HD6809,HD68A09,HD68B09

MPU (Micro Processing Unit)

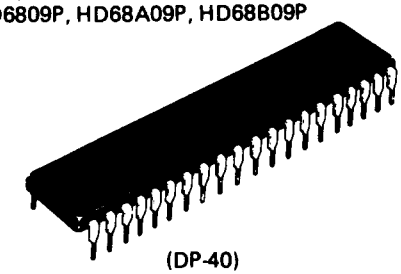
The HD6809 is a revolutionary high performance 8-bit microprocessor which supports modern programming techniques such as position independence, reentrancy, and modular programming.

This third-generation addition to the HMCS6800 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 HD6809 has the most complete set of addressing modes available on any 8-bit microprocessor today.

The HD6809 has hardware and software features which make it an ideal processor for higher level language execution or standard controller applications.

HD6809P, HD68A09P, HD68B09P



(DP-40)

HD6800 COMPATIBLE

- Hardware – Interfaces with All HMCS6800 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

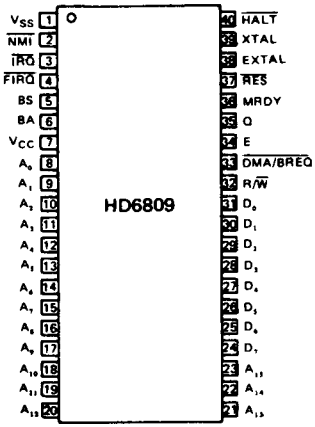
■ HARDWARE FEATURES

- On Chip Oscillator
- $\overline{\text{DMA/BREQ}}$ Allows DMA Operation or 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 Blocked After RESET Until After First Load of Stack Pointer
- Early Address Valid Allows Use With Slower Memories
- Early Write-Data for Dynamic Memories
- Compatible with MC6809, MC68A09 and MC68B09

■ SOFTWARE FEATURES

- 10 Addressing Modes
  - HMCS6800 Upward Compatible Addressing Modes
  - Direct Addressing Anywhere in Memory Map
  - Long Relative Branches
  - Program Counter Relative
  - True Indirect Addressing
  - Expanded Indexed Addressing:

■ PIN ARRANGEMENT



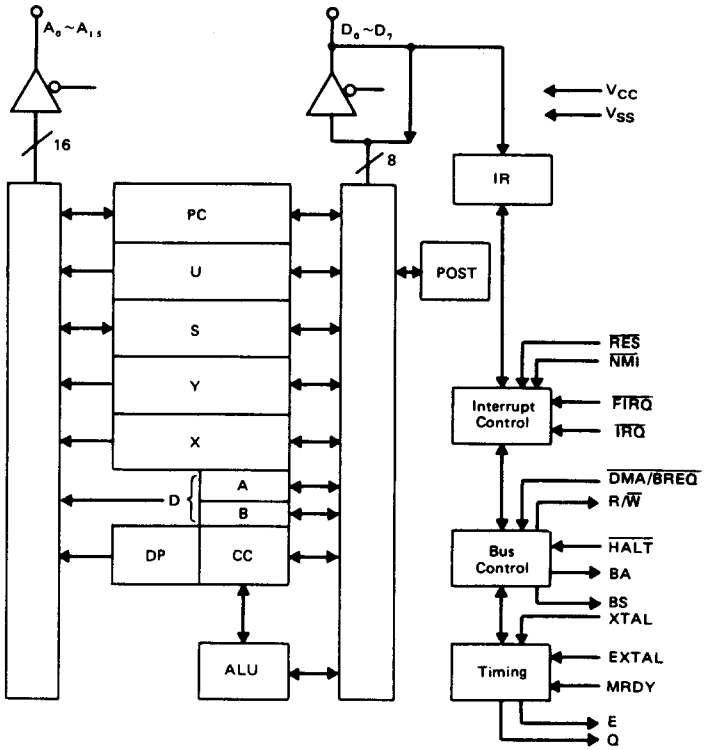
(Top View)

2

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 x 8 Unsigned Multiply
- 16-bit Arithmetic
- Transfer/Exchange All Registers
- Push/Pull Any Registers or Any Set of Registers
- Load Effective Address

■ BLOCK DIAGRAM



■ ABSOLUTE MAXIMUM RATINGS

Item	Symbol	Value	Unit
Supply Voltage	V_{CC}*	-0.3 ~ +7.0	V
Input Voltage	V_{in}*	-0.3 ~ +7.0	V
Operating Temperature	T_{opr}	-20 ~ +75	°C
Storage Temperature	T_{stg}	-55 ~ +150	°C

* With respect to V_{SS} (SYSTEM GND)

(NOTE) Permanent LSI damage may occur if maximum ratings are exceeded. Normal operation should be under recommended operating conditions. If these conditions are exceeded, it could affect reliability of LSI.

■ RECOMMENDED OPERATING CONDITIONS

Item	Symbol		min	typ	max	Unit
Supply Voltage	V_{CC}*		4.75	5.0	5.25	V
Input Voltage	V_{IL}*		-0.3	–	0.8	V
	V_{IH}*	Logic (Ta = 0 ~ +75°C)	2.0	–	V_{CC}	V
		Logic (Ta = -20 ~ 0°C)	2.2	–	V_{CC}	
		$\overline{RES}$	4.0	–	V_{CC}	
Operating Temperature	T_{opr}		-20	25	75	°C

* With respect to V_{SS} (SYSTEM GND)

■ ELECTRICAL CHARACTERISTICS

• DC CHARACTERISTICS (V_{CC} =5V±5%, V_{SS} = 0V, Ta = -20~+75°C, unless otherwise noted.)

Item		Symbol	Test Condition	HD6809			HD68A09			HD68B09			Unit
				min	typ*	max	min	typ*	max	min	typ*	max	
Input "High" Voltage	Except $\overline{RES}$	V_{IH}	Ta = 0 ~ +75°C	2.0	–	V_{CC}	2.0	–	V_{CC}	2.0	–	V_{CC}	V
			Ta = -20 ~ 0°C	2.2	–	V_{CC}	2.2	–	V_{CC}	2.2	–	V_{CC}	
	$\overline{RES}$			4.0	–	V_{CC}	4.0	–	V_{CC}	4.0	–	V_{CC}	
Input "Low" Voltage		V_{IL}		-0.3	–	0.8	-0.3	–	0.8	-0.3	–	0.8	V
Input Leakage Current	Except EXTAL, XTAL	Iin	Vin=0~5.25V, V_{CC}=max	-2.5	–	2.5	-2.5	–	2.5	-2.5	–	2.5	μA
Three State (Off State) Input Current	D_0~D_7	I_{TSI}	Vin=0.4~2.4V, V_{CC}=max	-10	–	10	-10	–	10	-10	–	10	μA
	A_0~A_{15}, R/$\overline{W}$			-100	–	100	-100	–	100	-100	–	100	
Output "High" Voltage	D_0~D_7	V_{OH}	I_{LOAD}=-205μA, V_{CC}=min	2.4	–	–	2.4	–	–	2.4	–	–	V
	A_0~A_{15}, R/$\overline{W}$, Q, E		I_{LOAD}=-145μA, V_{CC}=min	2.4	–	–	2.4	–	–	2.4	–	–	
	BA, BS		I_{LOAD}=-100μA, V_{CC}=min	2.4	–	–	2.4	–	–	2.4	–	–	
Output "Low" Voltage		V_{OL}	I_{LOAD}=2mA	–	–	0.5	–	–	0.5	–	–	0.5	V
Power Dissipation		P_D		–	–	1.0	–	–	1.0	–	–	1.0	W
Input Capacitance	D_0~D_7	Cin	Vin=0V, Ta=25°C, f=1MHz	–	10	15	–	10	15	–	10	15	pF
	Except D_0~D_7			–	7	10	–	7	10	–	7	10	
Output Capacitance	A_0~A_{15}, R/$\overline{W}$, BA, BS	Cout		–	–	12	–	–	12	–	–	12	pF

*Ta=25°C, V_{CC}=5V

- **AC CHARACTERISTICS (V_{CC}=5V±5%, V_{SS} = 0V, Ta = -20~+75°C, unless otherwise noted.)**

1. CLOCK TIMING

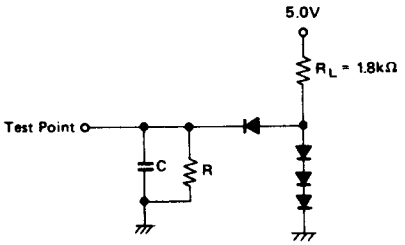
Item	Symbol	Test Condition	HD6809			HD68A09			HD68B09			Unit
			min	typ	max	min	typ	max	min	typ	max	
Frequency of Operation (Crystal or External Input)	f_{XTAL}	Fig. 2, Fig. 3	0.4	–	4	0.4	–	6	0.4	–	8	MHz
Cycle Time	t_{cyc}		1000	–	10000	667	–	10000	500	–	10000	ns
Total Up Time	t_{UT}		975	–	–	640	–	–	480	–	–	ns
Processor Clock "High"	t_{PWEH}		450	–	15500	280	–	15700	220	–	15700	ns
Processor Clock "Low"	t_{PWEL}		430	–	5000	280	–	5000	210	–	5000	ns
E Rise and Fall Time	t_{Er}, t_{Ef}		–	–	25	–	–	25	–	–	20	ns
E_{Low} to Q_{High} Time	t_{AVS}		200	–	250	130	–	165	80	–	125	ns
Q Clock "High"	t_{PWQH}		450	–	5000	280	–	5000	220	–	5000	ns
Q Clock "Low"	t_{PWQL}		450	–	15500	280	–	15700	220	–	15700	ns
Q Rise and Fall Time	t_{Qr}, t_{Qf}		–	–	25	–	–	25	–	–	20	ns
Q_{Low} to E Falling	t_{QE}		200	–	–	133	–	–	100	–	–	ns

2. BUS TIMING

Item		Symbol	Test Condition	HD6809			HD68A09			HD68B09			Unit
				min	typ	max	min	typ	max	min	typ	max	
Address Delay		t_{AD}	Fig. 2, Fig. 3	–	–	200	–	–	140	–	–	110	ns
Address Valid to Q_{High}		t_{AQ}		50	–	–	25	–	–	15	–	–	ns
Peripheral Read Access Time ($t_{UT}-t_{AD}-t_{DSR}=t_{ACC}$)		t_{ACC}		695	–	–	440	–	–	330	–	–	ns
Data Set Up Time (Read)		t_{DSR}		80	–	–	60	–	–	40	–	–	ns
Input Data Hold Time		t_{DHR}		10	–	–	10	–	–	10	–	–	ns
Address Hold Time	A_0~A_{15}, R/$\overline{W}$	t_{AH}	Fig. 2, Fig. 3 Ta=0~+75°C	20	–	–	20	–	–	20	–	–	ns
			Fig. 2, Fig. 3 Ta=-20~0°C	10	–	–	10	–	–	10	–	–	ns
Data Delay Time (Write)		t_{DDW}	Fig. 3	–	–	200	–	–	140	–	–	110	ns
Output Hold Time		t_{DHW}	Fig. 3 Ta=0~+75°C	30	–	–	30	–	–	30	–	–	ns
			Fig. 3 Ta=-20~0°C	20	–	–	20	–	–	20	–	–	ns

3. PROCESSOR CONTROL TIMING

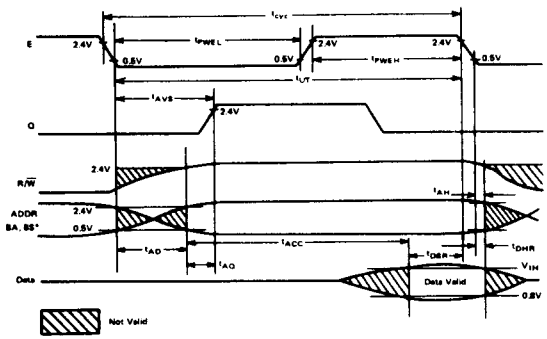
Item	Symbol	Test Condition	HD6809			HD68A09			HD68B09			Unit
			min	typ	max	min	typ	max	min	typ	max	
MRDY Set Up Time	t_{PCSM}	Fig. 6~Fig. 10 Fig. 14, Fig. 15	125	–	–	125	–	–	110	–	–	ns
Interrupts Set Up Time	t_{PCS}		200	–	–	140	–	–	110	–	–	ns
$\overline{HALT}$ Set Up Time	t_{PCSH}		200	–	–	140	–	–	110	–	–	ns
$\overline{RES}$ Set Up Time	t_{PCSR}		200	–	–	140	–	–	110	–	–	ns
DMA/$\overline{BREQ}$ Set Up Time	t_{PCSD}		125	–	–	125	–	–	110	–	–	ns
Processor Control Rise and Fall Time	t_{PCr}, t_{PCf}		–	–	100	–	–	100	–	–	100	ns
Crystal Oscillator Start Time	t_{RC}		–	–	50	–	–	30	–	–	30	ms



- C = 30pF (BA, BS)
  130pF ($D_0 \sim D_7$, E, Q)
  90pF ($A_0 \sim A_{15}$, R/$\overline{W}$)
- R = 11kΩ ($D_0 \sim D_7$)
  16kΩ ($A_0 \sim A_{15}$, E, Q, R/$\overline{W}$)
  24kΩ (BA, BS)

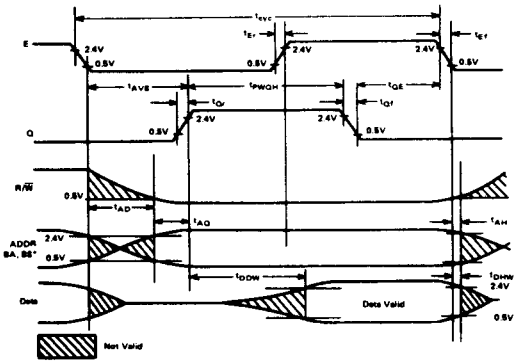
All diodes are 1S2074(H) or equivalent.
C includes Stray Capacitance.

Figure 1 Bus Timing Test Load



*Hold time for BA, BS not specified.

Figure 2 Read Data from Memory or Peripherals



*Hold time for BA, BS not specified.

Figure 3 Write Data to Memory or Peripherals

■ PROGRAMMING MODEL

As shown in Figure 4, the HD6809 adds three registers to the set available in the HD6800. 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 HD6809 serves to enhance the Direct Addressing Mode. The content of this register appears at the higher address outputs ($A_8 \sim A_{15}$) during Direct Addressing Instruction execution. This allows the direct mode to be used at any place in memory, under program control. To ensure HD6800 compatibility, all bits of this register are cleared during Processor Reset.

- **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.

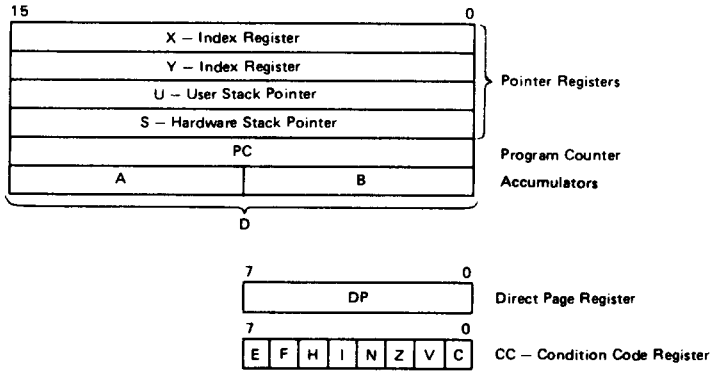


Figure 4 Programming Model of The Microprocessing Unit

- **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 HD6809 point to the top of the stack, in contrast to the HD6800 stack pointer, which pointed to the next free location on the stack. The User Stack Pointer (U) is controlled exclusively by the programmer thus allowing 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 HD6809 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 Fig. 5.

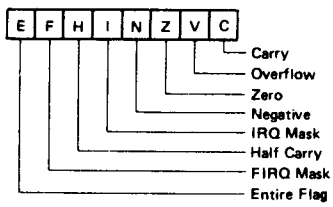


Figure 5 Condition Code Register Format

■ 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-1.

- **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 $\overline{IRQ}$ mask bit. The processor will not recognize interrupts from the $\overline{IRQ}$ line if this bit is set to a one. $\overline{NMI}$, $\overline{FIRQ}$, $\overline{IRQ}$, $\overline{RES}$, and SWI all are 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 $\overline{FIRQ}$ mask bit. The processor will not recognize interrupts from the $\overline{FIRQ}$ line if this bit is a one. $\overline{NMI}$, $\overline{FIRQ}$, SWI, and $\overline{RES}$ all set F to a one. $\overline{IRQ}$, 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.

■ SIGNAL DESCRIPTION

- **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 +5.0V ±5%.

- **Address Bus (A_0~A_{15})**

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 $FFFF_{16}$, R/$\overline{W}$ = "High", and BS = "Low"; this is a "dummy access" or $\overline{VMA}$ cycle. Addresses are valid on the rising edge of Q (see Figs. 2 and 3). All address bus drivers are made high impedance when output Bus Availalbe (BA) is "High". Each pin will drive one Schottky TTL load or four LS TTL loads, and typically 90 pF.

- **Data Bus (D_0~D_7)**

These eight pins provide communication with the system bi-directional data bus. Each pin will drive one Schottky TTL load or four LS TTL loads, and typically 130 pF.

- **Read/Write (R/$\overline{W}$)**

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/$\overline{W}$ is made high impedance when BA is "High". R/$\overline{W}$ is valid on the rising edge of Q. Refer to Figs. 2 and 3.

- **Reset ($\overline{RES}$)**

A "Low" level on this Schmitt-trigger input for greater than one bus cycle will reset the MPU, as shown in Fig. 6. The Reset vectors are fetched from locations $FFFE_{16}$ and $FFFF_{16}$ (Table 1) when Interrupt Acknowledge is true, ($\overline{BA}$ · BS=1). During initial power-on, the Reset line should be held "Low" until the clock oscillator is fully operational. See Fig. 7.

Because the HD6809 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.

Table 1 Memory Map for Interrupt Vectors

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

- **$\overline{HALT}$**

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 ($\overline{FIRQ}$, $\overline{IRQ}$) although $\overline{DMA/BREQ}$ will always be accepted, and $\overline{NMI}$ or $\overline{RES}$ will be latched for later response. During the Halt state Q and E continue to run normally. If the MPU is not running ($\overline{RES}$, $\overline{DMA/BREQ}$), a halted state (BA·BS=1) can be achieved by pulling $\overline{HALT}$ "Low" while $\overline{RES}$ is still "Low". If $\overline{DAM/BREQ}$ and $\overline{HALT}$ 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 Figs. 8 and 16.

- **Bus Available, Bus Status (BA, BS)**

The BA 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", an additional dead cycle will elapse before the MPU acquires the bus.

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

Table 2 MPU State Definition

BA	BS	MPU State
0	0	Normal (Running)
0	1	Interrupt or RESET Acknowledge
1	0	SYNC Acknowledge
1	1	HALT or Bus Grant

Interrupt Acknowledge is indicated during both cycles of a hardware-vector-fetch ($\overline{RES}$, $\overline{NMI}$, $\overline{FIRQ}$, $\overline{IRQ}$, 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 HD6809 is in a Halt or Bus Grant condition.

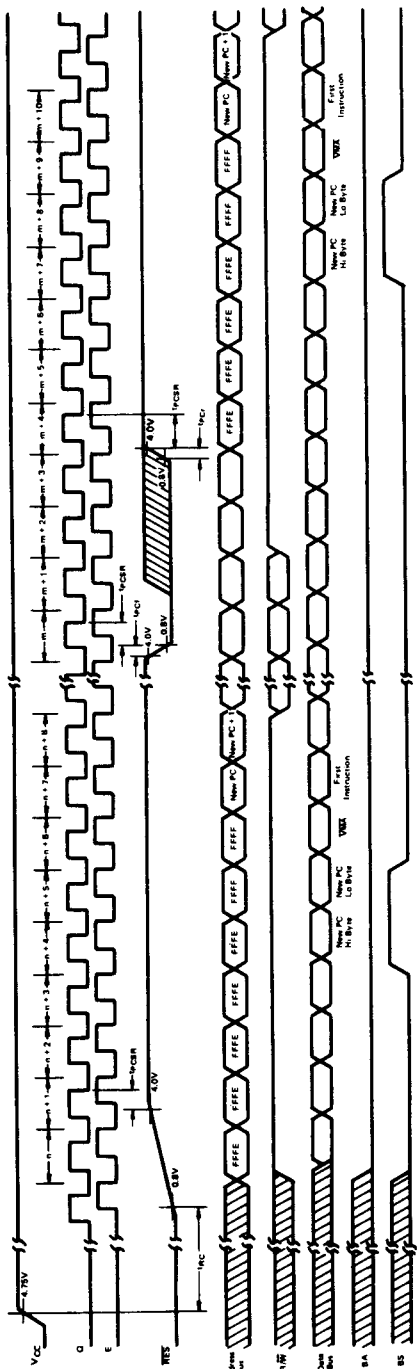
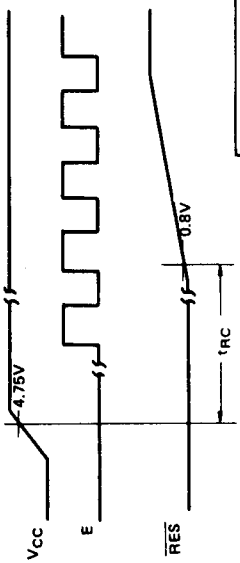
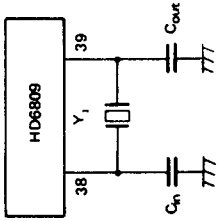


Figure 6 $\overline{RES}$ Timing





Y_1	C_{in}	C_{out}
8 MHz	18pF±20%	18 pF ± 20%
6 MHz	22pF±20%	22 pF ± 20%
4 MHz	22pF±20%	22 pF ± 20%

Figure 7 Crystal Connections and Oscillator Start Up

- **Non Maskable Interrupt ($\overline{NMI}$)***

A negative edge 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 $\overline{NMI}$, the entire machine state is saved on the hardware stack. After reset, an $\overline{NMI}$ will not be recognized until the first program load of the Hardware Stack Pointer (S). The pulse width of $\overline{NMI}$ "Low" must be at least one E cycle. If the $\overline{NMI}$ input does not meet the minimum set up with respect to Q, the interrupt will not be recognized until the next cycle. See Fig. 9.

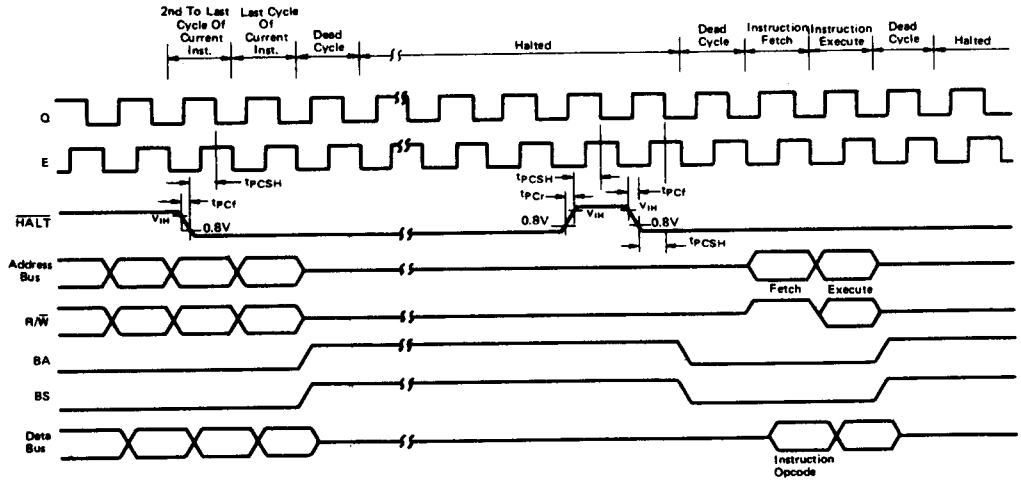


Figure 8 $\overline{HALT}$ and Single Instruction Execution for System Debug

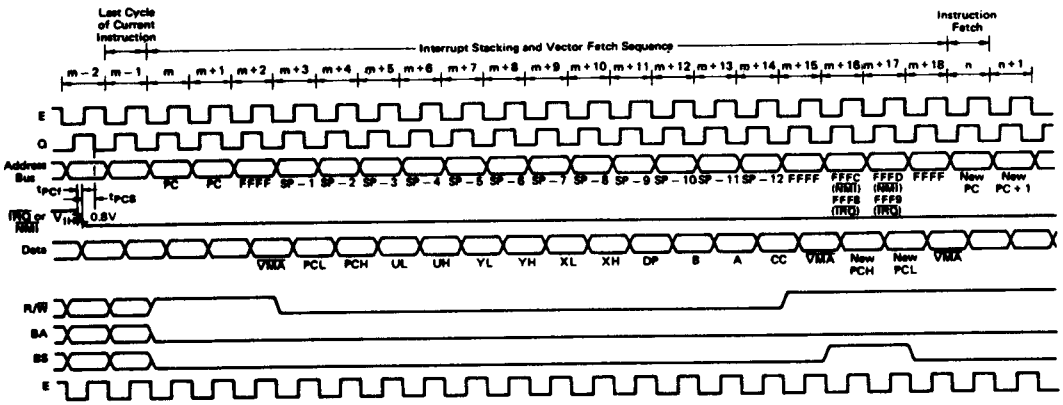


Figure 9 $\overline{IRQ}$ and $\overline{NMI}$ Interrupt Timing

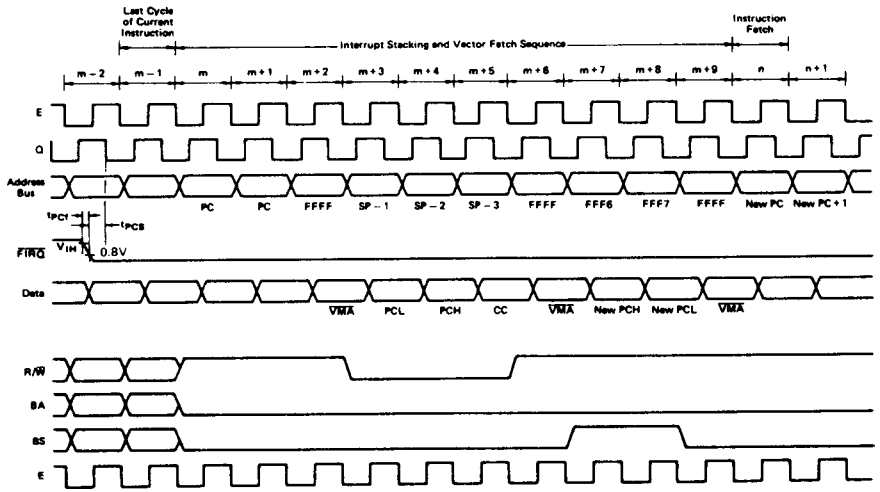


Figure 10 $\overline{\text{FIRQ}}$ Interrupt Timing

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

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 ($\overline{\text{IRQ}}$), 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 Fig. 10.

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

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

* $\overline{\text{NMI}}$, $\overline{\text{FIRQ}}$, and $\overline{\text{IRQ}}$ 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 CWAI condition is present. If $\overline{\text{IRQ}}$ and $\overline{\text{FIRQ}}$ do not remain "Low" until completion of the current instruction they may not be recognized. However, $\overline{\text{NMI}}$ is latched and need only remain "Low" for one cycle.

- **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 Fig. 7. Proper RF layout techniques should be observed in the layout of printed circuit boards.

<NOTE FOR BOARD DESIGN OF THE OSCILLATION CIRCUIT>

In designing the board, the following notes should be taken when the crystal oscillator is used.

1) Crystal oscillator and load capacity Cin, Cout must be placed near the LSI as much as possible.

〔Normal oscillation may be disturbed when external noise is induced to pin 38 and 39.〕

2) Pin 38 and 39 signal line should be wired apart from other signal line as much as possible. Don't wire them in parallel.

〔Normal oscillation may be disturbed when E or Q signal is feedbacked to pin 38 and 39.〕

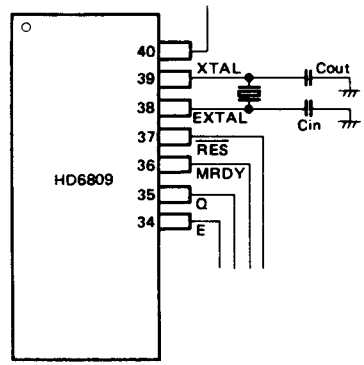


Figure 11 Board Design of the Oscillation Circuit.

<THE FOLLOWING DESIGN MUST BE AVOIDED>

A signal line or a power source line must not cross or go near the oscillation circuit line as shown in Fig. 12 to prevent the induction from these lines and perform the correct oscillation. The resistance among XTAL, EXTAL and other pins should be over 10MΩ.

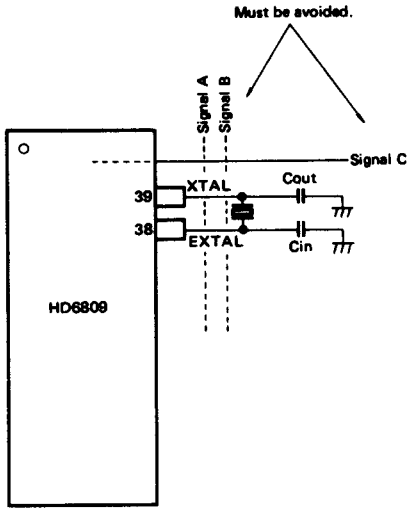


Figure 12 Example of Normal Oscillation may be Disturbed.

- **E, Q**

E is similar to the HD6800 bus timing signal ϕ_2; Q is a quadrature clock signal which leads E. Q has no parallel on the HD6800. 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 Fig. 13.

- **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 (1/4) bus cycles, thus allowing interface to slow memories, as shown in Fig. 14. A maximum

2

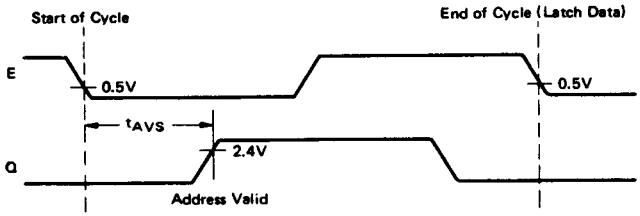


Figure 13 E/Q Relationship

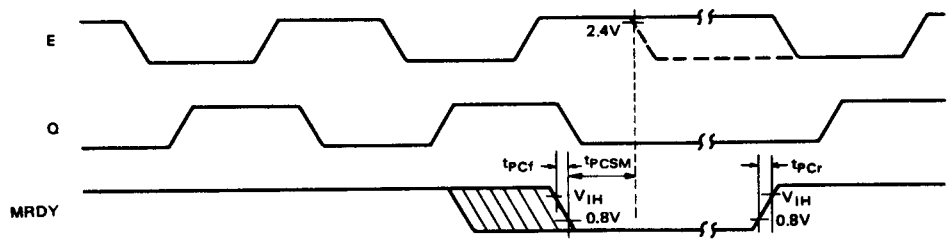


Figure 14 MRDY Timing

stretch is 10 microseconds. During nonvalid memory access ($\overline{VMA}$ 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 $\overline{HALT}$ and $\overline{DMA/BREQ}$).
Also MRDY has effect on stretching E and Q during Dead Cycle.

- **$\overline{DMA/BREQ}$**

The $\overline{DMA/BREQ}$ input provides a method of suspending execution and acquiring the MPU bus for another use, as shown in Fig. 15. Typical uses include DMA and dynamic memory refresh.

Transition of $\overline{DMA/BREQ}$ should occur during Q. A "Low" level on this pin will stop instruction execution at the end of the current cycle. The MPU will acknowledge $\overline{DMA/BREQ}$ by setting BA and BS to "High" level. 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 Fig. 16.

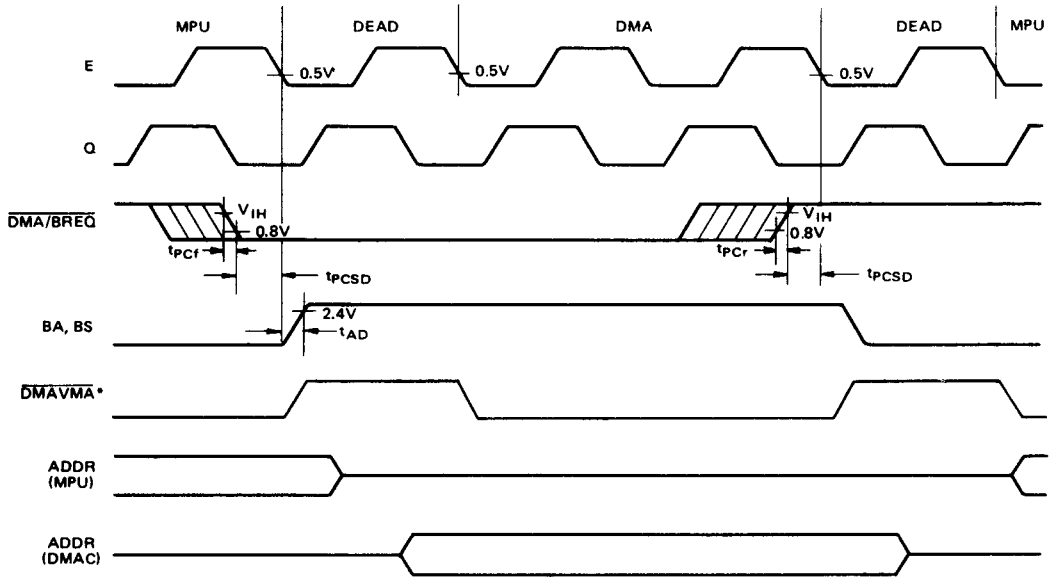
Typically, the DMA controller will request to use the bus by asserting $\overline{DMA/BREQ}$ pin "Low" on the leading edge of E. When the MPU replies by setting BA and BS to a one, that cycle will be a dead cycle used to transfer bus mastership to the DMA controller.

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

When BA goes "Low" (either as a result of $\overline{DMA/BREQ}$ = "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.

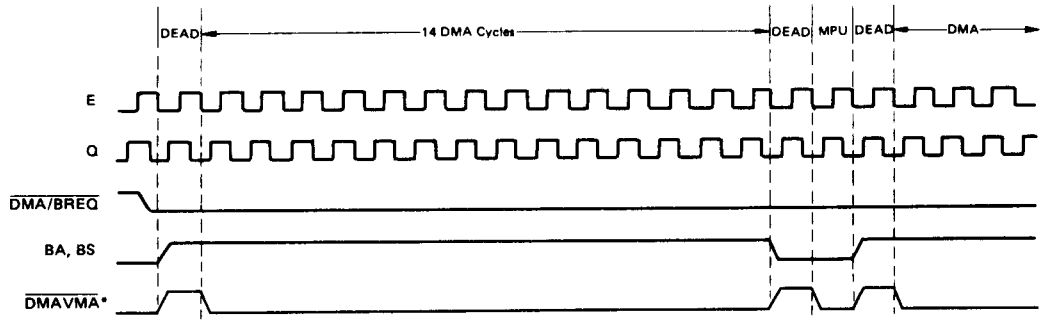
- **MPU OPERATION**

During normal operation, the MPU fetches an instruction from memory and then executes the requested function. This



*$\overline{DMAVMA}$ is a signal which is developed externally, but is a system requirement for DMA.

Figure 15 Typical DMA Timing (<14 Cycles)



*$\overline{DMAVMA}$ is a signal which is developed externally, but is a system requirement for DMA.

Figure 16 Auto – Refresh DMA Timing
(Reverse Cycle Stealing)

sequence begins at $\overline{RES}$ and is repeated indefinitely unless altered by a special instruction or hardware occurrence. Software instructions that alter normal MPU operation are: SWI, $\overline{SWI2}$, $\overline{SWI3}$, CWAI, RTI and SYNC. An interrupt, $\overline{HALT}$ or $\overline{DMA/BREQ}$ can also alter the normal execution of instructions. Fig. 17 illustrates the flow chart for the HD6809.

■ ADDRESSING MODES

The basic instructions of any computer are greatly enhanced by the presence of powerful addressing modes. The HD6809 has the most complete set of addressing modes available on any microcomputer today. For example, the HD6809 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 HD6809:

(1) Implied (Includes Accumulator)
(2) Immediate
(3) Extended
(4) Extended Indirect
(5) Direct
(6) Register
(7) Indexed
    - Zero-Offset
    - Constant Offset
    - Accumulator Offset
    - Auto Increment/Decrement
(8) Indexed Indirect
(9) Relative
(10) Program Counter Relative

• Implied (Includes Accumulator)

In this addressing mode, the opcode of the instruction contains all the address information necessary. Examples of Implied 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 follows the opcode of the instruction). The HD6809 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
```

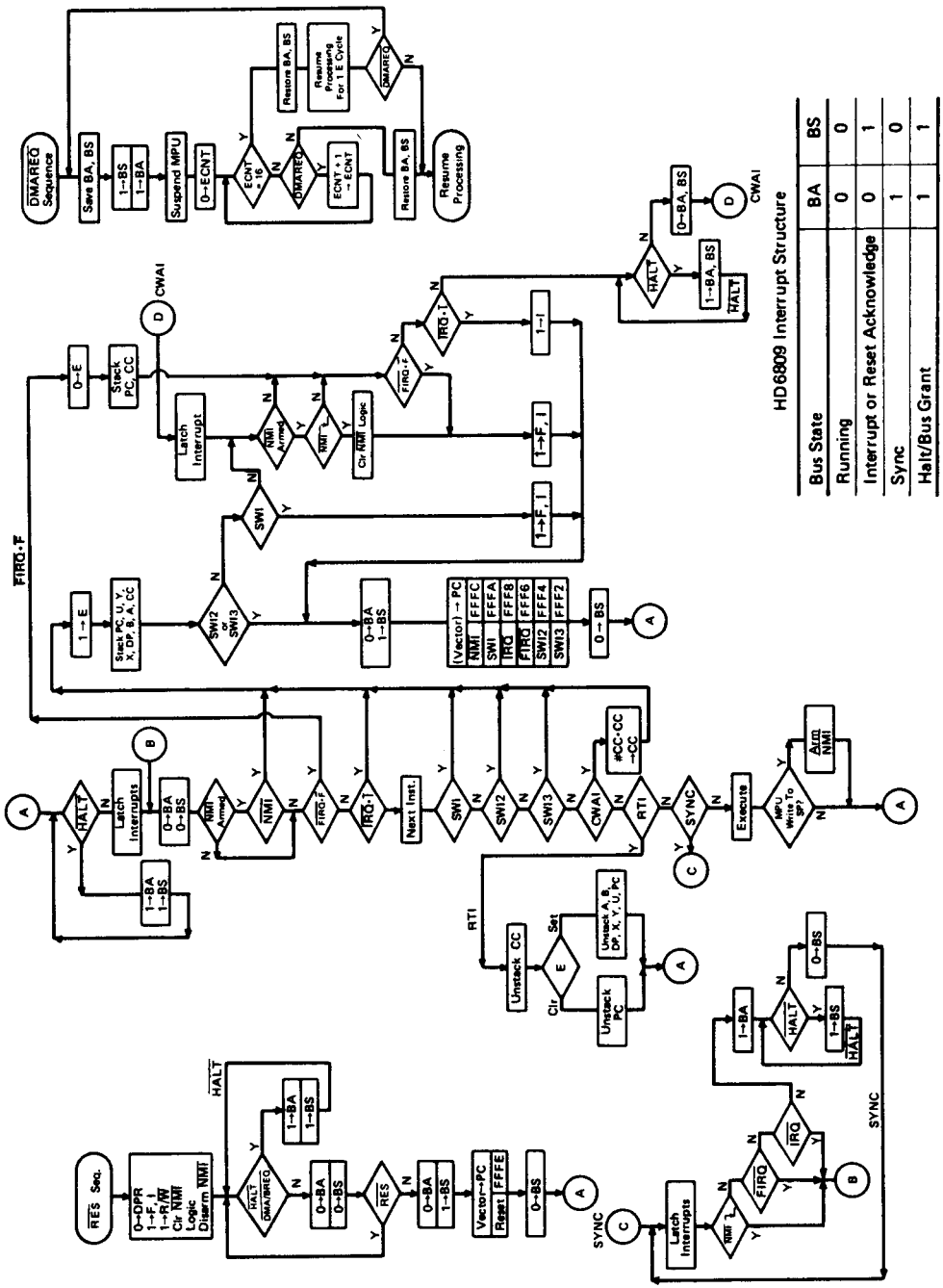
• Extended Indirect

As a special case of indexed addressing (discussed below), "1" 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 8-bit of the address to be used. The upper 8-bit 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



HD6809 Interrupt Structure

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

(NOTE) Asserting $\overline{RES}$ will result in entering the reset sequence from any point in the flow chart.

Figure 17 Flowchart for HD6809 Instruction

accessed without redefining the contents of the DP register. Since the DP register is set to $00 on Reset, direct addressing on the HD6809 is compatible with direct addressing on the HD6800. 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. Fig. 18 lists the legal formats for the postbyte. Table 3 gives the assembler form and the number of cycles and bytes

Post-Byte Register Bit								Indexed Addressing Mode
7	6	5	4	3	2	1	0	
0	R	R	×	×	×	×	×	EA = ,R + 5 Bit Offset
1	R	R	0	0	0	0	0	,R +
1	R	R	0/1	0	0	0	1	,R + +
1	R	R	0	0	0	1	0	, -R
1	R	R	0/1	0	0	1	1	, - - R
1	R	R	0/1	0	1	0	0	EA = ,R + 0 Offset
1	R	R	0/1	0	1	0	1	EA = ,R + ACCB Offset
1	R	R	0/1	0	1	1	0	EA = ,R + ACCA Offset
1	R	R	0/1	1	0	0	0	EA = , R + 8 Bit Offset
1	R	R	0/1	1	0	0	1	EA = ,R + 16 Bit Offset
1	R	R	0/1	1	0	1	1	EA = ,R + D Offset
1	×	×	0/1	1	1	0	0	EA = ,PC + 8 Bit Offset
1	×	×	0/1	1	1	0	1	EA = ,PC + 16 Bit Offset
1	R	R	1	1	1	1	1	EA = [,Address]

Addressing Mode Field

Indirect Field
(Sigh bit when b7 = 0)
{0 ..... Non Indirect
{1 ..... Indirect

Register Field : RR
00 = X
01 = Y
10 = U
11 = S

× = Don't Care

Figure 18 Index Addressing Postbyte Register Bit Assignments

Table 3 Indexed Addressing Mode

Type	Forms	Non Indirect				Indirect			
		Assembler Form	Postbyte OP Code	+ ~	+ #	Assembler Form	Postbyte OP Code	+ ~	+ #
Constant Offset From R (2's Complement Offsets)	No Offset	,R	1RR00100	0	0	[,R]	1RR10100	3	0
	5 Bit Offset	n, R	0RRnnnnn	1	0	defaults to 8-bit			
	8 Bit Offset	n, R	1RR01000	1	1	[n, R]	1RR11000	4	1
	16 Bit Offset	n, R	1RR01001	4	2	[n, R]	1RR11001	7	2
Accumulator Offset From R (2's 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 (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
× = Don't Care

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

$\overset{+}{\sim}$ and $\overset{+}{\#}$ indicate the number of additional cycles and bytes for the particular variation.

added to the basic values for indexed addressing for each variation.

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   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
```

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 allow 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 a 0 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 autoincrement
X → (temp)	do store operation

- **Indexed Indirect**

All of the indexing modes with the exception of auto increment/decrement by one, or a ±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 1 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 (1 byte offset) and long (2 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     LBEQ    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]
```

■ HD6809 INSTRUCTION SET

The instruction set of the HD6809 is similar to that of the HD6800 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 and addressing modes are described in detail below:

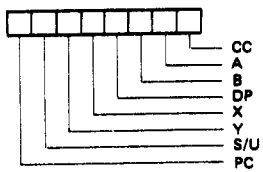
• PSHU/PSHS

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 in below.

PUSH/PULL POST BYTE



← Pull Order Push Order →

PC	U	Y	X	DP	B	A	CC

FFFF...← increasing memory address 0000

PC	S	Y	X	DP	B	A	CC

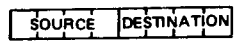
•TFR/EXG

Within the HD6809, 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. Three are denoted as follows:

0000 – D	0101 – PC
0001 – X	1000 – A
0010 – Y	1001 – B
0011 – U	1010 – CC
0100 – S	1011 – DP

(NOTE) All other combinations are undefined and INVALID.

TRANSFER/EXCHANGE POST BYTE



• 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 4.

The LEA instruction also allows the user to access data 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 autoincrement and autodecrement 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)

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

Table 4 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 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

- **MUL**

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

Long And Short Relative Branches

The HD6809 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 ($\overline{NMI}$) or maskable ($\overline{FIRQ}$, $\overline{IRQ}$) with its mask bit (F or I) clear, the processor will clear the Syne 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 ($\overline{FIRQ}$, $\overline{IRQ}$) with its mask bit (F or I) set, the processor will clear the Sync state and continue processing by executing the next inline instruction. Fig. 19 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 this HD6809, and are prioritized in the following order: SWI, SWI2, SWI3.

16-Bit Operation

The HD6809 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 illustrates the memory-access sequence corresponding to each possible instruction and addressing mode in the HD6809. 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 flow chart. $\overline{VMA}$ is an indication of $FFFF_{16}$ on the address bus, R/$\overline{W}$="High" and BS="Low". The following examples illustrate the use of the chart; see Fig. 20.

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

```
                    .
                    .
                    .
$8000               LBSR      CAT
                    .
                    .
                    .
$A000   CAT         .
```

CYCLE-BY-CYCLE FLOW

Cycle #	Address	Data	R/$\overline{W}$	Description
1	8000	17	1	Opcode Fetch
2	8001	1F	1	Offset High Byte
3	8002	FD	1	Offset Low Byte
4	FFFF	*	1	$\overline{VMA}$ Cycle
5	FFFF	*	1	$\overline{VMA}$ Cycle
6	A000	*	1	Computed Branch Address
7	FFFF	*	1	$\overline{VMA}$ Cycle
8	EFFF	03	0	Stack Low Order Byte of Return Address
9	EFFE	80	0	Stack High Order Byte of Return Address

Example 2: DEC (Extended)

```
$8000    DEC     $A000
$A000    FCB     $80
```

CYCLE-BY-CYCLE FLOW

Cycle #	Address	Data	R/$\overline{W}$	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	$\overline{VMA}$ Cycle
5	A000	80	1	Read the Data
6	FFFF	*	1	$\overline{VMA}$ Cycle
7	A000	7F	0	Store the Decremented Data

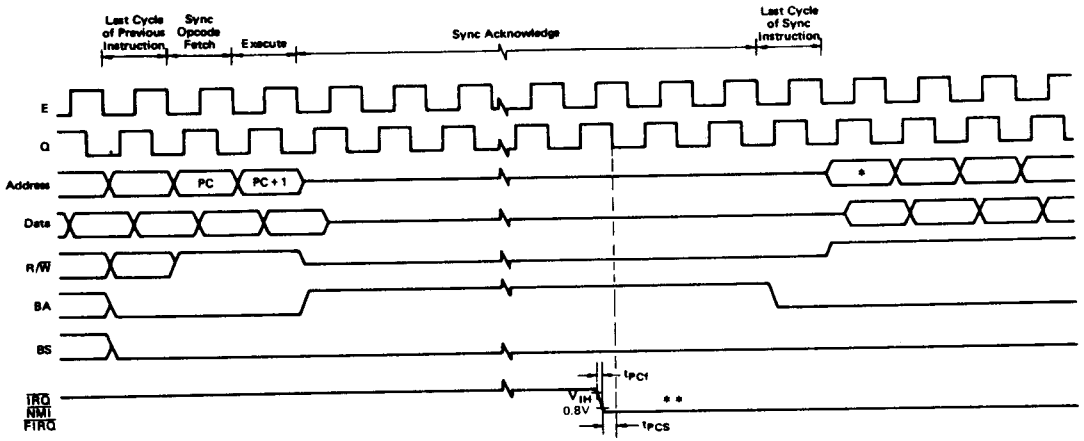
* The data bus has the data at that particular address.

■ HD6809 INSTRUCTION SET TABLES

The instructions of the HD6809 have been broken down into five different categories. They are as follows:

8-Bit operation (Table 5)
16-Bit operation (Table 6)
Index register/stack pointer instructions (Table 7)
Relative branches (long or short) (Table 8)
Miscellaneous instructions (Table 9)

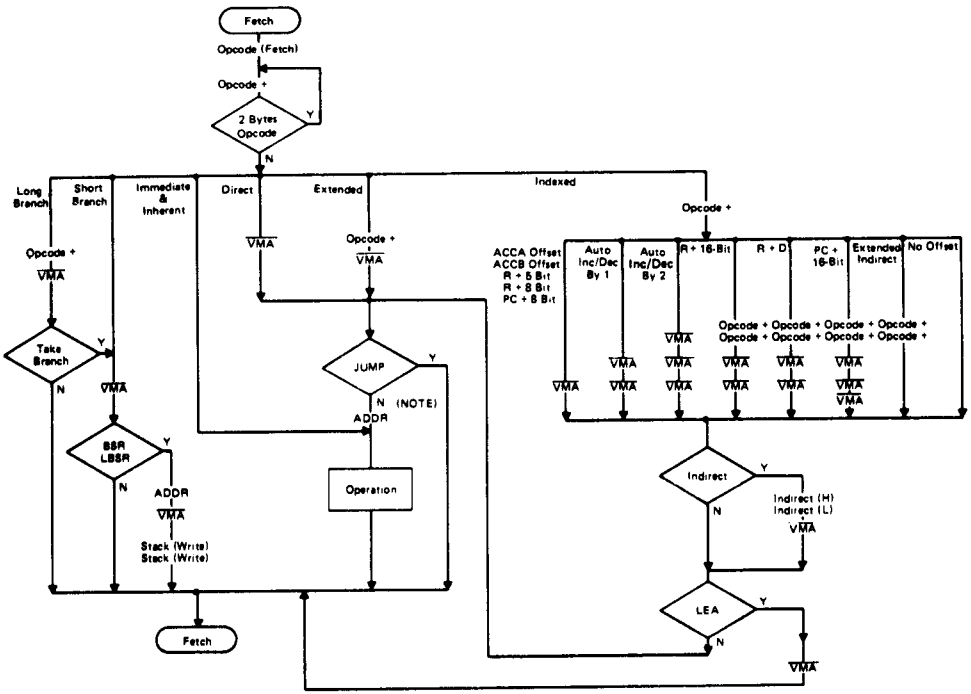
HD6809 instruction set tables and Hexadecimal Values of instructions are shown in Table 10 and Table 11.



(NOTES) * 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 FIRQ or IRQ) interrupt processing continues with this cycle as (m) on Figure 9 and 10 (Interrupt Timing).

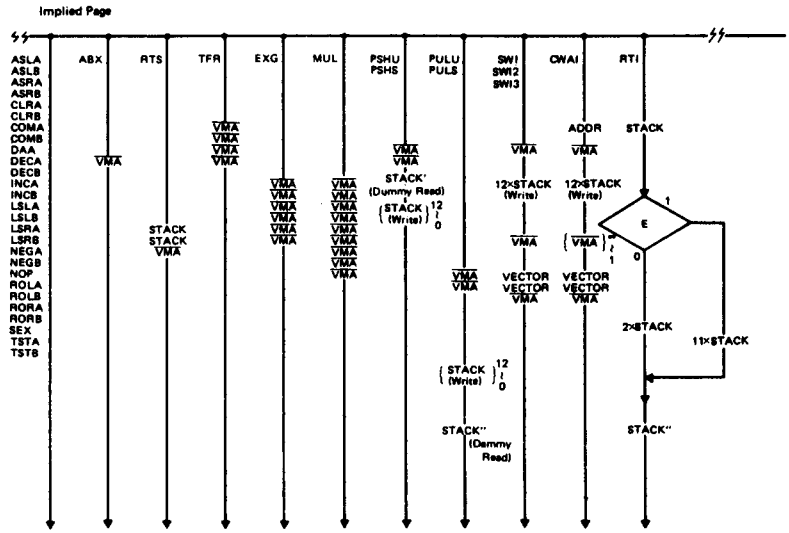
** If mask bits are clear, IRQ and FIRQ must be held "Low" for three cycles to guarantee that interrupt will be taken, although only one cycle is necessary to bring the processor out of SYNC.

Figure 19 Sync Timing



(NOTE) Write operation during store instruction.

Figure 20 Address Bus Cycle-by-Cycle Performance



(NOTE) STACK' : Address stored in stack pointer before execution.
STACK": Address set to stack pointer as the result of the execution.

Figure 20 Address Bus Cycle-by-Cycle Performance (Continued)

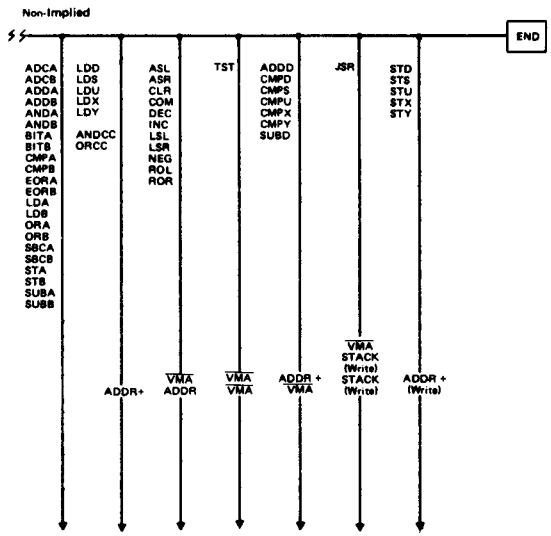


Figure 20 Address Bus Cycle-by-Cycle Performance (Continued)

Table 5 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 accumultor or memory location
DAA	Decimal adjust A accumulator
DEC, DECA, DECB	Decrement accumulator or memory location
EORA, EORB	Exclusive or memory with accumulator
EXG R1, R2	Exchange R1 with R2 (R1, R2 = A, B, CC, DP)
INC, INCA, INCB	Increment accumulator or memory location
LDA, LDB	Load accumulator from memory
LSL, LSLA, LSLB	Logical shift left accumulator or memory location
LSR, LSRA, LSRB	Logical shift right accumulator or memory location
MUL	Unsigned multiply ($A \times B \rightarrow D$)
NEG, NEGA, NEGB	Negate accumulator or memory
ORA, ORB	Or memory with accumulator
ROL, ROLA, ROLB	Rotate accumulator or memory left
ROR, RORA, RORB	Rotate accumulator or memory right
SBCA, SBCB	Subtract memory from accumulator with borrow
STA, STB	Store accumulator to 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) either stack with PSHS, PSHU (PULS, PULU) instructions.

Table 6 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 either stack with PSHS, PSHU (PULS, PULU) instructions.

Table 7 Index Register/Stack Pointer Instructions

Mnemonic(s)	Operation
CMPS, CMPU	Compare memory from stack pointer
CMPX, CMPY	Compare memory from index register
EXG R1, R2	Exchange D, X, Y, S, U or PC with D, X, Y, S, U or PC
LEAS, LEAU	Load effective address into stack pointer
LEAX, LEAY	Load effective address into index register
LDS, LDU	Load stack pointer from memory
LDX, LDY	Load index register from memory
PSHS	Push A, B, CC, DP, D, X, Y, U, or PC onto hardware stack
PSHU	Push A, B, CC, DP, D, X, Y, 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 user 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 8 Branch Instructions

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

Table 9 Miscellaneous Instructions

Mnemonic(s)	Operation
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 10. HD6809 Instruction Set Table

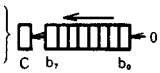
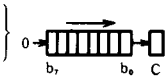
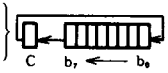
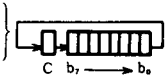
INSTRUCTION/ FORMS		ACCM OP	ACCM ~	ACCM #	INDEX REG OP	INDEX REG ~	INDEX REG #	DIRECT OP	DIRECT ~	DIRECT #	EXTND OP	EXTND ~	EXTND #	IMMED OP	IMMED ~	IMMED #	INDEX① OP	INDEX① ~	INDEX① #	RELATIVE OP	RELATIVE ~	RELATIVE #	DESCRIPTION	7 E	6 F	5 H	4 I	3 N	2 Z	1 V	0 C
ABX		3A	3	1																			B + X → X (UNSIGNED)	●	●	●	●	●	●	●	●
ADC	ADCA							99	4	2	B9	5	3	89	2	2	A9	4 +	2 +				A + M + C → A	●	●	↕	●	↕	↕	↕	↕
	ADCB							D9	4	2	F9	5	3	C9	2	2	E9	4 +	2 +				B + M + C → B	●	●	↕	●	↕	↕	↕	↕
ADD	ADDA							9B	4	2	BB	5	3	8B	2	2	AB	4 +	2 +				A + M → A	●	●	↕	●	↕	↕	↕	↕
	ADDB							DB	4	2	FB	5	3	CB	2	2	EB	4 +	2 +				B + M → B	●	●	↕	●	↕	↕	↕	↕
	ADDD							D3	6	2	F3	7	3	C3	4	3	E3	6 +	2 +				D + M:M + 1 → D	●	●	●	●	↕	↕	↕	↕
AND	ANDA							94	4	2	B4	5	3	84	2	2	A4	4 +	2 +				A ∧ M → A	●	●	●	●	↕	↕	R	●
	ANDB							D4	4	2	F4	5	3	C4	2	2	E4	4 +	2 +				B ∧ M → B	●	●	●	●	↕	↕	R	●
	ANDCC													1C	3	2							CC ∧ IMM → CC	(			⑦				)
ASL	ASLA	48	2	1																			A	●	●	⑧	●	↕	↕	↕	↕
	ASLB	58	2	1																			B	●	●	⑧	●	↕	↕	↕	↕
	ASL							08	6	2	78	7	3				68	6 +	2 +				M	●	●	⑧	●	↕	↕	↕	↕
ASR	ASRA	47	2	1																			A	●	●	⑧	●	↕	↕	●	↕
	ASRB	57	2	1																			B	●	●	⑧	●	↕	↕	●	↕
	ASR							07	6	2	77	7	3				67	6 +	2 +				M	●	●	⑧	●	↕	↕	●	↕
BCC	BCC																			24	3	2	Branch C = 0	●	●	●	●	●	●	●	●
	LBCC																			10	5(6)	4	Long Branch C = 0	●	●	●	●	●	●	●	●
BCS	BCS																			25	3	2	Branch C = 1	●	●	●	●	●	●	●	●
	LBCS																			10	5(6)	4	Long Branch C = 1	●	●	●	●	●	●	●	●
BEQ	BEQ																			27	3	2	Branch Z = 1	●	●	●	●	●	●	●	●
	LBEQ																			10	5(6)	4	Long Branch Z = 1	●	●	●	●	●	●	●	●
BGE	BGE																			2C	3	2	Branch N⊕V = 0	●	●	●	●	●	●	●	●
	LBGE																			10	5(6)	4	Long Branch N⊕V = 0	●	●	●	●	●	●	●	●
BGT	BGT																			2E	3	2	Branch Z∨(N⊕V) = 0	●	●	●	●	●	●	●	●
	LBGT																			10	5(6)	4	Long Branch Z∨(N⊕V) = 0	●	●	●	●	●	●	●	●
BHI	BHI																			22	3	2	Branch C∨Z = 0	●	●	●	●	●	●	●	●
	LBHI																			10	5(6)	4	Long Branch C∨Z = 0	●	●	●	●	●	●	●	●
BHS	BHS																			24	3	2	Branch C = 0	●	●	●	●	●	●	●	●
	LBHS																			10	5(6)	4	Long Branch C = 0	●	●	●	●	●	●	●	●
BIT	BITA							95	4	2	B5	5	3	85	2	2	A5	4 +	2 +				Bit Test A (M∧A)	●	●	●	●	↕	↕	R	●
	BITB							D5	4	2	F5	5	3	C5	2	2	E5	4 +	2 +				Bit Test B (M∧B)	●	●	●	●	↕	↕	R	●
BLE	BLE																			2F	3	2	Branch Z∨(N⊕V) = 1	●	●	●	●	●	●	●	●
	LBLE																			10	5(6)	4	Long Branch Z∨(N⊕V) = 1	●	●	●	●	●	●	●	●
BLO	BLO																			25	3	2	Branch C = 1	●	●	●	●	●	●	●	●
	LBLO																			10	5(6)	4	Long Branch C = 1	●	●	●	●	●	●	●	●
BLS	BLS																			23	3	2	Branch C∨Z = 1	●	●	●	●	●	●	●	●
	LBLS																			10	5(6)	4	Long Branch C∨Z = 1	●	●	●	●	●	●	●	●
BLT	BLT																			2D	3	2	Branch N⊕V = 1	●	●	●	●	●	●	●	●
	LBLT																			10	5(6)	4	Long Branch N⊕V = 1	●	●	●	●	●	●	●	●
BMI	BMI																			2B	3	2	Branch N = 1	●	●	●	●	●	●	●	●
	LBMI																			10	5(6)	4	Long Branch N = 1	●	●	●	●	●	●	●	●

(Continued)

INSTRUCTION/ FORMS		IMP ACCM	IMP REG		DIRECT			EXTND			IMMED			INDEX①			RELATIVE			DESCRIPTION	7	6	5	4	3	2	1	0
		OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~⑤	#		E	F	H	I	N	Z	V	C
BNE	BNE																2 6	3	2	Branch Z = 0	●	●	●	●	●	●	●	●
	LBNE																1 0 2 6	5(6)	4	Long Branch Z = 0	●	●	●	●	●	●	●	●
BPL	BPL																2 A	3	2	Branch N = 0	●	●	●	●	●	●	●	●
	LBPL																1 0 2 A	5(6)	4	Long Branch N = 0	●	●	●	●	●	●	●	●
BRA	BRA																2 0	3	2	Branch Always	●	●	●	●	●	●	●	●
	LBRA																1 6	5	3	Long Branch Always	●	●	●	●	●	●	●	●
BRN	BRN																2 1	3	2	Branch Never	●	●	●	●	●	●	●	●
	LBRN																1 0 2 1	5	4	Long Branch Never	●	●	●	●	●	●	●	●
BSR	BSR																8 D	7	2	Branch to Subroutine	●	●	●	●	●	●	●	●
	LBSR																1 7	9	3	Long Branch to Subroutine	●	●	●	●	●	●	●	●
BVC	BVC																2 8	3	2	Branch ⁻ V = 0	●	●	●	●	●	●	●	●
	LBVC																1 0 2 8	5(6)	4	Long Branch V = 0	●	●	●	●	●	●	●	●
BVS	BVS																2 9	3	2	Branch V = 1	●	●	●	●	●	●	●	●
	LBVS																1 0 2 9	5(6)	4	Long Branch V = 1	●	●	●	●	●	●	●	●
CLR	CLRA	4 F	2	1																0 → A	●	●	●	●	R	S	R	R
	CLRB	5 F	2	1																0 → B	●	●	●	●	R	S	R	R
	CLR				0 F	6	2	7 F	7	3				6 F	6 +	2 +				0 → M	●	●	●	●	R	S	R	R
CMP	CMPA				9 1	4	2	B 1	5	3	8 1	2	2	A 1	4 +	2 +				Compare M from A	●	●	⑧	●	↕	↕	↕	↕
	CMPB				D 1	4	2	F 1	5	3	C 1	2	2	E 1	4 +	2 +				Compare M from B	●	●	⑧	●	↕	↕	↕	↕
	CMPD				1 0	7	3	1 0	8	4	1 0	5	4	1 0	7 +	3 +				Compare M:M + 1	●	●	●	●	↕	↕	↕	↕
					9 3			B 3			8 3			A 3						from D								
	CMPS				1 1	7	3	1 1	8	4	1 1	5	4	1 1	7 +	3 +				Compare M:M + 1	●	●	●	●	↕	↕	↕	↕
					9 C			B C			8 C			A C						from S								
	CMPU				1 1	7	3	1 1	8	4	1 1	5	4	1 1	7 +	3 +				Compare M:M + 1	●	●	●	●	↕	↕	↕	↕
					9 3			B 3			8 3			A 3						from U								
	CMPX				9 C	6	2	B C	7	3	8 C	4	3	A C	6 +	2 +				Compare M:M + 1	●	●	●	●	↕	↕	↕	↕
																				from X								
	CMPY				1 0	7	3	1 0	8	4	1 0	5	4	1 0	7 +	3 +				Compare M:M + 1	●	●	●	●	↕	↕	↕	↕
					9 C			B C			8 C			A C						from Y								
COM	COMA	4 3	2	1																$\overline{A} \rightarrow A$	●	●	●	●	↕	↕	R	S
	COMB	5 3	2	1																$\overline{B} \rightarrow B$	●	●	●	●	↕	↕	R	S
	COM				0 3	6	2	7 3	7	3				6 3	6 +	2 +				$\overline{M} \rightarrow M$	●	●	●	●	↕	↕	R	S
CWAI		3C	20	2																CC ∧ IMM → CC; Wait for Interrupt	S	(	⑦	)				
DAA		1 9	2	1																Decimal Adjust A	●	●	●	●	↕	↕	⑧	↕
DEC	DECA	4 A	2	1																A − 1 → A	●	●	●	●	↕	↕	↕	●
	DECB	5 A	2	1																B − 1 → B	●	●	●	●	↕	↕	↕	●
	DEC				0 A	6	2	7 A	7	3				6 A	6 +	2 +				M − 1 → M	●	●	●	●	↕	↕	↕	●
EOR	EORA				9 8	4	2	B 8	5	3	8 8	2	2	A 8	4 +	2 +				A ⊕ M → A	●	●	●	●	↕	↕	R	●
	EORB				D 8	4	2	F 8	5	3	C 8	2	2	E 8	4 +	2 +				B ⊕ M → B	●	●	●	●	↕	↕	R	●
EXG	R1, R2	1 E	8	2																R1 → R2②	(			⑩				)
INC	INCA	4 C	2	1																A + 1 → A	●	●	●	●	↕	↕	↕	●
	INCB	5 C	2	1																B + 1 → B	●	●	●	●	↕	↕	↕	●
	INC				0 C	6	2	7 C	7	3				6 C	6 +	2 +				M + 1 → M	●	●	●	●	↕	↕	↕	●
JMP					0 E	3	2	7 E	4	3				6 E	3 +	2 +				EA③ → PC	●	●	●	●	●	●	●	●
JSR					9 D	7	2	B D	8	3				A D	7 +	2 +				Jump to Subroutine	●	●	●	●	●	●	●	●

(Continued)

2

INSTRUCTION/ FORMS		ACCM OP	ACCM ~	ACCM #	REG OP	REG ~	REG #	DIRECT OP	DIRECT ~	DIRECT #	EXTND OP	EXTND ~	EXTND #	IMMED OP	IMMED ~	IMMED #	INDEX OP	INDEX ~	INDEX #	RELATIVE OP	RELATIVE ~	RELATIVE #	DESCRIPTION	7 E	6 F	5 H	4 I	3 N	2 Z	1 V	0 C
LD	LDA							96	4	2	B6	5	3	86	2	2	A6	4+	2+				M→A	●	●	●	●	↕	↕	R	●
	LDB							D6	4	2	F6	5	3	C6	2	2	E6	4+	2+				M→B	●	●	●	●	↕	↕	R	●
	LDD							DC	5	2	FC	6	3	CC	3	3	EC	5+	2+				M:M+1→D	●	●	●	●	↕	↕	R	●
	LDS							10 DE	6	3	10 FE	7	4	10 CE	4	4	10 EE	6+	3+				M:M+1→S	●	●	●	●	↕	↕	R	●
	LDU							DE	5	2	FE	6	3	CE	3	3	EE	5+	2+				M:M+1→U	●	●	●	●	↕	↕	R	●
	LDX							9E	5	2	BE	6	3	8E	3	3	AE	5+	2+				M:M+1→X	●	●	●	●	↕	↕	R	●
	LDY							10 9E	6	3	10 BE	7	4	10 8E	4	4	10 AE	6+	3+				M:M+1→Y	●	●	●	●	↕	↕	R	●
LEA	LEAS																32	4+	2+				EA③→S	●	●	●	●	●	●	●	●
	LEAU																33	4+	2+				EA③→U	●	●	●	●	●	●	●	●
	LEAX																30	4+	2+				EA③→X	●	●	●	●	↕	●	●	●
	LEAY																31	4+	2+				EA③→Y	●	●	●	●	●	↕	●	●
LSL	LSLA	48	2	1																			A, B, M } 	●	●	●	●	↕	↕	↕	↕
	LSLB	58	2	1																				●	●	●	●	↕	↕	↕	↕
	LSL							08	6	2	78	7	3				68	6+	2+					●	●	●	●	↕	↕	↕	↕
LSR	LSRA	44	2	1																			A, B, M } 	●	●	●	●	R	↕	●	↕
	LSRB	54	2	1																				●	●	●	●	R	↕	●	↕
	LSR							04	6	2	74	7	3				64	6+	2+					●	●	●	●	R	↕	●	↕
MUL		3D	11	1																			A×B→D (Unsigned)	●	●	●	●	●	↕	●	⑨
NEG	NEGA	40	2	1																			$\overline{A}$+1→A	●	●	⑧	●	↕	↕	↕	↕
	NEGB	50	2	1																			$\overline{B}$+1→B	●	●	⑧	●	↕	↕	↕	↕
	NEG							00	6	2	70	7	3				60	6+	2+				$\overline{M}$+1→M	●	●	⑧	●	↕	↕	↕	↕
NOP		12	2	1																			No Operation	●	●	●	●	●	●	●	●
OR	ORA							9A	4	2	BA	5	3	8A	2	2	AA	4+	2+				A∨M→A	●	●	●	●	↕	↕	R	●
	ORB							DA	4	2	FA	5	3	CA	2	2	EA	4+	2+				B∨M→B	●	●	●	●	↕	↕	R	●
	ORCC													1A	3	2							CC∨IMM→CC	(			⑦				)
PSH	PSHS	34	5-[i]	2																			Push Registers on S Stack	●	●	●	●	●	●	●	●
	PSHU	36	5-[i]	2																			Push Registers on U Stack	●	●	●	●	●	●	●	●
PUL	PULS	35	5-[i]	2																			Pull Registers from S Stack	(			⑩				)
	PULU	37	5-[i]	2																			Pull Registers from U Stack	(			⑩				)
ROL	ROLA	49	2	1																			A, B, M } 	●	●	●	●	↕	↕	↕	↕
	ROLB	59	2	1																				●	●	●	●	↕	↕	↕	↕
	ROL							09	6	2	79	7	3				69	6+	2+					●	●	●	●	↕	↕	↕	↕
ROR	RORA	46	2	1																			A, B, M } 	●	●	●	●	↕	↕	●	↕
	RORB	56	2	1																				●	●	●	●	↕	↕	●	↕
	ROR							06	6	2	76	7	3				66	6+	2+					●	●	●	●	↕	↕	●	↕
RTI		3B	6 15	1																			Return from Interrupt	(			⑦				)
RTS		39	5	1																			Return from Subroutine	●	●	●	●	●	●	●	●
SBC	SBCA							92	4	2	B2	5	3	82	2	2	A2	4+	2+				A−M−C→A	●	●	⑧	●	↕	↕	↕	↕
	SBCB							D2	4	2	F2	5	3	C2	2	2	E2	4+	2+				B−M−C→B	●	●	⑧	●	↕	↕	↕	↕
SEX		1D	2	1																			Sign Extend B into A { Bのビット7=1 FF→A ; Bのビット7=0 0→A }	●	●	●	●	↕	↕	●	●

(Continued)

INSTRUCTIONS/ FORMS		IMP ACCM	IMP REG		DIRECT			EXTND			IMMED			INDEX①			RELATIVE			DESCRIPTION	7	6	5	4	3	2	1	0
		OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#		E	F	H	I	N	Z	V	C
ST	STA				97	4	2	B7	5	3				A7	4+	2+				A→M	●	●	●	●	↕	↕	R	●
	STB				D7	4	2	F7	5	3				E7	4+	2+				B→M	●	●	●	●	↕	↕	R	●
	STD				DD	5	2	FD	6	3				ED	5+	2+				D→M:M+1	●	●	●	●	↕	↕	R	●
	STS				10	6	3	10	7	4				10	6+	3+				S→M:M+1	●	●	●	●	↕	↕	R	●
					DF			FF						EF														
	STU				DF	5	2	FF	6	3				EF	5+	2+				U→M:M+1	●	●	●	●	↕	↕	R	●
	STX				9F	5	2	BF	6	3				AF	5+	2+				X→M:M+1	●	●	●	●	↕	↕	R	●
	STY				10	6	3	10	7	4				10	6+	3+				Y→M:M+1	●	●	●	●	↕	↕	R	●
					9F			BF						AF														
SUB	SUBA				90	4	2	B0	5	3	80	2	2	A0	4+	2+				A−M→A	●	●	⑧	●	↕	↕	↕	↕
	SUBB				D0	4	2	F0	5	3	C0	2	2	E0	4+	2+				B−M→B	●	●	⑧	●	↕	↕	↕	↕
	SUBD				93	6	2	B3	7	3	83	4	3	A3	6+	2+				D−M:M+1→D	●	●	●	●	↕	↕	↕	↕
SWI	SWI⑥	3F	19	1																Software interrupt 1	S	S	●	S	●	●	●	●
	SWI2⑥	10	20	2																Software interrupt 2	S	●	●	●	●	●	●	●
		3F																										
	SWI3⑥	11	20	2																Software interrupt 3	S	●	●	●	●	●	●	●
		3F																										
SYNC		13	≥2	1																Synchronize to interrupt	●	●	●	●	●	●	●	●
TFR	R1, R2	1F	6	2																R1 → R2②	(				⑩			)
TST	TSTA	4D	2	1																Test A	●	●	●	●	↕	↕	R	●
	TSTB	5D	2	1																Test B	●	●	●	●	↕	↕	R	●
	TST				0D	6	2	7D	7	3				6D	6+	2+				Test M	●	●	●	●	↕	↕	R	●

(NOTES)

① This column gives a base cycle and byte count. To obtain total count, and the values obtained from the INDEXED ADDRESSING MODES 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
③ EA is the effective address.
④ The PSH and PUL instructions require 5 cycle plus 1 cycle for each byte pushed or pulled.
⑤ 5(6) means: 5 cycles if branch not taken, 6 cycles if taken.
⑥ SWI sets 1 and F bits. SW12 and SW13 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.
⑩ Condition Codes set as a direct result of the instruction if CC is specified, and not affected otherwise.

LEGEND:

OP	Operation Code (Hexadecimal)	Z	Zero (byte)
~	Number of MPU Cycles	V	Overflow, 2's complement
#	Number of Program Bytes	C	Carry from bit 7
+	Arithmetic Plus	↑	Test and set if true, cleared otherwise
−	Arithmetic Minus	●	Not Affected
×	Multiply	CC	Condition Code Register
M̄	Complement of M	:	Concatenation
→	Transfer Into	∨	Logical or
H	Half-carry (from bit 3)	∧	Logical and
N	Negative (sign bit)	⊕	Logical Exclusive or

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

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

(to be continued)

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

2 Bytes Opcode

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

(NOTE): All unused opcodes are both undefined and illegal

■ **NOTE FOR USE**

[1] **Exceptional Operation of HD6809**

(a) **Exceptional Operations of $\overline{\text{DMA/BREQ}}$, BA signals (#1)**

HD6809 acknowledges the input signal level of $\overline{\text{DMA/BREQ}}$ at the end of each cycle, then determines whether the next sequence is MPU or DMA. When "Low" level is detected, HD6809 executes DMA sequence by setting BA, BS to "High" level. However, in the conditions shown below the assertion of BA, BS delays one clock cycle.

< Conditions for the exception >

(1) $\overline{\text{DMA/BREQ}}$: "Low" for 6~13 cycles

(2) $\overline{\text{DMA/BREQ}}$: "High" for 3 cycles

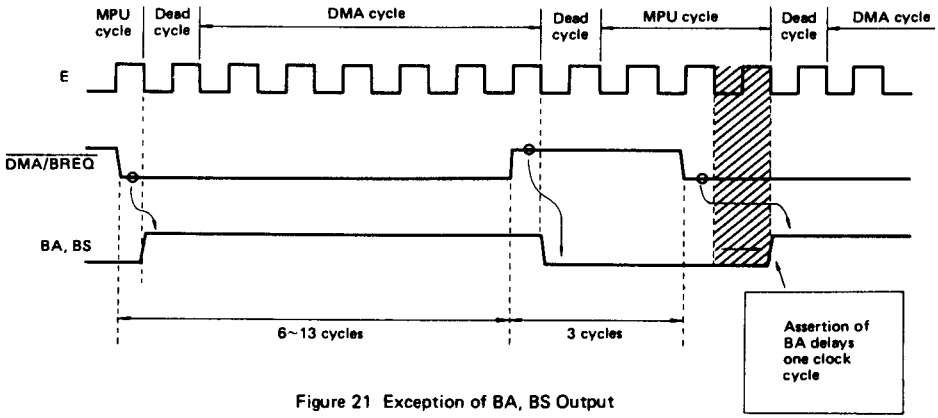


Figure 21 Exception of BA, BS Output

(b) **Exceptional Operations of $\overline{\text{DMA/BREQ}}$, BA signals (#2)**

HD6809 includes a self refresh counter for the reverce cycle steal. And it is only cleared if $\overline{\text{DMA/BREQ}}$ is inactive ("High") for 3 or more MPU cycles. So 1 or 2 inactive cycle(s) doesn't affect the self refresh counter.

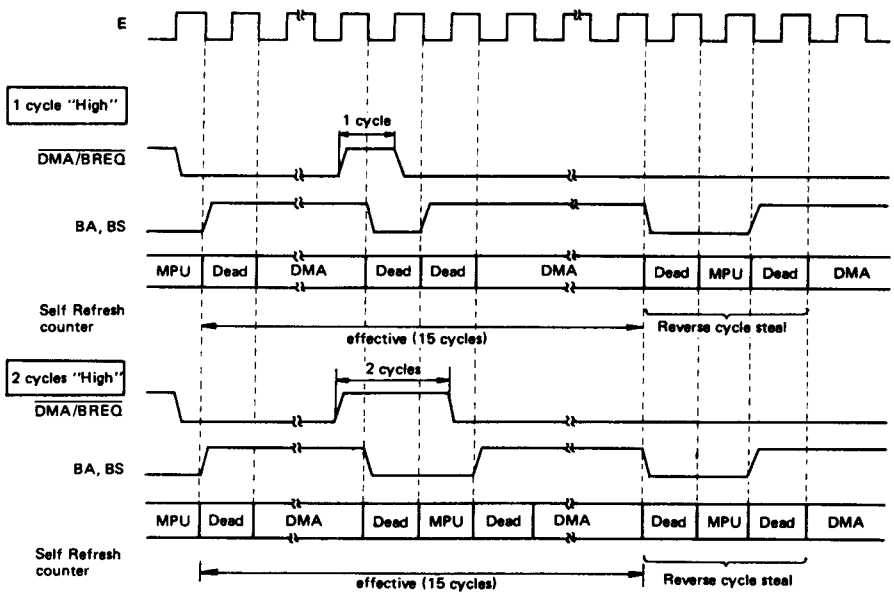


Figure 22 Exception of $\overline{\text{DMA/BREQ}}$

(c) **How to avoid these exceptional operations**

It is necessary to provide 4 or more cycles for inactive $\overline{DMA/BREQ}$ level as shown in Fig. 23.

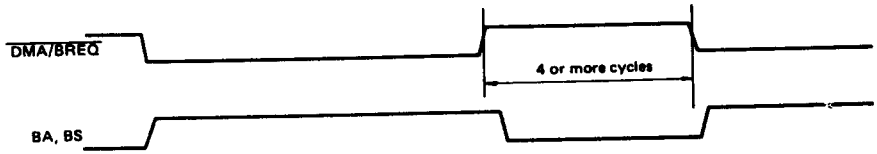


Figure 23 How to Avoid Exceptional Operations

[2] **Restriction for DMA Transfer**

There is a restriction for the DMA transfer in the HD6809 (MPU), HD6844 (DMAC) system. Please take care of following.

(a) **An Example of the System Configuration**

This restriction is applied to the following system.

(1) $\overline{DMA/BREQ}$ is used for DMA request.

(2) "Halt Burst Mode" is used for DMA transfer

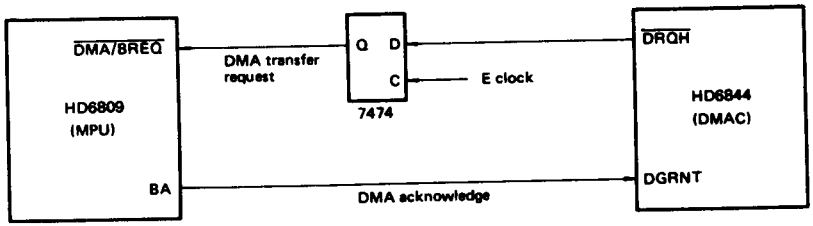


Figure 24 An Example of HD6809, HD6844 System

(The restriction is also applied to the system which doesn't use 7474 Flip-Flop. Fig. 24, Fig. 25 shows an example which uses 7474 for synchronizing DMA request with E.)

(b) **Restriction**

"The number of transfer Byte per one DMA Burst transfer must be less than or equal to 14."

Halt burst DMA transfer should be less than or equal to 14 cycles. In another word, the number stored into DMA Byte count register should be 0~14.

★ Please than care of the section [1](b) if 2 or more DMA channels are used for the DMA transfer.

(c) **Incorrect operation of HD6809, HD6844 system**

"Incorrect Operation" will occur if the number of DMA transfer Byte is more than 14 bytes. If $\overline{DMA/BREQ}$ is kept in "Low" level HD6809 performs reverse cycle steals once in 14 DMA cycles by taking back the bus control. In this case, however, the action taken by MPU is a little bit different from the DMAC.

As shown in Fig. 25, DMA controller can't stop DMA transfer (Ⓐ) by BA falling edge and excutes an extra DMA cycle during HD6809 dead cycle. So MPU cycle is excuted right after DMA cycle, the Bus confliction occurs at the beginning of MPU cycle.

(d) **How to impliment Halt Bust DMA transfer (> 14 cycles)**

Please use $\overline{HALT}$ input of HD6809 for the DMA request instead of $\overline{DMA/BREQ}$.

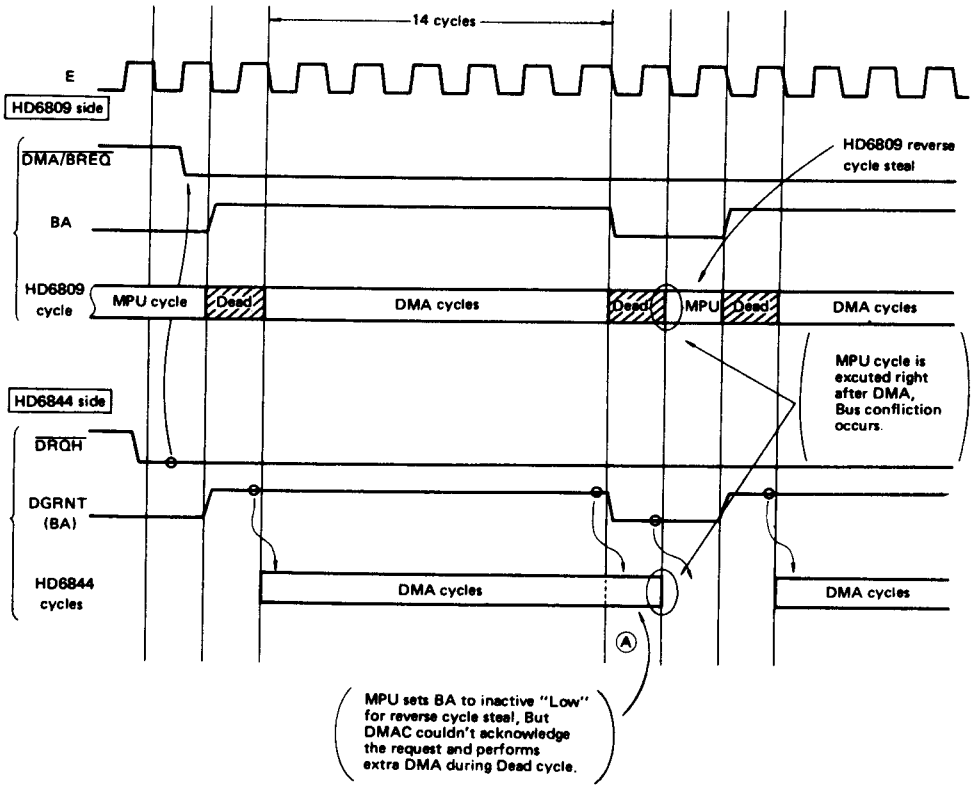


Figure 25 Comparison of HD6809, HD6844 DMA cycles

[3] Note for CLR Instruction

Cycle-by-cycle flow of CLR instruction (Direct, Extended, Indexed Addressing Mode) is shown below. In this sequence the content of the memory location specified by the operand is read before writing "00" into it. Note that status Flags, such as IRQ Flag, will be cleared by this extra data read operation when accessing the control/status register (sharing the same address between read and write) of peripheral devices.

Example: CLR (Extended)

$8000	CLR	$A000
$A000	FCB	$80

Cycle #	Address	Data	R/$\overline{W}$	Description
1	8000	7F	1	Opcode Fetch
2	8001	A0	1	Operand Address, High Byte
3	8002	00	1	Operand Address, Low Byte
4	FFFF	*	1	$\overline{VMA}$ Cycle
5	A000	80	1	Read the Data
6	FFFF	*	1	$\overline{VMA}$ Cycle
7	A000	00	0	Store Fixed "00" into Specified Location

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

[4] Note for MRDY

HD6809 require synchronization of the MRDY input with the 4f clock. The synchronization necessitates an external oscillator as shown in Figure 26. The negative transition of the MRDY signal, normally derived from the chip select decoding, must meet the t_{PCS} timing. MRDY's positive transition must occur with the rising edge of 4f.

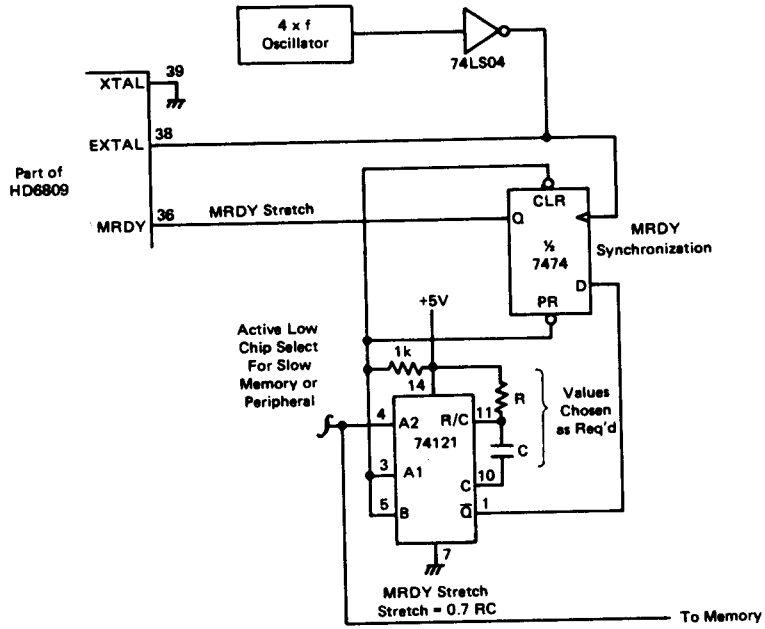


Figure 26 MRDY Synchronization

2