M6809PM (AD)

MC6809-MC6809E
8-BIT MICROPROCESSOR
PROGRAMMING MANUAL

Original Issue: March 1, 1981

TABLE OF CONTENTS

Paragraph No. *Title* *Page No.*

SECTION 1
GENERAL DESCRIPTION

TABLE OF CONTENTS
(CONTINUED)

SECTION 2
ADDRESSING MODES

SECTION 3
INTERRUPT CAPABILITIES

TABLE OF CONTENTS
(CONCLUDED)

Paragraph No. *Title* *Page No.*

SECTION 4
PROGRAMMING

APPENDIX A
INSTRUCTION SET DETAILS

APPENDIX B
ASSIST09 MONITOR PROGRAM

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS
(CONTINUED)

APPENDIX C
MACHINE CODE TO INSTRUCTION CROSS REFERENCE

APPENDIX D
PROGRAMMING AID

APPENDIX E
ASCII CHARACTER SET

TABLE OF CONTENTS (CONTINUED)

LIST OF ILLUSTRATIONS

LIST OF TABLES

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.

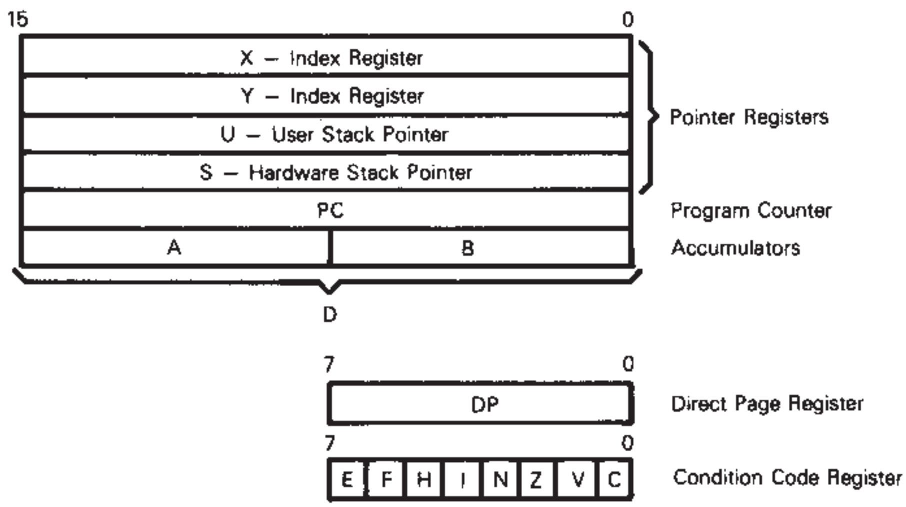


Figure 1-1. Programming Model

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 compatiblity.

1.10 CONDITION CODE REGISTER (CC)

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

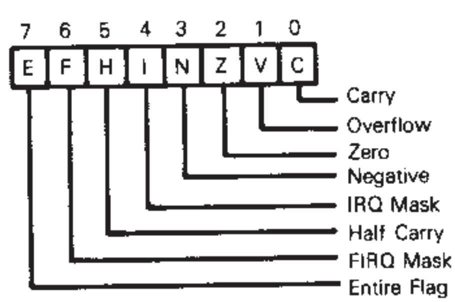


Figure 1-2. Condition Code Register

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.

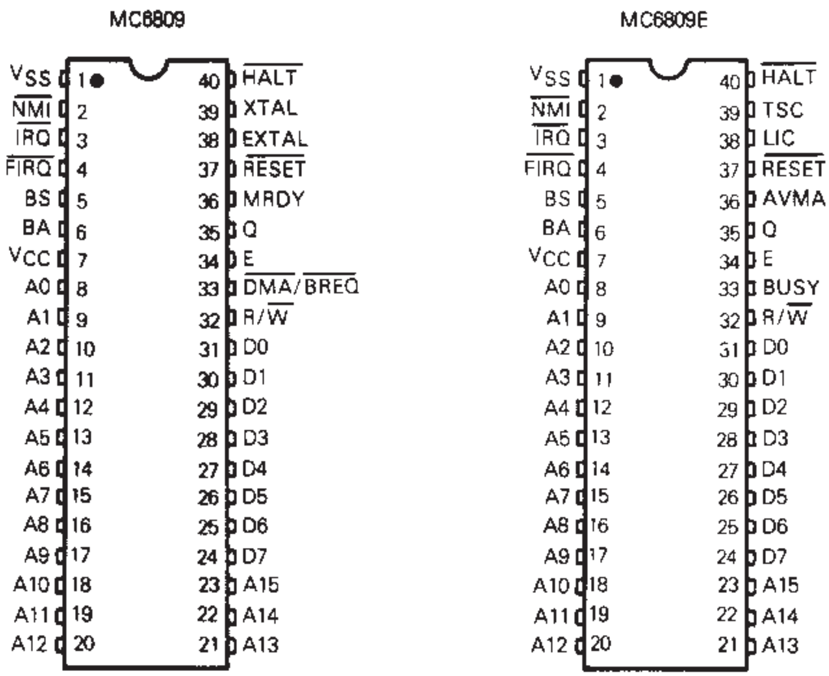


Figure 1-3. Processor Pin Assignments

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 (ϕ2) 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/$\overline{W}$ 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 $FFFF_{16}$, and read/write (R/$\overline{W}$) 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 ($R/\overline{W}$). 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 $R/\overline{W}$ output is valid at the leading edge of the Q clock. The $R/\overline{W}$ 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

BA	BS	Processor State
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 acknolwedge 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 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 ($\overline{\text{DMA}}/\overline{\text{BREQ}}$) (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 $\overline{\text{DMA}}/\overline{\text{BREQ}}$ 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 indivisable 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 [illegible]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 (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 (twos Complement Offset)	A Accumulator Offset	1RR00110	1RR10110
	B Accumulator Offset	1RR00101	1RR10101
	D Accumulator Offset	1RR01011	1RR11011
Auto Increment/Decrement from 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 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:

Autoincrement		Autodecrement	
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

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 ($\overline{NMI}$), the fast maskable interrupt request ($\overline{FIRQ}$), and the normal maskable interrupt request ($\overline{IRQ}$). 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 $\overline{FIRQ}$ 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, $\overline{NMI}$, SWI, $\overline{FIRQ}$, $\overline{IRQ}$, 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, $\overline{NMI}$, $\overline{FIRQ}$, and $\overline{IRQ}$ are listed alphabetically at the end of Appendix A.

Table 3-1. Interrupt Vector Locations

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

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

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 (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 $\overline{\text{IRQ}}$ 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.

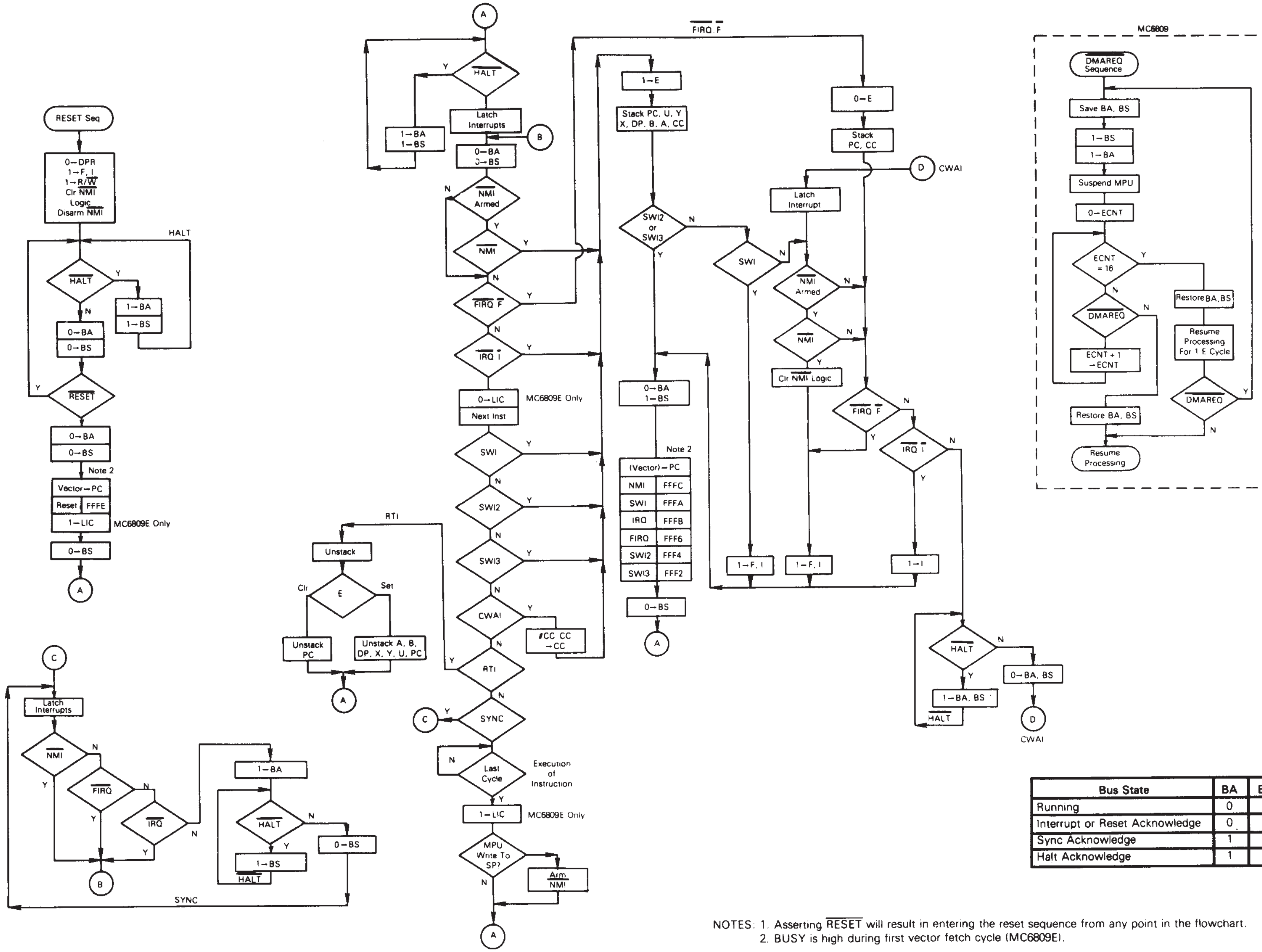
After accepting a normal interrupt request, the processor sets the E flag, saves the entire machine state, and then sets the I bit to mask any further interrupt request inputs. After servicing the original interrupt, the user may clear the I bit to allow multiple-level normal interrupts.

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 $\overline{\text{IRQ}}$).



Bus State	BA	BS
Running	0	0
Interrupt or Reset Acknowledge	0	1
Sync Acknowledge	1	0
Halt Acknowledge	1	1

NOTES: 1. Asserting $\overline{\text{RESET}}$ will result in entering the reset sequence from any point in the flowchart.
2. BUSY is high during first vector fetch cycle (MC6809E).

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 addresing 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.

4.2.2 POSITION-INDEPENDENT CODE. Position-independent code means that the same machine language code can be placed anywhere in memory and still function correctly. The M6809 has a long relative (16-bit offset) branch mode along with the common MC6800 branches, plus program-counter relative addressing. Program-counter relative addressing uses the program counter like an indexable register, which allows all instructions that reference memory to also reference data relative to the program counter. The M6809 also has load effective address (LEA) instructions which allow the user to point to data in a ROM in a position-independent manner.

An important rule for generating position-independent code is: NEVER USE ABSOLUTE ADDRESSING.

Program-counter relative addressing on the M6809 is a form of indexed addressing that uses the program counter as the base register for a constant-offset indexing operation. However, the M6809 assembler treats the PCR address field differently from that used in other indexed instructions. In PCR addressing, the assembly time location value is subtracted from the (constant) value of the PCR offset. The resulting distance to the desired symbol is the value placed into the machine language object code. During execution, the processor adds the value of the run time PC to the distance to get a position-independent absolute address.

The PCR indexed addressing form can be used to point at any location relative to the program regardless of position in memory. The PCR form of indexed addressing allows access to tables within the program space in a position-independent manner via use of the load effective address instruction.

In a program which is completely position-independent, some absolute locations are usually required, particularly for I/O. If the locations of I/O devices are placed on the stack (as globals) by a small setup routine before the standard package is invoked, all internal modules can do their I/O through that pointer (e.g., STA [ACIAD, U]), allowing the hardware to be easily changed, if desired. Only the single, small, and obvious setup routine need be rewritten for each different hardware configuration.

Global, permanent, and temporary values need to be easily available in a position-independent manner. Use the stack for this data since the stacked data is directly accessible. Stack the absolute address of I/O devices before calling any standard software package since the package can use the stacked addresses for I/O in any system.

The LEA instructions allow access to tables, data, or immediate values in the text of the program in a position-independent manner as shown in the following example:

```
                              .
                              .
                              .
                LEAX          MSG1,PCR
                LBSR          PDATA
                              .
                              .
                              .
MSG1            FCC           /PRINT THIS!/
```

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.

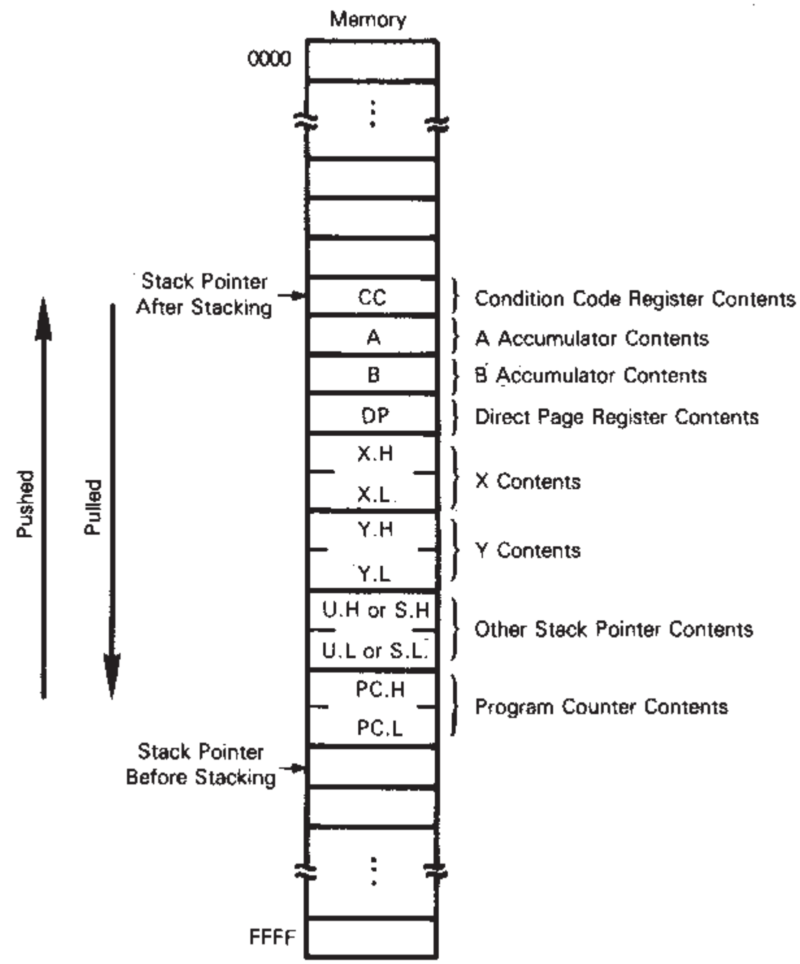


Figure 4-1. Stacking Order

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 temorary (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

Instruction	Description
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)

Instruction	Description
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 × B → 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 PC 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
	SIMPLE BRANCHES
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
	SIGNED BRANCHES
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)
	UNSIGNED BRANCHES
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)
	OTHER BRANCHES
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

Symbol	Meaning
←	Is transferred to
Λ	Boolean AND
V	Boolean OR
⊕	Boolean exclusive OR
‾ (Overline)	Boolean NOT
:	Concatenation
+	Arithmetic plus
−	Arithmetic minus
X	Arithmetic multiply

Table A-2. Register Notation

Abbreviation	Meaning
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 signifcant 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: $R' \leftarrow R \wedge M$

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 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:

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 $\leftarrow$ MI
IFF $C = 1$ then $PC' \leftarrow PC + TEMP$

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

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 ← MI
IFF Z $\wedge$ [N $\oplus$ V] = 0 then PC′ ← 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 Test

BIT

Source Form: Bit P

Operation: TEMP ← R Λ 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 v [N ⊕ 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 ← MI
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 twos 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 $\leftarrow$ MI PC' $\leftarrow$ PC + TEMP
Condition Codes:	Not affected.
Description:	Causes an unconditional branch.
Addressing Mode:	Relative

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

Branch to Subroutine

BSR

Source Forms: BSR dd; LBSR DDDD

Operation:

TEMP ← MI
SP' ← SP − 1, (SP) ← PCL
SP' ← SP − 1, (SP) ← PCH
PC' ← 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 twos complement result was valid. When used after an operation on twos 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 twos complement result was invalid. When used after an operation on twos complement binary values, this instruction will branch if there was an overflow.

Addressing Mode: Relative

Source Form:	CLR Q
Operation:	TEMP ← M M ← 00_{16}
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 ← 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: $M' \leftarrow 0 + \overline{M}$

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 $\wedge$ 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

Source Form: DAA

Operation: ACCA′ ← 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

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

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′ ← EA

Condition Codes: Not affected.

Description: Program control is transferred to the effective address.

Addressing Modes: Extended
Direct
Indexed

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 otheriwse.
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' ← 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: C ← [b7 | | | | | | | b0] ← 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 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

Source Form: MUL

Operation: ACCA':ACCB' ← ACCA x 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′ ← 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′←R v 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:

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

push order----→

Operation:

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

Source Form: PSHU register list
PSHU #LABEL
Postbyte:

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

push order----->

Operation:

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

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′ ←(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++).

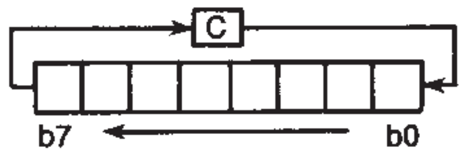
ROL

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

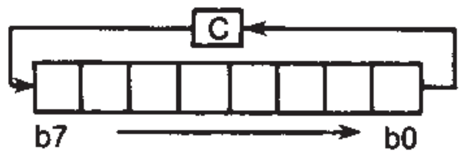
ROR

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

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

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′ ← FF_{16}
else ACCA′ ← 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 twos complement 8-bit value in accumulator B into a twos 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' ← 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′ ← 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′ ← (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 excuted, 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

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.

$\overline{\text{FIRQ}}$ Fast Interrupt Request (Hardware Interrupt) $\overline{\text{FIRQ}}$

Operation: IFF F bit clear, then: SP′←SP – 1, (SP)←PCL
SP′←SP – 1, (SP)←PCH
Clear E (subset state is saved)
SP′←SP – 1, (SP)←CCR
Set F, I (mask further interrupts)
PC′←(FFF6):(FFF7)

Condition Codes: Not affected.

Description: A $\overline{\text{FIRQ}}$ (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

Operation: IFF I bit clear, then:

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

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

Operation:

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
Set E (entire state save)
SP′ ← SP − 1, (SP) ← CCR
Set I, F (mask interrupts)
PC′ ← (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 $\overline{NMI}$ 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: $CCR' \leftarrow X1X1XXXX$
$DPR' \leftarrow 00_{16}$
$PC' \leftarrow (FFFE):(FFFF)$

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.

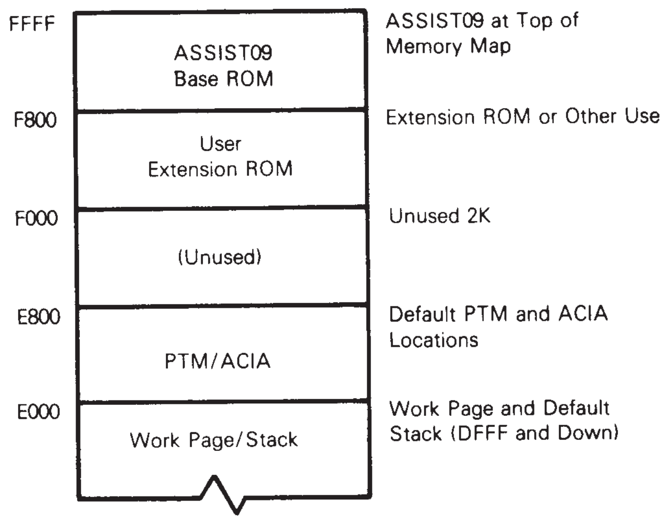


Figure B-1. Memory Map

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 intitialization 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 comand 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.

MIKBUG is a trademark of Motorola Inc.

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 supress 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 supressed 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
```

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.

Example:

```
MONITR  EQU  8               INPUT CODE FOR MONITR

LOOP    CLRA                 PREPARE ZERO PAGE REGISTER AND
*                            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 $\overline{\text{RESET}}$. 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 intiialization 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:

Input: A = Character to send
DP = ASSIST09 work page
.PAD = Character and new line padding counts (in vector table)
.PAUSE = Pause routine (in vector table)

Output: Character sent to the output device

Volatile Registers: 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 overlayed 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 redispatch 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   .SWI

Softare Interrupt Vector Appendage

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 serivce 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   .SWI2

Software Interrupt 2 Vector Appendage

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 interurpt which has its interrupt vector address located at hexadecimal FFF2. The default handler prints the registers and enters the ASSIST09 command handler.

PLEASE NOTE:

I did not scan this ASSIST09 listing from the Motorola book. The listing was very large, and I was scanning a bound book which required each page to be scanned individually. Instead, I assembled the ASSIST09 source code using my own ASM09 assembler and placed the resultant listing into these pages. This has the added advantage that the text is searchable and can be extracted if you wish to use code snippets in other places. It does mean however that this listing is not precicely identical to the one originally printed in the Motorola MC6809-MC6809E Programming Reference manual.

Also note: I found the source code on which this listing is based via an internet search. It appears to be the original Motorola source code, however it had been modified both in function (code changes) and source format (different assembler). I have attempted to restore it as closely as possible to the original - please notify me if you find errors. For the most part, the source was compatible with my assembler. I did have to change the single-quote character constants to double-quote format ('a' instead of 'a), and some of the directives are slightly different. (TITLE instead of TTL for example).

Dave Dunfield

```
0000      *************************************
0000      * COPYRIGHT (C) MOTOROLA, INC. 1979 *
0000      *************************************
0000
0000      *************************************
0000      *  THIS IS THE BASE ASSIST09 ROM.
0000      *  IT MAY RUN WITH OR WITHOUT THE
0000      *  EXTENSION ROM WHICH
0000      *  WHEN PRESENT WILL BE AUTOMATICALLY
0000      *  INCORPORATED BY THE BLDVTR
0000      *  SUBROUTINE.
0000      *************************************
0000
0000      *********************************************
0000      *       GLOBAL MODULE EQUATES
0000      ********************************************
F800      ROMBEG  EQU     $F800           ROM START ASSEMBLY ADDRESS
E700      RAMOFS  EQU     -$1900          ROM OFFSET TO RAM WORK PAGE
0800      ROMSIZ  EQU     2048            ROM SIZE
F000      ROM2OF  EQU     ROMBEG-ROMSIZ   START OF EXTENSION ROM
E008      ACIA    EQU     $E008           DEFAULT ACIA ADDRESS
E000      PTM     EQU     $E000           DEFAULT PTM ADDRESS
0000      DFTCHP  EQU     0               DEFAULT CHARACTER PAD COUNT
0005      DFTNLP  EQU     5               DEFAULT NEW LINE PAD COUNT
003E      PROMPT  EQU     '>'             PROMPT CHARACTER
0008      NUMBKP  EQU     8               NUMBER OF BREAKPOINTS
0000      *********************************************
0000
0000      *********************************************
0000      *  MISCELANEOUS EQUATES
0000      *********************************************
0004      EOT     EQU     $04             END OF TRANSMISSION
0007      BELL    EQU     $07             BELL CHARACTER
000A      LF      EQU     $0A             LINE FEED
000D      CR      EQU     $0D             CARRIAGE RETURN
0010      DLE     EQU     $10             DATA LINK ESCAPE
0018      CAN     EQU     $18             CANCEL (CTL-X)
0000      * PTM ACCESS DEFINITIONS
E001      PTMSTA  EQU     PTM+1           READ STATUS REGISTER
E000      PTMC13  EQU     PTM             CONTROL REGISTERS 1 AND 3
E001      PTMC2   EQU     PTM+1           CONTROL REGISTER 2
E002      PTMTM1  EQU     PTM+2           LATCH 1
E004      PTMTM2  EQU     PTM+4           LATCH 2
E006      PTMTM3  EQU     PTM+6           LATCH 3
0000
008C      SKIP2   EQU     $8C             "CMPX #" OPCODE - SKIPS TWO BYTES
0000
0000      *******************************************
0000      *    ASSIST09 MONITOR SWI FUNCTIONS
0000      * THE FOLLOWING EQUATES DEFINE FUNCTIONS PROVIDED
0000      * BY THE ASSIST09 MONITOR VIA THE SWI INSTRUCTION.
0000      ******************************************
0000      INCHNP  EQU     0               INPUT CHAR IN A REG - NO PARITY
0001      OUTCH   EQU     1               OUTPUT CHAR FROM A REG
0002      PDATA1  EQU     2               OUTPUT STRING
0003      PDATA   EQU     3               OUTPUT CR/LF THEN STRING
0004      OUT2HS  EQU     4               OUTPUT TWO HEX AND SPACE
0005      OUT4HS  EQU     5               OUTPUT FOUR HEX AND SPACE
0006      PCRLF   EQU     6               OUTPUT CR/LF
0007      SPACE   EQU     7               OUTPUT A SPACE
0008      MONITR  EQU     8               ENTER ASSIST09 MONITOR
0009      VCTRSW  EQU     9               VECTOR EXAMINE/SWITCH
000A      BRKPT   EQU     10              USER PROGRAM BREAKPOINT
000B      PAUSE   EQU     11              TASK PAUSE FUNCTION
000B      NUMFUN  EQU     11              NUMBER OF AVAILABLE FUNCTIONS
0000      * NEXT SUB-CODES FOR ACCESSING THE VECTOR TABLE.
0000      * THEY ARE EQUIVALENT TO OFFSETS IN THE TABLE.
0000      * RELATIVE POSITIONING MUST BE MAINTAINED.
```

```
0000      .AVTBL  EQU    0         ADDRESS OF VECTOR TABLE
0002      .CMDL1  EQU    2         FIRST COMMAND LIST
0004      .RSVD   EQU    4         RESERVED HARDWARE VECTOR
0006      .SWI3   EQU    6         SWI3 ROUTINE
0008      .SWI2   EQU    8         SWI2 ROUTINE
000A      .FIRQ   EQU    10        FIRQ ROUTINE
000C      .IRQ    EQU    12        IRQ ROUTINE
000E      .SWI    EQU    14        SWI ROUTINE
0010      .NMI    EQU    16        NMI ROUTINE
0012      .RESET  EQU    18        RESET ROUTINE
0014      .CION   EQU    20        CONSOLE ON
0016      .CIDTA  EQU    22        CONSOLE INPUT DATA
0018      .CIOFF  EQU    24        CONSOLE INPUT OFF
001A      .COON   EQU    26        CONSOLE OUTPUT ON
001C      .CODTA  EQU    28        CONSOLE OUTPUT DATA
001E      .COOFF  EQU    30        CONSOLE OUTPUT OFF
0020      .HSDTA  EQU    32        HIGH SPEED PRINTDATA
0022      .BSON   EQU    34        PUNCH/LOAD ON
0024      .BSDTA  EQU    36        PUNCH/LOAD DATA
0026      .BSOFF  EQU    38        PUNCH/LOAD OFF
0028      .PAUSE  EQU    40        TASK PAUSE ROUTINE
002A      .EXPAN  EQU    42        EXPRESSION ANALYZER
002C      .CMDL2  EQU    44        SECOND COMMAND LIST
002E      .ACIA   EQU    46        ACIA ADDRESS
0030      .PAD    EQU    48        CHARACTER PAD AND NEW LINE PAD
0032      .ECHO   EQU    50        ECHO/LOAD AND NULL BKPT FLAG
0034      .PTM    EQU    52        PTM ADDRESS
001B      NUMVTR  EQU    52/2+1    NUMBER OF VECTORS
0034      HIVTR   EQU    52        HIGHEST VECTOR OFFSET
```

```
0000      *****************************************
0000      *              WORK AREA
0000      * THIS WORK AREA IS ASSIGNED TO THE PAGE ADDRESSED BY
0000      * -$1800,PCR FROM THE BASE ADDRESS OF THE ASSIST09
0000      * ROM.  THE DIRECT PAGE REGISTER DURING MOST ROUTINE
0000      * OPERATIONS WILL POINT TO THIS WORK AREA.  THE STACK
0000      * INITIALLY STARTS UNDER THE RESERVED WORK AREAS AS
0000      * DEFINED HEREIN.
0000      *****************************************
DF00      WORKPG  EQU    ROMBEG+RAMOFS   SETUP DIRECT PAGE ADDRESS
0000              SETDP  =WORKPG         NOTIFY ASSEMBLER
E000              ORG    WORKPG+256      READY PAGE DEFINITIONS
E000      * THE FOLLOWING THRU BKPTOP MUST RESIDE IN THIS ORDER
E000      * FOR PROPER INITIALIZATION
DFFC              ORG    *-4
DFFC      PAUSER  EQU    *               PAUSE ROUTINE
DFFB              ORG    *-1
DFFB      SWIBFL  EQU    *               BYPASS SWI AS BREAKPOINT FLAG
DFFA              ORG    *-1
DFFA      BKPTCT  EQU    *               BREAKPOINT COUNT
DFF8              ORG    *-2
DFF8      SLEVEL  EQU    *               STACK TRACE LEVEL
DFC2              ORG    *-(NUMVTR*2)
DFC2      VECTAB  EQU    *               VECTOR TABLE
DFB2              ORG    *-(2*NUMBKP)
DFB2      BKPTBL  EQU    *               BREAKPOINT TABLE
DFA2              ORG    *-(2*NUMBKP)
DFA2      BKPTOP  EQU    *               BREAKPOINT OPCODE TABLE
DFA0              ORG    *-2
DFA0      WINDOW  EQU    *               WINDOW
DF9E              ORG    *-2
DF9E      ADDR    EQU    *               ADDRESS POINTER VALUE
DF9D              ORG    *-1
DF9D      BASEPG  EQU    *               BASE PAGE VALUE
DF9B              ORG    *-2
DF9B      NUMBER  EQU    *               BINARY BUILD AREA
DF99              ORG    *-2
DF99      LASTOP  EQU    *               LAST OPCODE TRACED
DF97              ORG    *-2
DF97      RSTACK  EQU    *               RESET STACK POINTER
DF95              ORG    *-2
DF95      PSTACK  EQU    *               COMMAND RECOVERY STACK
DF93              ORG    *-2
DF93      PCNTER  EQU    *               LAST PROGRAM COUNTER
DF91              ORG    *-2
DF91      TRACEC  EQU    *               TRACE COUNT
DF90              ORG    *-1
DF90      SWICNT  EQU    *               TRACE "SWI" NEST LEVEL COUNT
DF8F              ORG    *-1             (MISFLG MUST FOLLOW SWICNT)
DF8F      MISFLG  EQU    *               LOAD CMD/THRU BREAKPOINT FLAG
DF8E              ORG    *-1
DF8E      DELIM   EQU    *               EXPRESSION DELIMITER/WORK BYTE
DF66              ORG    *-40
DF66      ROM2WK  EQU    *               EXTENSION ROM RESERVED AREA
DF51              ORG    *-21
DF51      TSTACK  EQU    *               TEMPORARY STACK HOLD
DF51      STACK   EQU    *               START OF INITIAL STACK
```

```
DF51                     ******************************************
DF51                     * DEFAULT THE ROM BEGINNING ADDRESS TO 'ROMBEG'
DF51                     * ASSIST09 IS POSITION ADDRESS INDEPENDENT, HOWEVER
DF51                     * WE ASSEMBLE ASSUMING CONTROL OF THE HARDWARE VECTORS.
DF51                     * NOTE THAT THE WORK RAM PAGE MUST BE 'RAMOFS'
DF51                     * FROM THE ROM BEGINNING ADDRESS.
DF51                     ********************************************
F800                             ORG     ROMBEG          ROM ASSEMBLY/DEFAULT ADDRESS
F800
F800                     *****************************************************
F800                     *            BLDVTR - BUILD ASSIST09 VECTOR TABLE
F800                     *  HARDWARE RESET CALLS THIS SUBROUTINE TO BUILD THE
F800                     *  ASSIST09 VECTOR TABLE.  THIS SUBROUTINE RESIDES AT
F800                     *  THE FIRST BYTE OF THE ASSIST09 ROM, AND CAN BE
F800                     *  CALLED VIA EXTERNAL CONTROL CODE FOR REMOTE
F800                     *  ASSIST09 EXECUTION.
F800                     * INPUT: S->VALID STACK RAM
F800                     * OUTPUT: U->VECTOR TABLE ADDRESS
F800                     *         DPR->ASSIST09 WORK AREA PAGE
F800                     *         THE VECTOR TABLE AND DEFAULTS ARE INITIALIZED
F800                     *  ALL REGISTERS VOLATILE
F800                     *************************************************
F800
F800  30 8D E7 BE        BLDVTR  LEAX    VECTAB,PCR      ADDRESS VECTOR TABLE
F804  1F 10                      TFR     X,D             OBTAIN BASE PAGE ADDRESS
F806  1F 8B                      TFR     A,DP            SETUP DPR
F808  97 9D                      STA     BASEPG          STORE FOR QUICK REFERENCE
F80A  33 84                      LEAU    ,X              RETURN TABLE TO CALLER
F80C  31 8C 35                   LEAY    <INITVT,PCR     LOAD FROM ADDR
F80F  EF 81                      STU     ,X++            INIT VECTOR TABLE ADDRESS
F811  C6 16                      LDB     #NUMVTR-5       NUMBER RELOCATABLE VECTORS
F813  34 04                      PSHS    B               STORE INDEX ON STACK
F815  1F 20              BLD2    TFR     Y,D             PREPARE ADDRESS RESOLVE
F817  E3 A1                      ADDD    ,Y++            TO ABSOLUTE ADDRESS
F819  ED 81                      STD     ,X++            INTO VECTOR TABLE
F81B  6A E4                      DEC     ,S              COUNT DOWN
F81D  26 F6                      BNE     BLD2            BRANCH IF MORE TO INSERT
F81F  C6 0D                      LDB     #INTVE-INTVS    STATIC VALUE INIT LENGTH
F821  A6 A0              BLD3    LDA     ,Y+             LOAD NEXT BYTE
F823  A7 80                      STA     ,X+             STORE INTO POSITION
F825  5A                         DECB                    COUNT DOWN
F826  26 F9                      BNE     BLD3            LOOP UNTIL DONE
F828  31 8D F7 D4                LEAY    ROM2OF,PCR      TEST POSSIBLE EXTENSION ROM
F82C  8E 20 FE                   LDX     #$20FE          LOAD "BRA *" FLAG PATTERN
F82F  AC A1                      CMPX    ,Y++            ? EXTENDED ROM HERE
F831  26 02                      BNE     BLDRTN          BRANCH NOT OUR ROM TO RETURN
F833  AD A4                      JSR     ,Y              CALL EXTENDED ROM INITIALIZE
F835  35 84              BLDRTN  PULS    PC,B            RETURN TO INITIALIZER
F837
F837                     *****************************************************
F837                     *                 RESET ENTRY POINT
F837                     *  HARDWARE RESET ENTERS HERE IF ASSIST09 IS ENABLED
F837                     *  TO RECEIVE THE MC6809 HARDWARE VECTORS.  WE CALL
F837                     *  THE BLDVTR SUBROUTINE TO INITIALIZE THE VECTOR
F837                     *  TABLE, STACK, AND THEN FIREUP THE MONITOR VIA SWI
F837                     *  CALL.
F837                     *******************************************************
F837  32 8D E7 16        RESET   LEAS    STACK,PCR       SETUP INITIAL STACK
F83B  8D C3                      BSR     BLDVTR          BUILD VECTOR TABLE
F83D  4F                 RESET2  CLRA                    ISSUE STARTUP MESSAGE
F83E  1F 8B                      TFR     A,DP            DEFAULT TO PAGE ZERO
F840  3F                         SWI                     PERFORM MONITOR FIREUP
F841  08                         FCB     MONITR          TO ENTER COMMAND PROCESSING
F842  20 F9                      BRA     RESET2          REENTER MONITOR IF 'CONTINUE'
F844
F844                     *****************************************************
F844                     *          INITVT - INITIAL VECTOR TABLE
F844                     *  THIS TABLE IS RELOCATED TO RAM AND REPRESENTS THE
```

```
F844                *  INITIAL STATE OF THE VECTOR TABLE. ALL ADDRESSES
F844                *  ARE CONVERTED TO ABSOLUTE FORM.  THIS TABLE STARTS
F844                *  WITH THE SECOND ENTRY, ENDS WITH STATIC CONSTANT
F844                *  INITIALIZATION DATA WHICH CARRIES BEYOND THE TABLE.
F844                ************************************************
F844  01 58         INITVT  FDB     CMDTBL-*        DEFAULT FIRST COMMAND TABLE
F846  02 92                 FDB     RSRVDR-*        DEFAULT UNDEFINED HARDWARE VECTOR
F848  02 90                 FDB     SWI3R-*         DEFAULT SWI3
F84A  02 8E                 FDB     SWI2R-*         DEFAULT SWI2
F84C  02 70                 FDB     FIRQR-*         DEFAULT FIRQ
F84E  02 8A                 FDB     IRQR-*          DEFAULT IRQ ROUTINE
F850  00 45                 FDB     SWIR-*          DEFAULT SWI ROUTINE
F852  02 2B                 FDB     NMIR-*          DEFAULT NMI ROUTINE
F854  FF E3                 FDB     RESET-*         RESTART VECTOR
F856  02 90                 FDB     CION-*          DEFAULT CION
F858  02 84                 FDB     CIDTA-*         DEFAULT CIDTA
F85A  02 96                 FDB     CIOFF-*         DEFAULT CIOFF
F85C  02 8A                 FDB     COON-*          DEFAULT COON
F85E  02 93                 FDB     CODTA-*         DEFAULT CODTA
F860  02 90                 FDB     COOFF-*         DEFAULT COOFF
F862  03 9A                 FDB     HSDTA-*         DEFAULT HSDTA
F864  02 B7                 FDB     BSON-*          DEFAULT BSON
F866  02 D2                 FDB     BSDTA-*         DEFAULT BSDTA
F868  02 BF                 FDB     BSOFF-*         DEFAULT BSOFF
F86A  E7 92                 FDB     PAUSER-*        DEFAULT PAUSE ROUTINE
F86C  04 7D                 FDB     EXP1-*          DEFAULT EXPRESSION ANALYZER
F86E  01 2D                 FDB     CMDTB2-*        DEFAULT SECOND COMMAND TABLE
F870                * CONSTANTS
F870  E0 08         INTVS   FDB     ACIA            DEFAULT ACIA
F872  00 05                 FCB     DFTCHP,DFTNLP   DEFAULT NULL PADDS
F874  00 00                 FDB     0               DEFAULT ECHO
F876  E0 00                 FDB     PTM             DEFAULT PTM
F878  00 00                 FDB     0               INITIAL STACK TRACE LEVEL
F87A  00                    FCB     0               INITIAL BREAKPOINT COUNT
F87B  00                    FCB     0               SWI BREAKPOINT LEVEL
F87C  39                    FCB     $39             DEFAULT PAUSE ROUTINE (RTS)
F87D                INTVE   EQU     *
F87D                *B
F87D
F87D                ***********************************************
F87D                *             ASSIST09 SWI HANDLER
F87D                *  THE SWI HANDLER PROVIDES ALL INTERFACING NECESSARY
F87D                *  FOR A USER PROGRAM.  A FUNCTION BYTE IS ASSUMED TO
F87D                *  FOLLOW THE SWI INSTRUCTION.  IT IS BOUND CHECKED
F87D                *  AND THE PROPER ROUTINE IS GIVEN CONTROL.  THIS
F87D                *  INVOCATION MAY ALSO BE A BREAKPOINT INTERRUPT.
F87D                *  IF SO, THE BREAKPOINT HANDLER IS ENTERED.
F87D                * INPUT: MACHINE STATE DEFINED FOR SWI
F87D                * OUTPUT: VARIES ACCORDING TO FUNCTION CALLED. PC ON
F87D                *     CALLERS STACK INCREMENTED BY ONE IF VALID CALL.
F87D                * VOLATILE REGISTERS: SEE FUNCTIONS CALLED
F87D                * STATE: RUNS DISABLED UNLESS FUNCTION CLEARS I FLAG.
F87D                ************************************************
F87D
F87D                * SWI FUNCTION VECTOR TABLE
F87D  01 94         SWIVTB  FDB     ZINCH-SWIVTB    INCHNP
F87F  01 B1                 FDB     ZOTCH1-SWIVTB   OUTCH
F881  01 CB                 FDB     ZPDTA1-SWIVTB   PDATA1
F883  01 C3                 FDB     ZPDATA-SWIVTB   PDATA
F885  01 75                 FDB     ZOT2HS-SWIVTB   OUT2HS
F887  01 73                 FDB     ZOT4HS-SWIVTB   OUT4HS
F889  01 C0                 FDB     ZPCRLF-SWIVTB   PCRLF
F88B  01 79                 FDB     ZSPACE-SWIVTB   SPACE
F88D  00 55                 FDB     ZMONTR-SWIVTB   MONITR
F88F  01 7D                 FDB     ZVSWTH-SWIVTB   VCTRSW
F891  02 56                 FDB     ZBKPNT-SWIVTB   BREAKPOINT
F893  01 D1                 FDB     ZPAUSE-SWIVTB   TASK PAUSE
F895
```

```
F895  6A 8D E6 F7          SWIR    DEC     SWICNT,PCR      UP "SWI" LEVEL FOR TRACE
F899  17 02 25                     LBSR    LDDP            SETUP PAGE AND VERIFY STACK
F89C                       * CHECK FOR BREAKPOINT TRAP
F89C  EE 6A                        LDU     10,S            LOAD PROGRAM COUNTER
F89E  33 5F                        LEAU    -1,U            BACK TO SWI ADDRESS
F8A0  0D FB                        TST     SWIBFL          ? THIS "SWI" BREAKPOINT
F8A2  26 11                        BNE     SWIDNE          BRANCH IF SO TO LET THROUGH
F8A4  17 06 9B                     LBSR    CBKLDR          OBTAIN BREAKPOINT POINTERS
F8A7  50                           NEGB                    OBTAIN POSITIVE COUNT
F8A8  5A                   SWILP   DECB                    COUNT DOWN
F8A9  2B 0A                        BMI     SWIDNE          BRANCH WHEN DONE
F8AB  11 A3 A1                     CMPU    ,Y++            ? WAS THIS A BREAKPOINT
F8AE  26 F8                        BNE     SWILP           BRANCH IF NOT
F8B0  EF 6A                        STU     10,S            SET PROGRAM COUNTER BACK
F8B2  16 02 1E                     LBRA    ZBKPNT          GO DO BREAKPOINT
F8B5  0F FB                SWIDNE  CLR     SWIBFL          CLEAR IN CASE SET
F8B7  37 06                        PULU    D               OBTAIN FUNCTION BYTE, UP PC
F8B9  C1 0B                        CMPB    #NUMFUN         ? TOO HIGH
F8BB  10 22 02 0F                  LBHI    ERROR           YES, DO BREAKPOINT
F8BF  EF 6A                        STU     10,S            BUMP PROGRAM COUNTER PAST SWI
F8C1  58                           ASLB    FUNCTION        CODE TIMES TWO
F8C2  33 8C B8                     LEAU    SWIVTB,PCR      OBTAIN VECTOR BRANCH ADDRESS
F8C5  EC C5                        LDD     B,U             LOAD OFFSET
F8C7  6E CB                        JMP     D,U             JUMP TO ROUTINE
F8C9
F8C9                       **********************************************
F8C9                       * REGISTERS TO FUNCTION ROUTINES:
F8C9                       *  DP-> WORK AREA PAGE
F8C9                       *  D,Y,U=UNRELIABLE            X=AS CALLED FROM USER
F8C9                       *  S=AS FROM SWI INTERRUPT
F8C9                       *********************************************
F8C9
F8C9                       ************************************************
F8C9                       *            [SWI FUNCTION 8]
F8C9                       *              MONITOR ENTRY
F8C9                       *  FIREUP THE ASSIST09 MONITOR.
F8C9                       *  THE STACK WITH ITS VALUES FOR THE DIRECT PAGE
F8C9                       *  REGISTER AND CONDITION CODE FLAGS ARE USED AS IS.
F8C9                       *    1) INITIALIZE CONSOLE I/O
F8C9                       *    2) OPTIONALLY PRINT SIGNON
F8C9                       *    3) INITIALIZE PTM FOR SINGLE STEPPING
F8C9                       *    4) ENTER COMMAND PROCESSOR
F8C9                       * INPUT: A=0 INIT CONSOLE AND PRINT STARTUP MESSAGE
F8C9                       *        A#0 OMIT CONSOLE INIT AND STARTUP MESSAGE
F8C9                       ***********************************************
F8C9
F8C9  41 53 53 49 53 54 +  SIGNON  FCC     /ASSIST09/      SIGNON EYE-CATCHER
F8D1  04                           FCB     EOT
F8D2
F8D2  10 DF 97             ZMONTR  STS     RSTACK          SAVE FOR BAD STACK RECOVERY
F8D5  6D 61                        TST     1,S             ? INIT CONSOLE AND SEND MSG
F8D7  26 0D                        BNE     ZMONT2          BRANCH IF NOT
F8D9  AD 9D E6 F9                  JSR     [VECTAB+.CION,PCR]      READY CONSOLE INPUT
F8DD  AD 9D E6 FB                  JSR     [VECTAB+.COON,PCR]      READY CONSOLE OUTPUT
F8E1  30 8C E5                     LEAX    SIGNON,PCR      READY SIGNON EYE-CATCHER
F8E4  3F                           SWI                     PERFORM
F8E5  03                           FCB     PDATA           PRINT STRING
F8E6  9E F6                ZMONT2  LDX     VECTAB+.PTM     LOAD PTM ADDRESS
F8E8  27 0D                        BEQ     CMD             BRANCH IF NOT TO USE A PTM
F8EA  6F 02                        CLR     PTMTM1-PTM,X    SET LATCH TO CLEAR RESET
F8EC  6F 03                        CLR     PTMTM1+1-PTM,X  AND SET GATE HIGH
F8EE  CC 01 A6                     LDD     #$01A6          SETUP TIMER 1 MODE
F8F1  A7 01                        STA     PTMC2-PTM,X     SETUP FOR CONTROL REGISTER1
F8F3  E7 00                        STB     PTMC13-PTM,X    SET OUTPUT ENABLED/
F8F5                       *     SINGLE SHOT/ DUAL 8 BIT/INTERNAL MODE/OPERATE
F8F5  6F 01                        CLR     PTMC2-PTM,X     SET CR2 BACK TO RESET FORM
F8F7                       * FALL INTO COMMAND PROCESSOR
F8F7
```

```
F8F7                  ***********************************************
F8F7                  *         COMMAND HANDLER
F8F7                  *  BREAKPOINTS ARE REMOVED AT THIS TIME.
F8F7                  *  PROMPT FOR A COMMAND, AND STORE ALL CHARACTERS
F8F7                  *  UNTIL A SEPARATOR ON THE STACK.
F8F7                  *  SEARCH FOR FIRST MATCHING COMMAND SUBSET,
F8F7                  *  CALL IT OR GIVE '?' RESPONSE.
F8F7                  *  DURING COMMAND SEARCH:
F8F7                  *      B=OFFSET TO NEXT ENTRY ON X
F8F7                  *      U=SAVED S
F8F7                  *      U-1=ENTRY SIZE+2
F8F7                  *      U-2=VALID NUMBER FLAG (>=0 VALID)/COMPARE CNT
F8F7                  *      U-3=CARRIAGE RETURN FLAG (0=CR HAS BEEN DONE)
F8F7                  *      U-4=START OF COMMAND STORE
F8F7                  *      S+0=END OF COMMAND STORE
F8F7                  *********************************************
F8F7  3F              CMD     SWI                     TO NEW LINE
F8F8  06                      FCB     PCRLF           FUNCTION
F8F9                  * DISARM THE BREAKPOINTS
F8F9  17 06 46        CMDNEP  LBSR    CBKLDR          OBTAIN BREAKPOINT POINTERS
F8FC  2A 0C                   BPL     CMDNOL          BRANCH IF NOT ARMED OR NONE
F8FE  50                      NEGB                    MAKE POSITIVE
F8FF  D7 FA                   STB     BKPTCT          FLAG AS DISARMED
F901  5A              CMDDDL  DECB                    ? FINISHED
F902  2B 06                   BMI     CMDNOL          BRANCH IF SO
F904  A6 30                   LDA     -NUMBKP*2,Y     LOAD OPCODE STORED
F906  A7 B1                   STA     [,Y++]          STORE BACK OVER "SWI"
F908  20 F7                   BRA     CMDDDL          LOOP UNTIL DONE
F90A  AE 6A           CMDNOL  LDX     10,S            LOAD USERS PROGRAM COUNTER
F90C  9F 93                   STX     PCNTER          SAVE FOR EXPRESSION ANALYZER
F90E  86 3E                   LDA     #PROMPT         LOAD PROMPT CHARACTER
F910  3F                      SWI                     SEND TO OUTPUT HANDLER
F911  01                      FCB     OUTCH           FUNCTION
F912  33 E4                   LEAU    ,S              REMEMBER STACK RESTORE ADDRESS
F914  DF 95                   STU     PSTACK          REMEMBER STACK FOR ERROR USE
F916  4F                      CLRA                    PREPARE ZERO
F917  5F                      CLRB                    PREPARE ZERO
F918  DD 9B                   STD     NUMBER          CLEAR NUMBER BUILD AREA
F91A  DD 8F                   STD     MISFLG          CLEAR MISCEL. AND SWICNT FLAGS
F91C  DD 91                   STD     TRACEC          CLEAR TRACE COUNT
F91E  C6 02                   LDB     #2              SET D TO TWO
F920  34 07                   PSHS    D,CC            PLACE DEFAULTS ONTO STACK
F922                  * CHECK FOR "QUICK" COMMANDS.
F922  17 04 54                LBSR    READ            OBTAIN FIRST CHARACTER
F925  30 8D 05 81             LEAX    CDOT+2,PCR      PRESET FOR SINGLE TRACE
F929  81 2E                   CMPA    #'.'            ? QUICK TRACE
F92B  27 5A                   BEQ     CMDXQT          BRANCH EQUAL FOR TRACE ONE
F92D  30 8D 04 E9             LEAX    CMPADP+2,PCR    READY MEMORY ENTRY POINT
F931  81 2F                   CMPA    #'/'            ? OPEN LAST USED MEMORY
F933  27 52                   BEQ     CMDXQT          BRANCH TO DO IT IF SO
F935                  * PROCESS NEXT CHARACTER
F935  81 20           CMD2    CMPA    #' '            ? BLANK OR DELIMITER
F937  23 14                   BLS     CMDGOT          BRANCH YES, WE HAVE IT
F939  34 02                   PSHS    A               BUILD ONTO STACK
F93B  6C 5F                   INC     -1,U            COUNT THIS CHARACTER
F93D  81 2F                   CMPA    #'/'            ? MEMORY COMMAND
F93F  27 4F                   BEQ     CMDMEM          BRANCH IF SO
F941  17 04 0B                LBSR    BLDHXC          TREAT AS HEX VALUE
F944  27 02                   BEQ     CMD3            BRANCH IF STILL VALID NUMBER
F946  6A 5E                   DEC     -2,U            FLAG AS INVALID NUMBER
F948  17 04 2E        CMD3    LBSR    READ            OBTAIN NEXT CHARACTER
F94B  20 E8                   BRA     CMD2            TEST NEXT CHARACTER
F94D                  * GOT COMMAND, NOW SEARCH TABLES
F94D  80 0D           CMDGOT  SUBA    #CR             SET ZERO IF CARRIAGE RETURN
F94F  A7 5D                   STA     -3,U            SETUP FLAG
F951  9E C4                   LDX     VECTAB+.CMDL1   START WITH FIRST CMD LIST
F953  E6 80           CMDSCH  LDB     ,X+             LOAD ENTRY LENGTH
F955  2A 10                   BPL     CMDSME          BRANCH IF NOT LIST END
```

```
F957  9E EE                 429          LDX     VECTAB+.CMDL2   NOW TO SECOND CMD LIST
F959  5C                    430          INCB                    ? TO CONTINUE TO DEFAULT LIST
F95A  27 F7                 431          BEQ     CMDSCH          BRANCH IF SO
F95C  10 DE 95              432  CMDBAD  LDS     PSTACK          RESTORE STACK
F95F  30 8D 01 5A           433          LEAX    ERRMSG,PCR      POINT TO ERROR STRING
F963  3F                    434          SWI                     SEND OUT
F964  02                    435          FCB     PDATA1          TO CONSOLE
F965  20 90                 436          BRA     CMD             AND TRY AGAIN
F967                        437  * SEARCH NEXT ENTRY
F967  5A                    438  CMDSME  DECB                    TAKE ACCOUNT OF LENGTH BYTE
F968  E1 5F                 439          CMPB    -1,U            ? ENTERED LONGER THAN ENTRY
F96A  24 03                 440          BHS     CMDSIZ          BRANCH IF NOT TOO LONG
F96C  3A                    441  CMDFLS  ABX     SKIP            TO NEXT ENTRY
F96D  20 E4                 442          BRA     CMDSCH          AND TRY NEXT
F96F  31 5D                 443  CMDSIZ  LEAY    -3,U            PREPARE TO COMPARE
F971  A6 5F                 444          LDA     -1,U            LOAD SIZE+2
F973  80 02                 445          SUBA    #2              TO ACTUAL SIZE ENTERED
F975  A7 5E                 446          STA     -2,U            SAVE SIZE FOR COUNTDOWN
F977  5A                    447  CMDCMP  DECB                    DOWN ONE BYTE
F978  A6 80                 448          LDA     ,X+             NEXT COMMAND CHARACTER
F97A  A1 A2                 449          CMPA    ,-Y             ? SAME AS THAT ENTERED
F97C  26 EE                 450          BNE     CMDFLS          BRANCH TO FLUSH  IF NOT
F97E  6A 5E                 451          DEC     -2,U            COUNT DOWN LENGTH OF ENTRY
F980  26 F5                 452          BNE     CMDCMP          BRANCH IF MORE TO TEST
F982  3A                    453          ABX     TO              NEXT ENTRY
F983  EC 1E                 454          LDD     -2,X            LOAD OFFSET
F985  30 8B                 455          LEAX    D,X             COMPUTE ROUTINE ADDRESS+2
F987  6D 5D                 456  CMDXQT  TST     -3,U            SET CC FOR CARRIAGE RETURN TEST
F989  32 C4                 457          LEAS    ,U              DELETE STACK WORK AREA
F98B  AD 1E                 458          JSR     -2,X            CALL COMMAND
F98D  16 FF 7A              459          LBRA    CMDNOL          GO GET NEXT COMMAND
F990  6D 5E                 460  CMDMEM  TST     -2,U            ? VALID HEX NUMBER ENTERED
F992  2B C8                 461          BMI     CMDBAD          BRANCH ERROR IF NOT
F994  30 88 AE              462          LEAX    <CMEMN-CMPADP,X TO DIFFERENT ENTRY
F997  DC 9B                 463          LDD     NUMBER          LOAD NUMBER ENTERED
F999  20 EC                 464          BRA     CMDXQT          AND ENTER MEMORY COMMAND
F99B                        465
F99B                        466  ** COMMANDS ARE ENTERED AS A SUBROUTINE WITH:
F99B                        467  **    DPR->ASSIST09 DIRECT PAGE WORK AREA
F99B                        468  **    Z=1 CARRIAGE RETURN ENTERED
F99B                        469  **    Z=0 NON CARRIAGE RETURN DELIMITER
F99B                        470  **    S=NORMAL RETURN ADDRESS
F99B                        471  ** THE LABEL "CMDBAD" MAY BE ENTERED TO ISSUE AN
F99B                        472  ** AN ERROR FLAG (*).
F99B                        473
F99B                        474  **************************************************
F99B                        475  *        ASSIST09 COMMAND TABLES
F99B                        476  *  THESE ARE THE DEFAULT COMMAND TABLES.  EXTERNAL
F99B                        477  *  TABLES OF THE SAME FORMAT MAY EXTEND/REPLACE
F99B                        478  *  THESE BY USING THE VECTOR SWAP FUNCTION.
F99B                        479  *
F99B                        480  * ENTRY FORMAT:
F99B                        481  *     +0...TOTAL SIZE OF ENTRY (INCLUDING THIS BYTE)
F99B                        482  *     +1...COMMAND STRING
F99B                        483  *     +N...TWO BYTE OFFSET TO COMMAND (ENTRYADDR-*)
F99B                        484  *
F99B                        485  *  THE TABLES TERMINATE WITH A ONE BYTE -1 OR -2.
F99B                        486  *  THE -1 CONTINUES THE COMMAND SEARCH WITH THE
F99B                        487  *         SECOND COMMAND TABLE.
F99B                        488  *  THE -2 TERMINATES COMMAND SEARCHES.
F99B                        489  ***************************************************
F99B                        490
F99B                        491  * THIS IS THE DEFAULT LIST FOR THE SECOND COMMAND
F99B                        492  * LIST ENTRY.
F99B  FE                    493  CMDTB2  FCB     -2              STOP COMMAND SEARCHES
F99C                        494
F99C                        495  * THIS IS THE DEFAULT LIST FOR THE FIRST COMMAND
F99C                        496  * LIST ENTRY.
```

```
F99C                CMDTBL  EQU     *               MONITOR COMMAND TABLE
F99C  04                    FCB     4
F99D  42                    FCC     /B/             'BREAKPOINT' COMMAND
F99E  05 4D                 FDB     CBKPT-*
F9A0  04                    FCB     4
F9A1  43                    FCC     /C/             'CALL' COMMAND
F9A2  04 17                 FDB     CCALL-*
F9A4  04                    FCB     4
F9A5  44                    FCC     /D/             'DISPLAY' COMMAND
F9A6  04 9D                 FDB     CDISP-*
F9A8  04                    FCB     4
F9A9  45                    FCC     /E/             'ENCODE' COMMAND
F9AA  05 9F                 FDB     CENCDE-*
F9AC  04                    FCB     4
F9AD  47                    FCC     /G/             'GO' COMMAND
F9AE  03 D2                 FDB     CGO-*
F9B0  04                    FCB     4
F9B1  4C                    FCC     /L/             'LOAD' COMMAND
F9B2  04 DD                 FDB     CLOAD-*
F9B4  04                    FCB     4
F9B5  4D                    FCC     /M/             'MEMORY' COMMAND
F9B6  04 0D                 FDB     CMEM-*
F9B8  04                    FCB     4
F9B9  4E                    FCC     /N/             'NULLS' COMMAND
F9BA  04 FD                 FDB     CNULLS-*
F9BC  04                    FCB     4
F9BD  4F                    FCC     /O/             'OFFSET' COMMAND
F9BE  05 0A                 FDB     COFFS-*
F9C0  04                    FCB     4
F9C1  50                    FCC     /P/             'PUNCH' COMMAND
F9C2  04 AF                 FDB     CPUNCH-*
F9C4  04                    FCB     4
F9C5  52                    FCC     /R/             'REGISTERS' COMMAND
F9C6  02 84                 FDB     CREG-*
F9C8  04                    FCB     4
F9C9  53                    FCC     /S/             'STLEVEL' COMMAND
F9CA  04 F2                 FDB     CSTLEV-*
F9CC  04                    FCB     4
F9CD  54                    FCC     /T/             'TRACE' COMMAND
F9CE  04 D6                 FDB     CTRACE-*
F9D0  04                    FCB     4
F9D1  56                    FCC     /V/             'VERIFY' COMMAND
F9D2  04 CF                 FDB     CVER-*
F9D4  04                    FCB     4
F9D5  57                    FCC     /W/             'WINDOW' COMMAND
F9D6  04 68                 FDB     CWINDO-*
F9D8  FF                    FCB     -1              END, CONTINUE WITH THE SECOND
F9D9
F9D9                ***********************************************
F9D9                *              [SWI FUNCTIONS 4 AND 5]
F9D9                *      4 - OUT2HS - DECODE BYTE TO HEX AND ADD SPACE
F9D9                *      5 - OUT4HS - DECODE WORD TO HEX AND ADD SPACE
F9D9                * INPUT: X->BYTE OR WORD TO DECODE
F9D9                * OUTPUT: CHARACTERS SENT TO OUTPUT HANDLER
F9D9                *         X->NEXT BYTE OR WORD
F9D9                ************************************************
F9D9
F9D9  A6 80         ZOUT2H  LDA     ,X+             LOAD NEXT BYTE
F9DB  34 06                 PSHS    D               SAVE - DO NOT REREAD
F9DD  C6 10                 LDB     #16             SHIFT BY 4 BITS
F9DF  3D                    MUL                     WITH MULTIPLY
F9E0  8D 04                 BSR     ZOUTHX          SEND OUT AS HEX
F9E2  35 06                 PULS    D               RESTORE BYTES
F9E4  84 0F                 ANDA    #$0F            ISOLATE RIGHT HEX
F9E6  8B 90         ZOUTHX  ADDA    #$90            PREPARE A-F ADJUST
F9E8  19                    DAA     ADJUST
F9E9  89 40                 ADCA    #$40            PREPARE CHARACTER BITS
F9EB  19                    DAA     ADJUST
```

```
F9EC  6E 9D E5 EE      SEND    JMP     [VECTAB+.CODTA,PCR]     SEND TO OUT HANDLER
F9F0
F9F0  8D E7            ZOT4HS  BSR     ZOUT2H          CONVERT FIRST BYTE
F9F2  8D E5            ZOT2HS  BSR     ZOUT2H          CONVERT BYTE TO HEX
F9F4  AF 64                    STX     4,S             UPDATE USERS X REGISTER
F9F6                   * FALL INTO SPACE ROUTINE
F9F6
F9F6                   ************************************************
F9F6                   *             [SWI FUNCTION 7]
F9F6                   *          SPACE - SEND BLANK TO OUTPUT HANDLER
F9F6                   * INPUT: NONE
F9F6                   * OUTPUT: BLANK SEND TO CONSOLE HANDLER
F9F6                   ************************************************
F9F6  86 20            ZSPACE  LDA     #' '            LOAD BLANK
F9F8  20 3D                    BRA     ZOTCH2          SEND AND RETURN
F9FA
F9FA                   ***********************************************
F9FA                   *             [SWI FUNCTION 9]
F9FA                   *          SWAP VECTOR TABLE ENTRY
F9FA                   * INPUT: A=VECTOR TABLE CODE (OFFSET)
F9FA                   *        X=0 OR REPLACEMENT VALUE
F9FA                   * OUTPUT: X=PREVIOUS VALUE
F9FA                   ***********************************************
F9FA  A6 61            ZVSWTH  LDA     1,S             LOAD REQUESTERS A
F9FC  81 34                    CMPA    #HIVTR          ? SUB-CODE TOO HIGH
F9FE  22 39                    BHI     ZOTCH3          IGNORE CALL IF SO
FA00  10 9E C2                 LDY     VECTAB+.AVTBL   LOAD VECTOR TABLE ADDRESS
FA03  EE A6                    LDU     A,Y             U=OLD ENTRY
FA05  EF 64                    STU     4,S             RETURN OLD VALUE TO CALLERS X
FA07  AF 7E                    STX     -2,S            ? X=0
FA09  27 2E                    BEQ     ZOTCH3          YES, DO NOT CHANGE ENTRY
FA0B  AF A6                    STX     A,Y             REPLACE ENTRY
FA0D  20 2A                    BRA     ZOTCH3          RETURN FROM SWI
FA0F                   *D
FA0F
FA0F                   ************************************************
FA0F                   *                    [SWI FUNCTION 0]
FA0F                   *   INCHNP - OBTAIN INPUT CHAR IN A (NO PARITY)
FA0F                   *  NULLS AND RUBOUTS ARE IGNORED.
FA0F                   *  AUTOMATIC LINE FEED IS SENT UPON RECIEVING A
FA0F                   *      CARRIAGE RETURN.
FA0F                   *  UNLESS WE ARE LOADING FROM TAPE.
FA0F                   ************************************************
FA0F  8D 5D            ZINCHP  BSR     XQPAUS          RELEASE PROCESSOR
FA11  8D 5F            ZINCH   BSR     XQCIDT          CALL INPUT DATA APPENDAGE
FA13  24 FA                    BCC     ZINCHP          LOOP IF NONE AVAILABLE
FA15  4D                       TSTA    ?               TEST FOR NULL
FA16  27 F9                    BEQ     ZINCH           IGNORE NULL
FA18  81 7F                    CMPA    #$7F            ? RUBOUT
FA1A  27 F5                    BEQ     ZINCH           BRANCH YES TO IGNORE
FA1C  A7 61                    STA     1,S             STORE INTO CALLERS A
FA1E  0D 8F                    TST     MISFLG          ? LOAD IN PROGRESS
FA20  26 17                    BNE     ZOTCH3          BRANCH IF SO TO NOT ECHO
FA22  81 0D                    CMPA    #CR             ? CARRIAGE RETURN
FA24  26 04                    BNE     ZIN2            NO, TEST ECHO BYTE
FA26  86 0A                    LDA     #LF             LOAD LINE FEED
FA28  8D C2                    BSR     SEND            ALWAYS ECHO LINE FEED
FA2A  0D F4            ZIN2    TST     VECTAB+.ECHO    ? ECHO DESIRED
FA2C  26 0B                    BNE     ZOTCH3          NO, RETURN
FA2E                   * FALL THROUGH TO OUTCH
FA2E
FA2E                   ************************************************
FA2E                   *             [SWI FUNCTION 1]
FA2E                   *          OUTCH - OUTPUT CHARACTER FROM A
FA2E                   *  INPUT:  NONE
FA2E                   *  OUTPUT: IF LINEFEED IS THE OUTPUT CHARACTER THEN
FA2E                   *            C=0 NO CTL-X RECIEVED, C=1 CTL-X RECIEVED
FA2E                   ************************************************
```

```
FA2E  A6 61          ZOTCH1  LDA     1,S             LOAD CHARACTER TO SEND
FA30  30 8C 09               LEAX    <ZPCRLS,PCR     DEFAULT FOR LINE FEED
FA33  81 0A                  CMPA    #LF             ? LINE FEED
FA35  27 0F                  BEQ     ZPDTLP          BRANCH TO CHECK PAUSE IF SO
FA37  8D B3          ZOTCH2  BSR     SEND            SEND TO OUTPUT ROUTINE
FA39  0C 90          ZOTCH3  INC     SWICNT          BUMP UP "SWI" TRACE NEST LEVEL
FA3B  3B                     RTI     RETURN          FROM "SWI" FUNCTION
FA3C
FA3C                 **************************************************
FA3C                 *               [SWI FUNCTION 6]
FA3C                 *         PCRLF - SEND CR/LF TO CONSOLE HANDLER
FA3C                 *  INPUT: NONE
FA3C                 *  OUTPUT: CR AND LF SENT TO HANDLER
FA3C                 *           C=0 NO CTL-X, C=1 CTL-X RECIEVED
FA3C                 **************************************************
FA3C
FA3C  04             ZPCRLS  FCB     EOT             NULL STRING
FA3D
FA3D  30 8C FC       ZPCRLF  LEAX    ZPCRLS,PCR      READY CR,LF STRING
FA40                 * FALL INTO CR/LF CODE
FA40
FA40                 **************************************************
FA40                 *              [SWI FUNCTION 3]
FA40                 *         PDATA - OUTPUT CR/LF AND STRING
FA40                 * INPUT: X->STRING
FA40                 * OUTPUT: CR/LF AND STRING SENT TO OUTPUT CONSOLE
FA40                 *          HANDLER.
FA40                 *      C=0 NO CTL-X, C=1 CTL-X RECIEVED
FA40                 * NOTE: LINE FEED MUST FOLLOW CARRIAGE RETURN FOR
FA40                 *        PROPER PUNCH DATA.
FA40                 **************************************************
FA40  86 0D          ZPDATA  LDA     #CR             LOAD CARRIAGE RETURN
FA42  8D A8                  BSR     SEND            SEND IT
FA44  86 0A                  LDA     #LF             LOAD LINE FEED
FA46                 * FALL INTO PDATA1
FA46
FA46                 *************************************************
FA46                 *              [SWI FUNCTION 2]
FA46                 *          PDATA1 - OUTPUT STRING TILL EOT ($04)
FA46                 *  THIS ROUTINE PAUSES IF AN INPUT BYTE BECOMES
FA46                 *  AVAILABLE DURING OUTPUT TRANSMISSION UNTIL A
FA46                 *  SECOND IS RECIEVED.
FA46                 * INPUT: X->STRING
FA46                 * OUTPUT: STRING SENT TO OUTPUT CONSOLE DRIVER
FA46                 *          C=0 NO CTL-X, C=1 CTL-X RECIEVED
FA46                 *************************************************
FA46  8D A4          ZPDTLP  BSR     SEND            SEND CHARACTER TO DRIVER
FA48  A6 80          ZPDTA1  LDA     ,X+             LOAD NEXT CHARACTER
FA4A  81 04                  CMPA    #EOT            ? EOT
FA4C  26 F8                  BNE     ZPDTLP          LOOP IF NOT
FA4E                 * FALL INTO PAUSE CHECK FUNCTION
FA4E
FA4E                 *******************************************
FA4E                 *             [SWI FUNCTION 12]
FA4E                 *      PAUSE - RETURN TO TASK DISPATCHING AND CHECK
FA4E                 *              FOR FREEZE CONDITION OR CTL-X BREAK
FA4E                 *  THIS FUNCTION ENTERS THE TASK PAUSE HANDLER SO
FA4E                 *  OPTIONALLY OTHER 6809 PROCESSES MAY GAIN CONTROL.
FA4E                 *  UPON RETURN, CHECK FOR A 'FREEZE' CONDITION
FA4E                 *  WITH A RESULTING WAIT LOOP, OR CONDITION CODE
FA4E                 *  RETURN IF A CONTROL-X IS ENTERED FROM THE INPUT
FA4E                 *  HANDLER.
FA4E                 * OUTPUT: C=1 IF CTL-X HAS ENTERED, C=0 OTHERWISE
FA4E                 ******************************************
FA4E  8D 1E          ZPAUSE  BSR     XQPAUS          RELEASE CONTROL AT EVERY LINE
FA50  8D 06                  BSR     CHKABT          CHECK FOR FREEZE OR ABORT
FA52  1F A9                  TFR     CC,B            PREPARE TO REPLACE CC
FA54  E7 E4                  STB     ,S              OVERLAY OLD ONE ON STACK
```

```
FA56  20 E1                      BRA     ZOTCH3          RETURN FROM "SWI"
FA58
FA58                      * CHKABT - SCAN FOR INPUT PAUSE/ABORT DURING OUTPUT
FA58                      * OUTPUT: C=0 OK, C=1 ABORT (CTL-X ISSUED)
FA58                      * VOLATILE: U,X,D
FA58  8D 18               CHKABT BSR     XQCIDT          ATTEMPT INPUT
FA5A  24 05                      BCC     CHKRTN          BRANCH NO TO RETURN
FA5C  81 18                      CMPA    #CAN            ? CTL-X FOR ABORT
FA5E  26 02                      BNE     CHKWT           BRANCH NO TO PAUSE
FA60  53                  CHKSEC COMB    SET             CARRY
FA61  39                  CHKRTN RTS                     RETURN TO CALLER WITH CC SET
FA62  8D 0A               CHKWT  BSR     XQPAUS          PAUSE FOR A MOMENT
FA64  8D 0C                      BSR     XQCIDT          ? KEY FOR START
FA66  24 FA                      BCC     CHKWT           LOOP UNTIL RECIEVED
FA68  81 18                      CMPA    #CAN            ? ABORT SIGNALED FROM WAIT
FA6A  27 F4                      BEQ     CHKSEC          BRANCH YES
FA6C  4F                         CLRA                    SET C=0 FOR NO ABORT
FA6D  39                         RTS                     AND RETURN
FA6E
FA6E                      * SAVE MEMORY WITH JUMPS
FA6E  6E 9D E5 78         XQPAUS JMP     [VECTAB+.PAUSE,PCR]     TO PAUSE ROUTINE
FA72  AD 9D E5 62         XQCIDT JSR     [VECTAB+.CIDTA,PCR]     TO INPUT ROUTINE
FA76  84 7F                      ANDA    #$7F            STRIP PARITY
FA78  39                         RTS                     RETURN TO CALLER
FA79
FA79                      ********************************************
FA79                      *          NMI DEFAULT INTERRUPT HANDLER
FA79                      *  THE NMI HANDLER IS USED FOR TRACING INSTRUCTIONS.
FA79                      *  TRACE PRINTOUTS OCCUR ONLY AS LONG AS THE STACK
FA79                      *  TRACE LEVEL IS NOT BREACHED BY FALLING BELOW IT.
FA79                      *  TRACING CONTINUES UNTIL THE COUNT TURNS ZERO OR
FA79                      *  A CTL-X IS ENTERED FROM THE INPUT CONSOLE DEVICE.
FA79                      ********************************************
FA79
FA79  4F 50 2D 04         MSHOWP FCB     'O','P','-',EOT OPCODE PREP
FA7D
FA7D  8D 42               NMIR   BSR     LDDP            LOAD PAGE AND VERIFY STACK
FA7F  0D 8F                      TST     MISFLG          ? THRU A BREAKPOINT
FA81  26 34                      BNE     NMICON          BRANCH IF SO TO CONTINUE
FA83  0D 90                      TST     SWICNT          ? INHIBIT "SWI" DURING TRACE
FA85  2B 29                      BMI     NMITRC          BRANCH YES
FA87  30 6C                      LEAX    12,S            OBTAIN USERS STACK POINTER
FA89  9C F8                      CMPX    SLEVEL          ? TO TRACE HERE
FA8B  25 23                      BLO     NMITRC          BRANCH IF TOO LOW TO DISPLAY
FA8D  30 8C E9                   LEAX    MSHOWP,PCR      LOAD OP PREP
FA90  3F                         SWI                     SEND TO CONSOLE
FA91  02                         FCB     PDATA1          FUNCTION
FA92  09 8E                      ROL     DELIM           SAVE CARRY BIT
FA94  30 8D E5 01                LEAX    LASTOP,PCR      POINT TO LAST OP
FA98  3F                         SWI                     SEND OUT AS HEX
FA99  05                         FCB     OUT4HS          FUNCTION
FA9A  8D 17                      BSR     REGPRS          FOLLOW MEMORY WITH REGISTERS
FA9C  25 37                      BCS     ZBKCMD          BRANCH IF "CANCEL"
FA9E  06 8E                      ROR     DELIM           RESTORE CARRY BIT
FAA0  25 33                      BCS     ZBKCMD          BRANCH IF "CANCEL"
FAA2  9E 91                      LDX     TRACEC          LOAD TRACE COUNT
FAA4  27 2F                      BEQ     ZBKCMD          IF ZERO TO COMMAND HANDLER
FAA6  30 1F                      LEAX    -1,X            MINUS ONE
FAA8  9F 91                      STX     TRACEC          REFRESH
FAAA  27 29                      BEQ     ZBKCMD          STOP TRACE WHEN ZERO
FAAC  8D AA                      BSR     CHKABT          ? ABORT THE TRACE
FAAE  25 25                      BCS     ZBKCMD          BRANCH YES TO COMMAND HANDLER
FAB0  16 03 F7            NMITRC LBRA    CTRCE3          NO, TRACE ANOTHER INSTRUCTION
FAB3
FAB3  17 01 B9            REGPRS LBSR    REGPRT          PRINT REGISTERS AS FROM COMMAND
FAB6  39                         RTS                     RETURN TO CALLER
FAB7
FAB7                      * JUST EXECUTED THRU A BRKPNT.  NOW CONTINUE NORMALLY
```

```
FAB7  0F 8F                NMICON  CLR     MISFLG          CLEAR THRU FLAG
FAB9  17 02 EB                     LBSR    ARMBK2          ARM BREAKPOINTS
FABC  3B                   RTI     RTI     AND             CONTINUE USERS PROGRAM
FABD
FABD                       * LDDP - SETUP DIRECT PAGE REGISTER, VERIFY STACK.
FABD                       * AN INVALID STACK CAUSES A RETURN TO THE COMMAND
FABD                       * HANDLER.
FABD                       * INPUT: FULLY STACKED REGISTERS FROM AN INTERRUPT
FABD                       * OUTPUT: DPR LOADED TO WORK PAGE
FABD
FABD  3F 07 20 04          ERRMSG  FCB     '?',BELL,$20,EOT ERROR RESPONSE
FAC1
FAC1  E6 8D E4 D8          LDDP    LDB     BASEPG,PCR      LOAD DIRECT PAGE HIGH BYTE
FAC5  1F 9B                        TFR     B,DP            SETUP DIRECT PAGE REGISTER
FAC7  A1 63                        CMPA    3,S             ? IS STACK VALID
FAC9  27 25                        BEQ     RTS             YES, RETURN
FACB  10 DE 97                     LDS     RSTACK          RESET TO INITIAL STACK POINTER
FACE  30 8C EC             ERROR   LEAX    ERRMSG,PCR      LOAD ERROR REPORT
FAD1  3F                           SWI                     SEND OUT BEFORE REGISTERS
FAD2  03                           FCB     PDATA           ON NEXT LINE
FAD3                       * FALL INTO BREAKPOINT HANDLER
FAD3
FAD3                       **********************************************
FAD3                       *            [SWI FUNCTION 10]
FAD3                       *        BREAKPOINT PROGRAM FUNCTION
FAD3                       *  PRINT REGISTERS AND GO TO COMMAND HANLER
FAD3                       ***********************************************
FAD3  8D DE                ZBKPNT  BSR     REGPRS          PRINT OUT REGISTERS
FAD5  16 FE 21             ZBKCMD  LBRA    CMDNEP          NOW ENTER COMMAND HANDLER
FAD8
FAD8                       ********************************************
FAD8                       *    IRQ, RESERVED, SWI2 AND SWI3 INTERRUPT HANDLERS
FAD8                       *  THE DEFAULT HANDLING IS TO CAUSE A BREAKPOINT.
FAD8                       ********************************************
FAD8                       SWI2R   EQU     *               SWI2 ENTRY
FAD8                       SWI3R   EQU     *               SWI3 ENTRY
FAD8                       IRQR    EQU     *               IRQ ENTRY
FAD8  8D E7                RSRVDR  BSR     LDDP            SET BASE PAGE, VALIDATE STACK
FADA  20 F7                        BRA     ZBKPNT          FORCE A BREAKPOINT
FADC
FADC                       *****************************************
FADC                       *       FIRQ HANDLER
FADC                       *  JUST RETURN FOR THE FIRQ INTERRUPT
FADC                       *****************************************
FABC                       FIRQR   EQU     RTI             IMMEDIATE RETURN
FADC
FADC                       **************************************************
FADC                       *      DEFAULT I/O DRIVERS
FADC                       **************************************************
FADC
FADC                       * CIDTA - RETURN CONSOLE INPUT CHARACTER
FADC                       * OUTPUT: C=0 IF NO DATA READY, C=1 A=CHARACTER
FADC                       * U VOLATILE
FADC  DE F0                CIDTA   LDU     VECTAB+.ACIA    LOAD ACIA ADDRESS
FADE  A6 C4                        LDA     ,U              LOAD STATUS REGISTER
FAE0  44                           LSRA                    TEST RECEIVER REGISTER FLAG
FAE1  24 02                        BCC     CIRTN           RETURN IF NOTHING
FAE3  A6 41                        LDA     1,U             LOAD DATA BYTE
FAE5  39                   CIRTN   RTS                     RETURN TO CALLER
FAE6
FAE6                       * CION - INPUT CONSOLE INITIALIZATION
FAE6                       * COON - OUTPUT CONSOLE INITIALIZATION
FAE6                       * A,X  VOLATILE
FAE6                       CION    EQU     *
FAE6  86 03                COON    LDA     #3              RESET ACIA CODE
FAE8  9E F0                        LDX     VECTAB+.ACIA    LOAD ACIA ADDRESS
FAEA  A7 84                        STA     ,X              STORE INTO STATUS REGISTER
FAEC  86 51                        LDA     #$51            SET CONTROL
```

```
FAEE  A7 84                    STA     ,X               REGISTER UP
FAF0  39                 RTS     RTS                      RETURN TO CALLER
FAF1
FAF1                     * THE FOLLOWING HAVE NO DUTIES TO PERFORM
FAF0                     CIOFF   EQU     RTS              CONSOLE INPUT OFF
FAF0                     COOFF   EQU     RTS              CONSOLE OUTPUT OFF
FAF1
FAF1                     * CODTA - OUTPUT CHARACTER TO CONSOLE DEVICE
FAF1                     * INPUT: A=CHARACTER TO SEND
FAF1                     * OUTPUT: CHAR SENT TO TERMINAL WITH PROPER PADDING
FAF1                     * ALL REGISTERS TRANSPARENT
FAF1
FAF1  34 47              CODTA   PSHS    U,D,CC           SAVE REGISTERS,WORK BYTE
FAF3  DE F0                      LDU     VECTAB+.ACIA     ADDRESS ACIA
FAF5  8D 1B                      BSR     CODTAO           CALL OUTPUT CHAR SUBROTINE
FAF7  81 10                      CMPA    #DLE             ? DATA LINE ESCAPE
FAF9  27 12                      BEQ     CODTRT           YES, RETURN
FAFB  D6 F2                      LDB     VECTAB+.PAD      DEFAULT TO CHAR PAD COUNT
FAFD  81 0D                      CMPA    #CR              ? CR
FAFF  26 02                      BNE     CODTPD           BRANCH NO
FB01  D6 F3                      LDB     VECTAB+.PAD+1    LOAD NEW LINE PAD COUNT
FB03  4F                 CODTPD  CLRA                     CREATE NULL
FB04  E7 E4                      STB     ,S               SAVE COUNT
FB06  8C                         FCB     SKIP2            ENTER LOOP
FB07  8D 09              CODTLP  BSR     CODTAO           SEND NULL
FB09  6A E4                      DEC     ,S               ? FINISHED
FB0B  2A FA                      BPL     CODTLP           NO, CONTINUE WITH MORE
FB0D  35 C7              CODTRT  PULS    PC,U,D,CC        RESTORE REGISTERS AND RETURN
FB0F
FB0F  17 FF 5C           CODTAD  LBSR    XQPAUS           TEMPORARY GIVE UP CONTROL
FB12  E6 C4              CODTAO  LDB     ,U               LOAD ACIA CONTROL REGISTER
FB14  C5 02                      BITB    #$02             ? TX REGISTER CLEAR >LSAB FIXME
FB16  26 F7                      BNE     CODTAD           RELEASE CONTROL IF NOT
FB18  A7 41                      STA     1,U              STORE INTO DATA REGISTER
FB1A  39                         RTS                      RETURN TO CALLER
FB1B                     *E
FB1B
FB1B                     * BSON - TURN ON READ/VERIFY/PUNCH MECHANISM
FB1B                     * A IS VOLATILE
FB1B
FB1B  86 11              BSON    LDA     #$11             SET READ CODE
FB1D  6D 66                      TST     6,S              ? READ OR VERIFY
FB1F  26 01                      BNE     BSON2            BRANCH YES
FB21  4C                         INCA                     SET TO WRITE
FB22  3F                 BSON2   SWI                      PERFORM OUTPUT
FB23  01                         FCB     OUTCH            FUNCTION
FB24  0C 8F                      INC     MISFLG           SET LOAD IN PROGRESS FLAG
FB26  39                         RTS                      RETURN TO CALLER
FB27
FB27                     * BSOFF - TURN OFF READ/VERIFY/PUNCH MECHANISM
FB27                     * A,X VOLATILE
FB27  86 14              BSOFF   LDA     #$14             TO DC4 - STOP
FB29  3F                         SWI                      SEND OUT
FB2A  01                         FCB     OUTCH            FUNCTION
FB2B  4A                         DECA                     CHANGE TO DC3 (X-OFF)
FB2C  3F                         SWI                      SEND OUT
FB2D  01                         FCB     OUTCH            FUNCTION
FB2E  0A 8F                      DEC     MISFLG           CLEAR LOAD IN PROGRESS FLAG
FB30  8E 61 A8                   LDX     #25000           DELAY 1 SECOND (2MHZ CLOCK)
FB33  30 1F              BSOFLP  LEAX    -1,X             COUNT DOWN
FB35  26 FC                      BNE     BSOFLP           LOOP TILL DONE
FB37  39                         RTS                      RETURN TO CALLER
FB38
FB38                     * BSDTA - READ/VERIFY/PUNCH HANDLER
FB38                     * INPUT: S+6=CODE BYTE, VERIFY(-1),PUNCH(0),LOAD(1)
FB38                     *        S+4=START ADDRESS
FB38                     *        S+2=STOP ADDRESS
FB38                     *        S+0=RETURN ADDRESS
```

```
FB38                 * OUTPUT: Z=1 NORMAL COMPLETION, Z=0 INVALID LOAD/VER
FB38                 * REGISTERS ARE VOLATILE
FB38
FB38  EE 62          BSDTA   LDU     2,S             U=TO ADDRESS OR OFFSET
FB3A  6D 66                  TST     6,S             ? PUNCH
FB3C  27 54                  BEQ     BSDPUN          BRANCH YES
FB3E                 * DURING READ/VERIFY: S+2=MSB ADDRESS SAVE BYTE
FB3E                 *                     S+1=BYTE COUNTER
FB3E                 *                     S+0=CHECKSUM
FB3E                 *                     U HOLDS OFFSET
FB3E  32 7D                  LEAS    -3,S            ROOM FOR WORK/COUNTER/CHECKSUM
FB40  3F             BSDLD1  SWI                     GET NEXT CHARACTER
FB41  00                     FCB     INCHNP          FUNCTION
FB42  81 53          BSDLD2  CMPA    #'S'            ? START OF S1/S9
FB44  26 FA                  BNE     BSDLD1          BRANCH NOT
FB46  3F                     SWI                     GET NEXT CHARACTER
FB47  00                     FCB     INCHNP          FUNCTION
FB48  81 39                  CMPA    #'9'            ? HAVE S9
FB4A  27 22                  BEQ     BSDSRT          YES, RETURN GOOD CODE
FB4C  81 31                  CMPA    #'1'            ? HAVE NEW RECORD
FB4E  26 F2                  BNE     BSDLD2          BRANCH IF NOT
FB50  6F E4                  CLR     ,S              CLEAR CHECKSUM
FB52  8D 21                  BSR     BYTE            OBTAIN BYTE COUNT
FB54  E7 61                  STB     1,S             SAVE FOR DECREMENT
FB56                 * READ ADDRESS
FB56  8D 1D                  BSR     BYTE            OBTAIN HIGH VALUE
FB58  E7 62                  STB     2,S             SAVE IT
FB5A  8D 19                  BSR     BYTE            OBTAIN LOW VALUE
FB5C  A6 62                  LDA     2,S             MAKE D=VALUE
FB5E  31 CB                  LEAY    D,U             Y=ADDRESS+OFFSET
FB60                 * STORE TEXT
FB60  8D 13          BSDNXT  BSR     BYTE            NEXT BYTE
FB62  27 0C                  BEQ     BSDEOL          BRANCH IF CHECKSUM
FB64  6D 69                  TST     9,S             ? VERIFY ONLY
FB66  2B 02                  BMI     BSDCMP          YES, ONLY COMPARE
FB68  E7 A4                  STB     ,Y              STORE INTO MEMORY
FB6A  E1 A0          BSDCMP  CMPB    ,Y+             ? VALID RAM
FB6C  27 F2                  BEQ     BSDNXT          YES, CONTINUE READING
FB6E  35 92          BSDSRT  PULS    PC,X,A          RETURN WITH Z SET PROPER
FB70
FB70  4C             BSDEOL  INCA                    ? VALID CHECKSUM
FB71  27 CD                  BEQ     BSDLD1          BRANCH YES
FB73  20 F9                  BRA     BSDSRT          RETURN Z=0 INVALID
FB75
FB75                 * BYTE BUILDS 8 BIT VALUE FROM TWO HEX DIGITS IN
FB75  8D 12          BYTE    BSR     BYTHEX          OBTAIN FIRST HEX
FB77  C6 10                  LDB     #16             PREPARE SHIFT
FB79  3D                     MUL                     OVER TO A
FB7A  8D 0D                  BSR     BYTHEX          OBTAIN SECOND HEX
FB7C  34 04                  PSHS    B               SAVE HIGH HEX
FB7E  AB E0                  ADDA    ,S+             COMBINE BOTH SIDES
FB80  1F 89                  TFR     A,B             SEND BACK IN B
FB82  AB 62                  ADDA    2,S             COMPUTE NEW CHECKSUM
FB84  A7 62                  STA     2,S             STORE BACK
FB86  6A 63                  DEC     3,S             DECREMENT BYTE COUNT
FB88  39             BYTRTS  RTS                     RETURN TO CALLER
FB89
FB89  3F             BYTHEX  SWI                     GET NEXT HEX
FB8A  00                     FCB     INCHNP          CHARACTER
FB8B  17 01 D4               LBSR    CNVHEX          CONVERT TO HEX
FB8E  27 F8                  BEQ     BYTRTS          RETURN IF VALID HEX
FB90  35 F2                  PULS    PC,U,Y,X,A      RETURN TO CALLER WITH Z=0
FB92
FB92                 * PUNCH STACK USE: S+8=TO ADDRESS
FB92                 *                  S+6=RETURN ADDRESS
FB92                 *                  S+4=SAVED PADDING VALUES
FB92                 *                  S+2 FROM ADDRESS
FB92                 *                  S+1=FRAME COUNT/CHECKSUM
```

```
FB92                      *               S+0=BYTE COUNT
FB92  DE F2               BSDPUN  LDU     VECTAB+.PAD     LOAD PADDING VALUES
FB94  AE 64                       LDX     4,S             X=FROM ADDRESS
FB96  34 56                       PSHS    U,X,D           CREATE STACK WORK AREA
FB98  CC 00 18                    LDD     #24             SET A=0, B=24
FB9B  D7 F2                       STB     VECTAB+.PAD     SETUP 24 CHARACTER PADS
FB9D  3F                          SWI                     SEND NULLS OUT
FB9E  01                          FCB     OUTCH           FUNCTION
FB9F  C6 04                       LDB     #4              SETUP NEW LINE PAD TO 4
FBA1  DD F2                       STD     VECTAB+.PAD     SETUP PUNCH PADDING
FBA3                      * CALCULATE SIZE
FBA3  EC 68               BSPGO   LDD     8,S             LOAD TO
FBA5  A3 62                       SUBD    2,S             MINUS FROM=LENGTH
FBA7  10 83 00 18                 CMPD    #24             ? MORE THAN 23
FBAB  25 02                       BLO     BSPOK           NO, OK
FBAD  C6 17                       LDB     #23             FORCE TO 23 MAX
FBAF  5C                  BSPOK   INCB                    PREPARE COUNTER
FBB0  E7 E4                       STB     ,S              STORE BYTE COUNT
FBB2  CB 03                       ADDB    #3              ADJUST TO FRAME COUNT
FBB4  E7 61                       STB     1,S             SAVE
FBB6                      *PUNCH CR,LF,NULS,S,1
FBB6  30 8C 33                    LEAX    <BSPSTR,PCR     LOAD START RECORD HEADER
FBB9  3F                          SWI                     SEND OUT
FBBA  03                          FCB     PDATA           FUNCTION
FBBB                      * SEND FRAME COUNT
FBBB  5F                          CLRB                    INITIALIZE CHECKSUM
FBBC  30 61                       LEAX    1,S             POINT TO FRAME COUNT AND ADDR
FBBE  8D 27                       BSR     BSPUN2          SEND FRAME COUNT
FBC0                      *DATA ADDRESS
FBC0  8D 25                       BSR     BSPUN2          SEND ADDRESS HI
FBC2  8D 23                       BSR     BSPUN2          SEND ADDRESS LOW
FBC4                      *PUNCH DATA
FBC4  AE 62                       LDX     2,S             LOAD START DATA ADDRESS
FBC6  8D 1F               BSPMRE  BSR     BSPUN2          SEND OUT NEXT BYTE
FBC8  6A E4                       DEC     ,S              ? FINAL BYTE
FBCA  26 FA                       BNE     BSPMRE          LOOP IF NOT DONE
FBCC  AF 62                       STX     2,S             UPDATE FROM ADDRESS VALUE
FBCE                      *PUNCH CHECKSUM
FBCE  53                          COMB    COMPLEMENT
FBCF  E7 61                       STB     1,S             STORE FOR SENDOUT
FBD1  30 61                       LEAX    1,S             POINT TO IT
FBD3  8D 14                       BSR     BSPUNC          SEND OUT AS HEX
FBD5  AE 68                       LDX     8,S             LOAD TOP ADDRESS
FBD7  AC 62                       CMPX    2,S             ? DONE
FBD9  24 C8                       BHS     BSPGO           BRANCH NOT
FBDB  30 8C 11                    LEAX    <BSPEOF,PCR     PREPARE END OF FILE
FBDE  3F                          SWI                     SEND OUT STRING
FBDF  03                          FCB     PDATA           FUNCTION
FBE0  EC 64                       LDD     4,S             RECOVER PAD COUNTS
FBE2  DD F2                       STD     VECTAB+.PAD     RESTORE
FBE4  4F                          CLRA                    SET Z=1 FOR OK RETURN
FBE5  35 D6                       PULS    PC,U,X,D        RETURN WITH OK CODE
FBE7
FBE7  EB 84               BSPUN2  ADDB    ,X              ADD TO CHECKSUM
FBE9  16 FD ED            BSPUNC  LBRA    ZOUT2H          SEND OUT AS HEX AND RETURN
FBEC
FBEC  53 31 04            BSPSTR  FCB     'S','1',EOT     CR,LF,NULLS,S,1
FBEF  53 39 30 33 30 30 + BSPEOF  FCC     /S9030000FC/    EOF STRING
FBF9  0D 0A 04                    FCB     CR,LF,EOT
FBFC
FBFC                      * HSDTA - HIGH SPEED PRINT MEMORY
FBFC                      * INPUT: S+4=START ADDRESS
FBFC                      *        S+2=STOP ADDRESS
FBFC                      *        S+0=RETURN ADDRESS
FBFC                      * X,D VOLATILE
FBFC
FBFC                      *  SEND TITLE
FBFC  3F                  HSDTA   SWI                     SEND NEW LINE
```

```
FBFD  06                       FCB    PCRLF           FUNCTION
FBFE  C6 06                    LDB    #6              PREPARE 6 SPACES
FC00  3F               HSBLNK  SWI                    SEND BLANK
FC01  07                       FCB    SPACE           FUNCTION
FC02  5A                       DECB                   COUNT DOWN
FC03  26 FB                    BNE    HSBLNK          LOOP IF MORE
FC05  5F                       CLRB                   SETUP BYTE COUNT
FC06  1F 98            HSHTTL  TFR    B,A             PREPARE FOR CONVERT
FC08  17 FD DB                 LBSR   ZOUTHX          CONVERT TO A HEX DIGIT
FC0B  3F                       SWI                    SEND BLANK
FC0C  07                       FCB    SPACE           FUNCTION
FC0D  3F                       SWI                    SEND ANOTHER
FC0E  07                       FCB    SPACE           BLANK
FC0F  5C                       INCB                   UP ANOTHER
FC10  C1 10                    CMPB   #$10            ? PAST 'F'
FC12  25 F2                    BLO    HSHTTL          LOOP UNTIL SO
FC14  3F               HSHLNE  SWI                    TO NEXT LINE
FC15  06                       FCB    PCRLF           FUNCTION
FC16  25 2F                    BCS    HSDRTN          RETURN IF USER ENTERED CTL-X
FC18  30 64                    LEAX   4,S             POINT AT ADDRESS TO CONVERT
FC1A  3F                       SWI                    PRINT OUT ADDRESS
FC1B  05                       FCB    OUT4HS          FUNCTION
FC1C  AE 64                    LDX    4,S             LOAD ADDRESS PROPER
FC1E  C6 10                    LDB    #16             NEXT SIXTEEN
FC20  3F               HSHNXT  SWI                    CONVERT BYTE TO HEX AND SEND
FC21  04                       FCB    OUT2HS          FUNCTION
FC22  5A                       DECB                   COUNT DOWN
FC23  26 FB                    BNE    HSHNXT          LOOP IF NOT SIXTEENTH
FC25  3F                       SWI                    SEND BLANK
FC26  07                       FCB    SPACE           FUNCTION
FC27  AE 64                    LDX    4,S             RELOAD FROM ADDRESS
FC29  C6 10                    LDB    #16             COUNT
FC2B  A6 80            HSHCHR  LDA    ,X+             NEXT BYTE
FC2D  2B 04                    BMI    HSHDOT          TOO LARGE, TO A DOT
FC2F  81 20                    CMPA   #' '            ? LOWER THAN A BLANK
FC31  24 02                    BHS    HSHCOK          NO, BRANCH OK
FC33  86 2E            HSHDOT  LDA    #'.'            CONVERT INVALID TO A BLANK
FC35  3F               HSHCOK  SWI                    SEND CHARACTER
FC36  01                       FCB    OUTCH           FUNCTION
FC37  5A                       DECB                   ? DONE
FC38  26 F1                    BNE    HSHCHR          BRANCH NO
FC3A  AC 62                    CMPX   2,S             ? PAST LAST ADDRESS
FC3C  24 09                    BHS    HSDRTN          QUIT IF SO
FC3E  AF 64                    STX    4,S             UPDATE FROM ADDRESS
FC40  A6 65                    LDA    5,S             LOAD LOW BYTE ADDRESS
FC42  48                       ASLA   ?               TO SECTION BOUNDRY
FC43  26 CF                    BNE    HSHLNE          BRANCH IF NOT
FC45  20 B5                    BRA    HSDTA           BRANCH IF SO
FC47  3F               HSDRTN  SWI                    SEND NEW LINE
FC48  06                       FCB    PCRLF           FUNCTION
FC49  39                       RTS                    RETURN TO CALLER
FC4A                   *F
FC4A
FC4A                   ***********************************************
FC4A                   *    A S S I S T 0 9   C O M M A N D S
FC4A                   ***********************************************
FC4A
FC4A                   *************REGISTERS - DISPLAY AND CHANGE REGISTERS
FC4A  8D 23            CREG    BSR    REGPRT          PRINT REGISTERS
FC4C  4C                       INCA                   SET FOR CHANGE FUNCTION
FC4D  8D 21                    BSR    REGCHG          GO CHANGE, DISPLAY REGISTERS
FC4F  39                       RTS                    RETURN TO COMMAND PROCESSOR
FC50
FC50                   ********************************************
FC50                   *     REGPRT - PRINT/CHANGE REGISTERS SUBROUTINE
FC50                   *  WILL ABORT TO 'CMDBAD' IF OVERFLOW DETECTED DURING
FC50                   *  A CHANGE OPERATION.  CHANGE DISPLAYS REGISTERS WHEN
FC50                   *  DONE.
```

```
FC50                  * REGISTER MASK LIST CONSISTS OF:
FC50                  *  A) CHARACTERS DENOTING REGISTER
FC50                  *  B) ZERO FOR ONE BYTE, -1 FOR TWO
FC50                  *  C) OFFSET ON STACK TO REGISTER POSITION
FC50                  * INPUT: SP+4=STACKED REGISTERS
FC50                  *        A=0 PRINT, A#0 PRINT AND CHANGE
FC50                  * OUTPUT: (ONLY FOR REGISTER DISPLAY)
FC50                  *         C=1 CONTROL-X ENTERED, C=0 OTHERWISE
FC50                  * VOLATILE: D,X (CHANGE)
FC50                  *           B,X (DISPLAY)
FC50                  *******************************************
FC50  50 43 FF 13     REGMSK  FCB    'P','C',-1,19   PC REG
FC54  41 00 0A                FCB    'A',0,10 A REG
FC57  42 00 0B                FCB    'B',0,11 B REG
FC5A  58 FF 0D                FCB    'X',-1,13 X REG
FC5D  59 FF 0F                FCB    'Y',-1,15 Y REG
FC60  55 FF 11                FCB    'U',-1,17 U REG
FC63  53 FF 01                FCB    'S',-1,1 S REG
FC66  43 43 00 09             FCB    'C','C',0,9     CC REG
FC6A  44 50 00 0C             FCB    'D','P',0,12    DP REG
FC6E  00                      FCB    0               END OF LIST
FC6F
FC6F  4F              REGPRT  CLRA                   SETUP PRINT ONLY FLAG
FC70  30 E8 10        REGCHG  LEAX   4+12,S          READY STACK VALUE
FC73  34 32                   PSHS   Y,X,A           SAVE ON STACK WITH OPTION
FC75  31 8C D8                LEAY   REGMSK,PCR      LOAD REGISTER MASK
FC78  EC A0           REGP1   LDD    ,Y+             LOAD NEXT CHAR OR <=0
FC7A  4D                      TSTA   ?               END OF CHARACTERS
FC7B  2F 04                   BLE    REGP2           BRANCH NOT CHARACTER
FC7D  3F                      SWI                    SEND TO CONSOLE
FC7E  01                      FCB    OUTCH           FUNCTION BYTE
FC7F  20 F7                   BRA    REGP1           CHECK NEXT
FC81  86 2D           REGP2   LDA    #'-'            READY '-'
FC83  3F                      SWI                    SEND OUT
FC84  01                      FCB    OUTCH           WITH OUTCH
FC85  30 E5                   LEAX   B,S             X->REGISTER TO PRINT
FC87  6D E4                   TST    ,S              ? CHANGE OPTION
FC89  26 12                   BNE    REGCNG          BRANCH YES
FC8B  6D 3F                   TST    -1,Y            ? ONE OR TWO BYTES
FC8D  27 03                   BEQ    REGP3           BRANCH ZERO MEANS ONE
FC8F  3F                      SWI                    PERFORM WORD HEX
FC90  05                      FCB    OUT4HS          FUNCTION
FC91  8C                      FCB    SKIP2           SKIP BYTE PRINT
FC92  3F              REGP3   SWI                    PERFORM BYTE HEX
FC93  04                      FCB    OUT2HS          FUNCTION
FC94  EC A0           REG4    LDD    ,Y+             TO FRONT OF NEXT ENTRY
FC96  5D                      TSTB   ?               END OF ENTRIES
FC97  26 DF                   BNE    REGP1           LOOP IF MORE
FC99  3F                      SWI                    FORCE NEW LINE
FC9A  06                      FCB    PCRLF           FUNCTION
FC9B  35 B2           REGRTN  PULS   PC,Y,X,A        RESTORE STACK AND RETURN
FC9D
FC9D  8D 40           REGCNG  BSR    BLDNNB          INPUT BINARY NUMBER
FC9F  27 10                   BEQ    REGNXC          IF CHANGE THEN JUMP
FCA1  81 0D                   CMPA   #CR             ? NO MORE DESIRED
FCA3  27 1E                   BEQ    REGAGN          BRANCH NOPE
FCA5  E6 3F                   LDB    -1,Y            LOAD SIZE FLAG
FCA7  5A                      DECB                   MINUS ONE
FCA8  50                      NEGB                   MAKE POSITIVE
FCA9  58                      ASLB   TIMES           TWO (=2 OR =4)
FCAA  3F              REGSKP  SWI                    PERFORM SPACES
FCAB  07                      FCB    SPACE           FUNCTION
FCAC  5A                      DECB
FCAD  26 FB                   BNE    REGSKP          LOOP IF MORE
FCAF  20 E3                   BRA    REG4            CONTINUE WITH NEXT REGISTER
FCB1  A7 E4           REGNXC  STA    ,S              SAVE DELIMITER IN OPTION
FCB3                  *                  (ALWAYS > 0)
FCB3  DC 9B                   LDD    NUMBER          OBTAIN BINARY RESULT
```

```
FCB5  6D 3F                     TST     -1,Y            ? TWO BYTES WORTH
FCB7  26 02                     BNE     REGTWO          BRANCH YES
FCB9  A6 82                     LDA     ,-X             SETUP FOR TWO
FCBB  ED 84             REGTWO  STD     ,X              STORE IN NEW VALUE
FCBD  A6 E4                     LDA     ,S              RECOVER DELIMITER
FCBF  81 0D                     CMPA    #CR             ? END OF CHANGES
FCC1  26 D1                     BNE     REG4            NO, KEEP ON TRUCK'N
FCC3                    * MOVE STACKED DATA TO NEW STACK IN CASE STACK
FCC3                    * POINTER HAS CHANGED
FCC3  30 8D E2 8A       REGAGN  LEAX    TSTACK,PCR      LOAD TEMP AREA
FCC7  C6 15                     LDB     #21             LOAD COUNT
FCC9  35 02             REGTF1  PULS    A               NEXT BYTE
FCCB  A7 80                     STA     ,X+             STORE INTO TEMP
FCCD  5A                        DECB                    COUNT DOWN
FCCE  26 F9                     BNE     REGTF1          LOOP IF MORE
FCD0  10 EE 88 EC               LDS     -20,X           LOAD NEW STACK POINTER
FCD4  C6 15                     LDB     #21             LOAD COUNT AGAIN
FCD6  A6 82             REGTF2  LDA     ,-X             NEXT TO STORE
FCD8  34 02                     PSHS    A               BACK ONTO NEW STACK
FCDA  5A                        DECB                    COUNT DOWN
FCDB  26 F9                     BNE     REGTF2          LOOP IF MORE
FCDD  20 BC                     BRA     REGRTN          GO RESTART COMMAND
FCDF
FCDF                    ********************************************
FCDF                    *  BLDNUM - BUILDS BINARY VALUE FROM INPUT HEX
FCDF                    *  THE ACTIVE EXPRESSION HANDLER IS USED.
FCDF                    * INPUT: S=RETURN ADDRESS
FCDF                    * OUTPUT: A=DELIMITER WHICH TERMINATED VALUE
FCDF                    *                            (IF DELM NOT ZERO)
FCDF                    *        "NUMBER"=WORD BINARY RESULT
FCDF                    *        Z=1 IF INPUT RECIEVED, Z=0 IF NO HEX RECIEVED
FCDF                    *  REGISTERS ARE TRANSPARENT
FCDF                    *********************************************
FCDF
FCDF                    * EXECUTE SINGLE OR EXTENDED ROM EXPRESSION HANDLER
FCDF                    *
FCDF                    * THE FLAG "DELIM" IS USED AS FOLLOWS:
FCDF                    *   DELIM=0  NO LEADING BLANKS, NO FORCED TERMINATOR
FCDF                    *   DELIM=CHR  ACCEPT LEADING 'CHR'S, FORCED TERMINATOR
FCDF  4F                BLDNNB  CLRA                    NO DYNAMIC DELIMITER
FCE0  8C                        FCB     SKIP2           SKIP NEXT INSTRUCTION
FCE1                    * BUILD WITH LEADING BLANKS
FCE1  86 20             BLDNUM  LDA     #' '            ALLOW LEADING BLANKS
FCE3  97 8E                     STA     DELIM           STORE AS DELIMITER
FCE5  6E 9D E3 03               JMP     [VECTAB+.EXPAN,PCR]     TO EXP ANALYZER
FCE9
FCE9                    * THIS IS THE DEFAULT SINGLE ROM ANALYZER. WE ACCEPT:
FCE9                    *    1) HEX INPUT
FCE9                    *    2) 'M' FOR LAST MEMORY EXAMINE ADDRESS
FCE9                    *    3) 'P' FOR PROGRAM COUNTER ADDRESS
FCE9                    *    4) 'W' FOR WINDOW VALUE
FCE9                    *    5) '@' FOR INDIRECT VALUE
FCE9  34 14             EXP1    PSHS    X,B             SAVE REGISTERS
FCEB  8D 5C             EXPDLM  BSR     BLDHXI          CLEAR NUMBER, CHECK FIRST CHAR
FCED  27 18                     BEQ     EXP2            IF HEX DIGIT CONTINUE BUILDING
FCEF                    * SKIP BLANKS IF DESIRED
FCEF  91 8E                     CMPA    DELIM           ? CORRECT DELIMITER
FCF1  27 F8                     BEQ     EXPDLM          YES, IGNORE IT
FCF3                    * TEST FOR M OR P
FCF3  9E 9E                     LDX     ADDR            DEFAULT FOR 'M'
FCF5  81 4D                     CMPA    #'M'            ? MEMORY EXAMINE ADDR WANTED
FCF7  27 16                     BEQ     EXPTDL          BRANCH IF SO
FCF9  9E 93                     LDX     PCNTER          DEFAULT FOR 'P'
FCFB  81 50                     CMPA    #'P'            ? LAST PROGRAM COUNTER WANTED
FCFD  27 10                     BEQ     EXPTDL          BRANCH IF SO
FCFF  9E A0                     LDX     WINDOW          DEFAULT TO WINDOW
FD01  81 57                     CMPA    #'W'            ? WINDOW WANTED
FD03  27 0A                     BEQ     EXPTDL
```

```
FD05  35 94      EXPRTN  PULS    PC,X,B          RETURN AND RESTORE REGISTERS
FD07             * GOT HEX, NOW CONTINUE BUILDING
FD07  8D 44      EXP2    BSR     BLDHEX          COMPUTE NEXT DIGIT
FD09  27 FC              BEQ     EXP2            CONTINUE IF MORE
FD0B  20 0A              BRA     EXPCDL          SEARCH FOR +/-
FD0D             * STORE VALUE AND CHECK IF NEED DELIMITER
FD0D  AE 84      EXPTDI  LDX     ,X              INDIRECTION DESIRED
FD0F  9F 9B      EXPTDL  STX     NUMBER          STORE RESULT
FD11  0D 8E              TST     DELIM           ? TO FORCE A DELIMITER
FD13  27 F0              BEQ     EXPRTN          RETURN IF NOT WITH VALUE
FD15  8D 62              BSR     READ            OBTAIN NEXT CHARACTER
FD17             * TEST FOR + OR -
FD17  9E 9B      EXPCDL  LDX     NUMBER          LOAD LAST VALUE
FD19  81 2B              CMPA    #'+'            ? ADD OPERATOR
FD1B  26 0E              BNE     EXPCHM          BRANCH NOT
FD1D  8D 23              BSR     EXPTRM          COMPUTE NEXT TERM
FD1F  34 02              PSHS    A               SAVE DELIMITER
FD21  DC 9B              LDD     NUMBER          LOAD NEW TERM
FD23  30 8B      EXPADD  LEAX    D,X             ADD TO X
FD25  9F 9B              STX     NUMBER          STORE AS NEW RESULT
FD27  35 02              PULS    A               RESTORE DELIMITER
FD29  20 EC              BRA     EXPCDL          NOW TEST IT
FD2B  81 2D      EXPCHM  CMPA    #'-'            ? SUBTRACT OPERATOR
FD2D  27 07              BEQ     EXPSUB          BRANCH IF SO
FD2F  81 40              CMPA    #'@'            ? INDIRECTION DESIRED
FD31  27 DA              BEQ     EXPTDI          BRANCH IF SO
FD33  5F                 CLRB                    SET DELIMITER RETURN
FD34  20 CF              BRA     EXPRTN          AND RETURN TO CALLER
FD36  8D 0A      EXPSUB  BSR     EXPTRM          OBTAIN NEXT TERM
FD38  34 02              PSHS    A               SAVE DELIMITER
FD3A  DC 9B              LDD     NUMBER          LOAD UP NEXT TERM
FD3C  40                 NEGA                    NEGATE A
FD3D  50                 NEGB                    NEGATE B
FD3E  82 00              SBCA    #0              CORRECT FOR A
FD40  20 E1              BRA     EXPADD          GO ADD TO EXPRESION
FD42             * COMPUTE NEXT EXPRESSION TERM
FD42             * OUTPUT: X=OLD VALUE
FD42             *         'NUMBER'=NEXT TERM
FD42  8D 9D      EXPTRM  BSR     BLDNUM          OBTAIN NEXT VALUE
FD44  27 32              BEQ     CNVRTS          RETURN IF VALID NUMBER
FD46  16 FC 13   BLDBAD  LBRA    CMDBAD          ABORT COMMAND IF INVALID
FD49
FD49             *********************************************
FD49             *  BUILD BINARY VALUE USING INPUT CHARACTERS.
FD49             * INPUT: A=ASCII HEX VALUE OR DELIMITER
FD49             *        SP+0=RETURN ADDRESS
FD49             *        SP+2=16 BIT RESULT AREA
FD49             * OUTPUT: Z=1 A=BINARY VALUE
FD49             *          Z=0 IF INVALID HEX CHARACTER (A UNCHANGED)
FD49             * VOLATILE: D
FD49             ****************************************
FD49  0F 9B      BLDHXI  CLR     NUMBER          CLEAR NUMBER
FD4B  0F 9C              CLR     NUMBER+1        CLEAR NUMBER
FD4D  8D 2A      BLDHEX  BSR     READ            GET INPUT CHARACTER
FD4F  8D 11      BLDHXC  BSR     CNVHEX          CONVERT AND TEST CHARACTER
FD51  26 25              BNE     CNVRTS          RETURN IF NOT A NUMBER
FD53  C6 10              LDB     #16             PREPARE SHIFT
FD55  3D                 MUL                     BY FOUR PLACES
FD56  86 04              LDA     #4              ROTATE BINARY INTO VALUE
FD58  58         BLDSHF  ASLB    OBTAIN          NEXT BIT
FD59  09 9C              ROL     NUMBER+1        INTO LOW BYTE
FD5B  09 9B              ROL     NUMBER          INTO HI BYTE
FD5D  4A                 DECA                    COUNT DOWN
FD5E  26 F8              BNE     BLDSHF          BRANCH IF MORE TO DO
FD60  20 14              BRA     CNVOK           SET GOOD RETURN CODE
FD62
FD62             ****************************************
FD62             * CONVERT ASCII CHARACTER TO BINARY BYTE
```

```
FD62                 * INPUT: A=ASCII
FD62                 * OUTPUT: Z=1 A=BINARY VALUE
FD62                 *         Z=0 IF INVALID
FD62                 * ALL REGISTERS TRANSPARENT
FD62                 * (A UNALTERED IF INVALID HEX)
FD62                 **************************************
FD62  81 30          CNVHEX  CMPA    #'0'            ? LOWER THAN A ZERO
FD64  25 12                  BLO     CNVRTS          BRANCH NOT VALUE
FD66  81 39                  CMPA    #'9'            ? POSSIBLE A-F
FD68  2F 0A                  BLE     CNVGOT          BRANCH NO TO ACCEPT
FD6A  81 41                  CMPA    #'A'            ? LESS THEN TEN
FD6C  25 0A                  BLO     CNVRTS          RETURN IF MINUS (INVALID)
FD6E  81 46                  CMPA    #'F'            ? NOT TOO LARGE
FD70  22 06                  BHI     CNVRTS          NO, RETURN TOO LARGE
FD72  80 07                  SUBA    #7              DOWN TO BINARY
FD74  84 0F          CNVGOT  ANDA    #$0F            CLEAR HIGH HEX
FD76  1A 04          CNVOK   ORCC    #4              FORCE ZERO ON FOR VALID HEX
FD78  39             CNVRTS  RTS                     RETURN TO CALLER
FD79
FD79                 * GET INPUT CHAR, ABORT COMMAND IF CONTROL-X (CANCEL)
FD79  3F             READ    SWI                     GET NEXT CHARACTER
FD7A  00                     FCB     INCHNP          FUNCTION
FD7B  81 18                  CMPA    #CAN            ? ABORT COMMAND
FD7D  27 C7                  BEQ     BLDBAD          BRANCH TO ABORT IF SO
FD7F  39                     RTS                     RETURN TO CALLER
FD80                 *G
FD80
FD80                 ***************GO - START PROGRAM EXECUTION
FD80  8D 01          CGO     BSR     GOADDR          BUILD ADDRESS IF NEEDED
FD82  3B                     RTI     START           EXECUTING
FD83
FD83                 * FIND OPTIONAL NEW PROGRAM COUNTER. ALSO ARM THE
FD83                 * BREAKPOINTS.
FD83  35 30          GOADDR  PULS    Y,X             RECOVER RETURN ADDRESS
FD85  34 10                  PSHS    X               STORE RETURN BACK
FD87  26 19                  BNE     GONDFT          IF NO CARRIAGE RETURN THEN NEW PC
FD89                 * DEFAULT PROGRAM COUNTER, SO FALL THROUGH IF
FD89                 * IMMEDIATE BREAKPOINT.
FD89  17 01 B6               LBSR    CBKLDR          SEARCH BREAKPOINTS
FD8C  AE 6C                  LDX     12,S            LOAD PROGRAM COUNTER
FD8E  5A             ARMBLP  DECB                    COUNT DOWN
FD8F  2B 16                  BMI     ARMBK2          DONE, NONE TO SINGLE TRACE
FD91  A6 30                  LDA     -NUMBKP*2,Y     PRE-FETCH OPCODE
FD93  AC A1                  CMPX    ,Y++            ? IS THIS A BREAKPOINT
FD95  26 F7                  BNE     ARMBLP          LOOP IF NOT
FD97  81 3F                  CMPA    #$3F            ? SWI BREAKPOINTED
FD99  26 02                  BNE     ARMNSW          NO, SKIP SETTING OF PASS FLAG
FD9B  97 FB                  STA     SWIBFL          SHOW UPCOMMING SWI NOT BRKPNT
FD9D  0C 8F          ARMNSW  INC     MISFLG          FLAG THRU A BREAKPOINT
FD9F  16 01 06               LBRA    CDOT            DO SINGLE TRACE W/O BREAKPOINTS
FDA2                 * OBTAIN NEW PROGRAM COUNTER
FDA2  17 00 BB       GONDFT  LBSR    CDNUM           OBTAIN NEW PROGRAM COUNTER
FDA5  ED 6C                  STD     12,S            STORE INTO STACK
FDA7  17 01 98       ARMBK2  LBSR    CBKLDR          OBTAIN TABLE
FDAA  00 FA                  NEG     BKPTCT          COMPLEMENT TO SHOW ARMED
FDAC  5A             ARMLOP  DECB                    ? DONE
FDAD  2B C9                  BMI     CNVRTS          RETURN WHEN DONE
FDAF  A6 B4                  LDA     [,Y]            LOAD OPCODE
FDB1  A7 30                  STA     -NUMBKP*2,Y     STORE INTO OPCODE TABLE
FDB3  86 3F                  LDA     #$3F            READY "SWI" OPCODE
FDB5  A7 B1                  STA     [,Y++]          STORE AND MOVE UP TABLE
FDB7  20 F3                  BRA     ARMLOP          AND CONTINUE
FDB9
FDB9                 ******************CALL - CALL ADDRESS AS SUBROUTINE
FDB9  8D C8          CCALL   BSR     GOADDR          FETCH ADDRESS IF NEEDED
FDBB  35 7F                  PULS    U,Y,X,DP,D,CC   RESTORE USERS REGISTERS
FDBD  AD F1                  JSR     [,S++]          CALL USER SUBROUTINE
FDBF  3F             CGOBRK  SWI                     PERFORM BREAKPOINT
```

```
FDC0  0A                      FCB     BRKPT           FUNCTION
FDC1  20 FC                   BRA     CGOBRK          LOOP UNTIL USER CHANGES PC
FDC3
FDC3                  ***************MEMORY - DISPLAY/CHANGE MEMORY
FDC3                  * CMEMN AND CMPADP ARE DIRECT ENTRY POINTS FROM
FDC3                  * THE COMMAND HANDLER FOR QUICK COMMANDS
FDC3  17 00 9A        CMEM    LBSR    CDNUM           OBTAIN ADDRESS
FDC6  DD 9E           CMEMN   STD     ADDR            STORE DEFAULT
FDC8  9E 9E           CMEM2   LDX     ADDR            LOAD POINTER
FDCA  17 FC 0C                LBSR    ZOUT2H          SEND OUT HEX VALUE OF BYTE
FDCD  86 2D                   LDA     #'-'            LOAD DELIMITER
FDCF  3F                      SWI                     SEND OUT
FDD0  01                      FCB     OUTCH           FUNCTION
FDD1  17 FF 0B        CMEM4   LBSR    BLDNNB          OBTAIN NEW BYTE VALUE
FDD4  27 0A                   BEQ     CMENUM          BRANCH IF NUMBER
FDD6                  * COMA - SKIP BYTE
FDD6  81 2C                   CMPA    #','            ? COMMA
FDD8  26 0E                   BNE     CMNOTC          BRANCH NOT
FDDA  9F 9E                   STX     ADDR            UPDATE POINTER
FDDC  30 01                   LEAX    1,X             TO NEXT BYTE
FDDE  20 F1                   BRA     CMEM4           AND INPUT IT
FDE0  D6 9C           CMENUM  LDB     NUMBER+1        LOAD LOW BYTE VALUE
FDE2  8D 47                   BSR     MUPDAT          GO OVERLAY MEMORY BYTE
FDE4  81 2C                   CMPA    #','            ? CONTINUE WITH NO DISPLAY
FDE6  27 E9                   BEQ     CMEM4           BRANCH YES
FDE8                  * QUOTED STRING
FDE8  81 27           CMNOTC  CMPA    #$27            ? QUOTED STRING
FDEA  26 0C                   BNE     CMNOTQ          BRANCH NO
FDEC  8D 8B           CMESTR  BSR     READ            OBTAIN NEXT CHARACTER
FDEE  81 27                   CMPA    #$27            ? END OF QUOTED STRING
FDF0  27 0C                   BEQ     CMSPCE          YES, QUIT STRING MODE
FDF2  1F 89                   TFR     A,B             TO B FOR SUBROUTINE
FDF4  8D 35                   BSR     MUPDAT          GO UPDATE BYTE
FDF6  20 F4                   BRA     CMESTR          GET NEXT CHARACTER
FDF8                  * BLANK - NEXT BYTE
FDF8  81 20           CMNOTQ  CMPA    #$20            ? BLANK FOR NEXT BYTE
FDFA  26 06                   BNE     CMNOTB          BRANCH NOT
FDFC  9F 9E                   STX     ADDR            UPDATE POINTER
FDFE  3F              CMSPCE  SWI                     GIVE SPACE
FDFF  07                      FCB     SPACE           FUNCTION
FE00  20 C6                   BRA     CMEM2           NOW PROMPT FOR NEXT
FE02                  * LINE FEED - NEXT BYTE WITH ADDRESS
FE02  81 0A           CMNOTB  CMPA    #LF             ? LINE FEED FOR NEXT BYTE
FE04  26 08                   BNE     CMNOTL          BRANCH NO
FE06  86 0D                   LDA     #CR             GIVE CARRIAGE RETURN
FE08  3F                      SWI                     TO CONSOLE
FE09  01                      FCB     OUTCH           HANDLER
FE0A  9F 9E                   STX     ADDR            STORE NEXT ADDRESS
FE0C  20 0A                   BRA     CMPADP          BRANCH TO SHOW
FE0E                  * UP ARROW - PREVIOUS BYTE AND ADDRESS
FE0E  81 5E           CMNOTL  CMPA    #'^'            ? UP ARROW FOR PREVIOUS BYTE
FE10  26 0A                   BNE     CMNOTU          BRANCH NOT
FE12  30 1E                   LEAX    -2,X            DOWN TO PREVIOUS BYTE
FE14  9F 9E                   STX     ADDR            STORE NEW POINTER
FE16  3F              CMPADS  SWI                     FORCE NEW LINE
FE17  06                      FCB     PCRLF           FUNCTION
FE18  8D 07           CMPADP  BSR     PRTADR          GO PRINT ITS VALUE
FE1A  20 AC                   BRA     CMEM2           THEN PROMPT FOR INPUT
FE1C                  * SLASH - NEXT BYTE WITH ADDRESS
FE1C  81 2F           CMNOTU  CMPA    #'/'            ? SLASH FOR CURRENT DISPLAY
FE1E  27 F6                   BEQ     CMPADS          YES, SEND ADDRESS
FE20  39                      RTS                     RETURN FROM COMMAND
FE21
FE21                  * PRINT CURRENT ADDRESS
FE21  9E 9E           PRTADR  LDX     ADDR            LOAD POINTER VALUE
FE23  34 10                   PSHS    X               SAVE X ON STACK
FE25  30 E4                   LEAX    ,S              POINT TO IT FOR DISPLAY
FE27  3F                      SWI                     DISPLAY POINTER IN HEX
```

```
FE28  05                         FCB     OUT4HS          FUNCTION
FE29  35 90                      PULS    PC,X            RECOVER POINTER AND RETURN
FE2B
FE2B                   * UPDATE BYTE
FE2B  9E 9E            MUPDAT    LDX     ADDR            LOAD NEXT BYTE POINTER
FE2D  E7 80                      STB     ,X+             STORE AND INCREMENT X
FE2F  E1 1F                      CMPB    -1,X            ? SUCCESFULL STORE
FE31  26 03                      BNE     MUPBAD          BRANCH FOR '?' IF NOT
FE33  9F 9E                      STX     ADDR            STORE NEW POINTER VALUE
FE35  39                         RTS                     BACK TO CALLER
FE36  34 02            MUPBAD    PSHS    A               SAVE A REGISTER
FE38  86 3F                      LDA     #'?'            SHOW INVALID
FE3A  3F                         SWI                     SEND OUT
FE3B  01                         FCB     OUTCH           FUNCTION
FE3C  35 82                      PULS    PC,A            RETURN TO CALLER
FE3E
FE3E                   ********************WINDOW  -  SET WINDOW VALUE
FE3E  8D 20            CWINDO    BSR     CDNUM           OBTAIN WINDOW VALUE
FE40  DD A0                      STD     WINDOW          STORE IT IN
FE42  39                         RTS                     END COMMAND
FE43
FE43                   ******************DISPLAY - HIGH SPEED DISPLAY MEMORY
FE43  8D 1B            CDISP     BSR     CDNUM           FETCH ADDRESS
FE45  C4 F0                      ANDB    #$F0            FORCE TO 16 BOUNDRY
FE47  1F 02                      TFR     D,Y             SAVE IN Y
FE49  30 2F                      LEAX    15,Y            DEFAULT LENGTH
FE4B  25 04                      BCS     CDISPS          BRANCH IF END OF INPUT
FE4D  8D 11                      BSR     CDNUM           OBTAIN COUNT
FE4F  30 AB                      LEAX    D,Y             ASSUME COUNT, COMPUTE END ADDR
FE51  34 30            CDISPS    PSHS    Y,X             SETUP PARAMETERS FOR HSDATA
FE53  10 A3 62                   CMPD    2,S             ? WAS IT COUNT
FE56  23 02                      BLS     CDCNT           BRANCH YES
FE58  ED E4                      STD     ,S              STORE HIGH ADDRESS
FE5A  AD 9D E1 84      CDCNT     JSR     [VECTAB+.HSDTA,PCR]     CALL PRINT ROUTINE
FE5E  35 E0                      PULS    PC,U,Y          CLEAN STACK AND END COMMAND
FE60
FE60                   * OBTAIN NUMBER - ABORT IF NONE
FE60                   * ONLY DELIMITERS OF CR, BLANK, OR '/' ARE ACCEPTED
FE60                   * OUTPUT: D=VALUE, C=1 IF CARRIAGE RETURN DELMITER,
FE60                   *                                    ELSE C=0
FE60  17 FE 7E         CDNUM     LBSR    BLDNUM          OBTAIN NUMBER
FE63  26 09                      BNE     CDBADN          BRANCH IF INVALID
FE65  81 2F                      CMPA    #'/'            ? VALID DELIMITER
FE67  22 05                      BHI     CDBADN          BRANCH IF NOT FOR ERROR
FE69  81 0E                      CMPA    #CR+1           LEAVE COMPARE FOR CARRIAGE RET
FE6B  DC 9B                      LDD     NUMBER          LOAD NUMBER
FE6D  39                         RTS                     RETURN WITH COMPARE
FE6E  16 FA EB         CDBADN    LBRA    CMDBAD          RETURN TO ERROR MECHANISM
FE71
FE71                   *****************PUNCH - PUNCH MEMORY IN S1-S9 FORMAT
FE71  8D ED            CPUNCH    BSR     CDNUM           OBTAIN START ADDRESS
FE73  1F 02                      TFR     D,Y             SAVE IN Y
FE75  8D E9                      BSR     CDNUM           OBTAIN END ADDRESS
FE77  6F E2                      CLR     ,-S             SETUP PUNCH FUNCTION CODE
FE79  34 26                      PSHS    Y,D             STORE VALUES ON STACK
FE7B  AD 9D E1 65      CCALBS    JSR     [VECTAB+.BSON,PCR]      INITIALIZE HANDLER
FE7F  AD 9D E1 63                JSR     [VECTAB+.BSDTA,PCR]     PERFORM FUNCTION
FE83  34 01                      PSHS    CC              SAVE RETURN CODE
FE85  AD 9D E1 5F                JSR     [VECTAB+.BSOFF,PCR]     TURN OFF HANDLER
FE89  35 01                      PULS    CC              OBTAIN CONDITION CODE SAVED
FE8B  26 E1                      BNE     CDBADN          BRANCH IF ERROR
FE8D  35 B2                      PULS    PC,Y,X,A        RETURN FROM COMMAND
FE8F
FE8F                   *****************LOAD - LOAD MEMORY FROM S1-S9 FORMAT
FE8F  8D 01            CLOAD     BSR     CLVOFS          CALL SETUP AND PASS CODE
FE91  01                         FCB     1               LOAD FUNCTION CODE FOR PACKET
FE92
FE92  33 F1            CLVOFS    LEAU    [,S++]          LOAD CODE IN HIGH BYTE OF U
```

```
FE94  33 D4                       LEAU    [,U]             NOT CHANGING CC AND RESTORE S
FE96  27 03                       BEQ     CLVDFT           BRANCH IF CARRIAGE RETURN NEXT
FE98  8D C6                       BSR     CDNUM            OBTAIN OFFSET
FE9A  8C                          FCB     SKIP2            SKIP DEFAULT OFFSET
FE9B  4F                  CLVDFT  CLRA                     CREATE ZERO OFFSET
FE9C  5F                          CLRB                     AS DEFAULT
FE9D  34 4E                       PSHS    U,DP,D           SETUP CODE, NULL WORD, OFFSET
FE9F  20 DA                       BRA     CCALBS           ENTER CALL TO BS ROUTINES
FEA1
FEA1                      ******************VERIFY - COMPARE MEMORY WITH FILES
FEA1  8D EF               CVER    BSR     CLVOFS           COMPUTE OFFSET IF ANY
FEA3  FF                          FCB     -1               VERIFY FNCTN CODE FOR PACKET
FEA4
FEA4                      *******************TRACE - TRACE INSTRUCTIONS
FEA4                      ******************* . - SINGLE STEP TRACE
FEA4  8D BA               CTRACE  BSR     CDNUM            OBTAIN TRACE COUNT
FEA6  DD 91                       STD     TRACEC           STORE COUNT
FEA8  32 62               CDOT    LEAS    2,S              RID COMMAND RETURN FROM STACK
FEAA  EE F8 0A            CTRCE3  LDU     [10,S]           LOAD OPCODE TO EXECUTE
FEAD  DF 99                       STU     LASTOP           STORE FOR TRACE INTERRUPT
FEAF  DE F6                       LDU     VECTAB+.PTM      LOAD PTM ADDRESS
FEB1  CC 07 01                    LDD     #$0701           7,1 CYCLES DOWN+CYCLES UP
FEB4  ED 42                       STD     PTMTM1-PTM,U     START NMI TIMEOUT
FEB6  3B                          RTI     RETURN           FOR ONE INSTRUCTION
FEB7
FEB7                      *************NULLS  -  SET NEW LINE AND CHAR PADDING
FEB7  8D A7               CNULLS  BSR     CDNUM            OBTAIN NEW LINE PAD
FEB9  DD F2                       STD     VECTAB+.PAD      RESET VALUES
FEBB  39                          RTS                      END COMMAND
FEBC
FEBC                      ******************STLEVEL - SET STACK TRACE LEVEL
FEBC  27 05               CSTLEV  BEQ     STLDFT           TAKE DEFAULT
FEBE  8D A0                       BSR     CDNUM            OBTAIN NEW STACK LEVEL
FEC0  DD F8                       STD     SLEVEL           STORE NEW ENTRY
FEC2  39                          RTS                      TO COMMAND HANDLER
FEC3  30 6E               STLDFT  LEAX    14,S             COMPUTE NMI COMPARE
FEC5  9F F8                       STX     SLEVEL           AND STORE IT
FEC7  39                          RTS                      END COMMAND
FEC8
FEC8                      ******************OFFSET - COMPUTE SHORT AND LONG
FEC8                      ******************                   BRANCH OFFSETS
FEC8  8D 96               COFFS   BSR     CDNUM            OBTAIN INSTRUCTION ADDRESS
FECA  1F 01                       TFR     D,X              USE AS FROM ADDRESS
FECC  8D 92                       BSR     CDNUM            OBTAIN TO ADDRESS
FECE                      * D=TO INSTRUCTION, X=FROM INSTRUCTION OFFSET BYTE(S)
FECE  30 01                       LEAX    1,X              ADJUST FOR *+2 SHORT BRANCH
FED0  34 30                       PSHS    Y,X              STORE WORK WORD AND VALUE ON S
FED2  A3 E4                       SUBD    ,S               FIND OFFSET
FED4  ED E4                       STD     ,S               SAVE OVER STACK
FED6  30 61                       LEAX    1,S              POINT FOR ONE BYTE DISPLAY
FED8  1D                          SEX     SIGN             EXTEND LOW BYTE
FED9  A1 E4                       CMPA    ,S               ? VALID ONE BYTE OFFSET
FEDB  26 02                       BNE     COFNO1           BRANCH IF NOT
FEDD  3F                          SWI                      SHOW ONE BYTE OFFSET
FEDE  04                          FCB     OUT2HS           FUNCTION
FEDF  EE E4               COFNO1  LDU     ,S               RELOAD OFFSET
FEE1  33 5F                       LEAU    -1,U             CONVERT TO LONG BRANCH OFFSET
FEE3  EF 84                       STU     ,X               STORE BACK WHERE X POINTS NOW
FEE5  3F                          SWI                      SHOW TWO BYTE OFFSET
FEE6  05                          FCB     OUT4HS           FUNCTION
FEE7  3F                          SWI                      FORCE NEW LINE
FEE8  06                          FCB     PCRLF            FUNCTION
FEE9  35 96                       PULS    PC,X,D           RESTORE STACK AND END COMMAND
FEEB                      *H
FEEB
FEEB                      *************BREAKPOINT - DISPLAY/ENTER/DELETE/CLEAR
FEEB                      *************                BREAKPOINTS
FEEB  27 23               CBKPT   BEQ     CBKDSP           BRANCH DISPLAY OF JUST 'B'
```

```
FEED  17 FD F1                   LBSR    BLDNUM          ATTEMPT VALUE ENTRY
FEF0  27 2C                      BEQ     CBKADD          BRANCH TO ADD IF SO
FEF2  81 2D                      CMPA    #'-'            ? CORRECT DELIMITER
FEF4  26 3F                      BNE     CBKERR          NO, BRANCH FOR ERROR
FEF6  17 FD E8                   LBSR    BLDNUM          ATTEMPT DELETE VALUE
FEF9  27 03                      BEQ     CBKDLE          GOT ONE, GO DELETE IT
FEFB  0F FA                      CLR     BKPTCT          WAS 'B -', SO ZERO COUNT
FEFD  39                 CBKRTS  RTS                     END COMMAND
FEFE                     * DELETE THE ENTRY
FEFE  8D 40              CBKDLE  BSR     CBKSET          SETUP REGISTERS AND VALUE
FF00  5A                 CBKDLP  DECB                    ? ANY ENTRIES IN TABLE
FF01  2B 32                      BMI     CBKERR          BRANCH NO, ERROR
FF03  AC A1                      CMPX    ,Y++            ? IS THIS THE ENTRY
FF05  26 F9                      BNE     CBKDLP          NO, TRY NEXT
FF07                     * FOUND, NOW MOVE OTHERS UP IN ITS PLACE
FF07  AE A1              CBKDLM  LDX     ,Y++            LOAD NEXT ONE UP
FF09  AF 3C                      STX     -4,Y            MOVE DOWN BY ONE
FF0B  5A                         DECB                    ? DONE
FF0C  2A F9                      BPL     CBKDLM          NO, CONTINUE MOVE
FF0E  0A FA                      DEC     BKPTCT          DECREMENT BREAKPOINT COUNT
FF10  8D 2E              CBKDSP  BSR     CBKSET          SETUP REGISTERS AND LOAD VALUE
FF12  27 E9                      BEQ     CBKRTS          RETURN IF NONE TO DISPLY
FF14  30 A1              CBKDSL  LEAX    ,Y++            POINT TO NEXT ENTRY
FF16  3F                         SWI                     DISPLAY IN HEX
FF17  05                         FCB     OUT4HS          FUNCTION
FF18  5A                         DECB                    COUNT DOWN
FF19  26 F9                      BNE     CBKDSL          LOOP IF NGABLE RAM
FF1B  3F                         SWI                     SKIP TO NEW LINK
FF1C  06                         FCB     PCRLF           FUNCTIONRTS
FF1D  39                         RTS
FF1E                     * ADD NEW ENTRY
FF1E  8D 20              CBKADD  BSR     CBKSET          SETUP REGISTERS
FF20  C1 08                      CMPB    #NUMBKP         ? ALREADY FULL
FF22  27 11                      BEQ     CBKERR          BRANCH ERROR IF SO
FF24  A6 84                      LDA     ,X              LOAD BYTE TO TRAP
FF26  E7 84                      STB     ,X              TRY TO CHANGE
FF28  E1 84                      CMPB    ,X              ? CHANGEABLE RAM
FF2A  26 09                      BNE     CBKERR          BRANCH ERROR IF NOT
FF2C  A7 84                      STA     ,X              RESTORE BYTE
FF2E  5A                 CBKADL  DECB                    COUNT DOWN
FF2F  2B 07                      BMI     CBKADT          BRANCH IF DONE TO ADD IT
FF31  AC A1                      CMPX    ,Y++            ? ENTRY ALREADY HERE
FF33  26 F9                      BNE     CBKADL          LOOP IF NOT
FF35  16 FA 24           CBKERR  LBRA    CMDBAD          RETURN TO ERROR PRODUCE
FF38  AF A4              CBKADT  STX     ,Y              ADD THIS ENTRY
FF3A  6F 31                      CLR     -NUMBKP*2+1,Y   CLEAR OPTIONAL BYTE
FF3C  0C FA                      INC     BKPTCT          ADD ONE TO COUNT
FF3E  20 D0                      BRA     CBKDSP          AND NOW DISPLAY ALL OF 'EM
FF40                     * SETUP REGISTERS FOR SCAN
FF40  9E 9B              CBKSET  LDX     NUMBER          LOAD VALUE DESIRED
FF42  31 8D E0 6C        CBKLDR  LEAY    BKPTBL,PCR      LOAD START OF TABLE
FF46  D6 FA                      LDB     BKPTCT          LOAD ENTRY COUNT
FF48  39                         RTS                     RETURN
FF49
FF49                     *****************ENCODE  -  ENCODE A POSTBYTE
FF49  6F E2              CENCDE  CLR     ,-S             DEFAULT TO NOT INDIRECT
FF4B  5F                         CLRB                    ZERO POSTBYTE VALUE
FF4C  30 8C 3F                   LEAX    <CONV1,PCR      START TABLE SEARCH
FF4F  3F                         SWI                     OBTAIN FIRST CHARACTER
FF50  00                         FCB     INCHNP          FUNCTION
FF51  81 5B                      CMPA    #'['            ? INDIRECT HERE
FF53  26 06                      BNE     CEN2            BRANCH IF NOT
FF55  86 10                      LDA     #$10            SET INDIRECT BIT ON
FF57  A7 E4                      STA     ,S              SAVE FOR LATER
FF59  3F                 CENGET  SWI                     OBTAIN NEXT CHARACTER
FF5A  00                         FCB     INCHNP          FUNCTION
FF5B  81 0D              CEN2    CMPA    #CR             ? END OF ENTRY
FF5D  27 0C                      BEQ     CEND1           BRANCH YES
```

```
FF5F  6D 84                     CENLP1  TST     ,X              ? END OF TABLE
FF61  2B D2                             BMI     CBKERR          BRANCH ERROR IF SO
FF63  A1 81                             CMPA    ,X++            ? THIS THE CHARACTER
FF65  26 F8                             BNE     CENLP1          BRANCH IF NOT
FF67  EB 1F                             ADDB    -1,X            ADD THIS VALUE
FF69  20 EE                             BRA     CENGET          GET NEXT INPUT
FF6B  30 8C 49                  CEND1   LEAX    <CONV2,PCR      POINT AT TABLE 2
FF6E  1F 98                             TFR     B,A             SAVE COPY IN A
FF70  84 60                             ANDA    #$60            ISOLATE REGISTER MASK
FF72  AA E4                             ORA     ,S              ADD IN INDIRECTION BIT
FF74  A7 E4                             STA     ,S              SAVE BACK AS POSTBYTE SKELETON
FF76  C4 9F                             ANDB    #$9F            CLEAR REGISTER BITS
FF78  6D 84                     CENLP2  TST     ,X              ? END OF TABLE
FF7A  27 B9                             BEQ     CBKERR          BRANCH ERROR IF SO
FF7C  E1 81                             CMPB    ,X++            ? SAME VALUE
FF7E  26 F8                             BNE     CENLP2          LOOP IF NOT
FF80  E6 1F                             LDB     -1,X            LOAD RESULT VALUE
FF82  EA E4                             ORB     ,S              ADD TO BASE SKELETON
FF84  E7 E4                             STB     ,S              SAVE POSTBYTE ON STACK
FF86  30 E4                             LEAX    ,S              POINT TO IT
FF88  3F                                SWI                     SEND OUT AS HEX
FF89  04                                FCB     OUT2HS          FUNCTION
FF8A  3F                                SWI                     TO NEXT LINE
FF8B  06                                FCB     PCRLF           FUNCTION
FF8C  35 84                             PULS    PC,B            END OF COMMAND
FF8E
FF8E                            * TABLE ONE DEFINES VALID INPUT IN SEQUENCE
FF8E  41 04 42 05 44 06 +       CONV1   FCB     'A',$04,'B',$05,'D',$06,'H',$01
FF96  48 01 48 01 48 00 +               FCB     'H',$01,'H',$01,'H',$00,',',$00
FF9E  2D 09 2D 01 53 70 +               FCB     '-',$09,'-',$01,'S',$70,'Y',$30
FFA6  55 50 58 10 2B 07 +               FCB     'U',$50,'X',$10,'+',$07,'+',$01
FFAE  50 80 43 00 52 00 +               FCB     'P',$80,'C',$00,'R',$00,']',$00
FFB6  FF                                FCB     $FF             END OF TABLE
FFB7                            *CONV2 USES ABOVE CONVERSION TO SET POSTBYTE
FFB7                            *                             BIT SKELETON.
FFB7  10 84 11 00               CONV2   FDB     $1084,$1100     R,        H,R
FFBB  12 88 13 89                       FDB     $1288,$1389     HH,R      HHHH,R
FFBF  14 86 15 85                       FDB     $1486,$1585     A,R       B,R
FFC3  16 8B 17 80                       FDB     $168B,$1780     D,R       ,R+
FFC7  18 81 19 82                       FDB     $1881,$1982     ,R++      ,-R
FFCB  1A 83 82 8C                       FDB     $1A83,$828C     ,--R      HH,PCR
FFCF  83 8D 03 9F                       FDB     $838D,$039F     HHHH,PCR  [HHHH]
FFD3  00                                FCB     0               END OF TABLE
FFD4
FFD4                            ****************************************************
FFD4                            *             DEFAULT INTERRUPT TRANSFERS          *
FFD4                            ****************************************************
FFD4  6E 9D DF EE               RSRVD   JMP     [VECTAB+.RSVD,PCR]      RESERVED VECTOR
FFD8  6E 9D DF EC               SWI3    JMP     [VECTAB+.SWI3,PCR]      SWI3 VECTOR
FFDC  6E 9D DF EA               SWI2    JMP     [VECTAB+.SWI2,PCR]      SWI2 VECTOR
FFE0  6E 9D DF E8               FIRQ    JMP     [VECTAB+.FIRQ,PCR]      FIRQ VECTOR
FFE4  6E 9D DF E6               IRQ     JMP     [VECTAB+.IRQ,PCR]       IRQ VECTOR
FFE8  6E 9D DF E4               SWI     JMP     [VECTAB+.SWI,PCR]       SWI VECTOR
FFEC  6E 9D DF E2               NMI     JMP     [VECTAB+.NMI,PCR]       NMI VECTOR
FFF0
FFF0                            *****************************************************
FFF0                            *             ASSIST09 HARDWARE VECTOR TABLE
FFF0                            *  THIS TABLE IS USED IF THE ASSIST09 ROM ADDRESSES
FFF0                            *  THE MC6809 HARDWARE VECTORS.
FFF0                            *****************************************************
FFF0                                    ORG     ROMBEG+ROMSIZ-16        SETUP HARDWARE VECTORS
FFF0  FF D4                             FDB     RSRVD           RESERVED SLOT
FFF2  FF D8                             FDB     SWI3            SOFTWARE INTERRUPT 3
FFF4  FF DC                             FDB     SWI2            SOFTWARE INTERRUPT 2
FFF6  FF E0                             FDB     FIRQ            FAST INTERRUPT REQUEST
FFF8  FF E4                             FDB     IRQ             INTERRUPT REQUEST
FFFA  FF E8                             FDB     SWI             SOFTWARE INTERRUPT
FFFC  FF EC                             FDB     NMI             NON-MASKABLE INTERRUPT
```

```
FFFE  F8 37               1721          FDB     RESET           RESTART
0000                      1722
```

SYMBOL TABLE:

```
.ACIA   -002E   .AVTBL  -0000   .BSDTA  -0024   .BSOFF  -0026   .BSON   -0022
.CIDTA  -0016   .CIOFF  -0018   .CION   -0014   .CMDL1  -0002   .CMDL2  -002C
.CODTA  -001C   .COOFF  -001E   .COON   -001A   .ECHO   -0032   .EXPAN  -002A
.FIRQ   -000A   .HSDTA  -0020   .IRQ    -000C   .NMI    -0010   .PAD    -0030
.PAUSE  -0028   .PTM    -0034   .RESET  -0012   .RSVD   -0004   .SWI    -000E
.SWI2   -0008   .SWI3   -0006   ACIA    -E008   ADDR    -DF9E   ARMBK2  -FDA7
ARMBLP  -FD8E   ARMLOP  -FDAC   ARMNSW  -FD9D   BASEPG  -DF9D   BELL    -0007
BKPTBL  -DFB2   BKPTCT  -DFFA   BKPTOP  -DFA2   BLD2    -F815   BLD3    -F821
BLDBAD  -FD46   BLDHEX  -FD4D   BLDHXC  -FD4F   BLDHXI  -FD49   BLDNNB  -FCDF
BLDNUM  -FCE1   BLDRTN  -F835   BLDSHF  -FD58   BLDVTR  -F800   BRKPT   -000A
BSDCMP  -FB6A   BSDEOL  -FB70   BSDLD1  -FB40   BSDLD2  -FB42   BSDNXT  -FB60
BSDPUN  -FB92   BSDSRT  -FB6E   BSDTA   -FB38   BSOFF   -FB27   BSOFLP  -FB33
BSON    -FB1B   BSON2   -FB22   BSPEOF  -FBEF   BSPGO   -FBA3   BSPMRE  -FBC6
BSPOK   -FBAF   BSPSTR  -FBEC   BSPUN2  -FBE7   BSPUNC  -FBE9   BYTE    -FB75
BYTHEX  -FB89   BYTRTS  -FB88   CAN     -0018   CBKADD  -FF1E   CBKADL  -FF2E
CBKADT  -FF38   CBKDLE  -FEFE   CBKDLM  -FF07   CBKDLP  -FF00   CBKDSL  -FF14
CBKDSP  -FF10   CBKERR  -FF35   CBKLDR  -FF42   CBKPT   -FEEB   CBKRTS  -FEFD
CBKSET  -FF40   CCALBS  -FE7B   CCALL   -FDB9   CDBADN  -FE6E   CDCNT   -FE5A
CDISP   -FE43   CDISPS  -FE51   CDNUM   -FE60   CDOT    -FEA8   CEN2    -FF5B
CENCDE  -FF49   CEND1   -FF6B   CENGET  -FF59   CENLP1  -FF5F   CENLP2  -FF78
CGO     -FD80   CGOBRK  -FDBF   CHKABT  -FA58   CHKRTN  -FA61   CHKSEC  -FA60
CHKWT   -FA62   CIDTA   -FADC   CIOFF   -FAF0   CION    -FAE6   CIRTN   -FAE5
CLOAD   -FE8F   CLVDFT  -FE9B   CLVOFS  -FE92   CMD     -F8F7   CMD2    -F935
CMD3    -F948   CMDBAD  -F95C   CMDCMP  -F977   CMDDDL  -F901   CMDFLS  -F96C
CMDGOT  -F94D   CMDMEM  -F990   CMDNEP  -F8F9   CMDNOL  -F90A   CMDSCH  -F953
CMDSIZ  -F96F   CMDSME  -F967   CMDTB2  -F99B   CMDTBL  -F99C   CMDXQT  -F987
CMEM    -FDC3   CMEM2   -FDC8   CMEM4   -FDD1   CMEMN   -FDC6   CMENUM  -FDE0
CMESTR  -FDEC   CMNOTB  -FE02   CMNOTC  -FDE8   CMNOTL  -FE0E   CMNOTQ  -FDF8
CMNOTU  -FE1C   CMPADP  -FE18   CMPADS  -FE16   CMSPCE  -FDFE   CNULLS  -FEB7
CNVGOT  -FD74   CNVHEX  -FD62   CNVOK   -FD76   CNVRTS  -FD78   CODTA   -FAF1
CODTAD  -FB0F   CODTAO  -FB12   CODTLP  -FB07   CODTPD  -FB03   CODTRT  -FB0D
COFFS   -FEC8   COFNO1  -FEDF   CONV1   -FF8E   CONV2   -FFB7   COOFF   -FAF0
COON    -FAE6   CPUNCH  -FE71   CR      -000D   CREG    -FC4A   CSTLEV  -FEBC
CTRACE  -FEA4   CTRCE3  -FEAA   CVER    -FEA1   CWINDO  -FE3E   DELIM   -DF8E
DFTCHP  -0000   DFTNLP  -0005   DLE     -0010   EOT     -0004   ERRMSG  -FABD
ERROR   -FACE   EXP1    -FCE9   EXP2    -FD07   EXPADD  -FD23   EXPCDL  -FD17
EXPCHM  -FD2B   EXPDLM  -FCEB   EXPRTN  -FD05   EXPSUB  -FD36   EXPTDI  -FD0D
EXPTDL  -FD0F   EXPTRM  -FD42   FIRQ    -FFE0   FIRQR   -FABC   GOADDR  -FD83
GONDFT  -FDA2   HIVTR   -0034   HSBLNK  -FC00   HSDRTN  -FC47   HSDTA   -FBFC
HSHCHR  -FC2B   HSHCOK  -FC35   HSHDOT  -FC33   HSHLNE  -FC14   HSHNXT  -FC20
HSHTTL  -FC06   INCHNP  -0000   INITVT  -F844   INTVE   -F87D   INTVS   -F870
IRQ     -FFE4   IRQR    -FAD8   LASTOP  -DF99   LDDP    -FAC1   LF      -000A
MISFLG  -DF8F   MONITR  -0008   MSHOWP  -FA79   MUPBAD  -FE36   MUPDAT  -FE2B
NMI     -FFEC   NMICON  -FAB7   NMIR    -FA7D   NMITRC  -FAB0   NUMBER  -DF9B
NUMBKP  -0008   NUMFUN  -000B   NUMVTR  -001B   OUT2HS  -0004   OUT4HS  -0005
OUTCH   -0001   PAUSE   -000B   PAUSER  -DFFC   PCNTER  -DF93   PCRLF   -0006
PDATA   -0003   PDATA1  -0002   PROMPT  -003E   PRTADR  -FE21   PSTACK  -DF95
PTM     -E000   PTMC13  -E000   PTMC2   -E001   PTMSTA  -E001   PTMTM1  -E002
PTMTM2  -E004   PTMTM3  -E006   RAMOFS  -E700   READ    -FD79   REG4    -FC94
REGAGN  -FCC3   REGCHG  -FC70   REGCNG  -FC9D   REGMSK  -FC50   REGNXC  -FCB1
REGP1   -FC78   REGP2   -FC81   REGP3   -FC92   REGPRS  -FAB3   REGPRT  -FC6F
REGRTN  -FC9B   REGSKP  -FCAA   REGTF1  -FCC9   REGTF2  -FCD6   REGTWO  -FCBB
RESET   -F837   RESET2  -F83D   ROM2OF  -F000   ROM2WK  -DF66   ROMBEG  -F800
ROMSIZ  -0800   RSRVD   -FFD4   RSRVDR  -FAD8   RSTACK  -DF97   RTI     -FABC
RTS     -FAF0   SEND    -F9EC   SIGNON  -F8C9   SKIP2   -008C   SLEVEL  -DFF8
SPACE   -0007   STACK   -DF51   STLDFT  -FEC3   SWI     -FFE8   SWI2    -FFDC
SWI2R   -FAD8   SWI3    -FFD8   SWI3R   -FAD8   SWIBFL  -DFFB   SWICNT  -DF90
SWIDNE  -F8B5   SWILP   -F8A8   SWIR    -F895   SWIVTB  -F87D   TRACEC  -DF91
TSTACK  -DF51   VCTRSW  -0009   VECTAB  -DFC2   WINDOW  -DFA0   WORKPG  -DF00
XQCIDT  -FA72   XQPAUS  -FA6E   ZBKCMD  -FAD5   ZBKPNT  -FAD3   ZIN2    -FA2A
ZINCH   -FA11   ZINCHP  -FA0F   ZMONT2  -F8E6   ZMONTR  -F8D2   ZOT2HS  -F9F2
ZOT4HS  -F9F0   ZOTCH1  -FA2E   ZOTCH2  -FA37   ZOTCH3  -FA39   ZOUT2H  -F9D9
ZOUTHX  -F9E6   ZPAUSE  -FA4E   ZPCRLF  -FA3D   ZPCRLS  -FA3C   ZPDATA  -FA40
ZPDTA1  -FA48   ZPDTLP  -FA46   ZSPACE  -F9F6   ZVSWTH  -F9FA
```

```
  Symbol   Define       References
---------- ------ ----------------------
.ACIA          92
.AVTBL         69
.BSDTA         87
.BSOFF         88
.BSON          86
.CIDTA         80
.CIOFF         81
.CION          79
.CMDL1         70
.CMDL2         91
.CODTA         83
.COOFF         84
.COON          82
.ECHO          94
.EXPAN         90
.FIRQ          74
.HSDTA         85
.IRQ           75
.NMI           77
.PAD           93
.PAUSE         89
.PTM           95
.RESET         78
.RSVD          71
.SWI           76
.SWI2          73
.SWI3          72
ACIA           21   253   822   834   850
ADDR          130  1236  1388  1389  1399  1418  1428  1434  1445  1453  1457
ARMBK2       1366   770  1354
ARMBLP       1353  1357
ARMLOP       1368  1374
ARMNSW       1361  1359
BASEPG        132   183   781
BELL           33   779
BKPTBL        124  1635
BKPTCT        118   383  1367  1591  1604  1631  1636
BKPTOP        126
BLD2          189   193
BLD3          195   198
BLDBAD       1285  1336
BLDHEX       1298  1247
BLDHXC       1299   418
BLDHXI       1296  1230
BLDNNB       1216  1161  1394
BLDNUM       1219  1283  1489  1585  1589
BLDRTN        204   202
BLDSHF       1304  1308
BLDVTR        180   215
BRKPT          63  1381
BSDCMP        941   939
BSDEOL        945   937
BSDLD1        916   919   946
BSDLD2        918   925
BSDNXT        936   942
BSDPUN        974   910
BSDSRT        943   923   947
BSDTA         908   247  1505
BSOFF         888   248  1507
BSOFLP        896   897
BSON          877   246  1504
BSON2         881   879
BSPEOF       1030  1018
BSPGO         984  1017
BSPMRE       1006  1008
BSPOK         989   987
BSPSTR       1029   994
BSPUN2       1026  1000  1002  1003  1006
BSPUNC       1027  1014
BYTE          950   927   930   932   936
BYTHEX        962   950   953
BYTRTS        960   965
CAN            37   708   715  1335
CBKADD       1616  1586
CBKADL       1624  1627
CBKADT       1629  1625
CBKDLE       1594  1590
CBKDLM       1600  1603
CBKDLP       1595  1598
CBKDSL       1607  1611
CBKDSP       1605  1584  1632
CBKERR       1628  1588  1596  1618  1622  1654  1666
```

CBKLDR	1635	300	380	1351	1366						
CBKPT	1584	500									
CBKRTS	1592	1606									
CBKSET	1634	1594	1605	1616							
CCALBS	1504	1524									
CCALL	1377	503									
CDBADN	1496	1490	1492	1509							
CDCNT	1482	1480									
CDISP	1471	506									
CDISPS	1478	1475									
CDNUM	1489	1364	1387	1466	1471	1476	1499	1501	1519	1532	1543
		1549	1558	1560							
CDOT	1534	405	1362								
CEN2	1651	1646									
CENCDE	1640	509									
CEND1	1659	1652									
CENGET	1649	1658									
CENLP1	1653	1656									
CENLP2	1665	1668									
CGO	1341	512									
CGOBRK	1380	1382									
CHKABT	706	698	761								
CHKRTN	711	707									
CHKSEC	710	716									
CHKWT	712	709	714								
CIDTA	822	240	722								
CIOFF	841	241									
CION	832	239	345								
CIRTN	827	825									
CLOAD	1513	515									
CLVDFT	1521	1518									
CLVOFS	1516	1513	1527								
CMD	377	351	436								
CMD2	412	422									
CMD3	421	419									
CMDBAD	432	461	1285	1496	1628						
CMDCMP	447	452									
CMDDDL	384	388									
CMDFLS	441	450									
CMDGOT	424	413									
CMDMEM	460	417									
CMDNEP	380	797									
CMDNOL	389	381	385	459							
CMDSCH	427	431	442								
CMDSIZ	443	440									
CMDSME	438	428									
CMDTB2	493	251									
CMDTBL	497	230									
CMDXQT	456	407	410	464							
CMEM	1387	518									
CMEM2	1389	1421	1438								
CMEM4	1394	1401	1405								
CMEMN	1388	462									
CMENUM	1402	1395									
CMESTR	1409	1414									
CMNOTB	1423	1417									
CMNOTC	1407	1398									
CMNOTL	1431	1424									
CMNOTQ	1416	1408									
CMNOTU	1440	1432									
CMPADP	1437	408	462	1429							
CMPADS	1435	1441									
CMSPCE	1419	1411									
CNULLS	1543	521									
CNVGOT	1328	1322									
CNVHEX	1319	964	1299								
CNVOK	1329	1309									
CNVRTS	1330	1284	1300	1320	1324	1326	1369				
CODTA	849	243	565								
CODTAD	866	869									
CODTAO	867	851	861								
CODTLP	861	863									
CODTPD	858	856									
CODTRT	864	853									
COFFS	1558	524									
COFNO1	1572	1569									
CONV1	1680	1642									
CONV2	1688	1659									
COOFF	842	244									
COON	833	242	346								
CPUNCH	1499	527									
CR	35	424	618	664	855	1031	1163	1182	1425	1493	1651
CREG	1099	530									
CSTLEV	1548	533									

```
CTRACE    1532   536
CTRCE3    1535   763
CVER      1527   539
CWINDO    1466   542
DELIM      150   748   754  1220  1233  1253
DFTCHP      23   254
DFTNLP      24   254
DLE         36   852
EOT         32   340   649   681   735   779  1029  1031
ERRMSG     779   433   786
ERROR      786   311
EXP1      1229   250
EXP2      1247  1231  1248
EXPADD    1263  1279
EXPCDL    1257  1249  1266
EXPCHM    1267  1259
EXPDLM    1230  1234
EXPRTN    1245  1254  1272
EXPSUB    1273  1268
EXPTDI    1251  1270
EXPTDL    1252  1238  1241  1244
EXPTRM    1283  1260  1273
FIRQ      1703  1703  1717
FIRQR      813   234
GOADDR    1346  1341  1377
GONDFT    1364  1348
HIVTR       97   589
HSBLNK    1043  1046
HSDRTN    1089  1059  1083
HSDTA     1040   245  1088  1482
HSHCHR    1073  1081
HSHCOK    1078  1076
HSHDOT    1077  1074
HSHLNE    1057  1087
HSHNXT    1065  1068
HSHTTL    1048  1056
INCHNP      53   917   921   963  1334  1644  1650
INITVT     230   185
INTVE      261   194
INTVS      253   194
IRQ       1704  1704  1718
IRQR       805   235
LASTOP     136   749  1536
LDDP       781   294   737   806
LF          34   620   635   666  1031  1423
MISFLG     148   399   616   738   769   883   894  1361
MONITR      61   219
MSHOWP     735   745
MUPBAD    1459  1456
MUPDAT    1453  1403  1413
NMI       1706  1706  1720
NMICON     769   739
NMIR       737   237
NMITRC     763   741   744
NUMBER     134   398   463  1176  1252  1257  1262  1264  1275  1296  1297
                1305  1306  1402  1494  1634
NUMBKP      26   123   125   386  1355  1371  1617  1630
NUMFUN      65   310
NUMVTR      96   121   187
OUT2HS      57  1066  1153  1571  1674
OUT4HS      58   751  1062  1150  1449  1576  1609
OUTCH       54   393   882   890   893   980  1079  1139  1143  1393  1427
                1462
PAUSE       64   721
PAUSER     114   249
PCNTER     142   390  1239
PCRLF       59   378  1041  1058  1090  1158  1436  1578  1613  1676
PDATA       56   349   788   996  1020
PDATA1      55   435   747
PROMPT      25   391
PRTADR    1445  1437
PSTACK     140   395   432
PTM         22    39    40    41    42    43    44   256   350   352   353
                 355   356   358  1537  1539
PTMC13      40   356
PTMC2       41   355   358
PTMSTA      39
PTMTM1      42   352   353  1539
PTMTM2      43
PTMTM3      44
RAMOFS      18   108
READ      1333   404   421  1255  1298  1409
REG4      1154  1173  1183
REGAGN    1186  1164
```

```
REGCHG       1132  1101
REGCNG       1161  1146
REGMSK       1120  1134
REGNXC       1174  1162
REGP1        1135  1140  1156
REGP2        1141  1137
REGP3        1152  1148
REGPRS        765   752   796
REGPRT       1131   765  1099
REGRTN       1159  1198
REGSKP       1169  1172
REGTF1       1188  1191
REGTF2       1194  1197
REGTWO       1180  1178
RESET         214   238  1721
RESET2        216   220
ROM2OF         20   199
ROM2WK        152
ROMBEG         17    20   108   164  1713
ROMSIZ         19    20  1713
RSRVD        1700  1714
RSRVDR        806   231
RSTACK        138   342   785
RTI           771   813
RTS           838   784   841   842
SEND          565   621   637   665   679
SIGNON        339   347
SKIP2          46   860  1151  1217  1520
SLEVEL        120   743  1550  1553
SPACE          60  1044  1051  1053  1070  1170  1420
STACK         155   214
STLDFT       1552  1548
SWI          1705  1705  1719
SWI2         1702  1702  1716
SWI2R         803   233
SWI3         1701  1701  1715
SWI3R         804   232
SWIBFL        116   298   308  1360
SWICNT        146   293   638   740
SWIDNE        308   299   303
SWILP         302   305
SWIR          293   236
SWIVTB        280   280   281   282   283   284   285   286   287   288   289
                    290   291   314
TRACEC        144   400   756   759  1533
TSTACK        154  1186
VCTRSW         62
VECTAB        122   180   345   346   350   426   429   565   591   622   721
                    722   822   834   850   854   857   974   978   982  1022
                   1221  1482  1504  1505  1507  1537  1544  1700  1701  1702
                   1703  1704  1705  1706
WINDOW        128  1242  1467
WORKPG        108   109   110
XQCIDT        722   609   706   713
XQPAUS        721   608   697   712   866
ZBKCMD        797   753   755   757   760   762
ZBKPNT        796   290   307   807
ZIN2          622   619
ZINCH         609   280   612   614
ZINCHP        608   610
ZMONT2        350   344
ZMONTR        342   288
ZOT2HS        568   284
ZOT4HS        567   285
ZOTCH1        633   281
ZOTCH2        637   579
ZOTCH3        638   590   595   597   617   623   701
ZOUT2H        554   567   568  1027  1390
ZOUTHX        561   558  1049
ZPAUSE        697   291
ZPCRLF        651   286
ZPCRLS        649   634   651
ZPDATA        664   283
ZPDTA1        680   282
ZPDTLP        679   636   682
ZSPACE        578   287
ZVSWTH        588   289

Symbols: 319, References: 610
```

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+	62	•			
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+	2+
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	~	#
90	SUBA	Direct	4	2
91	CMPA		4	2
92	SBCA		4	2
93	SUBD		6	2
94	ANDA		4	2
95	BITA		4	2
96	LDA		4	2
97	STA		4	2
98	EORA		4	2
99	ADCA		4	2
9A	ORA		4	2
9B	ADDA		4	2
9C	CMPX		6	2
9D	JSR		7	2
9E	LDX		5	2
9F	STX	Direct	5	2
A0	SUBA	Indexed	4+	2+
A1	CMPA		4+	2+
A2	SBCA		4+	2+
A3	SUBD		6+	2+
A4	ANDA		4+	2+
A5	BITA		4+	2+
A6	LDA		4+	2+
A7	STA		4+	2+
A8	EORA		4+	2+
A9	ADCA		4+	2+
AA	ORA		4+	2+
AB	ADDA		4+	2+
AC	CMPX		6+	2+
AD	JSR		7+	2+
AE	LDX		5+	2+
AF	STX	Indexed	5+	2+
B0	SUBA	Extended	5	3
B1	CMPA		5	3
B2	SBCA		5	3
B3	SUBD		7	3
B4	ANDA		5	3
B5	BITA		5	3
B6	LDA		5	3
B7	STA		5	3
B8	EORA		5	3
B9	ADCA		5	3
BA	ORA		5	3
BB	ADDA		5	3
BC	CMPX		7	3
BD	JSR		8	3
BE	LDX		6	3
BF	STX	Extended	6	3

NOTE: All unused opcodes are both undefined and illegal

OP	Mnem	Mode	~	#
C0	SUBB	Immed	2	2
C1	CMPB		2	2
C2	SBCB		2	2
C3	ADDD		4	3
C4	ANDB		2	2
C5	BITB	Immed	2	2
C6	LDB	Immed	2	2
C7	•			
C8	EORB		2	2
C9	ADCB		2	2
CA	ORB		2	2
CB	ADDB		2	2
CC	LDD		3	3
CD	•			
CE	LDU	Immed	3	3
CF	•			
D0	SUBB	Direct	4	2
D1	CMPB		4	2
D2	SBCB		4	2
D3	ADDD		6	2
D4	ANDB		4	2
D5	BITB		4	2
D6	LDB		4	2
D7	STB		4	2
D8	EORB		4	2
D9	ADCB		4	2
DA	ORB		4	2
DB	ADDB		4	2
DC	LDD		5	2
DD	STD		5	2
DE	LDU		5	2
DF	STU	Direct	5	2
E0	SUBB	Indexed	4+	2+
E1	CMPB		4+	2+
E2	SBCB		4+	2+
E3	ADDD		6+	2+
E4	ANDB		4+	2+
E5	BITB		4+	2+
E6	LDB		4+	2+
E7	STB		4+	2+
E8	EORB		4+	2+
E9	ADCB		4+	2+
EA	ORB		4+	2+
EB	ADDB		4+	2+
EC	LDD		5+	2+
ED	STD		5+	2+
EE	LDU		5+	2+
EF	STU	Indexed	5+	2+
F0	SUBB	Extended	5	3
F1	CMPB		5	3
F2	SBCB		5	3
F3	ADDD		7	3
F4	ANDB		5	3
F5	BITB		5	3
F6	LDB		5	3
F7	STB		5	3
F8	EORB		5	3
F9	ADCB		5	3
FA	ORB		5	3
FB	ADDB	Extended	5	3
FC	LDD	Extended	6	3
FD	STD		6	3
FE	LDU		6	3
FF	STU	Extended	6	3

Page 2 and 3 Machine Codes

OP	Mnem	Mode	~	#
1021	LBRN	Relative	5	4
1022	LBHI		5(6)	4
1023	LBLS		5(6)	4
1024	LBHS, LBCC		5(6)	4
1025	LBCS, LBLO		5(6)	4
1026	LBNE		5(6)	4
1027	LBEQ		5(6)	4
1028	LBVC		5(6)	4
1029	LBVS		5(6)	4
102A	LBPL		5(6)	4
102B	LBMI		5(6)	4
102C	LBGE		5(6)	4
102D	LBLT		5(6)	4
102E	LBGT		5(6)	4
102F	LBLE	Relative	5(6)	4
103F	SWI2	Inherent	20	2
1083	CMPD	Immed	5	4
108C	CMPY		5	4
108E	LDY	Immed	4	4
1093	CMPD	Direct	7	3
109C	CMPY		7	3
109E	LDY		6	3
109F	STY	Direct	6	3
10A3	CMPD	Indexed	7+	3+
10AC	CMPY		7+	3+
10AE	LDY		6+	3+
10AF	STY	Indexed	6+	3+
10B3	CMPD	Extended	8	4
10BC	CMPY		8	4
10BE	LDY		7	4
10BF	STY	Extended	7	4
10CE	LDS	Immed	4	4
10DE	LDS	Direct	6	3
10DF	STS	Direct	6	3
10EE	LDS	Indexed	6+	3+
10EF	STS	Indexed	6+	3+
10FE	LDS	Extended	7	4
10FF	STS	Extended	7	4
113F	SWI3	Inherent	20	2
1183	CMPU	Immed	5	4
118C	CMPS	Immed	5	4
1193	CMPU	Direct	7	3
119C	CMPS	Direct	7	3
11A3	CMPU	Indexed	7+	3+
11AC	CMPS	Indexed	7+	3+
11B3	CMPU	Extended	8	4
11BC	CMPS	Extended	8	4

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 Relative OP	~	#	Description	5 H	3 N	2 Z	1 V	0 C
BCC	BCC	24	3	2	Branch C=0	•	•	•	•	•
	LBCC	10 24	5(6)	4	Long Branch C=0	•	•	•	•	•
BCS	BCS	25	3	2	Branch C=1	•	•	•	•	•
	LBCS	10 25	5(6)	4	Long Branch C=1	•	•	•	•	•
BEQ	BEQ	27	3	2	Branch Z=0	•	•	•	•	•
	LBEQ	10 27	5(6)	4	Long Branch Z=0	•	•	•	•	•
BGE	BGE	2C	3	2	Branch ≥ Zero	•	•	•	•	•
	LBGE	10 2C	5(6)	4	Long Branch ≥ Zero	•	•	•	•	•
BGT	BGT	2E	3	2	Branch > Zero	•	•	•	•	•
	LBGT	10 2E	5(6)	4	Long Branch > Zero	•	•	•	•	•
BHI	BHI	22	3	2	Branch Higher	•	•	•	•	•
	LBHI	10 22	5(6)	4	Long Branch Higher	•	•	•	•	•
BHS	BHS	24	3	2	Branch Higher or Same	•	•	•	•	•
	LBHS	10 24	5(6)	4	Long Branch Higher or Same	•	•	•	•	•
BLE	BLE	2F	3	2	Branch ≤ Zero	•	•	•	•	•
	LBLE	10 2F	5(6)	4	Long Branch ≤ Zero	•	•	•	•	•
BLO	BLO	25	3	2	Branch lower	•	•	•	•	•
	LBLO	10 25	5(6)	4	Long Branch Lower	•	•	•	•	•
BLS	BLS	23	3	2	Branch Lower or Same	•	•	•	•	•
	LBLS	10 23	5(6)	4	Long Branch Lower or Same	•	•	•	•	•
BLT	BLT	2D	3	2	Branch < Zero	•	•	•	•	•
	LBLT	10 2D	5(6)	4	Long Branch < Zero	•	•	•	•	•
BMI	BMI	2B	3	2	Branch Minus	•	•	•	•	•
	LBMI	10 2B	5(6)	4	Long Branch Minus	•	•	•	•	•
BNE	BNE	26	3	2	Branch Z≠0	•	•	•	•	•
	LBNE	10 26	5(6)	4	Long Branch Z≠0	•	•	•	•	•
BPL	BPL	2A	3	2	Branch Plus	•	•	•	•	•
	LBPL	10 2A	5(6)	4	Long Branch Plus	•	•	•	•	•
BRA	BRA	20	3	2	Branch Always	•	•	•	•	•
	LBRA	16	5	3	Long Branch Always	•	•	•	•	•
BRN	BRN	21	3	2	Branch Never	•	•	•	•	•
	LBRN	10 21	5	4	Long Branch Never	•	•	•	•	•
BSR	BSR	8D	7	2	Branch to Subroutine	•	•	•	•	•
	LBSR	17	9	3	Long Branch to Subroutine	•	•	•	•	•
BVC	BVC	28	3	2	Branch V=0	•	•	•	•	•
	LBVC	10 28	5(6)	4	Long Branch V=0	•	•	•	•	•
BVS	BVS	29	3	2	Branch V=1	•	•	•	•	•
	LBVS	10 29	5(6)	4	Long Branch V=1	•	•	•	•	•

Table D-1. Programming Aid (Continued)

SIMPLE BRANCHES

	OP	~	#
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)

Test	True	OP	False	OP
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)

Test	True	OP	False	OP
$r>m$	BGT	2E	BLE	2F
$r\geq m$	BGE	2C	BLT	2D
$r=m$	BEQ	27	BNE	26
$r\leq m$	BLE	2F	BGT	2E
$r<m$	BLT	2D	BGE	2C

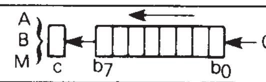
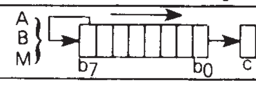
UNSIGNED CONDITIONAL BRANCHES (Notes 1-4)

Test	True	OP	False	OP
$r>m$	BHI	22	BLS	23
$r\geq m$	BHS	24	BLO	25
$r=m$	BEQ	27	BNE	26
$r\leq m$	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								
		Op	~	#	Op	~	#	Op	~	#	Op	~	#	Op	~	#		H	N	Z	V	C
ABX														3A	3	1	B + X → X (Unsigned)	•	•	•	•	•
ADC	ADCA	89	2	2	99	4	2	A9	4+	2+	B9	5	3				A + M + C → A	↕	↕	↕	↕	↕
	ADCB	C9	2	2	D9	4	2	E9	4+	2+	F9	5	3				B + M + C → B	↕	↕	↕	↕	↕
ADD	ADDA	8B	2	2	9B	4	2	AB	4+	2+	BB	5	3				A + M → A	↕	↕	↕	↕	↕
	ADDB	CB	2	2	DB	4	2	EB	4+	2+	FB	5	3				B + M → B	↕	↕	↕	↕	↕
	ADDD	C3	4	3	D3	6	2	E3	6+	2+	F3	7	3				D + M:M + 1 → D	•	↕	↕	↕	↕
AND	ANDA	84	2	2	94	4	2	A4	4+	2+	B4	5	3				A Λ M → A	•	↕	↕	0	•
	ANDB	C4	2	2	D4	4	2	E4	4+	2+	F4	5	3				B Λ M → B	•	↕	↕	0	•
	ANDCC	1C	3	2													CC Λ IMM → CC					7
ASL	ASLA													48	2	1		8	↕	↕	↕	↕
	ASLB													58	2	1		8	↕	↕	↕	↕
	ASL				08	6	2	68	6+	2+	78	7	3					8	↕	↕	↕	↕
ASR	ASRB													47	2	1		8	↕	↕	•	↕
	ASR													57	2	1		8	↕	↕	•	↕
	ASR				07	6	2	67	6+	2+	77	7	3					8	↕	↕	•	↕
BIT	BITA	85	2	2	95	4	2	A5	4+	2+	B5	5	3				Bit Test A (M Λ A)	•	↕	↕	0	•
	BITB	C5	2	2	D5	4	2	E5	4+	2+	F5	5	3				Bit Test B (M Λ B)	•	↕	↕	0	•
CLR	CLRA													4F	2	1	0 → A	•	0	1	0	0
	CLRB													5F	2	1	0 → B	•	0	1	0	0
	CLR				0F	6	2	6F	6+	2+	7F	7	3				0 → M	•	0	1	0	0
CMP	CMPA	81	2	2	91	4	2	A1	4+	2+	B1	5	3				Compare M from A	8	↕	↕	↕	↕
	CMPB	C1	2	2	D1	4	2	E1	4+	2+	F1	5	3				Compare M from B	8	↕	↕	↕	↕
	CMPD	10	5	4	10	7	3	10	7+	3+	10	8	4				Compare M:M + 1 from D	•	↕	↕	↕	↕
		83			93			A3			B3											
	CMPS	11	5	4	11	7	3	11	7+	3+	11	8	4				Compare M:M + 1 from S	•	↕	↕	↕	↕
		8C			9C			AC			BC											
	CMPU	11	5	4	11	7	3	11	7+	3+	11	8	4				Compare M:M + 1 from U	•	↕	↕	↕	↕
		83			93			A3			B3											
	CMPX	8C	4	3	9C	6	2	AC	6+	2+	BC	7	3				Compare M:M + 1 from X	•	↕	↕	↕	↕
	CMPY	10	5	4	10	7	3	10	7+	3+	10	8	4				Compare M:M + 1 from Y	•	↕	↕	↕	↕
		8C			9C			AC			BC											
COM	COMA													43	2	1	$\overline{A}$ → A	•	↕	↕	0	1
	COMB													53	2	1	$\overline{B}$ → B	•	↕	↕	0	1
	COM				03	6	2	63	6+	2+	73	7	3				$\overline{M}$ → M	•	↕	↕	0	1
CWAI		3C	≥20	2													CC Λ IMM → CC Wait for Interrupt					7
DAA														19	2	1	Decimal Adjust A	•	↕	↕	0	↕
DEC	DECA													4A	2	1	A − 1 → A	•	↕	↕	↕	•
	DECB													5A	2	1	B − 1 → B	•	↕	↕	↕	•
	DEC				0A	6	2	6A	6+	2+	7A	7	3				M − 1 → M	•	↕	↕	↕	•
EOR	EORA	88	2	2	98	4	2	A8	4+	2+	B8	5	3				A ⊻ M → A	•	↕	↕	0	•
	EORB	C8	2	2	D8	4	2	E8	4+	2+	F8	5	3				B ⊻ M → B	•	↕	↕	0	•
EXG	R1, R2	1E	8	2													R1 → R2[2]	•	•	•	•	•
INC	INCA													4C	2	1	A + 1 → A	•	↕	↕	↕	•
	INCB													5C	2	1	B + 1 → B	•	↕	↕	↕	•
	INC				0C	6	2	6C	6+	2+	7C	7	3				M + 1 → M	•	↕	↕	↕	•
JMP					0E	3	2	6E	3+	2+	7E	4	3				EA[3] → PC	•	•	•	•	•
JSR					9D	7	2	AD	7+	2+	BD	8	3				Jump to Subroutine	•	•	•	•	•
LD	LDA	86	2	2	96	4	2	A6	4+	2+	B6	5	3				M → A	•	↕	↕	0	•
	LDB	C6	2	2	D6	4	2	E6	4+	2+	F6	5	3				M → B	•	↕	↕	0	•
	LDD	CC	3	3	DC	5	2	EC	5+	2+	FC	6	3				M:M + 1 → D	•	↕	↕	0	•
	LDS	10	4	4	10	6	3	10	6+	3+	10	7	4				M:M + 1 → S	•	↕	↕	0	•
		CE			DE			EE			FE											
	LDU	CE	3	3	DE	5	2	EE	5+	2+	FE	6	3				M:M + 1 → U	•	↕	↕	0	•
	LDX	8E	3	3	9E	5	2	AE	5+	2+	BE	6	3				M:M + 1 → X	•	↕	↕	0	•
	LDY	10	4	4	10	6	3	10	6+	3+	10	7	4				M:M + 1 → Y	•	↕	↕	0	•
		8E			9E			AE			BE											
LEA	LEAS							32	4+	2+							EA[3] → S	•	•	•	•	•
	LEAU							33	4+	2+							EA[3] → U	•	•	•	•	•
	LEAX							30	4+	2+							EA[3] → X	•	•	↕	•	•
	LEAY							31	4+	2+							EA[3] → Y	•	•	↕	•	•

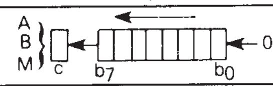
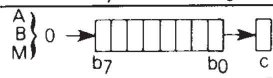
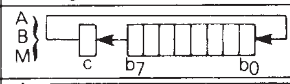
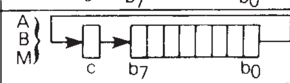
Legend:

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

$\overline{M}$ 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
Λ Logical and
⊻ Logical Exclusive or

Table D-1. Programming Aid (Continued)

Instruction	Forms	Immediate Op	Immediate ~	Immediate #	Direct Op	Direct ~	Direct #	Indexed[1] Op	Indexed[1] ~	Indexed[1] #	Extended Op	Extended ~	Extended #	Inherent Op	Inherent ~	Inherent #	Description	5 H	3 N	2 Z	1 V	0 C
LSL	LSLA													48	2	1		•	↕	↕	↕	↕
	LSLB													58	2	1		•	↕	↕	↕	↕
	LSL				08	6	2	68	6+	2+	78	7	3					•	↕	↕	↕	↕
LSR	LSRA													44	2	1		•	0	↕	•	↕
	LSRB													54	2	1		•	0	↕	•	↕
	LSR				04	6	2	64	6+	2+	74		3					•	0	↕	•	↕
MUL														3D	11	1	A × B → D (Unsigned)	•	•	↕	•	9
NEG	NEGA													40	2	1	$\overline{A}$ + 1 → A	8	↕	↕	↕	↕
	NEGB													50	2	1	$\overline{B}$ + 1 → B	8	↕	↕	↕	↕
	NEG				00	6	2	60	6+	2+	70	7	3				$\overline{M}$ + 1 → M	8	↕	↕	↕	↕
NOP														12	2	1	No Operation	•	•	•	•	•
OR	ORA	8A	2	2	9A	4	2	AA	4+	2+	BA	5	3				A V M → A	•	↕	↕	0	•
	ORB	CA	2	2	DA	4	2	EA	+	2+	FA	5	3				B V M → B	•	↕	↕	0	•
	ORCC	1A	3	2													CC V IMM → CC				7	
PSH	PSHS	34	5+[4]	2													Push Registers on S Stack	•	•	•	•	•
	PSHU	36	5+[4]	2													Push Registers on U Stack	•	•	•	•	•
PUL	PULS	35	5+[4]	2													Pull Registers from S Stack	•	•	•	•	•
	PULU	37	5+[4]	2													Pull Registers from U Stack	•	•	•	•	•
ROL	ROLA													49	2	1		•	↕	↕	↕	↕
	ROLB													59	2	1		•	↕	↕	↕	↕
	ROL				09	6	2	69	6+	2+	79	7	3					•	↕	↕	↕	↕
ROR	RORA													46	2	1		•	↕	↕	•	↕
	RORB													56	2	1		•	↕	↕	•	↕
	ROR				06	6	2	66	6+	2+	76	7	3					•	↕	↕	•	↕
RTI														3B	6/15	1	Return From Interrupt					7
RTS														39	5	1	Return from Subroutine	•	•	•	•	•
SBC	SBCA	82	2	2	92	4	2	A2	4+	2+	B2	5	3				A − M − C → A	8	↕	↕	↕	↕
	SBCB	C2	2	2	D2	4	2	E2	4+	2+	F2	5	3				B − M − C → B	8	↕	↕	↕	↕
SEX														1D	2	1	Sign Extend B into A	•	↕	↕	0	•
ST	STA				97	4	2	A7	4+	2+	B7	5	3				A → M	•	↕	↕	0	•
	STB				D7	4	2	E7	4+	2+	F7	5	3				B → M	•	↕	↕	0	•
	STD				DD	5	2	ED	5+	2+	FD	6	3				D → M:M + 1	•	↕	↕	0	•
	STS				10 DF	6	3	10 EF	6+	3+	10 FF	7	4				S → M:M + 1	•	↕	↕	0	•
	STU				DF	5	2	EF	5+	2+	FF	6	3				U → M:M + 1	•	↕	↕	0	•
	STX				9F	5	2	AF	5+	2+	BF	6	3				X → M:M + 1	•	↕	↕	0	•
	STY				10 9F	6	3	10 AF	6+	3+	10 BF	7	4				Y → M:M + 1	•	↕	↕	0	•
SUB	SUBA	80	2	2	90	4	2	A0	4+	2+	B0	5	3				A − M → A	8	↕	↕	↕	↕
	SUBB	C0	2	2	D0	4	2	E0	4+	2+	F0	5	3				B − M → B	8	↕	↕	↕	↕
	SUBD	83	4	3	93	6	2	A3	6+	2+	B3	7	3				D − M:M + 1 → D	•	↕	↕	↕	↕
SWI	SWI[6]													3F	19	1	Software Interrupt 1	•	•	•	•	•
	SWI2[6]													10 3F	20	2	Software Interrupt 2	•	•	•	•	•
	SWI3[6]													11 3F	20	1	Software Interrupt 3	•	•	•	•	•
SYNC														13	≥4	1	Synchronize to Interrupt	•	•	•	•	•
TFR	R1, R2	1F	6	2													R1 → R2[2]	•	•	•	•	•
TST	TSTA													4D	2	1	Test A	•	↕	↕	0	•
	TSTB													5D	2	1	Test B	•	↕	↕	0	•
	TST				0D	6	2	6D	6+	2+	7D	7	3				Test M	•	↕	↕	0	•

Notes:

1. 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.
2. 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
3. EA is the effective address.
4. The PSH and PUL instructions require 5 cycles plus 1 cycle for each **byte** pushed or pulled.
5. 5(6) means: 5 cycles if branch not taken, 6 cycles if taken (Branch instructions).
6. SWI sets I and F bits. SWI2 and SWI3 do not affect I and F.
7. Conditions Codes set as a direct result of the instruction.
8. Value of half-carry flag is undefined.
9. 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.

Bits b7						0	0	0	0	1	1	1	1
b6						0	0	1	1	0	0	1	1
b5						0	1	0	1	0	1	0	1
b4	b3	b2	b1		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-existant 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 occurrance 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

Symbol	Name
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 mchine 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

Least Significant Four Bits	DIR 0000 0	0001 1	REL 0010 2	0011 3	ACCA 0100 4	ACCB 0101 5	IND 0110 6	EXT 0111 7	IMM 1000 8	DIR 1001 9	IND 1010 A	EXT 1011 B	IMM 1100 C	DIR 1101 D	IND 1110 E	EXT 1111 F	
0000 0	6 NEG	PAGE2	3 BRA	4+1 LEAX	2	2 NEG	6+1	7	2	4 SUBA	4+1	5	2	4 SUBB	4+1	5	0
0001 1	—	PAGE3	3 BRN/ 5 LBRN	4+1 LEAY	—	—	—	—	2	4 CMPA	4+1	5	2	4 CMPB	4+1	5	1
0010 2	—	2 NOP	3 BHI/ 5(6) LBHI	4+1 LEAS	—	—	—	—	2	4 SBCA	4+1	5	2	4 SBCB	4+1	5	2
0011 3	6 COM	2 SYNC	3 BLS/ 5(6) LBLS	4+1 LEAU	2	2 COM	6+1	7	4,6,6+1,7 SUBD	5,7,7+1,8 CMPD		5,7,7+1,8 CMPU	4	6 ADDD	6+1	7	3
0100 4	6 LSR	—	3 BHS 5(6) (BCC)	5+1/by PSHS	2	2 LSR	6+1	7	2	4 ANDA	4+1	5	2	4 ANDB	4+1	5	4
0101 5	—	—	3 BLO 5(6) (BCS)	5+1/by PULS	—	—	—	—	2	4 BITA	4+1	5	2	4 BITB	4+1	5	5
0110 6	6 ROR	5 LBRA	3 BNE/ 5(6) LBNE	5+1/by PSHU	2	2 ROR	6+1	7	2	4 LDA	4+1	5	2	4 LDB	4+1	5	6
0111 7	6 ASR	9 LBSR	3 BEQ/ 5(6) LBEQ	5+1/by PULU	2	2 ASR	6+1	7	—	4	4+1 STA	5	—	4	4+1 STB	5	7
1000 8	6 ASL (LSL)	—	3 BVC/ 5(6) LBVC	—	2	2 ASL (LSL)	6+1	7	2	4 EORA	4+1	5	2	4 EORB	4+1	5	8
1001 9	6 ROL	2 DAA	3 BVS/ 5(6) LBVS	5 RTS	2	2 ROL	6+1	7	2	4 ADCA	4+1	5	2	4 ADCB	4+1	5	9
1010 A	6 DEC	3 ORCC	3 BPL/ 5(6) LBPL	3 ABX	2	2 DEC	6+1	7	2	4 ORA	4+1	5	2	4 ORB	4+1	5	A
1011 B	—	—	3 BMI/ 5(6) LBMI	6/15 RTI	—	—	—	—	2	4 ADDA	4+1	5	2	4 ADDB	4+1	5	B
1100 C	6 INC	3 ANDCC	3 BGE/ 5(6) LBGE	20 CWAI	2	2 INC	6+1	7	4,6,6+1,7 CMPX	5,7,7+1,8 CMPY		5,7,7+1,8 CMPS	3	5 LDD	5+1	6	C
1101 D	6 TST	2 SEX	3 BLT/ 5(6) LBLT	11 MUL	2	2 TST	6+1	7	7 BSR	7	7+1 JSR	8	—	5	5+1 STD	6	D
1110 E	3 JMP	8 EXG	3 BGT/ 5(6) LBGT	—	—	—	3+1	4 JMP	3,5,5+1,6 LDX		4,6,6+1,7 LDY		3,5,5+1,6 LDU		4,6,6+1,7 LDS		E
1111 F	6 CLR	7 TFR	3 BLE/ 5(6) LBLE	19/20/20 SWI/2/3	2	2 CLR	6+1	7	—	5,5+1,6 STX		6,6+1,7 STY	—	5,5+1,6 STU		6,6+1,7 STS	F

Table F-2. Indexed Addressing Mode Data

Type	Forms	Non Indirect				Indirect			
		Assembler Form	Postbyte OP Code	$\overset{+}{\sim}$	$\overset{+}{\#}$	Assembler Form	Postbyte OP Code	$\overset{+}{\sim}$	$\overset{+}{\#}$
Constant Offset From R (twos complement offset)	No Offset	,R	1RR00100	0	0	[,R]	1RR10100	3	0
	5 Bit Offset	n, R	0RRnnnnn	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 (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 (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

$\overset{+}{\sim}$ and $\overset{+}{\#}$ 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.

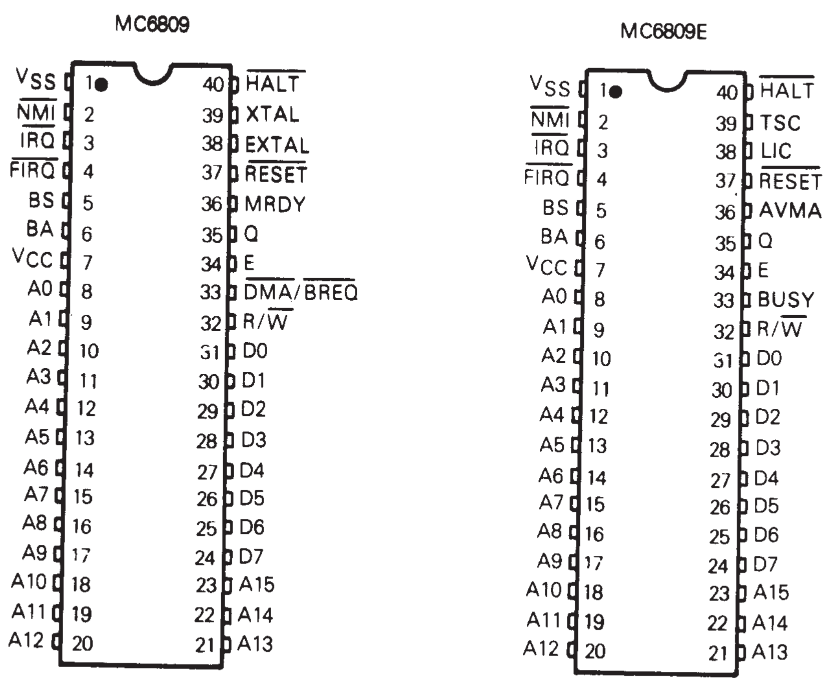


Figure G-1. Pin Assignments

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 m=	2^n n=	Value	16^m m=	2^n 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
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	3	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,584	E	224	E	14
F	61,440	F	3,840	F	240	F	15