

## EF6809

### 8-BIT MICROPROCESSING UNIT

The EF6809 is a revolutionary, high performance 8-bit microprocessor which supports modern programming techniques such as position independence, reentrancy, and modular programming.

This third generation addition to the 6800 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 EF6809 has the most complete set of addressing modes available on any 8-bit microprocessor today.

The EF6809 has hardware and software features which make it an ideal processor for higher level language execution or standard controller applications.

#### EF6800 COMPATIBLE

- Hardware — Interfaces with All 6800 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 =  $4 \times f$ )
- DMA/BREQ Allows DMA Operation on Memory Refresh
- Fast Interrupt Request Input Stacks Only Condition Code Register and Program Counter
- 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 RESET
- Single 5-Volt Supply Operation
- NMI Inhibited After 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

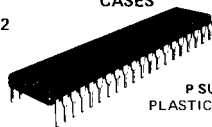
### HMOS

(HIGH DENSITY N-CHANNEL, SILICON-GATE)

### 8-BIT MICROPROCESSING UNIT

#### CASES

CB-182



P SUFFIX  
PLASTIC PACKAGE

ALSO AVAILABLE

J SUFFIX  
CERDIP PACKAGE

C SUFFIX  
CERAMIC PACKAGE

CB-521



FN SUFFIX  
PLCC 44

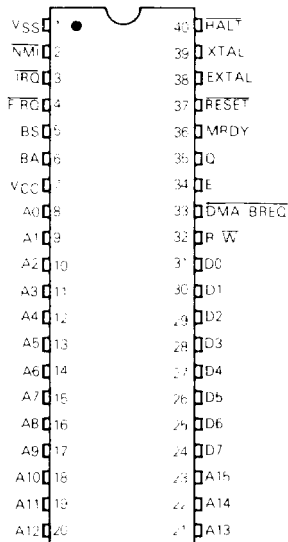
CB-708



E SUFFIX  
LCCC 44

Hi-Rel versions available - See chapter 9

#### PIN ASSIGNMENT



## MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	$V_{CC}$	5.0	V
Input Voltage	$V_{IH}$	5.0	V
Operating Temperature Range	$T_A$	-40 to +125	°C
Storage Temperature Range	$T_{STG}$	-65 to +125	°C

## THERMAL CHARACTERISTICS

Characteristic	Symbol	Value	Unit
Thermal Resistance Circuit	$\theta_{JA}$	50	°C/W
Ceramic		60	
Plastic		100	

## POWER CONSIDERATIONS

The average chip junction temperature,  $T_{J(AV)}$ , in °C can be obtained from

$$T_{J(AV)} = T_A + (P_D) \cdot \theta_{JA} \quad (1)$$

Where:

$T_A$  = Ambient Temperature, °C

$\theta_{JA}$  = Package Thermal Resistance, Junction to Ambient, °C/W

$P_D$  =  $P_{INT} + P_{PORT}$

$P_{INT}$  =  $I_{CC} \times V_{CC}$ , Watts = Chip Internal Power

$P_{PORT}$  =  $P_{PORT}$  Power Dissipation, Watts = User Determined

For most applications  $P_{PORT} \ll P_{INT}$  and can be neglected.  $P_{PORT}$  may become significant if the device is configured to drive Darlington bases or sink LED loads.

An approximate relationship between  $P_D$  and  $T_{J(AV)}$  if  $P_{PORT}$  is neglected is:

$$P_D = K - (T_{J(AV)} + 273) \quad (2)$$

Solving equations 1 and 2 for K gives:

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

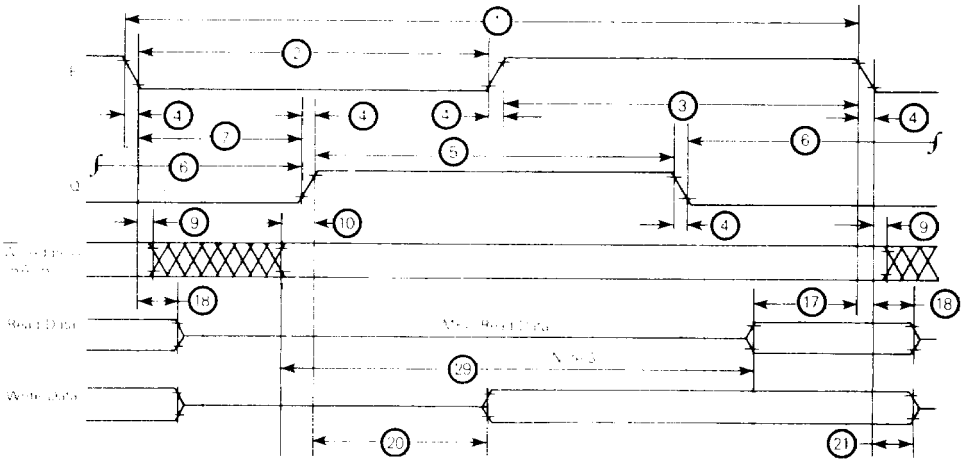
Where K is a constant pertaining to the particular part. K can be determined from equation 3 by measuring  $P_D$  (at equilibrium) for a known  $T_A$ . Using this value of K the values of  $P_D$  and  $T_{J(AV)}$  can be obtained by solving equations (1) and (2) iteratively for any value of  $T_A$ .

ELECTRICAL CHARACTERISTICS ( $V_{CC} = 5.0 \text{ V} \pm 5\%$ ,  $V_{SS} = 0$ ,  $T_A = T_U$  to  $T_H$  unless otherwise noted)

Characteristic	Symbol	Min	Typ	Max	Unit
Input High Voltage Logic, EXTA, RESET	$V_{IH}$	$V_{SS} + 2.0$	—	$V_{CC}$	V
Input Low Voltage Logic, EXTA, RESET	$V_{IL}$	$V_{SS} - 0.3$	—	$V_{SS} + 0.8$	V
Input Leakage Current ( $V_{in} = 0$ to 5.25 V, $V_{CC} = \text{max}$ )	$I_{in}$	—	—	2.5	$\mu\text{A}$
dc Output High Voltage ( $I_{Load} = -205 \mu\text{A}$ , $V_{CC} = \text{max}$ ) ( $I_{Load} = -145 \mu\text{A}$ , $V_{CC} = \text{min}$ ) ( $I_{Load} = -100 \mu\text{A}$ , $V_{CC} = \text{min}$ )	$V_{OH}$	$V_{SS} + 2.4$ $V_{SS} + 2.4$ $V_{SS} + 2.4$	—	—	V
dc Output Low Voltage ( $I_{Load} = 2.0 \text{ mA}$ , $V_{CC} = \text{min}$ )	$V_{OL}$	—	—	$V_{SS} + 0.5$	V
Internal Power Dissipation (Measured at $T_A = 0^\circ\text{C}$ in Steady State Operation)	$P_{INT}$	—	—	1.0	W
Capacitance* ( $V_{in} = 0$ , $T_A = 25^\circ\text{C}$ , $f = 1.0 \text{ MHz}$ )	$C_{in}$	—	10 10	15 15	pF
Frequency of Operation (Crystal or External Input)	$f_{XTAL}$	0.4 0.4 0.4	—	4 6 8	MHz
Hi-Z (Off State) Input Current ( $V_{in} = 0.4$ to 2.4 V, $V_{CC} = \text{max}$ )	$I_{SI}$	—	2.5	10 100	$\mu\text{A}$

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

1. The driver can sink or source current from the output without damage due to the internal diodes of the output transistors. However, it is advised that the maximum sink current be kept to avoid application of any voltage greater than maximum rated voltages. For this reason, impedance should be applied to the peripheral enhanced if unused inputs are tied to an appropriate logic voltage level, e.g., either  $V_{SS}$  or  $V_{CC}$ .



## BUS TIMING CHARACTERISTICS (See Note 1)

Ident. Number	Characteristics	Symbol	EF6809		EF68A09		EF68B09		Unit
			Min	Max	Min	Max	Min	Max	
1	Address Hold Time (See Note 1)	$t_{AH}$	10	10	0.667	10	0.5	10	$\mu$ s
2	Bus Setup Time	$t_{BUS}$	450	500	290	5000	210	5000	ns
3	Bus Hold Time	$t_{BHD}$	450	15500	290	15700	220	15700	ns
4	Bus Rise and Fall Time	$t_{BR}$		25		25		20	ns
5	Delay Time to Data	$t_{D}$	430	5000	280	5000	210	5000	ns
6	Delay Time to Data	$t_{D}$	450	15500	290	15700	220	15700	ns
7	Delay Time to Data	$t_{D}$	200	250	130	165	80	125	ns
8	Address Hold Time (See Note 1)	$t_{AH}$	20		20		20		ns
9	BA, BS, RW, and Address Valid to RD/WR	$t_{AV}$	90		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
25	Data Delay Time from RD/WR	$t_{DDQ}$		200		140		110	ns
27	Write Data Hold Time*	$t_{DHW}$	30		30		30		ns
28	Write Address Valid to RD/WR	$t_{AVC}$	660		440		330		ns
29	Processor Control Setup Time (MPDY, DMA, BREQ, TRST, RES*) (Equation 6 and 7)	$t_{PCS}$	20		140		110		ns
30	Crystal Oscillator Start Time (Equation 8)	$t_{OSC}$		100		100		100	ms
31	Processor Control Rise and Fall Time (Equation 9)	$t_{PC}$		100		100		100	ns

\* Address and data hold times are performed a setup after the  $t_{D}$  period.

## NOTES:

1. V<sub>CC</sub> and V<sub>DD</sub> levels shown are V<sub>CC</sub> ≤ 0.4 V, V<sub>DD</sub> ≥ 2.4 V, unless otherwise specified.
2. Measurement points shown are 0.6 V and 2.0 V, unless otherwise specified.
3. Usable access times computed by  $t_{1-4} \cdot t_{max} + 10 \cdot t_{1-7}$ .
4. Read time  $t_{D}$  for BA and BS is not specified.
5. Maximum delay during MPDY or DMA\_BREQ is 18  $\mu$ s.

FIGURE 2 – EF6809 EXPANDED BLOCK DIAGRAM

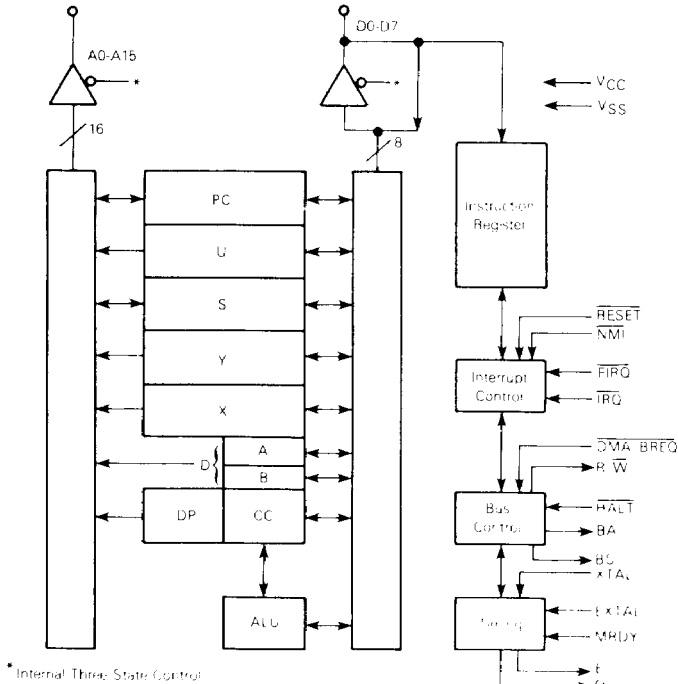
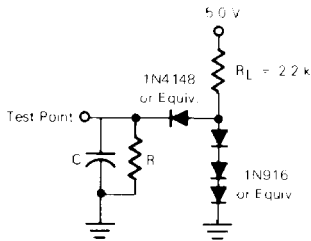


FIGURE 3 – BUS TIMING TEST LOAD



C = 30 pF for BA, BS  
130 pF for D0-D7, E, Q  
90 pF for A0-A15, R/W

R = 11.7 k $\Omega$  for D0-D7  
16.5 k $\Omega$  for A0-A15, E, Q, R/W  
24 k $\Omega$  for BA, BS

### PROGRAMMING MODEL

As shown in Figure 4, the EJ6809 adds three registers to the set available in the EF6800. The added registers include a direct page register, the user stack pointer, and a second index register.

### ACCUMULATORS (A, B, D)

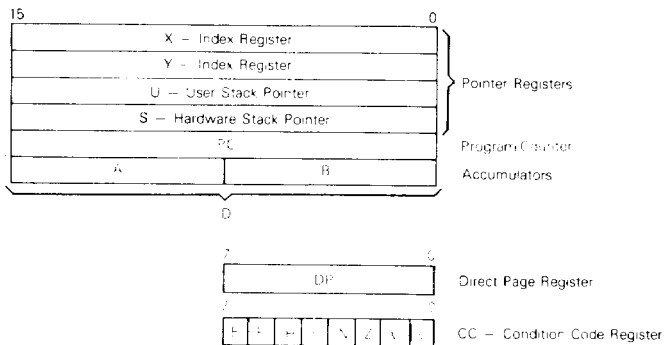
The A and B registers are general purpose accumulators which are used for arithmetic calculations and manipulation of data.

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 EF6809 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 6800 compatibility, all bits of this register are cleared during processor reset.

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 EF6809 point to the top of the stack, in contrast to the EF6800 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 EF6809 to be used efficiently as a stack processor, greatly enhancing its ability to support higher level languages and modular programming.

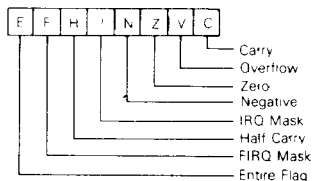
**PROGRAM COUNTER**

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**

**BIT 0 (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.

**BIT 1 (V)**

Bit 1 is the overflow flag, and is set to a one by an operation which causes a signed two's 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.

**BIT 2 (Z)**

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

**BIT 3 (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 two's-complement result will leave N set to a one.

**BIT 4 (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. NMI,  $\overline{\text{FIRQ}}$ ,  $\overline{\text{IRQ}}$ ,  $\overline{\text{RESET}}$ , and SWI1 all set I to a one. SWI2 and SWI3 do not affect I.

**BIT 5 (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.

**BIT 6 (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. NMI,  $\overline{\text{FIRQ}}$ , SWI1, and  $\overline{\text{RESET}}$  all set F to a one.  $\overline{\text{IRQ}}$ , SWI2, and SWI3 do not affect F.

**BIT 7 (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<sub>SS</sub>, V<sub>CC</sub>)**

Two pins are used to supply power to the part: V<sub>SS</sub> is ground or 0 volts, while V<sub>CC</sub> is +5.0 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 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 (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. R/ $\overline{\text{W}}$  is made high impedance when BA is high. R/ $\overline{\text{W}}$  is valid on the rising edge of Q.

**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  $\text{FFF}_{16}$  and  $\text{FFF}_{16}$  (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 EF6809  $\overline{\text{RESET}}$  pin has a Schmitt-trigger input with a threshold voltage higher than that of standard peripherals, a simple R-C 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.

**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 DMA/ $\overline{\text{BREQ}}$  will always be accepted, and 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 ( $\overline{\text{RESET}}$ , DMA/ $\overline{\text{BREQ}}$ ), 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 DMA/ $\overline{\text{BREQ}}$  and  $\overline{\text{HALT}}$  are both pulled low, the processor will reach the last cycle of the instruction (by reverse cycle stealing) when the machine will become halted. See Figure 8.

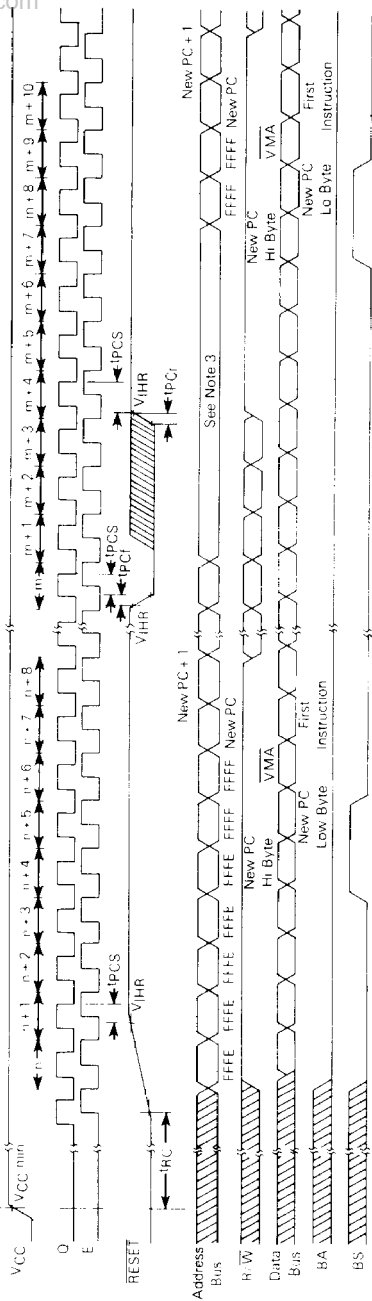
**BUS AVAILABLE, BUS STATUS (BA, BS)**

The bus available output is an indication of a status 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 combined with BA, represents the MPU state (valid with leading edge of Q).

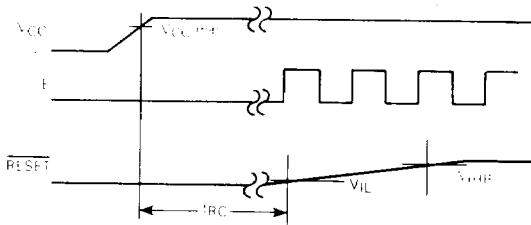
MPU State		MPU State Definition
BA	BS	
0	0	Normal (Running)
0	1	Interrupt or Reset Acknowledge
1	0	Sync Acknowledge
1	1	Halt or Bus Grant Acknowledge

FIGURE 6 — RESET TIMING



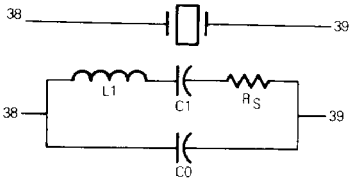
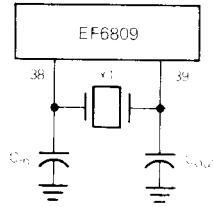
- NOTES: 1. Parts with date codes prefixed by 7F or 8A will come out of RESET one cycle sooner than shown.  
 2. Timing measurements are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts, unless otherwise noted.  
 3. FFFF appears on the bus during RESET low time. Following 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

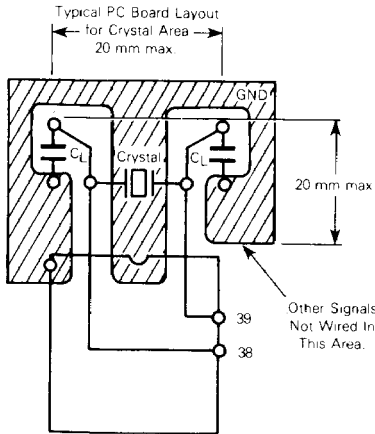
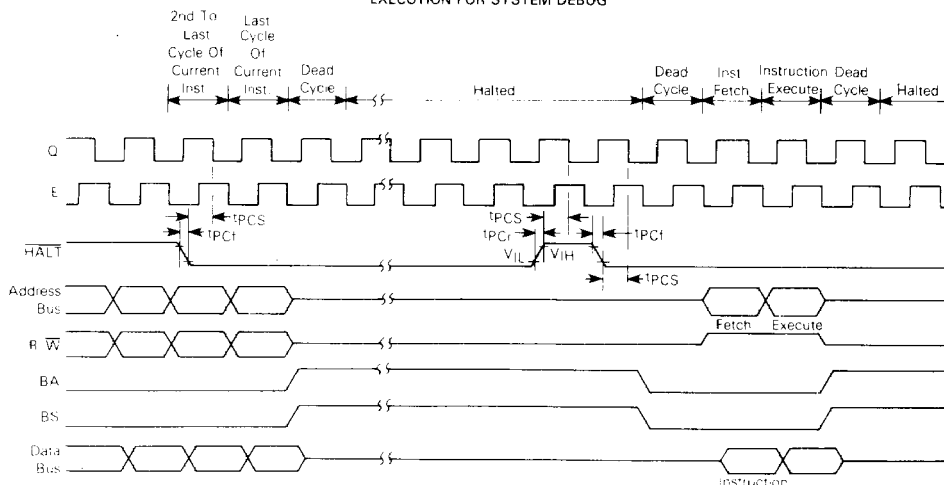




FIGURE 8 – HALT AND SINGLE INSTRUCTION EXECUTION FOR SYSTEM DEBUG



NOTE: Waveform measurements for all inputs and outputs are specified at  $V_{DD} = 5V$  and  $V_{SS} = 0.8V$  unless otherwise specified.

**INTERRUPT ACKNOWLEDGE** is indicated during every cycle of a hardware vector fetch (RESET, NMI,  $\overline{FIRQ}$ ,  $\overline{IRQ}$ , SW1, SW2, SW3). 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 a low vectoring by device. See Table 1.

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

**HALT/BUS GRANT** is true when the MCR809 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	SW1
FFF8	FFF9	$\overline{IRQ}$
FFF6	FFF7	$\overline{FIRQ}$
FFF4	FFF5	SW2
FFF2	FFF3	SW3
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, and also has a higher priority than  $\overline{FIRQ}$ ,  $\overline{IRQ}$ , 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 (SP). 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 ( $\overline{FIRQ}$ )\*

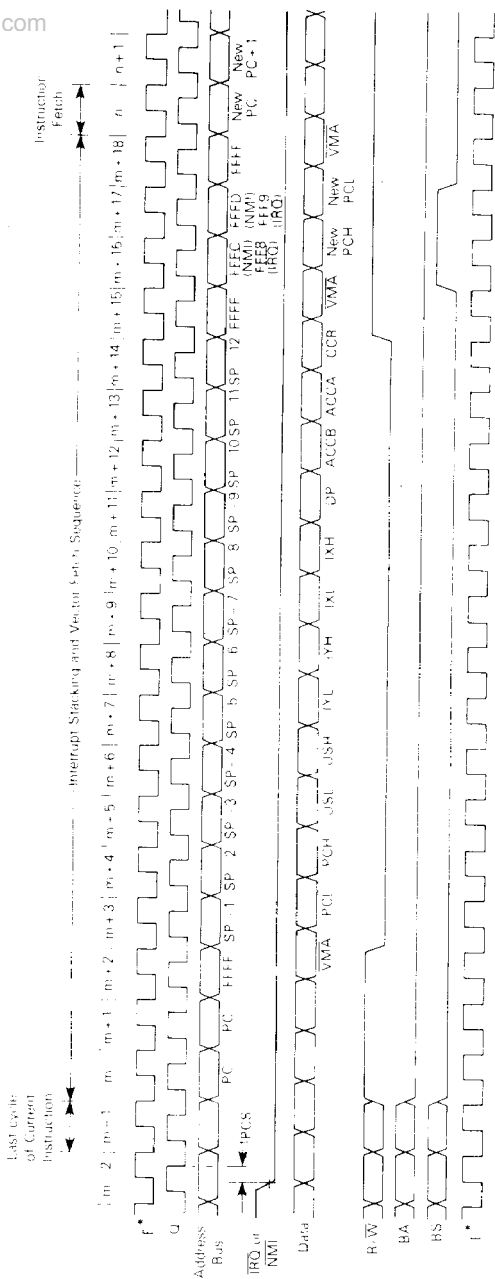
A low level on this input pin will initiate a fast interrupt sequence, provided its mask bit (FI) in the CC is clear. This sequence has priority over the standard interrupt request ( $\overline{IRQ}$ ), and is fast in the sense that it stacks only the contents of the condition code register and the program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

#### INTERRUPT REQUEST ( $\overline{IRQ}$ )\*

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

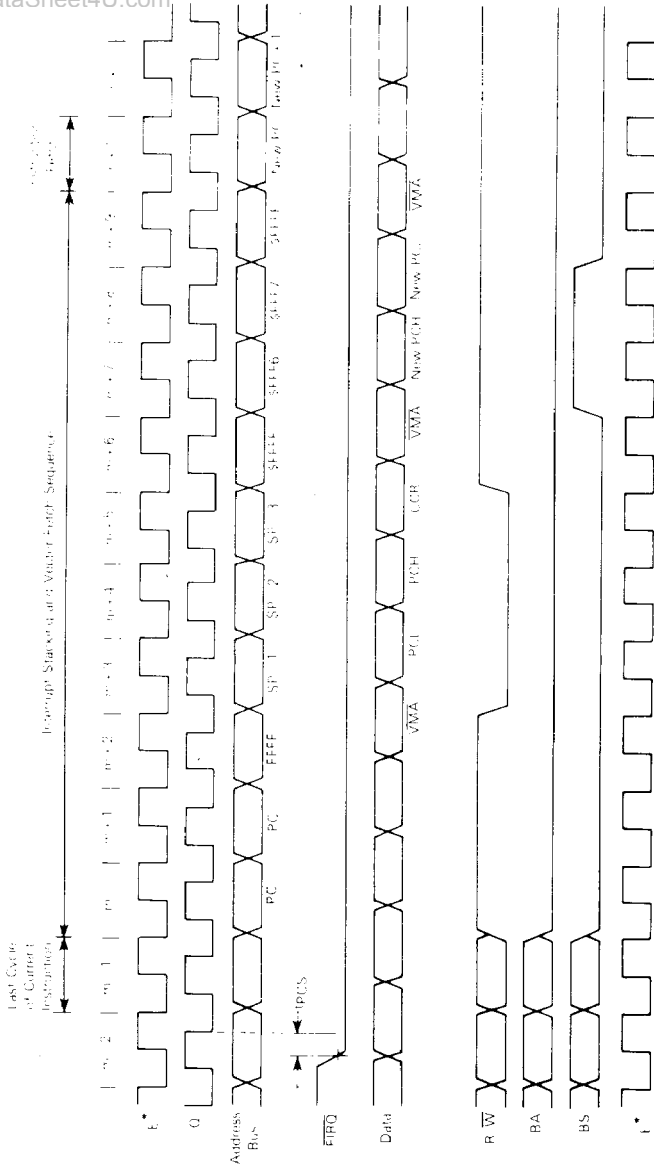
\*NMI,  $\overline{FIRQ}$ , and  $\overline{IRQ}$  requests are sampled on the falling edge of Q. One cycle is required for synchronization before these interrupts are recognized. The pending interrupt will not be serviced until completion of the current instruction, unless a SYNC or CVAL condition is present. If  $\overline{IRQ}$  and  $\overline{FIRQ}$  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 acted between the falling edge of RESET and the rising edge of BS indicating RESET acknowledgement.

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.  
 \* t<sub>clock</sub> shown for reference only.

FIGURE 10 - FIRO INTERRUPT TIMING



NOTE: Waveform measurements for all inputs and outputs are specified at logic 'high' = 2.0 V and logic 'low' = 0.4 V unless otherwise specified.  
 t<sub>c</sub> 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 EF6800 bus timing signal phase 2; Q is a quadrature clock signal which leads E. Q has no parallel on the EF6800. 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, F and Q may be stretched in integral multiples of quarter (1/4) bus cycles, thus allowing interface to slow memories, as shown in Figure 12(a). During non-valid memory access (DMA cycles), MRDY has no effect on stretching F 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\*

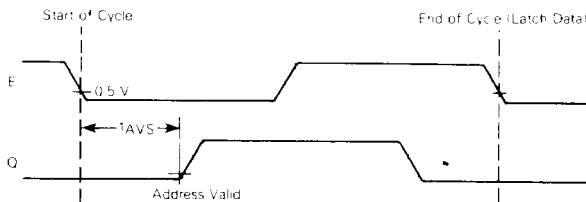
The DMA/BREQ input provides a method of suspending execution and acquiring the MPU bus for another use, as shown in Figure 13. Typical uses include DMA and dynamic memory refresh.

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, the next cycle will be a dead cycle used to transfer bus master control to the DMA controller.

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

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.

- \* The on-board clock generator furnishes E and Q to both the system and the MPU. When MRDY is pulled low, both the system clocks and the internal MPU clocks are stretched. Assertion of DMA/BREQ input stops the internal MPU clocks while allowing the external system clocks to RUN (i.e., release the bus to a DMA controller). The internal MPU clocks resume operation after DMA/BREQ is released or after 16 bus cycles (14 DMA, two dead), whichever occurs first. While DMA/BREQ is asserted it is sometimes necessary to pull MRDY low to allow DMA to/from slow memory peripherals. As both MRDY and DMA/BREQ control the internal MPU clocks, care must be exercised not to violate the maximum  $t_{CYC}$  specification for MRDY or DMA/BREQ. (Maximum  $t_{CYC}$  during MRDY or DMA/BREQ is 16  $\mu\text{s}$ .)

www.DataSheet4U.com  
 If  $\overline{\text{DMA BREQ}} = \text{HIGH}$  or MPU will refresh, the DMA device should be taken off the bus. Another read cycle will happen 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  $\overline{\text{RESET}}$  and is repeated indefinitely unless altered by a special instruction or hardware occurrence. Software instructions that alter normal MPU operation are SW1, SW2, SW3, CWAI, RTI, and SYNC. An interrupt, HALT, or  $\overline{\text{DMA BREQ}}$  can also alter the normal execution of instructions. Figure 15 illustrates the flowchart for the EF6809.

FIGURE 12 — MRDY TIMING AND SYNCHRONIZATION

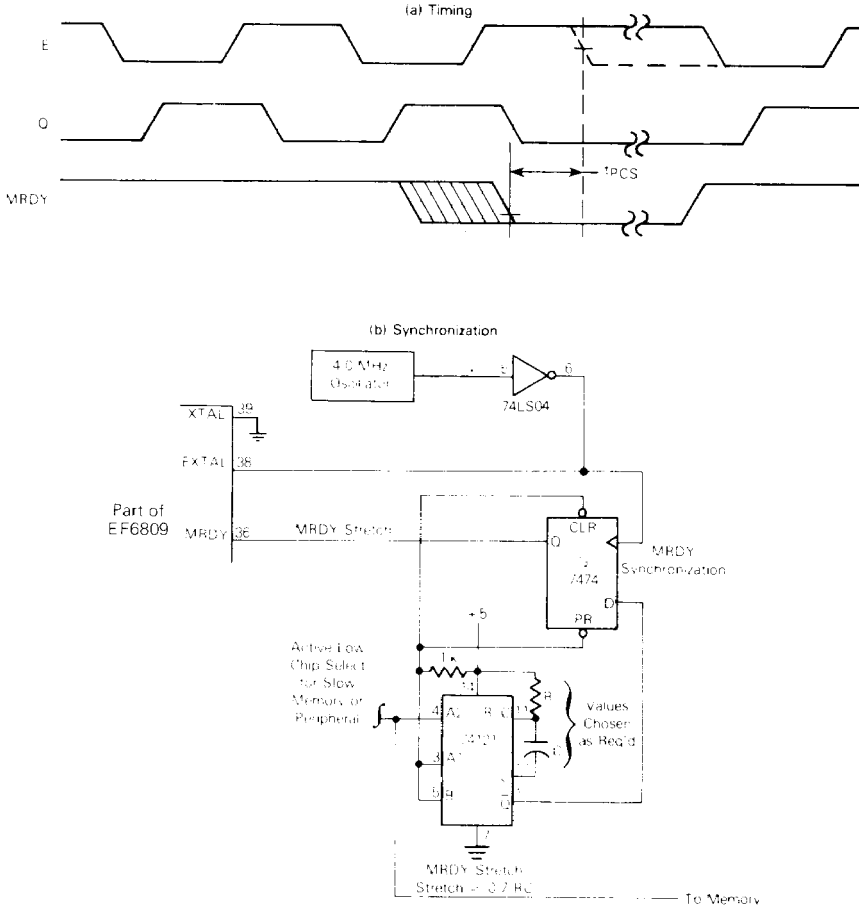


FIGURE 13 — TYPICAL DMA TIMING (< 14 CYCLES)

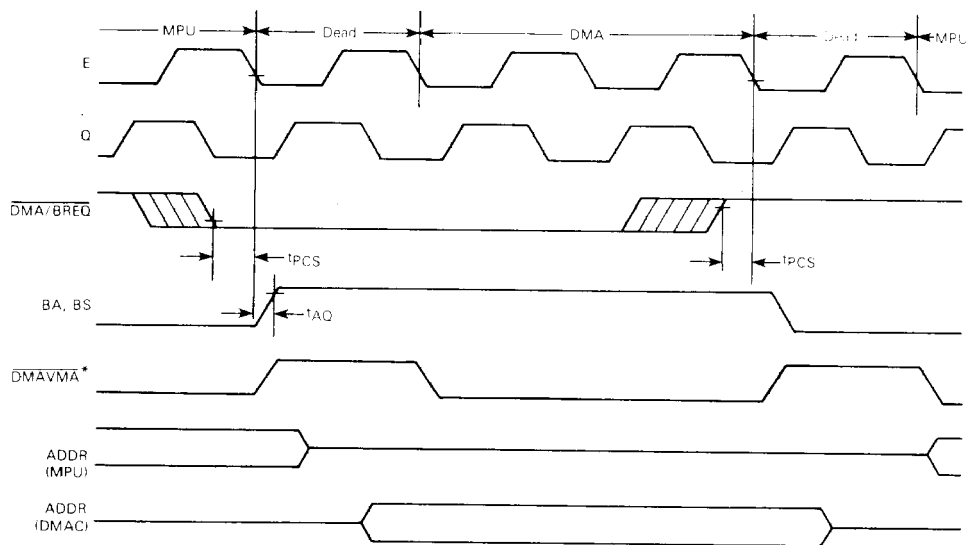
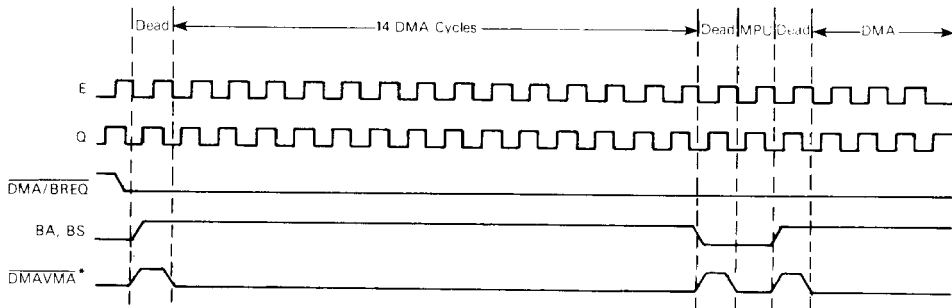


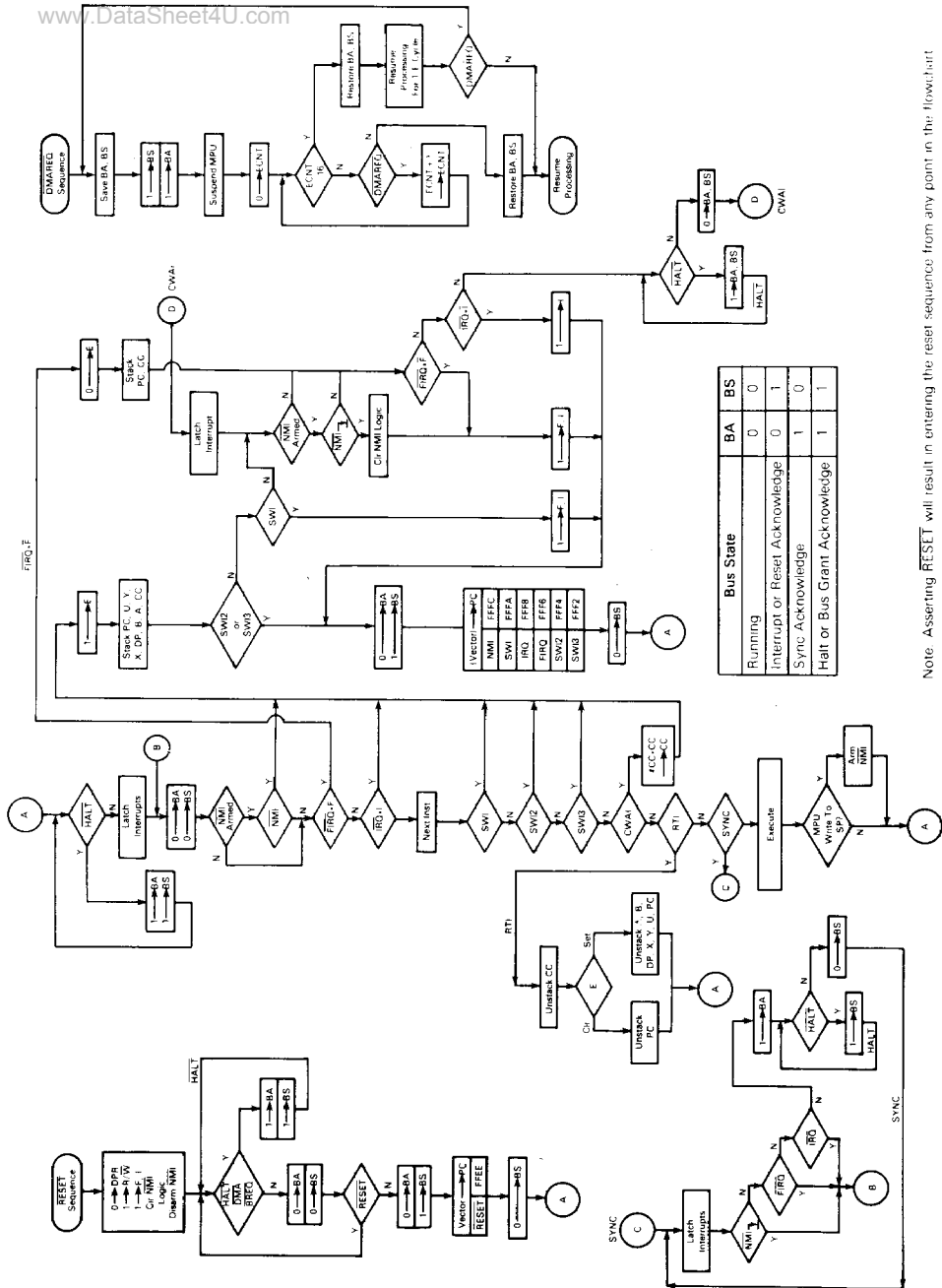
FIGURE 14 — AUTO-REFRESH DMA TIMING (> 14 CYCLES)  
(REVERSE CYCLE STEALING)



\* DMAVMA is a signal which is developed externally, but is a system requirement for DMA

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 15 — FLOWCHART FOR EF6809 INSTRUCTIONS



Note: Asserting RESET will result in entering the reset sequence from any point in the flowchart.

The basic instructions of this computer are greatly enhanced by the presence of 16 different addressing modes. The EF6809 has the most complete set of addressing modes available on any microcomputer today. For example, the EF6809 has 99 basic instructions, however, it encompasses 1464 different variations of instructions and addressing modes. The addressing modes support modern programming techniques. The following addressing modes are available on the EF6809:

- Inherent (includes accumulator)
- Immediate
- Extended Indirect
  - 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 CLR B.

#### IMMEDIATE ADDRESSING

In immediate addressing, the effective address of the instruction is the location immediately following the opcode. Thus, the data to be used in the instruction is immediately following the opcode of the instruction. The EF6809 uses 8-bit and 16-bit immediate values depending on the word of operand specified by the opcode. Examples of instructions with immediate addressing are:

```
LDA #26
LDX #5000
LDY #CAT
```

#### NOTE

# signifies immediate addressing, \$ signifies hex or decimal 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 [SFFFH]
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 EF6809 is compatible with direct addressing on the 6800. Indirect is not allowed in direct addressing. Some examples of direct addressing are:

```
LDA $90
SETDP $10 (assembler directive)
LDB $100
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
PULL X, Y, D Pull D, X, and Y from S
```

#### 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

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

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

**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 0,X
LDA 0,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 offsets are available
- 5 bit (- 16 to + 15)
- 8 bit (- 128 to + 127)
- 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
```

TABLE 2 — INDEXED ADDRESSING MODE

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

+ 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 positive 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  (X),X    EA is now $F100
$F010 $F1
$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  (X),S
LDA  (B),Y
LDD  (X),--
```

## RELATIVE ADDRESSING

The bytes following the branch opcode 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-form 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:

	BEO	CAT	(short)
	BGT	DOG	(short)
CAT	LBEO	RAT	(long)
DOG	LBGT	RABBIT	(long)
	•		
	•		
	•		
RAT	NOF		
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 or referenced relative to the program counter. Examples are:

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

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

```
LDA  (CAT,PCR);
LDU  (DOG,PCR);
```

The instruction set of the EF6809E is similar to that of the 6800 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.

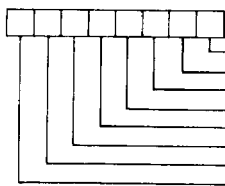
### PSHU/PSHS

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

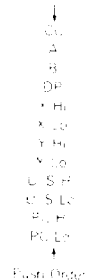
The pull instructions have the same capability of the push instruction, in reverse order. The byte immediately following the push or pull opcode determines which register or registers are to be pushed or pulled. The actual push/pull sequence is fixed, each bit defines a unique register to push or pull,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

Push/Pull Postbyte



Stacking order

PUL Order



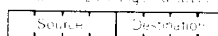
Push Order

Increasing Memory

### TFR/EXG

Within the EF6809E, any register may be transferred to or exchanged with another of like size, i.e., 8 bit to 8 bit or 16 bit to 16 bit. Bits 4-7 of postbyte define the source register, while bits 0-3 represent the destination register. These are denoted as follows:

Transfer/Exchange Postbyte



Register File

0000 = D (A, B)	1000 = A
0001 = X	1001 = B
0010 = Y	1010 = CCR
0011 = U	1011 = DPB
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
LSR    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 ← b + temp (mod 16-bit pointer register)
3. temp ← a (load a)

LEAA (b) ← b

1. b ← temp (calculate EA with preincrement)
2. b ← 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 Clear Up Stack
LEAX 5, S	S ← 5 → X	Transfers As Well As Adds

Autoincrement by two and auto-decrement by two instructions work similarly. Note that LEAX, X+ does not change X, however, LEAX, X- does decrement. LEAX, 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 EF6809 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 (NM $\bar{I}$ ) or maskable (FIRQ, IRQ) with its mask bit (IF or II) clear, the processor will clear the sync state and perform the normal interrupt stacking and service routine. Since  $\overline{FIRQ}$  and  $\overline{IRQ}$  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 (F $\overline{IRQ}$ ,  $\overline{IRQ}$ ) with its mask bit IF or II set, the processor will clear the sync state and continue processing by executing the next in-line instruction. Figure 17 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 EF6809, and are prioritized in the following order: SWI, SWI2, SWI3.

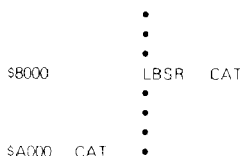
**16-BIT OPERATION**

The EF6809 has the capability of processing 16-bit data. These instructions include loads, stores, compares, adds, subtracts, transfers, exchanges, pushes, and pops.

**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 EF6809. 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 operand of each opcode will follow the flowchart. VMA is an indicator of FFFF $\bar{16}$  on the address bus. R/W = 1 and BS = 1. The following examples illustrate the use of the chart.

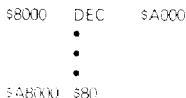
**Example 1: LBSR (Branch Taken)**  
Before Execution: SP = F000



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	06	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	FFFF	80	0	Stack High Order Byte of Return Address
9	FFFF	03	0	Stack Low Order Byte of Return Address

**Example 2: DEC (Extended)**



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	FFF	*	1	VMA Cycle
5	A000	8C	1	Read the Data
6	FFF	*	1	VMA Cycle
7	A000	7F	0	Store the Decrementing Data

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

**INSTRUCTION SET TABLES**

The instructions of the EF6809 have been broken down into five different categories. They are as follows:

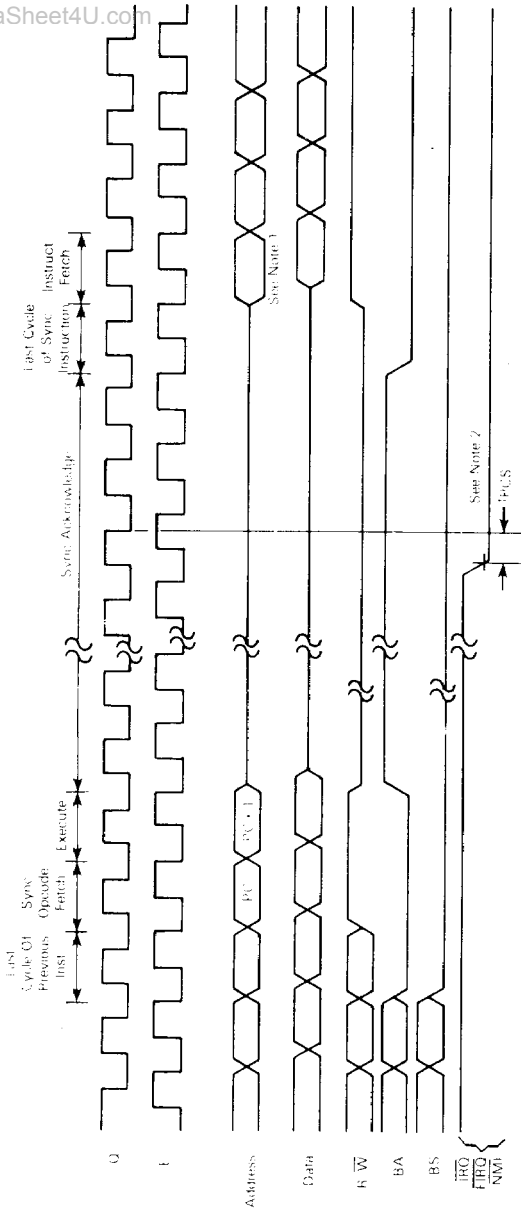
- 8-bit operation (Table 4)
- 16-bit operation (Table 5)
- index register/stack pointer instructions (Table 6)
- Relative branches (long or short) (Table 7)
- Miscellaneous instructions (Table 8)

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 EF6809.

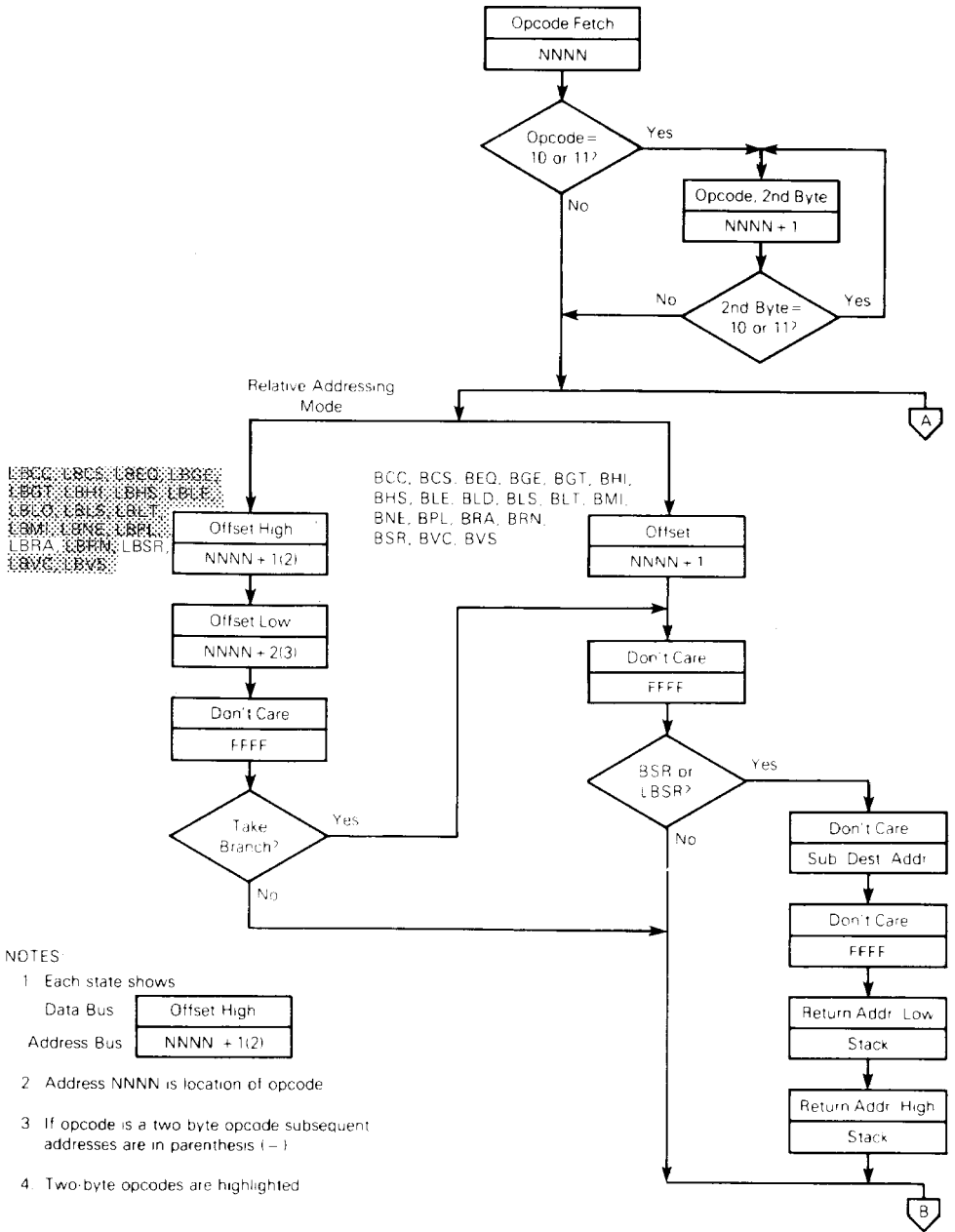
FIGURE 17 - SYNC TIMING



NOTES:

1. If the unacknowledged backbit is not what the peripheral is expecting, it is possible that the address has changed from address 1 to address 1. However, if the next backbit is accepted, NMI or an unacknowledged FREQ or IRQ, although processing continues with this cycle, as per Figures 3 and 10 (inherent timing) the processor will not finish the processing of this cycle.
2. If back bits are clear, IRQ and FREQ 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 an input or output are specified at time high, t<sub>1</sub>, and t<sub>2</sub>, or t<sub>1</sub>, t<sub>2</sub>, t<sub>3</sub>, t<sub>4</sub>, address, otherwise specified.

FIGURE 18 – CYCLE-BY-CYCLE PERFORMANCE (Sheet 1 of 9)



www.DataSheet4U.com **FIGURE 18 – CYCLE-BY-CYCLE PERFORMANCE** (Sheet 2 of 9)

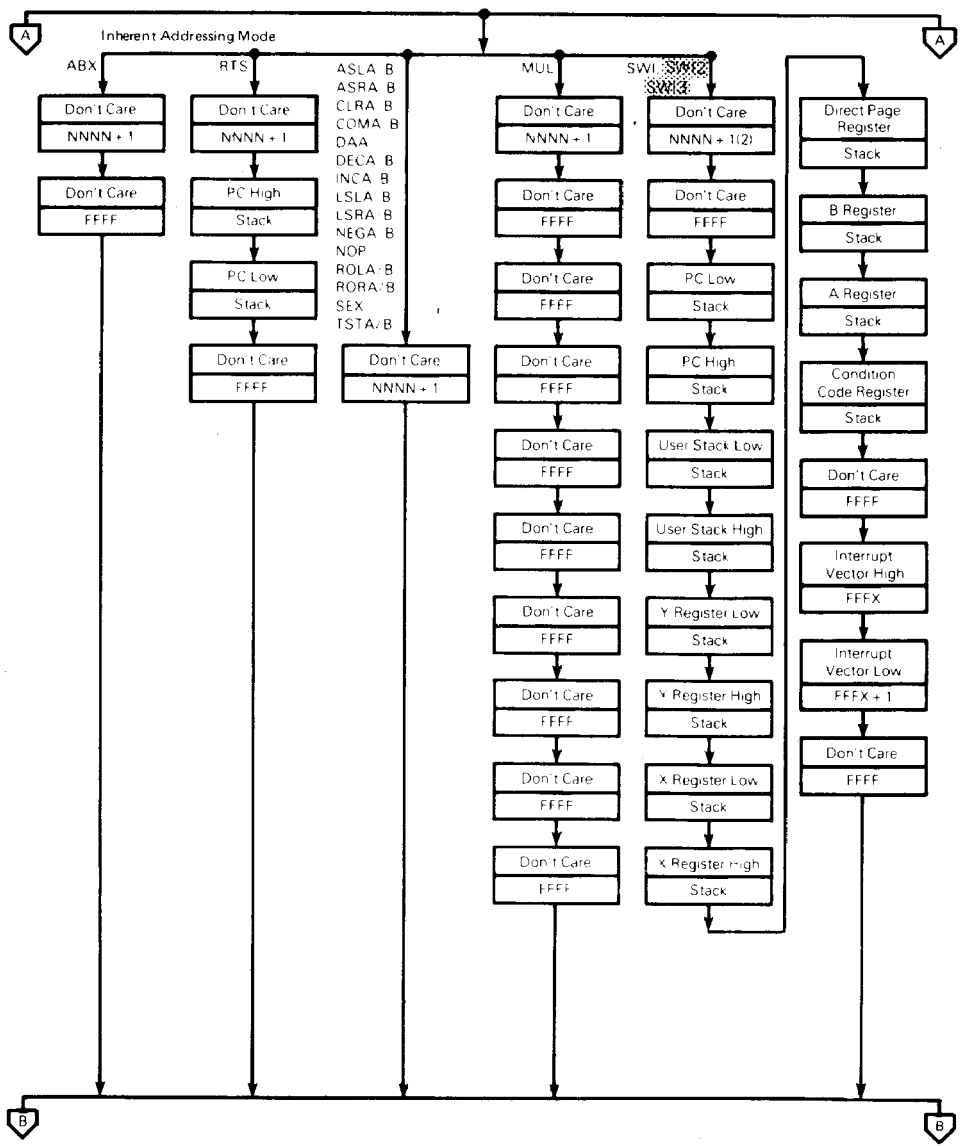
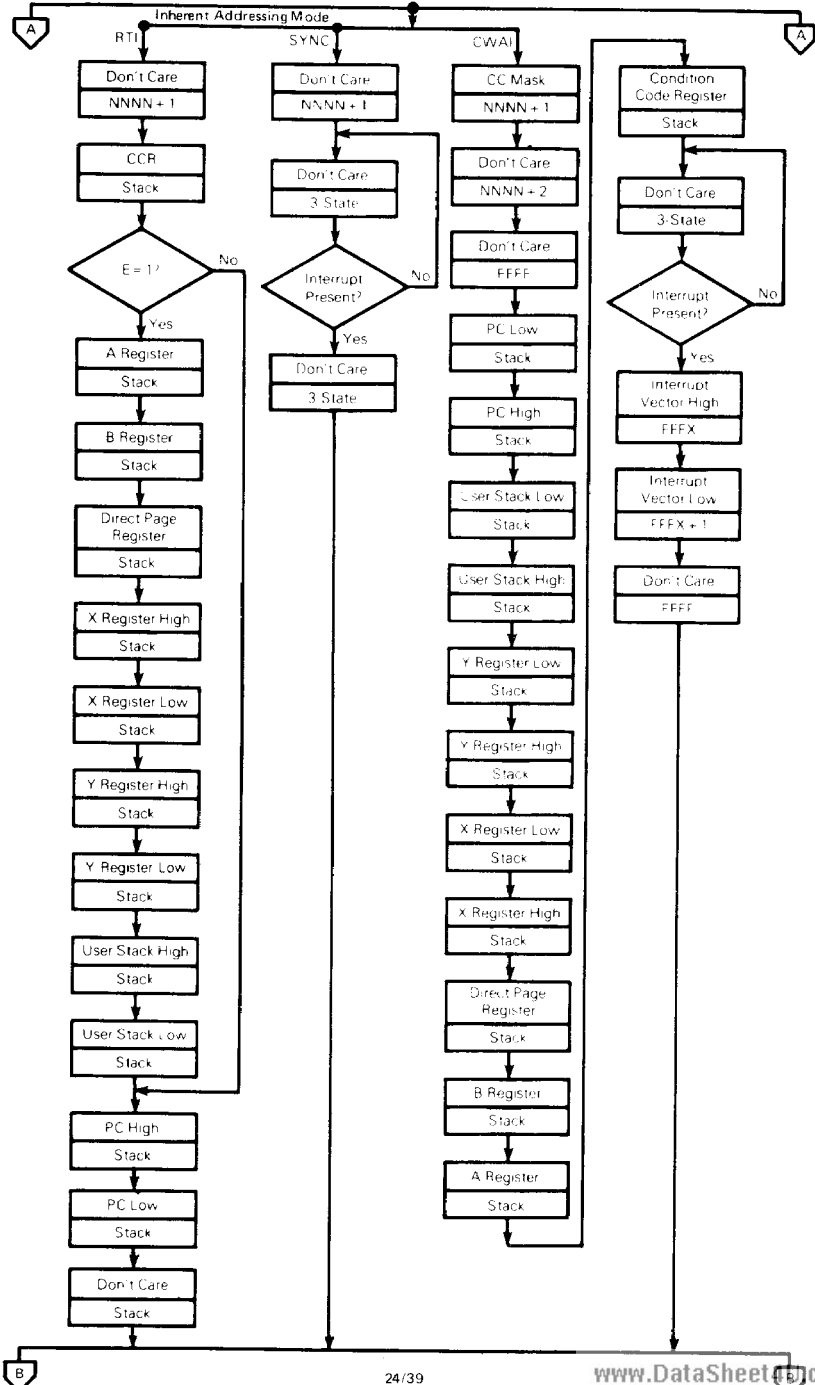


FIGURE 18 – CYCLE-BY-CYCLE PERFORMANCE (Sheet 3 of 9)





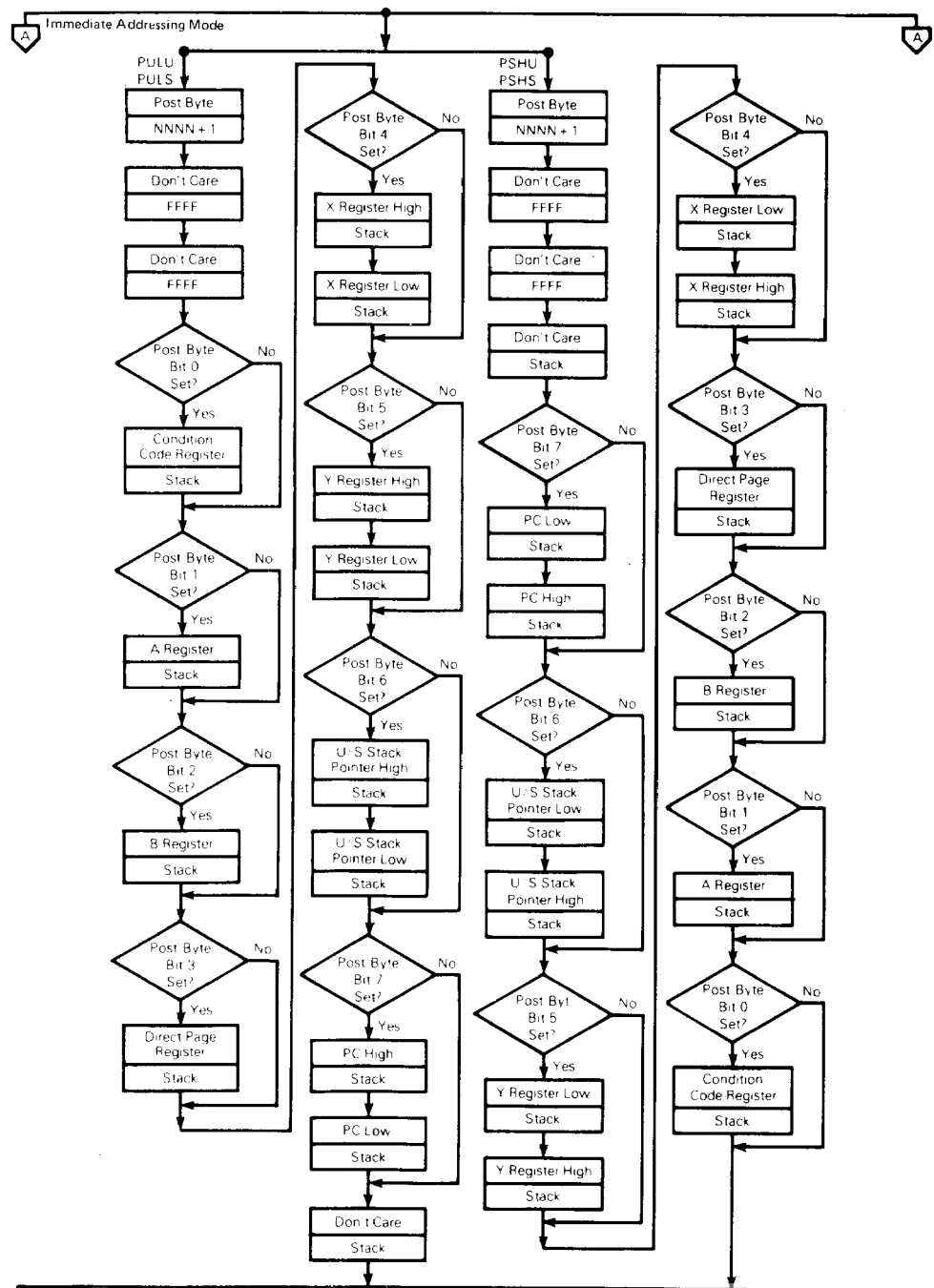
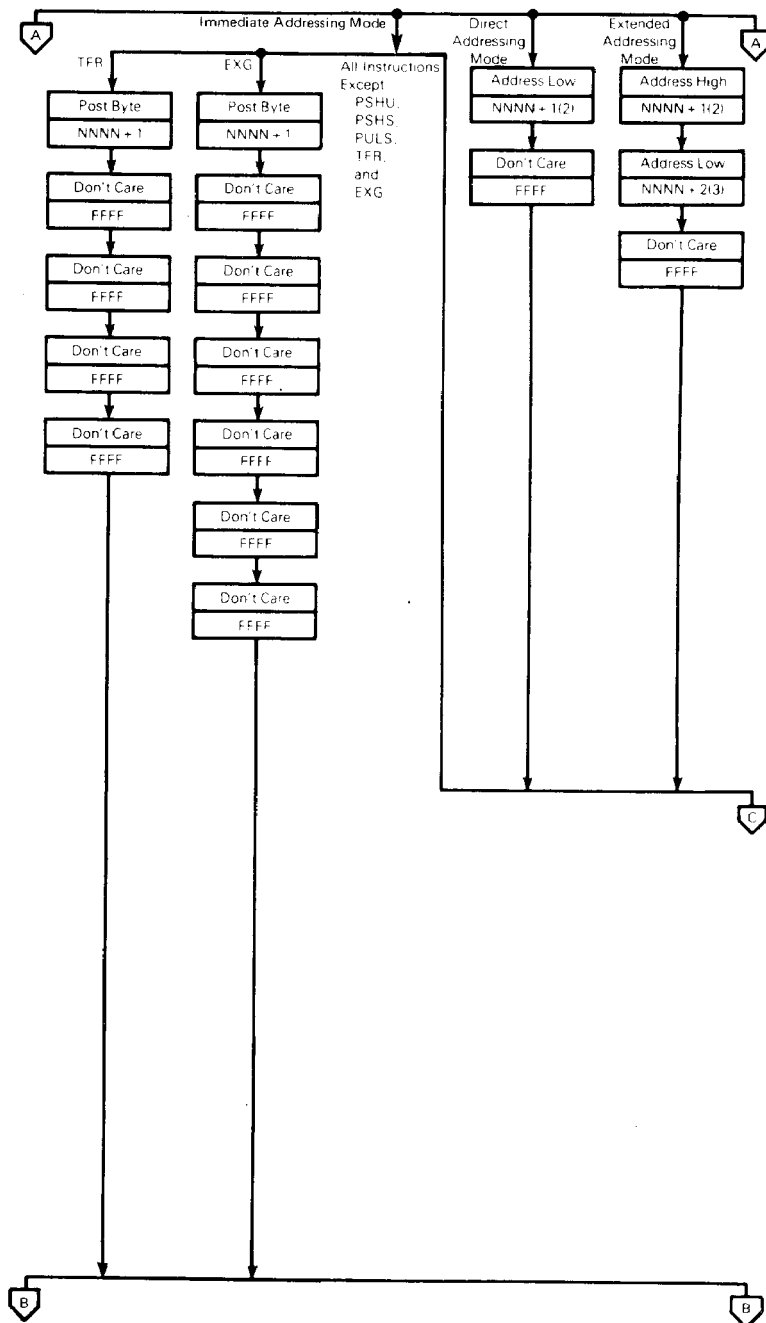
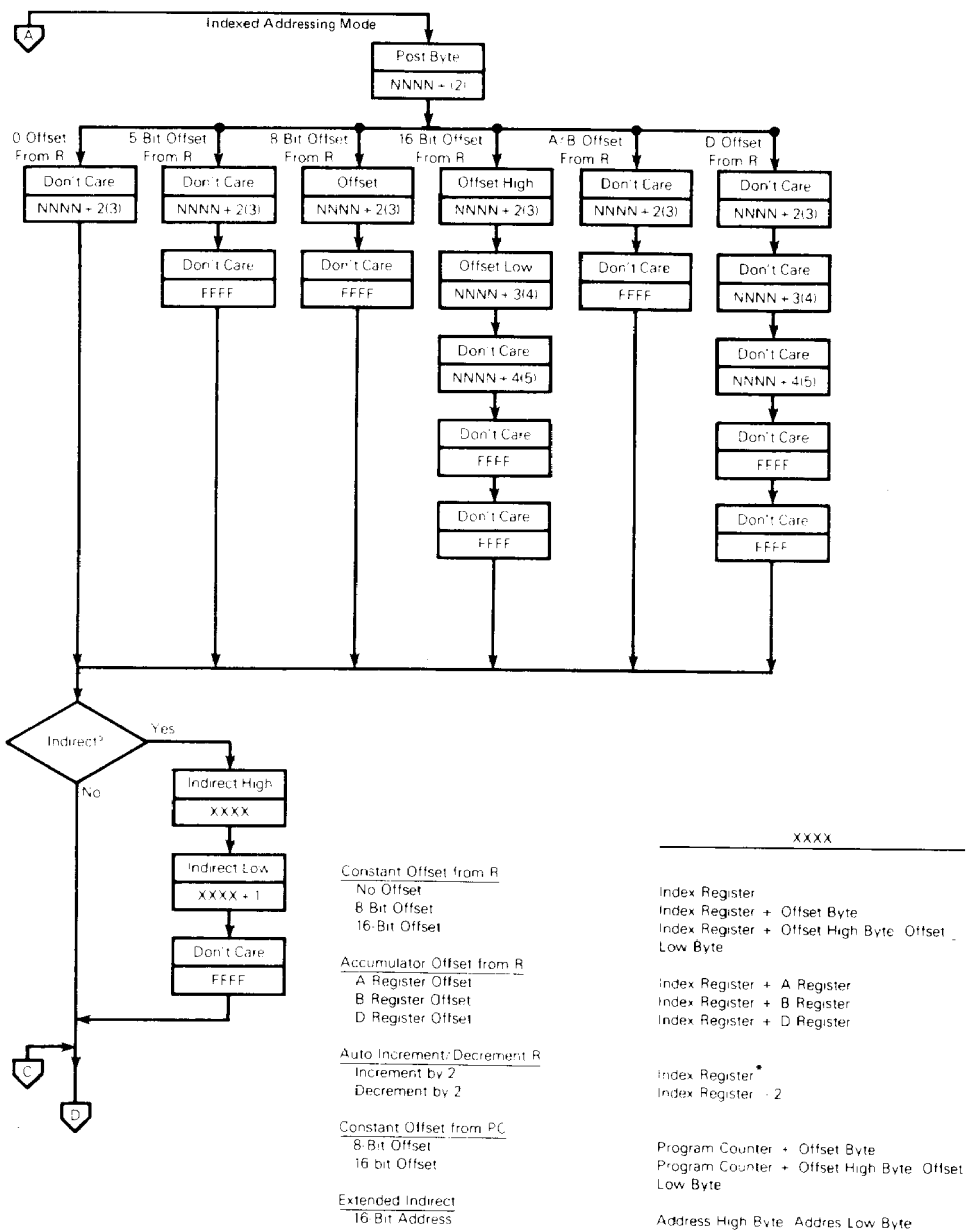


FIGURE 18 – CYCLE-BY-CYCLE PERFORMANCE (Sheet 5 of 9)

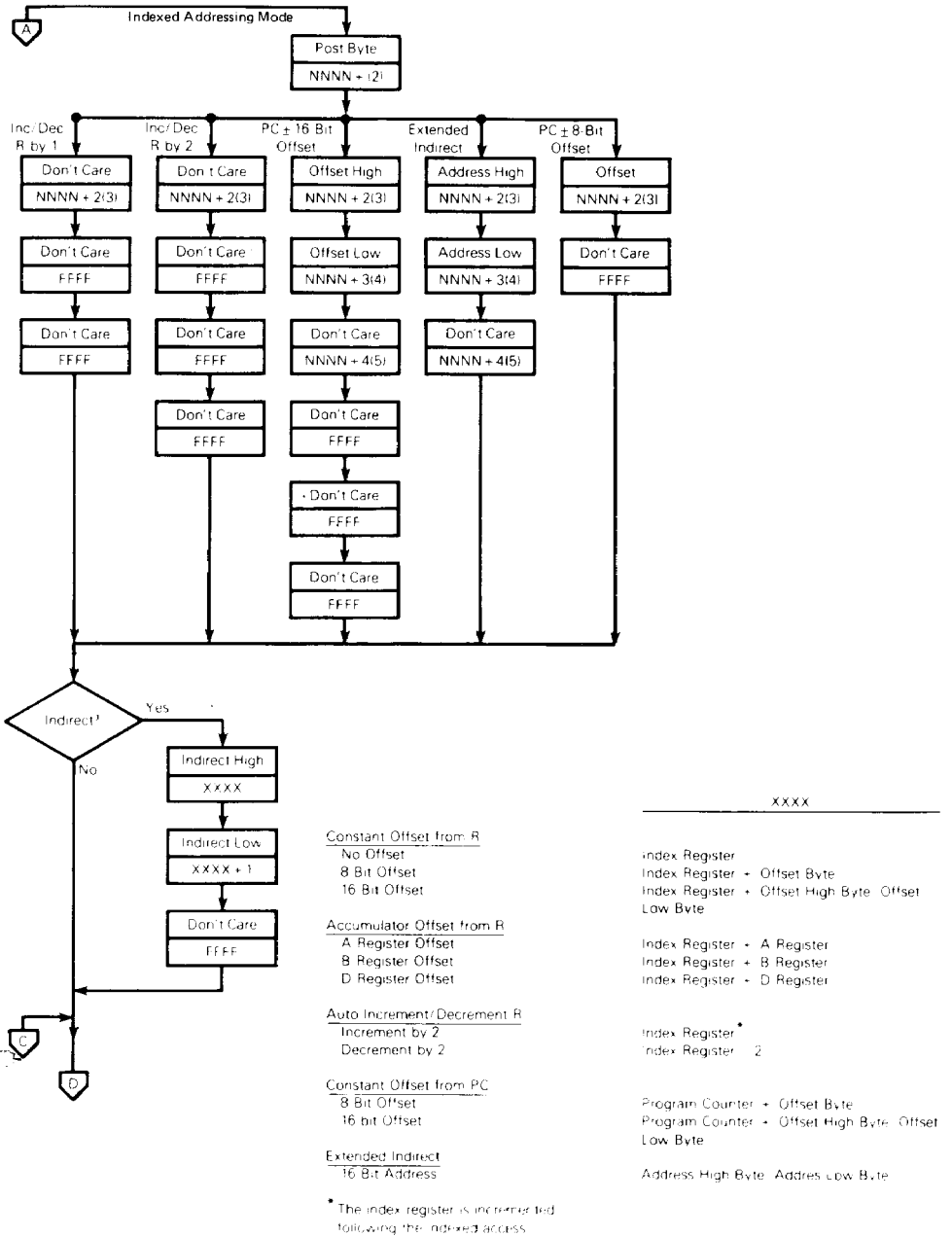


www.DataSheet4U.com **FIGURE 18 – CYCLE-BY-CYCLE PERFORMANCE (Sheet 6 of 9)**

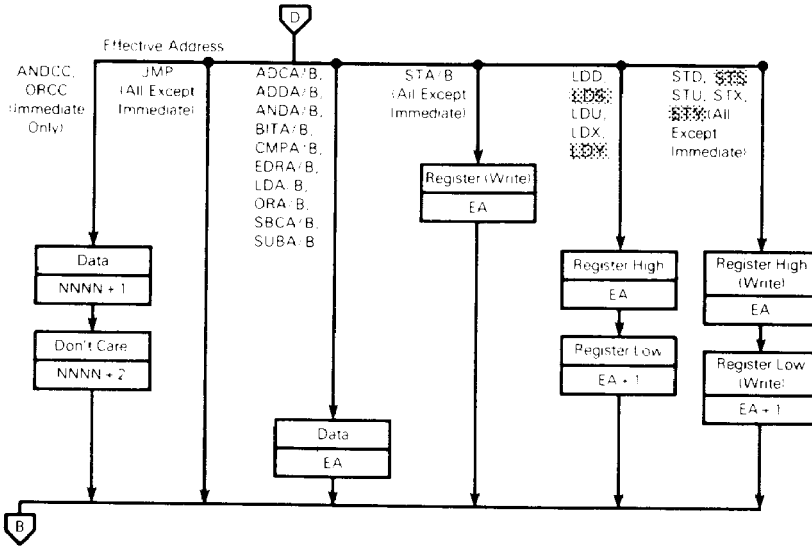


\* The index register is incremented following the indexed access.

FIGURE 18 – CYCLE-BY-CYCLE PERFORMANCE (Sheet 7 of 9)



www.DataSheet4U.com **FIGURE 18 – CYCLE-BY-CYCLE PERFORMANCE** (Sheet 8 of 9)



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

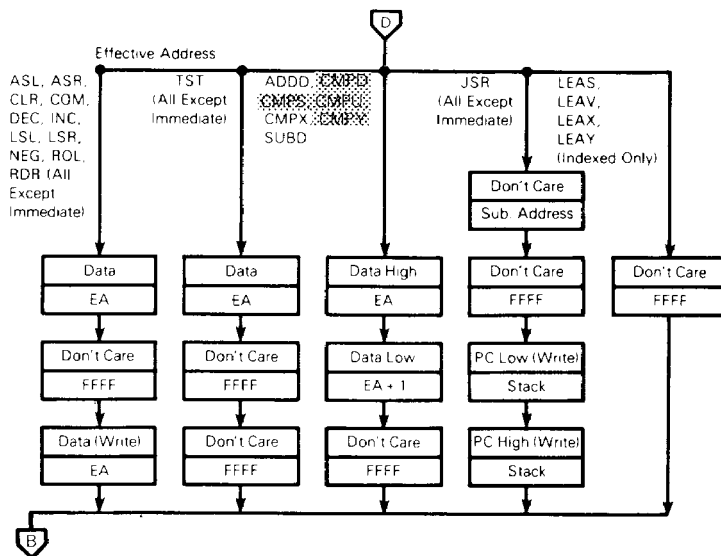
- Program Counter + Offset Byte
- Program Counter + Offset High Byte / Offset Low Byte

Direct Page Register Address Low

Address High / Address Low

NNNN + 1

\* The index register is incremented following the indexed access.



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

www.DataSheet4U.com 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, S, or PC with D, X, Y, S, 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	Pop A, B, CC, DP, D, X, Y, U, or PC from hardware stack
PULL	Pop 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
TFP R1, R2	Transfer D, X, Y, S, 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, LBEC	Branch if equal
BNE, LBNE	Branch if not equal
BMI, LBM	Branch if minus
BPI, LBPL	Branch if plus
BCC, LBCC	Branch if carry clear
BVC, LBVC	Branch if overflow clear
<b>SIGNED BRANCHES</b>	
BGT, LBGT	Branch if greater (signed)
BVS, LBVS	Branch if valid 2's complement result
BGE, LBGE	Branch if greater than or equal (signed)
BEQ, LBEC	Branch if equal
BNE, LBNE	Branch if not equal
BLE, LBLE	Branch if less than or equal (signed)
BVC, LBVC	Branch if valid 2's 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, LBEC	Branch if equal
SNE, LBNE	Branch if not equal
BLS, LBLs	Branch if lower or same (unsigned)
BCL, LBCL	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 – HEXADEXIMAL 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	*				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		Immed	5+	2	65		*			
06	ROR		6	2	36	PSHU		Immed	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	*					68		ASL, LSL	6+	2+	
09	ROL		6	2	39	RTS		Inherent ↑	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	CWAt	≥ 20	2		6C	INC	6+	2-				
0D	TST	6	2	3D	MUL	Inherent ↓	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	LSP	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	DFC	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	4	3					
2D	BLT	3	2	5D	ISTB	2	1	8D	BSR	7	2					
2E	BGT	3	2	5E	*			8E	LDX	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 9 - HEXADEXIMAL VALUES OF MACHINE CODES (CONTINUED)

OP	Mnem	Mode	-	#	OP	Mnem	Mode	-	#	OP	Mnem	Mode	-	#						
80	SCBA	Direct	4	2	80	SURB	Direct	4	2	Page 2 and 3 Machine Codes										
81	CMPA	↑	4	2	81	CMPB	↑	4	2											
82	SBCA	↑	4	2	82	SBCB	↑	4	2											
83	SEBD	↑	4	2	83	ADDB	↑	4	2											
84	ANDA	↑	4	2	84	ANDB	↑	4	2											
85	BITA	↑	4	2	85	BTB	↑	4	2											
86	LDA	↑	4	2	86	LDB	↑	4	2											
87	STA	↑	4	2	87	STB	↑	4	2											
88	EQRA	↑	4	2	88	EQRB	↑	4	2											
89	ADCA	↑	4	2	89	ADCB	↑	4	2											
8A	ORA	↑	4	2	8A	ORB	↑	4	2											
8B	ADDA	↑	4	2	8B	ADDB	↑	4	2											
8C	CMPX	↑	4	2	8C	LDB	↑	4	2											
8D	LSR	↑	4	2	8D	STB	↑	4	2											
8E	LDX	↑	4	2	8E	STB	↑	4	2											
8F	STX	↑	4	2	8F	STB	↑	4	2											
90	SCBA	Extended	5	3	90	SURB	Direct	4	2							1021	LBH7	Relative	5	4
91	CMPA	↑	4	2	91	CMPB	↑	4	2							1022	LBH	↑	5	4
92	SBCA	↑	4	2	92	SBCB	↑	4	2							1023	LBLS	↑	5	4
93	SEBD	↑	4	2	93	ADDB	↑	4	2							1024	LBHS	↑	5	4
94	ANDA	↑	4	2	94	ANDB	↑	4	2	1025	LBUS	↑	5	4						
95	BITA	↑	4	2	95	BTB	↑	4	2	1026	LBUS	↑	5	4						
96	LDA	↑	4	2	96	LDB	↑	4	2	1027	LBUS	↑	5	4						
97	STA	↑	4	2	97	STB	↑	4	2	1028	LBUS	↑	5	4						
98	EQRA	↑	4	2	98	EQRB	↑	4	2	1029	LBUS	↑	5	4						
99	ADCA	↑	4	2	99	ADCB	↑	4	2	1030	LBUS	↑	5	4						
9A	ORA	↑	4	2	9A	ORB	↑	4	2	1031	LBUS	↑	5	4						
9B	ADDA	↑	4	2	9B	ADDB	↑	4	2	1032	LBUS	↑	5	4						
9C	CMPX	↑	4	2	9C	LDB	↑	4	2	1033	LBUS	↑	5	4						
9D	LSR	↑	4	2	9D	STB	↑	4	2	1034	LBUS	↑	5	4						
9E	LDX	↑	4	2	9E	STB	↑	4	2	1035	LBUS	↑	5	4						
9F	STX	↑	4	2	9F	STB	↑	4	2	1036	LBUS	↑	5	4						
80	SCBA	Extended	5	3	80	SURB	Direct	4	2	1037	LBUS	↑	5	4						
81	CMPA	↑	4	2	81	CMPB	↑	4	2	1038	LBUS	↑	5	4						
82	SBCA	↑	4	2	82	SBCB	↑	4	2	1039	LBUS	↑	5	4						
83	SEBD	↑	4	2	83	ADDB	↑	4	2	1040	LBUS	↑	5	4						
84	ANDA	↑	4	2	84	ANDB	↑	4	2	1041	LBUS	↑	5	4						
85	BITA	↑	4	2	85	BTB	↑	4	2	1042	LBUS	↑	5	4						
86	LDA	↑	4	2	86	LDB	↑	4	2	1043	LBUS	↑	5	4						
87	STA	↑	4	2	87	STB	↑	4	2	1044	LBUS	↑	5	4						
88	EQRA	↑	4	2	88	EQRB	↑	4	2	1045	LBUS	↑	5	4						
89	ADCA	↑	4	2	89	ADCB	↑	4	2	1046	LBUS	↑	5	4						
8A	ORA	↑	4	2	8A	ORB	↑	4	2	1047	LBUS	↑	5	4						
8B	ADDA	↑	4	2	8B	ADDB	↑	4	2	1048	LBUS	↑	5	4						
8C	CMPX	↑	4	2	8C	LDB	↑	4	2	1049	LBUS	↑	5	4						
8D	LSR	↑	4	2	8D	STB	↑	4	2	1050	LBUS	↑	5	4						
8E	LDX	↑	4	2	8E	STB	↑	4	2	1051	LBUS	↑	5	4						
8F	STX	↑	4	2	8F	STB	↑	4	2	1052	LBUS	↑	5	4						
NOTE: All unused opcodes are both unimplemented and illegal.					80	SURB	Extended	5	3	1053	LBUS	↑	5	4						
					81	CMPB	↑	4	2	1054	LBUS	↑	5	4						
					82	SBCB	↑	4	2	1055	LBUS	↑	5	4						
					83	ADDB	↑	4	2	1056	LBUS	↑	5	4						
					84	ANDB	↑	4	2	1057	LBUS	↑	5	4						
					85	BTB	↑	4	2	1058	LBUS	↑	5	4						
					86	LDB	↑	4	2	1059	LBUS	↑	5	4						
					87	STB	↑	4	2	1060	LBUS	↑	5	4						
					88	EQRB	↑	4	2	1061	LBUS	↑	5	4						
					89	ADCB	↑	4	2	1062	LBUS	↑	5	4						
					8A	ORB	↑	4	2	1063	LBUS	↑	5	4						
					8B	ADDB	↑	4	2	1064	LBUS	↑	5	4						
8C	LDB	↑	4	2	1065	LBUS	↑	5	4											
8D	STB	↑	4	2	1066	LBUS	↑	5	4											
8E	LDB	↑	4	2	1067	LBUS	↑	5	4											
8F	STB	↑	4	2	1068	LBUS	↑	5	4											

Instruction	Forms	Addressing Modes												Description												
		Immediate			Direct			Indexed			Extended			Inherent			5	3	2	1	0					
		Op	-	#	Op	-	#	Op	-	#	Op	-	#	Op	-	#	3A	3	1	H	N	Z	V	C		
ABX																										
ADC	ADCA ADCB	89 C9	2 2	2 2	99 D9	4 4	2 2	A9 E9	4 4	2 2	B9 F9	5 5	3 3													
ADD	ADDA ADDB ADDD	88 CB C3	2 2 4	2 2 3	98 DB D3	4 4 6	2 2 2	A8 E8 E3	4 4 6	2 2 2	B8 F8 F3	5 5 7	3 3 3													
AND	ANDA ANDB ANDCC	94 C4 C0	2 2 3	2 2 2	94 D4 D0	4 4 6	2 2 2	A4 E4 E0	4 4 6	2 2 2	B4 F4 F0	5 5 7	3 3 3													
ASL	ASLA ASLB ASL													98 58	2 2	1 1										
ASR	ASRA ASRB ASR													47 57	2 2	1 1										
BIT	BITA BITB	B5 C5	2 2	2 2	B5 C5	4 4	2 2	A5 D5	4 4	2 2	B5 F5	5 5	3 3													
CLR	CLRA CLRB CLR													4F 5F	2 2	1 1										
CMF	CMFA CMFB CMFD CMFS CMF <sub>x</sub> CMF <sub>y</sub>	81 C1 C3 83 85 86 87	2 2 3 2 2 2	2 2 4 2 2 2	91 D1 D3 93 95 96 97	4 4 6 4 4 4	2 2 2 2 2 2	A1 E1 E3 A3 A5 A6 A7	4 4 6 4 4 4 4	2 2 2 2 2 2	B1 F1 F3 B3 B5 B6 B7	5 5 7 5 5 5 5	3 3 3 3 3 3 3													
COM	COMA COMB COM													47 57	2 2	1 1										
CWAIT		9C	≥20	2																						
DAA														79	2	1										
DEC	DECA DECB DEC													3A 5A	2 2	1 1										
FOR	FORA FORB	88 CB	2 2	2 2	98 DB	4 4	2 2	A8 E8	4 4	2 2	B8 F8	5 5	3 3													
EXG	RT <sub>1</sub> ↔ <sub>2</sub>	1E	3	2																						
INC	INCA INCB INL													1C 5C	2 2	1 1										
IMP																										
JSR																										
LD	LDA LDB LDD LDS LDL LDX LDY	86 C6 C0 10 CE CE 10	2 2 3 4 3 3 4	2 2 3 4 3 3 4	96 D6 D0 10 DE DE 10	4 4 6 6 6 6 6	2 2 2 2 2 2 2	A6 E6 F6 10 EE EE AE	4 4 6 6 6 6 6	2 2 2 2 2 2 2	B6 F6 10 10 FE FE BE	5 5 6 6 6 6 6	3 3 3 3 3 3 3													
LEA	LEAU LEAX LEAY													32 33 30 31	4 4 4 4	2 2 2 2										

LEGEND

OP Operation Code (Hexadecimal)

- Number of MPU Cycles

# Number of Program Bytes

+ Arithmetic Plus

- Arithmetic Minus

• Multiply

$\bar{M}$  Complement of M

- Transfer Into

H Half-carry (from bit 3)

N Negative (sign bit)

Z Zero result

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

∧ Logical and

⊕ 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	-	#			
LSC	LSTA LSLB LSL				08	6	2	6H	6	2				48	6	2	48	6	2	A ← B B ← A	•	•	•	•	•
LSR	LSPB LSRB LSR				04	6	2	6H	6	2				44	6	2	44	6	2	A ← B B ← A	•	•	•	•	•
MUL														30	6	2				A ← B × D	•	•	•	•	•
NEG	NEGA NEGB NEG				30	6	2	6H	6	2				40	6	2	40	6	2	A ← -A B ← -B	•	•	•	•	•
NOP														4C	6	2				No operation	•	•	•	•	•
OR	ORA ORB ORCC	8A CA 1A	2 2 3	2	9A DA	4 4	2	AA AA	4 4	2	BA BA	6 6	2	4	3				A ← A OR B B ← B OR A C ← C OR 1	•	•	•	•	•	
PSH	PSHS PSHL	34 36	5 6	4 4																Push register(s) onto stack	•	•	•	•	•
PUL	PULLS PULL	35 37	5 6	4 4																Pop register(s) from stack	•	•	•	•	•
ROL	ROLA ROLB ROL							39	6	2										A ← (A << 1) OR (A >> 1) B ← (B << 1) OR (B >> 1)	•	•	•	•	•
ROP	ROPA ROPB ROP							3E	6	2	9E	6	2							A ← (A >> 1) OR (A << 1) B ← (B >> 1) OR (B << 1)	•	•	•	•	•
RTI																				Return from interrupt	•	•	•	•	•
RTS																				Return from subroutine	•	•	•	•	•
SBC	SBCA SBCB	82 C2	2 2	2	92 D2	4 4	2	A A	4 4	2	B B	6 6	2	4	3				A ← A - B B ← B - A	•	•	•	•	•	
SEX																				Set extended sign	•	•	•	•	•
ST	STA STB STD STS STL STX STY				97 D7 D0 7E 8F 9F 7E	4 4 6 6 6 6 6	2	EA EA EA EA EA EA EA	4 4 4 4 4 4 4	2	BA BA BA BA BA BA BA	6 6 6 6 6 6 6	2	4	3				Store register(s) to memory	•	•	•	•	•	
SUB	SUBA SUBB SUBD	8C CC 83	2 2 4	2	9C DC 93	4 4 6	2	EA EA EA	4 4 4	2	BA BA BA	6 6 6	2	4	3				A ← A - B B ← B - A D ← D - B	•	•	•	•	•	
SWI	SWI <sup>6</sup> SWI <sup>2</sup> SWI <sup>4</sup>													4F 4F 4F	19 20 20				Software interrupt	•	•	•	•	•	
SYNC														3F	≥4				Synchronize memory	•	•	•	•	•	
TFR	R1, R2	1F	6	2																R1 ← R2	•	•	•	•	•
TST	TSTA TSTB TST				00	6	2	8D 6+	6+	2+	7D	6	2	3	4D 5D	2				Test A Test B Test M	•	•	•	•	•

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, Table 2.
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

Branch instructions

Instruction	Forms	Addressing Mode				Description	5	3	2	1	0
		Relative		#	#						
		OP	~								
BCC	BCC	24	3	2	Branch <= 0	*	*	*	*	*	
	LBCC	10 24	5/6	4	Long Branch <= 0	*	*	*	*	*	
BCS	BCCS	25	3	2	Branch <= 1	*	*	*	*	*	
	LBCCS	10 25	5/6	4	Long Branch <= 1	*	*	*	*	*	
BEQ	BEQ	27	3	2	Branch = 0	*	*	*	*	*	
	LBEC	10 27	5/6	4	Long Branch = 0	*	*	*	*	*	
BGE	BGE	2C	3	2	Branch >= 1	*	*	*	*	*	
	LBGE	10 2C	5/6	4	Long Branch >= 1	*	*	*	*	*	
BGT	BGT	2E	3	2	Branch >= 2	*	*	*	*	*	
	LBGT	10 2E	5/6	4	Long Branch >= 2	*	*	*	*	*	
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	28	3	2	Branch <= 2	*	*	*	*	*	
	LBLE	10 28	5/6	4	Long Branch <= 2	*	*	*	*	*	
BLO	BLO	2A	3	2	Branch Lower	*	*	*	*	*	
	LBLO	10 2A	5/6	4	Long Branch Lower	*	*	*	*	*	

Instruction	Forms	Addressing Mode				Description	5	3	2	1	0
		Relative		#	#						
		OP	~								
BLS	BLS	23	3	2	Branch Lower or Same	*	*	*	*	*	
	LBLS	10 23	5/6	4	Long Branch Lower or Same	*	*	*	*	*	
BLT	B LT	2D	3	2	Branch < Zero	*	*	*	*	*	
	LB LT	10 2D	5/6	4	Long Branch < Zero	*	*	*	*	*	
BVM	BVM	2B	3	2	Branch Minus	*	*	*	*	*	
	LBVM	10 2B	5/6	4	Long Branch Minus	*	*	*	*	*	
BNE	BNE	26	3	2	Branch Not Equal	*	*	*	*	*	
	LBNE	10 26	5/6	4	Long Branch Not Equal	*	*	*	*	*	
BPL	BPL	2A	3	2	Branch Plus	*	*	*	*	*	
	LBPL	10 2A	5/6	4	Long Branch Plus	*	*	*	*	*	
BRA	BRA	2C	3	2	Branch Always	*	*	*	*	*	
	LBRA	10 2C	5/6	4	Long Branch Always	*	*	*	*	*	
BRN	BRN	21	3	2	Branch Never	*	*	*	*	*	
	LB RN	10 21	5/6	4	Long Branch Never	*	*	*	*	*	
BSR	BSR	2D	3	2	Branch to Subtract	*	*	*	*	*	
	LB SR	10 2D	5/6	4	Long Branch to Subtract	*	*	*	*	*	
BVC	BVC	2E	3	2	Branch <= 2	*	*	*	*	*	
	LBVC	10 2E	5/6	4	Long Branch <= 2	*	*	*	*	*	
BVS	BVS	2A	3	2	Branch <= 1	*	*	*	*	*	
	LBVS	10 2A	5/6	4	Long Branch <= 1	*	*	*	*	*	

SIMPLE BRANCHES

	OP	~	#
BRA	2C	3	2
LBRA	16	5	3
BRN	21	3	2
LB RN	1021	5	4
BSR	8D	7	2
LB SR	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)

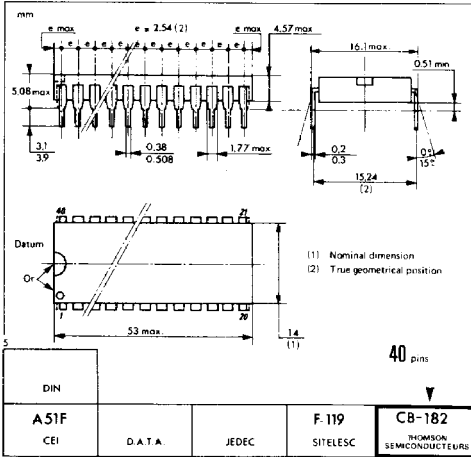
Test	True	OP	False	OP
r > m	BGT	2E	BLE	2F
r ≥ m	BGE	2C	BLT	2D
r = m	BEQ	27	BNE	26
r ≤ m	BLE	2F	BGT	2E
r < m	BLT	2D	BGE	2C

UNSIGNED CONDITIONAL BRANCHES (Notes 1-4)

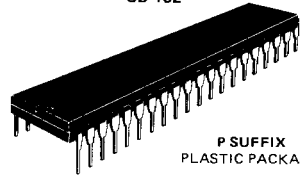
Test	True	OP	False	OP
r > m	BHI	22	BLS	23
r ≥ m	BHS	24	BLO	25
r = m	BEQ	27	BNE	26
r ≤ m	BLS	23	BHI	22
r < m	BLO	25	BHS	24

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.



CB-182



P SUFFIX  
PLASTIC PACKAGE

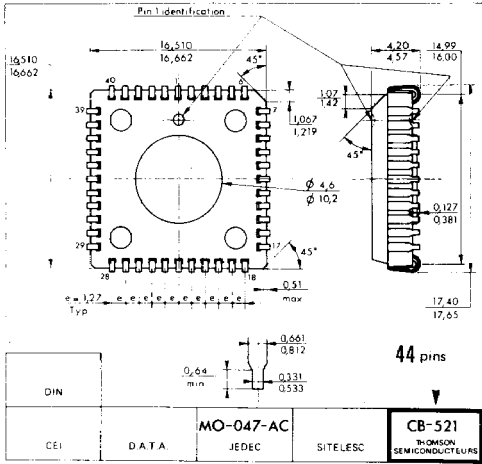
ALSO AVAILABLE

J SUFFIX  
CERDIP PACKAGE

C SUFFIX  
CERAMIC PACKAGE

ORDERING INFORMATION

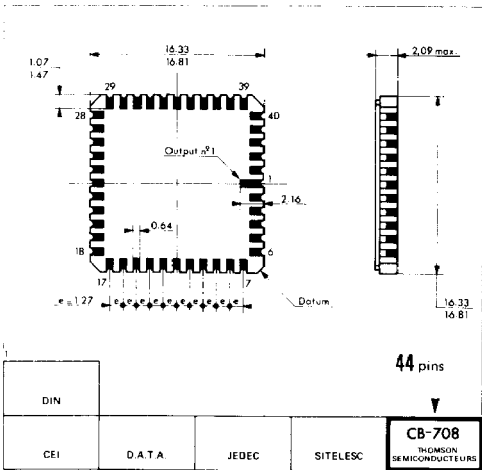
EF68A09		C	M	B/B									
Device		Package			Oper. temp.				Screening level				
The table below horizontally shows all available suffix combinations for package, operating temperature and screening level. Other possibilities on request.													
DEVICE	PACKAGE					OPER. TEMP				SCREENING LEVEL			
	C	J	P	E	FN	L*	V	M	Std	D	G/B	B/B	
EF6809 (1.0 MHz)	•	•	•		•	•			•				
	•	•	•				•		•				
	•	•		•					•		•	•	
EF68A09 (1.5 MHz)	•	•	•			•			•				
	•	•	•		•		•		•				
	•	•		•				•	•		•	•	
EF68B09 (2.0 MHz)		•	•			•			•				
	•	•					•		•		•		
Examples : EF6809C, EF6809CV, EF6809CM													
Package : C : Ceramic DIL, J : Cerdip DIL, P : Plastic DIL, E : LCCC, FN : PLCC.													
Oper. temp. : L* : 0°C to +70°C, V : -40°C to +85°C, M : -55°C to +125°C, * : may be omitted.													
Screening level : Std : (no-end suffix), D : NFC 96883 level D,													
G/B : NFC 96883 level G, B/B : NFC 96883 level B and MIL-STD-883C level B.													



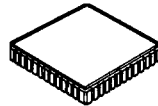
CB-521



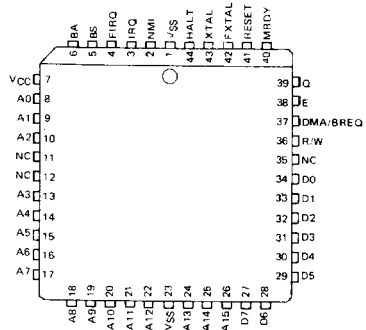
FN SUFFIX  
PLCC 44



CB-708



E SUFFIX  
LCCC 44



These specifications are subject to change without notice.  
Please inquire with our sales offices about the availability of the different packages.

NOTES

---

This Page Intentionally Left Blank