

## 8-Bit Microprocessing Unit

The MC6809 is a high-performance 8-bit microprocessor which supports modern programming techniques such as position independence, re-entrancy, and modular programming.

This third-generation addition to the M6800 Family has major architectural improvements which include additional registers, instructions, and addressing modes.

The basic instructions of any computer are greatly enhanced by the presence of powerful addressing modes. The MC6809 has the most complete set of addressing modes available on any 8-bit microprocessor.

The MC6809 has hardware and software features which make it an ideal processor for higher level language execution or standard controller applications.

### MC6800 COMPATIBLE

- Hardware — Interfaces with All M6800 Peripherals
- Software — Upward Source Code Compatible Instruction Set and Addressing Modes

### ARCHITECTURAL FEATURES

- Two 16-Bit Index Registers
- Two 16-Bit Indexable Stack Pointers
- Two 8-Bit Accumulators Can Be Concatenated To Form One 16-Bit Accumulator
- Direct Page Register Allows Direct Addressing Throughout Memory

### HARDWARE FEATURES

- On-Chip Oscillator (Crystal Frequency =  $4 \times E$ )
- $\overline{DMA\ BREQ}$  Allows DMA Operation on Memory Refresh
- Fast Interrupt Request Input Stacks Only Condition Code Register and Program Counter
- $\overline{MRDY}$  Input Extends Data Access Times for Use with Slow Memory
- Interrupt Acknowledge Output Allows Vectoring by Devices
- Sync Acknowledge Output Allows for Synchronization to External Event
- Single Bus-Cycle  $\overline{RESET}$
- Single 5-Volt Supply Operation
- NMI Inhibited After  $\overline{RESET}$  Until After First Load of Stack Pointer
- Early Address Valid Allows Use with Slower Memories
- Early Write Data for Dynamic Memories

### SOFTWARE FEATURES

- 10 Addressing Modes
  - 6800 Upward Compatible Addressing Modes
  - Direct Addressing Anywhere in Memory Map
  - Long Relative Branches
  - Program Counter Relative
  - True Indirect Addressing
  - Expanded Indexed Addressing:
    - 0-, 5-, 8-, or 16-Bit Constant Offsets
    - 8- or 16-Bit Accumulator Offsets
    - Auto Increment/Decrement by 1 or 2
- Improved Stack Manipulation
- 1464 Instructions with Unique Addressing Modes
- $8 \times 8$  Unsigned Multiply
- 16-Bit Arithmetic
- Transfer/Exchange All Registers
- Push/Pull Any Registers or Any Set of Registers
- Load Effective Address

This document contains information on a new product. Specifications and information herein are subject to change without notice.

## MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V <sub>CC</sub>	-0.3 to +7.0	V
Input Voltage	V <sub>in</sub>	-0.3 to +7.0	V
Operating Temperature Range MC6809, MC68A09, MC68B09 MC6809C, MC68A09C, MC68B09C	T <sub>A</sub>	T <sub>L</sub> to T <sub>H</sub> 0 to +70 -40 to +85	°C
Storage Temperature Range	T <sub>stg</sub>	-55 to +150	°C

## THERMAL CHARACTERISTICS

Characteristic	Symbol	Value	Unit
Thermal Resistance Cerdip Plastic	θ <sub>JA</sub>	60 100	C/W

## POWER CONSIDERATIONS

The average chip-junction temperature, T<sub>J</sub>, in °C can be obtained from:

$$T_J = T_A + (P_D \cdot \theta_{JA}) \quad (1)$$

where:

- T<sub>A</sub> = Ambient Temperature, °C
- θ<sub>JA</sub> = Package Thermal Resistance, Junction-to-Ambient, C/W
- P<sub>D</sub> = P<sub>INT</sub> + P<sub>PORT</sub>
- P<sub>INT</sub> = I<sub>CC</sub> × V<sub>CC</sub>, Watts — Chip Internal Power
- P<sub>PORT</sub> = Port Power Dissipation, Watts — User Determined

For most applications P<sub>PORT</sub> < P<sub>INT</sub> and can be neglected. P<sub>PORT</sub> may become significant if the device is configured to drive Darlington bases or sink LED loads.

An approximate relationship between P<sub>D</sub> and T<sub>J</sub> (if P<sub>PORT</sub> is neglected) is:

$$P_D = K \cdot (T_J - 273 \text{ C}) \quad (2)$$

Solving equations (1) and (2) for K gives:

$$K = P_D \cdot (T_A - 273 \text{ C}) - \theta_{JA} \cdot P_D^2 \quad (3)$$

where K is a constant pertaining to the particular part. K can be determined from equation (3) by measuring P<sub>D</sub> (at equilibrium) for a known T<sub>A</sub>. Using this value of K, the values of P<sub>D</sub> and T<sub>J</sub> can be obtained by solving equations (1) and (2) iteratively for any value of T<sub>A</sub>.

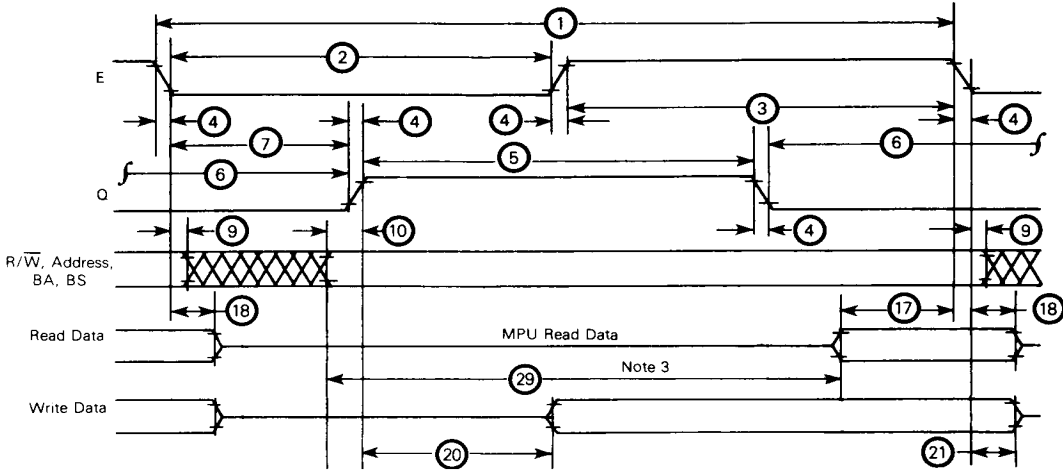
ELECTRICAL CHARACTERISTICS (V<sub>CC</sub> = 5.0 V ± 5%, V<sub>SS</sub> = 0, T<sub>A</sub> = T<sub>L</sub> to T<sub>H</sub> unless otherwise noted)

Characteristic	Symbol	Min	Typ	Max	Unit
Input High Voltage Logic, EXTERNAL RESET	V <sub>IH</sub> V <sub>IHR</sub>	V <sub>SS</sub> + 2.0 V <sub>SS</sub> + 4.0	—	V <sub>CC</sub> V <sub>CC</sub>	V
Input Low Voltage Logic, EXTERNAL, RESET	V <sub>IL</sub>	V <sub>SS</sub> - 0.3	—	V <sub>SS</sub> + 0.8	V
Input Leakage Current (V <sub>in</sub> = 0 to 5.25 V, V <sub>CC</sub> = max)	I <sub>in</sub>	—	—	2.5	μA
dc Output High Voltage (I <sub>Load</sub> = -205 μA, V <sub>CC</sub> = min) (I <sub>Load</sub> = -145 μA, V <sub>CC</sub> = min) (I <sub>Load</sub> = -100 μA, V <sub>CC</sub> = min)	V <sub>OH</sub>	V <sub>SS</sub> + 2.4 V <sub>SS</sub> + 2.4 V <sub>SS</sub> + 2.4	— — —	— — —	V
dc Output Low Voltage (I <sub>Load</sub> = 2.0 mA, V <sub>CC</sub> = min)	V <sub>OL</sub>	—	—	V <sub>SS</sub> + 0.5	V
Internal Power Dissipation (Measured at T <sub>A</sub> = 0°C in Steady State Operation)	P <sub>INT</sub>	—	—	1.0	W
Capacitance * (V <sub>in</sub> = 0, T <sub>A</sub> = 25°C, f = 1.0 MHz)	C <sub>in</sub>	—	10	15	pF
	C <sub>out</sub>	—	—	15	pF
Frequency of Operation (Crystal or External Input)	f <sub>XTAL</sub>	0.4 0.4 0.4	— — —	4 6 8	MHz
Hi-Z (Off State) Input Current (V <sub>in</sub> = 0.4 to 2.4 V, V <sub>CC</sub> = max)	I <sub>TSI</sub>	—	2.0	10	μA

\* Capacitances are periodically tested rather than 100% tested.

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum rated voltages to this high impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage levels (e.g., either V<sub>SS</sub> or V<sub>CC</sub>).

FIGURE 1 – BUS TIMING



BUS TIMING CHARACTERISTICS (See Notes 1 and 2)

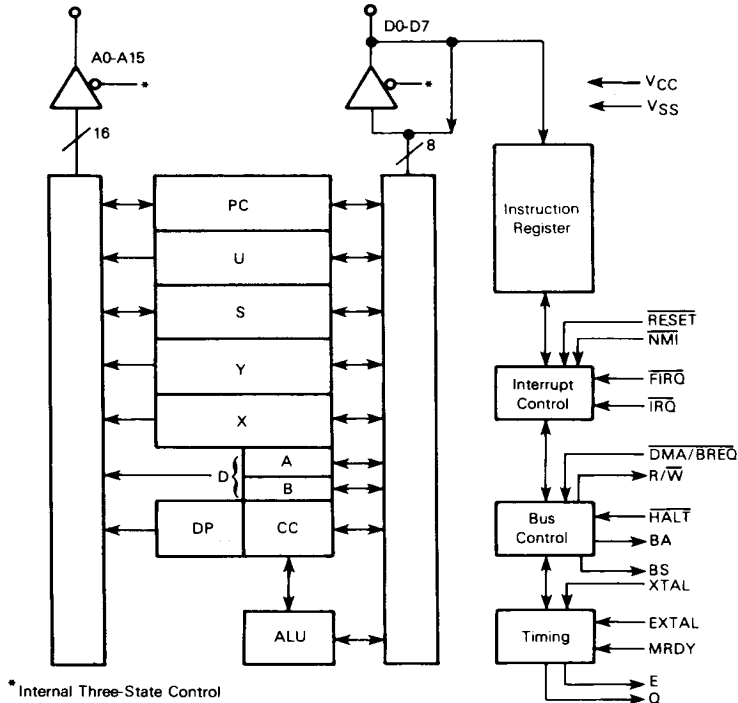
Ident. Number	Characteristics	Symbol	MC6809		MC68A09		MC68B09		Unit
			Min	Max	Min	Max	Min	Max	
1	Cycle Time (See Note 5)	$t_{cyc}$	1.0	10	0.667	10	0.5	10	$\mu$ s
2	Pulse Width, E Low	PW <sub>EL</sub>	430	5000	280	5000	210	5000	ns
3	Pulse Width, E High	PW <sub>EH</sub>	450	15500	280	15700	220	15700	ns
4	Clock Rise and Fall Time	$t_r, t_f$	—	25	—	25	—	20	ns
5	Pulse Width, Q High	PW <sub>QH</sub>	430	5000	280	5000	210	5000	ns
6	Pulse Width, Q Low	PW <sub>QL</sub>	450	15500	280	15700	220	15700	ns
7	Delay Time, E to Q Rise	$t_{AVS}$	200	250	130	165	80	125	ns
9	Address Hold Time* (See Note 4)	$t_{AH}$	20	—	20	—	20	—	ns
10	BA, BS, R/W, and Address Valid Time to Q Rise	$t_{AQ}$	50	—	25	—	15	—	ns
17	Read Data Setup Time	$t_{DSR}$	80	—	60	—	40	—	ns
18	Read Data Hold Time*	$t_{DHR}$	10	—	10	—	10	—	ns
20	Data Delay Time from Q	$t_{DDQ}$	—	200	—	140	—	110	ns
21	Write Data Hold Time*	$t_{DHW}$	30	—	30	—	30	—	ns
29	Usable Access Time (See Note 3)	$t_{ACC}$	695	—	440	—	330	—	ns
	Processor Control Setup Time (MRDY, Interrupts, DMA/BREQ, HALT, RESET) (Figures 6, 8, 9, 10, 12, and 13)	$t_{PCS}$	200	—	140	—	110	—	ns
	Crystal Oscillator Start Time (Figures 6 and 7)	$t_{RC}$	—	100	—	100	—	100	ms
	Processor Control Rise and Fall Time (Figures 6 and 8)	$t_{PCr}, t_{PCf}$	—	100	—	100	—	100	ns

\* Address and data hold times are periodically tested rather than 100% tested.

NOTES:

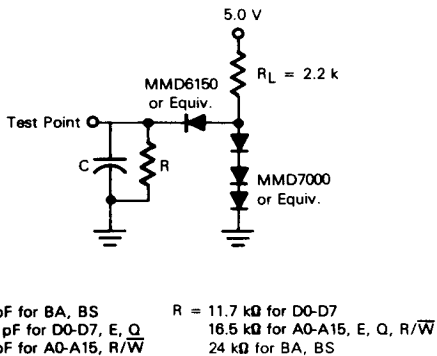
- Voltage levels shown are  $V_L = 0.4$  V,  $V_H = 2.4$  V, unless otherwise specified.
- Measurement points shown are 0.8 V and 2.0 V, unless otherwise specified.
- Usable access time is computed by: 1 - 4 - 7 max + 10 - 17.
- Hold time (Ⓣ) for BA and BS is not specified.
- Maximum  $t_{cyc}$  during MRDY or DMA BREQ is 16  $\mu$ s.
- MC6809 - 1.0 MHz, MC68A09 - 1.5 MHz, MC68B09 - 2.0 MHz.

FIGURE 2 — MC6809 EXPANDED BLOCK DIAGRAM



\* Internal Three-State Control

FIGURE 3 — BUS TIMING TEST LOAD



**PROGRAMMING MODEL**

As shown in Figure 4, the MC6809 adds three registers to the set available in the MC6800. The added registers include a direct page register, the user stack pointer, and a second index register.

**ACCUMULATORS (A, B, D)**

The A and B registers are general purpose accumulators which are used for arithmetic calculations and manipulation of data.

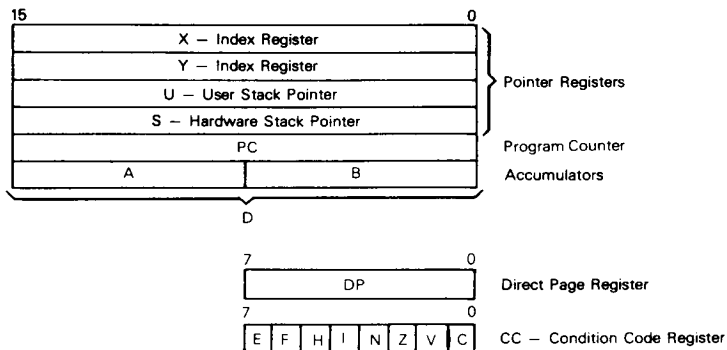
Certain instructions concatenate the A and B registers to form a single 16-bit accumulator. This is referred to as the D register, and is formed with the A register as the most significant byte.

**DIRECT PAGE REGISTER (DP)**

The direct page register of the MC6809 serves to enhance the direct addressing mode. The content of this register appears at the higher address outputs (A8-A15) during direct addressing instruction execution. This allows the direct mode to be used at any place in memory, under program control. To ensure M6800 compatibility, all bits of this register are cleared during processor reset.

3

FIGURE 4 — PROGRAMMING MODEL OF THE MICROPROCESSING UNIT



**INDEX REGISTERS (X, Y)**

The index registers are used in indexed mode of addressing. The 16-bit address in this register takes part in the calculation of effective addresses. This address may be used to point to data directly or may be modified by an optional constant or register offset. During some indexed modes, the contents of the index register are incremented or decremented to point to the next item of tabular type data. All four pointer registers (X, Y, U, S) may be used as index registers.

**STACK POINTER (U, S)**

The hardware stack pointer (S) is used automatically by the processor during subroutine calls and interrupts. The stack pointers of the MC6809 point to the top of the stack, in contrast to the MC6800 stack pointer, which pointed to the next free location on the stack. The user stack pointer (U) is controlled exclusively by the programmer. This allows arguments to be passed to and from subroutines with ease. Both stack pointers have the same indexed mode addressing capabilities as the X and Y registers, but also support Push and Pull instructions. This allows the MC6809 to be used efficiently as a stack processor, greatly enhancing its ability to support higher level languages and modular programming.

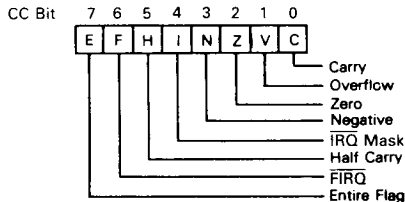
**PROGRAM COUNTER**

The program counter is used by the processor to point to the address of the next instruction to be executed by the processor. Relative addressing is provided allowing the program counter to be used like an index register in some situations.

**CONDITION CODE REGISTER**

The condition code register defines the state of the processor at any given time. See Figure 5:

FIGURE 5 — CONDITION CODE REGISTER FORMAT



**CONDITION CODE REGISTER DESCRIPTION**

**CARRY FLAG (C)**

Bit 0 is the carry flag, and is usually the carry from the binary ALU. C is also used to represent a 'borrow' from subtract-like instructions (CMP, NEG, SUB, SBC) and is the complement of the carry from the binary ALU.

**OVERFLOW FLAG (V)**

Bit 1 is the overflow flag, and is set to a one by an operation which causes a signed twos complement arithmetic overflow. This overflow is detected in an operation in which the carry from the MSB in the ALU does not match the carry from the MSB-1.

**ZERO FLAG (Z)**

Bit 2 is the zero flag, and is set to a one if the result of the previous operation was identically zero.

**NEGATIVE FLAG (N)**

Bit 3 is the negative flag, which contains exactly the value of the MSB of the result of the preceding operation. Thus, a negative twos-complement result will leave N set to a one.

**IRQ MASK (I)**

Bit 4 is the  $\overline{\text{IRQ}}$  mask bit. The processor will not recognize interrupts from the  $\overline{\text{IRQ}}$  line if this bit is set to a one.  $\overline{\text{NMI}}$ ,  $\overline{\text{FIRQ}}$ ,  $\overline{\text{IRQ}}$ ,  $\overline{\text{RESET}}$ , and  $\overline{\text{SWI}}$  all set I to a one.  $\overline{\text{SWI2}}$  and  $\overline{\text{SWI3}}$  do not affect I.

**HALF CARRY (H)**

Bit 5 is the half-carry bit, and is used to indicate a carry from bit 3 in the ALU as a result of an 8-bit addition only (ADC or ADD). This bit is used by the DAA instruction to perform a BCD decimal add adjust operation. The state of this flag is undefined in all subtract-like instructions.

 **$\overline{\text{FIRQ}}$  MASK (F)**

Bit 6 is the  $\overline{\text{FIRQ}}$  mask bit. The processor will not recognize interrupts from the  $\overline{\text{FIRQ}}$  line if this bit is a one.  $\overline{\text{NMI}}$ ,  $\overline{\text{FIRQ}}$ ,  $\overline{\text{SWI}}$ , and  $\overline{\text{RESET}}$  all set F to a one.  $\overline{\text{IRQ}}$ ,  $\overline{\text{SWI2}}$ , and  $\overline{\text{SWI3}}$  do not affect F.

**ENTIRE FLAG (E)**

Bit 7 is the entire flag, and when set to a one indicates that the complete machine state (all the registers) was stacked, as opposed to the subset state (PC and CC). The E bit of the stacked CC is used on a return from interrupt (RTI) to determine the extent of the unstacking. Therefore, the current E left in the condition code register represents past action.

**PIN DESCRIPTIONS****POWER ( $V_{SS}$ ,  $V_{CC}$ )**

Two pins are used to supply power to the part:  $V_{SS}$  is ground or 0 volts, while  $V_{CC}$  is  $+5.0\text{ V} \pm 5\%$ .

**ADDRESS BUS (A0-A15)**

Sixteen pins are used to output address information from the MPU onto the address bus. When the processor does not require the bus for a data transfer, it will output address  $\text{FFFF}_{16}$ ,  $\text{R}/\overline{\text{W}} = 1$ , and  $\text{BS} = 0$ ; this is a "dummy access" or  $\overline{\text{VMA}}$  cycle. Addresses are valid on the rising edge of Q. All address bus drivers are made high impedance when output bus available (BA) is high. Each pin will drive one Schottky TTL load or four LSTTL loads, and 90 pF.

**DATA BUS (D0-D7)**

These eight pins provide communication with the system bidirectional data bus. Each pin will drive one Schottky TTL load or four LSTTL loads, and 130 pF.

**READ/WRITE ( $\text{R}/\overline{\text{W}}$ )**

This signal indicates the direction of data transfer on the data bus. A low indicates that the MPU is writing data onto the data bus.  $\text{R}/\overline{\text{W}}$  is made high impedance when BA is high.  $\text{R}/\overline{\text{W}}$  is valid on the rising edge of Q.

 **$\overline{\text{RESET}}$** 

A low level on this Schmitt-trigger input for greater than one bus cycle will reset the MPU, as shown in Figure 6. The reset vectors are fetched from locations  $\text{FFFE}_{16}$  and  $\text{FFFF}_{16}$  (Table 1) when interrupt acknowledge is true,  $(\text{BA} \bullet \text{BS} = 1)$ . During initial power on, the  $\overline{\text{RESET}}$  line should be held low until the clock oscillator is fully operational. See Figure 7.

Because the  $\overline{\text{RESET}}$  pin has a Schmitt-trigger input with a threshold voltage higher than that of standard peripherals, a simple RC network may be used to reset the entire system. This higher threshold voltage ensures that all peripherals are out of the reset state before the processor.

 **$\overline{\text{HALT}}$** 

A low level on this input pin will cause the MPU to stop running at the end of the present instruction and remain halted indefinitely without loss of data. When halted, the BA output is driven high indicating the buses are high impedance. BS is also high which indicates the processor is in the halt or bus grant state. While halted, the MPU will not respond to external real-time requests ( $\overline{\text{FIRQ}}$ ,  $\overline{\text{IRQ}}$ ) although  $\overline{\text{DMA BREQ}}$  will always be accepted, and  $\overline{\text{NMI}}$  or  $\overline{\text{RESET}}$  will be latched for later response. During the halt state, Q and E continue to run normally. If the MPU is not running, ( $\overline{\text{RESET}}$ ,  $\overline{\text{DMA BREQ}}$ ), a halted state ( $\text{BA} \bullet \text{BS} = 1$ ) can be achieved by pulling  $\overline{\text{HALT}}$  low while  $\overline{\text{RESET}}$  is still low. If  $\overline{\text{DMA BREQ}}$  and  $\overline{\text{HALT}}$  are both pulled low, the processor will reach the last cycle of the instruction (by reverse cycle stealing) where the machine will then become halted. See Figure 8.

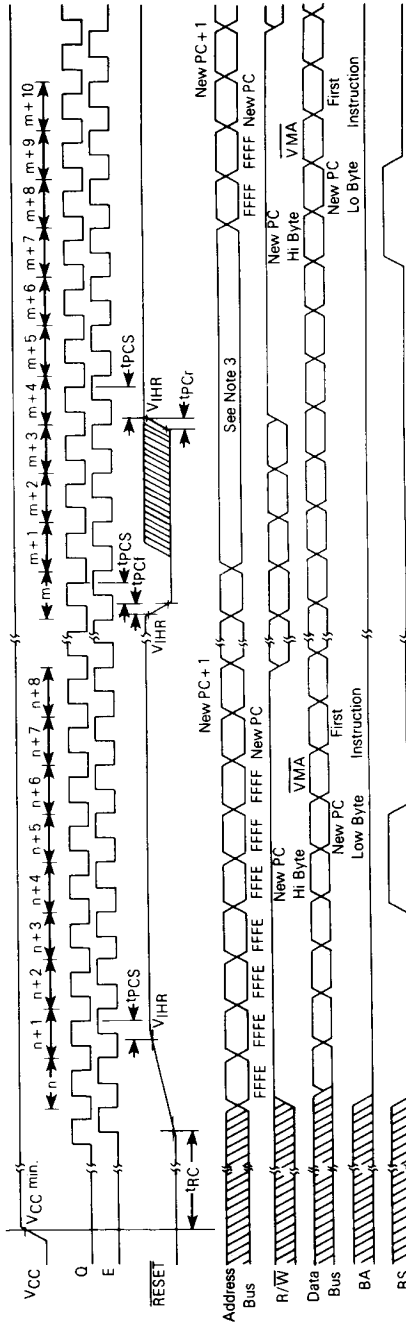
**BUS AVAILABLE, BUS STATUS (BA, BS)**

The bus available output is an indication of an internal control signal which makes the MOS buses of the MPU high impedance. This signal does not imply that the bus will be available for more than one cycle. When BA goes low, a dead cycle will elapse before the MPU acquires the bus.

The bus status output signal, when decoded with BA, represents the MPU state (valid with leading edge of Q).

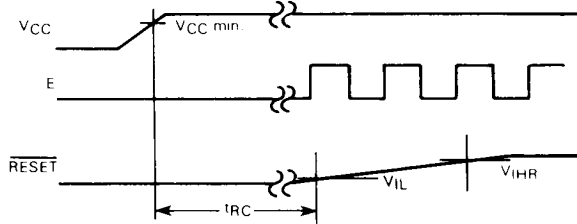
MPU State		MPU State Definition
BA	BS	
0	0	Normal (Running)
0	1	Interrupt or Reset Acknowledge
1	0	Sync Acknowledge
1	1	Halt or Bus Grant Acknowledge

FIGURE 6 - RESET TIMING



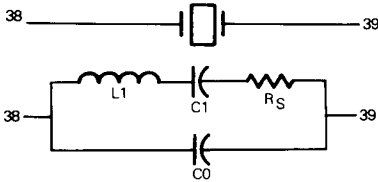
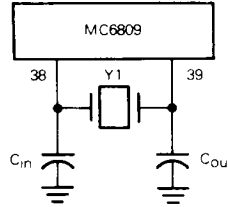
- NOTES:
1. Parts with date codes prefixed by 7F or 5A will come out of RESET one cycle sooner than shown.
  2. Timing measurements are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts, unless otherwise noted.
  3. FFFF appears on the bus during RESET low time. Following the active transition of the RESET line, three more FFFF cycles will appear followed by the vector fetch.

FIGURE 7 — CRYSTAL CONNECTIONS AND OSCILLATOR START UP



NOTE: Waveform measurements for all inputs and outputs are specified at logic high 2.0 V and logic low 0.8 V unless otherwise specified.

Y1	C <sub>in</sub>	C <sub>out</sub>
8 MHz	18 pF	18 pF
6 MHz	20 pF	20 pF
4 MHz	24 pF	24 pF



Nominal Crystal Parameters

	3.58 MHz	4.00 MHz	6.0 MHz	8.0 MHz
R <sub>S</sub>	60 Ω	50 Ω	30-50 Ω	20-40 Ω
C <sub>0</sub>	3.5 pF	6.5 pF	4-6 pF	4-6 pF
C <sub>1</sub>	0.015 pF	0.025 pF	0.01-0.02 pF	0.01-0.02 pF
Q	>40 k	>30 k	>20 k	>20 k

All parameters are 10%

NOTE: These are representative AT-cut crystal parameters only. Crystals of other types of cut may also be used.

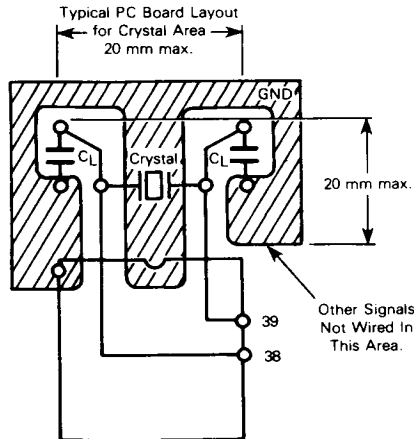
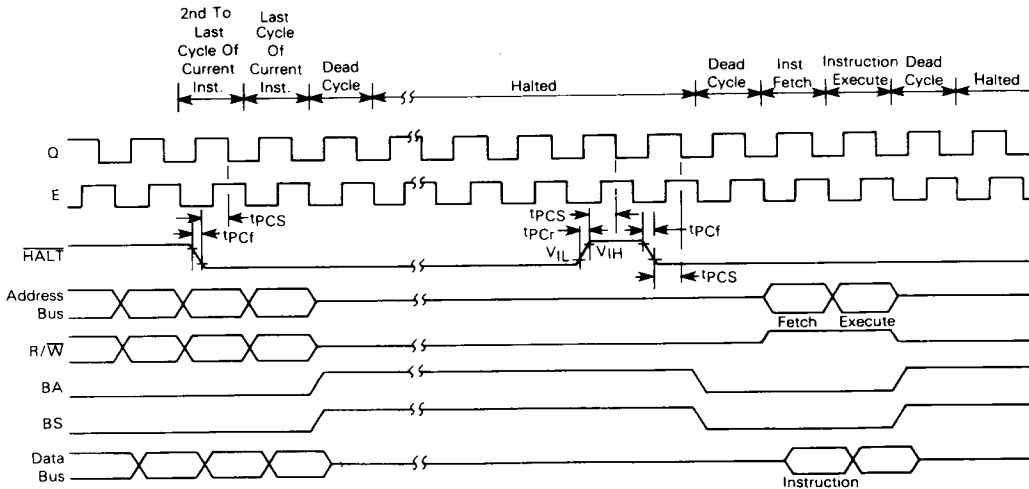




FIGURE 8 — HALT AND SINGLE INSTRUCTION EXECUTION FOR SYSTEM DEBUG



NOTE: Waveform measurements for all inputs and outputs are specified at logic high 2.0 V and logic low 0.8 V unless otherwise specified.

**INTERRUPT ACKNOWLEDGE** is indicated during both cycles of a hardware-vector-fetch (RESET, NMI, FIRQ, IRQ, SWI, SWI2, SWI3). This signal, plus decoding of the lower four address lines, can provide the user with an indication of which interrupt level is being serviced and allow vectoring by device. See Table 1.

**SYNC ACKNOWLEDGE** is indicated while the MPU is waiting for external synchronization on an interrupt line.

**HALT/BUS GRANT** is true when the MC6809 is in a halt or bus grant condition.

TABLE 1 — MEMORY MAP FOR INTERRUPT VECTORS

Memory Map For Vector Locations		Interrupt Vector Description
MS	LS	
FFFE	FFFF	RESET
FFFC	FFFD	NMI
FFFA	FFFB	SWI
FFF8	FFF9	IRQ
FFF6	FFF7	FIRQ
FFF4	FFF5	SWI2
FFF2	FFF3	SWI3
FFF0	FFF1	Reserved

**NON MASKABLE INTERRUPT (NMI)\***

A negative transition on this input requests that a non-maskable interrupt sequence be generated. A non-maskable

interrupt cannot be inhibited by the program. It also has a higher priority than FIRQ, IRQ, or software interrupts. During recognition of an NMI, the entire machine state is saved on the hardware stack. After reset, an NMI will not be recognized until the first program load of the hardware stack pointer (S). The pulse width of NMI low must be at least one E cycle. If the NMI input does not meet the minimum set up with respect to Q, the interrupt will not be recognized until the next cycle. See Figure 9.

**FAST-INTERRUPT REQUEST (FIRQ)\***

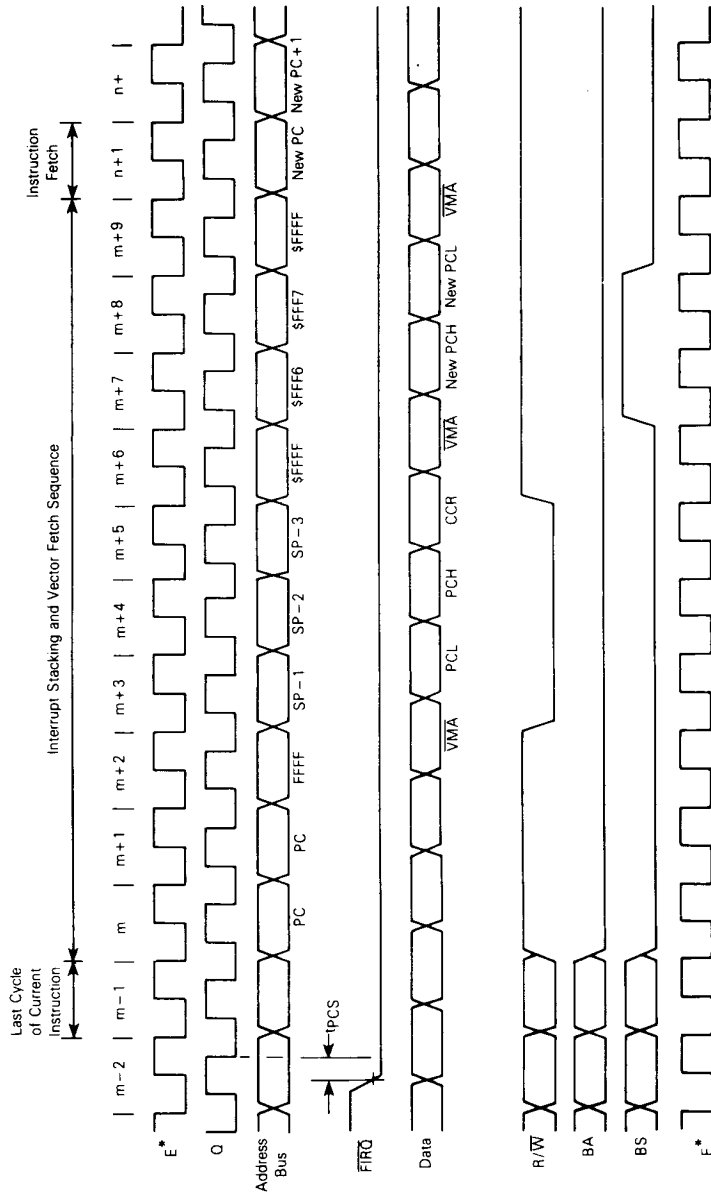
A low level on this input pin will initiate a fast interrupt sequence, provided its mask bit (F) in the CC is clear. This sequence has priority over the standard interrupt request (IRQ), and is fast in the sense that it stacks only the contents of the condition code register and the program counter. The interrupt service routine should clear the source of the interrupt before doing an RTI. See Figure 10.

**INTERRUPT REQUEST (IRQ)\***

A low level input on this pin will initiate an interrupt request sequence provided the mask bit (I) in the CC is clear. Since IRQ stacks the entire machine state, it provides a slower response to interrupts than FIRQ. IRQ also has a lower priority than FIRQ. Again, the interrupt service routine should clear the source of the interrupt before doing an RTI. See Figure 9.

\*NMI, FIRQ, and IRQ requests are sampled on the falling edge of Q. One cycle is required for synchronization before these interrupts are recognized. The pending interrupt(s) will not be serviced until completion of the current instruction unless a SYNC or CWA1 condition is present. If IRQ and FIRQ do not remain low until completion of the current instruction they may not be recognized. However, NMI is latched and need only remain low for one cycle. No interrupts are recognized or latched between the falling edge of RESET and the rising edge of BS indicating RESET acknowledge.

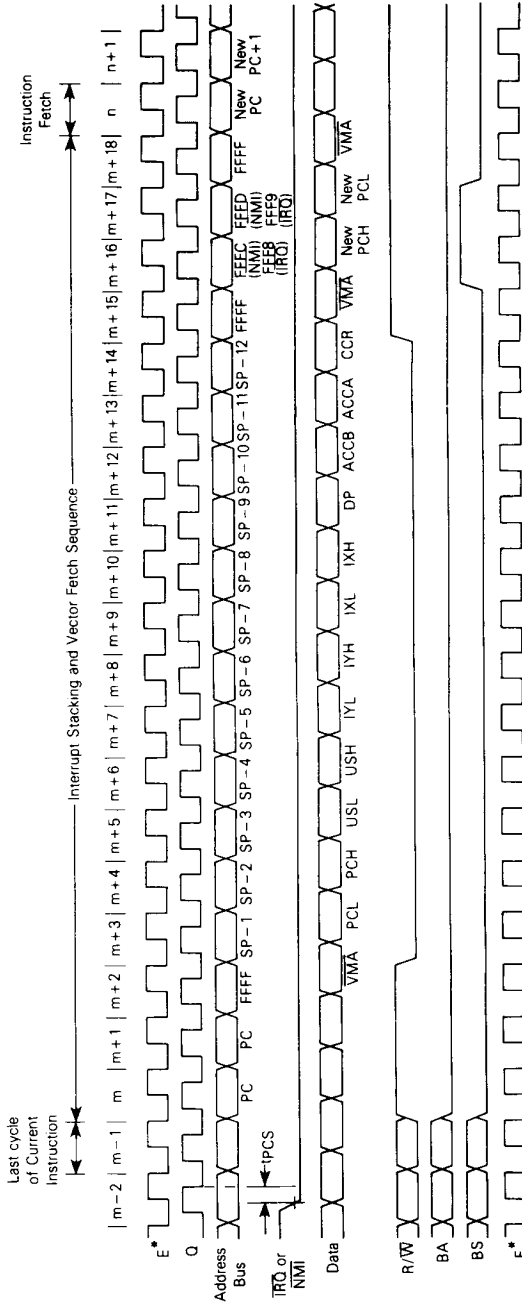
FIGURE 10 — FIRO INTERRUPT TIMING



NOTE: Waveform measurements for all inputs and outputs are specified at logic high = 2.0 V and logic low = 0.8 V unless otherwise specified.  
 \* E clock shown for reference only.

3

FIGURE 9 — IRQ AND NMI INTERRUPT TIMING



NOTE: Waveform measurements for all inputs and outputs are specified at logic high = 2.0 V and logic low = 0.8 V unless otherwise specified.

\* E clock shown for reference only.

**XTAL, EXTAL**

These inputs are used to connect the on-chip oscillator to an external parallel-resonant crystal. Alternately, the pin EXTAL may be used as a TTL level input for external timing by grounding XTAL. The crystal or external frequency is four times the bus frequency. See Figure 7. Proper RF layout techniques should be observed in the layout of printed circuit boards.

**E, Q**

E is similar to the MC6800 bus timing signal phase 2; Q is a quadrature clock signal which leads E. Q has no parallel on the MC6800. Addresses from the MPU will be valid with the leading edge of Q. Data is latched on the falling edge of E. Timing for E and Q is shown in Figure 11.

**MRDY**

This input control signal allows stretching of E and Q to extend data-access time. E and Q operate normally while MRDY is high. When MRDY is low, E and Q may be stretched in integral multiples of quarter ( $\frac{1}{4}$ ) bus cycles, thus allowing interface to slow memories, as shown in Figure 12(a). During non-valid memory access ( $\overline{\text{VMA}}$  cycles), MRDY has no effect on stretching E and Q; this inhibits slowing the processor during "don't care" bus accesses. MRDY may also be used to stretch clocks (for slow memory) when bus control has been transferred to an external device (through the use of HALT and DMA/BREQ).

**DMA/BREQ**

The DMA/BREQ input provides a method of suspending execution and acquiring the MPU bus for another use, as shown in Figure 13. Typical uses include DMA and dynamic memory refresh.

A low level on this pin will stop instruction execution at the end of the current cycle unless pre-empted by self-refresh. The MPU will acknowledge DMA/BREQ by setting BA and BS to a one. The requesting device will now have up to 15 bus cycles before the MPU retrieves the bus for self-refresh. Self-refresh requires one bus cycle with a leading and trailing dead cycle. See Figure 14. The self-refresh counter is only cleared if DMA/BREQ is inactive for two or more MPU cycles.

Typically, the DMA controller will request to use the bus by asserting DMA/BREQ pin low on the leading edge of E. When the MPU replies by setting BA and BS to a one, that cycle will be a dead cycle used to transfer bus mastership to the DMA controller.

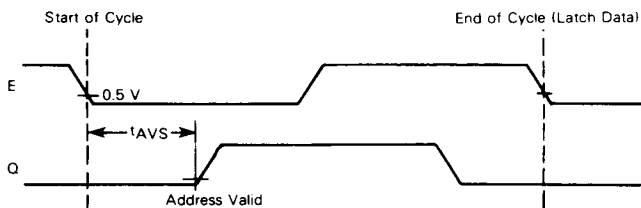
False memory accesses may be prevented during any dead cycles by developing a system  $\overline{\text{DMAVMA}}$  signal which is LOW in any cycle when BA has changed.

When BA goes low (either as a result of DMA BREQ HIGH or MPU self-refresh), the DMA device should be taken off the bus. Another dead cycle will elapse before the MPU accesses memory to allow transfer of bus mastership without contention.

**MPU OPERATION**

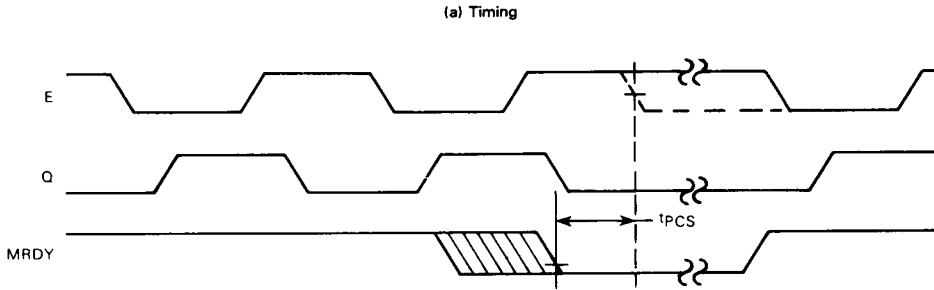
During normal operation, the MPU fetches an instruction from memory and then executes the requested function. This sequence begins after RESET and is repeated indefinitely unless altered by a special instruction or hardware occurrence. Software instructions that alter normal MPU operation are: SWI, SWI2, SWI3, CWA1, RT1, and SYNC. An interrupt, HALT, or DMA BREQ can also alter the normal execution of instructions. Figure 15 is the flowchart for the MC6809.

FIGURE 11 — E/Q RELATIONSHIP



NOTE: Waveform measurements for all inputs and outputs are specified at logic high 2.0 V and logic low 0.8 V unless otherwise specified.

FIGURE 12 — MRDY TIMING AND SYNCHRONIZATION



(b) Synchronization

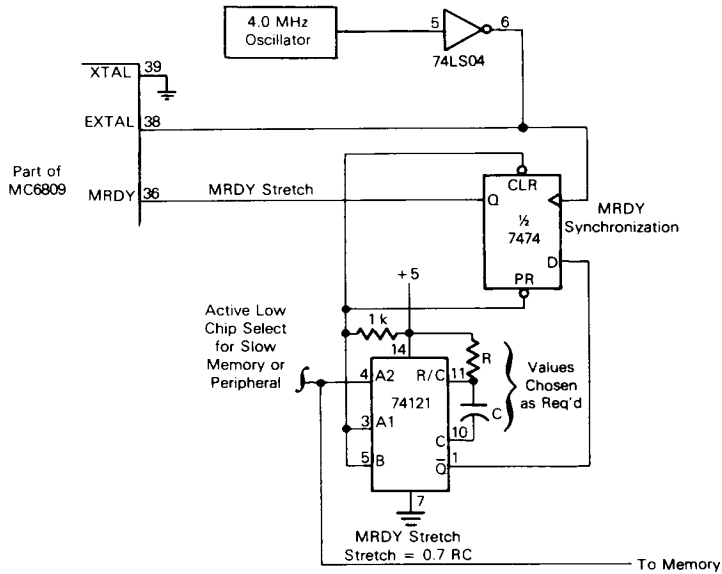


FIGURE 13 — TYPICAL DMA TIMING (< 14 CYCLES)

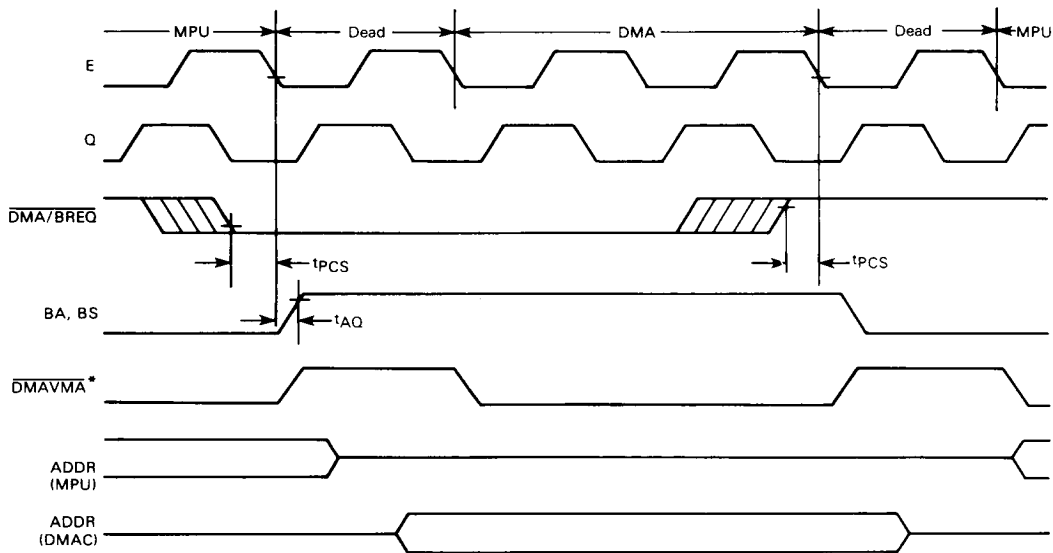
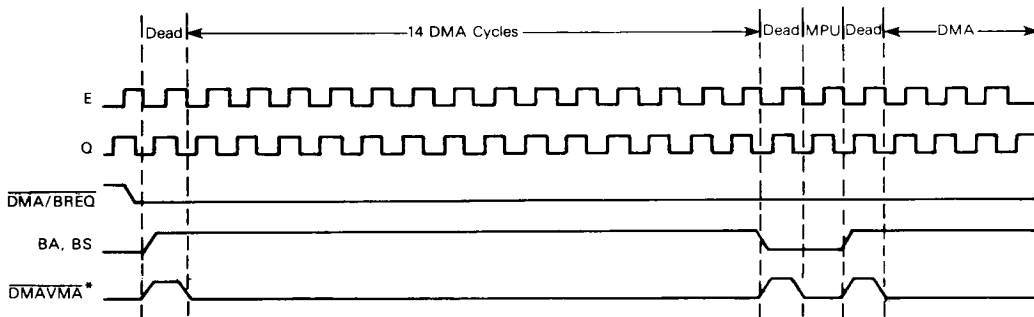


FIGURE 14 — AUTO-REFRESH DMA TIMING (> 14 CYCLES)  
(REVERSE CYCLE STEALING)



\* DMAVMA is a signal which is developed externally, but is a system requirement for DMA. Refer to Application Note AN-820.

NOTE: Waveform measurements for all inputs and outputs are specified at logic high 2.0 V and logic low 0.8 V unless otherwise specified.



## ADDRESSING MODES

The basic instructions of any computer are greatly enhanced by the presence of powerful addressing modes. The MC6809 has the most complete set of addressing modes available on any microcomputer. For example, the MC6809 has 59 basic instructions; however, it recognizes 1464 different variations of instructions and addressing modes. The addressing modes support modern programming techniques. The following addressing modes are available on the MC6809:

- Inherent (Includes Accumulator)
- Immediate
- Extended
  - Extended Indirect
- Direct
- Register
- Indexed
  - Zero-Offset
  - Constant Offset
  - Accumulator Offset
  - Auto Increment/Decrement
  - Indexed Indirect
- Relative
  - Short/Long Relative Branching
  - Program Counter Relative Addressing

**INHERENT (INCLUDES ACCUMULATOR)**

In this addressing mode, the opcode of the instruction contains all the address information necessary. Examples of inherent addressing are: ABX, DAA, SWI, ASRA, and CLRB.

**IMMEDIATE ADDRESSING**

In immediate addressing, the effective address of the data is the location immediately following the opcode (i.e., the data to be used in the instruction immediately following the opcode of the instruction). The MC6809 uses both 8- and 16-bit immediate values depending on the size of argument specified by the opcode. Examples of instructions with immediate addressing are:

```
LDA #$20
LDX #$F000
LDY #CAT
```

**NOTE**

# signifies immediate addressing; \$ signifies hexadecimal value.

**EXTENDED ADDRESSING**

In extended addressing, the contents of the two bytes immediately following the opcode fully specify the 16-bit effective address used by the instruction. Note that the address generated by an extended instruction defines an absolute address and is not position independent. Examples of extended addressing include:

```
LDA CAT
STX MOUSE
LDD $2000
```

**EXTENDED INDIRECT** — As in the special case of indexed addressing (discussed below), one level of indirection may be added to extended addressing. In extended indirect, the two bytes following the postbyte of an indexed instruction contain the address of the data.

```
LDA [CAT]
LDX [$FFFE]
STU [DOG]
```

**DIRECT ADDRESSING**

Direct addressing is similar to extended addressing except that only one byte of address follows the opcode. This byte specifies the lower eight bits of the address to be used. The upper eight bits of the address are supplied by the direct page register. Since only one byte of address is required in direct addressing, this mode requires less memory and executes faster than extended addressing. Of course, only 256 locations (one page) can be accessed without redefining the contents of the DP register. Since the DP register is set to \$00 on reset, direct addressing on the MC6809 is compatible with direct addressing on the M6800. Indirection is not allowed in direct addressing. Some examples of direct addressing are:

```
LDA $30
SETDP $10 (assembler directive)
LDB $1030
LDD < CAT
```

**NOTE**

< is an assembler directive which forces direct addressing.

**REGISTER ADDRESSING**

Some opcodes are followed by a byte that defines a register or set of registers to be used by the instruction. This is called a postbyte. Some examples of register addressing are:

TFR	X, Y	Transfers X into Y
EXG	A, B	Exchanges A with B
PSHS	A, B, X, Y	Push Y, X, B and A onto S
PULU	X, Y, D	Pull D, X, and Y from U

**INDEXED ADDRESSING**

In all indexed addressing, one of the pointer registers (X, Y, U, S, and sometimes PC) is used in a calculation of the effective address of the operand to be used by the instruction. Five basic types of indexing are available and are discussed below. The postbyte of an indexed instruction specifies the basic type and variation of the addressing mode as well as the pointer register to be used. Figure 16 lists the legal formats for the postbyte. Table 2 gives the assembler form and the number of cycles and bytes added to the basic values for indexed addressing for each variation.



FIGURE 16 — INDEXED ADDRESSING POSTBYTE REGISTER BIT ASSIGNMENTS

Postbyte Register Bit								Indexed Addressing Mode
7	6	5	4	3	2	1	0	
0	R	R	d	d	d	d	d	EA = ,R + 5 Bit Offset
1	R	R	0	0	0	0	0	,R +
1	R	R	i	0	0	0	1	,R + +
1	R	R	0	0	0	1	0	, - R
1	R	R	i	0	0	1	1	, - - R
1	R	R	i	0	1	0	0	EA = ,R + 0 Offset
1	R	R	i	0	1	0	1	EA = ,R + ACCB Offset
1	R	R	i	0	1	1	0	EA = ,R + ACCA Offset
1	R	R	i	1	0	0	0	EA = ,R + 8 Bit Offset
1	R	R	i	1	0	0	1	EA = ,R + 16 Bit Offset
1	R	R	i	1	0	1	1	EA = ,R + D Offset
1	x	x	i	1	1	0	0	EA = ,PC + 8 Bit Offset
1	x	x	i	1	1	0	1	EA = ,PC + 16 Bit Offset
1	R	R	i	1	1	1	1	EA = [,Address]

Addressing Mode Field

Indirect Field  
(Sign bit when b7 = 0)

Register Field: RR

x = Don't Care  
d = Offset Bit  
i = 0 = Not Indirect  
1 = Indirect

00 = X  
01 = Y  
10 = U  
11 = S

**ZERO-OFFSET INDEXED** — In this mode, the selected pointer register contains the effective address of the data to be used by the instruction. This is the fastest indexing mode.

Examples are:  
LDD O, X  
LDA ,S

**CONSTANT OFFSET INDEXED** — In this mode, a two's-complement offset and the contents of one of the pointer registers are added to form the effective address of the operand. The pointer register's initial content is unchanged by the addition. Three sizes of offset are available:

- 5-bit (-16 to -15)
- 8-bit (-128 to +128)
- 16-bit (-32768 to +32767)

The two's complement 5-bit offset is included in the postbyte and, therefore, is most efficient in use of bytes and cycles. The two's complement 8-bit offset is contained in a single byte following the postbyte. The two's complement 16-bit offset is in the two bytes following the postbyte. In most cases the programmer need not be concerned with the size of this offset since the assembler will select the optimal size automatically.

Examples of constant-offset indexing are:  
LDA 23,X  
LDX -2,S  
LDY 300,X  
LDU CAT,Y

3

TABLE 2 — INDEXED ADDRESSING MODE

Type	Forms	Non Indirect		+	+	Indirect		+	+
		Assembler Form	Postbyte Opcode			Assembler Form	Postbyte Opcode		
Constant Offset From R	No Offset	,R	1RR00100	0	0	[,R]	1RR10100	3	0
(2s Complement Offsets)	5-Bit Offset	n, R	ORRnnnnn	1	0	defaults to 8-bit			
	8-Bit Offset	n, R	1RR01000	1	1	[n, R]	1RR11000	4	1
	16-Bit Offset	n, R	1RR01001	4	2	[n, R]	1RR11001	7	2
Accumulator Offset From R (2s Complement Offsets)	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 (2s Complement Offsets)	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  
RR: 00 = X, 01 = Y, 10 = U, 11 = S  
x = Don't Care

+ and # indicate the number of additional cycles and bytes for the particular variation.

**ACCUMULATOR-OFFSET INDEXED** — This mode is similar to constant offset indexed except that the two's-complement value in one of the accumulators (A, B, or D) and the contents of one of the pointer registers are added to form the effective address of the operand. The contents of both the accumulator and the pointer register are unchanged by the addition. The postbyte specifies which accumulator to use as an offset and no additional bytes are required. The advantage of an accumulator offset is that the value of the offset can be calculated by a program at run-time.

Some examples are:

```
LDA B,Y
LDX D,Y
LEAX B,X
```

**AUTO INCREMENT/DECREMENT INDEXED** — In the auto increment addressing mode, the pointer register contains the address of the operand. Then, after the pointer register is used it is incremented by one or two. This addressing mode is useful in stepping through tables, moving data, or for the creation of software stacks. In auto decrement, the pointer register is decremented prior to use as the address of the data. The use of auto decrement is similar to that of auto increment; but the tables, etc., are scanned from the high to low addresses. The size of the increment/decrement can be either one or two to allow for tables of either 8- or 16-bit data to be accessed and is selectable by the programmer. The pre-decrement, post-increment nature of these modes allows them to be used to create additional software stacks that behave identically to the U and S stacks.

Some examples of the auto increment/decrement addressing modes are:

```
LDA ,X+
STD ,Y++
LDB ,-Y
LDX ,--S
```

Care should be taken in performing operations on 16-bit pointer registers (X, Y, U, S) where the same register is used to calculate the effective address.

Consider the following instruction:

```
STX 0,X++ (X initialized to 0)
```

The desired result is to store zero in locations \$0000 and \$0001, then increment X to point to \$0002. In reality, the following occurs:

```
0→temp      calculate the EA; temp is a holding register
X+2→X       perform auto increment
X→(temp)    do store operation
```

**INDEXED INDIRECT** — All of the indexing modes, with the exception of auto increment/decrement by one or a  $\pm 4$ -bit offset, may have an additional level of indirection specified. In indirect addressing, the effective address is contained at the location specified by the contents of the index register plus any offset. In the example below, the A accumulator is loaded indirectly using an effective address calculated from the index register and an offset.

Before Execution

A = XX (don't care)

X = \$F000

```
$0100 LDA [$10,X] EA is now $F010
```

```
$F010 $F1          $F150 is now the
$F011 $50          new EA
```

```
$F150 $AA
```

After Execution

A = \$AA Actual Data Loaded

X = \$F000

All modes of indexed indirect are included except those which are meaningless (e.g., auto increment/decrement by one indirect). Some examples of indexed indirect are:

```
LDA [,X]
LDD [10,S]
LDA [B,Y]
LDD [,X++]
```

### RELATIVE ADDRESSING

The byte(s) following the branch opcode is (are) treated as a signed offset which may be added to the program counter. If the branch condition is true, then the calculated address (PC + signed offset) is loaded into the program counter. Program execution continues at the new location as indicated by the PC; short (one byte offset) and long (two bytes offset) relative addressing modes are available. All of memory can be reached in long relative addressing as an effective address is interpreted modulo  $2^{16}$ . Some examples of relative addressing are:

```
BEQ CAT (short)
BGT DOG (short)
CAT LBEG RAT (long)
DOG LBGT RABBIT (long)
•
•
•
RAT NOP
RABBIT NOP
```

**PROGRAM COUNTER RELATIVE** — The PC can be used as the pointer register with 8- or 16-bit signed offsets. As in relative addressing, the offset is added to the current PC to create the effective address. The effective address is then used as the address of the operand or data. Program counter relative addressing is used for writing position independent programs. Tables related to a particular routine will maintain the same relationship after the routine is moved, if referenced relative to the program counter. Examples are:

```
LDA CAT, PCR
LEAX TABLE, PCR
```

Since program counter relative is a type of indexing, an additional level of indirection is available.

```
LDA [CAT, PCR]
LDU [DOG, PCR]
```

**INSTRUCTION SET**

The instruction set of the MC6809 is similar to that of the MC6800 and is upward compatible at the source code level. The number of opcodes has been reduced from 72 to 59, but because of the expanded architecture and additional addressing modes, the number of available opcodes (with different addressing modes) has risen from 197 to 1464.

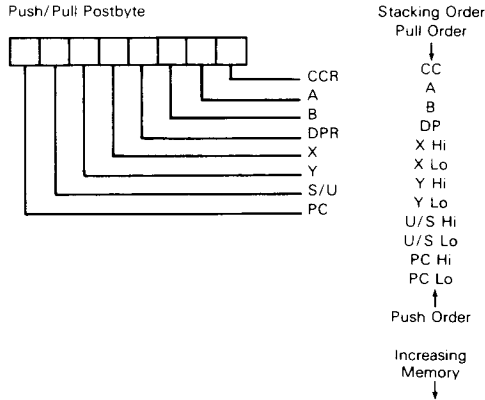
Some of the new instructions are described in detail below.

**PSHU/PSHS**

The push instructions have the capability of pushing onto either the hardware stack (S) or user stack (U) any single register or set of registers with a single instruction.

**PULU/PULS**

The pull instructions have the same capability of the push instruction, in reverse order. The byte immediately following the push or pull opcode determines which register or registers are to be pushed or pulled. The actual push/pull sequence is fixed; each bit defines a unique register to push or pull, as shown below.



**TFR/EXG**

Within the MC6809, any register may be transferred to or exchanged with another or like size, i.e., 8 bit to 8 bit or 16 bit to 16 bit. Bits 4-7 of post byte define the source register, while bits 0-3 represent the destination register. These are denoted as follows:

Transfer/Exchange Postbyte	
Source	Destination
Register Field	
0000 = D (A:B)	1000 = A
0001 = X	1001 = B
0010 = Y	1010 = CCR
0011 = U	1011 = DPR
0100 = S	
0101 = PC	

**NOTE**

All other combinations are undefined and INVALID.

**LEAX/LEAY/LEAU/LEAS**

The LEA (load effective address) works by calculating the effective address used in an indexed instruction and stores that address value, rather than the data at that address, in a pointer register. This makes all the features of the internal addressing hardware available to the programmer. Some of the implications of this instruction are illustrated in Table 3.

The LEA instruction also allows the user to access data and tables in a position independent manner. For example:

```

LEAX  MSG1, PCR
LBSR  PDATA (print message routine)
.
.
MSG1  FCC  'MESSAGE'
```

This sample program prints: 'MESSAGE'. By writing MSG1, PCR, the assembler computes the distance between the present address and MSG1. This result is placed as a constant into the LEAX instruction which will be indexed from the PC value at the time of execution. No matter where the code is located when it is executed, the computed offset from the PC will put the absolute address of MSG1 into the X pointer register. This code is totally position independent.

The LEA instructions are very powerful and use an internal holding register (temp). Care must be exercised when using the LEA instructions with the auto increment and auto decrement addressing modes due to the sequence of internal operations. The LEA internal sequence is outlined as follows:

LEAa ,b+ (any of the 16-bit pointer registers X, Y, U, or S may be substituted for a and b)

1. b → temp (calculate the EA)
2. b + 1 → b (modify b, postincrement)
3. temp → a (load a)

LEAa , - b

1. b - 1 → temp (calculate EA with predecrement)
2. b - 1 → b (modify b, predecrement)
3. temp → a (load a)

**TABLE 3 — LEA EXAMPLES**

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 A 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



Auto increment-by-two and auto decrement-by-two instructions work similarly. Note that LEAX ,X+ does not change X; however, LEAX, -X does decrement; LEAX 1, X should be used to increment X by one.

**MUL**

Multiplies the unsigned binary numbers in the A and B accumulator and places the unsigned result into the 16-bit D accumulator. The unsigned multiply also allows multiple-precision multiplications.

**LONG AND SHORT RELATIVE BRANCHES**

The MC6809 has the capability of program counter relative branching throughout the entire memory map. In this mode, if the branch is to be taken, the 8- or 16-bit signed offset is added to the value of the program counter to be used as the effective address. This allows the program to branch anywhere in the 64K memory map. Position-independent code can be easily generated through the use of relative branching. Both short (8-bit) and long (16-bit) branches are available.

**SYNC**

After encountering a sync instruction, the MPU enters a sync state, stops processing instructions, and waits for an interrupt. If the pending interrupt is non-maskable (NMI) or maskable (FIRQ, IREQ) with its mask bit (F or I) clear, the processor will clear the sync state and perform the normal interrupt stacking and service routine. Since FIRQ and IREQ are not edge-triggered, a low level with a minimum duration of three bus cycles is required to assure that the interrupt will be taken. If the pending interrupt is maskable (FIRQ, IREQ) with its mask bit (F or I) set, the processor will clear the sync state and continue processing by executing the next in-line instruction. Figure 18 depicts sync timing.

**SOFTWARE INTERRUPTS**

A software interrupt is an instruction which will cause an interrupt and its associated vector fetch. These software interrupts are useful in operating system calls, software debugging, trace operations, memory mapping, and software development systems. Three levels of SWI are available on the MC6809, and are prioritized in the following order: SWI, SWI2, SWI3.

**16-BIT OPERATION**

The MC6809 has the capability of processing 16-bit data. These instructions include loads, stores, compares, adds, subtracts, transfers, exchanges, pushes, and pulls.

**CYCLE-BY-CYCLE OPERATION**

The address bus cycle-by-cycle performance chart (Figure 18) illustrates the memory-access sequence corresponding to each possible instruction and addressing mode in the MC6809. Each instruction begins with an opcode fetch. While that opcode is being internally decoded, the next program byte is always fetched. (Most instructions will use the next byte, so this technique considerably speeds throughput.) Next, the operation of each opcode will follow the flowchart. VMA is an indication of FFFF<sub>16</sub> on the address bus, R/W = 1 and BS = 0. The following examples illustrate the use of the chart.

**Example 1: LBSR (Branch Taken)**  
Before Execution SP = F000

```

      •
      •
      •
$8000  LBSR  CAT
      •
      •
      •
$A000  CAT  •
    
```

**CYCLE-BY-CYCLE FLOW**

Cycle #	Address	Data	R/W	Description
1	8000	17	1	Opcode Fetch
2	8001	20	1	Offset High Byte
3	8002	00	1	Offset Low Byte
4	FFFF	*	1	VMA Cycle
5	FFFF	*	1	VMA Cycle
6	A000	*	1	Computed Branch Address
7	FFFF	*	1	VMA Cycle
8	EEEE	80	0	Stack High Order Byte of Return Address
9	EFEE	03	0	Stack Low Order Byte of Return Address

**Example 2: DEC (Extended)**

```

$8000  DEC  $A000
      •
      •
      •
$A8000  $80
    
```

**CYCLE-BY-CYCLE FLOW**

Cycle #	Address	Data	R/W	Description
1	8000	7A	1	Opcode Fetch
2	8001	A0	1	Operand Address, High Byte
3	8002	00	1	Operand Address, Low Byte
4	FFFF	*	1	VMA Cycle
5	A000	80	1	Read the Data
6	FFFF	*	1	VMA Cycle
7	A000	7F	0	Store the Decrement Data

\* The data bus has the data at that particular address.

**INSTRUCTION SET TABLES**

The instructions of the MC6809 have been broken down into five different categories. They are as follows:

- 8-bit operation (Table 4)
- 16-bit operation (Table 5)
- Index register/stack pointer instructions (Table 6)
- Relative branches (long or short) (Table 7)
- Miscellaneous instructions (Table 8)

Hexadecimal values for the instructions are given in Table 9.

**PROGRAMMING AID**

Figure 19 contains a compilation of data that will assist in programming the MC6809.

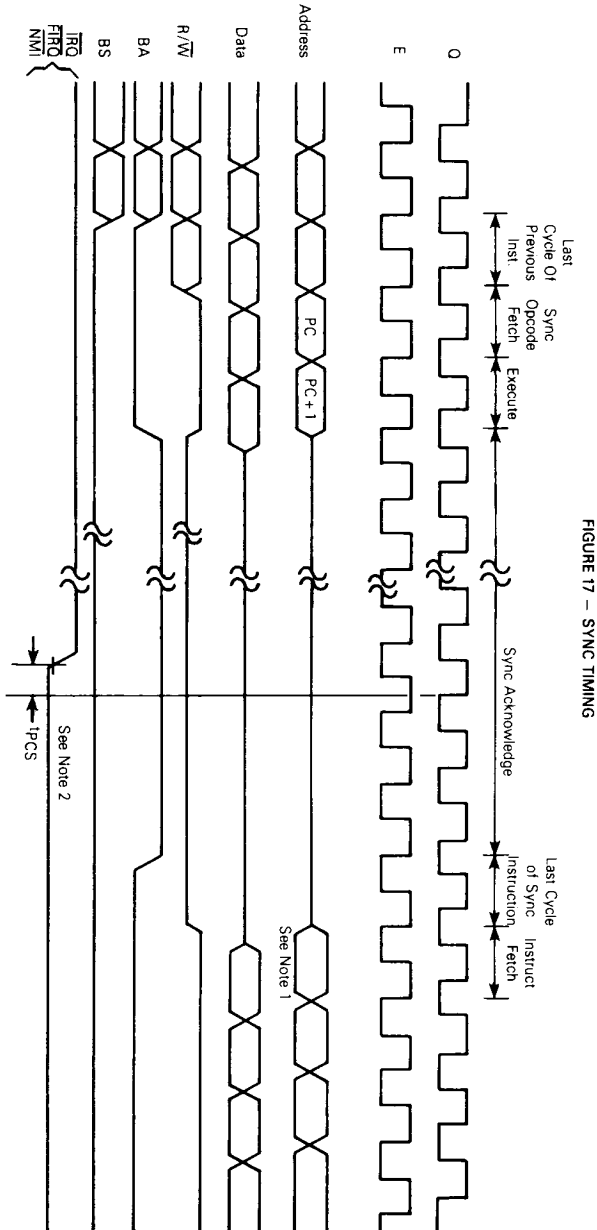


FIGURE 17 — SYNC TIMING

- NOTES:
1. If the associated mask bit is set when the interrupt is requested, this cycle will be an instruction fetch from address location PC+1. However, if the interrupt is accepted (NMI or an unmasked FRO or FRI) interrupt processing continues with this cycle as in on Figures 9 and 10 (Interrupt Timing).
  2. If mask bits are clear, FRO and FRI must be held low for three cycles to guarantee interrupt to be taken, although only one cycle is necessary to bring the processor out of SYNC.
  3. Waveform measurements for all inputs and outputs are specified at logic high 2.0 V and logic low 0.8 V unless otherwise specified.



FIGURE 18 — CYCLE-BY-CYCLE PERFORMANCE (Sheet 2 of 5)

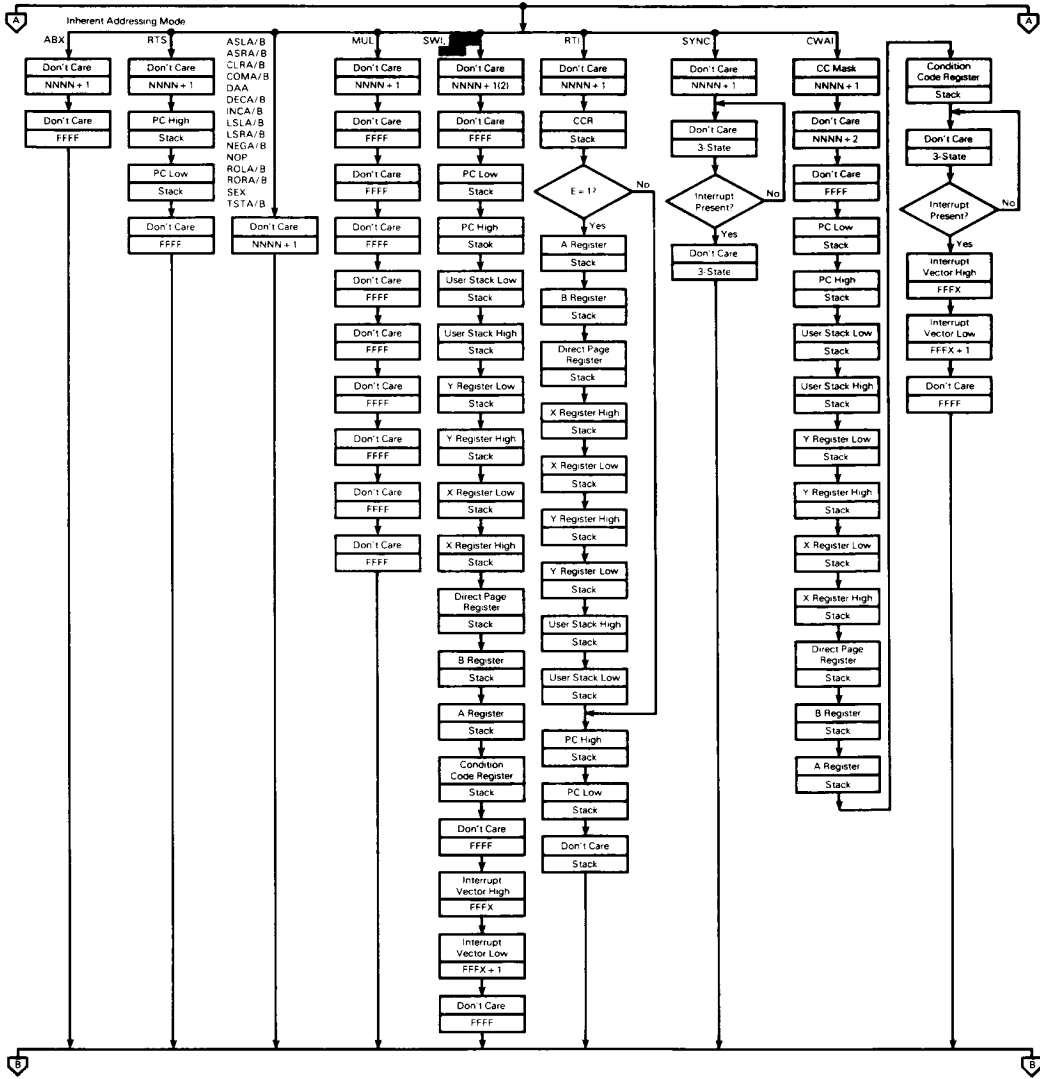
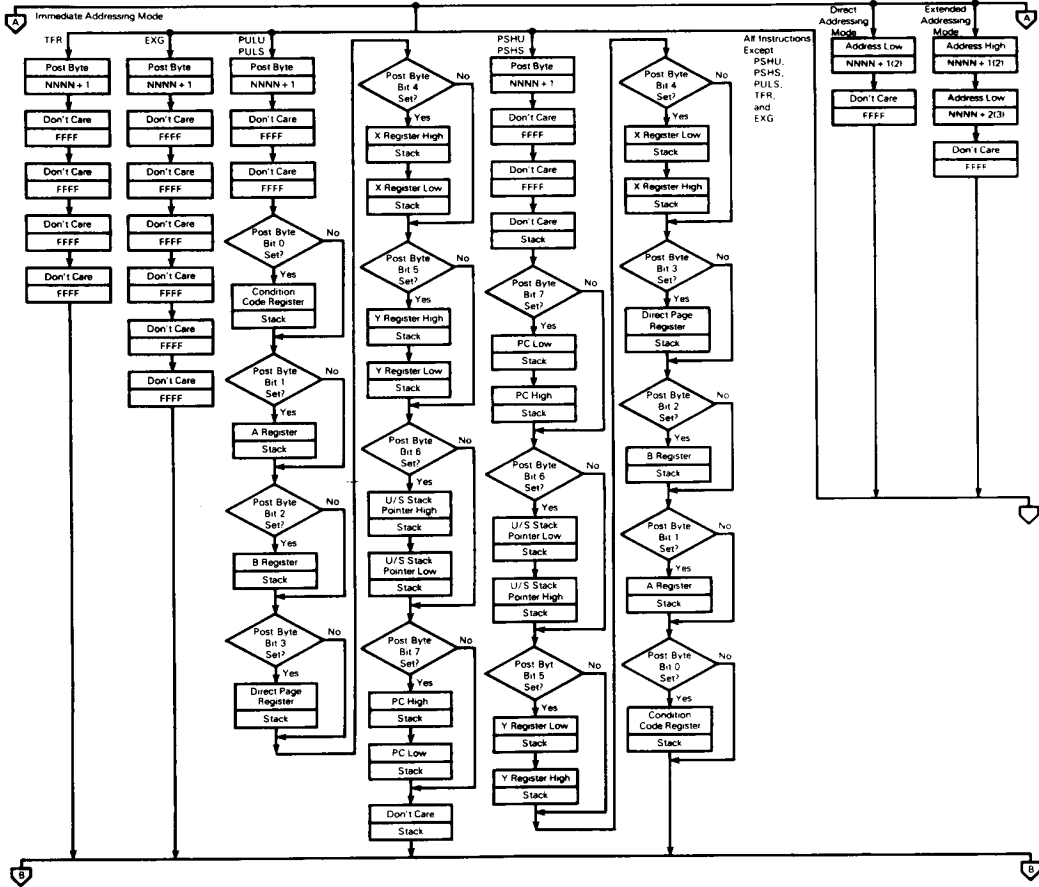


FIGURE 18 — CYCLE-BY-CYCLE PERFORMANCE (Sheet 3 of 5)



3



FIGURE 18 — CYCLE-BY-CYCLE PERFORMANCE (Sheet 4 of 5)

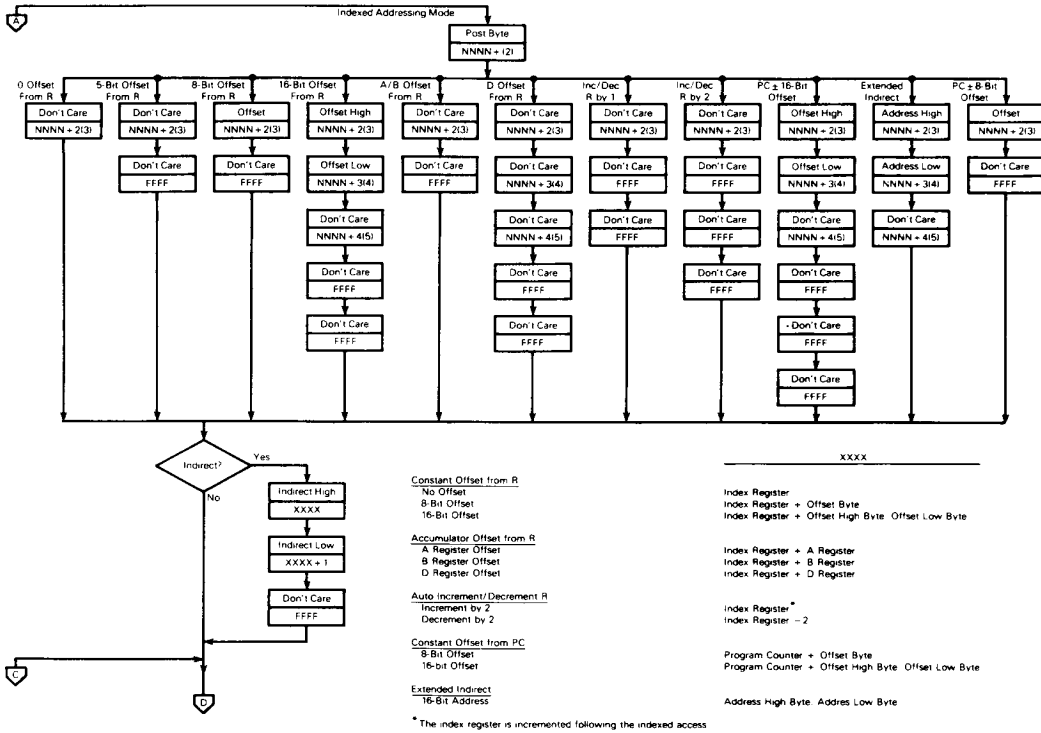
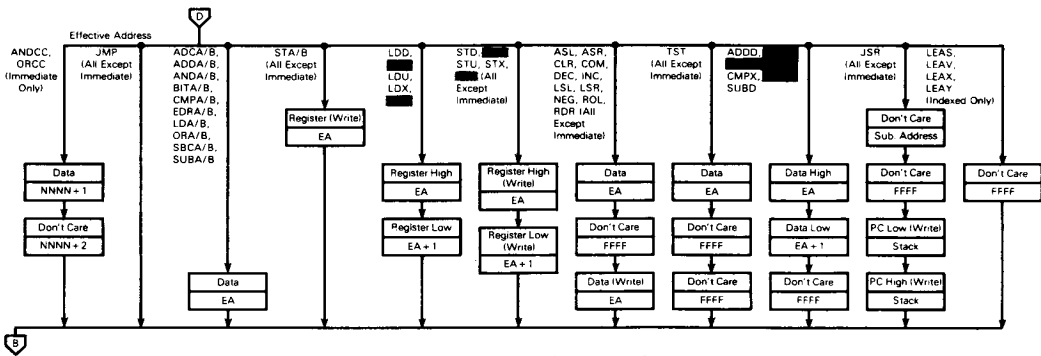


FIGURE 18 -- CYCLE-BY-CYCLE PERFORMANCE (Sheet 5 of 5)



Effective Address (EA)

- Constant Offset from R
    - No Offset
    - 5-Bit Offset
    - 8-Bit Offset
    - 16-Bit Offset
  - Accumulator Offset from R
    - A Register Offset
    - B Register Offset
    - D Register Offset
  - Auto Increment/Decrement R
    - Increment by 1
    - Increment by 2
    - Decrement by 1
    - Decrement by 2
  - Constant Offset from PC
    - 8-Bit Offset
    - 16-Bit Offset
  - Direct
  - Extended
  - Immediate
- Index Register
  - Index Register
  - Index Register + Post Byte
  - Index Register + Post Byte High Post Byte Low
  - Index Register + A Register
  - Index Register + B Register
  - Index Register + D Register
  - Index Register\*
  - Index Register
  - Index Register - 1
  - Index Register - 2
  - Program Counter + Offset Byte
  - Program Counter + Offset High Byte Offset Low Byte
  - Direct Page Register Address Low
  - Address High Address Low
  - NNNN + 1

\* The index register is incremented following the indexed access

3

TABLE 4 — 8-BIT ACCUMULATOR AND MEMORY INSTRUCTIONS

Mnemonic(s)	Operation
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 memory
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) stack with either PSHS, PSHU (PULS, PULU) instructions.

TABLE 5 — 16-BIT ACCUMULATOR AND MEMORY INSTRUCTIONS

Mnemonic(s)	Operation
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 stack with either PSHS, PSHU (PULS, PULU) instructions.

TABLE 6 — INDEX REGISTER/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, X, 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, S, 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 7 — BRANCH INSTRUCTIONS

Instruction	Description
<b>SIMPLE BRANCHES</b>	
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BMI, LBMI	Branch if minus
BPL, LBPL	Branch if plus
BCS, LBSCS	Branch if carry set
BCC, LBCC	Branch if carry clear
BVS, LBVS	Branch if overflow set
BVC, LBVC	Branch if overflow clear
<b>SIGNED BRANCHES</b>	
BGT, LBGT	Branch if greater (signed)
BVS, LBVS	Branch if invalid 2s 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 2s complement result
BLT, LBLT	Branch if less than (signed)
<b>UNSIGNED BRANCHES</b>	
BHI, LBHI	Branch if higher (unsigned)
BCC, LBCC	Branch if higher or same (unsigned)
BHS, LBHS	Branch if higher or same (unsigned)
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BLS, LBLS	Branch if lower or same (unsigned)
BCS, LBSCS	Branch if lower (unsigned)
BLO, LBLO	Branch if lower (unsigned)
<b>OTHER BRANCHES</b>	
BSR, LBSR	Branch to subroutine
BRA, LBRA	Branch always
BRN, LBRN	Branch never

TABLE 8 — 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

TABLE 9 — HEXADECIMAL VALUES OF MACHINE CODES

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	*	↑	6	2	31	LEAY	Indexed	4+	2+	61	*	↑	6+	2+
02	*				32	LEAS		4+	2+					
03	COM	↓	6	2	33	LEAU	Indexed	4+	2+	62	*	↓	6+	2+
04	LSR				34	PSHS		5+	2					
05	*	↑	6	2	35	PULS	Immed	5+	2	63	COM	↑	6+	2+
06	ROR				36	PSHU		5+	2					
07	ASR	↓	6	2	37	PULU	Immed	5+	2	64	LSR	↓	6+	2+
08	ASL, LSL				38	*		65	*					
09	ROL	↑	6	2	39	RTS	Inherent	5	1	66	ROR	↑	6+	2+
0A	DEC				3A	ABX		3	1					
0B	*	↓	6	2	3B	RTI	↑	6/15	1	67	ASR	↓	6+	2+
0C	INC				3C	CWAI		≥20	2					
0D	TST	↑	6	2	3D	MUL	Inherent	11	1	68	ASL, LSL	↑	6+	2+
0E	JMP				3E	*		69	ROL					
0F	CLR	Direct	6	2	3F	SWI	Inherent	19	1	6A	DEC	↓	6+	2+
10	Page 2	—	—	—	40	NEGA	↑	2	1	70	NEG	↑	7	3
11	Page 3	—	—	—	41	*		—	—	—	71			
12	NOP	Inherent	2	1	42	*	↑	2	1	72	*	↑	7	3
13	SYNC	Inherent	≥4	1	43	COMA		2	1	73	COM			
14	*	↓	5	3	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			
18	*	↑	2	1	48	ASLA, LSLA	↑	2	1	78	ASL, LSL	↑	7	3
19	DAA				49	ROLA		2	1					
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			
1E	EXG	Immed	8	2	4E	*	↑	2	1	7E	JMP	↑	4	3
1F	TFR	Immed	6	2	4F	CLRA		Inherent	2	1	7F			
20	BRA	↑	3	2	50	NEGB	↑	2	1	80	SUBA	↑	2	2
21	BRN				51	*		81	CMPA					
22	BHI	↓	3	2	52	*	↑	2	1	82	SBCA	↓	2	2
23	BLS				53	COMB		83	SUBD					
24	BHS, BCC	↑	3	2	54	LSRB	↑	2	1	84	ANDA	↓	2	2
25	BLO, BCS				55	*		85	BITA					
26	BNE	↓	3	2	56	RORB	↑	2	1	86	LDA	↑	2	2
27	BEQ				57	ASRB		87	*					
28	BVC	↑	3	2	58	ASLB, LSLB	↑	2	1	88	EORA	↓	2	2
29	BVS				59	ROLB		89	ADCA					
2A	BPL	↓	3	2	5A	DECB	↑	2	1	8A	ORA	↑	2	2
2B	BMI				5B	*		8B	ADDA					
2C	BGE	↑	3	2	5C	INCB	↑	2	1	8C	CMPX	↓	4	3
2D	BLT				5D	TSTB		8D	BSR					
2E	BGT	↓	3	2	5E	*	↑	2	1	8E	LDX	↑	7	2
2F	BLE				Relative	3		2	5F	CLRB	Inherent			

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



TABLE 9 — HEXADECIMAL VALUES OF MACHINE CODES (CONTINUED)

OP	Mnem	Mode	~	#	OP	Mnem	Mode	~	#	OP	Mnem	Mode	~	#				
90	SUBA	Direct ↑ ↓	4	2	C0	SUBB	Immed ↑ ↓	2	2	Page 2 and 3 Machine Codes ↓ Relative Immed ↓ Inherent Immed ↓ LDY Direct ↓ Direct Indexed ↓ Indexed Extended ↑ Extended Immed Direct Direct Indexed Extended Extended Indexed Extended Extended					5	4		
91	CMPA		4	2	C1	CMPB		2	2						1021	LB RN	5(6)	4
92	SBCA		4	2	C2	SBCB		2	2						1022	LBHI	5(6)	4
93	SUBD		6	2	C3	ADDD		4	3						1023	LBLS	5(6)	4
94	ANDA		4	2	C4	ANDB		2	2						1024	LBHS, LBCC	5(6)	4
95	BITA		4	2	C5	BITB		2	2						1025	LB CS, LBLO	5(6)	4
96	LDA		4	2	C6	LDB		2	2						1026	LBNE	5(6)	4
97	STA		4	2	C7	*		2	2						1027	LB EQ	5(6)	4
98	EORA		4	2	C8	EORB		2	2						1028	LBVC	5(6)	4
99	ADCA		4	2	C9	ADCB		2	2						1029	LBVS	5(6)	4
9A	ORA	4	2	CA	ORB	2	2	102A	LBPL	5(6)	4							
9B	ADDA	4	2	CB	ADDB	2	2	102B	LBMI	5(6)	4							
9C	CMPX	6	2	CC	LDD	3	3	102C	LBGE	5(6)	4							
9D	JSR	7	2	CD	*	3	3	102D	LB LT	5(6)	4							
9E	LDX	5	2	CE	LDU	3	3	102E	LBGT	5(6)	4							
9F	STX	5	2	CF	*	3	3	102F	LBLE	5(6)	4							
A0	SUBA	Indexed ↑ ↓	4+	2+	D0	SUBB	Direct ↑ ↓	4	2	1030	SWI2	20	2					
A1	CMPA		4+	2+	D1	CMPB		4	2	1031	CMPD	5	4					
A2	SBCA		4+	2+	D2	SBCB		4	2	1032	CMPY	5	4					
A3	SUBD		6+	2+	D3	ADDD		6	2	1033	CMPY	7	3					
A4	ANDA		4+	2+	D4	ANDB		4	2	1034	CMPY	7	3					
A5	BITA		4+	2+	D5	BITB		4	2	1035	CMPY	6	3					
A6	LDA		4+	2+	D6	LDB		4	2	1036	CMPY	7	3					
A7	STA		4+	2+	D7	STB		4	2	1037	CMPY	6	3					
A8	EORA		4+	2+	D8	EORB		4	2	1038	CMPY	7	3					
A9	ADCA		4+	2+	D9	ADCB		4	2	1039	CMPY	6	3					
AA	ORA		4+	2+	DA	ORB		4	2	103A	CMPD	7	3					
AB	ADDA		4+	2+	DB	ADDB		4	2	103B	CMPD	6	3					
AC	CMPX		6+	2+	DC	LDD		5	2	103C	CMPY	7	3					
AD	JSR		7+	2+	DD	STD		5	2	103D	CMPY	6	3					
AE	LDX		5+	2+	DE	LDU		5	2	103E	CMPD	8	4					
AF	STX		5+	2+	DF	STU		5	2	103F	CMPY	8	4					
B0	SUBA	Extended ↑ ↓	5	3	E0	SUBB	Indexed ↑ ↓	4+	2+	1040	LDY	7	4					
B1	CMPA		5	3	E1	CMPB		4+	2+	1041	STY	7	4					
B2	SBCA		5	3	E2	SBCB		4+	2+	1042	LDS	4	3					
B3	SUBD		7	3	E3	ADDD		6+	2+	1043	LDS	6	3					
B4	ANDA		5	3	E4	ANDB		4+	2+	1044	STS	6	3					
B5	BITA		5	3	E5	BITB		4+	2+	1045	LDS	6	3					
B6	LDA		5	3	E6	LDB		4+	2+	1046	STS	6	3					
B7	STA		5	3	E7	STB		4+	2+	1047	STS	7	4					
B8	EORA		5	3	E8	EORB		4+	2+	1048	STS	7	4					
B9	ADCA		5	3	E9	ADCB		4+	2+	1049	STS	7	4					
BA	ORA		5	3	EA	ORB		4+	2+	104A	STS	7	4					
BB	ADDA		5	3	EB	ADDB		4+	2+	104B	STS	7	4					
BC	CMPX		7	3	EC	LDD		5+	2+	104C	STS	7	4					
BD	JSR		8	3	ED	STD		5+	2+	104D	STS	7	4					
BE	LDX		6	3	EE	LDU		5+	2+	104E	STS	7	4					
BF	STX		6	3	EF	STU		5+	2+	104F	STS	7	4					
NOTE: All unused opcodes are both undefined and illegal					F0	SUBB	Extended ↑ ↓	5	3	1050	CMPU	5	4					
					F1	CMPB		5	3	1051	CMPU	5	4					
					F2	SBCB		5	3	1052	CMPU	5	4					
					F3	ADDD		7	3	1053	CMPU	7	3					
					F4	ANDB		5	3	1054	CMPU	5	3					
					F5	BITB		5	3	1055	CMPU	5	3					
					F6	LDB		5	3	1056	CMPU	5	3					
					F7	STB		5	3	1057	CMPU	5	3					
					F8	EORB		5	3	1058	CMPU	5	3					
					F9	ADCB		5	3	1059	CMPU	5	3					
					FA	ORB		5	3	105A	CMPU	5	3					
					FB	ADDB		5	3	105B	CMPU	5	3					
					FC	LDD		6	3	105C	CMPU	6	3					
					FD	STD		6	3	105D	CMPU	6	3					
					FE	LDU		6	3	105E	CMPU	6	3					
					FF	STU		6	3	105F	CMPU	6	3					

3

MC6809

FIGURE 19 — PROGRAMMING AID

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						
ADC	ADCA ADCB	89 C9	2	2	99 D9	4	2	A9 E9	4+	2+	B9 F9	5	3									
ADD	ADDA ADDB ADDD	8B CB C3	2	2	9B DB D3	4	2	AB EB E3	4+	2+	BB FB F3	5	3									
AND	ANDA ANDB ANDCC	84 C4 1C	2	2	94 D4	4	2	A4 E4	4+	2+	B4 F4	5	3									
ASL	ASLA ASLB ASL				08	6	2	68	6+	2+	78	7	3									
ASR	ASRA ASRB ASR				07	6	2	67	6+	2+	77	7	3									
BIT	BITA BITB	85 C5	2	2	95 D5	4	2	A5 E5	4+	2+	B5 F5	5	3									
CLR	CLRA CLRB CLR				0F	6	2	6F	6+	2+	7F	7	3									
CMP	CMPA CMPB CMPD CMPS CMPU CMPX CMPY	81 C1 10 83 11 8C 11 83 8C 10 8C	2	2	91 D1 10 93 11 9C 11 93 9C 10 9C	4	2	A1 E1 10 A3 11 AC 11 A3 AC 10 AC	4+	2+	B1 F1 10 B3 11 BC 11 B3 BC 10 BC	5	3									
COM	COMA COMB COM				03	6	2	63	6+	2+	73	7	3									
CWAI		3C	≥20	2																		
DAA														19	2	1						
DEC	DECA DECB DEC				0A	6	2	6A	6+	2+	7A	7	3									
EOR	EORA EORB	88 C8	2	2	98 D8	4	2	A8 E8	4+	2+	B8 F8	5	3									
EXG	R1, R2	1E	8	2																		
INC	INCA INCB INC				0C	6	2	6C	6+	2+	7C	7	3									
JMP					0E	3	2	6E	3+	2+	7E	4	3									
JSR					9D	7	2	AD	7+	2+	BD	8	3									
LD	LDA LDB LDD LDS CE LDU LDX LDY	86 C6 CC 10 CE 8E 10 8E	2	2	96 D6 DC 10 DE 9E 5 9E	4	2	A6 E6 5 10 EE AE 5 AE	4+	2+	B6 F6 6 7 FE 6 BE 10 BE	5	3									
LEA	LEAS LEAU LEAX LEAY							32 33 30 31	4+	2+												

LEGEND:

OP	Operation Code (Hexadecimal)	M	Complement of M	!	Test and set if true, cleared otherwise
~	Number of MPU Cycles	-	Transfer Into	•	Not Affected
#	Number of Program Bytes	H	Half-carry (from bit 3)	CC	Condition Code Register
+	Arithmetic Plus	N	Negative (sign bit)	:	Concatenation
-	Arithmetic Minus	Z	Zero result	∨	Logical or
•	Multiply	V	Overflow, 2's complement	∧	Logical and
		C	Carry from ALU	⊕	Logical Exclusive or



FIGURE 19 — PROGRAMMING AID (CONTINUED)

Instruction	Forms	Addressing Modes												Description	5	3	2	1	0								
		Immediate			Direct			Indexed <sup>1</sup>			Extended									Inherent							
		Op	-	#	Op	-	#	Op	-	#	Op	-	#							Op	-	#	H	N	Z	V	C
LSL	LSLA														48	2	1						•	•	•	•	•
	LSLB LSL				08	6	2	68	6+	2+	78	7	3	58	2	1						•	•	•	•	•	
LSR	LSRA														44	2	1						•	•	•	•	•
	LSRB LSR				04	6	2	64	6+	2+	74	7	3	54	2	1						•	•	•	•	•	
MUL															3D	11	1	A × B → D (Unsigned)					•	•	•	•	9
NEG	NEGA														40	2	1	A ← 1 - A					•	•	•	•	•
	NEGB NEG				00	6	2	60	6+	2+	70	7	3	50	2	1	B ← 1 - B M ← 1 - M					•	•	•	•	•	
NOP															12	2	1	No Operation					•	•	•	•	•
OR	ORA	8A	2	2	9A	4	2	AA	4+	2+	BA	5	3				A V M ← A					•	•	•	•	•	
	ORB	CA	2	2	DA	4	2	EA	4+	2+	FA	5	3				B V M ← B					•	•	•	•	•	
	ORCC	1A	3	2														CC V IMM ← CC					•	•	•	•	7
PSH	PSHS	34	5+	4														Push Registers on S Stack					•	•	•	•	•
	PSHU	36	5+	4														Push Registers on U Stack					•	•	•	•	•
PUL	PULS	35	5+	4														Pull Registers from S Stack					•	•	•	•	•
	PULU	37	5+	4														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 ROR				06	6	2	66	6+	2+	76	7	3	56	2	1						•	•	•	•	•	
RTI															3B	6-15	1	Return From Interrupt					•	•	•	•	•
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					•	•	•	•	•	
	SBCB	C2	2	2	D2	4	2	E2	4+	2+	F2	5	3				B ← M - C → B					•	•	•	•	•	
SEX															1D	2	1	Sign Extend Bit into A					•	•	•	•	•
ST	STA				97	4	2	A7	4+	2+	B7	5	3				A ← M					•	•	•	•	•	
	STB				D7	4	2	E7	4+	2+	F7	5	3				B ← M					•	•	•	•	•	
	STD				DD	5	2	ED	5+	2+	FD	6	3				D ← M M + 1					•	•	•	•	•	
	STS				10	6	3	10	6+	3+	10	7	4				S ← M M + 1					•	•	•	•	•	
	STU				DF			EF			FF						U ← M M + 1					•	•	•	•	•	
	STX				DF	5	2	EF	5+	2+	FF	6	3				X ← M M + 1					•	•	•	•	•	
	STY				9F	5	2	AF	5+	2+	BF	6	3				Y ← M M + 1					•	•	•	•	•	
					9F	10	6	3	10	6+	3+	BF	7	4									•	•	•	•	•
SUB	SUBA	80	2	2	90	4	2	A0	4+	2+	B0	5	3				A ← M - A					•	•	•	•	•	
	SUBB	C0	2	2	D0	4	2	E0	4+	2+	F0	5	3				B ← M - B					•	•	•	•	•	
	SUBD	83	4	3	93	6	2	A3	6+	2+	B3	7	3				D ← M M + 1 → D					•	•	•	•	•	
SWI	SWI <sup>6</sup>														3F	19	1	Software Interrupt 1					•	•	•	•	•
	SWI <sup>26</sup>														10	20	2	Software Interrupt 2					•	•	•	•	•
	SWI <sup>36</sup>														3F								•	•	•	•	•
															11	20	1	Software Interrupt 3					•	•	•	•	•
SYNC														13	≥ 4	1	Synchronize to Interrupt					•	•	•	•	•	
TFR	R1, R2	1F	6	2													R1 ← R2 <sup>2</sup>					•	•	•	•	•	
TST	TSTA														4D	2	1	Test A					•	•	•	•	•
	TSTB				0D	6	2	6D	6+	2+	7D	7	3	5D	2	1	Test B					•	•	•	•	•	
	TST																	Test M					•	•	•	•	•

NOTES:

- This column gives a base cycle and byte count. To obtain total count, add the values obtained from the INDEXED ADDRESSING MODE table, Table 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
- EA is the effective address.
- The PSH and PUL instructions require 5 cycles plus 1 cycle for each byte pushed or pulled.
- 5(6) means: 5 cycles if branch not taken, 6 cycles if taken (Branch instructions).
- SWI sets I and F bits. SWI2 and SWI3 do not affect I and F.
- Conditions Codes set as a direct result of the instruction.
- Value of half-carry flag is undefined.
- Special Case — Carry set if b7 is SET.



FIGURE 19 – PROGRAMMING AID (CONTINUED)

Branch Instructions

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

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

SIMPLE BRANCHES

	OP	~	#
BRA	20	3	2
LBRA	16	5	3
BRN	21	3	2
LB RN	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 ≥ m	BGE	2C	BLT	2D
r = m	BEQ	27	BNE	26
r ≤ m	BLE	2F	BGT	2E
r < m	BLT	2D	BGE	2C

UNSIGNED CONDITIONAL BRANCHES (Notes 1-4)

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

NOTES:

1. All conditional branches have both short and long variations.
2. All short branches are two bytes and require three 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 four bytes and six cycles if the branch is taken or five cycles if the branch is not taken.

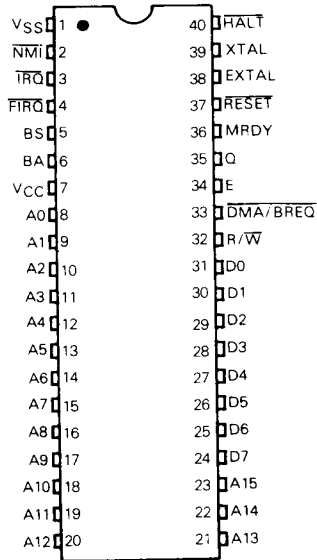


# MC6809

## ORDERING INFORMATION

Package Type	Frequency	Temperature Range	Order Number
Plastic P Suffix	1.0 MHz	0°C to 70°C	MC6809P
	1.0 MHz	-40°C to 85°C	MC6809CP
	1.5 MHz	0°C to 70°C	MC68A09P
	1.5 MHz	-40°C to 85°C	MC68A09CP
	2.0 MHz	0°C to 70°C	MC68B09P
	2.0 MHz	-40°C to 85°C	MC68B09CP
Cerdip S Suffix	1.0 MHz	0°C to 70°C	MC6809S
	1.0 MHz	-40°C to 85°C	MC6809CS
	1.5 MHz	0°C to 70°C	MC68A09S
	1.5 MHz	-40°C to 85°C	MC68A09CS
	2.0 MHz	0°C to 70°C	MC68B09S
	2.0 MHz	-40°C to 85°C	MC68B09CS

## PIN ASSIGNMENT



MOTOROLA MICROPROCESSOR DATA

3-1396