HD63B09EP Technical Reference Guide

By Chet Simpson
Additions by Alan DeKok

INDEX

Introduction

The HD63B09EP microprocessor by Hitachi, is a MC68B09E compatible chip containing additional registers and an additional instruction set. The 6309 was thought to be a flakey chip though, because it would sometimes crash or change the values of registers when it encountered an addressing mode or opcode invalid to the 6809. This was later found to be an extended instruction set and a feature that would trap some programming errors and jump to a specified location in memory.

Hitachi licensed the rights of the 6809 instruction set from Motorola to make a 6809 compatible chip. When they finished the design, they found there was a lot of unused space in the chip. With this in mind they added extra registers and expanded on the instruction set, but due to the licensing agreement with Motorola, they were unable to release the information about the extra features.

Not only did the chip have an expanded instruction set, but it also had a native mode that would run many of the instructions in fewer clock cycles and a mode select for the FIRQ (Fast Interrupt ReQuest) that would enable it to opperate the same as the IRQ.

In fact, all new instructions will execute in emulation mode, which was originally seen when 'illegal' 6809 instructions produced odd results when run on a computer with a 6309 installed.

The additional instruction set was first written about in the April 1988 issue of "Oh!FM", a Japanese magazine, and was later brought to the attention of the 6809 community by Hirotsugu Kakagawa. He followed up a series of '6809-6309 differences' messages on comp.sys.m6809 by posting a detailed explanation of the new features and instructions of the 6309. This opened a whole new door to those who wished to use the 6309 in place of the 6809.

The information in this reference is of technical nature and makes no attempt to teach assembly language programming. It is ONLY a technical reference guide for those who already know assembly and wish to use these features in their programs. Although all of the opcodes for the 6309/6809 chip are listed in the appendix, only the additional features supplied by the 6309 will be discussed in detail.

Summary of Features

More registers:
- one 8/16 bit 'zero' register
- Two 8bit accumulators.
- One 16bit concatenated register
- One 16bit value register.
- One 8bit mode/error register.
- One 32bit concatenated register

Two modes: MC68B09E emulation mode and HD63B09EP native mode.

Reduced execution cycles when running in native mode.

Many additional instructions.

Error trapping of illegal instructions and zero divisions.

Description of Additional Registers

The 6309 has 7 additional registers. Only 4 of these are actual registers. 2 are combinations of registers, and the last is a constant-value register. These registers are:

```
ACCE  - 8 bit accumulator.
ACCF  - 8 bit accumulator.
W     - 16 bit concatenated register (ACCE and ACCF combined).
Q     - 32 bit concatenated register (ACCA, ACCB ,ACCE and ACCF
                                      combined).
V     - 16 bit register (which can only be accessed with the
                         inter-register instructions).
0     - zero register
MD    - 8 bit mode/error register.
```

ACCE and ACCF both work in much the same manner as the ACCA and ACCB accumulators. This makes for easier programming in math and data oriented routines.

The W register is like the D register in the 6809. It is a concatenated register containing the values of ACCE and ACCF as one 16 bit value. ACCE is contained in the high 8 bits and ACCF is contained in the low 8 bits.

The Q register is a 32 bit concatenated register. This register is composed of the concatenation of D and W, which in turn are composed of the registers ACCA, ACCB, ACCE and ACCF respectively. This register is used mostly with the additional math instructions supplied with the 6309 which will be discussed later.

The V register is a 16 bit register that can only be accessed with inter-register instructions such a TFR and EXG. The contents of this register will not change if the CPU is reset, allowing this register to be used as a constant value for the program.

The 0 register is always zero, independant of writes to it. It enables a zero value to be used in inter-register operations without accessing memory, or changing the value of another register.

The MD register is a mode and error register and works much in the same way as the CC register. The bit definitions are as follows:

```
Write bits
  Bit 0   -  Execution mode of the 6309.
             If clear ( 0 ), the cpu is in 6809 emulation mode.
             If set ( 1 ), the cpu is in 6309 native mode.
  Bit 1   -  FIRQ mode
             If clear ( 0 ), the FIRQ will occur normally.
             If set ( 1 ) , the FIRQ will operate the same as the
                            IRQ

  Bits 2 to 5 are unused

Read bits - One of these bits is set when the 6309 traps an error
  Bit 6   -  This bit is set ( 1 ) if an illegal instruction is
             encountered
  Bit 7   -  This bit is set ( 1 ) if a zero division occurs.
```

Modes of Operation

The 6309 has two modes of operation; 6809 Emulation mode in which the chip acts and executes instructions the same as the 6809, and 6309 Native mode which stores an extra two bytes on the stack when an interrupt (IRQ) occurs, and executes instructions in fewer clock cycles.

When in native mode, the W register (2 additional bytes) is stored (PSHS) on the system stack when an interrupt occurs, it is stored on the stack right after the D (general data) register. Since ALL register values are stored on the system stack when an IRQ (NOT FIRQ - See FIRQ modes for more information) occurs, great care should be taken when writing or patching those routines to run in native mode.

Pull <- CC,A,B,E*,F*,DP,Xhi,Xlo,Yhi,Ylo,Uhi,Ulo,PChi,PClo <- Push

* indicates the additional registers stored on the system stack

When in native mode those interrupt routines which modify the return address by modifying the 10th and 11th byte offsets from the stack (STX 10,S or STY 10,S etc.) will have to be changed to modify the 12th and 13th byte offsets from the stack (STX 12,S or STY 12,S etc.). If those routines are not patched to run in native mode they will either get stuck in a continuous loop or will crash the system due to the fact that they are not returning to the correct address. This poses a MAJOR problem for OS-9 Level II since its main interrupt handling routine relies highly on the changing of the return (PC) address on the stack. Disk read/write and formatting routines also rely heavily on changing the return address during an NMI (Non-Maskable Interrupt).

To patch those routines which do modify the return address, the program or routine must be disassembled or modified with a disk sector editing program. Look for instructions such as STX 10,S or STY 10,S that has an RTI (Return from Interrupt) instruction within the next few lines of the routine. The line containing STX 10,S or STY 10,S should be changed to STX 12,S or STY 12,S respectively.

Remember, after those routines are patched, those programs using them will NOT work in emulation mode and will require native mode to be enabled upon startup.

Native Mode and Timing Loops

There is at least one more problem that needs to be addressed. Those are routines which are dependant on timing loops for accuarate operation. Since the 6309 executes instructions faster when in native mode, those routines that use timing loops would be effected. Since this can pose a problem and can create erratic operation, the delay value or routine will need to be changed for the routine to operate correctly.

Those routines are usually serial-printer routines, cassette read/write timimg routines, software clocks and some disk read/write routines.

Modes of the Fast Interrupt Request (FIRQ)

The designers of the 6309 decided that with the additional instructions and native mode of operation, the FIRQ may be used more than it usually is. With this in mind they decided to allow you to make the FIRQ run the same as the IRQ and store (PSHS) all the current values of the registers on the system stack. Normally, the FIRQ only stores the CC (condition code) and the PC (Program Counter/return address) on the stack, so to keep compatability with the 6809, they included it as a selectable feature in the MD (Mode/status) register.

Inter-Register Instructions

The new Inter-Register instructions (ADCR, ADDR, CMPR, EORR, ORR, SBCR, and SUBR) all work the same as their register/memory (ADCA, ADDA, etc.) counterparts except that they operate between registers. All of the new instructions use the same post-byte information as the normal TFR instruction and use the format of R0,R1 (register 0 and Register 1 respectively) with the result going into R1. See Block Transfers for information on the TFR block move instructions.

Mixed-size inter-register operations default to using identical sized register. So TFR A,X actually executes as TFR D,X. You could also do 'lea(d) d,pc' calculations by doing 'addr pc,d'. As the new inter-register instructions can now perform math using the PC register, REALLY odd possibilities exist. Try looking at code like 'eorr d,pc', and figuring out where it ends up.

Inter-register instructions with 16-bit r1 and CC or DP (8-bit r2) are legal, but the results are unknown.

Bit Manipulation of Memory Locations

The AIM, EIM, OIM and TIM instructions all do logical bit manipulations to locations in memory, with the result stored into the location, and the respective bits for each instruction set in the CC register. They can be used in the DIRECT, INDEXED or EXTENDED adressing modes.

Instruction descriptions:

AIM - AND IN MEMORY
EIM - EOR IN MEMORY
OIM - OR IN MEMORY
TIM - TEST bits IN MEMORY

Instruction format: X, post byte, operand

Where X is the instruction op-code, post-byte contains the bits to AND, OR, EOR or TEST against the memory location, and the operand is the memory location or indexing post-byte depending on the mode of operation.

Mnemonic format:

Instruction logical operation value, memory location or index operation

Mnemonic example:

```
AIM #$0F,$E00
```

The example takes the contents of memory location $E00, does a LOGICAL and with the Value #$0F and then stores the result back into $E00.

Bit Transfers Between Memory Locations and Registers

The BAND, BIAND, BOR, BIOR, BEOR, BIEOR, LDBT, and STBT all do logical operations to bits for the n-th bit in a memory location and the m-th bit of a register. The LDBT and STBT instructions allow you to transfer certain bits between registers and memory locations. All instructions allow you to specify which register to use, which bit location to use in the register, which bit location to use in the memory location, and the memory location to use. This allows you to transfer/or do a logical operation with the 7th bit of a register and the 3rd bit of a memory location. All bits are accessible on either the register or memory locations. The only limitations are that the instructions can only be used with the A and B accumulators and the CC (condition Code) registers. It should also be noted that these instructions can only be used in the DIRECT addressing mode.

Instruction description:

BAND - AND a bit in a register with bit from memory location
BIAND - AND a bit in a register with the complement of the bit in memory
BOR - OR a bit in a register with a bit from a memory location
BIOR - OR a bit in a register with the complement of the bit in memory
BEOR - EOR a bit in a register with a bit from a memory location
BIEOR - EOR a bit in a register with the complement of the bit in memory
LDBT - Load a bit from a memory location into a bit in a register
STBT - Store a bit from a register into a memory location.

Instruction format:

x, post-byte, memory location

Where X is the instruction op-code, the post-byte contains the register, source and destination bit information and the memory location is the 8 bit value of the memory location to be used (Remember only DIRECT mode is allowed with these instructions).

Mnemonic format:

instruction, register, source bit, destination bit, memory location

Mnemonic example:

```
BOR A,1,7,$00
```

The example would take the first (1) bit of register A (A) and OR it into the 7th (7) bit of memory location $00 ($00) of the direct page (DP register value)

The post-byte of these instructions are not the same as the post-byte used in any other operation (indexed or inter-register) as all of the information (register, source and destination bit) is contained in one post-byte value.

Block Transfers

Block transfers are used to move a certain number of bytes from one place in memory to another with the use of one instruction. Two 16 bit registers (X, Y, U or S) are used to specify the source and destination addresses, and the size of the block to be transferred is specified with the W register. It should be noted that even though the IRQ and FIRQ only occur after the current instruction is finished, block moves can be interrupted. After the interrupt returns, the last byte read is read once more. i.e. It is read _twice_ by the CPU This can cause problems with memory mapped I/O devices, so caution is advised when using the block transfers. There isn't much control over these 4 instructions so the only thing applicable for them would be large block moves such as scrolling the screen or clearing an area in memory with a certain value.

TFM r0+,r1 and TFM r0,r1+ can be considered a poor mans DMA channel. Since all the data is either copied into or read from one memory location.

Four types of block transfers have been provided.

Mnemonic examples:

(R0 - source address register, R1 - destination address register.)

TFM r0+,r1+
- Transfer from R0 to R1 in incrementing order.

TFM r0-,r1-
- Transfer from R0 to R1 in decrementing order.

TFM r0+,r1
- Pour from R0 into R1, only incrementing R0 (R1 stays the same).

TFM r0,r1+
- Read from R0 into R1, only incrementing R0 (R1 stays the same).

Mnemonic example:

```
LDW #$100
LDX #$600
LDY #$700
TFM X+,Y+
```

The example would move 256 (LDW #$0100) bytes from #$600 (LDX #$0600) in memory to #$700 (LDY #$0700) in memory, incrementing the value of each register (X and Y), and decrementing the value of the W register each time a byte if moved.

When moves like this are done, the pointer registers (X and Y in the example) will not be the same value they were before the transfer was initiated, but will but will be their original values PLUS the value of the W register (#$100 in the example). So in the example once the move is complete, the value of X will be returned as #$700 and the value of Y will be returned as #$800. The value of W register will be 0.

It is illegal to use any of the CC, DP, W, V, 0, or PC registers as either a source or destination register. Note that the D register CAN be used with the TFM instructions.

New math commands

The 6309 has 3 additional math instructions. A 16 bit by 16 bit signed multiply (MULD), a 16 bit by 8 bit signed divide (DIVD) and a 32 bit by 16 bit signed divide (DIVQ). These instructions can all be used in Immediate, direct, indexed and extended addressing modes.

The MULD (16 bit by 16 bit) instruction does a signed multiply of the contents of the D register and a value from memory (or in direct mode). The signed result is stored in the Q register.

The DIVD (16 bit by 8 bit) instruction does a signed divide of the contents of the D register with a value from memory (or in direct mode). The signed result is stored with the quotient in W and the modulo (remainder) in D.

The DIVQ (32 bit by 16 bit) instruction does a signed divide of the contents of the Q register with a value from memory (or in direct mode). The signed result is stored with the quotient in W and the modulo (remainder) in D.

Error Trapping

The 6309 has an internal error trapping handler that will jump to a specific location in memory when either an error is encountered in the DIVision instructions (only divide by zero) or an illegal instruction is encountered. When an error is encountered, the 6309 will jump to the memory location contained in $FFF0 (and $FFF1) which was originally reserved by the 6809.

The trap may cause problems with machines that have $FF00 hardcoded with the values $0000. A new EPROM should be burned to correct for the new behaviour of the 6309.

As many people know, an illegal instruction trap is extremely useful for debugging programs, as it prevents the entire machine from crashing when a bug is encountered.

Note that many pseudo-legal instructions on the 6809 are now illegal on the 6309, e.g. $1020xxxx executes as an LBRA on a 6809, but results in a trap on a 6309.

Additional Instructions

The 6309 has MANY new instructions. Most are variations of old instructions of the 6809 for use with the new registers. The new instruction set can be used in both native and emulation mode. Here is a list of the new instructions of the 6309:

ADCD
- Adds immediate or memory operand to the D register plus the current status of the carry with the result going to D.

ADCR
- Adds two registers together plus the current status of the carry.

ADDE , ADDF, ADDW
- Add of immediate or memory operand to E, F or W with results going to E, F or W

ADDR
- Adds two registers together

ANDD
- Logical AND of immediate or memory operand to D register with result going to D.

ANDR
- Logical AND of a register with the contents of another register

ASLD (Same as LSLD)
- Arithmetic shift left. Shifts D one bit left, clearing LSB.

ASRD
- Arithmetic shift right of the D register with sign extending.

BITD
- Test any bit or bits of the D register.

BITMD
- Test any bit or bits of the MD (mode) register.

CLRD, CLRE, CLRF, CLRW
- Clear register D, E, F or W to zero.

CMPE, CMPF, CMPW
- Compares the contents of E, F or W with the immediate or memory operand. Sets all CC except H on result.

CMPR
- Compares one register to another and sets all CC bits except H on result.

COMD, COME, COMF, COMW
- One's complement D ,E, F, or W. Changes all zero's to one's and all one's to zero's.

DECD, DECE, DECF, DECW
- Decrement D, E, F, or W by 1.

DIVD, DIVQ
- Does a 16 bit by 8 bit (DIVD) or a 32 bit by 16 bit (DIVQ) signed divide with immediate or memory operand with quotient in W and modulo (remainder) in D.

EORD
 - Logical exclusive OR of D and immediate or memory operand.

EORR
 - Logical exclusive OR of one register with the value of another register.

INCD, INCE, INCF, INCW
 - Increment D, E, F or W by 1.

LDE, LDF, LDQ, LDW, LDMD
 - Standard loading of E, F, Q, W or MD with immediate data value or operand from memory. (LDMD only valid with IMMEDIATE mode)

LSLD (Same as ASLD)
 - Logical shift left. Shifts D one bit left, clearing LSB.

LSRD, LSRW
 - Logical shift right. Shifts D or W one bit right, clearing MSB.

MULD
 - Performs as 16bit by 16bit signed multiply with immediate or operand from memory. Result stored in Q.

NEGD
 - Two's complement D register.

ORD
 - Logical OR of register D and immediate or memory operand.

ORR
 - Logical OR of one register with another.

PSHSW, PSHUW
 - Stores contents of the W register on the (system or user) stack.

PULSW, PULUW
 - Pull value from (system or user) stack into register W.

ROLD, ROLW
 - Rotate D or W one bit left through the Carry Condition code.

RORD, RORW
 - Rotate D or W one bit right through the Carry Condition code.

SBCD
 - Subtract an immediate or memory operand plus any borrow in Carry from contents of D. Result stored in D.

SBCR
 - Subtract the value of one register from another plus any borrow in the CC carry.

SEXW
 - sign exdend the W register into the D register.

STE, STF, STQ, STW
 - Store register E, F, Q or W to memory location (E,F), two memory locations(W), or four memory locations (Q).

SUBE, SUBF, SUBW
 - Subtract immediate or memory operand from E, F or W. Result stored back in same register.

SUBR
 - Subtract the value of one register from another.

TFM (Block transfer)
 - Transfer W number of bytes from one location to another. Returns pointer registers offset of the starting value in the W register and returns the W register as 0. Indexed operation only

TSTD, TSTE, TSTF, TSTW
 - Test contents of D, E, F or W by setting N and X condition codes based on data in register.

The Opcode and Mnemonics opcode reference tables are both complete listings that contain both the Opcode instruction and the HEX equivalant in all available addressing modes. The first table is arranged sequentially by the binary opcodes, while the second table is arranged alphabetically by the Mnemonic instructions.

At the end of the second table there are data tables containing information on Bit transfer/manipulation, branch instructions, inter-register instructions, and general register and stack information. These are all helpful to the serious assembly language programmer, who should always have one.

Opcode table

Opcode (* 6309)	Mnemonic	Mode	Cycles 6809	Cycles (6309)	Length
00	NEG	Direct	6	(5)	2
* 01	OIM	Direct	6		3
* 02	AIM	Direct	6		3
03	COM	Direct	6	(5)	2
04	LSR	Direct	6	(5)	2
* 05	EIM	Direct	6		3
06	ROR	Direct	6	(5)	2
07	ASR	Direct	6	(5)	2
08	ASL/LSL	Direct	6	(5)	2
09	ROL	Direct	6	(5)	2
0A	DEC	Direct	6	(5)	2
* 0B	TIM	Direct	6		
0C	INC	Direct	6	(5)	2
0D	TST	Direct	6	(4)	2
0E	JMP	Direct	3	(2)	2
0F	CLR	Direct	6	(5)	2
10	(PREBYTE)				
11	(PREBYTE)				
12	NOP	Inherent	2	(1)	1
13	SYNC	Inherent	2	(1)	1
* 14	SEXW	Inherent	4		1
16	LBRA	Relative	5	(4)	3
17	LBSR	Relative	9	(7)	3
19	DAA	Inherent	2	(1)	1
1A	ORCC	Immediate	3	(2)	2
1C	ANDCC	Immediate	3		2
1D	SEX	Inherent	2	(1)	1
1E	EXG	Immediate	8	(5)	2
1F	TFR	Immediate	6	(4)	2
20	BRA	Relative	3		2
21	BRN	Relative	3		2
22	BHI	Relative	3		2
23	BLS	Relative	3		2
24	BHS/BCC	Relative	3		2
25	BLO/BCS	Relative	3		2
26	BNE	Relative	3		2
27	BEQ	Relative	3		2

Opcode (* 6309)	Mnemonic	Mode	Cycles	Length
28	BVC	Relative	3	2
29	BVS	Relative	3	2
2A	BPL	Relative	3	2
2B	BMI	Relative	3	2
2C	BGE	Relative	3	2
2D	BLT	Relative	3	2
2E	BGT	Relative	3	2
2F	BLE	Relative	3	2
30	LEAX	Indexed	4+	2
31	LEAY	Indexed	4+	2
32	LEAS	Indexed	4+	2
33	LEAU	Indexed	4+	2
34	PSHS	Immediate	5+ (4+)	2
35	PULS	Immediate	5+ (4+)	2
36	PSHU	Immediate	5+ (4+)	2
37	PULU	Immediate	5+ (4+)	2
39	RTS	Inherent	5 (1)	1
3A	ABX	Inherent	3 (1)	1
3B	RTI	Inherent	6/15 (17)	1
3C	CWAI	Immediate	22 (20)	2
3D	MUL	Inherent	11 (10)	1
3F	SWI	Inherent	19 (21)	1
40	NEGA	Inherent	2 (1)	1
43	COMA	Inherent	2 (1)	1
44	LSRA	Inherent	2 (1)	1
46	RORA	Inherent	2 (1)	1
47	ASRA	Inherent	2 (1)	1
48	ASLA/LSLA	Inherent	2 (1)	1
49	ROLA	Inherent	2 (1)	1
4A	DECA	Inherent	2 (1)	1
4C	INCA	Inherent	2 (1)	1
4D	TSTA	Inherent	2 (1)	1
4F	CLRA	Inherent	2 (1)	1
50	NEGB	Inherent	2 (1)	1
53	COMB	Inherent	2 (1)	1
54	LSRB	Inherent	2 (1)	1
56	RORB	Inherent	2 (1)	1
57	ASRB	Inherent	2 (1)	1
58	ASLB/LSLB	Inherent	2 (1)	1
59	ROLB	Inherent	2 (1)	1
5A	DECB	Inherent	2 (1)	1
5C	INCB	Inherent	2 (1)	1
5D	TSTB	Inherent	2 (1)	1
5F	CLRB	Inherent	2 (1)	1
60	NEG	Indexed	6+	2+
* 61	OIM	Indexed	6+	3+
* 62	AIM	Indexed	7	3+
63	COM	Indexed	6+	2+
64	LSR	Indexed	6+	2+
* 65	EIM	Indexed	7+	3+
66	ROR	Indexed	6+	2+
67	ASR	Indexed	6+	2+
68	ASL/LSL	Indexed	6+	2+
69	ROL	Indexed	6+	2+

Opcode (* 6309)	Mnemonic	Mode	Cycles	Length
6A	DEC	Indexed	6+	2+
* 6B	TIM	Indexed	7+	3+
6C	INC	Indexed	6+	2+
6D	TST	Indexed	6+ (5+)	2+
6E	JMP	Indexed	3+	2+
6F	CLR	Indexed	6+	2+
70	NEG	Extended	7 (6)	3
* 71	OIM	Extended	7	4
* 72	AIM	Extended	7	4
73	COM	Extended	7 (6)	3
74	LSR	Extended	7 (6)	3
76	ROR	Extended	7 (6)	3
* 75	EIM	Extended	7	4
77	ASR	Extended	7 (6)	3
78	ASL/LSL	Extended	7 (6)	3
79	ROL	Extended	7 (6)	3
7A	DEC	Extended	7 (6)	3
* 7B	TIM	Extended	7	4
7C	INC	Extended	7 (6)	3
7D	TST	Extended	7 (5)	3
7E	JMP	Extended	4 (3)	3
7F	CLR	Extended	7 (6)	3
80	SUBA	Immediate	2	2
81	CMPA	Immediate	2	2
82	SBCA	Immediate	2	2
83	SUBD	Immediate	4 (3)	3
84	ANDA	Immediate	2	2
85	BITA	Immediate	2	2
86	LDA	Immediate	2	2
88	EORA	Immediate	2	2
89	ADCA	Immediate	2	2
8A	ORA	Immediate	2	2
8B	ADDA	Immediate	2	2
8C	CMPX	Immediate	4 (3)	3
8D	BSR	Relative	7 (6)	2
8E	LDX	Immediate	3	3
90	SUBA	Direct	4 (3)	2
91	CMPA	Direct	4 (3)	2
92	SBCA	Direct	4 (3)	2
93	SUBD	Direct	6 (4)	2
94	ANDA	Direct	4 (3)	2
95	BITA	Direct	4 (3)	2
96	LDA	Direct	4 (3)	2
97	STA	Direct	4 (3)	2
98	EORA	Direct	4 (3)	2
99	ADCA	Direct	4 (3)	2
9A	ORA	Direct	4 (3)	2
9B	ADDA	Direct	4 (3)	2
9C	CMPX	Direct	6 (4)	2
9D	JSR	Direct	7 (6)	2
9E	LDX	Direct	5 (4)	2
9F	STX	Direct	5 (4)	2
A0	SUBA	Indexed	4+	2+
A1	CMPA	Indexed	4+	2+

Opcode (* 6309)	Mnemonic	Mode	Cycles	Length
A2	SBCA	Indexed	4+	2+
A3	SUBD	Indexed	6+ (5+)	2+
A4	ANDA	Indexed	4+	2+
A5	BITA	Indexed	4+	2+
A6	LDA	Indexed	4+	2+
A7	STA	Indexed	4+	2+
A8	EORA	Indexed	4+	2+
A9	ADCA	Indexed	4+	2+
AA	ORA	Indexed	4+	2+
AB	ADDA	Indexed	4+	2+
AC	CMPX	Indexed	6+ (5+)	2+
AD	JSR	Indexed	7+ (6+)	2+
AE	LDX	Indexed	5+	2+
AF	STX	Indexed	5+	2+
B0	SUBA	Extended	5 (4)	3
B1	CMPA	Extended	5 (4)	3
B2	SBCA	Extended	5 (4)	3
B3	SUBD	Extended	7 (5)	3
B4	ANDA	Extended	5 (4)	3
B5	BITA	Extended	5 (4)	3
B6	LDA	Extended	5 (4)	3
B7	STA	Extended	5 (4)	3
B8	EORA	Extended	5 (4)	3
B9	ADCA	Extended	5 (4)	3
BA	ORA	Extended	5 (4)	3
BB	ADDA	Extended	5 (4)	3
BC	CMPX	Extended	7 (5)	3
BD	JSR	Extended	8 (7)	3
BE	LDX	Extended	6 (5)	3
BF	STX	Extended	6 (5)	3
C0	SUBB	Immediate	2	2
C1	CMPB	Immediate	2	2
C2	SBCB	Immediate	2	2
C3	ADDD	Immediate	4 (3)	3
C4	ANDB	Immediate	2	2
C5	BITB	Immediate	2	2
C6	LDB	Immediate	2	2
C8	EORB	Immediate	2	2
C9	ADCB	Immediate	2	2
CA	ORB	Immediate	2	2
CB	ADDB	Immediate	2	2
CC	LDD	Immediate	3	3
* CD	LDQ	Immediate	5	5
CE	LDU	Immediate	3	3
D0	SUBB	Direct	4 (3)	2
D1	CMPB	Direct	4 (3)	2
D2	SBCB	Direct	4 (3)	2
D3	ADDD	Direct	6 (4)	2
D4	ANDB	Direct	4 (3)	2
D5	BITB	Direct	4 (3)	2
D6	LDB	Direct	4 (3)	2
D7	STB	Direct	4 (3)	2
D8	EORB	Direct	4 (3)	2
D9	ADCB	Direct	4 (3)	2

Opcode (* 6309)	Mnemonic	Mode	Cycles	Length
DA	ORB	Direct	4 (3)	2
DB	ADDB	Direct	4 (3)	2
DC	LDD	Direct	5 (4)	2
DD	STD	Direct	5 (4)	2
DE	LDU	Direct	5 (4)	2
DF	STU	Direct	5 (4)	2
E0	SUBB	Indexed	4+	2+
E1	CMPB	Indexed	4+	2+
E2	SBCB	Indexed	4+	2+
E3	ADDD	Indexed	6+ (5+)	2+
E4	ANDB	Indexed	4+	2+
E5	BITB	Indexed	4+	2+
E6	LDB	Indexed	4+	2+
E7	STB	Indexed	4+	2+
E8	EORB	Indexed	4+	2+
E9	ADCB	Indexed	4+	2+
EA	ORB	Indexed	4+	2+
EB	ADDB	Indexed	4+	2+
EC	LDD	Indexed	5+	2+
ED	STD	Indexed	5+	2+
EE	LDU	Indexed	5+	2+
EF	STU	Indexed	5+	2+
F0	SUBB	Extended	5 (4)	3
F1	CMPB	Extended	5 (4)	3
F2	SBCB	Extended	5 (4)	3
F3	ADDD	Extended	7 (5)	3
F4	ANDB	Extended	5 (4)	3
F5	BITB	Extended	5 (4)	3
F6	LDB	Extended	5 (4)	3
F7	STB	Extended	5 (4)	3
F8	EORB	Extended	5 (4)	3
F9	ADCB	Extended	5 (4)	3
FA	ORB	Extended	5 (4)	3
FB	ADDB	Extended	5 (4)	3
FC	LDD	Extended	6 (5)	3
FD	STD	Extended	6 (5)	3
FE	LDU	Extended	6 (5)	3
FF	STU	Extended	6 (5)	3
1021	LBRN	Reletive	5/6 ( )	4
1022	LBHI	Reletive	5/6 ( )	4
1023	LBLS	Reletive	5/6 ( )	4
1024	LBHS/LBCC	Reletive	5/6 ( )	4
1025	LBCS/LBLO	Reletive	5/6 ( )	4
1026	LBNE	Reletive	5/6 ( )	4
1027	LBEQ	Reletive	5/6 ( )	4
1028	LBVC	Reletive	5/6 ( )	4
1029	LBVS	Reletive	5/6 ( )	4
102A	LBPL	Reletive	5/6 ( )	4
102B	LBMI	Reletive	5/6 ( )	4
102C	LBGE	Reletive	5/6 ( )	4
102D	LBLT	Reletive	5/6 ( )	4
102E	LBGT	Reletive	5/6 ( )	4
102F	LBLE	Reletive	5/6 ( )	4
* 1030	ADDR	Register	4	3

	Opcode (* 6309)	Mnemonic	Mode	Cycles	Length
	* 1031	ADCR	Register	4	3
	* 1032	SUBR	Register	4	3
	* 1033	SBCR	Register	4	3
	* 1034	ANDR	Register	4	3
	* 1035	ORR	Register	4	3
	* 1036	EORR	Register	4	3
	* 1037	CMPR	Register	4	3
	* 1038	PSHSW	Register	6	2
	* 1039	PULSW	Register	6	2
	* 103A	PSHUW	Register	6	2
	* 103B	PULUW	Register	6	2
	103F	SWI2	Inherent	20 (22)	2
	* 1040	NEGD	Inherent	3 (2)	2
	* 1043	COMD	Inherent	3 (2)	2
	* 1044	LSRD	Inherent	3 (2)	2
	* 1046	RORD	Inherent	3 (2)	2
	* 1047	ASRD	Inherent	3 (2)	2
	* 1048	ASLD/LSLD	Inherent	3 (2)	2
	* 1049	ROLD	Inherent	3 (2)	2
	* 104A	DECD	Inherent	3 (2)	2
	* 104C	INCD	Inherent	3 (2)	2
	* 104D	TSTD	Inherent	3 (2)	2
	* 104F	CLRD	Inherent	3 (2)	2
	* 1053	COMW	Inherent	3 (2)	2
	* 1054	LSRW	Inherent	3 (2)	2
?	* 1056	RORW	Inherent	3 (2)	2
	* 1059	ROLW	Inherent	3 (2)	2
	* 105A	DECW	Inherent	3 (2)	2
	* 105C	INCW	Inherent	3 (2)	2
	* 105D	TSTW	Inherent	3 (2)	2
	* 105F	CLRW	Inherent	3 (2)	2
	* 1080	SUBW	Immediate	5 (4)	4
	* 1081	CMPW	Immediate	5 (4)	4
	* 1082	SBCD	Immediate	5 (4)	4
	1083	CMPD	Immediate	5 (4)	4
	* 1084	ANDD	Immediate	5 (4)	4
	* 1085	BITD	Immediate	5 (4)	4
	* 1086	LDW	Immediate	5 (4)	4
	* 1088	EORD	Immediate	5 (4)	4
	* 1089	ADCD	Immediate	5 (4)	4
	* 108A	ORD	Immediate	5 (4)	4
	* 108B	ADDW	Immediate	5 (4)	4
	108C	CMPY	Immediate	5 (4)	4
	108E	LDY	Immediate	5 (4)	4
	* 1090	SUBW	Direct	7 (5)	3
	* 1091	CMPW	Direct	7 (5)	3
	* 1092	SBCD	Direct	7 (5)	3
	1093	CMPD	Direct	7 (5)	3
	* 1094	ANDD	Direct	7 (5)	3
	* 1095	BITD	Direct	7 (5)	3
	* 1096	LDW	Direct	6 (5)	3
	* 1097	STW	Direct	6 (5)	3
	* 1098	EORD	Direct	7 (5)	3
	* 1099	ADCD	Direct	7 (5)	3

Opcode (* 6309)	Mnemonic	Mode	Cycles	Length
* 109A	ORD	Direct	7 (5)	3
* 109B	ADDW	Direct	7 (5)	3
109C	CMPY	Direct	7 (5)	3
109E	LDY	Direct	6 (5)	3
109F	STY	Direct	6 (5)	3
* 10A0	SUBW	Indexed	7+ (6+)	3+
* 10A1	CMPW	Indexed	7+ (6+)	3+
* 10A2	SBCD	Indexed	7+ (6+)	3+
10A3	CMPD	Indexed	7+ (6+)	3+
* 10A4	ANDD	Indexed	7+ (6+)	3+
* 10A5	BITD	Indexed	7+ (6+)	3+
* 10A6	LDW	Indexed	6+	3+
* 10A7	STW	Indexed	6+	3+
* 10A8	EORD	Indexed	7+ (6+)	3+
* 10A9	ADCD	Indexed	7+ (6+)	3+
* 10AA	ORD	Indexed	7+ (6+)	3+
* 10AB	ADDW	Indexed	7+ (6+)	3+
10AC	CMPY	Indexed	7+ (6+)	3+
10AE	LDY	Indexed	6	3+
10AF	STY	Indexed	6	3+
* 10B0	SUBW	Extended	8 (6)	4
* 10B1	CMPW	Extended	8 (6)	4
* 10B2	SBCD	Extended	8 (6)	4
10B3	CMPD	Extended	8 (6)	4
* 10B4	ANDD	Extended	8 (6)	4
* 10B5	BITD	Extended	8 (6)	4
* 10B6	LDW	Extended	7 (6)	4
* 10B7	STW	Extended	7 (6)	4
* 10B8	EORD	Extended	8 (6)	4
* 10B9	ADCD	Extended	8 (6)	4
* 10BA	ORD	Extended	8 (6)	4
* 10BB	ADDW	Extended	8 (6)	4
10BC	CMPY	Extended	8 (6)	4
10BE	LDY	Extended	7 (6)	4
10BF	STY	Extended	7 (6)	4
10CE	LDS	Immediate	4	4
* 10DC	LDQ	Direct	8 (7)	3
* 10DD	STQ	Direct	8 (7)	3
10DE	LDS	Direct	6 (5)	3
10DF	STS	Direct	6 (5)	3
* 10DC	LDQ	Indexed	8+	3+
* 10DD	STQ	Indexed	8+	3+
10EE	LDS	Indexed	6+	3+
10EF	STS	Indexed	6+	3+
* 10DC	LDQ	Extended	9 (8)	4
* 10DD	STQ	Extended	9 (8)	4
10FE	LDS	Extended	7 (6)	4
10FF	STS	Extended	7 (6)	4
* 1130	BAND	Memory	7 (6)	4
* 1131	BIAND	Memory	7 (6)	4
* 1132	BOR	Memory	7 (6)	4
* 1133	BIOR	Memory	7 (6)	4
* 1134	BEOR	Memory	7 (6)	4
* 1135	BIEOR	Memory	7 (6)	4

Opcode (* 6309)	Mnemonic	Mode	Cycles	Length
* 1136	LDBT	Memory	7 (6)	4
* 1137	STBT	Memory	8 (7)	4
* 1138	TFM R+,R+	Register	6+3n	3
* 1139	TFM R-,R-	Register	6+3n	3
* 113A	TFM R+,R	Register	6+3n	3
* 113B	TFM R,R+	Register	6+3n	3
* 113C	BITMD	Immediate	4	3
* 113D	LDMD	Immediate	5	5
113F	SWI2	Inherent	20 ( )	2
* 1143	COME	Inherent	3 (2)	2
* 114A	DECE	Inherent	3 (2)	2
* 114C	INCE	Inherent	3 (2)	2
* 114D	TSTE	Inherent	3 (2)	2
* 114F	CLRE	Inherent	3 (2)	2
* 1153	COMF	Inherent	3 (2)	2
* 115A	DECF	Inherent	3 (2)	2
* 115C	INCF	Inherent	3 (2)	2
* 115D	TSTF	Inherent	3 (2)	2
* 115F	CLRF	Inherent	3 (2)	2
11AC	CMPS	Indexed	7 ( )	3
* 1180	SUBE	Immediate	3	3
* 1181	CMPE	Immediate	3	3
1183	CMPU	Immediate	5 (4)	4
* 1186	LDE	Immediate	3	3
* 118B	ADDE	Immediate	3	3
118C	CMPS	Immediate	5 (4)	4
* 118D	DIVD	Immediate	25	4
* 118E	DIVQ	Immediate	36	4
* 118F	MULD	Immediate	28	4
* 1190	SUBE	Direct	5 (4)	3
* 1191	CMPE	Direct	5 (4)	3
1193	CMPU	Direct	7 (5)	3
* 1196	LDE	Direct	5 (4)	3
* 1197	STE	Direct	5 (4)	3
* 119B	ADDE	Direct	5 (4)	3
119C	CMPS	Direct	7 (5)	3
* 119D	DIVD	Direct	27 (26)	3
* 119E	DIVQ	Direct	36 (35)	3
* 119F	MULD	Direct	30 (29)	3
* 11A0	SUBE	Indexed	5+	3+
* 11A1	CMPE	Indexed	5+	3+
11A3	CMPU	Indexed	7+ (6+)	3+
* 11A6	LDE	Indexed	5+	3+
* 11A7	STE	Indexed	5+	3+
* 11AB	ADDE	Indexed	5+	3+
11AC	CMPS	Indexed	7+ (6+)	3+
* 11AD	DIVD	Indexed	27+	3+
* 11AE	DIVQ	Indexed	36+	3+
* 11AF	MULD	Indexed	30+	3+
* 11B0	SUBE	Extended	6 (5)	4

Opcode (* 6309)	Mnemonic	Mode	Cycles	Length
* 11B1	CMPE	Extended	6 (5)	4
11B3	CMPU	Extended	8 (6)	4
* 11B6	LDE	Extended	6 (5)	4
* 11B7	STE	Extended	6 (5)	4
* 11BB	ADDE	Extended	6 (5)	4
11BC	CMPS	Extended	8 (6)	4
* 11BD	DIVD	Extended	28 (27)	4
* 11BE	DIVQ	Extended	37 (36)	4
* 11BF	MULD	Extended	31 (30)	4
* 11C0	SUBF	Immediate	3	3
* 11C1	CMPF	Immediate	3	3
* 11C6	LDF	Immediate	3	3
* 11CB	ADDF	Immediate	3	3
* 11D0	SUBF	Direct	5 (4)	3
* 11D1	CMPF	Direct	5 (4)	3
* 11D6	LDF	Direct	5 (4)	3
* 11D7	STF	Direct	5 (4)	3
* 11DB	ADDF	Direct	5 (4)	3
* 11E0	SUBF	Indexed	5+	3+
* 11E1	CMPF	Indexed	5+	3+
* 11E6	LDF	Indexed	5+	3+
* 11E7	STF	Indexed	5+	3+
* 11EB	ADDF	Indexed	5+	3+
* 11F0	SUBF	Extended	6 (5)	4
* 11F1	CMPF	Extended	6 (5)	4
* 11F6	LDF	Extended	6 (5)	4
* 11F7	STF	Extended	6 (5)	4
* 11FB	ADDF	Extended	6 (5)	4

Mnem	Immed.			Direct			Indexed			Extended			Inherent		
	OP	~/~	#	OP	~/~	+	OP	~/~	#	OP	~/~	#	OP	~/~	#
ABX													3A	3/1	1
ADCA	89	2	2	99	4/3	2	A9	4+	2+	B9	5/4	3			
ADCB	C9	2	2	D9	4/3	2	E9	4+	2+	F9	5/3	3			
*ADCD	10	5/4	4	10	7/5	3	10	7+/6+	3+	10	8/6	4			
	89			99			A9			B9					
ADDA	8B	2	2	9B	4/3	2	AB	4+	2+	BB	5/4	3			
ADDB	CB	2	2	DB	4/3	2	EB	4+	2+	FB	5/4	3			
ADDD	C3	4/3	3	D3	6/4	2	E3	6+/5+	2+	F3	7/5	3			
*ADDE	11	3	3	11	5/4	3	11	5+	3+	11	6/5	4			
	8B			9B			AB			BB					
*ADDF	11	3	3	11	5/4	3	11	5+	3+	11	6/5	4			
	CB			DB			EB			FB					
*ADDW	10	5/4	4	10	7/5	3	10	7+/6+	3+	10	8/6	4			
	8B			9B			AB			BB					
*AIM				02	6	3	62	7+	3+	72	7	4			
ANDA	84	2	2	94	4/3	2	A4	4+	2	B4	5/4	3			
ANDB	C4	2	2	D4	4/3	2	E4	4+	2	F4	5/4	3			
ANDCC	1C	3	2												
*ANDD	10	5/4	4	10	7/5	3	10	7+/6+	3+	10	8/6	4			
	84			94			A4			B4					
ASLA													48	2/1	1
ASLB													58	2/1	1
*ASLD													10	3/2	2
													48		
ASL				08	6/5	2	68	6+	2+	78	7/6	3			
ASRA													47	2/1	1
ASRB													57	2/1	1
*ASRD													10	3/2	1
													47		
ASR				07	6/6	2	67	6+	2+	77	7/6	3			
BITA	85	2	2	95	4/3	2	A5	4+	2+	B5	5/4	3			
BITB	C5	2	2	D5	4/3	2	E5	4+	2+	F5	5/4	3			
BITD	10	5/4	4	10	7/5	3	10	7+/6+	3+	10	8/6	4			
	85			95			A5			B5					
BITMD	11	4	3												
	3C														
CLRA													4F	2/1	1
CLRB													5F	2/1	1
*CLRD													10	3/2	2
													4F		
*CLRE													11	3/2	2
													4F		
*CLRF													11	3/2	2
													5F		
*CLRW													10	3/2	2
													5F		
CLR				0F	6/5	2	6F	6+	2+	7F	7/6	3			

Mnem	Immed.			Direct			Indexed			Extended			Inherent		
	OP	~/~	#	OP	~/~	+	OP	~/~	#	OP	~/~	#	OP	~/~	#
CMPA	81	2	2	91	4/3	2	A1	4+	2+	B1	5/4	3			
CMPB	C1	2	2	D1	4/3	2	E1	4+	2+	F1	5/4	3			
CMPD	10	5/4	4	10	7/5	3	10	7+/6+	3+	10	8/6	4			
	83			93			A3			B3					
*CMPE	11	3	3	11	5/4	3	11	5+	3+	11	6/5	4			
	81			91			A1			B1					
*CMPF	11	3	3	11	5/4	3	11	5+	3+	11	6/5	4			
	C1			D1			E1			F1					
CMPS	11	5/4	4	11	7/5	3	11	7+/6+	3+	11	8/6	4			
	8C			9C			AC			BC					
CMPU	11	5/4	4	11	7/5	3	11	7+/6+	3+	11	8/6	4			
	83			93			A3			B3					
*CMPW	10	5/4	4	10	7/5	3	10	7+/6+	3+	10	8/6	4			
	81			91			A1			B1					
CMPX	8C	4/3	3	9C	6/4	2	AC	6+/5+	2+	BC	7/5	3			
CMPY	10	5/4	4	10	7/5	3	10	7+/6+	3+	10	8/6	4			
	8C			9C			AC			BC					
COMA													43	2/1	1
COMB													53	2/1	1
*COMD													10	3/2	2
													43		
*COME													11	3/2	2
													43		
*COMF													11	3/2	2
													53		
*COMW													10	3/2	2
													53		
COM				03	6/5	2	63	6+	2+	73	7/6	3			
CWAI	3C	22/20	2												
DAA													19	2/1	1
DECA													4A	2/1	1
DECB													5A	2/1	1
*DECD													10	3/2	2
													4A		
*DECE													11	3/2	2
													4A		
*DECF													11	3/2	2
													5A		
*DECW													10	3/2	2
													5A		
DEC				0A	6/5	2	6A	6+	2+	7A	7/6	3			
*DIVD	11	25	3	11	27/26	3	11	27+	3+	11	28/27	4			
	8D			9D			AD			BD					
*DIVQ	11	34	4	11	36/35	3	11	36+	3+	11	37/36	4			
	8E			9E			AE			BE					
*EIM				05	6	3	65	7+	3+	75	7	4			
EORA	88	2	2	98	4/3	2	A8	4+	2+	B8	5/4	3			
EORB	C8	2	#	D8	4/3	2	E8	4+	2+	F8	5/4	3			
*EORD	10	5/4	4	10	7/5	3	10	7+/6+	3+	10	8/6	4			
	88			98			A8			B8					

Mnem	Immed.			Direct			Indexed			Extended			Inherent		
	OP	~/~	#	OP	~/~	+	OP	~/~	#	OP	~/~	#	OP	~/~	#
EXG	1E	8/5	2												
INCA													4C	2/1	1
INCB													5C	2/1	1
*INCD													10	3/2	2
													4C		
*INCE													11	3/2	2
													4C		
*INCF													11	3/2	2
													5C		
*INCW													10	3/2	2
													5C		
INC				0C	6/5	2	6C	6+	2+	7C	7/6	3			
JMP				0E	3/2	2	6E	3+	2+	7E	4/3	3			
JSR				9D	7/6	2	AD	7+/6+	2+	BD	8/7	3			
LDA	86	2	2	96	4/3	2	A6	4+	2+	B6	5/4	3			
LDB	C6	2	2	D6	4/3	2	E6	4+	2+	F6	5/4	3			
LDD	CC	3	3	DC	5/4	2	EC	5+	2+	FC	6/5	3			
*LDE	11	3	3	11	5/4	3	11	5+	3+	11	6/5	4			
	86			96			A6			B6					
*LDF	11	3	3	11	5/4	3	11	5+	3+	11	6/5	4			
	C6			D6			E6			F6					
*LDQ	CD	5	5	10	8/7	3	10	8+	3+	10	9/8	4			
				DC			EC			FC					
LDS	10	4	4	10	6/5	3	10	6+	3+	10	7/6	4			
	CE			DE			EE			FE					
LDU	CE	3	3	DE	5/4	2	EE	5+	2+	FE	6/5	3			
*LDW	10	4	4	10	6/5	3	10	6+	3+	10	7/6	4			
	86			96			A6			B6					
LDX	8E	3	3	9E	5/4	2	AE	5+	2+	BE	6/5	3			
LDY	10	4	4	10	6/5	3	10	6+	3+	10	7/6	4			
	8E			9E			AE			BE					
*LDMD	11	5	3												
	3D														
LEAS							32	4+	2+						
LEAU							33	4+	2+						
LEAX							30	4+	2+						
LEAY							31	4+	2+						
LSLA/LSLB/LSLD/LSL - Same as ASL															
LSRA													44	2/1	1
LSRB													54	2/1	1
*LSRD													10	3/2	2
													44		
*LSRW													10	3/2	2
													54		
LSR				04	6/5	2	64	6+	2+	74	7/6	3			
MUL													3D	11/10	1
*MULD	11	28	4	11	30/29	3	11	30+	3+	11	31/30	4			
	8F			9F			AF			BF					

Mnem	Immed.			Direct			Indexed			Extended			Inherent		
	OP	~/~	#	OP	~/~	+	OP	~/~	#	OP	~/~	#	OP	~/~	#
NEGA													40	2/1	1
NEGB													50	2/1	1
*NEGD													10	3/2	2
													40		
NEG				00	6/5	2	60	6+	2+	70	7/6	3			
NOP													12	2/1	1
*OIM				01	6	3	61	7+	3+	71	7	4			
ORA	8A	2	2	9A	4/3	2	AA	4+	2	BA	5/4	3			
ORB	CA	2	2	DA	4/3	2	EA	4+	2	FA	5/4	3			
ORCC	1A	3/2	2												
*ORD	10	5/4	4	10	7/5	3	10	7+/6+	3+	10	8/6	4			
	8A			9A			AA			BA					
PSHS	34	5+/4+	2												
PSHU	36	5+/4+	2												
*PSHSW	10	6	2												
	38	6	2												
*PSHUW	10	6	2												
	3A	6	2												
PULS	35	5+/4+	2												
PULU	37	5+/4+	2												
*PULSW	10	6	2												
	39														
*PULUW	10	6	2												
	3B														
ROLA													49	2/1	1
ROLB													59	2/1	1
*ROLD													10	3/2	2
													49		
*ROLW													10	3/2	2
													59		
ROL				09	6/5	2	69	6+	2+	79	7/6	3			
RORA													46	2/1	1
RORB													56	2/1	1
*RORD													10	3/2	2
													46		
*RORW													10	3/2	2
													56		
ROR				06	6/5	2	66	6+	2+	76	7/6	3			
RTI													3B	6/17	1
														15/17	
RTS													39	5/4	1
SBCA	82	2	2	92	4/3	2	A2	4+	2+	B2	5/4	3			
SBCB	C2	2	2	D2	4/3	2	E2	4+	2+	F2	5/2	3			
*SBCD	10	5/4	4	10	7/5	3	10	7+/6+	3+	10	8/6	4			
	82			92			A2			B2					
SEX													1D	2/1	1
*SEXW													14	4	1

Mnem	Immed.			Direct			Indexed			Extended			Inherent		
	OP	~/~	#	OP	~/~	+	OP	~/~	#	OP	~/~	#	OP	~/~	#
STA				97	4/3	2	A7	4+	2+	B7	5/4	3			
STB				D7	4/3	2	E7	4+	2+	F7	5/4	3			
STD				DC	5/4	2	EC	5+	2+	FC	6/5	3			
*STE				11	5/4	3	11	5+	3+	11	6/5	4			
				97			A7			B7					
*STF				11	5/4	3	11	5+	3+	11	6/5	4			
				D7			E7			F7					
*STQ				10	8/7	3	10	8+	3+	10	9/8	4			
				DD			ED			FD					
*STS				10	6/5	3	10	6+	3+	10	7/6	4			
				DF			EF			FF					
STU				DF	5/4	2	EF	5+	2+	FF	6/5	3			
*STW				10	6/5	3	10	6+	3+	10	7/6	4			
				97			A7			B7					
STX				9F	5/4	2	AF	5+	2+	BF	6/5	3			
STY				10	6/5	3	10	6+	3+	10	7/6	4			
				9F			AF			BF					
SUBA	80	2	2	90	4/3	2	A0	4+	2+	B0	5/4	3			
SUBB	C0	2	2	D0	4/3	2	E0	4+	2+	F0	5/4	3			
SUBD	83	4/3	3	93	6/4	3	A3	6+/5+	2+	B3	7/5	3			
*SUBE	11	3	3	11	5/4	3	11	5+	3+	11	6/5	4			
	80			90			A0			B0					
*SUBF	11	3	3	11	5/4	3	11	5+	3+	11	6/5	4			
	C0			D0			E0			F0					
*SUBW	10	5/4	4	10	7/5	3	10	7+/6+	3+	10	8/6	4			
	80			90			A0			B0					
SWI													3F	19/21	1
SWI2													10	20/22	2
													3F		
SWI3													11	20/22	2
													3F		
SYNC													13	2+/1+	1
TFR 1	1F	6/4	2												
*TIM				0B	6	3	6B	7+	3+	7B	5	4			
TSTA													4D	2/1	1
TSTB													5D	2/1	1
*TSTD													10	3/2	2
													4D		
*TSTE													11	3/2	2
													4D		
*TSTF													11	3/2	2
													5D		
*TSTW													10	3/2	2
													5D		
TST				0D	6/4	2	6D	6+/5+	2+	7D	7/5	3			

Branch Instructions

Mnem	Immed.		
	OP	~/~	#
BCC	24	3	2
LBCC	10	5/6	4
	24		
BCS	25	3	2
LBCS	10	5/6	4
	25		
BEQ	27	3	2
LBEQ	10	5/6	4
	27		
BGE	2C	3	2
LBGE	10	5/6	4
	2C		
BGT	2E	3	2
LBGT	10	5/6	4
	2E		
BHI	22	3	2
LBHI	10	5/6	4
	22		
BHS	2F	3	2
LBHS	10	5/6	4
	2F		

Mnem	Immed.		
	OP	~/~	#
BLE	2F	3	2
LBLE	10	5/6	4
	2F		
BLO	25	3	2
LBLO	10	5/6	4
	25		
BLS	23	3	2
LBLS	10	5/6	4
	23		
BLT	2D	3	2
LBLT	10	5/6	4
	2D		
BMI	28	3	2
LBMI	10	5/6	4
	28		
BNE	26	3	2
LBNE	10	5/6	4
	26		

Mnem	Immed.		
	OP	~/~	#
BPL	2A	3	2
LBPL	10	5/6	4
	2A		
BRA	20	3	2
LBRA	16	5/4	3
BRN	21	3	2
LBRN	10	5/6	4
	21		
BSR	8D	7/6	2
LBSR	17	9/7	3
BVC	28	3	2
LBVC	10	5/6	4
	28		
BVS	29	3	2
LBVS	10	5/6	4
	29		

Bit Transfer/Manipulation

Mnem	Direct		
	OP	~/~	#
*BAND	11	7/6	4
	30		
*BIAND	11	7/6	4
	31		
*BOR	11	7/6	4
	32		
*BIOR	11	7/6	4
	33		
*BEOR	11	7/6	4
	34		
*BIEOR	11	7/6	4
	35		
*LDBT	11	7/6	4
	36		
*STBT	11	8/7	4
	37		

Post-Byte

7	6	5	4	3	2	1	0

Bits 7 and 6: Register

00 - CC 10 - B
01 - A 11 - Unused

Bits 5, 4 and 3: Source Bit

Bits 2, 1 and 0: Destination bit

Source/Destination Bit in binary form:

0 - 000 2 - 010 5 - 100 6 - 110
1 - 001 3 - 011 5 - 101 7 - 111

Both the source and destination bit portions of the post-byte are looked at by the 6309 as the actual bit NUMBER to transfer/store. Use the binary equivilant of the numbers (0 thru 7) and position them into the bit area of the post byte.

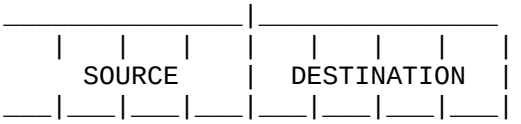
Logical Memory Operations

Mnem	Immed.		Direct			Indexed			Extended			Inherent		
	OP	~/~	#	OP	~/~	#	OP	~/~	#	OP	~/~	#	OP	~/~	#
*AIM				02	6	3	62	7+	3+	72	7	4			
*EIM				05	6	3	65	7+	3+	75	7	4			
*OIM				01	6	3	61	7+	3+	71	7	4			
*TIM				0B	6	3	6B	7+	3+	7B	5	4			

Inter-Register Instructions

Mnem	Forms	Register		
		OP	~/~	+
*ADCR	R0,R1	10	4	3
		31		
*ADDR	R0,R1	10	4	3
		30		
*ANDR	R0,R1	10	4	3
		34		
*CMPR	R0,R1	10	4	3
		37		
*EORR	R0,R1	10	4	3
		36		
EXG	R0,R1	1E	8/5	2
*ORR	R0,R1	10	4	3
		35		
*SBCR	R0,R1	10	4	3
		33		
*SUBR	R0,R1	10	4	3
		32		
TFR	R0,R1	1F	6/4	2
*TFM	R0+,R1+	11	6+3n	3
		38		
*TFM	R0-,R1-	11	6+3n	3
		39		
*TFM	R0+,R1	11	6+3n	3
		3A		
*TFM	R0,R1+	11	6+3n	3
		3B		

Transfer/Exchange and Inter-Register Post Byte



Register Field
(source or destination)

0000 - D (A:B)	1000 - A
0001 - X	1001 - B
0010 - Y	1010 - CCR
0011 - U	1011 - DPR
0100 - S	1100 - 0
0101 - PC	1101 - 0
0110 - W	1110 - E
0111 - V	1111 - F

The results of all Inter-Register operations are passsed into R1 with the exception of EXG which exchanges the values of registers and the TFR block transfers.

The register field codes %1100 and %1101 are both zero registers. They can be used as source or destination.

Indexed Address Modes and Post byte Information

Non-Indirect Modes					
Type	Forms	Assembler form	PostByte OP code	+/+ ~/~	+ #
Constant offset from R	No offset	,R	1rr00100	0	0
	5 bit offset	n,R	0rrnnnnn	1	0
	8 bit offset	n,R	1rr01000	1	1
	16 bit offset	n,R	1rr01001	4/3	2
Accumulator offset from R (Twos complement *offset) *	A - Register	A,R	1rr00110	1	0
	B - Register	B,R	1rr00101	1	0
	E - Register	E,R	1rr00111	1	0
	F - Register	F,R	1rr01010	1	0
	D - Register	D,R	1rr01011	4/2	0
	W - Register	W,R	1rr01110	4/1	0
Auto increment and decrement of R	Increment 1	,R+	1rr00000	2/1	0
	Increment 2	,R++	1rr00001	3/2	0
	Decrement 1	,-R	1rr00010	2/1	0
	Decrement 2	,--R	1rr00011	3/2	0
Constant offset from PC (Twos complement offset)	8 bit offset	n,PC	1xx01100	1	1
	16 bit offset	n,PC	1xx01101	5/3	2
*Relative to W *(Twos complement offset)	No Offset	,W	10001111	0	0
	16 bit offset	n,W	10101111	5/2	2
* AutoIncrement W	Increment 2	,W++	11001111	3/1	0
* AutoDecrement W	Decrement 2	,--W	11101111	3/1	0

Indirect Modes					
Type	Forms	Assembler form	Post--byte OP code	+ ~	+ #
Constant offset from R	No offset	[,R]	1rr10100	3	0
	5 bit offset	[n,R]	Defaults to 8 bit		
	8 bit offset	[n,R]	1rr11000	4	1
	16 bit offset	[n,R]	1rr11001	7	2
Accumulator offset from R (Twos complement *offset)	A - Register	[A,R]	1rr10110	4	0
	B - Register	[B,R]	1rr10101	4	0
	E - Register	[E,R]	1rr10111	1	0
*	F - Register	[F,R]	1rr11010	1	0
	D - Register	[D,R]	1rr11011	4	0
*	W - Register	[W,R]	1rr11110	4	0
Auto Increment and decrement of R	Increment 2	[,R++]	1rr10001	6	0
	Decrement 2	[,--R]	1rr10011	6	0
Constant offset from PC (Twos complement offset)	8 bit offset	[n,PC]	1xx11100	4	1
	16 bit offset	[n,PC]	1xx11101	8	2
Extended indirect	16 bit address	[n]	10011111	5	2
*Relative to W	No Offset	[,W]	10010000	0	0
*(Twos complement offset)	16 bit offset	[n,W]	10110000	5	2
* AutoIncrement W	Increment 2	[,W++]	11010000	3	0
* AutoDecrement W	Decrement 2	[,--W]	11110000	3	0

rr = X, Y, U or S X = 00 Y = 01
xx = Doesn't care U = 10 S = 11

+ and + ~ # indicates the additional number of cycles and bytes for the particular variation

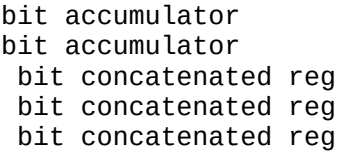
Register Descriptions

```
 X  - 16 bit index register
 Y  - 16 bit index register
 U  - 16 bit user-stack pointer
 S  - 16 bit system-stack pointer
 PC - 16 bit program counter register
*V  - 16 bit variable register (inter-register instructions only)
*0  - 8/16 bit zero register   (inter-register instructions only)
```

```
 A  - 8 bit accumulator
 B  - 8 bit accumulator
*E  - 8 bit accumulator
*F  - 8 bit accumulator
 D  - 16 bit concatenated reg.(A B)
*W  - 16 bit concatenated reg.(E F)
*Q  - 32 bit concatenated reg.(D W)
```

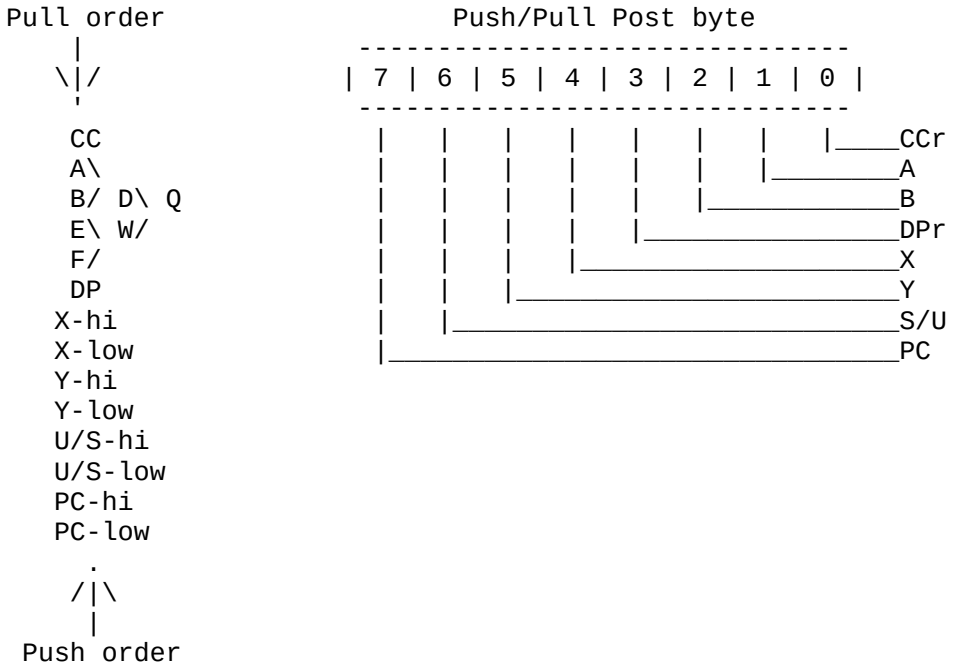
```
*MD - 8 bit mode/error register
 CC - 8 bit condition code register
 DP - 8 bit direct page register
```

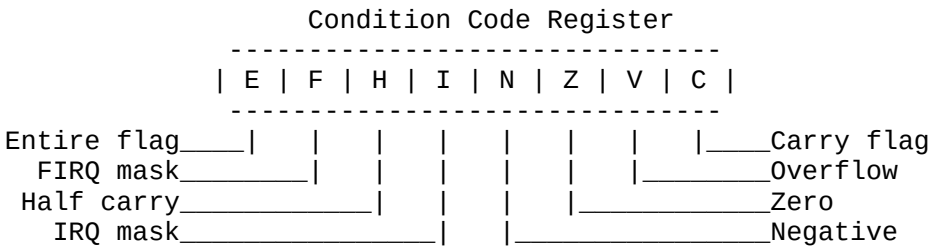
Accumulator structure map:



* Indicates new registers in 6309 CPU.

Push/Pull Order of Stack





The PSH(s,u) and PUL(s,u) instructions require one additional cycle for each byte pushed or pulled.

Alan DeKok's addition to the above...

The new features of the 6309 are closely related to the changes in design from the 6809. The 6309 is micro-coded, which allowed the designers to easily add new instructions and registers. It also has a one byte pre-fetch 'cache', which enables the 6309 to execute instructions like 'lsld' (2-bytes) in one clock cycle. The design of the 6809 series allows them to read one byte per clock cycle MAXIMUM, but there is a catch. Most instructions take more clock cycles to execute than bytes they contain. While the 6309 is performing internal calculations, the 'cache' hardware goes and reads the next instruction byte, leaving only one additional byte to be read to execute the 'lsld'. Reading this byte requires one clock cycle, and then the instruction is executed while the CPU fetches the next instruction.

The 6309 has a true 16-bit internal design.
e.g. the EXG instruction operates as

```
6809: read op-code
      read inter-register byte (r0,r1)
        r0_high -> temp_high
        r0_low  -> temp_low
        r1_high -> r0_high
        r1_low  -> r0_low
        r0_high -> r1_high
        r0_low  -> r1_low
```

8 actions, 8 clock cycles.

```
6809: read op-code
      read inter-register byte (r0,r1)
        r0 -> temp
        r1 -> r0
        r0 -> r1
```

5 actions, 5 clock cycles.

The 6309 native mode instruction execution clock lengths can be mostly accounted for by accounting for the pre-fetch cache and the internal 16-bit ALU.

TFM has some caveats. TFM r1-,r2- should NOT be used to setup the stack, as it's a POST-decrement instruction, not PRE-decrement.

Watch out for TFM r1,r2+ if you're reading from a peripherial. Why? The TFM uses the 1-byte 'cache' as an internal buffer for the byte that it's currently moving. The TFM instruction is interruptible (the only instruction that is), and code execution during the interrupt will destroy the byte in the cache.

On returning from the interrupt, the TFM will read the FROM address again to get the lost byte, which may be the wrong one. The visible effect of this is that block moves sometimes have a byte missing from the middle, and everything after that byte shifted down one address.

There are a few ways of checking of you're running on a 6309 or a 6809, these include:

```
:
 tfr 0,d  -> illegal registers are '$FFFF' on a 6809, $0000 on a 6309
 tstb     ->
 beq  Is6309
:
```

```
:
 ldb #$ff
 clrd      -> executes as a $10 (ignored) $4F (clra) on a 6809
 tstb
 beq  Is6309
:
```

It's a bit harder to check if the system is running in native mode or not. Most of the time it won't be necessary, but the only realy method is to do:

```
:
 pshs cc,d,dp,x,y,u  SAVE ALL REGISTERS AS CHECKING WILL TRASH THEM
 leax Is6309,pc
 pshs  x               save address of 6309 flag code
 leax Is6809,pc
 pshs  x               save address of 6809 flag code
 pshs  cc,d,dp,x,y,u   save registers
 orcc #ENTIRE          set to ALL registers on-stack
 rti                   go to 6309/6809 code

Is6309 clr <Flag   it's a 6309
 bra Continue

Is6809 leas 2,s   account for 6309 PC
  lda #$FF
  sta <Flag

Continue puls cc,d,dp,x,y,u  restore all registers
[etc...]
:
```

Note that the checks for both 6809/6309 and native/emulation will execute perfectly on both 6809 and 6309 systems, and will give the correct results in all cases.

In order to check for 6309 FIRQ operation (i.e. all registers saved), you'd have to do something like

```
[ enable FIRQ's ]
:
 leau -3,s       where stack will be if only CC and PC are saved
 stu <test       remember the pointer
loop tst <check  FIRQ happened yet?
  bne loop       no, wait for an FIRQ
[...]

FIRQ cmps <test  only CC, PC saved?
 bne  Is6309F    no, it's 6309 FIRQ mode
 clr <F.Flag     set to 6809 IRQ mode
 bra continue
```

```
Is6309F lda #$FF don't bother saving A as 6309 FIRQ mode already saves it
 sta <F.Flag     set the FIRQ flag

continue clr <check         we've done an FIRQ, so we can exit
 rti
:
```

The W,E, and F registers do not have the full immediate addressing mode capabilities that D,A, and B do. SBC, AND, BIT, EOR, ADC, OR with E,F,W are available only in register-register mode. LSR, ROR, ROL are available for W but not for E,F. ASR, ASL, LSL, NEG do not exist at all for W,E,F.

ASL can sort of be implemented by doing a ADDR R1,R1. (see later)

You can also do things like 'leax u,x' by doing a ADDR u,x.

Sadly, many of the new 6309 instructions are useless in everyday life. The bit manipulation instructions are interesting, but slow and mostly of limited value. Same with much of the DIV/MUL higher math. The AIM, etc. are very useful, though.

Programmer recommendations

Try to stay away from using the W register. It's got another pre-byte (like instructions using 'Y' or 'S'), and is correspondingly larger and slower. E and F are best used mainly instead of pushing loop counters onto the stack when you're running out of registers.

The V register is mostly pointless. If you're doing context switches, it isn't saved across interrupts unless you do so manually. Shuffling data back and forth between other registers and V is a lot of trouble. Any math, etc. involving V is generally done much faster using a real register. After going through 1meg+ of 6309 assembly code which is everything from an OS kernel to serial drivers to graphics drivers, I've never seen a use for the V register.

Of course, you could put '$FFFF' into V, and have registers for reg-reg addressing modes with bits all zero (0), and another with bits all 1 (V).

Pseudo-nops: tfr 0,0; exg 0,0

Extremely small software timing loops with large delays may be generated by performing a 'LDW',and then 'TFM r0,r0+'.

Many programs can be executed in 6309 native mode by patching only the IRQ code, if it accesses the stack. A 'LDMD #$01' may be performed as soon as your program starts executing, and will see an immediate 10-15% speed increase. Software timing loops must be checked!

Opcodes Hitachi left out of the 6309: and some round-about equivalents

```
E/F/W
-----
ADCr:  ADCR 0,r
ANDr:  ; ANDR V,r
ASLr/LSLr:  ADDR r,r
ASRr
BITr
EORr
NEGr:  COMr INCr
ORr
SBCr:  SBCR Z,r


E/F
---
LSRr
ROLr: ADCR r,r
RORr

Q (Long word =W1:W0)
--------------------
ADDQ: ADDW W0; ADCD W1
SUBQ: SUBW W0; SBCD W1
ASLQ: ASLW   ; ROLD
ROLQ: ROLW   ; ROLD
LSRQ: LSRD   ; RORW
RORQ: RORD   ; RORW
ASRQ: ASRD   ; RORW
COMQ: COMD   ; COMW
NEGQ: COMD   ; COMW ; SBCR 0,D
```