

388-609

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

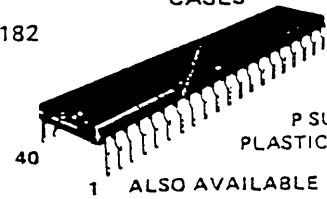
HMOS

(HIGH DENSITY N-CHANNEL, SILICON-GATE)

**8-BIT
MICROPROCESSING
UNIT**

CASES

CB-182



J SUFFIX
CERDIP PACKAGE

P SUFFIX
PLASTIC PACKAGE

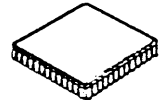
C SUFFIX
CERAMIC PACKAGE

CB-521



FN SUFFIX
PLCC 44

CB-708



E SUFFIX
LCCC 44

Hi-Rel versions available - See chapter 9

PIN ASSIGNMENT

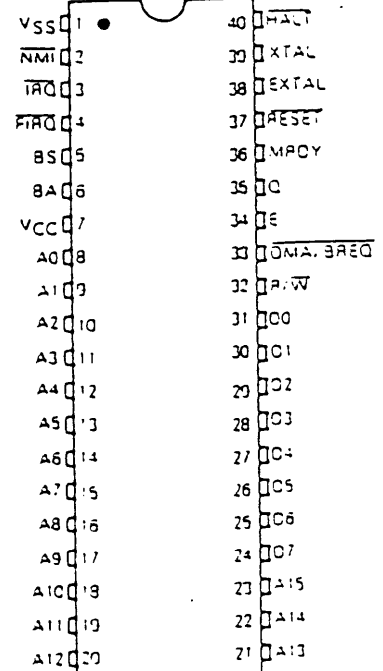


FIGURE 2 - EF6809 EXPANDED BLOCK DIAGRAM

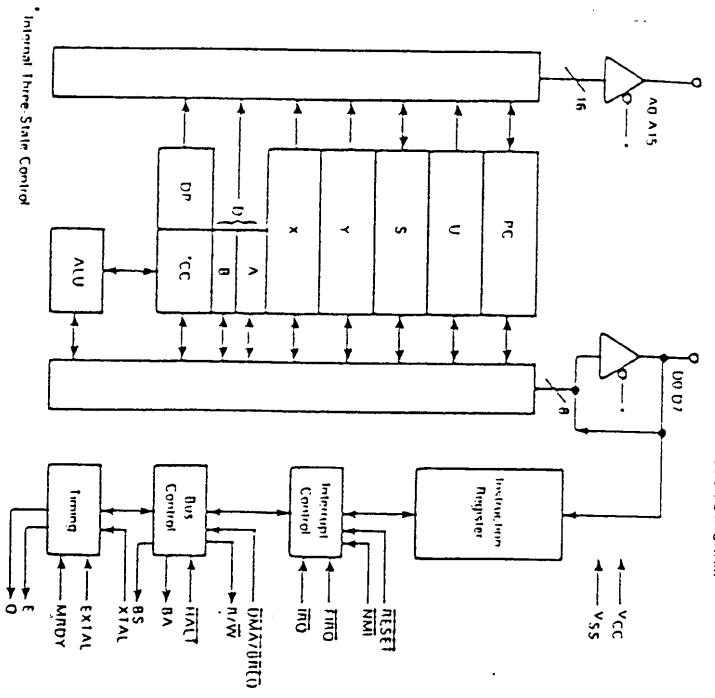
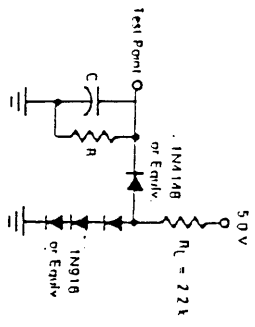


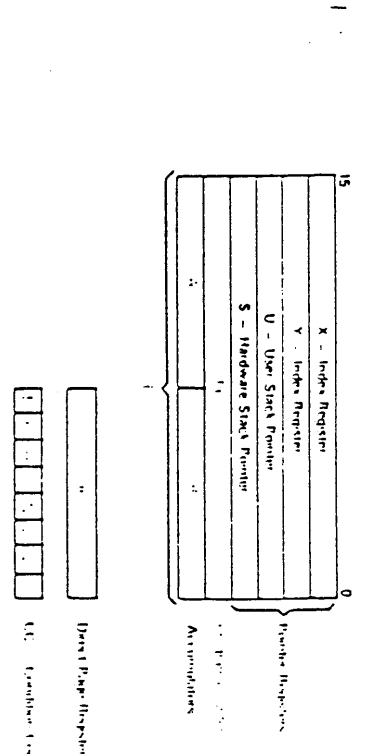
FIGURE 3 - BUS TIMING TEST LOAD



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

R = 11.7 kΩ for D0-D7
16.5 kΩ for A0-A15, E, O, R/W
24.1 kΩ for DA, BS

FIGURE 4 - PROGRAMMING MODEL OF THE MICROPROCESSING UNIT



INDEX REGISTERS (X, Y)

The index registers are used in indirect mode of addressing. The 16 bit address in this register is used in the calculation of effective addresses. The address may be used to point to data directly or may be modified by an optional constant or register offset. During some indirect 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 (SP) is used automatically by the processor during subroutine calls and returns. 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 (US) is controlled exclusively by the programmer. This allows arguments to be passed to and from subroutines with ease. Both stack pointers have the same enhanced stack-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.

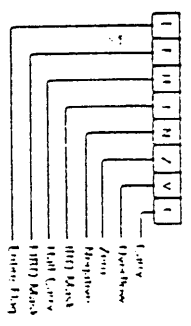
PROGRAM COUNTER

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.

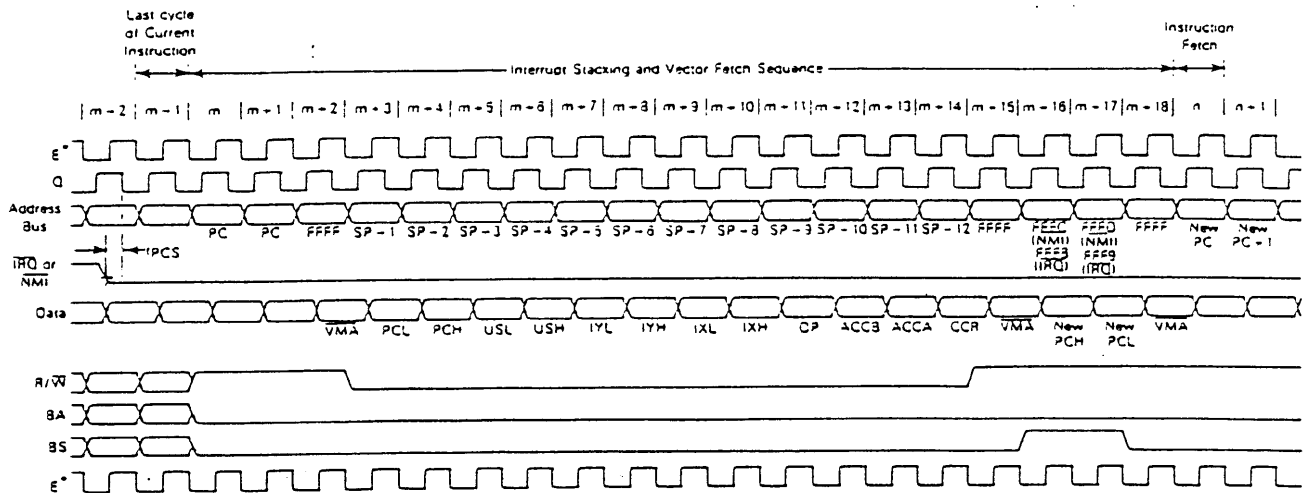
FIGURE 5 - CONDITION CODE REGISTER FORMAT



CONDITION CODE REGISTER DESCRIPTION

- BIT 15 (C)**
Bit 15 is the carry flag, and is usually the carry from the binary ALU. C is also used to represent a borrow from subtracted binary numbers (CM, M/G, S/D, S/D, C) and is the complement of the carry from the binary ALU.
- BIT 14 (Z)**
Bit 14 is the zero flag, and is set to one if the result of the previous operation was identically zero.
- BIT 13 (S)**
Bit 13 is the sign flag, and is set to one if the result of the previous operation was negative.
- BIT 12 (O)**
Bit 12 is the overflow flag, and is set to one if the result of the previous operation was outside the range of the previous operation.
- BIT 11 (P)**
Bit 11 is the parity flag, and is set to one if the number of ones in the result of the previous operation is odd.
- BIT 10 (A)**
Bit 10 is the auxiliary carry flag, and is set to one if there is a carry from bit 9 to bit 10 in the result of the previous operation.
- BIT 9 (H)**
Bit 9 is the half carry flag, and is set to one if there is a carry from bit 8 to bit 9 in the result of the previous operation.
- BIT 8 (I)**
Bit 8 is the interrupt mask flag, and is set to one if the processor is in interrupt mask mode.
- BIT 7 (E)**
Bit 7 is the interrupt enable flag, and is set to one if the processor is in interrupt enable mode.
- BIT 6 (I)**
Bit 6 is the interrupt mask flag, and is set to one if the processor is in interrupt mask mode.
- BIT 5 (E)**
Bit 5 is the interrupt enable flag, and is set to one if the processor is in interrupt enable mode.
- BIT 4 (I)**
Bit 4 is the interrupt mask flag, and is set to one if the processor is in interrupt mask mode.
- BIT 3 (E)**
Bit 3 is the interrupt enable flag, and is set to one if the processor is in interrupt enable mode.
- BIT 2 (I)**
Bit 2 is the interrupt mask flag, and is set to one if the processor is in interrupt mask mode.
- BIT 1 (E)**
Bit 1 is the interrupt enable flag, and is set to one if the processor is in interrupt enable mode.
- BIT 0 (I)**
Bit 0 is the interrupt mask flag, and is set to one if the processor is in interrupt mask mode.

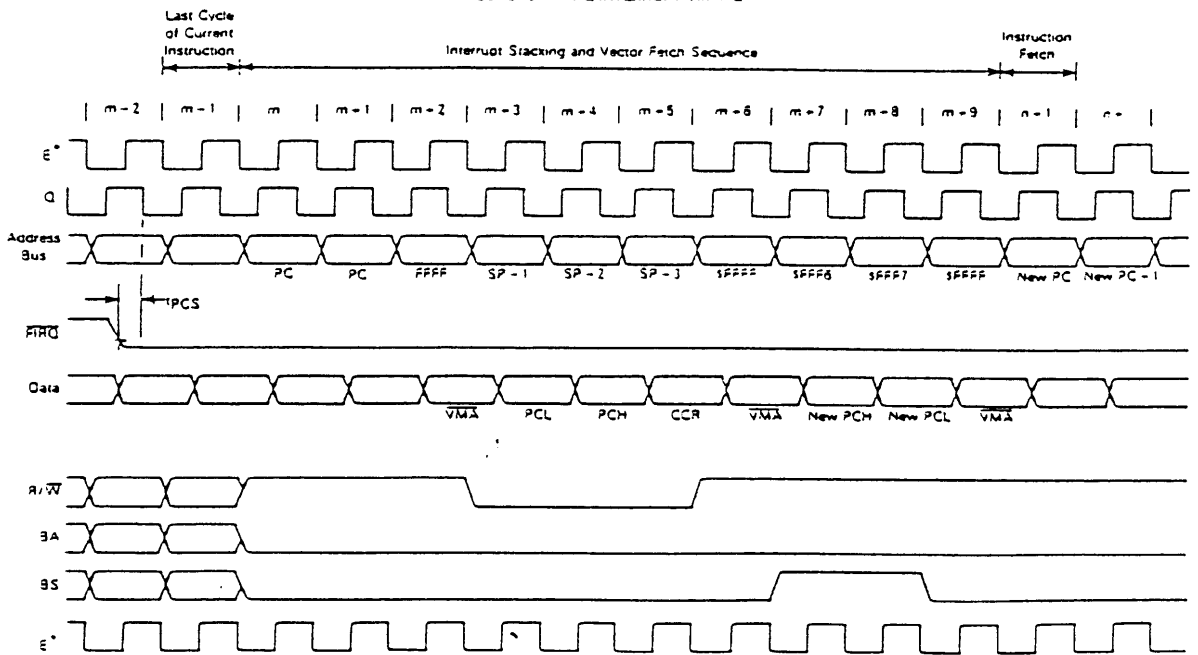
FIGURE 9 - $\overline{\text{IRQ}}$ AND $\overline{\text{NMI}}$ INTERRUPT TIMING



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

EF0009

FIGURE 10 - $\overline{\text{PIRQ}}$ INTERRUPT TIMING



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

EF0009

TIJMCEN CEMIC0N0N1JCTE11PC

11/18

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

E, O
 E is similar to the EF6800 bus timing signal phase 2. O is a quadrature clock signal which leads E. O has no relation to the EF6800. Addresses from the MPU will be valid with the leading edge of O. Data is latched on the falling edge of E timing for E and O is shown in Figure 11.

MNDY*
 This input control signal allows stretching of E and O to extend data access time. E and O operate normally while MNDY is high. When MNDY is low, E and O may be stretched in integral multiples of greater than 1 bus cycles, thus allowing interface to slow memories, as shown in Figure 12(a). During non valid memory access (DMA cycles), MNDY has no effect on stretching E and O. This inhibits slowing the processor during "don't care" bus accesses. MNDY may also be

used to stretch clocks for slow memory when bus control has been transferred to an external device through the use of **FACT** and **DMA/BHEO**.

The **DMA/BHEO** input provides a method of suspending execution and acquiring the MPU bus for parallel 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/BHEO** 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/BHEO** is inactive for two or more MPU cycles.

Typically, the DMA controller will request to use the bus by asserting **DMA/BHEO** 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. If slow memory accesses may be prevented during any dead cycles by developing a system **DMA/MA** signal which is LOW in any cycle when BA has changed.

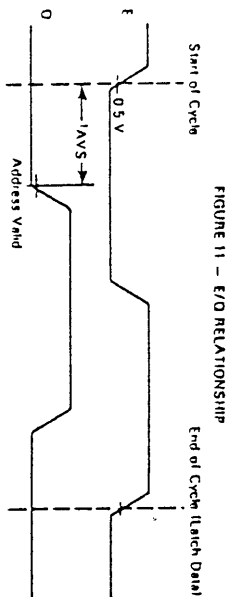


FIGURE 11 - E/O RELATIONSHIP

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 O to both the system and the MPU. When MNDY is pulled low, both the system clocks and the internal MPU clocks are stretched. Assertion of **DMA/BHEO** 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/BHEO** is released or after 16 bus cycles (4 DMA, two dead), whichever occurs first. While **DMA/BHEO** is asserted it is sometimes necessary to pull MNDY low to allow DMA for slow memory/peripherals. As both MNDY and **DMA/BHEO** control the internal MPU clocks, care must be exercised not to violate the maximum logic specification for MNDY or **DMA/BHEO**. Maximum logic specification for MNDY or **DMA/BHEO** is 10 μ s.

When BA goes low (either as a result of **DMA/BHEO** = 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 latches an instruction from memory and then executes the requested function.

This sequence begins after **RESET** and is repeated continually unless altered by a special instruction or hardware occurrence. Software instructions that alter normal MPU operation are SW1, SW2, SW3, CWA1, RT1, and SYNC. An interrupt, **FACT**, or **DMA/BHEO** can also alter the normal execution of instructions. Figure 15 illustrates the flow chart for the EF6809.

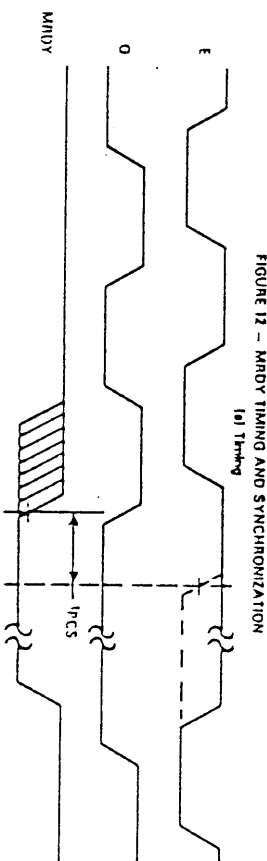


FIGURE 12 - MNDY TIMING AND SYNCHRONIZATION

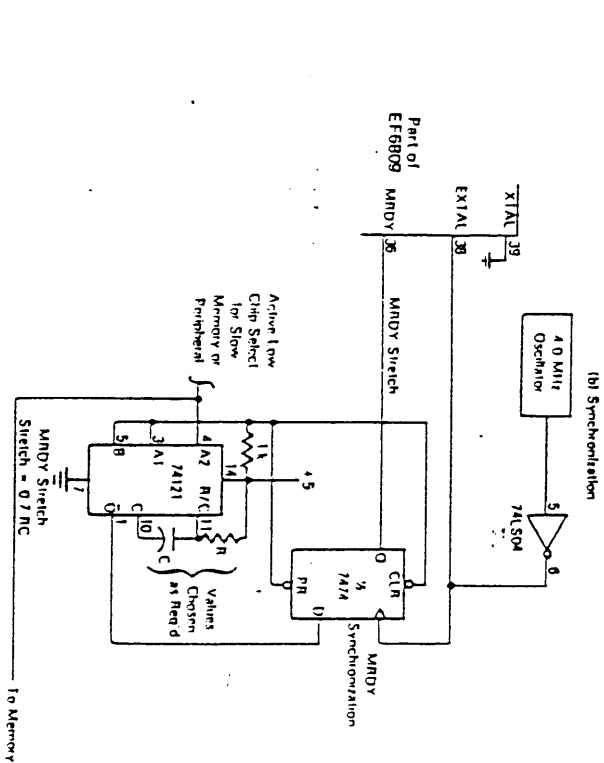


FIGURE 13 - TYPICAL DMA TIMING (< 14 CYCLES)

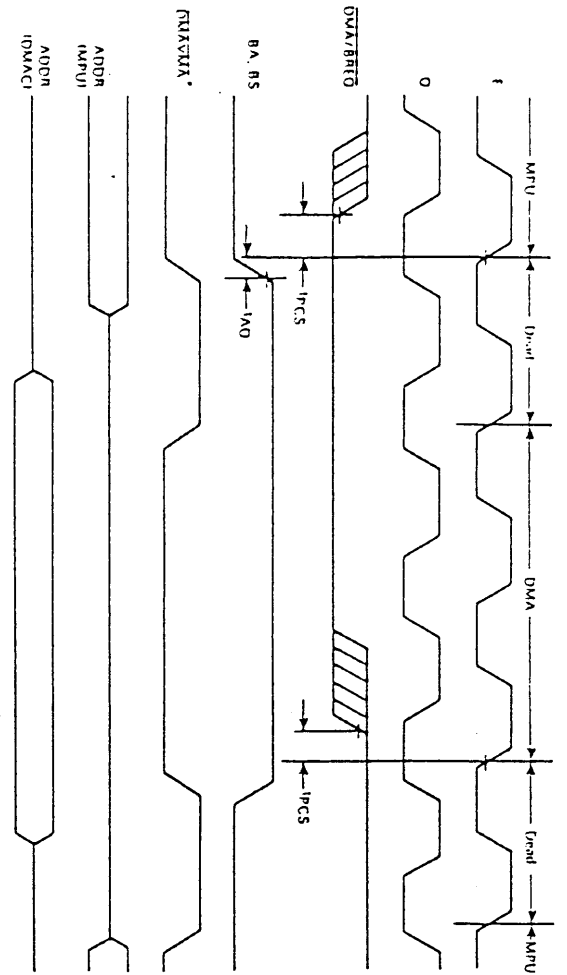
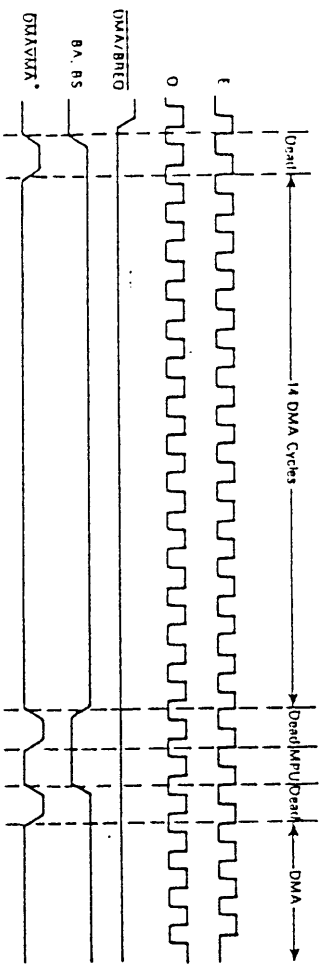


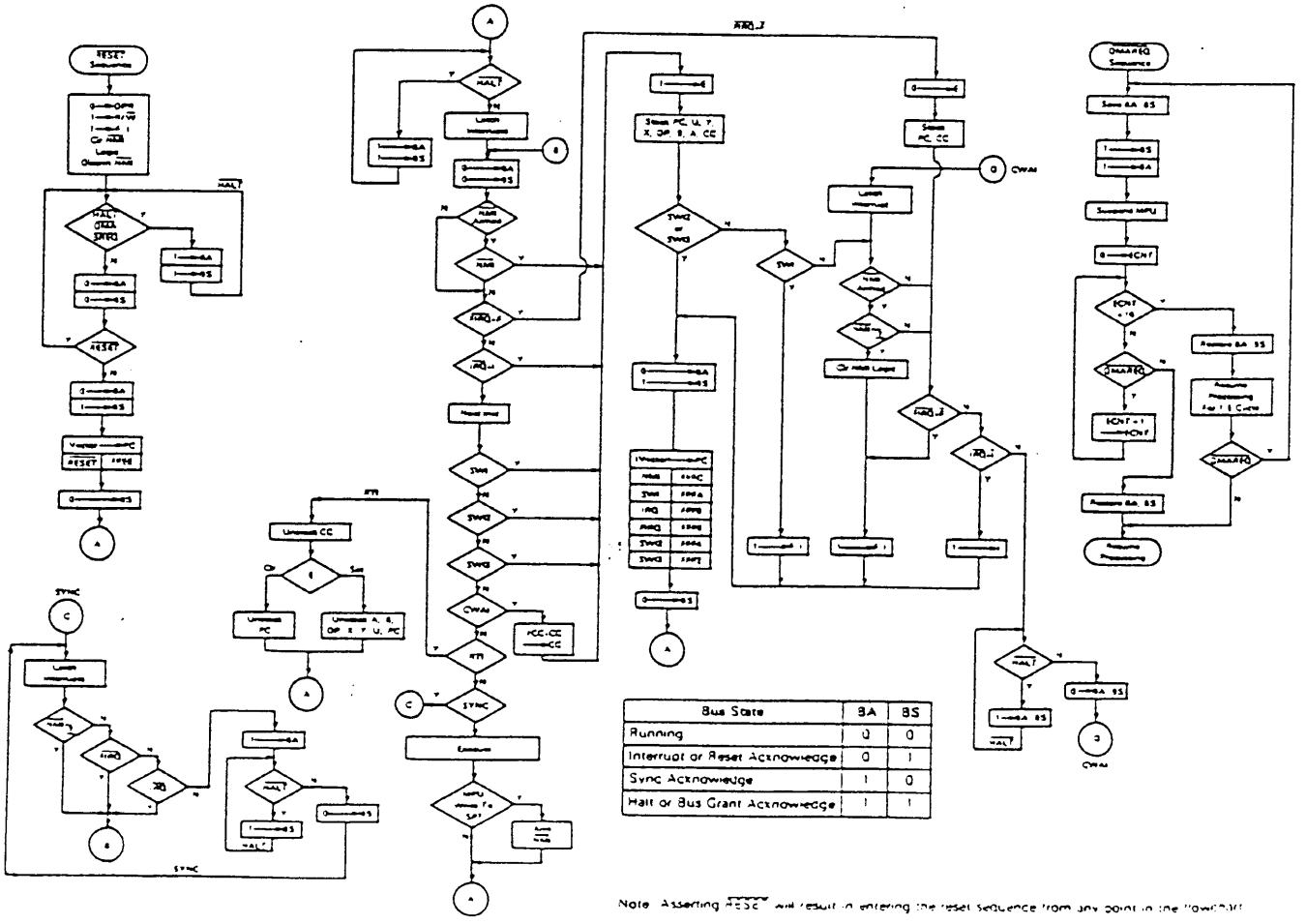
FIGURE 14 - AUTO-REFRESH DMA TIMING (> 14 CYCLES) (REVERSE CYCLE STEALING)



* MAXVMA is a signal which is developed externally. It 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



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

Note: Asserting \overline{RSTC} will result in entering the reset sequence from any point in the flowchart.

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 signifier 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.

```

LDA B,Y
LDX D,Y
LEAX R,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 stripping 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 labels, etc., are scanned from the high to low addresses. The size of the increment/decrement can be either one or two for tables of either B 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.

```

LDA .X+1
STD .Y+1
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+1
```

(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+1-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 1/4 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 = $1000
LDA $10,XI
    
```

EA is now \$F010

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 [0,S]
LDA [R,Y]
LDD [X+1]
    
```

RELATIVE ADDRESSING

The byte(s) following the branch operand is (are) treated as a signed offset which may be added to the program counter. If the branch condition is true, then the calculated address (PC + signed offset) is loaded into the program counter. Program execution continues at the new location as indicated by the PC. Short (one byte offset) and long (two bytes offset) relative addressing modes are available. All of memory can be reached in long relative addressing as an effective address is interpreted modulo 2¹⁶. Some examples of relative addressing are:

```

BEO CAT (short)
BGT DOG (short)
CAT [BEO] RAT (long)
DOG [BGT] RABBIT (long)
    
```

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

```

LDA CAT,PCR
LEAX TABLE,PCR
    
```

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

```

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

INSTRUCTION SET

The instruction set of the EF6B09 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 58, but because of the expanded architecture and additional addressing modes, the number of available opcodes with different addressing modes has risen from 192 to 1464.

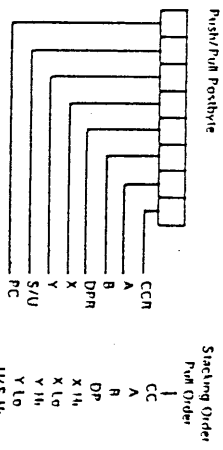
Some of the new instructions are described in detail below.

PSHU/PSHS

The push instructions have the capability of pushing onto either the hardware stack (SI) 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 operand determines which register or registers are to be pushed or pulled. The actual push/pull sequence is fixed, each bit defines a unique register to push or pull, as shown below.



TFR/EXG

Within the EF6B09, any register may be transferred to or exchanged with another of the size, i.e., 8 bit to 8 bit or 16 bit to 16 bit. Bits 4, 7 of possibly define the source register, while bits 0, 3 represent the destination register. These are denoted as follows:

TABLE 3 - LEA EXAMPLES

Instruction	Operation	Comment
LEAX 10,X	X ← 10 + X	Add 5-bit Constant 10 to X
LEAX \$00,X	X ← \$00 + X	Add 16-bit Constant \$00 to X
LEAY A,Y	Y ← A + Y	Add 8-bit A Accumulator to Y
LEAY D,Y	Y ← D + Y	Add 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 + S	Transfers AS Well As Adds

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 indices in a position independent manner. For example:

```

LEAX MSG1,PCU
LEAS MSG1,FCU
LEAS MSG1,PCU
    
```

MSG1 FCC MESSAGE:

This sample program prints "MESSAGE". By writing MSG1, PCU, 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+
    U, or S may be substituted for a and b)
    calculate the EA)
    1. b - temp (modify the EA)
    2. b+1 - b (modify b, postincrement)
    3. temp - a (load a)
    
```

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

MUL

Multiplicates the unsigned binary numbers in the A and B accumulators 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 B or 16-bit signed offset is added to the value of the program counter to be used as the relative address. This allows the program to branch anywhere in the 64K memory map. Position independent code can be easily generated through the use of relative branching. Both short (8-bit) and long (16-bit) branches are available.

SYNC

After encountering a sync instruction, the MPU enters a sync state, stops processing instructions, and waits for an interrupt. If the pending interrupt is non-maskable (NMI) or maskable (IRQ, FIRQ) with its mask bit (IF or IFL) clear, the processor will clear the sync state and perform the normal interrupt stacking and service routine. Since FIRQ and 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 (FIRQ, IRQ) with its mask bit (IF or IFL) 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.

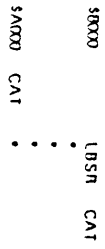
16 BIT OPERATION

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

CYCLE-BY-CYCLE OPERATION

The address bus cycle-by-cycle performance chart (Figure 18) illustrates the memory access sequence corresponding to each possible instruction and addressing mode in the EF6809. Each instruction begins with an opcode fetch. While that opcode is being internally decoded, the next program byte is always latched. (Most instructions will use the next byte, so this technique considerably speeds throughput.) Next, the operation of each opcode will follow the flowchart. VMA is an indication of FFF16 on the address bus, R/W = 1 and BS = 0. The following examples illustrate the use of the chart.

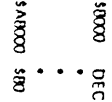
Example 1: LBSR (Branch Taken)
Before Execution SP = F000



CYCLE-BY-CYCLE FLOW

Cycle #	Address	Data	R/W	Description
1	8000	17	1	Opcode Fetch
2	8001	20	1	Offset High Byte
3	8002	00	1	Offset Low Byte
4	FFFF	*	1	VMA Cycle
5	FFFF	*	1	VMA Cycle
6	A000	*	1	Computed Branch Address
7	FFFF	*	1	VMA Cycle
8	FFFF	80	0	Stack High Order Byte of Return Address
9	FFFF	01	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	FFFF	*	1	VMA Cycle
5	A000	80	1	Read the Data
6	FFFF	*	1	VMA Cycle
7	A000	7F	0	Store the Decremental Data

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

INSTRUCTION SET TABLES

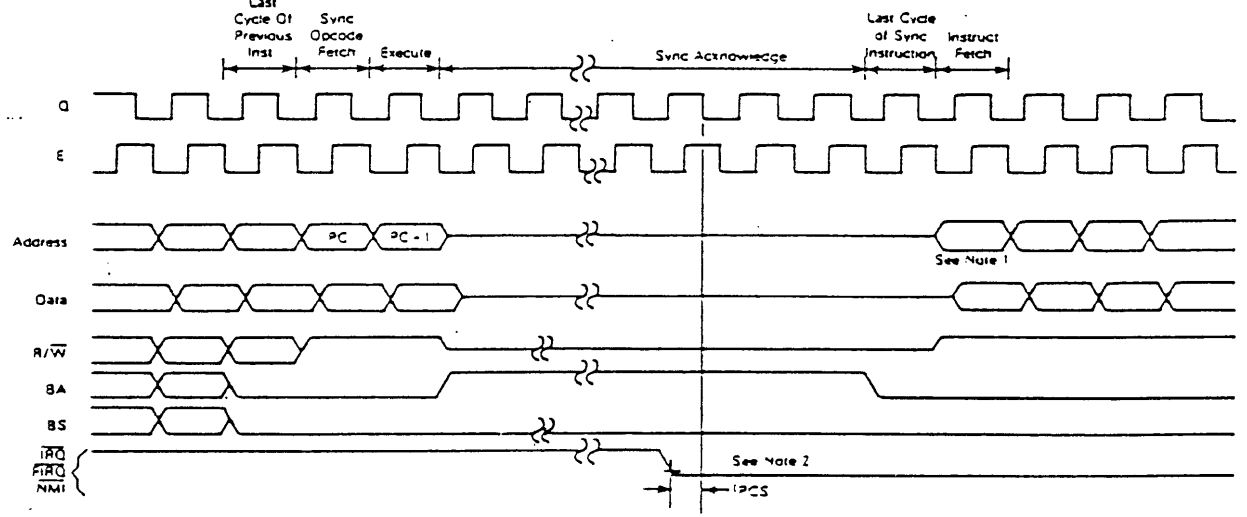
The instructions of the EF6809 have been broken down into five different categories. They are as follows:

- 8-bit operation (Table 4)
- 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

- 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 in Figures 9 and 10 (Interrupt Timing).
- If mask bits are clear, FIRQ and IRQ 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.
- Waveform measurements for all inputs and outputs are specified at logic high 2.0 V and logic low 0.3 V, unless otherwise specified.

FIGURE 18 - CYCLE-BY-CYCLE PERFORMANCE (Sheet 3 of 8)

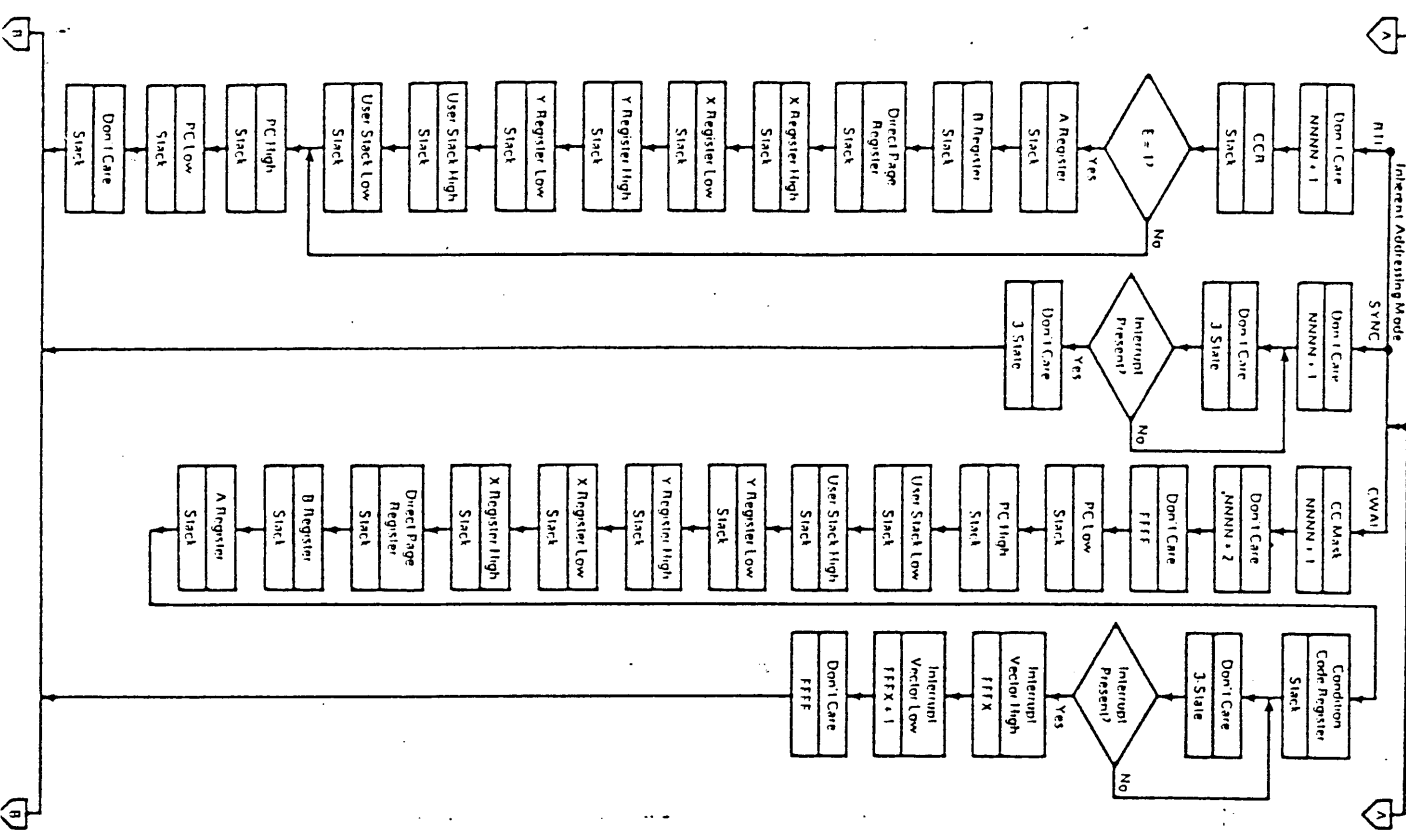


FIGURE 18 - CYCLE-BY-CYCLE PERFORMANCE (Sheet 4 of 8)

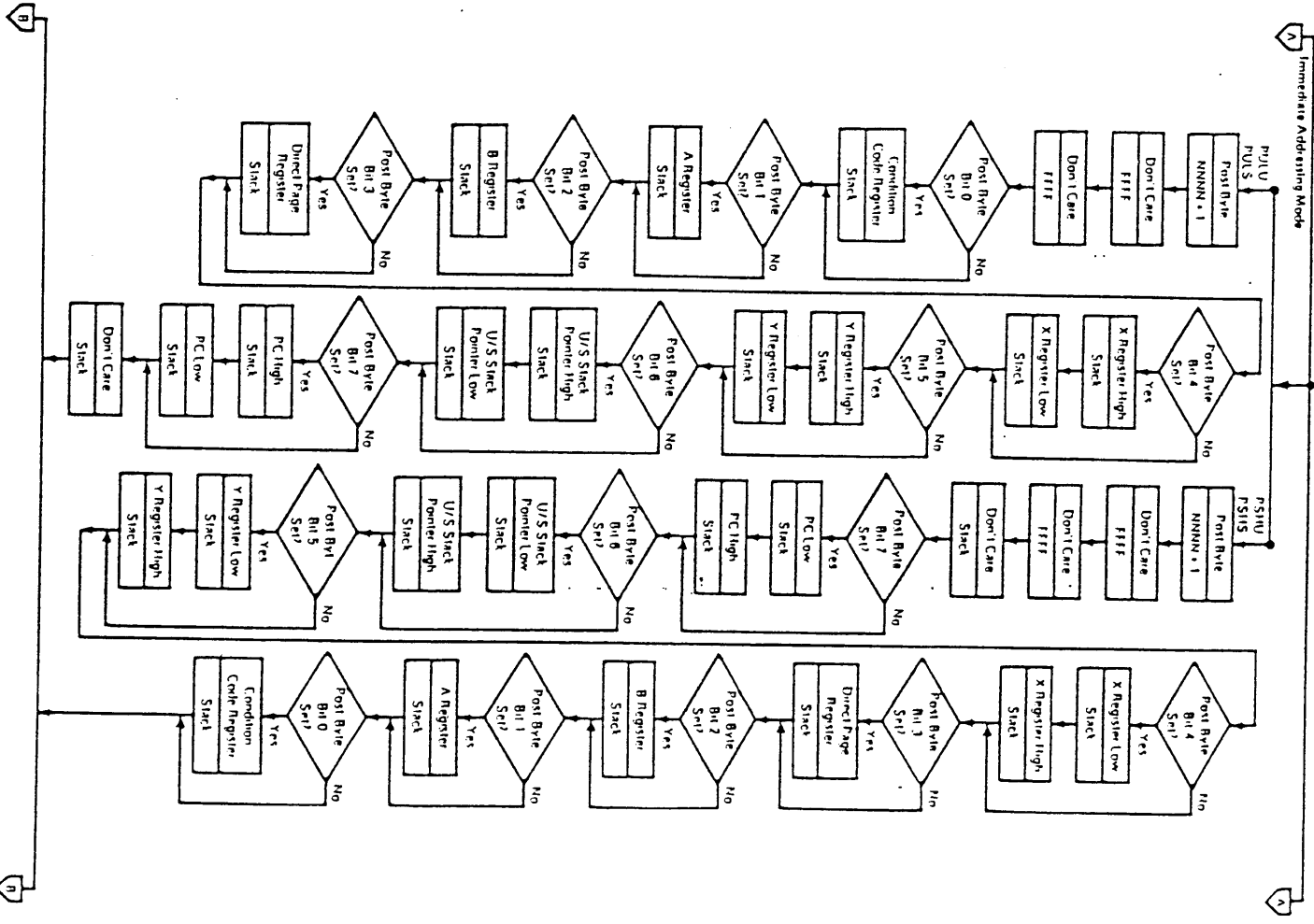


FIGURE 18 - CYCLE-BY-CYCLE PERFORMANCE (Sheet 5 of 8)

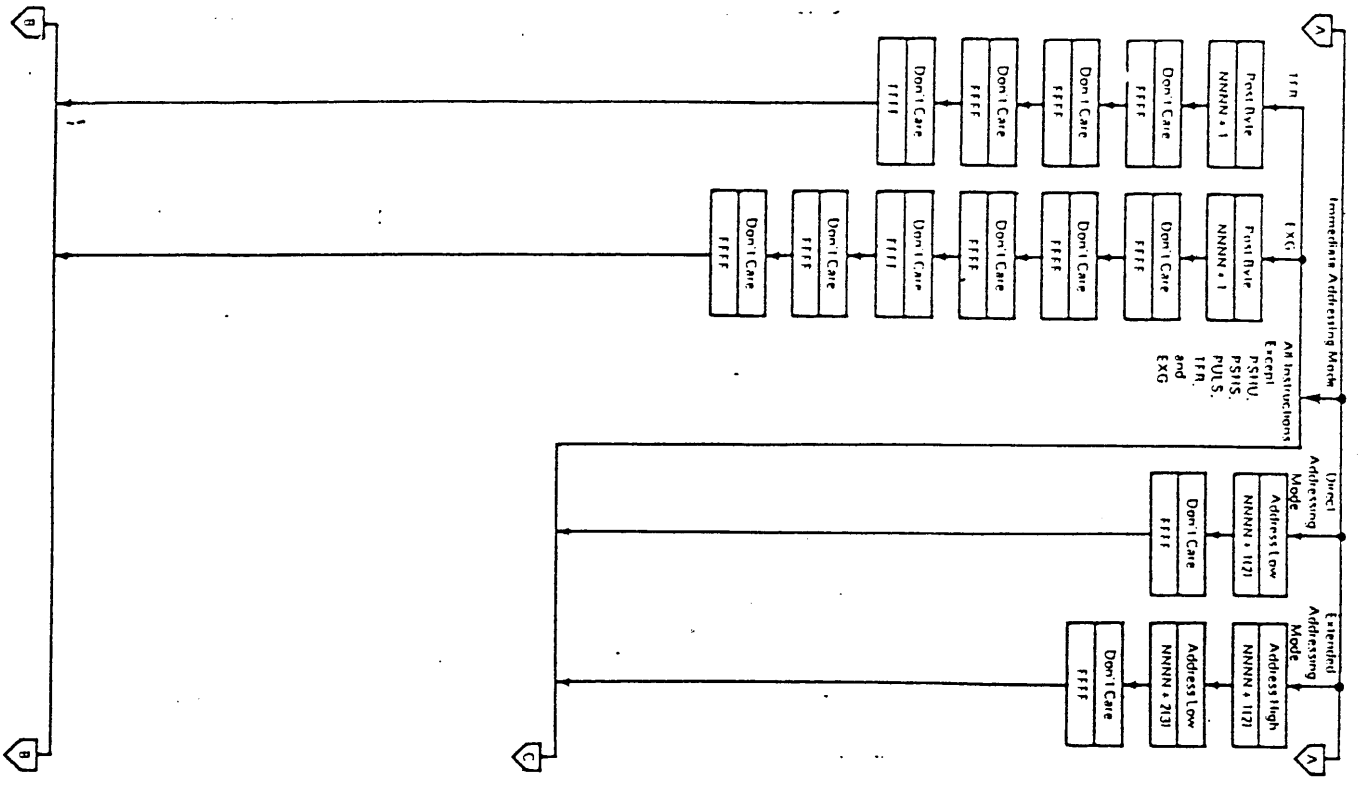
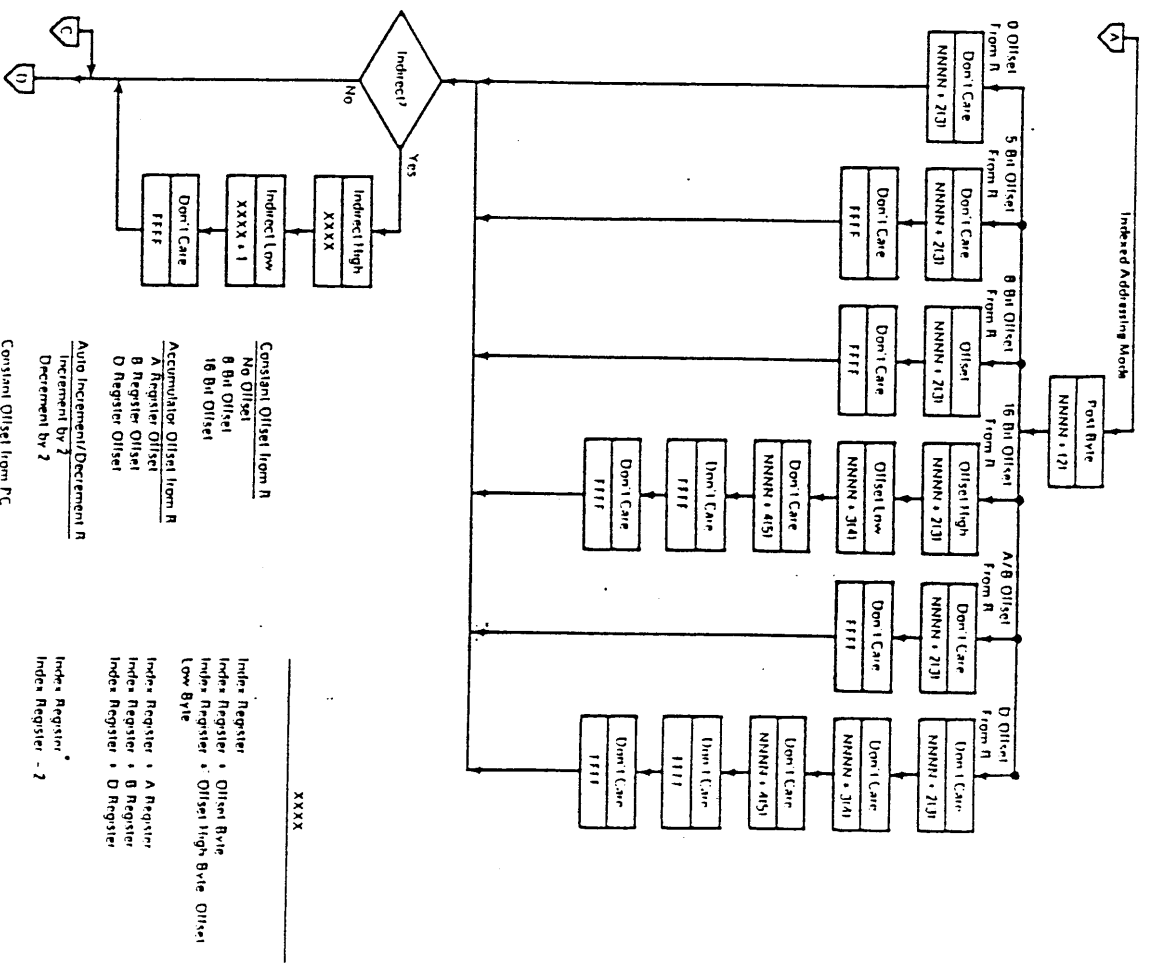


FIGURE 18 - CYCLE-BY-CYCLE PERFORMANCE (Sheet 6 of 8)



- Constant Offset from R
 - No 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 2
 - Decrement by 2
 - Constant Offset from PC
 - 8 Bit Offset
 - 16 bit Offset
 - Extended Indirect
 - 16 Bit Address
- * The index register is incremented following the indirect access

- Index Register:
 - Index Register + Offset High Byte
 - Index Register + Offset High Byte Offset Low Byte
 - Index Register + A Register
 - Index Register + B Register
 - Index Register + D Register
- Index Register:
 - Index Register + 2
- Index Register:
 - Program Counter + Offset High Byte Offset Low Byte
 - Address High Byte Address Low Byte

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

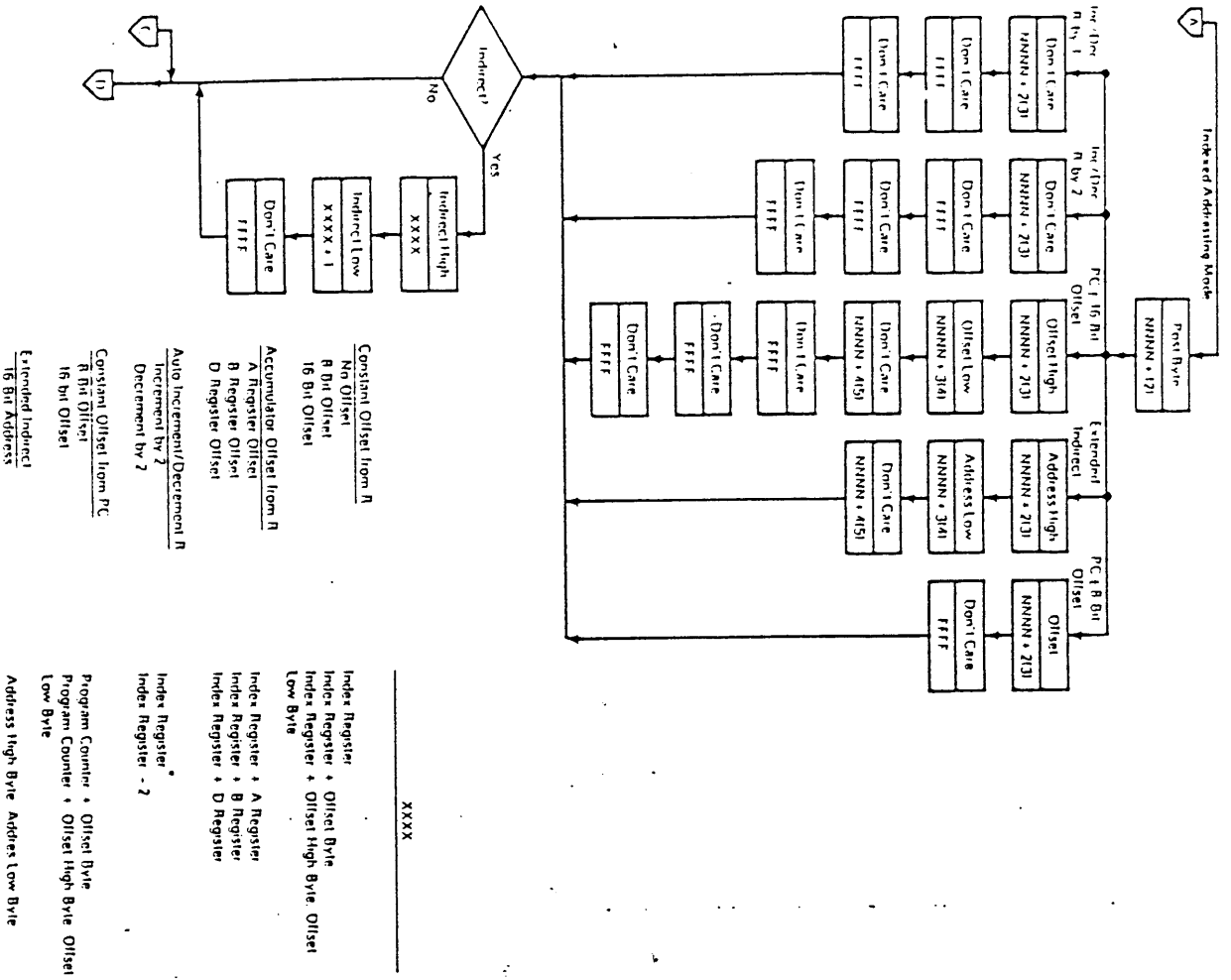


FIGURE 18 - CYCLE-BY-CYCLE PERFORMANCE (Sheet 8 of 9)

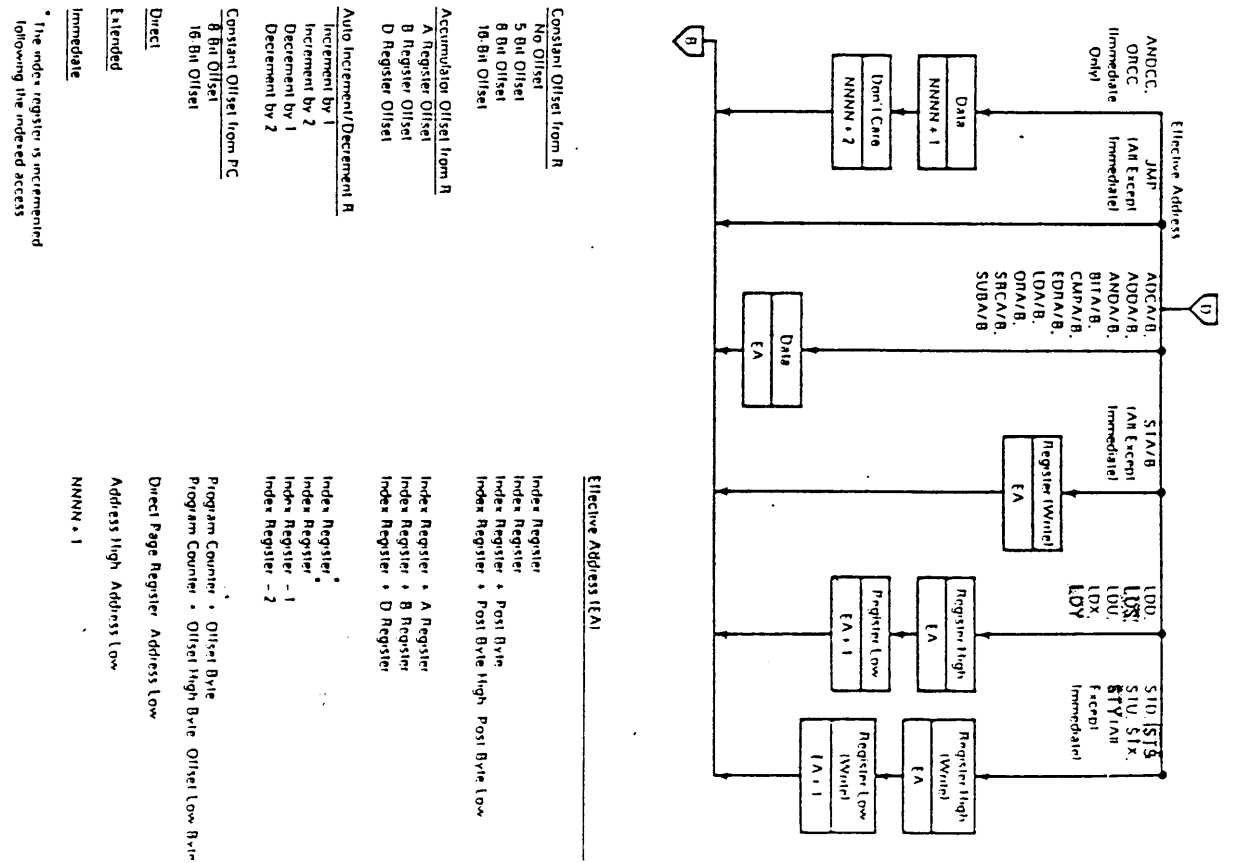
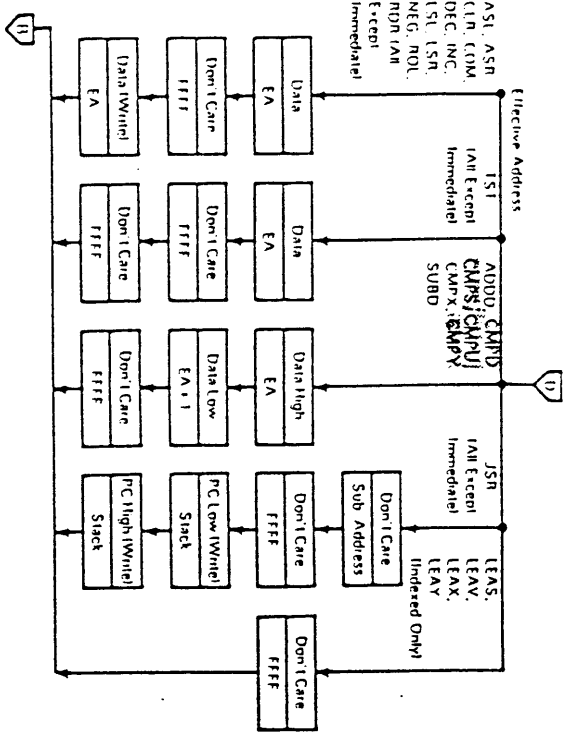


FIGURE 10 - CYCLE BY CYCLE PERFORMANCE (Sheet 9 of 9)



Effective Address (EAL)

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

* The index register is incremented following the indirect access

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 High Byte
 Program Counter + Offset High Byte Offset Low Byte

Direct Page Register Address Low

Address High Address Low

NNNN + 1

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
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, D, I)
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, LSR, LSRB	Logical shift right accumulator or memory location
MUL	Unsigned multiply (A × B = D)
NEG, NEGA, NEGB	Negate accumulator or memory (B memory with accumulator)
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
STA, STIA, STIB	Test accumulator or memory location
TFR R1, R2	Transfer R1 to R2 (R1, R2 = A, B, CC, D, I)

NOTE: A, B, CC, or D may be pushed to (popped from) stack with either PSHS, PSHU (PUSH, PULL) instructions.

TABLE 5 - 16 BIT ACCUMULATOR AND MEMORY INSTRUCTIONS

Mnemonic(s)	Operation
ADDD	Add memory to D accumulator
CMDD	Compare memory from D accumulator
EXG D, R	Exchange D with X, Y, S, U, or PC
LDD	Load D accumulator from memory
SLX	Sign extend B accumulator into A accumulator
STD	Store D accumulator to memory
SUBDD	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 (popped) to stack with either PSHS, PSHU (PUSH, PULL) instructions.

TABLE 6 - INDEX REGISTER/STACK POINTER INSTRUCTIONS

Instruction	Description
CMPS, CLD, STC	Compare memory from stack pointer
INC, RI, R2	Increment U, X, Y, or PC, add U, X, Y, or PC
LEAS, LEAV	Load effective address into stack pointer
LEAX, LEAX	Load effective address into register
LDS, LDU	Load stack pointer from memory
LOX, LOX	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, U, or PC onto hardware stack
PUSL	Push A, B, CC, DP, D, X, Y, U, or PC from hardware stack
STX, STX	Store index register to memory
TRN RI, R2	Transfer D, X, Y, S, U or PC to U, X, Y, S, U, or PC
ABX	Add B accumulator to X register

TABLE 7 - BRANCH INSTRUCTIONS

Instruction	Description
SIMPLE BRANCHES	
RIQ, LRIQ	Branch if equal
RIE, LRIE	Branch if not equal
RUI, LRUI	Branch if minus
RIL, LRIL	Branch if plus
RIE, LRIS	Branch if carry set
RIC, LRIC	Branch if carry clear
RVS, LRVS	Branch if overflow set
RVC, LRVC	Branch if overflow clear
SIGNED BRANCHES	
RGT, LRGT	Branch if greater (signed)
RVS, LRVS	Branch if signed 2's complement result
RGE, LRGE	Branch if greater than or equal (signed)
RIQ, LRIG	Branch if equal
RLE, LRLE	Branch if less than or equal (signed)
RLE, LRLE	Branch if less than or equal (signed)
RVC, LRVC	Branch if valid 2's complement result
RLI, LRLI	Branch if less than (signed)
UNSIGNED BRANCHES	
RII, LRIL	Branch if higher (unsigned)
RCC, LRCC	Branch if higher or same (unsigned)
RIS, LRIS	Branch if higher or same (unsigned)
RIQ, LRIG	Branch if equal
RNE, LRNE	Branch if not equal
RIS, LRIS	Branch if lower or same (unsigned)
RCS, LRCS	Branch if lower (unsigned)
RIQ, LRIG	Branch if lower (unsigned)
OTHER BRANCHES	
BSN, LBSN	Branch to subroutine
BNA, LBNB	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
JSN	Jump to subroutine
RII	Return from interrupt
RII	Return from subroutine
RII	Return from subroutine
SWI, SWI2, SWI3	Software interrupt (hardware redirect)
SZVC	Synchronize with interrupt line

TABLE 9 - HEXADECIMAL VALUES OF MACHINE CODES

OP	Mnem	Mode	OP	Mnem	Mode	OP	Mnem	Mode
00	NEG	Direct	30	LEAX	Inherent	70	NEG	Inherent
01	•	•	31	LEAV	Inherent	71	•	•
02	COM	Inherent	32	LEAS	Inherent	72	COM	Inherent
03	LEAS	Inherent	33	LEAV	Inherent	73	COM	Inherent
04	LSR	Inherent	34	PSHS	Inherent	74	LSN	Inherent
05	PUSL	Inherent	35	PUSL	Inherent	75	•	•
06	ROR	Inherent	36	PSHU	Inherent	76	ROR	Inherent
07	ASR	Inherent	37	PULU	Inherent	77	ROR	Inherent
08	ASL, LSL	Inherent	38	ASR	Inherent	78	ASL, LSL	Inherent
09	ROR	Inherent	39	RIS	Inherent	79	ASR	Inherent
0A	DEC	Inherent	3A	ABX	Inherent	7A	ASL, LSL	Inherent
0B	•	•	3B	R11	Inherent	7B	ROR	Inherent
0C	INC	Inherent	3C	CWAI	Inherent	7C	DEC	Inherent
0D	TSI	Inherent	3D	MUL	Inherent	7D	•	•
0E	JMP	Inherent	3E	SWI	Inherent	7E	JMP	Inherent
0F	CLR	Inherent	3F	•	•	7F	CLR	Inherent
10	Page 2	•	40	NEGA	Inherent	80	•	•
11	Page 3	•	41	•	•	81	•	•
12	NOP	Inherent	42	CONVA	Inherent	82	•	•
13	SYNC	Inherent	43	CONVA	Inherent	83	•	•
14	•	•	44	LSNA	Inherent	84	•	•
15	•	•	45	•	•	85	•	•
16	LORNA	Relative	46	RORNA	Inherent	86	•	•
17	LRSN	Relative	47	ASNA	Inherent	87	•	•
18	•	•	48	ASNA, LSVA	Inherent	88	•	•
19	DAA	Inherent	49	ROLA	Inherent	89	•	•
1A	ORCC	Inherent	4A	DECA	Inherent	8A	•	•
1B	•	•	4B	•	•	8B	•	•
1C	ANDCC	Inherent	4C	INCA	Inherent	8C	•	•
1D	SEX	Inherent	4D	STSA	Inherent	8D	•	•
1E	EXG	Inherent	4E	•	•	8E	•	•
1F	TRN	Inherent	4F	CLRA	Inherent	8F	•	•
20	BNA	Relative	50	NEGB	Inherent	90	•	•
21	BRN	Relative	51	•	•	91	•	•
22	BHI	Relative	52	•	•	92	•	•
23	BLS	Relative	53	•	•	93	•	•
24	BHS, DCC	Relative	54	•	•	94	•	•
25	BLO, BCS	Relative	55	•	•	95	•	•
26	BNE	Relative	56	•	•	96	•	•
27	BEO	Relative	57	•	•	97	•	•
28	BVC	Relative	58	•	•	98	•	•
29	BVS	Relative	59	•	•	99	•	•
2A	BPL	Relative	5A	•	•	9A	•	•
2B	DMI	Relative	5B	•	•	9B	•	•
2C	BGE	Relative	5C	•	•	9C	•	•
2D	BLT	Relative	5D	•	•	9D	•	•
2E	BGT	Relative	5E	•	•	9E	•	•
2F	BLE	Relative	5F	•	•	9F	•	•

LEGEND:

- Number of MPU cycles lines possible push pull or ordered mode cycles
- # Number of program bytes
- Denotes unused opcode

FIGURE 19 - PROGRAMMING AID (CONTINUED)

Instruction	Forms	Addressing Modes				Description	Interrupts				5	3	2	1	0
		Immediate	Direct	Indexed	Extended		Op	Op	Op	Op					
LSL	LSLA LSLR LSL														
LSR	LSRA LSRR LSR														
MAIL															
NEG	NEGA NEGB NEG														
NOP															
ORA	ORA ORR ORLCC														
PSH	PSHA PSHU PSL														
PUL	PULA PULU PUL														
ROL	ROLA ROLB ROL														
RRM	RRMA RRRB RRR														
RLL															
SBC	SBCA SBCB SBC														
SEK															
ST	STA STB STD STF														
STU	STUA STUB STU														
SUB	SUBA SUBB SUBC SUBD														
SWI	SWI ₀ SWI ₁ SWI ₂														
SYNC															
TFR															
TST	TSTA TSTB TST														

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

FIGURE 19 - PROGRAMMING AID (CONTINUED)

Branch Instructions

Instruction	Forms	Addressing Mode		Description	H	M	Z	V	C
		Op	Op						
BCC	BCC	74	3	Branch C=0	*	*	*	*	*
BCC	LBCC	10	5/6	Long Branch C=0	*	*	*	*	*
BCC	LBCC	24	4	Long Branch C=0	*	*	*	*	*
BCC	LBCC	10	5/6	Long Branch C=1	*	*	*	*	*
BCC	LBCC	24	4	Long Branch C=1	*	*	*	*	*
BEO	BEO	77	3	Branch Z=1	*	*	*	*	*
BEO	LBEO	10	5/6	Long Branch Z=1	*	*	*	*	*
BEO	LBEO	24	4	Long Branch Z=1	*	*	*	*	*
BEO	LBEO	10	5/6	Long Branch Z=0	*	*	*	*	*
BEO	LBEO	24	4	Long Branch Z=0	*	*	*	*	*
BGE	BGE	77	3	Branch > Zero	*	*	*	*	*
BGE	LBGE	10	5/6	Long Branch > Zero	*	*	*	*	*
BGE	LBGE	24	4	Long Branch > Zero	*	*	*	*	*
BGE	LBGE	10	5/6	Long Branch < Zero	*	*	*	*	*
BGE	LBGE	24	4	Long Branch < Zero	*	*	*	*	*
BGT	BGT	77	3	Branch Higher	*	*	*	*	*
BGT	LBGT	10	5/6	Long Branch Higher	*	*	*	*	*
BGT	LBGT	24	4	Long Branch Higher	*	*	*	*	*
BGT	LBGT	10	5/6	Long Branch Higher	*	*	*	*	*
BGT	LBGT	24	4	Long Branch Higher	*	*	*	*	*
BHI	BHI	77	3	Branch Higher or Same	*	*	*	*	*
BHI	LBHI	10	5/6	Long Branch Higher or Same	*	*	*	*	*
BHI	LBHI	24	4	Long Branch Higher or Same	*	*	*	*	*
BHI	LBHI	10	5/6	Long Branch Higher or Same	*	*	*	*	*
BHI	LBHI	24	4	Long Branch Higher or Same	*	*	*	*	*
BHS	BHS	74	3	Branch Higher or Same	*	*	*	*	*
BHS	LBHS	10	5/6	Long Branch Higher or Same	*	*	*	*	*
BHS	LBHS	24	4	Long Branch Higher or Same	*	*	*	*	*
BHS	LBHS	10	5/6	Long Branch Higher or Same	*	*	*	*	*
BHS	LBHS	24	4	Long Branch Higher or Same	*	*	*	*	*
BLE	BLE	71	3	Branch < Zero	*	*	*	*	*
BLE	LBLE	10	5/6	Long Branch < Zero	*	*	*	*	*
BLE	LBLE	24	4	Long Branch < Zero	*	*	*	*	*
BLE	LBLE	10	5/6	Long Branch < Zero	*	*	*	*	*
BLE	LBLE	24	4	Long Branch < Zero	*	*	*	*	*
BLO	BLO	75	3	Branch Lower	*	*	*	*	*
BLO	LBLO	10	5/6	Long Branch Lower	*	*	*	*	*
BLO	LBLO	24	4	Long Branch Lower	*	*	*	*	*
BLO	LBLO	10	5/6	Long Branch Lower	*	*	*	*	*
BLO	LBLO	24	4	Long Branch Lower	*	*	*	*	*

SIMPLE BRANCHES

Instruction	Test	True	OP	False	OP
BRA	r > m	20	3	2	
LBRA	r > m	18	5	3	
BRN	r = m	21	3	2	
LB RN	r = m	1021	5	4	
BSR	r < m	80	7	2	
LSBR	r < m	17	9	3	

SIMPLE CONDITIONAL BRANCHES (Notes 1-4)

Instruction	Test	True	OP	False	OP
N=1		BMI	28	BPL	2A
Z=1		BEO	27	BNE	26
V=1		BVS	29	BVC	28
C=1		BCS	25	BCC	24

SIGNED CONDITIONAL BRANCHES (Notes 1-4)

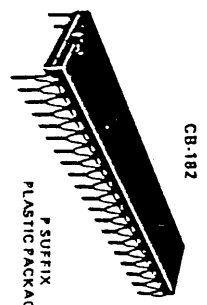
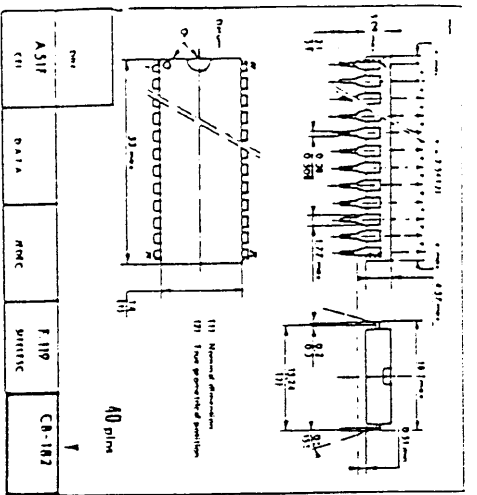
Instruction	Test	True	OP	False	OP
r > m	BGT	2E	2C	BLE	2F
r = m	BGE	2E	2D	BLT	2D
r = m	BEO	27	26	BNE	26
r < m	BLE	2F	2E	BGT	2E
r < m	BLT	2D	2C	BGE	2C

UNSIGNED CONDITIONAL BRANCHES (Notes 1-4)

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

- NOTES
- All conditional branches have both short and long variations.
 - All short branches are two bytes and require three cycles.
 - All conditional long branches are formed by prefixing the short branch opcode with \$10 and using a 16 bit destination offset.
 - All conditional long branches require four bytes and six cycles if the branch is taken or five cycles if the branch is not taken.

PHYSICAL DIMENSIONS



ALSO AVAILABLE
 J SUFFIX CERAMIC PACKAGE
 P SUFFIX PLASTIC PACKAGE

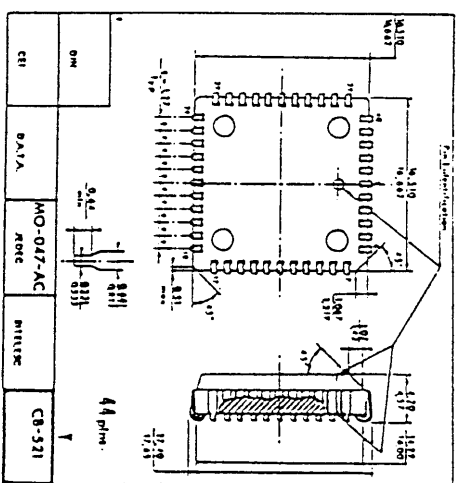
ORDERING INFORMATION

DEVICE	Device				Screening level							
	C	J	P	E	FN	L	V	M	BID	D	Q/B	B/B
EF6809 (1.0 MHz)	•	•	•	•	•	•	•	•	•	•	•	•
EF6809 (1.8 MHz)	•	•	•	•	•	•	•	•	•	•	•	•
EF6809 (3.0 MHz)	•	•	•	•	•	•	•	•	•	•	•	•

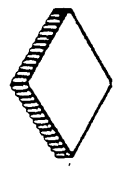
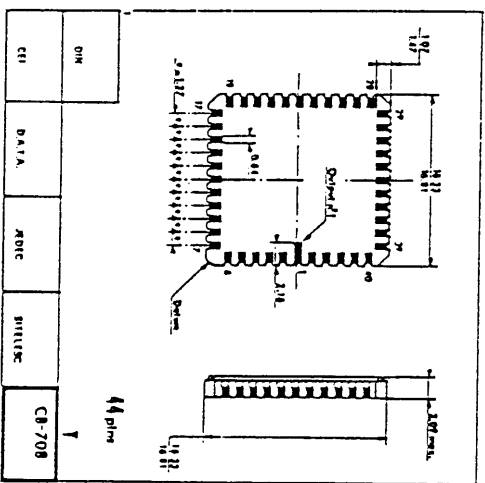
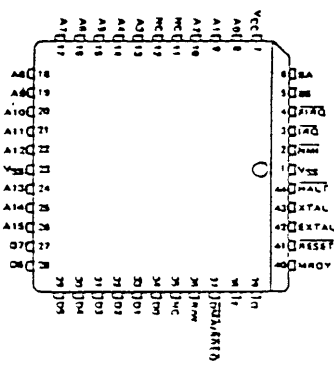
Examples: EF6809C, EF6809CV, EF6809CMA

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: Inv and suffix, D: N/C 96083 level D, Q/B: N/C 96083 level G, B/B: N/C 96083 level B and MIL-STD-883C level B.

PHYSICAL DIMENSIONS



FM SUFFIX PLCC 44



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.