides in the memory map such that it does not control the MC6809 hardware vectors, a user routine may wish to start it up and thus need to obtain the standard RESET vector code address. The ASSIST09 reset code assigns the default in the work page, calls the vector build subroutine, and then starts ASSIST09 proper with the MONITR service call.

## **.RSVD**

### **Reserved MC6809 Interrupt Vector Appendage**

## **.RSVD**

**Code:** 4

**Description:** This is a pointer to the reserved interrupt vector routine addressed at hexadecimal FFF0. This MC6809 hardware vector is not defined as yet. The default routine setup by ASSIST09 will cause a register display and entrance to the command handler.

# **.SWI**

## **Software Interrupt Vector Appendage**

# **.SWI**

**Code:** 14

**Description:** This vector entry contains the address of the Software Interrupt routine. Normally, ASSIST09 handles these interrupts to provide services for user programs. If a user handler is in place, however, these facilities cannot be used unless the user routine “passes on” such requests to the ASSIST09 default handler. This is easy to do, since the vector swap function passes back the address of the default handler when the switch is made by the user. This “front ending” allows a user routine to examine all service calls, or alter/replace/extend them to his requirements. Of course, the registers must be transparent across the transfer of control from the user to the standard handler. A “JMP” instruction branches directly to the routine pointed to by this vector entry when a SWI occurs. Therefore, the environment is that as defined for the “SWI” interrupt.

# **.SWI2**

## **Software Interrupt 2 Vector Appendage**

# **.SWI2**

**Code:** 8

**Description:** This entry contains a pointer to the SWI2 handler entered whenever that instruction is executed. The status of the stack and machine are those defined for the SWI2 interrupt which has its interrupt vector address at FFF4 hexadecimal. The default handler prints the registers and enters the ASSIST09 command handler.

# .SWI3

## Software Interrupt 3 Vector Appendage

# .SWI3

**Code:** 6

**Description:** This entry contains a pointer to the SWI3 handler entered whenever that instruction is executed. The status of the stack and machine are those defined for the SWI3 interrupt which has its interrupt vector address located at hexadecimal FFF2. The default handler prints the registers and enters the ASSIST09 command handler.

## B.11 MONITOR LISTING

The following pages contain a listing of the ASSIST09 monitor.

---

PAGE 001 ASSIST09.SA:0

ASSIST09 - MC6809 MONITOR

00001 TTL ASSIST09 - MC6809 MONITOR  
00002 OPT ABS,LLE=85,S,CRE

00004 \*\*\*\*\*  
00005 \* COPYRIGHT (C) MOTOROLA, INC. 1979 \*  
00006 \*\*\*\*\*

00008 \*\*\*\*\*  
00009 \* THIS IS THE BASE ASSIST09 ROM.  
00010 \* IT MAY RUN WITH OR WITHOUT THE  
00011 \* EXTENSION ROM WHICH  
00012 \* WHEN PRESENT WILL BE AUTOMATICALLY  
00013 \* INCORPORATED BY THE BLDVTR  
00014 \* SUBROUTINE.  
00015 \*\*\*\*\*

00017 \*\*\*\*\*  
00018 \* GLOBAL MODULE EQUATES  
00019 \*\*\*\*\*  
00020 F800 A ROMBEG EQU \$F800 ROM START ASSEMBLY ADDRESS  
00021 E700 A RAMOFS EQU -\$1900 ROM OFFSET TO RAM WORK PAGE  
00022 0800 A ROMSIZ EQU 2048 ROM SIZE  
00023 F000 A ROM2OF EQU ROMBEG-ROMSIZ START OF EXTENSION ROM  
00024 E008 A ACIA EQU \$E008 DEFAULT ACIA ADDRESS  
00025 E000 A PTM EQU \$E000 DEFAULT PTM ADDRESS  
00026 0000 A DFTCHP EQU 0 DEFAULT CHARACTER PAD COUNT  
00027 0005 A DFTNLP EQU 5 DEFAULT NEW LINE PAD COUNT  
00028 003E A PROMPT EQU '>' PROMPT CHARACTER  
00029 0008 A NUMBKP EQU 8 NUMBER OF BREAKPOINTS  
00030 \*\*\*\*\*

00032 \*\*\*\*\*  
00033 \* MISCELANEOUS EQUATES  
00034 \*\*\*\*\*  
00035 0004 A EOT EQU \$04 END OF TRANSMISSION  
00036 0007 A BELL EQU \$07 BELL CHARACTER  
00037 000A A LF EQU \$0A LINE FEED  
00038 000D A CR EQU \$0D CARRIAGE RETURN  
00039 0010 A DLE EQU \$10 DATA LINK ESCAPE  
00040 0018 A CAN EQU \$18 CANCEL (CTL-X)  
00041 \* PTM ACCESS DEFINITIONS  
00042 E001 A PTMSTA EQU PTM+1 READ STATUS REGISTER  
00043 E000 A PTMC13 EQU PTM CONTROL REGISTERS 1 AND 3  
00044 E001 A PTMC2 EQU PTM+1 CONTROL REGISTER 2  
00045 E002 A PTMTM1 EQU PTM+2 LATCH 1  
00046 E004 A PTMTM2 EQU PTM+4 LATCH 2  
00047 E006 A PTMTM3 EQU PTM+6 LATCH 3  
  
00049 008C A SKIP2 EQU \$8C "CMPX #" OP CODE - SKIPS TWO BYTES

00051 \*\*\*\*\*  
00052 \* ASSIST09 MONITOR SWI FUNCTIONS

---

```
00053          * THE FOLLOWING EQUATES DEFINE FUNCTIONS PROVIDED
00054          * BY THE ASSIST09 MONITOR VIA THE SWI INSTRUCTION.
00055          *****
00056          0000      A INCHNP EQU      0          INPUT CHAR IN A REG - NO PARITY
00057          0001      A OUTCH  EQU      1          OUTPUT CHAR FROM A REG
00058          0002      A PDATA1 EQU      2          OUTPUT STRING
00059          0003      A PDATA  EQU      3          OUTPUT CR/LF THEN STRING
00060          0004      A OUT2HS EQU      4          OUTPUT TWO HEX AND SPACE
00061          0005      A OUT4HS EQU      5          OUTPUT FOUR HEX AND SPACE
00062          0006      A PCRLF  EQU      6          OUTPUT CR/LF
00063          0007      A SPACE  EQU      7          OUTPUT A SPACE
00064          0008      A MONITR EQU      8          ENTER ASSIST09 MONITOR
00065          0009      A VCTRSW EQU      9          VECTOR EXAMINE/SWITCH
00066          000A      A BRKPT  EQU     10          USER PROGRAM BREAKPOINT
00067          000B      A PAUSE  EQU     11          TASK PAUSE FUNCTION
00068          000B      A NUMFUN EQU     11          NUMBER OF AVAILABLE FUNCTIONS
00069          * NEXT SUB-CODES FOR ACCESSING THE VECTOR TABLE.
00070          * THEY ARE EQUIVALENT TO OFFSETS IN THE TABLE.
00071          * RELATIVE POSITIONING MUST BE MAINTAINED.
00072          0000      A .AVTBL EQU      0          ADDRESS OF VECTOR TABLE
00073          0002      A .CMDL1 EQU      2          FIRST COMMAND LIST
00074          0004      A .RSVD  EQU      4          RESERVED HARDWARE VECTOR
00075          0006      A .SWI3  EQU      6          SWI3 ROUTINE
00076          0008      A .SWI2  EQU      8          SWI2 ROUTINE
00077          000A      A .FIRQ  EQU     10          FIRQ ROUTINE
00078          000C      A .IRQ   EQU     12          IRQ ROUTINE
00079          000E      A .SWI   EQU     14          SWI ROUTINE
00080          0010      A .NMI   EQU     16          NMI ROUTINE
00081          0012      A .RESET EQU     18          RESET ROUTINE
00082          0014      A .CION  EQU     20          CONSOLE ON
00083          0016      A .CIDTA EQU     22          CONSOLE INPUT DATA
00084          0018      A .CIOFF EQU     24          CONSOLE INPUT OFF
00085          001A      A .COON  EQU     26          CONSOLE OUTPUT ON
00086          001C      A .CODTA EQU     28          CONSOLE OUTPUT DATA
00087          001E      A .COOFF EQU     30          CONSOLE OUTPUT OFF
00088          0020      A .HSDTA EQU     32          HIGH SPEED PRINTDATA
00089          0022      A .BSON  EQU     34          PUNCH/LOAD ON
00090          0024      A .BSDTA EQU     36          PUNCH/LOAD DATA
00091          0026      A .BSOFF EQU     38          PUNCH/LOAD OFF
00092          0028      A .PAUSE EQU     40          TASK PAUSE ROUTINE
00093          002A      A .EXPAN EQU     42          EXPRESSION ANALYZER
00094          002C      A .CMDL2 EQU     44          SECOND COMMAND LIST
00095          002E      A .ACIA  EQU     46          ACIA ADDRESS
00096          0030      A .PAD   EQU     48          CHARACTER PAD AND NEW LINE PAD
00097          0032      A .ECHO  EQU     50          ECHO/LOAD AND NULL BKPT FLAG
00098          0034      A .PTM   EQU     52          PTM ADDRESS
00099          001B      A NUMVTR EQU    52/2+1      NUMBER OF VECTORS
00100          0034      A HIVTR  EQU     52          HIGHEST VECTOR OFFSET
```

```

00102 *****
00103 *
00104 * WORK AREA
00105 * THIS WORK AREA IS ASSIGNED TO THE PAGE ADDRESSED BY
00106 * -$1800,PCR FROM THE BASE ADDRESS OF THE ASSIST09
00107 * ROM. THE DIRECT PAGE REGISTER DURING MOST ROUTINE
00108 * OPERATIONS WILL POINT TO THIS WORK AREA. THE STACK
00109 * INITIALLY STARTS UNDER THE RESERVED WORK AREAS AS
00110 * DEFINED HEREIN.
00110 *****
00111 A WORKPG EQU ROMBEG+RAMOFS SETUP DIRECT PAGE ADDRESS
00112 A OODF SETDP WORKPG!>8 NOTIFY ASSEMBLER
00113A E000 ORG WORKPG+256 READY PAGE DEFINITIONS
00114 * THE FOLLOWING THRU BKPTOP MUST RESIDE IN THIS ORDER
00115 * FOR PROPER INITIALIZATION
00116A DFFC ORG *-4
00117 DFFC A PAUSER EQU * PAUSE ROUTINE
00118A DFFB ORG *-1
00119 DFFB A SWIBFL EQU * BYPASS SWI AS BREAKPOINT FLAG
00120A DFFA ORG *-1
00121 DFFA A BKPTCT EQU * BREAKPOINT COUNT
00122A DFF8 ORG *-2
00123 DFF8 A SLEVEL EQU * STACK TRACE LEVEL
00124A DFC2 ORG *-NUMVTR*2
00125 DFC2 A VECTAB EQU * VECTOR TABLE
00126A DFB2 ORG *-2*NUMBKP
00127 DFB2 A BKPTBL EQU * BREAKPOINT TABLE
00128A DFA2 ORG *-2*NUMBKP
00129 DFA2 A BKPTOP EQU * BREAKPOINT OP CODE TABLE
00130A DFA0 ORG *-2
00131 DFA0 A WINDOW EQU * WINDOW
00132A DF9E ORG *-2
00133 DF9E A ADDR EQU * ADDRESS POINTER VALUE
00134A DF9D ORG *-1
00135 DF9D A BASEPG EQU * BASE PAGE VALUE
00136A DF9B ORG *-2
00137 DF9B A NUMBER EQU * BINARY BUILD AREA
00138A DF99 ORG *-2
00139 DF99 A LASTOP EQU * LAST OP CODE TRACED
00140A DF97 ORG *-2
00141 DF97 A RSTACK EQU * RESET STACK POINTER
00142A DF95 ORG *-2
00143 DF95 A PSTACK EQU * COMMAND RECOVERY STACK
00144A DF93 ORG *-2
00145 DF93 A PCNTER EQU * LAST PROGRAM COUNTER
00146A DF91 ORG *-2
00147 DF91 A TRACEC EQU * TRACE COUNT
00148A DF90 ORG *-1
00149 DF90 A SWICNT EQU * TRACE "SWI" NEST LEVEL COUNT
00150A DF8F ORG *-1 (MISFLG MUST FOLLOW SWICNT)
00151 DF8F A MISFLG EQU * LOAD CMD/THRU BREAKPOINT FLAG
00152A DF8E ORG *-1
00153 DF8E A DELIM EQU * EXPRESSION DELIMITER/WORK BYTE
00154A DF66 ORG *-40
00155 DF66 A ROM2WK EQU * EXTENSION ROM RESERVED AREA
00156A DF51 ORG *-21
00157 DF51 A TSTACK EQU * TEMPORARY STACK HOLD
00158 DF51 A STACK EQU * START OF INITIAL STACK

```

---

```

00160 *****
00161 * DEFAULT THE ROM BEGINNING ADDRESS TO 'ROMBEG'
00162 * ASSIST09 IS POSITION ADDRESS INDEPENDENT, HOWEVER
00163 * WE ASSEMBLE ASSUMING CONTROL OF THE HARDWARE VECTORS.
00164 * NOTE THAT THE WORK RAM PAGE MUST BE 'RAMOFS'
00165 * FROM THE ROM BEGINNING ADDRESS.
00166 *****
00167A F800          ORG      ROMBEG      ROM ASSEMBLY/DEFAULT ADDRESS

```

```

00169 *****
00170 *          BLDVTR - BUILD ASSIST09 VECTOR TABLE
00171 * HARDWARE RESET CALLS THIS SUBROUTINE TO BUILD THE
00172 * ASSIST09 VECTOR TABLE. THIS SUBROUTINE RESIDES AT
00173 * THE FIRST BYTE OF THE ASSIST09 ROM, AND CAN BE
00174 * CALLED VIA EXTERNAL CONTROL CODE FOR REMOTE
00175 * ASSIST09 EXECUTION.
00176 * INPUT: S->VALID STACK RAM
00177 * OUTPUT: U->VECTOR TABLE ADDRESS
00178 *          DPR->ASSIST09 WORK AREA PAGE
00179 *          THE VECTOR TABLE AND DEFAULTS ARE INITIALIZED
00180 * ALL REGISTERS VOLATILE
00181 *****

```

```

00183A F800 30      8D E7BE  BLDVTR LEAX   VECTAB,PCR ADDRESS VECTOR TABLE
00184A F804 1F      10          A      TFR    X,D      OBTAIN BASE PAGE ADDRESS
00185A F806 1F      8B          A      TFR    A,DP    SETUP DPR
00186A F808 97      9D          A      STA    BASEPG  STORE FOR QUICK REFERENCE
00187A F80A 33      84          A      LEAU   ,X      RETURN TABLE TO CALLER
00188A F80C 31      8C 35      LEAY  <INITVT,PCR LOAD FROM ADDR
00189A F80F EF      81          A      STU    ,X++   INIT VECTOR TABLE ADDRESS
00190A F811 C6      16          A      LDB   #NUMVTR-5 NUMBER RELOCATABLE VECTORS
00191A F813 34      04          A      PSHS  B        STORE INDEX ON STACK
00192A F815 1F      20          A BLD2  TFR    Y,D    PREPARE ADDRESS RESOLVE
00193A F817 E3      A1          A      ADDD  ,Y++   TO ABSOLUTE ADDRESS
00194A F819 ED      81          A      STD   ,X++   INTO VECTOR TABLE
00195A F81B 6A      E4          A      DEC   ,S      COUNT DOWN
00196A F81D 26      F6 F815    BNE   BLD2   BRANCH IF MORE TO INSERT
00197A F81F C6      0D          A      LDB   #INTVE-INTVS STATIC VALUE INIT LENGTH
00198A F821 A6      A0          A BLD3  LDA    ,Y+    LOAD NEXT BYTE
00199A F823 A7      80          A      STA   ,X+    STORE INTO POSITION
00200A F825 5A          DECB          COUNT DOWN
00201A F826 26      F9 F821    BNE   BLD3   LOOP UNTIL DONE
00202A F828 31      8D F7D4    LEAY  ROM2OF,PCR TEST POSSIBLE EXTENSION ROM
00203A F82C 8E      20FE      A      LDX   #$20FE  LOAD "BRA *" FLAG PATTERN
00204A F82F AC      A1          A      CMPX  ,Y++   ? EXTENDED ROM HERE
00205A F831 26      02 F835    BNE   BLDRTN BRANCH NOT OUR ROM TO RETURN
00206A F833 AD      A4          A      JSR   ,Y      CALL EXTENDED ROM INITIALIZE
00207A F835 35      84          A BLDRTN PULS  PC,B    RETURN TO INITIALIZER

```

```

00209 *****
00210 *          RESET ENTRY POINT
00211 * HARDWARE RESET ENTERS HERE IF ASSIST09 IS ENABLED
00212 * TO RECEIVE THE MC6809 HARDWARE VECTORS. WE CALL
00213 * THE BLDVTR SUBROUTINE TO INITIALIZE THE VECTOR

```



---

```

00214          * TABLE, STACK, AND THEN FIREUP THE MONITOR VIA SWI
00215          * CALL.
00216          *****
00217A F837 32 8D E716 RESET LEAS STACK,PCR SETUP INITIAL STACK
00218A F83B 8D C3 F800      BSR BLDVTR BUILD VECTOR TABLE
00219A F83D 4F          RESET2 CLRA ISSUE STARTUP MESSAGE
00220A F83E 1F 8B      A      TFR A,DP DEFAULT TO PAGE ZERO
00221A F840 3F          SWI PERFORM MONITOR FIREUP
00222A F841          08      A      FCB MONITR TO ENTER COMMAND PROCESSING
00223A F842 20 F9 F83D      BRA RESET2 REENTER MONITOR IF 'CONTINUE'

```

```

00225          *****
00226          *
00227          * INITVT - INITIAL VECTOR TABLE
00228          * THIS TABLE IS RELOCATED TO RAM AND REPRESENTS THE
00229          * INITIAL STATE OF THE VECTOR TABLE. ALL ADDRESSES
00230          * ARE CONVERTED TO ABSOLUTE FORM. THIS TABLE STARTS
00231          * WITH THE SECOND ENTRY, ENDS WITH STATIC CONSTANT
00232          * INITIALIZATION DATA WHICH CARRIES BEYOND THE TABLE.
00233          *****
00233A F844 0158 A INITVT FDB CMDTBL-* DEFAULT FIRST COMMAND TABLE
00234A F846 0292 A      FDB RSRVDR-* DEFAULT UNDEFINED HARDWARE VECTOR
00235A F848 0290 A      FDB SWI3R-* DEFAULT SWI3
00236A F84A 028E A      FDB SWI2R-* DEFAULT SWI2
00237A F84C 0270 A      FDB FIRQR-* DEFAULT FIRQ
00238A F84E 028A A      FDB IRQR-* DEFAULT IRQ ROUTINE
00239A F850 0045 A      FDB SWIR-* DEFAULT SWI ROUTINE
00240A F852 022B A      FDB NMIR-* DEFAULT NMI ROUTINE
00241A F854 FFE3 A      FDB RESET-* RESTART VECTOR
00242A F856 0290 A      FDB CION-* DEFAULT CION
00243A F858 0284 A      FDB CIDTA-* DEFAULT CIDTA
00244A F85A 0296 A      FDB CIOFF-* DEFAULT CIOFF
00245A F85C 028A A      FDB COON-* DEFAULT COON
00246A F85E 0293 A      FDB CODTA-* DEFAULT CODTA
00247A F860 0290 A      FDB COOFF-* DEFAULT COOFF
00248A F862 039A A      FDB HSDTA-* DEFAULT HSDTA
00249A F864 02B7 A      FDB BSON-* DEFAULT BSON
00250A F866 02D2 A      FDB BSDTA-* DEFAULT BSDTA
00251A F868 02BF A      FDB BSOFF-* DEFAULT BSOFF
00252A F86A E792 A      FDB PAUSER-* DEFAULT PAUSE ROUTINE
00253A F86C 047D A      FDB EXPL-* DEFAULT EXPRESSION ANALYZER
00254A F86E 012D A      FDB CMDTB2-* DEFAULT SECOND COMMAND TABLE
00255          * CONSTANTS
00256A F870 E008 A INTVS FDB ACIA DEFAULT ACIA
00257A F872 00 A      FCB DFTCHP,DFTNLP DEFAULT NULL PADDS
00258A F874 0000 A      FDB 0 DEFAULT ECHO
00259A F876 E000 A      FDB PTM DEFAULT PTM
00260A F878 0000 A      FDB 0 INITIAL STACK TRACE LEVEL
00261A F87A 00 A      FCB 0 INITIAL BREAKPOINT COUNT
00262A F87B 00 A      FCB 0 SWI BREAKPOINT LEVEL
00263A F87C 39 A      FCB $39 DEFAULT PAUSE ROUTINE (RTS)
00264          F87D A INTVE EQU *
00265          *B

```

00267

\*\*\*\*\*

---

```

00268 *
00269 * ASSIST09 SWI HANDLER
00270 * THE SWI HANDLER PROVIDES ALL INTERFACING NECESSARY
00271 * FOR A USER PROGRAM. A FUNCTION BYTE IS ASSUMED TO
00272 * FOLLOW THE SWI INSTRUCTION. IT IS BOUND CHECKED
00273 * AND THE PROPER ROUTINE IS GIVEN CONTROL. THIS
00274 * INVOCATION MAY ALSO BE A BREAKPOINT INTERRUPT.
00275 * IF SO, THE BREAKPOINT HANDLER IS ENTERED.
00276 * INPUT: MACHINE STATE DEFINED FOR SWI
00277 * OUTPUT: VARIES ACCORDING TO FUNCTION CALLED. PC ON
00278 * CALLERS STACK INCREMENTED BY ONE IF VALID CALL.
00279 * VOLATILE REGISTERS: SEE FUNCTIONS CALLED
00280 * STATE: RUNS DISABLED UNLESS FUNCTION CLEARS I FLAG.
*****

```

```

00282 * SWI FUNCTION VECTOR TABLE
00283A F87D 0194 A SWIVTB FDB ZINCH-SWIVTB INCHNP
00284A F87F 01B1 A FDB ZOTCH1-SWIVTB OUTCH
00285A F881 01CB A FDB ZPDAT1-SWIVTB PDAT1
00286A F883 01C3 A FDB ZPDATA-SWIVTB PDATA
00287A F885 0175 A FDB ZOT2HS-SWIVTB OUT2HS
00288A F887 0173 A FDB ZOT4HS-SWIVTB OUT4HS
00289A F889 01C0 A FDB ZPCRLF-SWIVTB PCRLF
00290A F88B 0179 A FDB ZSPACE-SWIVTB SPACE
00291A F88D 0055 A FDB ZMONTR-SWIVTB MONTR
00292A F88F 017D A FDB ZVSWTH-SWIVTB VCTRSW
00293A F891 0256 A FDB ZBKPNT-SWIVTB BREAKPOINT
00294A F893 01D1 A FDB ZPAUSE-SWIVTB TASK PAUSE

```

```

00296A F895 6A 8D E6F7 SWIR DEC SWICNT,PCR UP "SWI" LEVEL FOR TRACE
00297A F899 17 0225 FAC1 LBSR LDDP SETUP PAGE AND VERIFY STACK
00298 * CHECK FOR BREAKPOINT TRAP
00299A F89C EE 6A A LDU 10,S LOAD PROGRAM COUNTER
00300A F89E 33 5F A LEAU -1,U BACK TO SWI ADDRESS
00301A F8A0 0D FB A TST SWIBFL ? THIS "SWI" BREAKPOINT
00302A F8A2 26 11 F8B5 BNE SWIDNE BRANCH IF SO TO LET THROUGH
00303A F8A4 17 069B FF42 LBSR CBKLDL OBTAIN BREAKPOINT POINTERS
00304A F8A7 50 NEGB OBTAIN POSITIVE COUNT
00305A F8A8 5A SWILP DECB COUNT DOWN
00306A F8A9 2B 0A F8B5 BMI SWIDNE BRANCH WHEN DONE
00307A F8AB 11A3 A1 A CMPL ,Y++ ? WAS THIS A BREAKPOINT
00308A F8AE 26 F8 F8A8 BNE SWILP BRANCH IF NOT
00309A F8B0 EF 6A A STU 10,S SET PROGRAM COUNTER BACK
00310A F8B2 16 021E FAD3 LBRA ZBKPNT GO DO BREAKPOINT
00311A F8B5 0F FB A SWIDNE CLR SWIBFL CLEAR IN CASE SET
00312A F8B7 37 06 A PULU D OBTAIN FUNCTION BYTE, UP PC
00313A F8B9 C1 0B A CMPB #NUMFUN ? TOO HIGH
00314A F8BB 1022 020F FACE LBHI ERROR YES, DO BREAKPOINT
00315A F8BF EF 6A A STU 10,S BUMP PROGRAM COUNTER PAST SWI
00316A F8C1 58 ASLB FUNCTION CODE TIMES TWO
00317A F8C2 33 8C B8 LEAU SWIVTB,PCR OBTAIN VECTOR BRANCH ADDRESS
00318A F8C5 EC C5 A LDD B,U LOAD OFFSET
00319A F8C7 6E CB A JMP D,U JUMP TO ROUTINE

```

```

00321 *****
00322 * REGISTERS TO FUNCTION ROUTINES:
00323 * DP-> WORK AREA PAGE
00324 * D,Y,U=UNRELIABLE X=AS CALLED FROM USER

```

---

PAGE 007 ASSIST09.SA:0

ASSIST09 - MC6809 MONITOR

00325  
00326

\* S=AS FROM SWI INTERRUPT  
\*\*\*\*\*

00328  
00329  
00330  
00331  
00332  
00333  
00334  
00335  
00336  
00337  
00338  
00339  
00340

\*\*\*\*\*  
\* [SWI FUNCTION 8]  
\* MONITOR ENTRY  
\* FIREUP THE ASSIST09 MONITOR.  
\* THE STACK WITH ITS VALUES FOR THE DIRECT PAGE  
\* REGISTER AND CONDITION CODE FLAGS ARE USED AS IS.  
\* 1) INITIALIZE CONSOLE I/O  
\* 2) OPTIONALLY PRINT SIGNON  
\* 3) INITIALIZE PTM FOR SINGLE STEPPING  
\* 4) ENTER COMMAND PROCESSOR  
\* INPUT: A=0 INIT CONSOLE AND PRINT STARTUP MESSAGE  
\* A#0 OMIT CONSOLE INIT AND STARTUP MESSAGE  
\*\*\*\*\*

00342A F8C9 41  
00343A F8D1 04

A SIGNON FCC /ASSIST09/SIGNON EYE-CATCHER  
A FCB EOF

00345A F8D2 10DF 97  
00346A F8D5 6D 61  
00347A F8D7 26 0D F8E6  
00348A F8D9 AD 9D E6F9  
00349A F8DD AD 9D E6FB  
00350A F8E1 30 8C E5  
00351A F8E4 3F  
00352A F8E5 03  
00353A F8E6 9E F6  
00354A F8E8 27 0D F8F7  
00355A F8EA 6F 02  
00356A F8EC 6F 03  
00357A F8EE CC 01A6  
00358A F8F1 A7 01  
00359A F8F3 E7 84  
00360  
00361A F8F5 6F 01  
00362

A ZMONTR STS RSTACK SAVE FOR BAD STACK RECOVERY  
A TST 1,S ? INIT CONSOLE AND SEND MSG  
BNE ZMONT2 BRANCH IF NOT  
JSR [VECTAB+.CION,PCR] READY CONSOLE INPUT  
JSR [VECTAB+.COON,PCR] READY CONSOLE OUTPUT  
LEAX SIGNON,PCR READY SIGNON EYE-CATCHER  
SWI PERFORM  
FCB PDATA PRINT STRING  
A ZMONT2 LDX VECTAB+.PTM LOAD PTM ADDRESS  
BEQ CMD BRANCH IF NOT TO USE A PTM  
A CLR PTMTM1-PTM,X SET LATCH TO CLEAR RESET  
A CLR PTMTM1+1-PTM,X AND SET GATE HIGH  
A LDD #S01A6 SETUP TIMER 1 MODE  
A STA PTMC2-PTM,X SETUP FOR CONTROL REGISTER1  
A STB PTMC13-PTM,X SET OUTPUT ENABLED/  
\* SINGLE SHOT/ DUAL 8 BIT/INTERNAL MODE/OPERATE  
A CLR PTMC2-PTM,X SET CR2 BACK TO RESET FORM  
\* FALL INTO COMMAND PROCESSOR

00364  
00365  
00366  
00367  
00368  
00369  
00370  
00371  
00372  
00373  
00374  
00375  
00376  
00377  
00378

\*\*\*\*\*  
\* COMMAND HANDLER  
\* BREAKPOINTS ARE REMOVED AT THIS TIME.  
\* PROMPT FOR A COMMAND, AND STORE ALL CHARACTERS  
\* UNTIL A SEPARATOR ON THE STACK.  
\* SEARCH FOR FIRST MATCHING COMMAND SUBSET,  
\* CALL IT OR GIVE '?' RESPONSE.  
\* DURING COMMAND SEARCH:  
\* B=OFFSET TO NEXT ENTRY ON X  
\* U=SAVED S  
\* U-1=ENTRY SIZE+2  
\* U-2=VALID NUMBER FLAG (>=0 VALID)/COMPARE CNT  
\* U-3=CARRIAGE RETURN FLAG (0=CR HAS BEEN DONE)  
\* U-4=START OF COMMAND STORE  
\* S+0=END OF COMMAND STORE

```

00379
00380A F8F7 3F          CMD   SWI           TO NEW LINE
00381A F8F8          06     A     FCB     PCRLF    FUNCTION
00382                * DISARM THE BREAKPOINTS
00383A F8F9 17      0646 FF42 CMDNEP LBSR    CBKLDL  OBTAIN BREAKPOINT POINTERS
00384A F8FC 2A      0C     F90A      BPL    CMDNOL  BRANCH IF NOT ARMED OR NONE
00385A F8FE 50                NEGB                MAKE POSITIVE
00386A F8FF D7      FA     A     STB     BKPTCT  FLAG AS DISARMED
00387A F901 5A                CMDDDL DECB                ? FINISHED
00388A F902 2B      06     F90A      BMI    CMDNOL  BRANCH IF SO
00389A F904 A6      30     A     LDA    -NUMBKP*2,Y LOAD OP CODE STORED
00390A F906 A7      B1     A     STA    [,Y++] STORE BACK OVER "SWI"
00391A F908 20      F7     F901      BRA    CMDDDL  LOOP UNTIL DONE
00392A F90A AE      6A     A     CMDNOL LDX    10,S    LOAD USERS PROGRAM COUNTER
00393A F90C 9F      93     A     STX    PCNTER  SAVE FOR EXPRESSION ANALYZER
00394A F90E 86      3E     A     LDA    #PROMPT LOAD PROMPT CHARACTER
00395A F910 3F                SWI                SEND TO OUTPUT HANDLER
00396A F911          01     A     FCB     OUTCH   FUNCTION
00397A F912 33      E4     A     LEAU   ,S     REMEMBER STACK RESTORE ADDRESS
00398A F914 DF      95     A     STU    PSTACK REMEMBER STACK FOR ERROR USE
00399A F916 4F                CLRA                PREPARE ZERO
00400A F917 5F                CLR B                PREPARE ZERO
00401A F918 DD      9B     A     STD    NUMBER CLEAR NUMBER BUILD AREA
00402A F91A DD      8F     A     STD    MISFLG CLEAR MISCEL. AND SWICNT FLAGS
00403A F91C DD      91     A     STD    TRACEC CLEAR TRACE COUNT
00404A F91E C6      02     A     LDAB  #2     SET D TO TWO
00405A F920 34      07     A     PSHS  D,CC  PLACE DEFAULTS ONTO STACK
00406                * CHECK FOR "QUICK" COMMANDS.
00407A F922 17      0454 FD79      LBSR  READ    OBTAIN FIRST CHARACTER
00408A F925 30      8D 0581      LEAX  CDOT+2,PCR PRESET FOR SINGLE TRACE
00409A F929 81      2E     A     CMPA  #'    ? QUICK TRACE
00410A F92B 27      5A     F987      BEQ   CMDXQT BRANCH EQUAL FOR TRACE ONE
00411A F92D 30      8D 04E9      LEAX  CMPADP+2,PCR READY MEMORY ENTRY POINT
00412A F931 81      2F     A     CMPA  #'/'  ? OPEN LAST USED MEMORY
00413A F933 27      52     F987      BEQ   CMDXQT BRANCH TO DO IT IF SO
00414                * PROCESS NEXT CHARACTER
00415A F935 81      20     A     CMD2  CMPA  #'    ? BLANK OR DELIMITER
00416A F937 23      14     F94D      BLS   CMDGOT BRANCH YES, WE HAVE IT
00417A F939 34      02     A     PSHS  A     BUILD ONTO STACK
00418A F93B 6C      5F     A     INC   -1,U  COUNT THIS CHARACTER
00419A F93D 81      2F     A     CMPA  #'/'  ? MEMORY COMMAND
00420A F93F 27      4F     F990      BEQ   CMDMEM BRANCH IF SO
00421A F941 17      040B FD4F      LBSR  BLDHXC TREAT AS HEX VALUE
00422A F944 27      02     F948      BEQ   CMD3   BRANCH IF STILL VALID NUMBER
00423A F946 6A      5E     A     DEC   -2,U  FLAG AS INVALID NUMBER
00424A F948 17      042E FD79 CMD3  LBSR  READ    OBTAIN NEXT CHARACTER
00425A F94B 20      E8     F935      BRA   CMD2   TEST NEXT CHARACTER
00426                * GOT COMMAND, NOW SEARCH TABLES
00427A F94D 80      0D     A     CMDGOT SUBA  #CR    SET ZERO IF CARRIAGE RETURN
00428A F94F A7      5D     A     STA   -3,U  SETUP FLAG
00429A F951 9E      C4     A     LDX   VECTAB+.CMDL1 START WITH FIRST CMD LIST
00430A F953 E6      80     A     CMDSCH LDB   ,X+    LOAD ENTRY LENGTH
00431A F955 2A      10     F967      BPL   CMDSME BRANCH IF NOT LIST END
00432A F957 9E      EE     A     LDX   VECTAB+.CMDL2 NOW TO SECOND CMD LIST
00433A F959 5C                INCB                ? TO CONTINUE TO DEFAULT LIST
00434A F95A 27      F7     F953      BEQ   CMDSCH BRANCH IF SO
00435A F95C 10DE 95      A     CMDBAD LDS   PSTACK RESTORE STACK
00436A F95F 30      8D 015A      LEAX  ERRMSG,PCR POINT TO ERROR STRING

```

---

|        |      |    |      |      |   |        |                 |                                 |
|--------|------|----|------|------|---|--------|-----------------|---------------------------------|
| 00437A | F963 | 3F |      |      |   | SWI    |                 | SEND OUT                        |
| 00438A | F964 |    | 02   | A    |   | FCB    | PDATA1          | TO CONSOLE                      |
| 00439A | F965 | 20 | 90   | F8F7 |   | BRA    | CMD             | AND TRY AGAIN                   |
| 00440  |      |    |      |      |   |        |                 | * SEARCH NEXT ENTRY             |
| 00441A | F967 | 5A |      |      |   | CMDSME | DECB            | TAKE ACCOUNT OF LENGTH BYTE     |
| 00442A | F968 | E1 | 5F   | A    |   | CMPB   | -1,U            | ? ENTERED LONGER THAN ENTRY     |
| 00443A | F96A | 24 | 03   | F96F |   | BHS    | CMDSIZ          | BRANCH IF NOT TOO LONG          |
| 00444A | F96C | 3A |      |      |   | CMDFLS | ABX             | SKIP TO NEXT ENTRY              |
| 00445A | F96D | 20 | E4   | F953 |   | BRA    | CMDSCH          | AND TRY NEXT                    |
| 00446A | F96F | 31 | 5D   | A    |   | CMDSIZ | LEAY            | PREPARE TO COMPARE              |
| 00447A | F971 | A6 | 5F   | A    |   | LDA    | -1,U            | LOAD SIZE+2                     |
| 00448A | F973 | 80 | 02   | A    |   | SUBA   | #2              | TO ACTUAL SIZE ENTERED          |
| 00449A | F975 | A7 | 5E   | A    |   | STA    | -2,U            | SAVE SIZE FOR COUNTDOWN         |
| 00450A | F977 | 5A |      |      |   | CMDCMP | DECB            | DOWN ONE BYTE                   |
| 00451A | F978 | A6 | 80   | A    |   | LDA    | ,X+             | NEXT COMMAND CHARACTER          |
| 00452A | F97A | A1 | A2   | A    |   | CMPA   | ,-Y             | ? SAME AS THAT ENTERED          |
| 00453A | F97C | 26 | EE   | F96C |   | BNE    | CMDFLS          | BRANCH TO FLUSH IF NOT          |
| 00454A | F97E | 6A | 5E   | A    |   | DEC    | -2,U            | COUNT DOWN LENGTH OF ENTRY      |
| 00455A | F980 | 26 | F5   | F977 |   | BNE    | CMDCMP          | BRANCH IF MORE TO TEST          |
| 00456A | F982 | 3A |      |      |   | ABX    |                 | TO NEXT ENTRY                   |
| 00457A | F983 | EC | 1E   | A    |   | LDD    | -2,X            | LOAD OFFSET                     |
| 00458A | F985 | 30 | 8B   | A    |   | LEAX   | D,X             | COMPUTE ROUTINE ADDRESS+2       |
| 00459A | F987 | 6D | 5D   | A    |   | CMDXQT | TST             | SET CC FOR CARRIAGE RETURN TEST |
| 00460A | F989 | 32 | C4   | A    |   | LEAS   | ,U              | DELETE STACK WORK AREA          |
| 00461A | F98B | AD | 1E   | A    |   | JSR    | -2,X            | CALL COMMAND                    |
| 00462A | F98D | 16 | FF7A | F90A |   | LBRA   | CMDNOL          | GO GET NEXT COMMAND             |
| 00463A | F990 | 6D | 5E   | A    |   | CMDMEM | TST             | ? VALID HEX NUMBER ENTERED      |
| 00464A | F992 | 2B | C8   | F95C |   | BMI    | CMDBAD          | BRANCH ERROR IF NOT             |
| 00465A | F994 | 30 | 88   | AE   | A | LEAX   | <CMEMN-CMPADP,X | TO DIFFERENT ENTRY              |
| 00466A | F997 | DC | 9B   | A    |   | LDD    | NUMBER          | LOAD NUMBER ENTERED             |
| 00467A | F999 | 20 | EC   | F987 |   | BRA    | CMDXQT          | AND ENTER MEMORY COMMAND        |

00469           \*\* COMMANDS ARE ENTERED AS A SUBROUTINE WITH:  
00470           \*\* DPR->ASSIST09 DIRECT PAGE WORK AREA  
00471           \*\* Z=1 CARRIAGE RETURN ENTERED  
00472           \*\* Z=0 NON CARRIAGE RETURN DELIMITER  
00473           \*\* S=NORMAL RETURN ADDRESS  
00474           \*\* THE LABEL "CMDBAD" MAY BE ENTERED TO ISSUE AN  
00475           \*\* AN ERROR FLAG (\*).

00477           \*\*\*\*\*  
00478           \*           ASSIST09 COMMAND TABLES  
00479           \* THESE ARE THE DEFAULT COMMAND TABLES. EXTERNAL  
00480           \* TABLES OF THE SAME FORMAT MAY EXTEND/REPLACE  
00481           \* THESE BY USING THE VECTOR SWAP FUNCTION.  
00482           \*  
00483           \* ENTRY FORMAT:  
00484           \*       +0...TOTAL SIZE OF ENTRY (INCLUDING THIS BYTE)  
00485           \*       +1...COMMAND STRING  
00486           \*       +N...TWO BYTE OFFSET TO COMMAND (ENTRYADDR-\*)  
00487           \*  
00488           \* THE TABLES TERMINATE WITH A ONE BYTE -1 OR -2.  
00489           \* THE -1 CONTINUES THE COMMAND SEARCH WITH THE  
00490           \*           SECOND COMMAND TABLE.  
00491           \* THE -2 TERMINATES COMMAND SEARCHES.  
00492           \*\*\*\*\*@\*\*\*\*\*

```

00494          * THIS IS THE DEFAULT LIST FOR THE SECOND COMMAND
00495          * LIST ENTRY.
00496A F99B      FE      A CMTB2 FCB      -2      STOP COMMAND SEARCHES

00498          * THIS IS THE DEFAULT LIST FOR THE FIRST COMMAND
00499          * LIST ENTRY.
00500          F99C      A CMTBTL EQU      *      MONITOR COMMAND TABLE
00501A F99C      04      A      FCB      4
00502A F99D      42      A      FCC      /B/      'BREAKPOINT' COMMAND
00503A F99E      054D    A      FDB      CBKPT-*
00504A F9A0      04      A      FCB      4
00505A F9A1      43      A      FCC      /C/      'CALL' COMMAND
00506A F9A2      0417    A      FDB      CCALL-*
00507A F9A4      04      A      FCB      4
00508A F9A5      44      A      FCC      /D/      'DISPLAY' COMMAND
00509A F9A6      049D    A      FDB      CDISP-*
00510A F9A8      04      A      FCB      4
00511A F9A9      45      A      FCC      /E/      'ENCODE' COMMAND
00512A F9AA      059F    A      FDB      CENCDE-*
00513A F9AC      04      A      FCB      4
00514A F9AD      47      A      FCC      /G/      'GO' COMMAND
00515A F9AE      03D2    A      FDB      CGO-*
00516A F9B0      04      A      FCB      4
00517A F9B1      4C      A      FCC      /L/      'LOAD' COMMAND
00518A F9B2      04DD    A      FDB      CLOAD-*
00519A F9B4      04      A      FCB      4
00520A F9B5      4D      A      FCC      /M/      'MEMORY' COMMAND
00521A F9B6      040D    A      FDB      CMEM-*
00522A F9B8      04      A      FCB      4
00523A F9B9      4E      A      FCC      /N/      'NULLS' COMMAND
00524A F9BA      04FD    A      FDB      CNULLS-*
00525A F9BC      04      A      FCB      4
00526A F9BD      4F      A      FCC      /O/      'OFFSET' COMMAND
00527A F9BE      050A    A      FDB      COFFS-*
00528A F9C0      04      A      FCB      4
00529A F9C1      50      A      FCC      /P/      'PUNCH' COMMAND
00530A F9C2      04AF    A      FDB      CPUNCH-*
00531A F9C4      04      A      FCB      4
00532A F9C5      52      A      FCC      /R/      'REGISTERS' COMMAND
00533A F9C6      0284    A      FDB      CREG-*
00534A F9C8      04      A      FCB      4
00535A F9C9      53      A      FCC      /S/      'STLEVEL' COMMAND
00536A F9CA      04F2    A      FDB      CSTLEV-*
00537A F9CC      04      A      FCB      4
00538A F9CD      54      A      FCC      /T/      'TRACE' COMMAND
00539A F9CE      04D6    A      FDB      CTRACE-*
00540A F9D0      04      A      FCB      4
00541A F9D1      56      A      FCC      /V/      'VERIFY' COMMAND
00542A F9D2      04CF    A      FDB      CVER-*
00543A F9D4      04      A      FCB      4
00544A F9D5      57      A      FCC      /W/      'WINDOW' COMMAND
00545A F9D6      0468    A      FDB      CWINDO-*
00546A F9D8      FF      A      FCB      -1      END, CONTINUE WITH THE SECOND

```

00548  
00549

\*\*\*\*\*  
\* [SWI FUNCTIONS 4 AND 5]

---

PAGE 011 ASSIST09.SA:0

ASSIST09 - MC6809 MONITOR

```
00550 * 4 - OUT2HS - DECODE BYTE TO HEX AND ADD SPACE
00551 * 5 - OUT4HS - DECODE WORD TO HEX AND ADD SPACE
00552 * INPUT: X->BYTE OR WORD TO DECODE
00553 * OUTPUT: CHARACTERS SENT TO OUTPUT HANDLER
00554 * X->NEXT BYTE OR WORD
00555 *****

00557A F9D9 A6 80 A ZOUT2H LDA ,X+ LOAD NEXT BYTE
00558A F9DB 34 06 A PSHS D SAVE - DO NOT REREAD
00559A F9DD C6 10 A LDB #16 SHIFT BY 4 BITS
00560A F9DF 3D MUL WITH MULTIPLY
00561A F9E0 8D 04 F9E6 BSR ZOUTHX SEND OUT AS HEX
00562A F9E2 35 06 A PULS D RESTORE BYTES
00563A F9E4 84 0F A ANDA #$0F ISOLATE RIGHT HEX
00564A F9E6 8B 90 A ZOUTHX ADDA #$90 PREPARE A-F ADJUST
00565A F9E8 19 DAA ADJUST
00566A F9E9 89 40 A ADCA #$40 PREPARE CHARACTER BITS
00567A F9EB 19 DAA ADJUST
00568A F9EC 6E 9D E5EE SEND JMP [VECTAB+.CODTA,PCR] SEND TO OUT HANDLER

00570A F9F0 8D E7 F9D9 ZOT4HS BSR ZOUT2H CONVERT FIRST BYTE
00571A F9F2 8D E5 F9D9 ZOT2HS BSR ZOUT2H CONVERT BYTE TO HEX
00572A F9F4 AF 64 A STX 4,S UPDATE USERS X REGISTER
00573 * FALL INTO SPACE ROUTINE

00575 *****
00576 * [SWI FUNCTION 7]
00577 * SPACE - SEND BLANK TO OUTPUT HANDLER
00578 * INPUT: NONE
00579 * OUTPUT: BLANK SEND TO CONSOLE HANDLER
00580 *****
00581A F9F6 86 20 A ZSPACE LDA #' LOAD BLANK
00582A F9F8 20 3D FA37 BRA ZOTCH2 SEND AND RETURN

00584 *****
00585 * [SWI FUNCTION 9]
00586 * SWAP VECTOR TABLE ENTRY
00587 * INPUT: A=VECTOR TABLE CODE (OFFSET)
00588 * X=0 OR REPLACEMENT VALUE
00589 * OUTPUT: X=PREVIOUS VALUE
00590 *****
00591A F9FA A6 61 A ZVSWTH LDA 1,S LOAD REQUESTERS A
00592A F9FC 81 34 A CMPA #HIVTR ? SUB-CODE TOO HIGH
00593A F9FE 22 39 FA39 BHI ZOTCH3 IGNORE CALL IF SO
00594A FA00 109E C2 A LDY VECTAB+.AVTBL LOAD VECTOR TABLE ADDRESS
00595A FA03 EE A6 A LDU A,Y U=OLD ENTRY
00596A FA05 EF 64 A STU 4,S RETURN OLD VALUE TO CALLERS X
00597A FA07 AF 7E A STX -2,S ? X=0
00598A FA09 27 2E FA39 BEQ ZOTCH3 YES, DO NOT CHANGE ENTRY
00599A FA0B AF A6 A STX A,Y REPLACE ENTRY
00600A FA0D 20 2A FA39 BRA ZOTCH3 RETURN FROM SWI
00601 *D
```

```

00603 *****
00604 * [SWI FUNCTION 0]
00605 * INCHNP - OBTAIN INPUT CHAR IN A (NO PARITY)
00606 * NULLS AND RUBOUTS ARE IGNORED.
00607 * AUTOMATIC LINE FEED IS SENT UPON RECEIVING A
00608 * CARRIAGE RETURN.
00609 * UNLESS WE ARE LOADING FROM TAPE.
00610 *****
00611A FA0F 8D 5D FA6E ZINCHP BSR XQPAUS RELEASE PROCESSOR
00612A FA11 8D 5F FA72 ZINCH BSR XQCIDT CALL INPUT DATA APPENDAGE
00613A FA13 24 FA FA0F BCC ZINCHP LOOP IF NONE AVAILABLE
00614A FA15 4D TSTA ? TEST FOR NULL
00615A FA16 27 F9 FA11 BEQ ZINCH IGNORE NULL
00616A FA18 81 7F A CMPA #$7F ? RUBOUT
00617A FA1A 27 F5 FA11 BEQ ZINCH BRANCH YES TO IGNORE
00618A FA1C A7 61 A STA 1,S STORE INTO CALLERS A
00619A FA1E 0D 8F A TST MISFLG ? LOAD IN PROGRESS
00620A FA20 26 17 FA39 BNE ZOTCH3 BRANCH IF SO TO NOT ECHO
00621A FA22 81 0D A CMPA #CR ? CARRIAGE RETURN
00622A FA24 26 04 FA2A BNE ZIN2 NO, TEST ECHO BYTE
00623A FA26 86 0A A LDA #LF LOAD LINE FEED
00624A FA28 8D C2 F9EC BSR SEND ALWAYS ECHO LINE FEED
00625A FA2A 0D F4 A ZIN2 TST VECTAB+.ECHO ? ECHO DESIRED
00626A FA2C 26 0B FA39 BNE ZOTCH3 NO, RETURN
00627 * FALL THROUGH TO OUTCH

```

```

00629 *****
00630 * [SWI FUNCTION 1]
00631 * OUTCH - OUTPUT CHARACTER FROM A
00632 * INPUT: NONE
00633 * OUTPUT: IF LINEFEED IS THE OUTPUT CHARACTER THEN
00634 * C=0 NO CTL-X RECEIVED, C=1 CTL-X RECEIVED
00635 *****
00636A FA2E A6 61 A ZOTCH1 LDA 1,S LOAD CHARACTER TO SEND
00637A FA30 30 8C 09 LEAX <ZPCRLS,PCR DEFAULT FOR LINE FEED
00638A FA33 81 0A A CMPA #LF ? LINE FEED
00639A FA35 27 0F FA46 BEQ ZPDTLP BRANCH TO CHECK PAUSE IF SO
00640A FA37 8D B3 F9EC ZOTCH2 BSR SEND SEND TO OUTPUT ROUTINE
00641A FA39 0C 90 A ZOTCH3 INC SWICNT BUMP UP "SWI" TRACE NEST LEVEL
00642A FA3B 3B RTI RETURN FROM "SWI" FUNCTION

```

```

00644 *****
00645 * [SWI FUNCTION 6]
00646 * PCRLF - SEND CR/LF TO CONSOLE HANDLER
00647 * INPUT: NONE
00648 * OUTPUT: CR AND LF SENT TO HANDLER
00649 * C=0 NO CTL-X, C=1 CTL-X RECEIVED
00650 *****
00652A FA3C 04 A ZPCRLS FCB EOT NULL STRING
00654A FA3D 30 8C FC ZPCRLF LEAX ZPCRLS,PCR READY CR,LF STRING
00655 * FALL INTO CR/LF CODE

```



```

00657 *****
00658 *           [SWI FUNCTION 3]
00659 *           PDATA - OUTPUT CR/LF AND STRING
00660 * INPUT: X->STRING
00661 * OUTPUT: CR/LF AND STRING SENT TO OUTPUT CONSOLE
00662 *           HANDLER.
00663 *           C=0 NO CTL-X, C=1 CTL-X RECIEVED
00664 * NOTE: LINE FEED MUST FOLLOW CARRIAGE RETURN FOR
00665 *           PROPER PUNCH DATA.
00666 *****
00667A FA40 86   0D   A ZPDATA LDA   #CR      LOAD CARRIAGE RETURN
00668A FA42 8D   A8   F9EC   BSR      SEND      SEND IT
00669A FA44 86   0A   A       LDA   #LF      LOAD LINE FEED
00670 * FALL INTO PDATA

00672 *****
00673 *           [SWI FUNCTION 2]
00674 *           PDATA - OUTPUT STRING TILL EOT ($04)
00675 * THIS ROUTINE PAUSES IF AN INPUT BYTE BECOMES
00676 * AVAILABLE DURING OUTPUT TRANSMISSION UNTIL A
00677 * SECOND IS RECIEVED.
00678 * INPUT: X->STRING
00679 * OUTPUT: STRING SENT TO OUTPUT CONSOLE DRIVER
00680 *           C=0 NO CTL-X, C=1 CTL-X RECIEVED
00681 *****
00682A FA46 8D   A4   F9EC ZPDTLP BSR      SEND      SEND CHARACTER TO DRIVER
00683A FA48 A6   80   A ZPDAL LDA   ,X+     LOAD NEXT CHARACTER
00684A FA4A 81   04   A       CMPA   #EOT    ? EOT
00685A FA4C 26   F8   FA46   BNE   ZPDTLP  LOOP IF NOT
00686 * FALL INTO PAUSE CHECK FUNCTION

00688 *****
00689 *           [SWI FUNCTION 12]
00690 *           PAUSE - RETURN TO TASK DISPATCHING AND CHECK
00691 *           FOR FREEZE CONDITION OR CTL-X BREAK
00692 * THIS FUNCTION ENTERS THE TASK PAUSE HANDLER SO
00693 * OPTIONALLY OTHER 6809 PROCESSES MAY GAIN CONTROL.
00694 * UPON RETURN, CHECK FOR A 'FREEZE' CONDITION
00695 * WITH A RESULTING WAIT LOOP, OR CONDITION CODE
00696 * RETURN IF A CONTROL-X IS ENTERED FROM THE INPUT
00697 * HANDLER.
00698 * OUTPUT: C=1 IF CTL-X HAS ENTERED, C=0 OTHERWISE
00699 *****
00700A FA4E 8D   1E   FA6E ZPAUSE BSR      XQPAUS  RELEASE CONTROL AT EVERY LINE
00701A FA50 8D   06   FA58   BSR      CHKABT  CHECK FOR FREEZE OR ABORT
00702A FA52 1F   A9   A       TFR      CC,B    PREPARE TO REPLACE CC
00703A FA54 E7   E4   A       STB      ,S     OVERLAY OLD ONE ON STACK
00704A FA56 20   E1   FA39   BRA      ZOTCH3  RETURN FROM "SWI"

00706 * CHKABT - SCAN FOR INPUT PAUSE/ABORT DURING OUTPUT
00707 * OUTPUT: C=0 OK, C=1 ABORT (CTL-X ISSUED)
00708 * VOLATILE: U,X,D
00709A FA58 8D   18   FA72 CHKABT BSR      XQCIDT  ATTEMPT INPUT
00710A FA5A 24   05   FA61   BCC     CHKRTN  BRANCH NO TO RETURN

```

---

PAGE 014 ASSIST09.SA:0

ASSIST09 - MC6809 MONITOR

|        |      |    |    |      |        |        |                              |
|--------|------|----|----|------|--------|--------|------------------------------|
| 00711A | FA5C | 81 | 18 | A    | CMPA   | #CAN   | ? CTL-X FOR ABORT            |
| 00712A | FA5E | 26 | 02 | FA62 | BNE    | CHKWT  | BRANCH NO TO PAUSE           |
| 00713A | FA60 | 53 |    |      | CHKSEC | COMB   | SET CARRY                    |
| 00714A | FA61 | 39 |    |      | CHKRTN | RTS    | RETURN TO CALLER WITH CC SET |
| 00715A | FA62 | 8D | 0A | FA6E | CHKWT  | BSR    | XQPAUS PAUSE FOR A MOMENT    |
| 00716A | FA64 | 8D | 0C | FA72 | BSR    | XQCIDT | ? KEY FOR START              |
| 00717A | FA66 | 24 | FA | FA62 | BCC    | CHKWT  | LOOP UNTIL RECIEVED          |
| 00718A | FA68 | 81 | 18 | A    | CMPA   | #CAN   | ? ABORT SIGNED FROM WAIT     |
| 00719A | FA6A | 27 | F4 | FA60 | BEQ    | CHKSEC | BRANCH YES                   |
| 00720A | FA6C | 4F |    |      | CLRA   |        | SET C=0 FOR NO ABORT         |
| 00721A | FA6D | 39 |    |      | RTS    |        | AND RETURN                   |

|        |      |    |    |      |        |       |                                      |
|--------|------|----|----|------|--------|-------|--------------------------------------|
| 00723  |      |    |    |      |        |       | * SAVE MEMORY WITH JUMPS             |
| 00724A | FA6E | 6E | 9D | E578 | XQPAUS | JMP   | [VECTAB+.PAUSE,PCR] TO PAUSE ROUTINE |
| 00725A | FA72 | AD | 9D | E562 | XQCIDT | JSR   | [VECTAB+.CIDTA,PCR] TO INPUT ROUTINE |
| 00726A | FA76 | 84 | 7F | A    | ANDA   | #\$7F | STRIP PARITY                         |
| 00727A | FA78 | 39 |    |      | RTS    |       | RETURN TO CALLER                     |

|       |  |  |  |  |  |  |   |
|-------|--|--|--|--|--|--|---|
| 00729 |  |  |  |  |  |  | *****   |
| 00730 |  |  |  |  |  |  | * NMI DEFAULT INTERRUPT HANDLER                     |
| 00731 |  |  |  |  |  |  | * THE NMI HANDLER IS USED FOR TRACING INSTRUCTIONS. |
| 00732 |  |  |  |  |  |  | * TRACE PRINTOUTS OCCUR ONLY AS LONG AS THE STACK   |
| 00733 |  |  |  |  |  |  | * TRACE LEVEL IS NOT BREACHED BY FALLING BELOW IT.  |
| 00734 |  |  |  |  |  |  | * TRACING CONTINUES UNTIL THE COUNT TURNS ZERO OR   |
| 00735 |  |  |  |  |  |  | * A CTL-X IS ENTERED FROM THE INPUT CONSOLE DEVICE. |
| 00736 |  |  |  |  |  |  | *****   |

|        |      |    |    |      |        |      |                                      |
|--------|------|----|----|------|--------|------|--------------------------------------|
| 00738A | FA79 |    | 4F | A    | MSHOWP | FCB  | 'O','P','- ,EOT OPCODE PREP          |
| 00740A | FA7D | 8D | 42 | FAC1 | NMIR   | BSR  | LDDP LOAD PAGE AND VERIFY STACK      |
| 00741A | FA7F | 0D | 8F | A    |        | TST  | MISFLG ? THRU A BREAKPOINT           |
| 00742A | FA81 | 26 | 34 | FAB7 |        | BNE  | NMICON BRANCH IF SO TO CONTINUE      |
| 00743A | FA83 | 0D | 90 | A    |        | TST  | SWICNT ? INHIBIT "SWI" DURING TRACE  |
| 00744A | FA85 | 2B | 29 | FAB0 |        | BMI  | NMITRC BRANCH YES                    |
| 00745A | FA87 | 30 | 6C | A    |        | LEAX | 12,S OBTAIN USERS STACK POINTER      |
| 00746A | FA89 | 9C | F8 | A    |        | CMPX | SLEVEL ? TO TRACE HERE               |
| 00747A | FA8B | 25 | 23 | FAB0 |        | BLO  | NMITRC BRANCH IF TOO LOW TO DISPLAY  |
| 00748A | FA8D | 30 | 8C | E9   |        | LEAX | MSHOWP,PCR LOAD OP PREP              |
| 00749A | FA90 | 3F |    |      |        | SWI  | SEND TO CONSOLE                      |
| 00750A | FA91 |    | 02 | A    |        | FCB  | PDATA1 FUNCTION                      |
| 00751A | FA92 | 09 | 8E | A    |        | ROL  | DELIM SAVE CARRY BIT                 |
| 00752A | FA94 | 30 | 8D | E501 |        | LEAX | LASTOP,PCR POINT TO LAST OP          |
| 00753A | FA98 | 3F |    |      |        | SWI  | SEND OUT AS HEX                      |
| 00754A | FA99 |    | 05 | A    |        | FCB  | OUT4HS FUNCTION                      |
| 00755A | FA9A | 8D | 17 | FAB3 |        | BSR  | REGPRS FOLLOW MEMORY WITH REGISTERS  |
| 00756A | FA9C | 25 | 37 | FAD5 |        | BCS  | ZBKCMD BRANCH IF "CANCEL"            |
| 00757A | FA9E | 06 | 8E | A    |        | ROR  | DELIM RESTORE CARRY BIT              |
| 00758A | FAA0 | 25 | 33 | FAD5 |        | BCS  | ZBKCMD BRANCH IF "CANCEL"            |
| 00759A | FAA2 | 9E | 91 | A    |        | LDX  | TRACEC LOAD TRACE COUNT              |
| 00760A | FAA4 | 27 | 2F | FAD5 |        | BEQ  | ZBKCMD IF ZERO TO COMMAND HANDLER    |
| 00761A | FAA6 | 30 | 1F | A    |        | LEAX | -1,X MINUS ONE                       |
| 00762A | FAA8 | 9F | 91 | A    |        | STX  | TRACEC REFRESH                       |
| 00763A | FAAA | 27 | 29 | FAD5 |        | BEQ  | ZBKCMD STOP TRACE WHEN ZERO          |
| 00764A | FAAC | 8D | AA | FA58 |        | BSR  | CHKABT ? ABORT THE TRACE             |
| 00765A | FAAE | 25 | 25 | FAD5 |        | BCS  | ZBKCMD BRANCH YES TO COMMAND HANDLER |

---

PAGE 015 ASSIST09.SA:0

ASSIST09 - MC6809 MONITOR

```
00766A FAB0 16 03F7 FEAA NMITRC LBRA CTRCE3 NO, TRACE ANOTHER INSTRUCTION
00768A FAB3 17 01B9 FC6F REGPRS LBSR REGPRT PRINT REGISTERS AS FROM COMMAND
00769A FAB6 39 RTS RETURN TO CALLER

00771 * JUST EXECUTED THRU A BRKPNT. NOW CONTINUE NORMALLY
00772A FAB7 0F 8F A NMICON CLR MISFLG CLEAR THRU FLAG
00773A FAB9 17 02EB FDA7 LBSR ARMBK2 ARM BREAKPOINTS
00774A FABC 3B RTI RTI AND CONTINUE USERS PROGRAM

00776 * LDDP - SETUP DIRECT PAGE REGISTER, VERIFY STACK.
00777 * AN INVALID STACK CAUSES A RETURN TO THE COMMAND
00778 * HANDLER.
00779 * INPUT: FULLY STACKED REGISTERS FROM AN INTERRUPT
00780 * OUTPUT: DPR LOADED TO WORK PAGE

00782A FABD 3F A ERRMSG FCB '? ,BELL,$20 ,EOT ERROR RESPONSE

00784A FAC1 E6 8D E4D8 LDDP LDB BASEPG,PCR LOAD DIRECT PAGE HIGH BYTE
00785A FAC5 1F 9B A TFR B,DP SETUP DIRECT PAGE REGISTER
00786A FAC7 A1 63 A CMPA 3,S ? IS STACK VALID
00787A FAC9 27 25 FAF0 BEQ RTS YES, RETURN
00788A FACB 10DE 97 A LDS RSTACK RESET TO INITIAL STACK POINTER
00789A FACE 30 8C EC ERROR LEAX ERRMSG,PCR LOAD ERROR REPORT
00790A FAD1 3F SWI SEND OUT BEFORE REGISTERS
00791A FAD2 03 A FCB PDATA ON NEXT LINE
00792 * FALL INTO BREAKPOINT HANDLER

00794 *****
00795 * [SWI FUNCTION 10]
00796 * BREAKPOINT PROGRAM FUNCTION
00797 * PRINT REGISTERS AND GO TO COMMAND HANLER
00798 *****
00799A FAD3 8D DE FAB3 ZBKPT BSR REGPRS PRINT OUT REGISTERS
00800A FAD5 16 FE21 F8F9 ZBKCMD LBRA CMDNEP NOW ENTER COMMAND HANDLER

00802 *****
00803 * IRQ, RESERVED, SWI2 AND SWI3 INTERRUPT HANDLERS
00804 * THE DEFAULT HANDLING IS TO CAUSE A BREAKPOINT.
00805 *****
00806 FAD8 A SWI2R EQU * SWI2 ENTRY
00807 FAD8 A SWI3R EQU * SWI3 ENTRY
00808 FAD8 A IRQR EQU * IRQ ENTRY
00809A FAD8 8D E7 FAC1 RSRVDR BSR LDDP SET BASE PAGE, VALIDATE STACK
00810A FADA 20 F7 FAD3 BRA ZBKPT FORCE A BREAKPOINT

00812 *****
00813 * FIRQ HANDLER
00814 * JUST RETURN FOR THE FIRQ INTERRUPT
00815 *****
00816 FABC A FIRQR EQU RTI IMMEDIATE RETURN
```

```

00818 *****
00819 *          DEFAULT I/O DRIVERS
00820 *****

00822 * CIDTA - RETURN CONSOLE INPUT CHARACTER
00823 * OUTPUT: C=0 IF NO DATA READY, C=1 A=CHARACTER
00824 * U VOLATILE
00825A FADC DE F0 A CIDTA LDU VECTAB+.ACIA LOAD ACIA ADDRESS
00826A FADE A6 C4 A LDA ,U LOAD STATUS REGISTER
00827A FAE0 44 LSRA TEST RECIEVER REGISTER FLAG
00828A FAE1 24 02 FAE5 BCC CIRTN RETURN IF NOTHING
00829A FAE3 A6 41 A LDA 1,U LOAD DATA BYTE
00830A FAE5 39 CIRTN RTS RETURN TO CALLER

00832 * CION - INPUT CONSOLE INITIALIZATION
00833 * COON - OUTPUT CONSOLE INITIALIZATION
00834 * A,X VOLATILE
00835 FAE6 A CION EQU *
00836A FAE6 86 03 A COON LDA #3 RESET ACIA CODE
00837A FAE8 9E F0 A LDA VECTAB+.ACIA LOAD ACIA ADDRESS
00838A FAEA A7 84 A STA ,X STORE INTO STATUS REGISTER
00839A FAEC 86 51 A LDA #$51 SET CONTROL
00840A FAEE A7 84 A STA ,X REGISTER UP
00841A FAF0 39 RTS RTS RETURN TO CALLER

00843 * THE FOLLOWING HAVE NO DUTIES TO PERFORM
00844 FAF0 A CIOFF EQU RTS CONSOLE INPUT OFF
00845 FAF0 A COOFF EQU RTS CONSOLE OUTPUT OFF

00847 * CODTA - OUTPUT CHARACTER TO CONSOLE DEVICE
00848 * INPUT: A=CHARACTER TO SEND
00849 * OUTPUT: CHAR SENT TO TERMINAL WITH PROPER PADDING
00850 * ALL REGISTERS TRANSPARENT

00852A FAF1 34 47 A CODTA PSHS U,D,CC SAVE REGISTERS,WORK BYTE
00853A FAF3 DE F0 A LDU VECTAB+.ACIA ADDRESS ACIA
00854A FAF5 8D 1B FB12 BSR CODTAO CALL OUTPUT CHAR SUBROTINE
00855A FAF7 81 10 A CMPA #DLE ? DATA LINE ESCAPE
00856A FAF9 27 12 FB0D BEQ CODTRT YES, RETURN
00857A FAFB D6 F2 A LDB VECTAB+.PAD DEFAULT TO CHAR PAD COUNT
00858A FAFD 81 0D A CMPA #CR ? CR
00859A FAFF 26 02 FB03 BNE CODTPD BRANCH NO
00860A FB01 D6 F3 A LDB VECTAB+.PAD+1 LOAD NEW LINE PAD COUNT
00861A FB03 4F CODTPD CLRA CREATE NULL
00862A FB04 E7 E4 A STB ,S SAVE COUNT
00863A FB06 8C A FCB SKIP2 ENTER LOOP
00864A FB07 8D 09 FB12 CODTLP BSR CODTAO SEND NULL
00865A FB09 6A E4 A DEC ,S ? FINISHED
00866A FB0B 2A FA FB07 BPL CODTLP NO, CONTINUE WITH MORE
00867A FB0D 35 C7 A CODTRT PULS PC,U,D,CC RESTORE REGISTERS AND RETURN

00869A FB0F 17 FF5C FA6E CODTAD LBSR XQPAUS TEMPORARY GIVE UP CONTROL
00870A FB12 E6 C4 A CODTAO LDB ,U LOAD ACIA CONTROL REGISTER
00871A FB14 C5 02 A BITB #$02 ? TX REGISTER CLEAR

```

---

PAGE 017 ASSIST09.SA:0

ASSIST09 - MC6809 MONITOR

00872A FB16 27 F7 FB0F BEQ CODTAD RELEASE CONTROL IF NOT  
00873A FB18 A7 41 A STA 1,U STORE INTO DATA REGISTER  
00874A FB1A 39 RTS RETURN TO CALLER  
00875 \*E

00877 \* BSON - TURN ON READ/VERIFY/PUNCH MECHANISM  
00878 \* A IS VOLATILE

00880A FB1B 86 11 A BSON LDA #\$11 SET READ CODE  
00881A FB1D 6D 66 A TST 6,S ? READ OR VERIFY  
00882A FB1F 26 01 FB22 BNE BSON2 BRANCH YES  
00883A FB21 4C INCA SET TO WRITE  
00884A FB22 3F BSON2 SWI PERFORM OUTPUT  
00885A FB23 01 A FCB OUTCH FUNCTION  
00886A FB24 0C 8F A INC MISFLG SET LOAD IN PROGRESS FLAG  
00887A FB26 39 RTS RETURN TO CALLER

00889 \* BSOFF - TURN OFF READ/VERIFY/PUNCH MECHANISM  
00890 \* A,X VOLATILE

00891A FB27 86 14 A BSOFF LDA #\$14 TO DC4 - STOP  
00892A FB29 3F SWI SEND OUT  
00893A FB2A 01 A FCB OUTCH FUNCTION  
00894A FB2B 4A DECA CHANGE TO DC3 (X-OFF)  
00895A FB2C 3F SWI SEND OUT  
00896A FB2D 01 A FCB OUTCH FUNCTION  
00897A FB2E 0A 8F A DEC MISFLG CLEAR LOAD IN PROGRESS FLAG  
00898A FB30 8E 61A8 A LDX #25000 DELAY 1 SECOND (2MHZ CLOCK)  
00899A FB33 30 1F A BSOFPL LEAX -1,X COUNT DOWN  
00900A FB35 26 FC FB33 BNE BSOFPL LOOP TILL DONE  
00901A FB37 39 RTS RETURN TO CALLER

00903 \* BSDTA - READ/VERIFY/PUNCH HANDLER  
00904 \* INPUT: S+6=CODE BYTE, VERIFY(-1),PUNCH(0),LOAD(1)  
00905 \* S+4=START ADDRESS  
00906 \* S+2=STOP ADDRESS  
00907 \* S+0=RETURN ADDRESS  
00908 \* OUTPUT: Z=1 NORMAL COMPLETION, Z=0 INVALID LOAD/VER  
00909 \* REGISTERS ARE VOLATILE

00911A FB38 EE 62 A BSDTA LDU 2,S U=TO ADDRESS OR OFFSET  
00912A FB3A 6D 66 A TST 6,S ? PUNCH  
00913A FB3C 27 54 FB92 BEQ BSDPUN BRANCH YES  
00914 \* DURING READ/VERIFY: S+2=MSB ADDRESS SAVE BYTE  
00915 \* S+1=BYTE COUNTER  
00916 \* S+0=CHECKSUM  
00917 \* U HOLDS OFFSET  
00918A FB3E 32 7D A LEAS -3,S ROOM FOR WORK/COUNTER/CHECKSUM  
00919A FB40 3F BSDLD1 SWI GET NEXT CHARACTER  
00920A FB41 00 A FCB INCHNP FUNCTION  
00921A FB42 81 53 A BSDLD2 CMPA #'S ? START OF S1/S9  
00922A FB44 26 FA FB40 BNE BSDLD1 BRANCH NOT  
00923A FB46 3F SWI GET NEXT CHARACTER

---

PAGE 018 ASSIST09.SA:0

ASSIST09 - MC6809 MONITOR

|        |      |    |      |      |        |                                      |
|--------|------|----|------|------|--------|--------------------------------------|
| 00924A | FB47 | 00 | A    | FCB  | INCHNP | FUNCTION                             |
| 00925A | FB48 | 81 | 39   | A    | CMPA   | #'9 ? HAVE S9                        |
| 00926A | FB4A | 27 | 22   | FB6E | BEQ    | BSDSRT YES, RETURN GOOD CODE         |
| 00927A | FB4C | 81 | 31   | A    | CMPA   | #'1 ? HAVE NEW RECORD                |
| 00928A | FB4E | 26 | F2   | FB42 | BNE    | BSDLD2 BRANCH IF NOT                 |
| 00929A | FB50 | 6F | E4   | A    | CLR    | ,S CLEAR CHECKSUM                    |
| 00930A | FB52 | 8D | 21   | FB75 | BSR    | BYTE OBTAIN BYTE COUNT               |
| 00931A | FB54 | E7 | 61   | A    | STB    | 1,S SAVE FOR DECREMENT               |
| 00932  |      |    |      |      |        |                                      |
| 00933A | FB56 | 8D | 1D   | FB75 | BSR    | BYTE OBTAIN HIGH VALUE               |
| 00934A | FB58 | E7 | 62   | A    | STB    | 2,S SAVE IT                          |
| 00935A | FB5A | 8D | 19   | FB75 | BSR    | BYTE OBTAIN LOW VALUE                |
| 00936A | FB5C | A6 | 62   | A    | LDA    | 2,S MAKE D=VALUE                     |
| 00937A | FB5E | 31 | CB   | A    | LEAY   | D,U Y=ADDRESS+OFFSET                 |
| 00938  |      |    |      |      |        |                                      |
| 00939A | FB60 | 8D | 13   | FB75 | BSDNXT | BSR BYTE NEXT BYTE                   |
| 00940A | FB62 | 27 | 0C   | FB70 | BEQ    | BSDEOL BRANCH IF CHECKSUM            |
| 00941A | FB64 | 6D | 69   | A    | TST    | 9,S ? VERIFY ONLY                    |
| 00942A | FB66 | 2B | 02   | FB6A | BMI    | BSDCMP YES, ONLY COMPARE             |
| 00943A | FB68 | E7 | A4   | A    | STB    | ,Y STORE INTO MEMORY                 |
| 00944A | FB6A | E1 | A0   | A    | BSDCMP | CMPB ,Y+ ? VALID RAM                 |
| 00945A | FB6C | 27 | F2   | FB60 | BEQ    | BSDNXT YES, CONTINUE READING         |
| 00946A | FB6E | 35 | 92   | A    | BSDSRT | PULS PC,X,A RETURN WITH Z SET PROPER |
| 00948A | FB70 | 4C |      |      | BSDEOL | INCA ? VALID CHECKSUM                |
| 00949A | FB71 | 27 | CD   | FB40 | BEQ    | BSDLD1 BRANCH YES                    |
| 00950A | FB73 | 20 | F9   | FB6E | BRA    | BSDSRT RETURN Z=0 INVALID            |
| 00952  |      |    |      |      |        |                                      |
| 00953A | FB75 | 8D | 12   | FB89 | BYTE   | BSR BYTHEX OBTAIN FIRST HEX          |
| 00954A | FB77 | C6 | 10   | A    | LDB    | #16 PREPARE SHIFT                    |
| 00955A | FB79 | 3D |      |      | MUL    | OVER TO A                            |
| 00956A | FB7A | 8D | 0D   | FB89 | BSR    | BYTHEX OBTAIN SECOND HEX             |
| 00957A | FB7C | 34 | 04   | A    | PSHS   | B SAVE HIGH HEX                      |
| 00958A | FB7E | AB | E0   | A    | ADDA   | ,S+ COMBINE BOTH SIDES               |
| 00959A | FB80 | 1F | 89   | A    | TFR    | A,B SEND BACK IN B                   |
| 00960A | FB82 | AB | 62   | A    | ADDA   | 2,S COMPUTE NEW CHECKSUM             |
| 00961A | FB84 | A7 | 62   | A    | STA    | 2,S STORE BACK                       |
| 00962A | FB86 | 6A | 63   | A    | DEC    | 3,S DECREMENT BYTE COUNT             |
| 00963A | FB88 | 39 |      |      | BYTRTS | RTS RETURN TO CALLER                 |
| 00965A | FB89 | 3F |      |      | BYTHEX | SWI GET NEXT HEX                     |
| 00966A | FB8A |    | 00   | A    | FCB    | INCHNP CHARACTER                     |
| 00967A | FB8B | 17 | 01D4 | FD62 | LBSR   | CNVHEX CONVERT TO HEX                |
| 00968A | FB8E | 27 | F8   | FB88 | BEQ    | BYTRTS RETURN IF VALID HEX           |
| 00969A | FB90 | 35 | F2   | A    | PULS   | PC,U,Y,X,A RETURN TO CALLER WITH Z=0 |
| 00971  |      |    |      |      |        |                                      |
| 00972  |      |    |      |      |        |                                      |
| 00973  |      |    |      |      |        |                                      |
| 00974  |      |    |      |      |        |                                      |
| 00975  |      |    |      |      |        |                                      |
| 00976  |      |    |      |      |        |                                      |
| 00977A | FB92 | DE | F2   | A    | BSDPUN | LDU VECTAB+.PAD LOAD PADDING VALUES  |
| 00978A | FB94 | AE | 64   | A    | LDX    | 4,S X=FROM ADDRESS                   |
| 00979A | FB96 | 34 | 56   | A    | PSHS   | U,X,D CREATE STACK WORK AREA         |
| 00980A | FB98 | CC | 0018 | A    | LDD    | #24 SET A=0, B=24                    |

---

|        |        |      |           |      |        |             |                                   |
|--------|--------|------|-----------|------|--------|-------------|-----------------------------------|
| 00981A | FB9B   | D7   | F2        | A    | STB    | VECTAB+.PAD | SETUP 24 CHARACTER PADS           |
| 00982A | FB9D   | 3F   |           |      | SWI    |             | SEND NULLS OUT                    |
| 00983A | FB9E   |      | 01        | A    | FCB    | OUTCH       | FUNCTION                          |
| 00984A | FB9F   | C6   | 04        | A    | LDB    | #4          | SETUP NEW LINE PAD TO 4           |
| 00985A | FBA1   | DD   | F2        | A    | STD    | VECTAB+.PAD | SETUP PUNCH PADDING               |
| 00986  |        |      |           |      |        | * CALCULATE | SIZE                              |
| 00987A | FBA3   | EC   | 68        | A    | BSPGO  | LDD         | 8,S LOAD TO                       |
| 00988A | FBA5   | A3   | 62        | A    | SUBD   | 2,S         | MINUS FROM=LENGTH                 |
| 00989A | FBA7   | 1083 | 0018      | A    | CMPD   | #24         | ? MORE THAN 23                    |
| 00990A | FBAB   | 25   | 02        | FBAF | BLO    | BSPOK       | NO, OK                            |
| 00991A | FBAD   | C6   | 17        | A    | LDB    | #23         | FORCE TO 23 MAX                   |
| 00992A | FBAF   | 5C   |           |      | BSPOK  | INCB        | PREPARE COUNTER                   |
| 00993A | FBB0   | E7   | E4        | A    | STB    | ,S          | STORE BYTE COUNT                  |
| 00994A | FBB2   | CB   | 03        | A    | ADDB   | #3          | ADJUST TO FRAME COUNT             |
| 00995A | FBB4   | E7   | 61        | A    | STB    | 1,S         | SAVE                              |
| 00996  |        |      |           |      |        | *PUNCH      | CR,LF,NULS,S,1                    |
| 00997A | FBB6   | 30   | 8C 33     |      | LEAX   | <BSPSTR,    | PCR LOAD START RECORD HEADER      |
| 00998A | FBB9   | 3F   |           |      | SWI    |             | SEND OUT                          |
| 00999A | FBBA   |      | 03        | A    | FCB    | PDATA       | FUNCTION                          |
| 01000  |        |      |           |      |        | * SEND      | FRAME COUNT                       |
| 01001A | FBBB   | 5F   |           |      | CLRB   |             | INITIALIZE CHECKSUM               |
| 01002A | FBBC   | 30   | 61        | A    | LEAX   | 1,S         | POINT TO FRAME COUNT AND ADDR     |
| 01003A | FBBE   | 8D   | 27        | FBE7 | BSR    | BSPUN2      | SEND FRAME COUNT                  |
| 01004  |        |      |           |      |        | *DATA       | ADDRESS                           |
| 01005A | FBC0   | 8D   | 25        | FBE7 | BSR    | BSPUN2      | SEND ADDRESS HI                   |
| 01006A | FBC2   | 8D   | 23        | FBE7 | BSR    | BSPUN2      | SEND ADDRESS LOW                  |
| 01007  |        |      |           |      |        | *PUNCH      | DATA                              |
| 01008A | FBC4   | AE   | 62        | A    | LDX    | 2,S         | LOAD START DATA ADDRESS           |
| 01009A | FBC6   | 8D   | 1F        | FBE7 | BSPMRE | BSR         | BSPUN2 SEND OUT NEXT BYTE         |
| 01010A | FBC8   | 6A   | E4        | A    | DEC    | ,S          | ? FINAL BYTE                      |
| 01011A | FBCA   | 26   | FA        | FBC6 | BNE    | BSPMRE      | LOOP IF NOT DONE                  |
| 01012A | FBCC   | AF   | 62        | A    | STX    | 2,S         | UPDATE FROM ADDRESS VALUE         |
| 01013  |        |      |           |      |        | *PUNCH      | CHECKSUM                          |
| 01014A | FBCE   | 53   |           |      | COMB   |             | COMPLEMENT                        |
| 01015A | FBCF   | E7   | 61        | A    | STB    | 1,S         | STORE FOR SENDOUT                 |
| 01016A | FBD1   | 30   | 61        | A    | LEAX   | 1,S         | POINT TO IT                       |
| 01017A | FBD3   | 8D   | 14        | FBE9 | BSR    | BSPUNC      | SEND OUT AS HEX                   |
| 01018A | FBD5   | AE   | 68        | A    | LDX    | 8,S         | LOAD TOP ADDRESS                  |
| 01019A | FBD7   | AC   | 62        | A    | CMPX   | 2,S         | ? DONE                            |
| 01020A | FBD9   | 24   | C8        | FBA3 | BHS    | BSPGO       | BRANCH NOT                        |
| 01021A | FBD B  | 30   | 8C 11     |      | LEAX   | <BSPEOF,    | PCR PREPARE END OF FILE           |
| 01022A | FBDE   | 3F   |           |      | SWI    |             | SEND OUT STRING                   |
| 01023A | FBD F  |      | 03        | A    | FCB    | PDATA       | FUNCTION                          |
| 01024A | FBE0   | EC   | 64        | A    | LDD    | 4,S         | RECOVER PAD COUNTS                |
| 01025A | FBE2   | DD   | F2        | A    | STD    | VECTAB+.PAD | RESTORE                           |
| 01026A | FBE4   | 4F   |           |      | CLRA   |             | SET Z=1 FOR OK RETURN             |
| 01027A | FBE5   | 35   | D6        | A    | PULS   | PC,U,X,D    | RETURN WITH OK CODE               |
| 01029A | FBE7   | EB   | 84        | A    | BSPUN2 | ADDB        | ,X ADD TO CHECKSUM                |
| 01030A | FBE9   | 16   | FDED F9D9 |      | BSPUNC | LBRA        | ZOUT2H SEND OUT AS HEX AND RETURN |
| 01032A | FBEC   |      | 53        | A    | BSPSTR | FCB         | 'S,'1,EOT CR,LF,NULS,S,1          |
| 01033A | FB E F |      | 53        | A    | BSPEOF | FCC         | /S903000FC/EOF STRING             |
| 01034A | FB F 9 |      | 0D        | A    | FCB    |             | CR,LF,EOT                         |

01036

\* HSDTA - HIGH SPEED PRINT MEMORY

---

```
01037          * INPUT: S+4=START ADDRESS
01038          *           S+2=STOP ADDRESS
01039          *           S+0=RETURN ADDRESS
01040          * X,D VOLATILE

01042          * SEND TITLE
01043A FBFC 3F          HSDTA SWI          SEND NEW LINE
01044A FBFD 06          A          FCB PCRLF          FUNCTION
01045A FBFE C6 06          A          LDB #6          PREPARE 6 SPACES
01046A FC00 3F          HSBLNK SWI          SEND BLANK
01047A FC01 07          A          FCB SPACE          FUNCTION
01048A FC02 5A          DECBB          COUNT DOWN
01049A FC03 26 FB FC00          BNE HSBLNK          LOOP IF MORE
01050A FC05 5F          CLRBB          SETUP BYTE COUNT
01051A FC06 1F 98          A HSHTTL TFR B,A          PREPARE FOR CONVERT
01052A FC08 17 FDDB F9E6          LBSR ZOUTHX          CONVERT TO A HEX DIGIT
01053A FC0B 3F          SWI          SEND BLANK
01054A FC0C 07          A          FCB SPACE          FUNCTION
01055A FC0D 3F          SWI          SEND ANOTHER
01056A FC0E 07          A          FCB SPACE          BLANK
01057A FC0F 5C          INCB          UP ANOTHER
01058A FC10 C1 10          A          CMPB #$10          ? PAST 'F'
01059A FC12 25 F2 FC06          BLO HSHTTL          LOOP UNTIL SO
01060A FC14 3F          HSHLNE SWI          TO NEXT LINE
01061A FC15 06          A          FCB PCRLF          FUNCTION
01062A FC16 25 2F FC47          BCS HSDRTN          RETURN IF USER ENTERED CTL-X
01063A FC18 30 64          A          LEAX 4,S          POINT AT ADDRESS TO CONVERT
01064A FC1A 3F          SWI          PRINT OUT ADDRESS
01065A FC1B 05          A          FCB OUT4HS          FUNCTION
01066A FC1C AE 64          A          LDX 4,S          LOAD ADDRESS PROPER
01067A FC1E C6 10          A          LDB #16          NEXT SIXTEEN
01068A FC20 3F          HSHNXT SWI          CONVERT BYTE TO HEX AND SEND
01069A FC21 04          A          FCB OUT2HS          FUNCTION
01070A FC22 5A          DECBB          COUNT DOWN
01071A FC23 26 FB FC20          BNE HSHNXT          LOOP IF NOT SIXTEENTH
01072A FC25 3F          SWI          SEND BLANK
01073A FC26 07          A          FCB SPACE          FUNCTION
01074A FC27 AE 64          A          LDX 4,S          RELOAD FROM ADDRESS
01075A FC29 C6 10          A          LDB #16          COUNT
01076A FC2B A6 80          A HSHCHR LDA ,X+          NEXT BYTE
01077A FC2D 2B 04 FC33          BMI HSHDOT          TOO LARGE, TO A DOT
01078A FC2F 81 20          A          CMPA #'          ? LOWER THAN A BLANK
01079A FC31 24 02 FC35          BHS HSHCOK          NO, BRANCH OK
01080A FC33 86 2E          A HSHDOT LDA #'          CONVERT INVALID TO A BLANK
01081A FC35 3F          HSHCOK SWI          SEND CHARACTER
01082A FC36 01          A          FCB OUTCH          FUNCTION
01083A FC37 5A          DECBB          ? DONE
01084A FC38 26 F1 FC2B          BNE HSHCHR          BRANCH NO
01085A FC3A AC 62          A          CPX 2,S          ? PAST LAST ADDRESS
01086A FC3C 24 09 FC47          BHS HSDRTN          QUIT IF SO
01087A FC3E AF 64          A          STX 4,S          UPDATE FROM ADDRESS
01088A FC40 A6 65          A          LDA 5,S          LOAD LOW BYTE ADDRESS
01089A FC42 48          ASLAA          ? TO SECTION BOUNDRY
01090A FC43 26 CF FC14          BNE HSHLNE          BRANCH IF NOT
01091A FC45 20 B5 FBFC          BRA HSDTA          BRANCH IF SO
01092A FC47 3F          HSDRTN SWI          SEND NEW LINE
01093A FC48 06          A          FCB PCRLF          FUNCTION
01094A FC49 39          RTS          RETURN TO CALLER
```



---

01095 \*F

```

01097 *****
01098 *   A S S I S T 0 9   C O M M A N D S
01099 *****

```

```

01101 *****REGISTERS - DISPLAY AND CHANGE REGISTERS
01102A FC4A 8D 23 FC6F CREG BSR REGPRT PRINT REGISTERS
01103A FC4C 4C          INCA SET FOR CHANGE FUNCTION
01104A FC4D 8D 21 FC70 BSR REGCHG GO CHANGE, DISPLAY REGISTERS
01105A FC4F 39          RTS RETURN TO COMMAND PROCESSOR

```

```

01107 *****
01108 *   REGPRT - PRINT/CHANGE REGISTERS SUBROUTINE
01109 * WILL ABORT TO 'CMBDAD' IF OVERFLOW DETECTED DURING
01110 * A CHANGE OPERATION. CHANGE DISPLAYS REGISTERS WHEN
01111 * DONE.
01112 * REGISTER MASK LIST CONSISTS OF:
01113 * A) CHARACTERS DENOTING REGISTER
01114 * B) ZERO FOR ONE BYTE, -1 FOR TWO
01115 * C) OFFSET ON STACK TO REGISTER POSITION
01116 * INPUT: SP+4=STACKED REGISTERS
01117 *        A=0 PRINT, A#0 PRINT AND CHANGE
01118 * OUTPUT: (ONLY FOR REGISTER DISPLAY)
01119 *        C=1 CONTROL-X ENTERED, C=0 OTHERWISE
01120 * VOLATILE: D,X (CHANGE)
01121 *           B,X (DISPLAY)
01122 *****

```

```

01123A FC50 50 A REGMSK FCB 'P','C,-1,19 PC REG
01124A FC54 41 A FCB 'A,0,10 A REG
01125A FC57 42 A FCB 'B,0,11 B REG
01126A FC5A 58 A FCB 'X,-1,13 X REG
01127A FC5D 59 A FCB 'Y,-1,15 Y REG
01128A FC60 55 A FCB 'U,-1,17 U REG
01129A FC63 53 A FCB 'S,-1,1 S REG
01130A FC66 43 A FCB 'C','C,0,9 CC REG
01131A FC6A 44 A FCB 'D','P,0,12 DP REG
01132A FC6E 00 A FCB 0 END OF LIST

```

```

01134A FC6F 4F          REGPRT CLRA SETUP PRINT ONLY FLAG
01135A FC70 30 E8 10 A REGCHG LEAX 4+12,S READY STACK VALUE
01136A FC73 34 32 A PSHS Y,X,A SAVE ON STACK WITH OPTION
01137A FC75 31 8C D8 LEAY REGMSK,PCR LOAD REGISTER MASK
01138A FC78 EC A0 A REGP1 LDD ,Y+ LOAD NEXT CHAR OR <=0
01139A FC7A 4D          TSTA ? END OF CHARACTERS
01140A FC7B 2F 04 FC81 BLE REGP2 BRANCH NOT CHARACTER
01141A FC7D 3F          SWI SEND TO CONSOLE
01142A FC7E 01 A FCB OUTCH FUNCTION BYTE
01143A FC7F 20 F7 FC78 BRA REGP1 CHECK NEXT
01144A FC81 86 2D A REGP2 LDA #'- READY '-'
01145A FC83 3F          SWI SEND OUT
01146A FC84 01 A FCB OUTCH WITH OUTCH
01147A FC85 30 E5 A LEAX B,S X->REGISTER TO PRINT
01148A FC87 6D E4 A TST ,S ? CHANGE OPTION

```

---

PAGE 022 ASSIST09.SA:0

ASSIST09 - MC6809 MONITOR

|        |      |      |    |      |        |  |            |                             |
|--------|------|------|----|------|--------|--|------------|-----------------------------|
| 01149A | FC89 | 26   | 12 | FC9D |        | BNE  | REGCNG     | BRANCH YES                  |
| 01150A | FC8B | 6D   | 3F | A    |        | TST  | -1,Y       | ? ONE OR TWO BYTES          |
| 01151A | FC8D | 27   | 03 | FC92 |        | BEQ  | REGP3      | BRANCH ZERO MEANS ONE       |
| 01152A | FC8F | 3F   |    |      |        | SWI  |            | PERFORM WORD HEX            |
| 01153A | FC90 |      | 05 | A    |        | FCB  | OUT4HS     | FUNCTION                    |
| 01154A | FC91 |      | 8C | A    |        | FCB  | SKIP2      | SKIP BYTE PRINT             |
| 01155A | FC92 | 3F   |    |      | REGP3  | SWI  |            | PERFORM BYTE HEX            |
| 01156A | FC93 |      | 04 | A    |        | FCB  | OUT2HS     | FUNCTION                    |
| 01157A | FC94 | EC   | A0 | A    | REG4   | LDD  | ,Y+        | TO FRONT OF NEXT ENTRY      |
| 01158A | FC96 | 5D   |    |      |        | TSTB   |            | ? END OF ENTRIES            |
| 01159A | FC97 | 26   | DF | FC78 |        | BNE  | REGP1      | LOOP IF MORE                |
| 01160A | FC99 | 3F   |    |      |        | SWI  |            | FORCE NEW LINE              |
| 01161A | FC9A |      | 06 | A    |        | FCB  | PCRLF      | FUNCTION                    |
| 01162A | FC9B | 35   | B2 | A    | REGRTN | PULS   | PC,Y,X,A   | RESTORE STACK AND RETURN    |
|        |      |      |    |      |        |  |            |                             |
| 01164A | FC9D | 8D   | 40 | FCDF | REGCNG | BSR  | BLDNNB     | INPUT BINARY NUMBER         |
| 01165A | FC9F | 27   | 10 | FCB1 |        | BEQ  | REGNXC     | IF CHANGE THEN JUMP         |
| 01166A | FCA1 | 81   | 0D | A    |        | CMPA   | #CR        | ? NO MORE DESIRED           |
| 01167A | FCA3 | 27   | 1E | FCC3 |        | BEQ  | REGAGN     | BRANCH NOPE                 |
| 01168A | FCA5 | E6   | 3F | A    |        | LDB  | -1,Y       | LOAD SIZE FLAG              |
| 01169A | FCA7 | 5A   |    |      |        | DECB   |            | MINUS ONE                   |
| 01170A | FCA8 | 50   |    |      |        | NEGB   |            | MAKE POSITIVE               |
| 01171A | FCA9 | 58   |    |      |        | ASLB   |            | TIMES TWO (=2 OR =4)        |
| 01172A | FCAA | 3F   |    |      | REGSKP | SWI  |            | PERFORM SPACES              |
| 01173A | FCAB |      | 07 | A    |        | FCB  | SPACE      | FUNCTION                    |
| 01174A | FCAC | 5A   |    |      |        | DECB   |            |                             |
| 01175A | FCAD | 26   | FB | FCAA |        | BNE  | REGSKP     | LOOP IF MORE                |
| 01176A | FCAF | 20   | E3 | FC94 |        | BRA  | REG4       | CONTINUE WITH NEXT REGISTER |
| 01177A | FCB1 | A7   | E4 | A    | REGNXC | STA  | ,S         | SAVE DELIMITER IN OPTION    |
| 01178  |      |      |    |      | *      |  |            | (ALWAYS > 0)                |
| 01179A | FCB3 | DC   | 9B | A    |        | LDD  | NUMBER     | OBTAIN BINARY RESULT        |
| 01180A | FCB5 | 6D   | 3F | A    |        | TST  | -1,Y       | ? TWO BYTES WORTH           |
| 01181A | FCB7 | 26   | 02 | FCBB |        | BNE  | REGTWO     | BRANCH YES                  |
| 01182A | FCB9 | A6   | 82 | A    |        | LDA  | ,-X        | SETUP FOR TWO               |
| 01183A | FCBB | ED   | 84 | A    | REGTWO | STD  | ,X         | STORE IN NEW VALUE          |
| 01184A | FCBD | A6   | E4 | A    |        | LDA  | ,S         | RECOVER DELIMITER           |
| 01185A | FCBF | 81   | 0D | A    |        | CMPA   | #CR        | ? END OF CHANGES            |
| 01186A | FCC1 | 26   | D1 | FC94 |        | BNE  | REG4       | NO, KEEP ON TRUCK'N         |
| 01187  |      |      |    |      | *      | MOVE STACKED DATA TO NEW STACK IN CASE STACK |            |                             |
| 01188  |      |      |    |      | *      | POINTER HAS CHANGED                          |            |                             |
| 01189A | FCC3 | 30   | 8D | E28A | REGAGN | LEAX   | TSTACK,PCR | LOAD TEMP AREA              |
| 01190A | FCC7 | C6   | 15 | A    |        | LDB  | #21        | LOAD COUNT                  |
| 01191A | FCC9 | 35   | 02 | A    | REGTF1 | PULS   | A          | NEXT BYTE                   |
| 01192A | FCCB | A7   | 80 | A    |        | STA  | ,X+        | STORE INTO TEMP             |
| 01193A | FCCD | 5A   |    |      |        | DECB   |            | COUNT DOWN                  |
| 01194A | FCCE | 26   | F9 | FCC9 |        | BNE  | REGTF1     | LOOP IF MORE                |
| 01195A | FCD0 | 10EE | 88 | EC   | A      | LDS  | -20,X      | LOAD NEW STACK POINTER      |
| 01196A | FCD4 | C6   | 15 | A    |        | LDB  | #21        | LOAD COUNT AGAIN            |
| 01197A | FCD6 | A6   | 82 | A    | REGTF2 | LDA  | ,-X        | NEXT TO STORE               |
| 01198A | FCD8 | 34   | 02 | A    |        | PSHS   | A          | BACK ONTO NEW STACK         |
| 01199A | FCDA | 5A   |    |      |        | DECB   |            | COUNT DOWN                  |
| 01200A | FCDB | 26   | F9 | FCD6 |        | BNE  | REGTF2     | LOOP IF MORE                |
| 01201A | FCDD | 20   | BC | FC9B |        | BRA  | REGRTN     | GO RESTART COMMAND          |

01203  
01204  
01205

\*\*\*\*\*  
\* BLDNUM - BUILDS BINARY VALUE FROM INPUT HEX  
\* THE ACTIVE EXPRESSION HANDLER IS USED.

---

```
01206 * INPUT: S=RETURN ADDRESS
01207 * OUTPUT: A=DELIMITER WHICH TERMINATED VALUE
01208 * (IF DELM NOT ZERO)
01209 * "NUMBER"=WORD BINARY RESULT
01210 * Z=1 IF INPUT RECIEVED, Z=0 IF NO HEX RECIEVED
01211 * REGISTERS ARE TRANSPARENT
01212 *****

01214 * EXECUTE SINGLE OR EXTENDED ROM EXPRESSION HANDLER
01215 *
01216 * THE FLAG "DELIM" IS USED AS FOLLOWS:
01217 * DELIM=0 NO LEADING BLANKS, NO FORCED TERMINATOR
01218 * DELIM=CHR ACCEPT LEADING 'CHR'S, FORCED TERMINATOR
01219A FCDF 4F BLDNNB CLRA NO DYNAMIC DELIMITER
01220A FCE0 8C A FCB SKIP2 SKIP NEXT INSTRUCTION
01221 * BUILD WITH LEADING BLANKS
01222A FCE1 86 20 A BLDNUM LDA #' ALLOW LEADING BLANKS
01223A FCE3 97 8E A STA DELIM STORE AS DELIMITER
01224A FCE5 6E 9D E303 JMP [VECTAB+.EXPAN,PCR] TO EXP ANALYZER

01226 * THIS IS THE DEFAULT SINGLE ROM ANALYZER. WE ACCEPT:
01227 * 1) HEX INPUT
01228 * 2) 'M' FOR LAST MEMORY EXAMINE ADDRESS
01229 * 3) 'P' FOR PROGRAM COUNTER ADDRESS
01230 * 4) 'W' FOR WINDOW VALUE
01231 * 5) '@' FOR INDIRECT VALUE
01232A FCE9 34 14 A EXP1 PSHS X,B SAVE REGISTERS
01233A FCEB 8D 5C FD49 EXPDLM BSR BLDHXI CLEAR NUMBER, CHECK FIRST CHAR
01234A FCED 27 18 FD07 BEQ EXP2 IF HEX DIGIT CONTINUE BUILDING
01235 * SKIP BLANKS IF DESIRED
01236A FCEF 91 8E A CMPA DELIM ? CORRECT DELIMITER
01237A FCF1 27 F8 FCEB BEQ EXPDLM YES, IGNORE IT
01238 * TEST FOR M OR P
01239A FCF3 9E 9E A LDX ADDR DEFAULT FOR 'M'
01240A FCF5 81 4D A CMPA #'M ? MEMORY EXAMINE ADDR WANTED
01241A FCF7 27 16 FD0F BEQ EXP2DL BRANCH IF SO
01242A FCF9 9E 93 A LDX PCNTER DEFAULT FOR 'P'
01243A FCFB 81 50 A CMPA #'P ? LAST PROGRAM COUNTER WANTED
01244A FCFD 27 10 FD0F BEQ EXP2DL BRANCH IF SO
01245A FCFF 9E A0 A LDX WINDOW DEFAULT TO WINDOW
01246A FD01 81 57 A CMPA #'W ? WINDOW WANTED
01247A FD03 27 0A FD0F BEQ EXP2DL
01248A FD05 35 94 A EXPRTN PULS PC,X,B RETURN AND RESTORE REGISTERS
01249 * GOT HEX, NOW CONTINUE BUILDING
01250A FD07 8D 44 FD4D EXP2 BSR BLDHEX COMPUTE NEXT DIGIT
01251A FD09 27 FC FD07 BEQ EXP2 CONTINUE IF MORE
01252A FD0B 20 0A FD17 BRA EXPCDL SEARCH FOR +/-
01253 * STORE VALUE AND CHECK IF NEED DELIMITER
01254A FD0D AE 84 A EXP2DI LDX ,X INDIRECTION DESIRED
01255A FD0F 9F 9B A EXP2DL STX NUMBER STORE RESULT
01256A FD11 0D 8E A TST DELIM ? TO FORCE A DELIMITER
01257A FD13 27 F0 FD05 BEQ EXPRTN RETURN IF NOT WITH VALUE
01258A FD15 8D 62 FD79 BSR READ OBTAIN NEXT CHARACTER
01259 * TEST FOR + OR -
01260A FD17 9E 9B A EXPCDL LDX NUMBER LOAD LAST VALUE
01261A FD19 81 2B A CMPA #' + ? ADD OPERATOR
01262A FD1B 26 0E FD2B BNE EXPCHM BRANCH NOT
01263A FD1D 8D 23 FD42 BSR EXPTRM COMPUTE NEXT TERM
```

```

01264A FD1F 34 02 A PSHS A SAVE DELIMITER
01265A FD21 DC 9B A LDD NUMBER LOAD NEW TERM
01266A FD23 30 8B A EXPADD LEAX D,X ADD TO X
01267A FD25 9F 9B A STX NUMBER STORE AS NEW RESULT
01268A FD27 35 02 A PULS A RESTORE DELIMITER
01269A FD29 20 EC FD17 BRA EXPCDL NOW TEST IT
01270A FD2B 81 2D A EXPCHM CMPA #'- ? SUBTRACT OPERATOR
01271A FD2D 27 07 FD36 BEQ EXPSUB BRANCH IF SO
01272A FD2F 81 40 A CMPA #'@ ? INDIRECTION DESIRED
01273A FD31 27 DA FD0D BEQ EXPTDI BRANCH IF SO
01274A FD33 5F CLRB SET DELIMITER RETURN
01275A FD34 20 CF FD05 BRA EXPRTN AND RETURN TO CALLER
01276A FD36 8D 0A FD42 EXPSUB BSR EXPTRM OBTAIN NEXT TERM
01277A FD38 34 02 A PSHS A SAVE DELIMITER
01278A FD3A DC 9B A LDD NUMBER LOAD UP NEXT TERM
01279A FD3C 40 NEGA NEGATE A
01280A FD3D 50 NEGB NEGATE B
01281A FD3E 82 00 A SBCA #0 CORRECT FOR A
01282A FD40 20 E1 FD23 BRA EXPADD GO ADD TO EXPRESION
01283 * COMPUTE NEXT EXPRESSION TERM
01284 * OUTPUT: X=OLD VALUE
01285 * 'NUMBER'=NEXT TERM
01286A FD42 8D 9D FCE1 EXPTRM BSR BLDNUM OBTAIN NEXT VALUE
01287A FD44 27 32 FD78 BEQ CNVRTS RETURN IF VALID NUMBER
01288A FD46 16 FC13 F95C BLDBAD LBRA CMDBAD ABORT COMMAND IF INVALID

```

```

01290 *****
01291 * BUILD BINARY VALUE USING INPUT CHARACTERS.
01292 * INPUT: A=ASCII HEX VALUE OR DELIMITER
01293 * SP+0=RETURN ADDRESS
01294 * SP+2=16 BIT RESULT AREA
01295 * OUTPUT: Z=1 A=BINARY VALUE
01296 * Z=0 IF INVALID HEX CHARACTER (A UNCHANGED)
01297 * VOLATILE: D

```

```

01298 *****
01299A FD49 0F 9B A BLDHXI CLR NUMBER CLEAR NUMBER
01300A FD4B 0F 9C A CLR CLR NUMBER+1 CLEAR NUMBER
01301A FD4D 8D 2A FD79 BLDHEX BSR READ GET INPUT CHARACTER
01302A FD4F 8D 11 FD62 BLDHXC BSR CNVHEX CONVERT AND TEST CHARACTER
01303A FD51 26 25 FD78 BNE CNVRTS RETURN IF NOT A NUMBER
01304A FD53 C6 10 A LDB #16 PREPARE SHIFT
01305A FD55 3D MUL BY FOUR PLACES
01306A FD56 86 04 A LDA #4 ROTATE BINARY INTO VALUE
01307A FD58 58 BLDSHF ASLB OBTAIN NEXT BIT
01308A FD59 09 9C A ROL NUMBER+1 INTO LOW BYTE
01309A FD5B 09 9B A ROL NUMBER INTO HI BYTE
01310A FD5D 4A DECA COUNT DOWN
01311A FD5E 26 F8 FD58 BNE BLDSHF BRANCH IF MORE TO DO
01312A FD60 20 14 FD76 BRA CNVOK SET GOOD RETURN CODE

```

```

01314 *****
01315 * CONVERT ASCII CHARACTER TO BINARY BYTE
01316 * INPUT: A=ASCII
01317 * OUTPUT: Z=1 A=BINARY VALUE
01318 * Z=0 IF INVALID
01319 * ALL REGISTERS TRANSPARENT

```

```

01320          * (A UNALTERED IF INVALID HEX)
01321          *****
01322A FD62 81 30          A CNVHEX CMPA #'0          ? LOWER THAN A ZERO
01323A FD64 25 12          FD78          BLO CNVRTS          BRANCH NOT VALUE
01324A FD66 81 39          A          CMPA #'9          ? POSSIBLE A-F
01325A FD68 2F 0A          FD74          BLE CNVGOT          BRANCH NO TO ACCEPT
01326A FD6A 81 41          A          CMPA #'A          ? LESS THEN TEN
01327A FD6C 25 0A          FD78          BLO CNVRTS          RETURN IF MINUS (INVALID)
01328A FD6E 81 46          A          CMPA #'F          ? NOT TOO LARGE
01329A FD70 22 06          FD78          BHI CNVRTS          NO, RETURN TOO LARGE
01330A FD72 80 07          A          SUBA #7          DOWN TO BINARY
01331A FD74 84 0F          A CNVGOT ANDA #$0F          CLEAR HIGH HEX
01332A FD76 1A 04          A CNVOK ORCC #4          FORCE ZERO ON FOR VALID HEX
01333A FD78 39          CNVRTS RTS          RETURN TO CALLER

01335          * GET INPUT CHAR, ABORT COMMAND IF CONTROL-X (CANCEL)
01336A FD79 3F          READ SWI          GET NEXT CHARACTER
01337A FD7A          00          A          FCB INCHNP          FUNCTION
01338A FD7B 81 18          A          CMPA #CAN          ? ABORT COMMAND
01339A FD7D 27 C7          FD46          BEQ BLDBAD          BRANCH TO ABORT IF SO
01340A FD7F 39          RTS          RETURN TO CALLER
01341          *G

01343          *****GO - START PROGRAM EXECUTION
01344A FD80 8D 01          FD83 CGO          BSR GOADDR          BUILD ADDRESS IF NEEDED
01345A FD82 3B          RTI          START EXECUTING

01347          * FIND OPTIONAL NEW PROGRAM COUNTER. ALSO ARM THE
01348          * BREAKPOINTS.
01349A FD83 35 30          A GOADDR PULS Y,X          RECOVER RETURN ADDRESS
01350A FD85 34 10          A          PSHS X          STORE RETURN BACK
01351A FD87 26 19          FDA2          BNE GONDFT          IF NO CARRIAGE RETURN THEN NEW PC
01352          * DEFAULT PROGRAM COUNTER, SO FALL THROUGH IF
01353          * IMMEDIATE BREAKPOINT.
01354A FD89 17 01B6 FF42          LBSR CBKLDLDR          SEARCH BREAKPOINTS
01355A FD8C AE 6C          A          LDH          12,S          LOAD PROGRAM COUNTER
01356A FD8E 5A          ARMBLP DECB          COUNT DOWN
01357A FD8F 2B 16          FDA7          BMI ARMBK2          DONE, NONE TO SINGLE TRACE
01358A FD91 A6 30          A          LDA -NUMBKP*2,Y          PRE-FETCH OPCODE
01359A FD93 AC A1          A          CMPX ,Y++          ? IS THIS A BREAKPOINT
01360A FD95 26 F7          FD8E          BNE ARMBLP          LOOP IF NOT
01361A FD97 81 3F          A          CMPA #$3F          ? SWI BREAKPOINTED
01362A FD99 26 02          FD9D          BNE ARMNSW          NO, SKIP SETTING OF PASS FLAG
01363A FD9B 97 FB          A          STA SWIBFL          SHOW UPCOMMING SWI NOT BRKPN
01364A FD9D 0C 8F          A ARMNSW INC MISFLG          FLAG THRU A BREAKPOINT
01365A FD9F 16 0106 FEA8          LBRA CDOT          DO SINGLE TRACE W/O BREAKPOINTS
01366          * OBTAIN NEW PROGRAM COUNTER
01367A FDA2 17 00BB FE60 GONDFT LBSR CDNUM          OBTAIN NEW PROGRAM COUNTER
01368A FDA5 ED 6C          A          STD          12,S          STORE INTO STACK
01369A FDA7 17 0198 FF42 ARMBK2 LBSR CBKLDLDR          OBTAIN TABLE
01370A FDAA 00 FA          A          NEG          BKPTCT          COMPLEMENT TO SHOW ARMED
01371A FDAC 5A          ARMLP DECB          ? DONE
01372A FDAD 2B C9          FD78          BMI CNVRTS          RETURN WHEN DONE
01373A FDAF A6 B4          A          LDA          [,Y]          LOAD OPCODE
01374A FDB1 A7 30          A          STA          -NUMBKP*2,Y          STORE INTO OPCODE TABLE

```

```
01375A FDB3 86 3F A LDA #$3F READY "SWI" OPCODE
01376A FDB5 A7 B1 A STA [,Y++] STORE AND MOVE UP TABLE
01377A FDB7 20 F3 FDAC BRA ARMLOP AND CONTINUE
```

```
01379 *****CALL - CALL ADDRESS AS SUBROUTINE
01380A FDB9 8D C8 FD83 CCALL BSR GOADDR FETCH ADDRESS IF NEEDED
01381A FDBB 35 7F A PULS U,Y,X,DP,D,CC RESTORE USERS REGISTERS
01382A FDBD AD F1 A JSR [,S++] CALL USER SUBROUTINE
01383A FDBF 3F CGOBRK SWI PERFORM BREAKPOINT
01384A FDC0 0A A FCB BRKPT FUNCTION
01385A FDC1 20 FC FDBF BRA CGOBRK LOOP UNTIL USER CHANGES PC
```

```
01387 *****MEMORY - DISPLAY/CHANGE MEMORY
01388 * CMEMN AND CMPADP ARE DIRECT ENTRY POINTS FROM
01389 * THE COMMAND HANDLER FOR QUICK COMMANDS
01390A FDC3 17 009A FE60 CMEM LBSR CDNUM OBTAIN ADDRESS
01391A FDC6 DD 9E A CMEMN STD ADDR STORE DEFAULT
01392A FDC8 9E 9E A CMEM2 LDX ADDR LOAD POINTER
01393A FDCA 17 FC0C F9D9 LBSR ZOUT2H SEND OUT HEX VALUE OF BYTE
01394A FDCD 86 2D A LDA #'- LOAD DELIMITER
01395A FDCF 3F SWI SEND OUT
01396A FDD0 01 A FCB OUTCH FUNCTION
01397A FDD1 17 FF0B FCDF CMEM4 LBSR BLDNNB OBTAIN NEW BYTE VALUE
01398A FDD4 27 0A FDE0 BEQ CMENUM BRANCH IF NUMBER
01399 * COMA - SKIP BYTE
01400A FDD6 81 2C A CMPA #' ? COMMA
01401A FDD8 26 0E FDE8 BNE CMNOTC BRANCH NOT
01402A FDDA 9F 9E A STX ADDR UPDATE POINTER
01403A FDDC 30 01 A LEAX 1,X TO NEXT BYTE
01404A FDDE 20 F1 FDD1 BRA CMEM4 AND INPUT IT
01405A FDE0 D6 9C A CMENUM LDB NUMBER+1 LOAD LOW BYTE VALUE
01406A FDE2 8D 47 FE2B BSR MUPDAT GO OVERLAY MEMORY BYTE
01407A FDE4 81 2C A CMPA #' ? CONTINUE WITH NO DISPLAY
01408A FDE6 27 E9 FDD1 BEQ CMEM4 BRANCH YES
01409 * QUOTED STRING
01410A FDE8 81 27 A CMNOTC CMPA #' ? QUOTED STRING
01411A FDEA 26 0C FDF8 BNE CMNOTQ BRANCH NO
01412A FDEC 8D 8B FD79 CMESTR BSR READ OBTAIN NEXT CHARACTER
01413A FDEE 81 27 A CMPA #' ? END OF QUOTED STRING
01414A FDF0 27 0C FDFE BEQ CMSPCE YES, QUIT STRING MODE
01415A FDF2 1F 89 A TFR A,B TO B FOR SUBROUTINE
01416A FDF4 8D 35 FE2B BSR MUPDAT GO UPDATE BYTE
01417A FDF6 20 F4 FDEC BRA CMESTR GET NEXT CHARACTER
01418 * BLANK - NEXT BYTE
01419A FDF8 81 20 A CMNOTQ CMPA #$20 ? BLANK FOR NEXT BYTE
01420A FDFA 26 06 FE02 BNE CMNOTB BRANCH NOT
01421A FDFC 9F 9E A STX ADDR UPDATE POINTER
01422A FDFF 3F CMSPCE SWI GIVE SPACE
01423A FDFE 07 A FCB SPACE FUNCTION
01424A FE00 20 C6 FDC8 BRA CMEM2 NOW PROMPT FOR NEXT
01425 * LINE FEED - NEXT BYTE WITH ADDRESS
01426A FE02 81 0A A CMNOTB CMPA #LF ? LINE FEED FOR NEXT BYTE
01427A FE04 26 08 FE0E BNE CMNOTL BRANCH NO
01428A FE06 86 0D A LDA #CR GIVE CARRIAGE RETURN
```

---

```
01429A FE08 3F          SWI          TO CONSOLE
01430A FE09          01          A          FCB          OUTCH        HANDLER
01431A FE0A 9F          9E          A          STX          ADDR          STORE NEXT ADDRESS
01432A FE0C 20          0A          FE18         BRA          CMPADP       BRANCH TO SHOW
01433          * UP ARROW - PREVIOUS BYTE AND ADDRESS
01434A FE0E 81          5E          A          CMNOTL  CMPA          #'@          ? UP ARROW FOR PREVIOUS BYTE
01435A FE10 26          0A          FE1C         BNE          CMNOTU       BRANCH NOT
01436A FE12 30          1E          A          LEAX         -2,X         DOWN TO PREVIOUS BYTE
01437A FE14 9F          9E          A          STX          ADDR          STORE NEW POINTER
01438A FE16 3F          CMPADS  SWI          FORCE NEW LINE
01439A FE17          06          A          FCB          PCRLF        FUNCTION
01440A FE18 8D          07          FE21  CMPADP  BSR          PRTADR       GO PRINT ITS VALUE
01441A FE1A 20          AC          FDC8         BRA          CMEM2       THEN PROMPT FOR INPUT
01442          * SLASH - NEXT BYTE WITH ADDRESS
01443A FE1C 81          2F          A          CMNOTU  CMPA          #'/'        ? SLASH FOR CURRENT DISPLAY
01444A FE1E 27          F6          FE16         BEQ          CMPADS       YES, SEND ADDRESS
01445A FE20 39          RTS          RETURN FROM COMMAND

01447          * PRINT CURRENT ADDRESS
01448A FE21 9E          9E          A          PRTADR  LDX          ADDR          LOAD POINTER VALUE
01449A FE23 34          10          A          PSHS          X          SAVE X ON STACK
01450A FE25 30          E4          A          LEAX          ,S         POINT TO IT FOR DISPLAY
01451A FE27 3F          SWI          DISPLAY POINTER IN HEX
01452A FE28          05          A          FCB          OUT4HS       FUNCTION
01453A FE29 35          90          A          PULS          PC,X        RECOVER POINTER AND RETURN

01455          * UPDATE BYTE
01456A FE2B 9E          9E          A          MUPDAT  LDX          ADDR          LOAD NEXT BYTE POINTER
01457A FE2D E7          80          A          STB          ,X+        STORE AND INCREMENT X
01458A FE2F E1          1F          A          CMPB          -1,X        ? SUCCESFULL STORE
01459A FE31 26          03          FE36         BNE          MUPBAD       BRANCH FOR '?' IF NOT
01460A FE33 9F          9E          A          STX          ADDR          STORE NEW POINTER VALUE
01461A FE35 39          RTS          BACK TO CALLER
01462A FE36 34          02          A          MUPBAD  PSHS          A          SAVE A REGISTER
01463A FE38 86          3F          A          LDA          #'?        SHOW INVALID
01464A FE3A 3F          SWI          SEND OUT
01465A FE3B          01          A          FCB          OUTCH        FUNCTION
01466A FE3C 35          82          A          PULS          PC,A        RETURN TO CALLER

01468          *****WINDOW - SET WINDOW VALUE
01469A FE3E 8D          20          FE60  CWINDO  BSR          CDNUM        OBTAIN WINDOW VALUE
01470A FE40 DD          A0          A          STD          WINDOW       STORE IT IN
01471A FE42 39          RTS          END COMMAND

01473          *****DISPLAY - HIGH SPEED DISPLAY MEMORY
01474A FE43 8D          1B          FE60  CDISP   BSR          CDNUM        FETCH ADDRESS
01475A FE45 C4          F0          A          ANDB         #$F0        FORCE TO 16 BOUNDRY
01476A FE47 1F          02          A          TFR          D,Y        SAVE IN Y
01477A FE49 30          2F          A          LEAX         15,Y        DEFAULT LENGTH
01478A FE4B 25          04          FE51         BCS          CDISPS       BRANCH IF END OF INPUT
01479A FE4D 8D          11          FE60         BSR          CDNUM        OBTAIN COUNT
01480A FE4F 30          AB          A          LEAX         D,Y        ASSUME COUNT, COMPUTE END ADDR
01481A FE51 34          30          A          CDISPS  PSHS          Y,X        SETUP PARAMETERS FOR HSDATA
01482A FE53 10A3 62          A          CMPD         2,S        ? WAS IT COUNT
```

```

01483A FE56 23 02 FE5A BLS CDCNT BRANCH YES
01484A FE58 ED E4 A STD ,S STORE HIGH ADDRESS
01485A FE5A AD 9D E184 CDCNT JSR [VECTAB+.HSDTA,PCR] CALL PRINT ROUTINE
01486A FE5E 35 E0 A PULS PC,U,Y CLEAN STACK AND END COMMAND

```

```

01488 * OBTAIN NUMBER - ABORT IF NONE
01489 * ONLY DELIMITERS OF CR, BLANK, OR '/' ARE ACCEPTED
01490 * OUTPUT: D=VALUE, C=1 IF CARRIAGE RETURN DELMITER,
01491 * ELSE C=0
01492A FE60 17 FE7E FCE1 CDNUM LBSR BLDNUM OBTAIN NUMBER
01493A FE63 26 09 FE6E BNE CDBADN BRANCH IF INVALID
01494A FE65 81 2F A CMPA #'/? VALID DELIMITER
01495A FE67 22 05 FE6E BHI CDBADN BRANCH IF NOT FOR ERROR
01496A FE69 81 0E A CMPA #CR+1 LEAVE COMPARE FOR CARRIAGE RET
01497A FE6B DC 9B A LDD NUMBER LOAD NUMBER
01498A FE6D 39 RTS RETURN WITH COMPARE
01499A FE6E 16 FAEB F95C CDBADN LBRA CMDBAD RETURN TO ERROR MECHANISM

```

```

01501 *****PUNCH - PUNCH MEMORY IN S1-S9 FORMAT
01502A FE71 8D ED FE60 CPUNCH BSR CDNUM OBTAIN START ADDRESS
01503A FE73 1F 02 A TFR D,Y SAVE IN Y
01504A FE75 8D E9 FE60 BSR CDNUM OBTAIN END ADDRESS
01505A FE77 6F E2 A CLR ,-S SETUP PUNCH FUNCTION CODE
01506A FE79 34 26 A PSHS Y,D STORE VALUES ON STACK
01507A FE7B AD 9D E165 CCALBS JSR [VECTAB+.BSON,PCR] INITIALIZE HANDLER
01508A FE7F AD 9D E163 JSR [VECTAB+.BSDTA,PCR] PERFORM FUNCTION
01509A FE83 34 01 A PSHS CC SAVE RETURN CODE
01510A FE85 AD 9D E15F JSR [VECTAB+.BSOFF,PCR] TURN OFF HANDLER
01511A FE89 35 01 A PULS CC OBTAIN CONDITION CODE SAVED
01512A FE8B 26 E1 FE6E BNE CDBADN BRANCH IF ERROR
01513A FE8D 35 B2 A PULS PC,Y,X,A RETURN FROM COMMAND

```

```

01515 *****LOAD - LOAD MEMORY FROM S1-S9 FORMAT
01516A FE8F 8D 01 FE92 CLOAD BSR CLVOFS CALL SETUP AND PASS CODE
01517A FE91 01 A FCB 1 LOAD FUNCTION CODE FOR PACKET
01519A FE92 33 F1 A CLVOFS LEAU [,S++] LOAD CODE IN HIGH BYTE OF U
01520A FE94 33 D4 A LEAU [,U] NOT CHANGING CC AND RESTORE S
01521A FE96 27 03 FE9B BEQ CLVDFT BRANCH IF CARRIAGE RETURN NEXT
01522A FE98 8D C6 FE60 BSR CDNUM OBTAIN OFFSET
01523A FE9A 8C A FCB SKIP2 SKIP DEFAULT OFFSET
01524A FE9B 4F CLVDFT CLRA CREATE ZERO OFFSET
01525A FE9C 5F CLRB AS DEFAULT
01526A FE9D 34 4E A PSHS U,DP,D SETUP CODE, NULL WORD, OFFSET
01527A FE9F 20 DA FE7B BRA CCALBS ENTER CALL TO BS ROUTINES

```

```

01529 *****VERIFY - COMPARE MEMORY WITH FILES
01530A FEAl 8D EF FE92 CVER BSR CLVOFS COMPUTE OFFSET IF ANY
01531A FEa3 FF A FCB -1 VERIFY FNCTN CODE FOR PACKET

```



---

01533 \*\*\*\*\*TRACE - TRACE INSTRUCTIONS  
01534 \*\*\*\*\* - SINGLE STEP TRACE  
01535A FEA4 8D BA FE60 CTRACE BSR CDNUM OBTAIN TRACE COUNT  
01536A FEA6 DD 91 A STD TRACEC STORE COUNT  
01537A FEA8 32 62 A CDOT LEAS 2,S RID COMMAND RETURN FROM STACK  
01538A FEAA EE F8 0A A CTRCE3 LDU [10,S] LOAD OPCODE TO EXECUTE  
01539A FEAD DF 99 A STU LASTOP STORE FOR TRACE INTERRUPT  
01540A FEAF DE F6 A LDU VECTAB+.PTM LOAD PTM ADDRESS  
01541A FEB1 CC 0701 A LDD #7!<8+1 CYCLES DOWN+CYCLES UP  
01542A FEB4 ED 42 A STD PTMTM1-PTM,U START NMI TIMEOUT  
01543A FEB6 3B RTI RETURN FOR ONE INSTRUCTION

01545 \*\*\*\*\*NULLS - SET NEW LINE AND CHAR PADDING  
01546A FEB7 8D A7 FE60 CNULLS BSR CDNUM OBTAIN NEW LINE PAD  
01547A FEB9 DD F2 A STD VECTAB+.PAD RESET VALUES  
01548A FEBB 39 RTS END COMMAND

01550 \*\*\*\*\*STLEVEL - SET STACK TRACE LEVEL  
01551A FEBC 27 05 FEC3 CSTLEV BEQ STLDFT TAKE DEFAULT  
01552A FEBE 8D A0 FE60 BSR CDNUM OBTAIN NEW STACK LEVEL  
01553A FEC0 DD F8 A STD SLEVEL STORE NEW ENTRY  
01554A FEC2 39 RTS TO COMMAND HANDLER  
01555A FEC3 30 6E A STLDFT LEAX 14,S COMPUTE NMI COMPARE  
01556A FEC5 9F F8 A STX SLEVEL AND STORE IT  
01557A FEC7 39 RTS END COMMAND

01559 \*\*\*\*\*OFFSET - COMPUTE SHORT AND LONG  
01560 \*\*\*\*\* BRANCH OFFSETS  
01561A FEC8 8D 96 FE60 COFFS BSR CDNUM OBTAIN INSTRUCTION ADDRESS  
01562A FECA 1F 01 A TFR D,X USE AS FROM ADDRESS  
01563A FECC 8D 92 FE60 BSR CDNUM OBTAIN TO ADDRESS  
01564 \* D=TO INSTRUCTION, X=FROM INSTRUCTION OFFSET BYTE(S)  
01565A FECE 30 01 A LEAX 1,X ADJUST FOR \*+2 SHORT BRANCH  
01566A FED0 34 30 A PSHS Y,X STORE WORK WORD AND VALUE ON S  
01567A FED2 A3 E4 A SUBD ,S FIND OFFSET  
01568A FED4 ED E4 A STD ,S SAVE OVER STACK  
01569A FED6 30 61 A LEAX 1,S POINT FOR ONE BYTE DISPLAY  
01570A FED8 1D SEX SIGN EXTEND LOW BYTE  
01571A FED9 A1 E4 A CMPA ,S ? VALID ONE BYTE OFFSET  
01572A FEDB 26 02 FEDF BNE COFNO1 BRANCH IF NOT  
01573A FEDD 3F SWI SHOW ONE BYTE OFFSET  
01574A FEDE 04 A FCB OUT2HS FUNCTION  
01575A FEDF EE E4 A COFNO1 LDU ,S RELOAD OFFSET  
01576A FEE1 33 5F A LEAU -1,U CONVERT TO LONG BRANCH OFFSET  
01577A FEE3 EF 84 A STU ,X STORE BACK WHERE X POINTS NOW  
01578A FEE5 3F SWI SHOW TWO BYTE OFFSET  
01579A FEE6 05 A FCB OUT4HS FUNCTION  
01580A FEE7 3F SWI FORCE NEW LINE  
01581A FEE8 06 A FCB PCRLF FUNCTION  
01582A FEE9 35 96 A PULS PC,X,D RESTORE STACK AND END COMMAND  
01583 \*H

```

01585 *****BREAKPOINT - DISPLAY/ENTER/DELETE/CLEAR
01586 *****
01587A FEEB 27 23 FF10 CBKPT BEQ CBKDSP BRANCH DISPLAY OF JUST 'B'
01588A FEED 17 FDF1 FCE1 LBSR BLDNUM ATTEMPT VALUE ENTRY
01589A FEF0 27 2C FF1E BEQ CBKADD BRANCH TO ADD IF SO
01590A FEF2 81 2D A CMPA #'- ? CORRECT DELIMITER
01591A FEF4 26 3F FF35 BNE CBKERR NO, BRANCH FOR ERROR
01592A FEF6 17 FDE8 FCE1 LBSR BLDNUM ATTEMPT DELETE VALUE
01593A FEF9 27 03 FEFE BEQ CBKDLE GOT ONE, GO DELETE IT
01594A FEFB 0F FA A CLR BKPTCT WAS 'B -', SO ZERO COUNT
01595A FEFD 39 CBKRTS RTS END COMMAND
01596 * DELETE THE ENTRY
01597A FEFE 8D 40 FF40 CBKDLE BSR CBKSET SETUP REGISTERS AND VALUE
01598A FF00 5A CBKDLF DECB ? ANY ENTRIES IN TABLE
01599A FF01 2B 32 FF35 BMI CBKERR BRANCH NO, ERROR
01600A FF03 AC A1 A CMPX ,Y++ ? IS THIS THE ENTRY
01601A FF05 26 F9 FF00 BNE CBKDLF NO, TRY NEXT
01602 * FOUND, NOW MOVE OTHERS UP IN ITS PLACE
01603A FF07 AE A1 A CBKDLF LDX ,Y++ LOAD NEXT ONE UP
01604A FF09 AF 3C A STX -4,Y MOVE DOWN BY ONE
01605A FF0B 5A DECB ? DONE
01606A FF0C 2A F9 FF07 BPL CBKDLF NO, CONTINUE MOVE
01607A FF0E 0A FA A DEC BKPTCT DECREMENT BREAKPOINT COUNT
01608A FF10 8D 2E FF40 CBKDSP BSR CBKSET SETUP REGISTERS AND LOAD VALUE
01609A FF12 27 E9 FEFD BEQ CBKRTS RETURN IF NONE TO DISPLY
01610A FF14 30 A1 A CBKDSL LEAX ,Y++ POINT TO NEXT ENTRY
01611A FF16 3F SWI DISPLAY IN HEX
01612A FF17 05 A FCB OUT4HS FUNCTION
01613A FF18 5A DECB COUNT DOWN
01614A FF19 26 F9 FF14 BNE CBKDSL LOOP IF MORE TO DO
01615A FF1B 3F SWI SKIP TO NEW LINE
01616A FF1C 06 A FCB PCRLF FUNCTION
01617A FF1D 39 RTS RETURN TO END COMMAND
01618 * ADD NEW ENTRY
01619A FF1E 8D 20 FF40 CBKADD BSR CBKSET SETUP REGISTERS
01620A FF20 C1 08 A CMPB #NUMBKP ? ALREADY FULL
01621A FF22 27 11 FF35 BEQ CBKERR BRANCH ERROR IF SO
01622A FF24 A6 84 A LDA ,X LOAD BYTE TO TRAP
01623A FF26 E7 84 A STB ,X TRY TO CHANGE
01624A FF28 E1 84 A CMPB ,X ? CHANGABLE RAM
01625A FF2A 26 09 FF35 BNE CBKERR BRANCH ERROR IF NOT
01626A FF2C A7 84 A STA ,X RESTORE BYTE
01627A FF2E 5A CBKADL DECB COUNT DOWN
01628A FF2F 2B 07 FF38 BMI CBKADT BRANCH IF DONE TO ADD IT
01629A FF31 AC A1 A CMPX ,Y++ ? ENTRY ALREADY HERE
01630A FF33 26 F9 FF2E BNE CBKADL LOOP IF NOT
01631A FF35 16 FA24 F95C CBKERR LBRA CMDBAD RETURN TO ERROR PRODUCE
01632A FF38 AF A4 A CBKADT STX ,Y ADD THIS ENTRY
01633A FF3A 6F 31 A CLR -NUMBKP*2+1,Y CLEAR OPTIONAL BYTE
01634A FF3C 0C FA A INC BKPTCT ADD ONE TO COUNT
01635A FF3E 20 D0 FF10 BRA CBKDSP AND NOW DISPLAY ALL OF 'EM
01636 * SETUP REGISTERS FOR SCAN
01637A FF40 9E 9B A CBKSET LDX NUMBER LOAD VALUE DESIRED
01638A FF42 31 8D E06C CBKLDL LEAY BKPTBL,PCR LOAD START OF TABLE
01639A FF46 D6 FA A LDB BKPTCT LOAD ENTRY COUNT
01640A FF48 39 RTS RETURN

```

---

```

01642          *****ENCODE - ENCODE A POSTBYTE
01643A FF49 6F E2 A CENCDE CLR ,-S DEFAULT TO NOT INDIRECT
01644A FF4B 5F CLRB CLRB ZERO POSTBYTE VALUE
01645A FF4C 30 8C 3F LEAX <CONV1,PCR START TABLE SEARCH
01646A FF4F 3F SWI SWI OBTAIN FIRST CHARACTER
01647A FF50 FCB INCHNP FUNCTION
01648A FF51 81 5B A CMPA #'[ ? INDIRECT HERE
01649A FF53 26 06 FF5B BNE CEN2 BRANCH IF NOT
01650A FF55 86 10 A LDA #$10 SET INDIRECT BIT ON
01651A FF57 A7 E4 A STA ,S SAVE FOR LATFR
01652A FF59 3F CENGET SWI OBTAIN NEXT CHARACTER
01653A FF5A 00 A FCB INCHNP FUNCTION
01654A FF5B 81 0D A CEN2 CMPA #CR ? END OF ENTRY
01655A FF5D 27 0C FF6B BEQ CEND1 BRANCH YES
01656A FF5F 6D 84 A CENLP1 TST ,X ? END OF TABLE
01657A FF61 2B D2 FF35 BMI CBKERR BRANCH ERROR IF SO
01658A FF63 A1 81 A CMPA ,X++ ? THIS THE CHARACTER
01659A FF65 26 F8 FF5F BNE CENLP1 BRANCH IF NOT
01660A FF67 EB 1F A ADDB -1,X ADD THIS VALUE
01661A FF69 20 EE FF59 BRA CENGET GET NEXT INPUT
01662A FF6B 30 8C 49 CEND1 LEAX <CONV2,PCR POINT AT TABLE 2
01663A FF6E 1F 98 A TFR B,A SAVE COPY IN A
01664A FF70 84 60 A ANDA #$60 ISOLATE REGISTER MASK
01665A FF72 AA E4 A ORA ,S ADD IN INDIRECTION BIT
01666A FF74 A7 E4 A STA ,S SAVE BACK AS POSTBYTE SKELETON
01667A FF76 C4 9F A ANDB #$9F CLEAR REGISTER BITS
01668A FF78 6D 84 A CENLP2 TST ,X ? END OF TABLE
01669A FF7A 27 B9 FF35 BEQ CBKERR BRANCH ERROR IF SO
01670A FF7C E1 81 A CMPB ,X++ ? SAME VALUE
01671A FF7E 26 F8 FF78 BNE CENLP2 LOOP IF NOT
01672A FF80 E6 1F A LDB -1,X LOAD RESULT VALUE
01673A FF82 EA E4 A ORB ,S ADD TO BASE SKELETON
01674A FF84 E7 E4 A STB ,S SAVE POSTBYTE ON STACK
01675A FF86 30 E4 A LEAX ,S POINT TO IT
01676A FF88 3F SWI SWI SEND OUT AS HEX
01677A FF89 04 A FCB OUT2HS FUNCTION
01678A FF8A 3F SWI SWI TO NEXT LINE
01679A FF8B 06 A FCB PCRLF FUNCTION
01680A FF8C 35 84 A PULS PC,B END OF COMMAND

```

```

01682          * TABLE ONE DEFINES VALID INPUT IN SEQUENCE
01683A FF8E 41 A CONV1 FCB 'A,$04,'B,$05,'D,$06,'H,$01
01684A FF96 48 A FCB 'H,$01,'H,$01,'H,$00,',,$00
01685A FF9E 2D A FCB '-,$09,'-',,$01,'S,$70,'Y,$30
01686A FFA6 55 A FCB 'U,$50,'X,$10,'+,$07,'+,$01
01687A FFAE 50 A FCB 'P,$80,'C,$00,'R,$00,'],,$00
01688A FFB6 FF A FCB $FF END OF TABLE
01689          *CONV2 USES ABOVE CONVERSION TO SET POSTBYTE
01690          * BIT SKELETON.
01691A FFB7 1084 A CONV2 FDB $1084,$1100 R, H,R
01692A FFB8 1288 A FDB $1288,$1389 HH,R HHHH,R
01693A FFBF 1486 A FDB $1486,$1585 A,R B,R
01694A FFC3 168B A FDB $168B,$1780 D,R ,R+
01695A FFC7 1881 A FDB $1881,$1982 ,R++ ,--R
01696A FFCB 1A83 A FDB $1A83,$828C ,--R HH,PCR
01697A FFCF 838D A FDB $838D,$039F HHHH,PCR [HHHH]
01698A FFD3 00 A FCB 0 END OF TABLE

```

---

```

01700 *****
01701 *                DEFAULT INTERRUPT TRANSFERS                *
01702 *****
01703A FFD4 6E    9D DFEE  RSRVD JMP  [VECTAB+.RSVD,PCR] RESERVED VECTOR
01704A FFD8 6E    9D DFEC  SWI3  JMP  [VECTAB+.SWI3,PCR] SWI3 VECTOR
01705A FFDC 6E    9D DFEA  SWI2  JMP  [VECTAB+.SWI2,PCR] SWI2 VECTOR
01706A FFE0 6E    9D DFE8  FIRQ  JMP  [VECTAB+.FIRQ,PCR] FIRQ VECTOR
01707A FFE4 6E    9D DFE6  IRQ   JMP  [VECTAB+.IRQ,PCR]  IRQ VECTOR
01708A FFE8 6E    9D DFE4  SWI   JMP  [VECTAB+.SWI,PCR] SWI VECTOR
01709A FFEC 6E    9D DFE2  NMI   JMP  [VECTAB+.NMI,PCR] NMI VECTOR

```

```

01711 *****
01712 *                ASSIST09 HARDWARE VECTOR TABLE                *
01713 * THIS TABLE IS USED IF THE ASSIST09 ROM ADDRESSES              *
01714 * THE MC6809 HARDWARE VECTORS.                                  *
01715 *****
01716A FFF0                ORG  ROMBEG+ROMSIZ-16 SETUP HARDWARE VECTORS
01717A FFF0                FDB  RSRVD    RESERVED SLOT
01718A FFF2                FDB  SWI3    SOFTWARE INTERRUPT 3
01719A FFF4                FDB  SWI2    SOFTWARE INTERRUPT 2
01720A FFF6                FDB  FIRQ    FAST INTERRUPT REQUEST
01721A FFF8                FDB  IRQ     INTERRUPT REQUEST
01722A FFFA                FDB  SWI     SOFTWARE INTERRUPT
01723A FFFC                FDB  NMI     NON-MASKABLE INTERRUPT
01724A FFFE                FDB  RESET   RESTART

```

```

01726                F837    A        END    RESET
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

```

```

002E .ACIA 00095*00825 00837 00853
0000 .AVTBL 00072*00594
0024 .BSDTA 00090*01508
0026 .BSOFF 00091*01510
0022 .BSON 00089*01507
0016 .CIDTA 00083*00725
0018 .CIOFF 00084*
0014 .CION 00082*00348
0002 .CMDL1 00073*00429
002C .CMDL2 00094*00432
001C .CODTA 00086*00568
001E .COOFF 00087*
001A .COON 00085*00349
0032 .ECHO 00097*00625
002A .EXPAN 00093*01224
000A .FIRQ 00077*01706
0020 .HSDTA 00088*01485
000C .IRQ 00078*01707
0010 .NMI 00080*01709
0030 .PAD 00096*00857 00860 00977 00981 00985 01025 01547
0028 .PAUSE 00092*00724
0034 .PTM 00098*00353 01540

```

```

0012 .RESET 00081*
0004 .RSVD 00074*01703
000E .SWI 00079*01708
0008 .SWI2 00076*01705
0006 .SWI3 00075*01704
E008 ACIA 00024*00256
DF9E ADDR 00133*01239 01391 01392 01402 01421 01431 01437 01448 01456 01460
FDA7 ARMBK2 00773 01357 01369*
FD8E ARMBLP 01356*01360
FDAC ARMLOP 01371*01377
FD9D ARMNSW 01362 01364*
DF9D BASEPG 00135*00186 00784
0007 BELL 00036*00782
DFB2 BKPTBL 00127*01638
DFFA BKPTCT 00121*00386 01370 01594 01607 01634 01639
DFA2 BKPTOP 00129*
F815 BLD2 00192*00196
F821 BLD3 00198*00201
FD46 BLDBAD 01288*01339
FD4D BLDHEX 01250 01301*
FD4F BLDHXC 00421 01302*
FD49 BLDHXI 01233 01299*
FCDF BLDNNB 01164 01219*01397
FCE1 BLDNUM 01222*01286 01492 01588 01592
F835 BLDRTN 00205 00207*
FD58 BLDSHF 01307*01311
F800 BLDVTR 00183*00218
000A BRKPT 00066*01384
FB6A BSDCMP 00942 00944*
FB70 BSDEOL 00940 00948*
FB40 BSDLD1 00919*00922 00949
FB42 BSDLD2 00921*00928
FB60 BSDNXT 00939*00945
FB92 BSPUN 00913 00977*
FB6E BSDSRT 00926 00946*00950
FB38 BSDTA 00250 00911*
FB27 BSOFF 00251 00891*
FB33 BSOFLP 00899*00900
FB1B BSON 00249 00880*
FB22 BSON2 00882 00884*
FBEF BSPEOF 01021 01033*
FBA3 BSPGO 00987*01020
FBC6 BSPMRE 01009*01011
FBAF BSPOK 00990 00992*
FBEC BSPSTR 00997 01032*
FBE7 BSPUN2 01003 01005 01006 01009 01029*
FBE9 BSPUNC 01017 01030*
FB75 BYTE 00930 00933 00935 00939 00953*
FB89 BYTHX 00953 00956 00965*
FB88 BYTRTS 00963*00968
0018 CAN 00040*00711 00718 01338
FF1E CBKADD 01589 01619*
FF2E CBKADL 01627*01630
FF38 CBKADT 01628 01632*
FEFE CBKDLE 01593 01597*
FF07 CBKDLM 01603*01606
FF00 CBKDLP 01598*01601
FF14 CBKDSL 01610*01614
  
```

FF10 CBKDSP 01587 01608\*01635  
FF35 CBKERR 01591 01599 01621 01625 01631\*01657 01669  
FF42 CBKLDL 00303 00383 01354 01369 01638\*  
FEEB CBKPT 00503 01587\*  
FEFD CBKRTS 01595\*01609  
FF40 CBKSET 01597 01608 01619 01637\*  
FE7B CCALBS 01507\*01527  
FDB9 CCALL 00506 01380\*  
FE6E CDBADN 01493 01495 01499\*01512  
FE5A CDCNT 01483 01485\*  
FE43 CDISP 00509 01474\*  
FE51 CDISPS 01478 01481\*  
FE60 CDNUM 01367 01390 01469 01474 01479 01492\*01502 01504 01522 01535 01546  
01552 01561 01563  
FEA8 CDOT 00408 01365 01537\*  
FF5B CEN2 01649 01654\*  
FF49 CENCDE 00512 01643\*  
FF6B CEND1 01655 01662\*  
FF59 CENGET 01652\*01661  
FF5F CENLP1 01656\*01659  
FF78 CENLP2 01668\*01671  
FD80 CGO 00515 01344\*  
FDBF CGOBRK 01383\*01385  
FA58 CHKABT 00701 00709\*00764  
FA61 CHKRTN 00710 00714\*  
FA60 CHKSEC 00713\*00719  
FA62 CHKWT 00712 00715\*00717  
FADC CIDTA 00243 00825\*  
FAF0 CIOFF 00244 00844\*  
FAE6 CION 00242 00835\*  
FAE5 CIRTN 00828 00830\*  
FE8F CLOAD 00518 01516\*  
FE9B CLVDFT 01521 01524\*  
FE92 CLVOFS 01516 01519\*01530  
F8F7 CMD 00354 00380\*00439  
F935 CMD2 00415\*00425  
F948 CMD3 00422 00424\*  
F95C CMDBAD 00435\*00464 01288 01499 01631  
F977 CMDCMP 00450\*00455  
F901 CMDDDL 00387\*00391  
F96C CMDFLS 00444\*00453  
F94D CMDGOT 00416 00427\*  
F990 CMDMEM 00420 00463\*  
F8F9 CMDNEP 00383\*00800  
F90A CMDNOL 00384 00388 00392\*00462  
F953 CMDSCH 00430\*00434 00445  
F96F CMDSIZ 00443 00446\*  
F967 CMDSME 00431 00441\*  
F99B CMDTB2 00254 00496\*  
F99C CMDTBL 00233 00500\*  
F987 CMDXQT 00410 00413 00459\*00467  
FDC3 CMEM 00521 01390\*  
FDC8 CMEM2 01392\*01424 01441  
FDD1 CMEM4 01397\*01404 01408  
FDC6 CMEMN 00465 01391\*  
FDE0 CMENUM 01398 01405\*  
FDEC CMESTR 01412\*01417  
FE02 CMNOTB 01420 01426\*

---  
FDE8 CMNOTC 01401 01410\*  
FE0E CMNOTL 01427 01434\*  
FDF8 CMNOTQ 01411 01419\*  
FE1C CMNOTU 01435 01443\*  
FE18 CMPADP 00411 00465 01432 01440\*  
FE16 CMPADS 01438\*01444  
FDFE CMSPCE 01414 01422\*  
FEB7 CNULLS 00524 01546\*  
FD74 CNVGOT 01325 01331\*  
FD62 CNVHEX 00967 01302 01322\*  
FD76 CNVOK 01312 01332\*  
FD78 CNVRTS 01287 01303 01323 01327 01329 01333\*01372  
FAF1 CODTA 00246 00852\*  
FB0F CODTAD 00869\*00872  
FB12 CODTAO 00854 00864 00870\*  
FB07 CODTLP 00864\*00866  
FB03 CODTPD 00859 00861\*  
FB0D CODTRT 00856 00867\*  
FEC8 COFFS 00527 01561\*  
FEDF COFNO1 01572 01575\*  
FF8E CONV1 01645 01683\*  
FFB7 CONV2 01662 01691\*  
FAF0 COOFF 00247 00845\*  
FAE6 COON 00245 00836\*  
FE71 CPUNCH 00530 01502\*  
000D CR 00038\*00427 00621 00667 00858 01034 01166 01185 01428 01496 01654  
FC4A CREG 00533 01102\*  
FEBC CSTLEV 00536 01551\*  
FEA4 CTRACE 00539 01535\*  
FEAA CTRCE3 00766 01538\*  
FEA1 CVER 00542 01530\*  
FE3E CWINDO 00545 01469\*  
DF8E DELIM 00153\*00751 00757 01223 01236 01256  
0000 DFTCHP 00026\*00257  
0005 DFTNLP 00027\*00257  
0010 DLE 00039\*00855  
0004 EOT 00035\*00343 00652 00684 00738 00782 01032 01034  
FABD ERRMSG 00436 00782\*00789  
FACE ERROR 00314 00789\*  
FCE9 EXPL 00253 01232\*  
FD07 EXP2 01234 01250\*01251  
FD23 EXPADD 01266\*01282  
FD17 EXPCDL 01252 01260\*01269  
FD2B EXPCHM 01262 01270\*  
FCEB EXPDLM 01233\*01237  
FD05 EXPRTN 01248\*01257 01275  
FD36 EXPSUB 01271 01276\*  
FD0D EXPTDI 01254\*01273  
FD0F EXPTDL 01241 01244 01247 01255\*  
FD42 EXPTRM 01263 01276 01286\*  
FFE0 FIRQ 01706\*01720  
FABC FIRQR 00237 00816\*  
FD83 GOADDR 01344 01349\*01380  
FDA2 GONDFT 01351 01367\*  
0034 HIVTR 00100\*00592  
FC00 HSBLNK 01046\*01049  
FC47 HSDRTN 01062 01086 01092\*  
FBFC HSDTA 00248 01043\*01091

```

FC2B HSHCHR 01076*01084
FC35 HSHCOK 01079 01081*
FC33 HSHDOT 01077 01080*
FC14 HSHLNE 01060*01090
FC20 HSHNXT 01068*01071
FC06 HSHRTL 01051*01059
0000 INCHNP 00056*00920 00924 00966 01337 01647 01653
F844 INITVT 00188 00233*
F87D INTVE 00197 00264*
F870 INTVS 00197 00256*
FFE4 IRQ 01707*01721
FAD8 IRQR 00238 00808*
DF99 LASTOP 00139*00752 01539
FAC1 LDDP 00297 00740 00784*00809
000A LF 00037*00623 00638 00669 01034 01426
DF8F MISFLG 00151*00402 00619 00741 00772 00886 00897 01364
0008 MONITR 00064*00222
FA79 MSHOWP 00738*00748
FE36 MUPBAD 01459 01462*
FE2B MUPDAT 01406 01416 01456*
FFEC NMI 01709*01723
FAB7 NMICON 00742 00772*
FA7D NMIR 00240 00740*
FAB0 NMITRC 00744 00747 00766*
DF9B NUMBER 00137*00401 00466 01179 01255 01260 01265 01267 01278 01299 01300
      01308 01309 01405 01497 01637
0008 NUMBKP 00029*00126 00128 00389 01358 01374 01620 01633
000B NUMFUN 00068*00313
001B NUMVTR 00099*00124 00190
0004 OUT2HS 00060*01069 01156 01574 01677
0005 OUT4HS 00061*00754 01065 01153 01452 01579 01612
0001 OUTCH 00057*00396 00885 00893 00896 00983 01082 01142 01146 01396 01430
      01465
000B PAUSE 00067*
DFFC PAUSER 00117*00252
DF93 PCNTER 00145*00393 01242
0006 PCRLF 00062*00381 01044 01061 01093 01161 01439 01581 01616 01679
0003 PDATA 00059*00352 00791 00999 01023
0002 PDATA1 00058*00438 00750
003E PROMPT 00028*00394
FE21 PRTADR 01440 01448*
DF95 PSTACK 00143*00398 00435
E000 PTM 00025*00042 00043 00044 00045 00046 00047 00259 00355 00356 00358
      00359 00361 01542
E000 PTMC13 00043*00359
E001 PTMC2 00044*00358 00361
E001 PTMSTA 00042*
E002 PTMTM1 00045*00355 00356 01542
E004 PTMTM2 00046*
E006 PTMTM3 00047*
E700 RAMOFS 00021*00111
FD79 READ 00407 00424 01258 01301 01336*01412
FC94 REG4 01157*01176 01186
FCC3 REGAGN 01167 01189*
FC70 REGCHG 01104 01135*
FC9D REGCNG 01149 01164*
FC50 REGMSK 01123*01137
FCB1 REGNXC 01165 01177*

```



FC78 REGP1 01138\*01143 01159  
 FC81 REGP2 01140 01144\*  
 FC92 REGP3 01151 01155\*  
 FAB3 REGPRS 00755 00768\*00799  
 FC6F REGPRT 00768 01102 01134\*  
 FC9B REGRTN 01162\*01201  
 FCAA REGSKP 01172\*01175  
 FCC9 REGTF1 01191\*01194  
 FCD6 REGTF2 01197\*01200  
 FCBB REGTWO 01181 01183\*  
 F837 RESET 00217\*00241 01724 01726  
 F83D RESET2 00219\*00223  
 F000 ROM2OF 00023\*00202  
 DF66 ROM2WK 00155\*  
 F800 ROMBEG 00020\*00023 00111 00167 01716  
 0800 ROMSIZ 00022\*00023 01716  
 FFD4 RSRVD 01703\*01717  
 FAD8 RSRVDR 00234 00809\*  
 DF97 RSTACK 00141\*00345 00788  
 FABC RTI 00774\*00816  
 FAF0 RTS 00787 00841\*00844 00845  
 F9EC SEND 00568\*00624 00640 00668 00682  
 F8C9 SIGNON 00342\*00350  
 008C SKIP2 00049\*00863 01154 01220 01523  
 DFF8 SLEVEL 00123\*00746 01553 01556  
 0007 SPACE 00063\*01047 01054 01056 01073 01173 01423  
 DF51 STACK 00158\*00217  
 FEC3 STLDFT 01551 01555\*  
 FFE8 SWI 01708\*01722  
 FFDC SWI2 01705\*01719  
 FAD8 SWI2R 00236 00806\*  
 FFD8 SWI3 01704\*01718  
 FAD8 SWI3R 00235 00807\*  
 DFFB SWIBFL 00119\*00301 00311 01363  
 DF90 SWICNT 00149\*00296 00641 00743  
 F8B5 SWIDNE 00302 00306 00311\*  
 F8A8 SWILP 00305\*00308  
 F895 SWIR 00239 00296\*  
 F87D SWIVTB 00283\*00283 00284 00285 00286 00287 00288 00289 00290 00291 00292  
 00293 00294 00317  
 DF91 TRACEC 00147\*00403 00759 00762 01536  
 DF51 TSTACK 00157\*01189  
 0009 VCTRSW 00065\*  
 DFC2 VECTAB 00125\*00183 00348 00349 00353 00429 00432 00568 00594 00625 00724  
 00725 00825 00837 00853 00857 00860 00977 00981 00985 01025 01224  
 01485 01507 01508 01510 01540 01547 01703 01704 01705 01706 01707  
 01708 01709  
 DFA0 WINDOW 00131\*01245 01470  
 DF00 WORKPG 00111\*00112 00113  
 FA72 XQCIDT 00612 00709 00716 00725\*  
 FA6E XQPAUS 00611 00700 00715 00724\*00869  
 FAD5 ZBKCMD 00756 00758 00760 00763 00765 00800\*  
 FAD3 ZBKPNT 00293 00310 00799\*00810  
 FA2A ZIN2 00622 00625\*  
 FA11 ZINCH 00283 00612\*00615 00617  
 FA0F ZINCHP 00611\*00613  
 F8E6 ZMONT2 00347 00353\*  
 F8D2 ZMONTR 00291 00345\*

---

PAGE 038 ASSIST09.SA:0

ASSIST09 - MC6809 MONITOR

F9F2 ZOT2HS 00287 00571\*  
F9F0 ZOT4HS 00288 00570\*  
FA2E ZOTCH1 00284 00636\*  
FA37 ZOTCH2 00582 00640\*  
FA39 ZOTCH3 00593 00598 00600 00620 00626 00641\*00704  
F9D9 ZOUT2H 00557\*00570 00571 01030 01393  
F9E6 ZOUTHX 00561 00564\*01052  
FA4E ZPAUSE 00294 00700\*  
FA3D ZPCRLF 00289 00654\*  
FA3C ZPCRLS 00637 00652\*00654  
FA40 ZPDATA 00286 00667\*  
FA48 ZPDTA1 00285 00683\*  
FA46 ZPDTLP 00639 00682\*00685  
F9F6 ZSPACE 00290 00581\*  
F9FA ZVSWTH 00292 00591\*

## **APPENDIX C MACHINE CODE TO INSTRUCTION CROSS REFERENCE**

### **C.1 INTRODUCTION**

This appendix contains a cross reference between the machine code, represented in hexadecimal and the instruction and addressing mode that it represents. The number of MPU cycles and the number of program bytes is also given. Refer to Table C-1.

**Table C-1. Machine Code to Instruction Cross Reference**

| OP | Mnem     | Mode     | ~  | #    | OP         | Mnem       | Mode     | ~        | #        | OP       | Mnem  | Mode    | ~        | #       |    |
|----|----------|----------|----|------|------------|------------|----------|----------|----------|----------|-------|---------|----------|---------|----|
| 00 | NEG      | Direct   | 6  | 2    | 30         | LEAX       | Indexed  | 4+       | 2+       | 60       | NEG   | Indexed | 6+       | 2+      |    |
| 01 | *        | ↑        |    |      | 31         | LEAY       | ↑        | 4+       | 2+       | 61       | *     | ↑       |          |         |    |
| 02 | *        |          | 32 | LEAS | ↓          | 4+         |          | 2+       |          |          |       |         |          |         |    |
| 03 | COM      |          | 6  | 2    | 33         | LEAU       | Indexed  | 4+       | 2+       | 63       | COM   |         | 6+       | 2+      |    |
| 04 | LSR      |          | 6  | 2    | 34         | PSHS       | Immed    | 5+       | 2        | 64       | LSR   |         | 6+       | 2+      |    |
| 05 | *        |          |    |      | 35         | PULS       | ↑        | 5+       | 2        | 65       | *     |         |          |         |    |
| 06 | ROR      |          | 6  | 2    | 36         | PSHU       |          | ↓        | 5+       | 2        | 66    |         | ROR      | 6+      | 2+ |
| 07 | ASR      |          | 6  | 2    | 37         | PULU       | Immed    | 5+       | 2        | 67       | ASR   |         | 6+       | 2+      |    |
| 08 | ASL, LSL |          | 6  | 2    | 38         | *          | Inherent | ↑        |          |          | 68    |         | ASL, LSL | 6+      | 2+ |
| 09 | ROL      |          | 6  | 2    | 39         | RTS        | 5        |          | 1        | 69       | ROL   |         | 6+       | 2+      |    |
| 0A | DEC      |          | 6  | 2    | 3A         | ABX        | 3        |          | 1        | 6A       | DEC   |         | 6+       | 2+      |    |
| 0B | *        |          |    | 3B   | RTI        | 6/15       | 1        |          | 6B       | *        |       |         |          |         |    |
| 0C | INC      | 6        | 2  | 3C   | CWAI       | 20         | 2        |          | 6C       | INC      | 6+    | 2+      |          |         |    |
| 0D | TST      | 6        | 2  | 3D   | MUL        | 11         | 1        |          | 6D       | TST      | 6+    | 2+      |          |         |    |
| 0E | JMP      | 3        | 2  | 3E   | *          | ↓          |          |          |          | 6E       | JMP   | 3+      | 2+       |         |    |
| 0F | CLR      | Direct   | 6  | 2    | 3F         |            | SWI      |          | Inherent | 19       | 1     | 6F      | CLR      | Indexed | 6+ |
| 10 | Page 2   | -        | -  | -    | 40         | NEGA       | Inherent |          | 2        | 1        | 70    | NEG     | Extended | 7       | 3  |
| 11 | Page 3   | -        | -  | -    | 41         | *          | ↑        |          |          |          | 71    | *       | ↑        |         |    |
| 12 | NOP      | Inherent | 2  | 1    | 42         | *          |          |          |          | 72       | *     |         |          |         |    |
| 13 | SYNC     | Inherent | 4  | 1    | 43         | COMA       |          | 2        | 1        | 73       | COM   | 7       |          | 3       |    |
| 14 | *        |          |    | 44   | LSRA       | 2          |          | 1        | 74       | LSR      | 7     | 3       |          |         |    |
| 15 | *        |          |    | 45   | *          |            |          |          | 75       | *        |       |         |          |         |    |
| 16 | LBRA     | Relative | 5  | 3    | 46         | RORA       |          | 2        | 1        | 76       | ROR   | 7       |          | 3       |    |
| 17 | LBSR     | Relative | 9  | 3    | 47         | ASRA       |          | 2        | 1        | 77       | ASR   | 7       |          | 3       |    |
| 18 | *        |          |    | 48   | ASLA, LSLA | 2          |          | 1        | 78       | ASL, LSL | 7     | 3       |          |         |    |
| 19 | DAA      | Inherent | 2  | 1    | 49         | ROLA       |          | 2        | 1        | 79       | ROL   | 7       |          | 3       |    |
| 1A | ORCC     | Immed    | 3  | 2    | 4A         | DECA       |          | 2        | 1        | 7A       | DEC   | 7       |          | 3       |    |
| 1B | *        | -        |    |      | 4B         | *          |          |          | 7B       | *        |       |         |          |         |    |
| 1C | ANDCC    | Immed    | 3  | 2    | 4C         | INCA       | 2        | 1        | 7C       | INC      | 7     | 3       |          |         |    |
| 1D | SEX      | Inherent | 2  | 1    | 4D         | TSTA       | 2        | 1        | 7D       | TST      | 7     | 3       |          |         |    |
| 1E | EXG      | Immed    | 8  | 2    | 4E         | *          | ↓        |          |          | 7E       | JMP   | 4       | 3        |         |    |
| 1F | TFR      | Immed    | 6  | 2    | 4F         | CLRA       |          | Inherent | 2        | 1        | 7F    | CLR     | Extended | 7       | 3  |
| 20 | BRA      | Relative | 3  | 2    | 50         | NEGB       |          | Inherent | 2        | 1        | 80    | SUBA    | Immed    | 2       | 2  |
| 21 | BRN      | ↑        | 3  | 2    | 51         | *          |          | ↑        |          |          | 81    | CMPA    | ↑        | 2       | 2  |
| 22 | BHI      |          | 3  | 2    | 52         | *          |          |          |          |          | 82    | SBCA    |          | 2       | 2  |
| 23 | BLS      |          | 3  | 2    | 53         | COMB       |          |          | 2        | 1        | 83    | SUBD    |          | 4       | 3  |
| 24 | BHS, BCC |          | 3  | 2    | 54         | LSRB       |          |          | 2        | 1        | 84    | ANDA    |          | 2       | 2  |
| 25 | BLO, BCS |          | 3  | 2    | 55         | *          |          |          |          |          | 85    | BITA    |          | 2       | 2  |
| 26 | BNE      |          | 3  | 2    | 56         | RORB       |          |          | 2        | 1        | 86    | LDA     |          | 2       | 2  |
| 27 | BEQ      |          | 3  | 2    | 57         | ASRB       |          |          | 2        | 1        | 87    | *       |          |         |    |
| 28 | BVC      |          | 3  | 2    | 58         | ASLB, LSLB | 2        |          | 1        | 88       | EORA  | 2       |          | 2       |    |
| 29 | BVS      |          | 3  | 2    | 59         | ROLB       | 2        |          | 1        | 89       | ADCA  | 2       |          | 2       |    |
| 2A | BPL      |          | 3  | 2    | 5A         | DECB       | 2        |          | 1        | 8A       | ORA   | 2       |          | 2       |    |
| 2B | BMI      | 3        | 2  | 5B   | *          |            |          | 8B       | ADDA     | 2        | 2     |         |          |         |    |
| 2C | BGE      | 3        | 2  | 5C   | INCB       | 2          | 1        | 8C       | CMPX     | Immed    | 4     | 3       |          |         |    |
| 2D | BLT      | 3        | 2  | 5D   | TSTB       | 2          | 1        | 8D       | BSR      | Relative | 7     | 2       |          |         |    |
| 2E | BGT      | 3        | 2  | 5E   | *          | ↓          |          |          | 8E       | LDX      | Immed | 3       | 3        |         |    |
| 2F | BLE      | Relative | 3  | 2    | 5F         |            | CLRB     | Inherent | 2        | 1        |       | 8F      | *        |         |    |

**LEGEND:**

- ~ Number of MPU cycles (less possible push pull or indexed-mode cycles)
- # Number of program bytes
- \* Denotes unused opcode

**Table C-1. Machine Code to Instruction Cross Reference (Continued)**

| OP | Mnem | Mode     | ~  | #  | OP | Mnem  | Mode     | ~  | #  | OP | Mnem | Mode | ~ | # |
|----|------|----------|----|----|----|-------|----------|----|----|----|------|------|---|---|
| 90 | SUBA | Direct   | 4  | 2  | C0 | SUBB  | Immed    | 2  | 2  |    |      |      |   |   |
| 91 | CMPA | 4        | 4  | 2  | C1 | CMPB  | 2        | 2  | 2  |    |      |      |   |   |
| 92 | SBCA | 4        | 4  | 2  | C2 | SBCB  | 2        | 2  | 2  |    |      |      |   |   |
| 93 | SUBD | 6        | 6  | 2  | C3 | ADDD  | 4        | 3  | 4  |    |      |      |   |   |
| 94 | ANDA | 4        | 4  | 2  | C4 | ANDB  | 2        | 2  | 2  |    |      |      |   |   |
| 95 | BITA | 4        | 4  | 2  | C5 | BITB  | Immed    | 2  | 2  |    |      |      |   |   |
| 96 | LDA  | 4        | 4  | 2  | C6 | LDB   | Immed    | 2  | 2  |    |      |      |   |   |
| 97 | STA  | 4        | 4  | 2  | C7 | *     |          |    |    |    |      |      |   |   |
| 98 | EORA | 4        | 4  | 2  | C8 | EORB  | 2        | 2  | 2  |    |      |      |   |   |
| 99 | ADCA | 4        | 4  | 2  | C9 | ADCB  | 2        | 2  | 2  |    |      |      |   |   |
| 9A | ORA  | 4        | 4  | 2  | CA | ORB   | 2        | 2  | 2  |    |      |      |   |   |
| 9B | ADDA | 4        | 4  | 2  | CB | ADG 3 | 2        | 2  | 2  |    |      |      |   |   |
| 9C | CMPX | 6        | 6  | 2  | CC | LDD   | 3        | 3  | 3  |    |      |      |   |   |
| 9D | JSR  | 7        | 7  | 2  | CD | *     |          |    |    |    |      |      |   |   |
| 9E | LDX  | 5        | 5  | 2  | CE | LDU   | Immed    | 3  | 3  |    |      |      |   |   |
| 9F | STX  | Direct   | 5  | 2  | CF | *     |          |    |    |    |      |      |   |   |
| A0 | SUBA | Indexed  | 4+ | 2+ | D0 | SUBB  | Direct   | 4  | 2  |    |      |      |   |   |
| A1 | CMPA | 4+       | 4+ | 2+ | D1 | CMPB  | 4        | 2  | 4  |    |      |      |   |   |
| A2 | SBCA | 4+       | 4+ | 2+ | D2 | SBCB  | 4        | 2  | 4  |    |      |      |   |   |
| A3 | SUBD | 6+       | 6+ | 2+ | D3 | ADDD  | 6        | 2  | 6  |    |      |      |   |   |
| A4 | ANDA | 4+       | 4+ | 2+ | D4 | ANDB  | 4        | 2  | 4  |    |      |      |   |   |
| A5 | BITA | 4+       | 4+ | 2+ | D5 | BITB  | 4        | 2  | 4  |    |      |      |   |   |
| A6 | LDA  | 4+       | 4+ | 2+ | D6 | LDB   | 4        | 2  | 4  |    |      |      |   |   |
| A7 | STA  | 4+       | 4+ | 2+ | D7 | STB   | 4        | 2  | 4  |    |      |      |   |   |
| A8 | EORA | 4+       | 4+ | 2+ | D8 | EORB  | 4        | 2  | 4  |    |      |      |   |   |
| A9 | ADCA | 4+       | 4+ | 2+ | D9 | ADCB  | 4        | 2  | 4  |    |      |      |   |   |
| AA | ORA  | 4+       | 4+ | 2+ | DA | ORB   | 4        | 2  | 4  |    |      |      |   |   |
| AB | ADDA | 4+       | 4+ | 2+ | DB | ADDB  | 4        | 2  | 4  |    |      |      |   |   |
| AC | CMPX | 6+       | 6+ | 2+ | DC | LDD   | 5        | 2  | 5  |    |      |      |   |   |
| AD | JSR  | 7+       | 7+ | 2+ | DD | STD   | 5        | 2  | 5  |    |      |      |   |   |
| AE | LDX  | 5+       | 5+ | 2+ | DE | LDU   | 5        | 2  | 5  |    |      |      |   |   |
| AF | STX  | Indexed  | 5+ | 2+ | DF | STU   | Direct   | 5  | 2  |    |      |      |   |   |
| B0 | SUBA | Extended | 5  | 3  | E0 | SUBB  | Indexed  | 4+ | 2+ |    |      |      |   |   |
| B1 | CMPA | 5        | 5  | 3  | E1 | CMPB  | 4+       | 2+ | 4+ |    |      |      |   |   |
| B2 | SBCA | 5        | 5  | 3  | E2 | SBCB  | 4+       | 2+ | 4+ |    |      |      |   |   |
| B3 | SUBD | 7        | 7  | 3  | E3 | ADDD  | 6+       | 2+ | 6+ |    |      |      |   |   |
| B4 | ANDA | 5        | 5  | 3  | E4 | ANDB  | 4+       | 2+ | 4+ |    |      |      |   |   |
| B5 | BITA | 5        | 5  | 3  | E5 | BITB  | 4+       | 2+ | 4+ |    |      |      |   |   |
| B6 | LDA  | 5        | 5  | 3  | E6 | LDB   | 4+       | 2+ | 4+ |    |      |      |   |   |
| B7 | STA  | 5        | 5  | 3  | E7 | STB   | 4+       | 2+ | 4+ |    |      |      |   |   |
| B8 | EORA | 5        | 5  | 3  | E8 | EORB  | 4+       | 2+ | 4+ |    |      |      |   |   |
| B9 | ADCA | 5        | 5  | 3  | E9 | ADCB  | 4+       | 2+ | 4+ |    |      |      |   |   |
| BA | ORA  | 5        | 5  | 3  | EA | ORB   | 4+       | 2+ | 4+ |    |      |      |   |   |
| BB | ADDA | 5        | 5  | 3  | EB | ADDB  | 4+       | 2+ | 4+ |    |      |      |   |   |
| BC | CMPX | 7        | 7  | 3  | EC | LDD   | 5+       | 2+ | 5+ |    |      |      |   |   |
| BD | JSR  | 8        | 8  | 3  | ED | STD   | 5+       | 2+ | 5+ |    |      |      |   |   |
| BE | LDX  | 6        | 6  | 3  | EE | LDU   | 5+       | 2+ | 5+ |    |      |      |   |   |
| BF | STX  | Extended | 6  | 3  | EF | STU   | Indexed  | 5+ | 2+ |    |      |      |   |   |
|    |      |          |    |    | F0 | SUBB  | Extended | 5  | 3  |    |      |      |   |   |
|    |      |          |    |    | F1 | CMPB  | 5        | 3  | 5  |    |      |      |   |   |
|    |      |          |    |    | F2 | SBCB  | 5        | 3  | 5  |    |      |      |   |   |
|    |      |          |    |    | F3 | ADDD  | 7        | 3  | 7  |    |      |      |   |   |
|    |      |          |    |    | F4 | ANDB  | 5        | 3  | 5  |    |      |      |   |   |
|    |      |          |    |    | F5 | BITB  | 5        | 3  | 5  |    |      |      |   |   |
|    |      |          |    |    | F6 | LDB   | 5        | 3  | 5  |    |      |      |   |   |
|    |      |          |    |    | F7 | STB   | 5        | 3  | 5  |    |      |      |   |   |
|    |      |          |    |    | F8 | EORB  | 5        | 3  | 5  |    |      |      |   |   |
|    |      |          |    |    | F9 | ADCB  | 5        | 3  | 5  |    |      |      |   |   |
|    |      |          |    |    | FA | ORB   | 5        | 3  | 5  |    |      |      |   |   |
|    |      |          |    |    | FB | ADDB  | Extended | 5  | 3  |    |      |      |   |   |
|    |      |          |    |    | FC | LDD   | Extended | 6  | 3  |    |      |      |   |   |
|    |      |          |    |    | FD | STD   | 6        | 3  | 6  |    |      |      |   |   |
|    |      |          |    |    | FE | LDU   | 6        | 3  | 6  |    |      |      |   |   |
|    |      |          |    |    | FF | STU   | Extended | 6  | 3  |    |      |      |   |   |

Page 2 and 3 Machine Codes

NOTE: All unused opcodes are both undefined and illegal



## APPENDIX D PROGRAMMING AID

### D.1 INTRODUCTION

This appendix contains a compilation of data that will assist you in programming the M6809 processor. Refer to Table D-1.

**Table D-1. Programming Aid**

Branch Instructions

| Instruction | Forms       | Addressing Mode |      |   | Description                   | 5 | 3 | 2 | 1 | 0 |
|-------------|-------------|-----------------|------|---|-------------------------------|---|---|---|---|---|
|             |             | Relative        |      |   |                               |   |   |   |   |   |
|             |             | OP              | ~    | # |                               |   |   |   |   |   |
| BCC         | BCC<br>LBCC | 24              | 3    | 2 | Branch C=0                    | • | • | • | • | • |
|             |             | 10              | 5(6) | 4 | Long Branch<br>C=0            | • | • | • | • | • |
|             |             | 24              |      |   |                               |   |   |   |   |   |
| BCS         | BCS<br>LBCS | 25              | 3    | 2 | Branch C=1                    | • | • | • | • | • |
|             |             | 10              | 5(6) | 4 | Long Branch<br>C=1            | • | • | • | • | • |
|             |             | 25              |      |   |                               |   |   |   |   |   |
| BEQ         | BEQ<br>LBEQ | 27              | 3    | 2 | Branch Z=0                    | • | • | • | • | • |
|             |             | 10              | 5(6) | 4 | Long Branch<br>Z=0            | • | • | • | • | • |
|             |             | 27              |      |   |                               |   |   |   |   |   |
| BGE         | BGE<br>LBGE | 2C              | 3    | 2 | Branch ≥ Zero                 | • | • | • | • | • |
|             |             | 10              | 5(6) | 4 | Long Branch ≥ Zero            | • | • | • | • | • |
|             |             | 2C              |      |   |                               |   |   |   |   |   |
| BGT         | BGT<br>LBGT | 2E              | 3    | 2 | Branch > Zero                 | • | • | • | • | • |
|             |             | 10              | 5(6) | 4 | Long Branch > Zero            | • | • | • | • | • |
|             |             | 2E              |      |   |                               |   |   |   |   |   |
| BHI         | BHI<br>LBHI | 22              | 3    | 2 | Branch Higher                 | • | • | • | • | • |
|             |             | 10              | 5(6) | 4 | Long Branch Higher            | • | • | • | • | • |
|             |             | 22              |      |   |                               |   |   |   |   |   |
| BHS         | BHS<br>LBHS | 24              | 3    | 2 | Branch Higher<br>or Same      | • | • | • | • | • |
|             |             | 10              | 5(6) | 4 | Long Branch Higher<br>or Same | • | • | • | • | • |
|             |             | 24              |      |   |                               |   |   |   |   |   |
| BLE         | BLE<br>LBLE | 2F              | 3    | 2 | Branch ≤ Zero                 | • | • | • | • | • |
|             |             | 10              | 5(6) | 4 | Long Branch ≤ Zero            | • | • | • | • | • |
|             |             | 2F              |      |   |                               |   |   |   |   |   |
| BLO         | BLO<br>LBLO | 25              | 3    | 2 | Branch lower                  | • | • | • | • | • |
|             |             | 10              | 5(6) | 4 | Long Branch Lower             | • | • | • | • | • |
|             |             | 25              |      |   |                               |   |   |   |   |   |

| Instruction | Forms       | Addressing Mode |      |   | Description                  | 5 | 3 | 2 | 1 | 0 |
|-------------|-------------|-----------------|------|---|------------------------------|---|---|---|---|---|
|             |             | Relative        |      |   |                              |   |   |   |   |   |
|             |             | OP              | ~    | # |                              |   |   |   |   |   |
| BLS         | BLS<br>LBLS | 23              | 3    | 2 | Branch Lower<br>or Same      | • | • | • | • | • |
|             |             | 10              | 5(6) | 4 | Long Branch Lower<br>or Same | • | • | • | • | • |
|             |             | 23              |      |   |                              |   |   |   |   |   |
| BLT         | BLT<br>LBLT | 2D              | 3    | 2 | Branch < Zero                | • | • | • | • | • |
|             |             | 10              | 5(6) | 4 | Long Branch < Zero           | • | • | • | • | • |
|             |             | 2D              |      |   |                              |   |   |   |   |   |
| BMI         | BMI<br>LBMI | 2B              | 3    | 2 | Branch Minus                 | • | • | • | • | • |
|             |             | 10              | 5(6) | 4 | Long Branch Minus            | • | • | • | • | • |
|             |             | 2B              |      |   |                              |   |   |   |   |   |
| BNE         | BNE<br>LBNE | 26              | 3    | 2 | Branch Z≠0                   | • | • | • | • | • |
|             |             | 10              | 5(6) | 4 | Long Branch<br>Z≠0           | • | • | • | • | • |
|             |             | 26              |      |   |                              |   |   |   |   |   |
| BPL         | BPL<br>LBPL | 2A              | 3    | 2 | Branch Plus                  | • | • | • | • | • |
|             |             | 10              | 5(6) | 4 | Long Branch Plus             | • | • | • | • | • |
|             |             | 2A              |      |   |                              |   |   |   |   |   |
| BRA         | BRA<br>LBRA | 20              | 3    | 2 | Branch Always                | • | • | • | • | • |
|             |             | 16              | 5    | 3 | Long Branch Always           | • | • | • | • | • |
|             |             | 20              |      |   |                              |   |   |   |   |   |
| BRN         | BRN<br>LBRN | 21              | 3    | 2 | Branch Never                 | • | • | • | • | • |
|             |             | 10              | 5    | 4 | Long Branch Never            | • | • | • | • | • |
|             |             | 21              |      |   |                              |   |   |   |   |   |
| BSR         | BSR<br>LBSR | 8D              | 7    | 2 | Branch to Subroutine         | • | • | • | • | • |
|             |             | 17              | 9    | 3 | Long Branch to<br>Subroutine | • | • | • | • | • |
|             |             | 8D              |      |   |                              |   |   |   |   |   |
| BVC         | BVC<br>LBVC | 28              | 3    | 2 | Branch V=0                   | • | • | • | • | • |
|             |             | 10              | 5(6) | 4 | Long Branch<br>V=0           | • | • | • | • | • |
|             |             | 28              |      |   |                              |   |   |   |   |   |
| BVS         | BVS<br>LBVS | 29              | 3    | 2 | Branch V=1                   | • | • | • | • | • |
|             |             | 10              | 5(6) | 4 | Long Branch<br>V=1           | • | • | • | • | • |
|             |             | 29              |      |   |                              |   |   |   |   |   |

**Table D-1. Programming Aid (Continued)**

**SIMPLE BRANCHES**

|      | <u>OP</u> | <u>~</u> | <u>#</u> |
|------|-----------|----------|----------|
| BRA  | 20        | 3        | 2        |
| LBRA | 16        | 5        | 3        |
| BRN  | 21        | 3        | 2        |
| LBRN | 1021      | 5        | 4        |
| BSR  | 8D        | 7        | 2        |
| LBSR | 17        | 9        | 3        |

**SIMPLE CONDITIONAL BRANCHES (Notes 1-4)**

| <u>Test</u> | <u>True</u> | <u>OP</u> | <u>False</u> | <u>OP</u> |
|-------------|-------------|-----------|--------------|-----------|
| N=1         | BMI         | 2B        | BPL          | 2A        |
| Z=1         | BEQ         | 27        | BNE          | 26        |
| V=1         | BVS         | 29        | BVC          | 28        |
| C=1         | BCS         | 25        | BCC          | 24        |

**SIGNED CONDITIONAL BRANCHES (Notes 1-4)**

| <u>Test</u> | <u>True</u> | <u>OP</u> | <u>False</u> | <u>OP</u> |
|-------------|-------------|-----------|--------------|-----------|
| r > m       | BGT         | 2E        | BLE          | 2F        |
| r ≥ m       | BGE         | 2C        | BLT          | 2D        |
| r = m       | BEQ         | 27        | BNE          | 26        |
| r ≤ m       | BLE         | 2F        | BGT          | 2E        |
| r < m       | BLT         | 2D        | BGE          | 2C        |

**UNSIGNED CONDITIONAL BRANCHES (Notes 1-4)**

| <u>Test</u> | <u>True</u> | <u>OP</u> | <u>False</u> | <u>OP</u> |
|-------------|-------------|-----------|--------------|-----------|
| r > m       | BHI         | 22        | BLS          | 23        |
| r ≥ m       | BHS         | 24        | BLO          | 25        |
| r = m       | BEQ         | 27        | BNE          | 26        |
| r ≤ m       | BLS         | 23        | BHI          | 22        |
| r < m       | BLO         | 25        | BHS          | 24        |

**Notes:**

1. All conditional branches have both short and long variations.
2. All short branches are 2 bytes and require 3 cycles.
3. All conditional long branches are formed by prefixing the short branch opcode with \$10 and using a 16-bit destination offset.
4. All conditional long branches require 4 bytes and 6 cycles if the branch is taken or 5 cycles if the branch is not taken.



**Table D-1. Programming Aid (Continued)**

| Instruction | Forms  | Addressing Modes   |   |   |  |  |  |  |                      |  |  |  |  |          |        |        | Description | 5   | 3                                   | 2           | 1 | 0 |   |   |   |
|-------------|--|--|---|---|--|--|--|--|----------------------|--|--|--|--|----------|--------|--------|-------------|---|-------------------------------------|-------------|---|---|---|---|---|
|             |  | Immediate  |   |   | Direct   |  |  | Indexed  |                      |  | Extended   |  |  | Inherent |        |        |             | H   | N                                   | Z           | V | C |   |   |   |
|             |  | Op   | ~   | #   | Op   | ~  | #  | Op   | ~                    | #  | Op   | ~  | #  | Op       | ~      | #      |             | Op  | ~                                   | #           |   |   |   |   |   |
| ABX         |  |  |   |   |  |  |  |  |                      |  |  |  |  |          |        | 3A     | 3           | 1   | B + X - X (Unsigned)                | •           | • | • | • | • |   |
| ADC         | ADCA<br>ADCB   | 89<br>C9   | 2<br>2  | 2<br>2                                    | 99<br>D9   | 4<br>4   | 2<br>2   | A9<br>E9   | 4+<br>4+             | 2+<br>2+   | B9<br>F9   | 5<br>5   | 3<br>3   |          |        |        |             | A + M + C - A<br>B + M + C - B  | †                                   | †           | † | † | † |   |   |
| ADD         | ADDA<br>ADDB<br>ADDD                                 | 8B<br>CB<br>C3   | 2<br>2<br>4   | 2<br>2<br>3                               | 9B<br>DB<br>D3   | 4<br>4<br>6                                    | 2<br>2<br>2                                    | AB<br>EB<br>E3   | 4+<br>4+<br>6+       | 2+<br>2+<br>2+   | BB<br>FB<br>F3   | 5<br>5<br>7                                    | 3<br>3<br>3                                    |          |        |        |             | A + M - A<br>B + M - B<br>D + M: M + 1 - D  | †                                   | †           | † | † | † |   |   |
| AND         | ANDA<br>ANDB<br>ANDCC                                | 84<br>C4<br>1C   | 2<br>2<br>3   | 2<br>2<br>2                               | 94<br>D4   | 4<br>4   | 2<br>2   | A4<br>E4   | 4+<br>4+             | 2+<br>2+   | B4<br>F4   | 5<br>5   | 3<br>3   |          |        |        |             | A Δ M - A<br>B Δ M - B<br>CC Δ IMM - CC   | •                                   | †           | † | 0 | • |   |   |
| ASL         | ASLA<br>ASLB<br>ASL                                  |  |   |   | 08   | 6  | 2  | 68   | 6+                   | 2+   | 78   | 7  | 3  | 48<br>58 | 2<br>2 | 1<br>1 |             |   |                                     | 8<br>8<br>8 | † | † | † | † | † |
| ASR         | ASRB<br>ASR<br>ASR                                   |  |   |   | 07   | 6  | 2  | 67   | 6+                   | 2+   | 77   | 7  | 3  | 47<br>57 | 2<br>2 | 1<br>1 |             |   |                                     | 8<br>8<br>8 | † | † | • | † | † |
| BIT         | BITA<br>BITB   | 85<br>C5   | 2<br>2  | 2<br>2                                    | 95<br>D5   | 4<br>4   | 2<br>2   | A5<br>E5   | 4+<br>4+             | 2+<br>2+   | B5<br>F5   | 5<br>5   | 3<br>3   |          |        |        |             | Bit Test A (M Δ A)<br>Bit Test B (M Δ B)  | •                                   | †           | † | 0 | • |   |   |
| CLR         | CLRA<br>CLRB<br>CLR                                  |  |   |   | 0F   | 6  | 2  | 6F   | 6+                   | 2+   | 7F   | 7  | 3  | 4F<br>5F | 2<br>2 | 1<br>1 |             |   | 0 - A<br>0 - B<br>0 - M             | •           | 0 | 1 | 0 | 0 |   |
| CMP         | CMPA<br>CMPB<br>CMPD<br>CMPS<br>CMPU<br>CMPX<br>CMPY | 81<br>C1<br>10<br>83<br>11<br>8C<br>11<br>83<br>8C<br>10<br>8C | 2<br>2<br>5<br>5<br>5<br>4<br>5<br>4<br>4<br>5<br>4 | 2<br>2<br>4<br>4<br>4<br>3<br>4<br>3<br>4 | 91<br>D1<br>10<br>93<br>11<br>9C<br>11<br>93<br>9C<br>10<br>9C | 4<br>4<br>7<br>7<br>7<br>6<br>7<br>6<br>7<br>7 | 2<br>2<br>3<br>3<br>3<br>2<br>3<br>2<br>3<br>3 | A1<br>E1<br>10<br>A3<br>11<br>AC<br>11<br>A3<br>AC<br>10<br>AC | 4+<br>4+<br>7+<br>7+ | 2+<br>2+<br>3+<br>3+<br>3+<br>2+<br>3+<br>2+<br>2+<br>3+<br>3+ | B1<br>F1<br>10<br>B3<br>11<br>BC<br>11<br>B3<br>BC<br>10<br>BC | 5<br>5<br>8<br>8<br>8<br>7<br>8<br>7<br>8<br>8 | 3<br>3<br>4<br>4<br>4<br>3<br>4<br>3<br>4<br>4 |          |        |        |             | Compare M from A<br>Compare M from B<br>Compare M: M + 1 from D<br>Compare M: M + 1 from S<br>Compare M: M + 1 from U<br>Compare M: M + 1 from X<br>Compare M: M + 1 from Y | 8<br>8<br>•                         | †           | † | † | † |   |   |
| COM         | COMA<br>COMB<br>COM                                  |  |   |   | 03   | 6  | 2  | 63   | 6+                   | 2+   | 73   | 7  | 3  | 43<br>53 | 2<br>2 | 1<br>1 |             |   | A - A<br>B - B<br>M - M             | •           | † | † | 0 | 1 |   |
| CWAI        |  | 3C   | ≥20   | 2   |  |  |  |  |                      |  |  |  |  |          |        |        |             | CC Δ IMM - CC Wait for Interrupt  |                                     |             |   |   | 7 |   |   |
| DAA         |  |  |   |   |  |  |  |  |                      |  |  |  |  | 19       | 2      | 1      |             |   | Decimal Adjust A                    | •           | † | † | 0 | † |   |
| DEC         | DECA<br>DECB<br>DEC                                  |  |   |   | 0A   | 6  | 2  | 6A   | 6+                   | 2+   | 7A   | 7  | 3  | 4A<br>5A | 2<br>2 | 1<br>1 |             |   | A - 1 - A<br>B - 1 - B<br>M - 1 - M | •           | † | † | † | • |   |
| EOR         | EORA<br>EORB   | 88<br>C8   | 2<br>2  | 2<br>2                                    | 98<br>D8   | 4<br>4   | 2<br>2   | A8<br>E8   | 4+<br>4+             | 2+<br>2+   | B8<br>F8   | 5<br>5   | 3<br>3   |          |        |        |             | A ↕ M - A<br>B ↕ M - B  | •                                   | †           | † | 0 | • |   |   |
| EXG         | R1, R2   | 1E   | 8   | 2   |  |  |  |  |                      |  |  |  |  |          |        |        |             | R1 - R2 <sup>2</sup>  | •                                   | •           | • | • | • |   |   |
| INC         | INCA<br>INCB<br>INC                                  |  |   |   | 0C   | 6  | 2  | 6C   | 6+                   | 2+   | 7C   | 7  | 3  | 4C<br>5C | 2<br>2 | 1<br>1 |             |   | A + 1 - A<br>B + 1 - B<br>M + 1 - M | •           | † | † | † | • |   |
| JMP         |  |  |   |   | 0E   | 3  | 2  | 6E   | 3+                   | 2+   | 7E   | 4  | 3  |          |        |        |             | EA <sup>3</sup> - PC  | •                                   | •           | • | • | • |   |   |
| JSR         |  |  |   |   | 9D   | 7  | 2  | AD   | 7+                   | 2+   | BD   | 8  | 3  |          |        |        |             | Jump to Subroutine  | •                                   | •           | • | • | • |   |   |
| LD          | LDA<br>LDB<br>LDD<br>LDS<br>LDU<br>LDX<br>LDY        | 86<br>C6<br>CC<br>10<br>CE<br>8E<br>10<br>8E                   | 2<br>2<br>3<br>4<br>3<br>3<br>4                     | 2<br>2<br>3<br>4<br>3<br>3<br>4           | 96<br>D6<br>DC<br>10<br>DE<br>9E<br>10<br>9E                   | 4<br>4<br>5<br>6<br>5<br>5<br>6                | 2<br>2<br>2<br>3<br>2<br>2<br>3                | A6<br>E6<br>EC<br>10<br>EE<br>AE<br>10<br>AE                   | 4+<br>4+<br>5+<br>6+ | 2+<br>2+<br>2+<br>3+   | B6<br>F6<br>FC<br>10<br>FE<br>BE<br>10<br>BE                   | 5<br>5<br>6<br>7<br>6<br>6<br>7                | 3<br>3<br>3<br>4<br>3<br>3<br>4                |          |        |        |             | M - A<br>M - B<br>M: M + 1 - D<br>M: M + 1 - S<br>M: M + 1 - U<br>M: M + 1 - X<br>M: M + 1 - Y  | •                                   | †           | † | 0 | • |   |   |
| LEA         | LEAS<br>LEAU<br>LEAX<br>LEAY                         |  |   |   |  |  |  | 32<br>33<br>30<br>31   | 4+<br>4+<br>4+<br>4+ | 2+<br>2+<br>2+<br>2+   |  |  |  |          |        |        |             | EA <sup>3</sup> - S<br>EA <sup>3</sup> - U<br>EA <sup>3</sup> - X<br>EA <sup>3</sup> - Y  | •                                   | •           | • | • | • |   |   |

Legend:

- OP Operation Code (Hexadecimal)
- ~ Number of MPU Cycles
- # Number of Program Bytes
- + Arithmetic Plus
- Arithmetic Minus
- Multiply

- M̄ Complement of M
- Transfer Into
- H Half-carry (from bit 3)
- N Negative (sign bit)
- Z Zero (Reset)
- V Overflow, 2's complement
- C Carry from ALU

- † Test and set if true, cleared otherwise
- Not Affected
- CC Condition Code Register
- : Concatenation
- V Logical or
- Δ Logical and
- ↕ Logical Exclusive or

**Table D-1. Programming Aid (Continued)**

| Instruction | Forms  | Addressing Modes |             |             |                |             |             |                      |                |                |                |             |             |          |   |  | Description                               | 5<br>H                                    | 3<br>N   | 2<br>Z   | 1<br>V   | 0<br>C   |   |   |  |   |   |   |   |   |   |   |
|-------------|--|------------------|-------------|-------------|----------------|-------------|-------------|----------------------|----------------|----------------|----------------|-------------|-------------|----------|---|--|---|---|--|--|--|--|---|---|--|---|---|---|---|---|---|---|
|             |  | Immediate        |             |             | Direct         |             |             | Indexed <sup>1</sup> |                |                | Extended       |             |             | Inherent |   |  |   |   |  |  |  |  |   |   |  |   |   |   |   |   |   |   |
|             |  | Op               | ~           | #           | Op             | ~           | #           | Op                   | ~              | #              | Op             | ~           | #           | Op       | ~ | #  |   |   |  |  |  |  |   |   |  |   |   |   |   |   |   |   |
| LSL         | LSLA<br>LSLB<br>LSL  |                  |             |             |                |             |             |                      |                |                |                |             |             |          |   | 48<br>58   | 2<br>2                                    | 1<br>1                                    |  | •  | •  | •  | •   | •   | •  |   |   |   |   |   |   |   |
| LSR         | LSRA<br>LSRB<br>LSR  |                  |             |             |                |             |             |                      |                |                |                |             |             |          |   | 44<br>54   | 2<br>2                                    | 1<br>1                                    |  | •  | •  | •  | •   | •   | •  |   |   |   |   |   |   |   |
| MUL         |  |                  |             |             |                |             |             |                      |                |                |                |             |             |          |   | 3D   | 11  | 1   | A x B -> D (Unsigned)  | •  | •  | •  | •   | •   | 9  |   |   |   |   |   |   |   |
| NEG         | NEGA<br>NEGB<br>NEG  |                  |             |             |                |             |             |                      |                |                |                |             |             |          |   | 40<br>50   | 2<br>2                                    | 1<br>1                                    | A + 1 - A<br>B + 1 - B<br>M + 1 - M                                  | 8  | •  | •  | •   | •   | •  | • |   |   |   |   |   |   |
| NOP         |  |                  |             |             |                |             |             |                      |                |                |                |             |             |          |   | 12   | 2   | 1   | No Operation   | •  | •  | •  | •   | •   | •  |   |   |   |   |   |   |   |
| OR          | ORA<br>ORB<br>ORCC   | 8A<br>CA<br>1A   | 2<br>2<br>3 | 2<br>2<br>2 | 9A<br>DA       | 4<br>4      | 2<br>2      | AA<br>EA             | 4+<br>+        | 2+<br>2+       | BA<br>FA       | 5<br>5      | 3<br>3      |          |   |  |   |   | A V M -> A<br>B V M -> B<br>CC V IMM -> CC                           | •  | •  | •  | •   | •   | •  |   |   |   |   |   |   |   |
| PSH         | PSHS<br>PSHU   | 34<br>36         | 5+4<br>5+4  | 2<br>2      |                |             |             |                      |                |                |                |             |             |          |   |  |   |   | Push Registers on S Stack<br>Push Registers on U Stack               | •  | •  | •  | •   | •   | •  |   |   |   |   |   |   |   |
| PUL         | PULS<br>PULU   | 35<br>37         | 5+4<br>5+4  | 2<br>2      |                |             |             |                      |                |                |                |             |             |          |   |  |   |   | Pull Registers from S Stack<br>Pull Registers from U Stack           | •  | •  | •  | •   | •   | •  |   |   |   |   |   |   |   |
| ROL         | ROLA<br>ROLB<br>ROL  |                  |             |             |                |             |             |                      |                |                |                |             |             |          |   | 49<br>59   | 2<br>2                                    | 1<br>1                                    |  | •  | •  | •  | •   | •   | •  |   |   |   |   |   |   |   |
| ROR         | RORA<br>RORB<br>ROR  |                  |             |             |                |             |             |                      |                |                |                |             |             |          |   | 46<br>56   | 2<br>2                                    | 1<br>1                                    |  | •  | •  | •  | •   | •   | •  |   |   |   |   |   |   |   |
| RTI         |  |                  |             |             |                |             |             |                      |                |                |                |             |             |          |   | 3B   | 6/15                                      | 1   | Return From Interrupt  |  |  |  |   |   | 7  |   |   |   |   |   |   |   |
| RTS         |  |                  |             |             |                |             |             |                      |                |                |                |             |             |          |   | 39   | 5   | 1   | Return from Subroutine   | •  | •  | •  | •   | •   | •  |   |   |   |   |   |   |   |
| SBC         | SBCA<br>SBCB   | 82<br>C2         | 2<br>2      | 2<br>2      | 92<br>D2       | 4<br>4      | 2<br>2      | A2<br>E2             | 4+<br>4+       | 2+<br>2+       | B2<br>F2       | 5<br>5      | 3<br>3      |          |   |  |   |   | A - M - C -> A<br>B - M - C -> B                                     | 8  | •  | •  | •   | •   | •  |   |   |   |   |   |   |   |
| SEX         |  |                  |             |             |                |             |             |                      |                |                |                |             |             |          |   | 1D   | 2   | 1   | Sign Extend B into A   | •  | •  | •  | •   | •   | •  |   |   |   |   |   |   |   |
| ST          | STA<br>STB<br>STD<br>STS<br>STU<br>STX<br>STY              |                  |             |             |                |             |             |                      |                |                |                |             |             |          |   | 97<br>D7<br>DD<br>10<br>DF<br>DF<br>9F<br>10<br>9F | 4<br>4<br>5<br>6<br>5<br>5<br>6<br>5<br>6 | 2<br>2<br>2<br>3<br>2<br>2<br>2<br>2<br>2 | A7<br>E7<br>ED<br>10<br>EF<br>EF<br>AF<br>AF<br>AF                   | 4+<br>4+<br>5+<br>6+<br>5+<br>5+<br>5+<br>5+<br>6+ | 2+<br>2+<br>2+<br>3+<br>2+<br>2+<br>2+<br>2+<br>3+ | B7<br>F7<br>FD<br>10<br>FF<br>FF<br>BF<br>10<br>BF | 5<br>5<br>6<br>7<br>6<br>6<br>6<br>7<br>7 | 3<br>3<br>3<br>4<br>3<br>3<br>3<br>4<br>4 | A - M<br>B - M<br>D - M: M + 1<br>S - M: M + 1<br>U - M: M + 1<br>X - M: M + 1<br>Y - M: M + 1 | • | • | • | • | • | • | • |
| SUB         | SUBA<br>SUBB<br>SUBD                                       | 80<br>C0<br>83   | 2<br>2<br>4 | 2<br>2<br>3 | 90<br>D0<br>93 | 4<br>4<br>6 | 2<br>2<br>2 | A0<br>E0<br>A3       | 4+<br>4+<br>6+ | 2+<br>2+<br>2+ | B0<br>F0<br>B3 | 5<br>5<br>7 | 3<br>3<br>3 |          |   |  |   |   | A - M -> A<br>B - M -> B<br>D - M: M + 1 -> D                        | 8  | •  | •  | •   | •   | •  |   |   |   |   |   |   |   |
| SWI         | SWI <sup>6</sup><br>SWI <sup>26</sup><br>SWI <sup>36</sup> |                  |             |             |                |             |             |                      |                |                |                |             |             |          |   | 3F<br>10<br>3F<br>11<br>3F                         | 19<br>20<br>2                             | 1<br>2<br>1                               | Software Interrupt 1<br>Software Interrupt 2<br>Software Interrupt 3 | •  | •  | •  | •   | •   | •  | • |   |   |   |   |   |   |
| SYNC        |  |                  |             |             |                |             |             |                      |                |                |                |             |             |          |   | 13   | ≥4  | 1   | Synchronize to Interrupt   | •  | •  | •  | •   | •   | •  |   |   |   |   |   |   |   |
| TFR         | R1, R2   | 1F               | 6           | 2           |                |             |             |                      |                |                |                |             |             |          |   |  |   |   | R1 - R2 <sup>2</sup>   | •  | •  | •  | •   | •   | •  |   |   |   |   |   |   |   |
| TST         | TSTA<br>TSTB<br>TST  |                  |             |             |                |             |             |                      |                |                |                |             |             |          |   | 4D<br>5D   | 2<br>2                                    | 1<br>1                                    | Test A<br>Test B<br>Test M   | •  | •  | •  | •   | •   | •  |   |   |   |   |   |   |   |

**Notes:**

- This column gives a base cycle and byte count. To obtain total count, add the values obtained from the INDEXED ADDRESSING MODE table, in Appendix F.
- R1 and R2 may be any pair of 8 bit or any pair of 16 bit registers.  
The 8 bit registers are: A, B, CC, DP  
The 16 bit registers are: X, Y, U, S, D, PC
- EA is the effective address.
- The PSH and PUL instructions require 5 cycles plus 1 cycle for each **byte** pushed or pulled.
- 5(6) means: 5 cycles if branch not taken, 6 cycles if taken (Branch instructions).
- SWI sets I and F bits. SWI2 and SWI3 do not affect I and F.
- Conditions Codes set as a direct result of the instruction.
- Value of half-carry flag is undefined.
- Special Case - Carry set if b7 is SET.

## APPENDIX E ASCII CHARACTER SET

### E.1 INTRODUCTION

This appendix contains the standard 112 character ASCII character set (7-bit code).

### E.2 CHARACTER REPRESENTATION AND CODE IDENTIFICATION

The ASCII character set is given in Figure E-1.

| <div style="display: inline-block; border: 1px solid black; padding: 2px;">                     b7<br/>b6<br/>b5<br/>b4<br/>b3<br/>b2<br/>b1<br/>b0<br/>Bits                 </div> |   |   |   |    | Column |     | 0   | 1  | 2 | 3 | 4 | 5 | 6   | 7 |
|---|---|---|---|----|--------|-----|-----|----|---|---|---|---|-----|---|
|   |   |   |   |    | Row    | Hex | 0   | 1  | 2 | 3 | 4 | 5 | 6   | 7 |
| 0   | 0 | 0 | 0 | 0  | 0      | NUL | DLE | SP | 0 | @ | P | ' | p   |   |
| 0   | 0 | 0 | 1 | 1  | 1      | SOH | DC1 | !  | 1 | A | Q | a | q   |   |
| 0   | 0 | 1 | 0 | 2  | 2      | STX | DC2 | "  | 2 | B | R | b | r   |   |
| 0   | 0 | 1 | 1 | 3  | 3      | ETX | DC3 | #  | 3 | C | S | c | s   |   |
| 0   | 1 | 0 | 0 | 4  | 4      | EOT | DC4 | \$ | 4 | D | T | d | t   |   |
| 0   | 1 | 0 | 1 | 5  | 5      | ENQ | NAK | %  | 5 | E | U | e | u   |   |
| 0   | 1 | 1 | 0 | 6  | 6      | ACK | SYN | &  | 6 | F | V | f | v   |   |
| 0   | 1 | 1 | 1 | 7  | 7      | BEL | ETB | '  | 7 | G | W | g | w   |   |
| 1   | 0 | 0 | 0 | 8  | 8      | BS  | CAN | (  | 8 | H | X | h | x   |   |
| 1   | 0 | 0 | 1 | 9  | 9      | HT  | EM  | )  | 9 | I | Y | i | y   |   |
| 1   | 0 | 1 | 0 | 10 | A      | LF  | SUB | *  | : | J | Z | j | z   |   |
| 1   | 0 | 1 | 1 | 11 | B      | VT  | ESC | +  | ; | K | [ | k | {   |   |
| 1   | 1 | 0 | 0 | 12 | C      | FF  | FS  | ,  | < | L | \ | l |     |   |
| 1   | 1 | 0 | 1 | 13 | D      | CR  | GS  | -  | = | M | ] | m | }   |   |
| 1   | 1 | 1 | 0 | 14 | E      | SO  | RS  | .  | > | N | ^ | n | ~   |   |
| 1   | 1 | 1 | 1 | 15 | F      | SI  | US  | /  | ? | O | _ | o | DEL |   |

**Figure E-1. ASCII Character Set**

Each 7-bit character is represented with bit seven as the high-order bit and bit one as the low-order bit as shown in the following example:

```

b7 b6 b5 b4 b3 b2 b1 b0
1  0  0  0  0  0  0  1

```

The bit representation for the character "A" is developed from the bit pattern for bits seven through five found above the column designated 4 and the bit pattern for bits four through one found to the left of the row designated 1.

A hexadecimal notation is commonly used to indicate the code for each character. This is easily developed by assuming a logic zero in the non-existent bit eight position for the column numbers and using the hexadecimal number for the row numbers.

### E.3 CONTROL CHARACTERS

The characters located in columns zero and one of Figure E-1 are considered control characters. By definition, these are characters whose occurrence in a particular context initiates, modifies, or stops an action that affects the recording, processing, transmission, or interpretation of data. Table E-1 provides the meanings of the control characters.

**Table E-1. Control Characters**

| Mnemonic | Meaning               | Mnemonic | Meaning                   |
|----------|-----------------------|----------|---------------------------|
| NUL      | Null                  | DLE      | Data Link Escape          |
| SOH      | Start of Heading      | DC1      | Device Control 1          |
| STX      | Start of Text         | DC2      | Device Control 2          |
| ETX      | End of Text           | DC3      | Device Control 3          |
| EOT      | End of Transmission   | DC4      | Device Control 4          |
| ENQ      | Enquiry               | NAK      | Negative Acknowledge      |
| ACK      | Acknowledge           | SYN      | Synchronous Idle          |
| BEL      | Bell                  | ETB      | End of Transmission Block |
| BS       | Backspace             | CAN      | Cancel                    |
| HT       | Horizontal Tabulation | EM       | End of Medium             |
| LF       | Line Feed             | SUB      | Substitute                |
| VT       | Vertical Tabulation   | ESC      | Escape                    |
| FF       | Form Feed             | FS       | File Separator            |
| CR       | Carriage Return       | GS       | Group Separator           |
| SO       | Shift Out             | RS       | Record Separator          |
| SI       | Shift In              | US       | Unit Separator            |
|          |                       | DEL      | Delete                    |

### E.4 GRAPHIC CHARACTERS

The characters in columns two through seven are considered graphic characters. These characters have a visual representation which is normally displayed or printed. These characters and their names are given in Table E-2.

**Table E-2. Graphic Characters**

| <b>Symbol</b> | <b>Name</b>  |
|---------------|--|
| SP            | Space (Normally Nonprinting)                             |
| !             | Exclamation Point  |
| ''            | Quotation Marks (Diaeresis)                              |
| #             | Number Sign  |
| \$            | Dollar Sign  |
| %             | Percent Sign   |
| &             | Ampersand  |
| '             | Apostrophe (Closing Single Quotation Mark; Acute Accent) |
| (             | Opening Parenthesis                                      |
| )             | Closing Parenthesis                                      |
| *             | Asterisk   |
| +             | Plus   |
| ,             | Comma (Cedilla)  |
| -             | Hyphen (Minus)   |
| .             | Period (Decimal Point)                                   |
| /             | Slant  |
| 0...9         | Digits 0 Through 9                                       |
| :             | Colon  |
| ;             | Semicolon  |
| <             | Less Than  |
| =             | Equals   |
| >             | Greater Than   |
| ?             | Question Mark  |
| @             | Commercial At  |
| A...Z         | Uppercase Latin Letters A Through Z                      |
| [             | Opening Bracket  |
| \             | Reverse Slant  |
| ]             | Closing Bracket  |
| ^             | Circumflex   |
| —             | Underline  |
| '             | Opening Single Quotation Mark (Grave Accent)             |
| a...z         | Lowercase Latin Letters a Through z                      |
| {             | Opening Brace  |
|               | Vertical Line  |
| }             | Closing Brace  |
| ~             | Tilde  |



## **APPENDIX F OPCODE MAP**

### **F.1 INTRODUCTION**

This appendix contains the opcode map and additional information for calculating required machine cycles.

### **F.2 OPCODE MAP**

Table F-1 is the opcode map for M6809 processors. The number(s) by each instruction indicates the number of machine cycles required to execute that instruction. When the number contains an "I" (e.g., 4 + I), it indicates that the indexed addressing mode is being used and that an additional number of machine cycles may be required. Refer to Table F-2 to determine the additional machine cycles to be added.

Some instructions in the opcode map have two numbers, the second one in parenthesis. This indicates that the instruction involves a branch. The parenthetical number applies if the branch is taken.

The "page 2, page 3" notation in column one means that all page 2 instructions are preceded by a hexadecimal 10 opcode and all page 3 instructions are preceded by a hexadecimal 11 opcode.

Table F-1. Opcode Map

| Most-Significant Four Bits |              |                     |                     |          |      |      |                     |                     |                     |                    |                  |                  |                  |                |     |
|----------------------------|--------------|---------------------|---------------------|----------|------|------|---------------------|---------------------|---------------------|--------------------|------------------|------------------|------------------|----------------|-----|
| DIR                        | REL          | ACCA                | ACCB                | IND      | EXT  | IMM  | DIR                 | IND                 | EXT                 | IMM                | DIR              | IND              | EXT              | IND            | EXT |
| 0000                       | 0010         | 0100                | 0101                | 0110     | 0111 | 1000 | 1001                | 1010                | 1011                | 1100               | 1101             | 1110             | 1111             |                |     |
| 0                          | 1            | 2                   | 3                   | 4        | 5    | 6    | 7                   | 8                   | 9                   | A                  | B                | C                | D                | E              | F   |
| 0000 0                     | NEG          | 3 BRA               | 4+1 LEAX            | 2        | 2    | 2    | 4                   | 4+1                 | 5                   | 2                  | 4                | SUBB             | 4+1              | 5              | 0   |
| 0001 1                     |              | 3 BRN/<br>5 LBRN    | 4+1 LEAY            | 2        | 2    | 2    | 4                   | 4+1                 | 5                   | 2                  | 4                | CMPB             | 4+1              | 5              | 1   |
| 0010 2                     |              | 2                   |                     | 2        | 2    | 2    | 4                   | 4+1                 | 5                   | 2                  | 4                | SBCB             | 4+1              | 5              | 2   |
| 0011 3                     | COM          | 2 SYNC              | 3 BLS/<br>5(6) LBLS | 4+1 LEAU | 2    | 2    | 4,6,6+1,7<br>SUBD / | 5,7,7+1,8<br>CMPD / | 5,7,7+1,8<br>CMPS / | 4                  | 6                | ADDD             | 6+1              | 7              | 3   |
| 0100 4                     | LSR          | 3 BHS<br>5(6) (BCC) | 5+1/by<br>PSHS      | 2        | 2    | 2    | 4                   | 4+1                 | 5                   | 2                  | 4                | ANDB             | 4+1              | 5              | 4   |
| 0101 5                     |              | 3 BLO<br>5(6) (BCS) | 5+1/by<br>PULS      | 2        | 2    | 2    | 4                   | 4+1                 | 5                   | 2                  | 4                | BITB             | 4+1              | 5              | 5   |
| 0110 6                     | ROR          | 3 BNE/<br>5(6) LBNE | 5+1/by<br>PSHU      | 2        | 2    | 2    | 4                   | 4+1                 | 5                   | 2                  | 4                | LDB              | 4+1              | 5              | 6   |
| 0111 7                     | ASR          | 3 BEQ/<br>5(6) LBEO | 5+1/by<br>PULU      | 2        | 2    | 2    | 4                   | 4+1                 | 5                   | 2                  | 4                | STB              | 4+1              | 5              | 7   |
| 1000 8                     | ASL<br>(LSL) | 3 BVC/<br>5(6) LBVC |                     | 2        | 2    | 2    | 4                   | EORA                | 5                   | 2                  | 4                | EORB             | 4+1              | 5              | 8   |
| 1001 9                     | ROL          | 3 BVS/<br>5(6) LBVS | 5                   | 2        | 2    | 2    | 4                   | ADCA                | 5                   | 2                  | 4                | ADCB             | 4+1              | 5              | 9   |
| 1010 A                     | DEC          | 3 BPL/<br>5(6) LBPL | 3                   | 2        | 2    | 2    | 4                   | ORA                 | 5                   | 2                  | 4                | ORB              | 4+1              | 5              | A   |
| 1011 B                     |              | 3 BMI/<br>5(6) LBMI | 6/15                | 2        | 2    | 2    | 4                   | ADDA                | 5                   | 2                  | 4                | ADDB             | 4+1              | 5              | B   |
| 1100 C                     | INC          | 3 BGE/<br>5(6) LBGE | 20                  | 2        | 2    | 2    | 4,6,6+1,7<br>CMPX / | 5,7,7+1,8<br>CMPY / | 5,7,7+1,8<br>CMPS / | 3                  | 5                | LDD              | 5+1              | 6              | C   |
| 1101 D                     | TST          | 3 BLT/<br>5(6) LBLT | 11                  | 2        | 2    | 2    | 7                   | 7+1                 | 8                   | 7                  | 5                | 5+1              | 6                | D              |     |
| 1110 E                     | JMP          | 3 BGT/<br>5(6) LBGT |                     | 2        | 2    | 2    | 3,5,5+1,6<br>LDX /  | 4,6,6+1,7<br>LDY /  | 4,6,6+1,7<br>LDS /  | 3,5,5+1,6<br>LDU / | 5,5+1,6<br>STU / | 5,5+1,6<br>STV / | 6,6+1,7<br>STS / | 6,6+1,7<br>STS | E   |
| 1111 F                     | CLR          | 3 BLE/<br>5(6) LBLE | 19/20/20<br>SWI/2/3 | 2        | 2    | 2    | 5,5+1,6<br>STX /    | 6,6+1,7<br>STY /    | 6,6+1,7<br>STS /    | 5,5+1,6<br>STU /   | 6,6+1,7<br>STS / | 6,6+1,7<br>STS   | 6,6+1,7<br>STS   | F              |     |



**Table F-2. Indexed Addressing Mode Data**

| Type  | Forms               | Non Indirect   |                  |        |        | Indirect          |                  |        |        |
|---|---------------------|----------------|------------------|--------|--------|-------------------|------------------|--------|--------|
|   |                     | Assembler Form | Postbyte OP Code | x<br>~ | +<br># | Assembler Form    | Postbyte OP Code | +<br>~ | +<br># |
| Constant Offset From R<br>(twos complement offset)    | No Offset           | ,R             | 1RR00100         | 0      | 0      | [R]               | 1RR10100         | 3      | 0      |
|   | 5 Bit Offset        | n, R           | ORRnnnnn         | 1      | 0      | defaults to 8-bit |                  |        |        |
|   | 8 Bit Offset        | n, R           | 1RR01000         | 1      | 1      | [n, R]            | 1RR11000         | 4      | 1      |
|   | 16 Bit Offset       | n, R           | 1RR01001         | 4      | 2      | [n, R]            | 1RR11001         | 7      | 2      |
| Accumulator Offset From R<br>(twos complement offset) | A — Register Offset | A, R           | 1RR00110         | 1      | 0      | [A, R]            | 1RR10110         | 4      | 0      |
|   | B — Register Offset | B, R           | 1RR00101         | 1      | 0      | [B, R]            | 1RR10101         | 4      | 0      |
|   | D — Register Offset | D, R           | 1RR01011         | 4      | 0      | [D, R]            | 1RR11011         | 7      | 0      |
| Auto Increment/Decrement R                            | Increment By 1      | ,R+            | 1RR00000         | 2      | 0      | not allowed       |                  |        |        |
|   | Increment By 2      | ,R++           | 1RR00001         | 3      | 0      | [R++]             | 1RR10001         | 6      | 0      |
|   | Decrement By 1      | ,-R            | 1RR00010         | 2      | 0      | not allowed       |                  |        |        |
|   | Decrement By 2      | ,--R           | 1RR00011         | 3      | 0      | [--R]             | 1RR10011         | 6      | 0      |
| Constant Offset From PC<br>(twos complement offset)   | 8 Bit Offset        | n, PCR         | 1XX01100         | 1      | 1      | [n, PCR]          | 1XX11100         | 4      | 1      |
|   | 16 Bit Offset       | n, PCR         | 1XX01101         | 5      | 2      | [n, PCR]          | 1XX11101         | 8      | 2      |
| Extended Indirect                                     | 16 Bit Address      | —              | —                | —      | —      | [n]               | 10011111         | 5      | 2      |

R = X, Y, U or S      X = 00      Y = 01  
 X = Don't Care      U = 10      S = 11

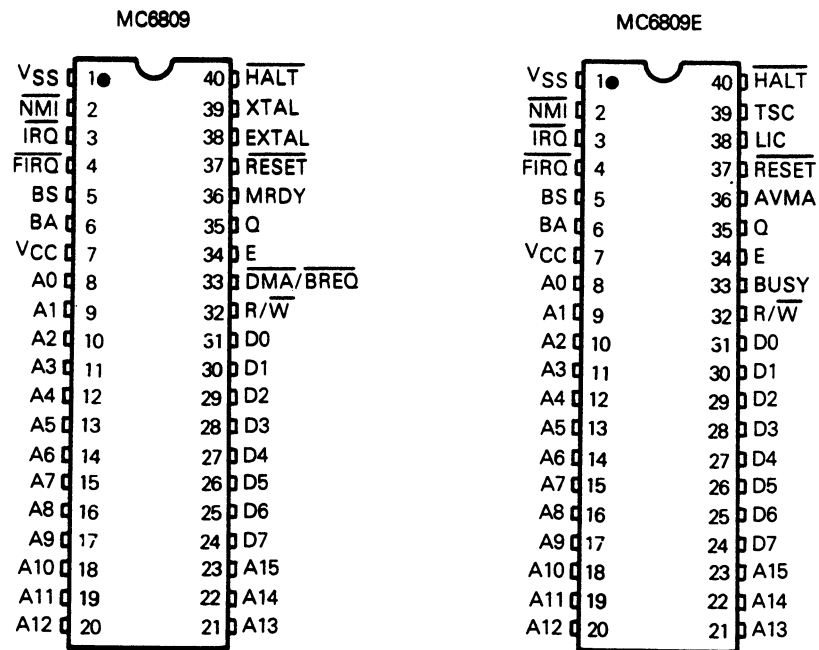
+ and + Indicate the number of additional cycles and bytes for the particular variation.  
 ~ #



## APPENDIX G PIN ASSIGNMENTS

### G.1 INTRODUCTION

This appendix is provided for a quick reference of the pin assignments for the MC6809 and MC6809E processors. Refer to Figure G-1. Descriptions of these pin assignments are given in Section 1.



**Figure G-1. Pin Assignments**



## APPENDIX H CONVERSION TABLES

### H.1 INTRODUCTION

This appendix provides some conversion tables for your convenience.

### H.2 POWERS OF 2, POWERS OF 16

Refer to Table H-1.

**Table H-1. Powers of 2; Powers of 16**

| $16^m$<br>m= | $2^n$<br>n= | Value  | $16^m$<br>m= | $2^n$<br>n= | Value         |
|--------------|-------------|--------|--------------|-------------|---------------|
| 0            | 0           | 1      | 4            | 16          | 65,536        |
| —            | 1           | 2      | —            | 17          | 131,072       |
| —            | 2           | 4      | —            | 18          | 262,144       |
| —            | 3           | 8      | —            | 19          | 524,288       |
| 1            | 4           | 16     | 5            | 20          | 1,048,576     |
| —            | 5           | 32     | —            | 21          | 2,097,152     |
| —            | 6           | 64     | —            | 22          | 4,194,304     |
| —            | 7           | 128    | —            | 23          | 8,388,608     |
| 2            | 8           | 256    | 6            | 24          | 16,777,216    |
| —            | 9           | 512    | —            | 25          | 33,554,432    |
| —            | 10          | 1,024  | —            | 26          | 67,108,864    |
| —            | 11          | 2,048  | —            | 27          | 134,217,728   |
| 3            | 12          | 4,096  | 7            | 28          | 268,435,456   |
| —            | 13          | 8,192  | —            | 29          | 536,870,912   |
| —            | 14          | 16,384 | —            | 30          | 1,073,741,824 |
| —            | 15          | 32,768 | —            | 31          | 2,147,483,648 |

### H.3 HEXADECIMAL AND DECIMAL CONVERSION

Table H-2 is a chart that can be used for converting numbers from either hexadecimal to decimal or decimal to hexadecimal.


**H.3.1 CONVERTING HEXADECIMAL TO DECIMAL.** Find the decimal weights for corresponding hexadecimal characters beginning with the least-significant character. The sum of the decimal weights is the decimal value of the hexadecimal number.

**H.3.2 CONVERTING DECIMAL TO HEXADECIMAL.** Find the highest decimal value in the table which is lower than or equal to the decimal number to be converted. The corresponding hexadecimal character is the most-significant digit of the final number. Subtract the decimal value found from the decimal number to be converted. Repeat the above step to determine the hexadecimal character. Repeat this process to find the subsequent hexadecimal numbers.

**Table H-2. Hexadecimal and Decimal Conversion Chart**

| 15  |        | Byte |       | 8    |     | 7   |      | Byte |     | 0    |     |
|-----|--------|------|-------|------|-----|-----|------|------|-----|------|-----|
| 15  | Char   | 12   | 11    | Char | 8   | 7   | Char | 4    | 3   | Char | 0   |
| Hex | Dec    | Hex  | Dec   | Hex  | Dec | Hex | Dec  | Hex  | Dec | Hex  | Dec |
| 0   | 0      | 0    | 0     | 0    | 0   | 0   | 0    | 0    | 0   | 0    | 0   |
| 1   | 4,096  | 1    | 256   | 1    | 16  | 1   | 16   | 1    | 1   | 1    | 1   |
| 2   | 8,192  | 2    | 512   | 2    | 32  | 2   | 32   | 2    | 2   | 2    | 2   |
| 3   | 12,288 | 3    | 768   | 3    | 48  | 3   | 48   | 3    | 3   | 3    | 3   |
| 4   | 16,384 | 4    | 1,024 | 4    | 64  | 4   | 64   | 4    | 4   | 4    | 4   |
| 5   | 20,480 | 5    | 1,280 | 5    | 80  | 5   | 80   | 5    | 5   | 5    | 5   |
| 6   | 24,576 | 6    | 1,536 | 6    | 96  | 6   | 96   | 6    | 6   | 6    | 6   |
| 7   | 28,672 | 7    | 1,792 | 7    | 112 | 7   | 112  | 7    | 7   | 7    | 7   |
| 8   | 32,768 | 8    | 2,048 | 8    | 128 | 8   | 128  | 8    | 8   | 8    | 8   |
| 9   | 36,864 | 9    | 2,304 | 9    | 144 | 9   | 144  | 9    | 9   | 9    | 9   |
| A   | 40,960 | A    | 2,560 | A    | 160 | A   | 160  | A    | A   | A    | 10  |
| B   | 45,056 | B    | 2,816 | B    | 176 | B   | 176  | B    | B   | B    | 11  |
| C   | 49,152 | C    | 3,072 | C    | 192 | C   | 192  | C    | C   | C    | 12  |
| D   | 53,248 | D    | 3,328 | D    | 208 | D   | 208  | D    | D   | D    | 13  |
| E   | 57,344 | E    | 3,584 | E    | 224 | E   | 224  | E    | E   | E    | 14  |
| F   | 61,440 | F    | 3,840 | F    | 240 | F   | 240  | F    | F   | F    | 15  |



Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

**Literature Distribution Centers:**

USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.

EUROPE: Motorola Ltd.; European Literature Centre; 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.

JAPAN: Nippon Motorola Ltd.; 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141, Japan.

ASIA PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No. 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.



**MOTOROLA**

PRINTED IN USA (1993) MPS/POD

M6809PM/AD

