M68HC16 Family
CPU16
Reference Manual

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. MOTOROLA and the Mororola logo are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

TABLE OF CONTENTS

Paragraph **Title** **Page**

SECTION 1OVERVIEW

SECTION 2NOTATION

SECTION 3 SYSTEM RESOURCES

TABLE OF CONTENTS

(Continued)

SECTION 4 DATA TYPES AND ADDRESSING MODES

SECTION 5 INSTRUCTION SET

TABLE OF CONTENTS

(Continued)

SECTION 6 INSTRUCTION GLOSSARY

SECTION 7 INSTRUCTION PROCESS

TABLE OF CONTENTS

(Continued)

SECTION 8 INSTRUCTION TIMING

SECTION 9 EXCEPTION PROCESSING

TABLE OF CONTENTS

(Continued)

SECTION 10 DEVELOPMENT SUPPORT

TABLE OF CONTENTS

(Continued)

SECTION 11 DIGITAL SIGNAL PROCESSING

APPENDIX A COMPARISON OF CPU16/M68HC11 CPU ASSEMBLY LANGUAGE

TABLE OF CONTENTS

(Continued)

TABLE OF CONTENTS

(Continued)

LIST OF ILLUSTRATIONS

Figure Title Page

LIST OF ILLUSTRATIONS
(Continued)

Figure **Title** **Page**

LIST OF TABLES

LIST OF TABLES
(Continued)

LIST OF TABLES
(Continued)

LIST OF TABLES
(Continued)

SECTION 1OVERVIEW

The CPU16 is a high-speed 16-bit central processing unit used in the M68HC16 family of modular microcontrollers. The CPU16 uses a prefetch mechanism and a three-instruction pipeline to reduce instruction execution time. The CPU16 instruction set has been optimized for high performance and high-level language support. Program diagnosis is enhanced by a background debugging mode.

The CPU16 has two 16-bit general-purpose accumulators and three 16-bit index registers. It supports 8-bit (byte), 16-bit (word), and 32-bit (long-word) load and store operations, as well as 16-bit and 32-bit signed fractional operations.

CPU16 memory space includes a 1 Mbyte data space and a 1 Mbyte program space. Twenty-bit addressing and transparent bank switching are used to implement extended memory. In addition, most instructions automatically handle bank boundaries.

The CPU16 provides M68HC11 users a migration path to higher performance. CPU16 architecture is a superset of M68HC11 CPU architecture — all M68HC11 CPU resources are available in the CPU16. The CPU16 and M68HC11 CPU instruction sets are source code compatible. M68HC11 CPU instructions are either directly implemented in the CPU16 instruction set, or have been replaced by equivalent instructions.

The CPU16 includes instructions and hardware to implement control-oriented digital signal processing functions with a minimum of interfacing. A multiply and accumulate unit provides the capability to multiply signed 16-bit fractional numbers and store the resulting 32-bit fixed point product in a 36-bit accumulator. Modulo addressing supports finite impulse response filters.

Documentation for the M68HC16 family follows the modular design concept. There is a comprehensive user's manual for each device in the product line, and a detailed reference manual for each of the individual on-chip modules.

SECTION 2NOTATION

The following notation, symbols, and conventions are used throughout the manual.

2.1 Register Notation

A	—	Accumulator A
AM	—	Accumulator M
B	—	Accumulator B
CCR	—	Condition code register
D	—	Accumulator D
E	—	Accumulator E
EK	—	Extended addressing extension field
IR	—	Multiply and accumulate multiplicand register
HR	—	Multiply and accumulate multiplier register
IX	—	Index register X
IY	—	Index register Y
IZ	—	Index register Z
K	—	Address extension register
PC	—	Program counter
PK	—	Program counter extension field
SK	—	Stack pointer extension field
SL	—	Multiply and accumulate sign latch
SP	—	Stack pointer
XK	—	Index register X extension field
YK	—	Index register Y extension field
ZK	—	Index register Z extension field
XMSK	—	Modulo addressing index register X mask
YMSK	—	Modulo addressing index register Y mask

2.2 Condition Code Register Bits

S — Stop disable control bit
MV — AM overflow indicator
H — Half carry indicator
EV — AM extended overflow indicator
N — Negative indicator
Z — Zero indicator
V — Two's complement overflow indicator
C — Carry/borrow indicator
IP — Interrupt priority field
SM — Saturation mode control bit
PK — Program counter extension field

2.3 Condition Code Register Activity

— — Bit not affected
Δ — Bit changes according to specified conditions
0 — Bit cleared
1 — Bit set

2.4 Condition Code Expressions

M — Memory location used in operation
R — Result of operation
S — Source data
X — Register used in operation

2.5 Memory Addressing

M — Address of one memory byte
M + 1 — Address of byte at M + $0001
M : M + 1 — Address of one memory word
$(...)_X$ — Contents of address pointed to by IX
$(...)_Y$ — Contents of address pointed to by IY
$(...)_Z$ — Contents of address pointed to by IZ

2.6 Addressing Modes

E, X	—	IX with E offset
E, Y	—	IY with E offset
E, Z	—	IZ with E offset
EXT	—	Extended
EXT20	—	20-bit extended
IMM8	—	8-bit immediate
IMM16	—	16-bit immediate
IND8, X	—	IX with unsigned 8-bit offset
IND8, Y	—	IY with unsigned 8-bit offset
IND8, Z	—	IZ with unsigned 8-bit offset
IND16, X	—	IX with signed 16-bit offset
IND16, Y	—	IY with signed 16-bit offset
IND16, Z	—	IZ with signed 16-bit offset
IND20, X	—	IX with signed 20-bit offset
IND20, Y	—	IY with signed 20-bit offset
IND20, Z	—	IZ with signed 20-bit offset
INH	—	Inherent
IXP	—	Post-modified indexed
REL8	—	8-bit relative
REL16	—	16-bit relative

2.7 Instruction Format

b	—	4-bit address extension
ii	—	8-bit immediate data sign-extended to 16 bits
jj	—	High-order byte of 16-bit immediate data
kk	—	Low-order byte of 16-bit immediate data
hh	—	High-order byte of 16-bit extended address
ll	—	Low-order byte of 16-bit extended address
gggg	—	16-bit signed offset
ff	—	8-bit unsigned offset
mm	—	8-bit mask
mmmm	—	16-bit mask
rr	—	8-bit unsigned relative offset
rrrr	—	16-bit signed relative offset
xo	—	MAC index register X offset
yo	—	MAC index register Y offset
z	—	4-bit zero extension

2.8 Symbols and Operators

+	—	Addition
-	—	Subtraction or negation (twos complement)
*	—	Multiplication
/	—	Division
>	—	Greater
<	—	Less
=	—	Equal
≥	—	Equal or greater
≤	—	Equal or less
≠	—	Not equal
•	—	AND
;	—	Inclusive OR (OR)
⊕	—	Exclusive OR (EOR)
$\overline{\text{NOT}}$	—	Complementation
:	—	Concatenation
⇒	—	Transferred
⇔	—	Exchanged
±	—	Sign bit; also used to show tolerance
«	—	Sign extension
%	—	Binary value
$	—	Hexadecimal value

2.9 Conventions

Logic level one is the voltage that corresponds to Boolean true (1) state.

Logic level zero is the voltage that corresponds to Boolean false (0) state.

Set refers specifically to establishing logic level one on a bit or bits.

Cleared refers specifically to establishing logic level zero on a bit or bits.

Asserted means that a signal is in active logic state. An active low signal changes from logic level one to logic level zero when asserted, and an active high signal changes from logic level zero to logic level one.

Negated means that an asserted signal changes logic state. An active low signal changes from logic level zero to logic level one when negated, and an active high signal changes from logic level one to logic level zero.

ADDR is the mnemonic for address bus. **DATA** is the mnemonic for data bus.

LSB means least significant bit or bits. **MSB** means most significant bit or bits. References to low and high bytes are spelled out.

LSW means least significant word or words. **MSW** means most significant word or words.

A specific mnemonic within a range is referred to by mnemonic and number. A35 is bit 35 of accumulator A; ADDR[7:0] form the low byte of the address bus. **A range of mnemonics** is referred to by mnemonic and the numbers that define the range. AM[35:30] are bits 35 to 30 of accumulator M; DATA[15:8] form the high byte of the data bus.

Parentheses are used to indicate the content of a register or memory location, rather than the register or memory location itself. (A) is the content of accumulator A. (M : M + 1) is the content of the word at address M.

SECTION 3 SYSTEM RESOURCES

This section provides information concerning CPU16 register organization, memory management, and bus interfacing. The CPU16 is a subcomponent of a modular microcontroller. Due to the diversity of modular microcontrollers, detailed information concerning interaction with other modules and external devices is contained in the microcontroller user's manual.

3.1 General

The CPU16 was designed to provide compatibility with the M68HC11 and to provide additional capabilities associated with 16- and 32-bit data sizes, 20-bit addressing, and digital signal processing. CPU16 registers are an integral part of the CPU and are not addressed as memory locations. The CPU16 register model contains all the resources of the M68HC11, plus additional resources.

The CPU16 treats all peripheral, I/O, and memory locations as parts of a pseudolinear 1 Megabyte address space. There are no special instructions for I/O that are separate from instructions for addressing memory. Address space is made up of 16 64-Kbyte banks. Specialized bank addressing techniques and support registers provide transparent access across bank boundaries.

The CPU16 interacts with external devices and with other modules within the microcontroller via a standardized bus and bus interface. There are bus protocols for memory and peripheral accesses, as well as for managing an hierarchy of interrupt priorities.

3.2 Register Model

Figure 3-1 shows the CPU16 register model. Registers are discussed in detail in the following paragraphs.

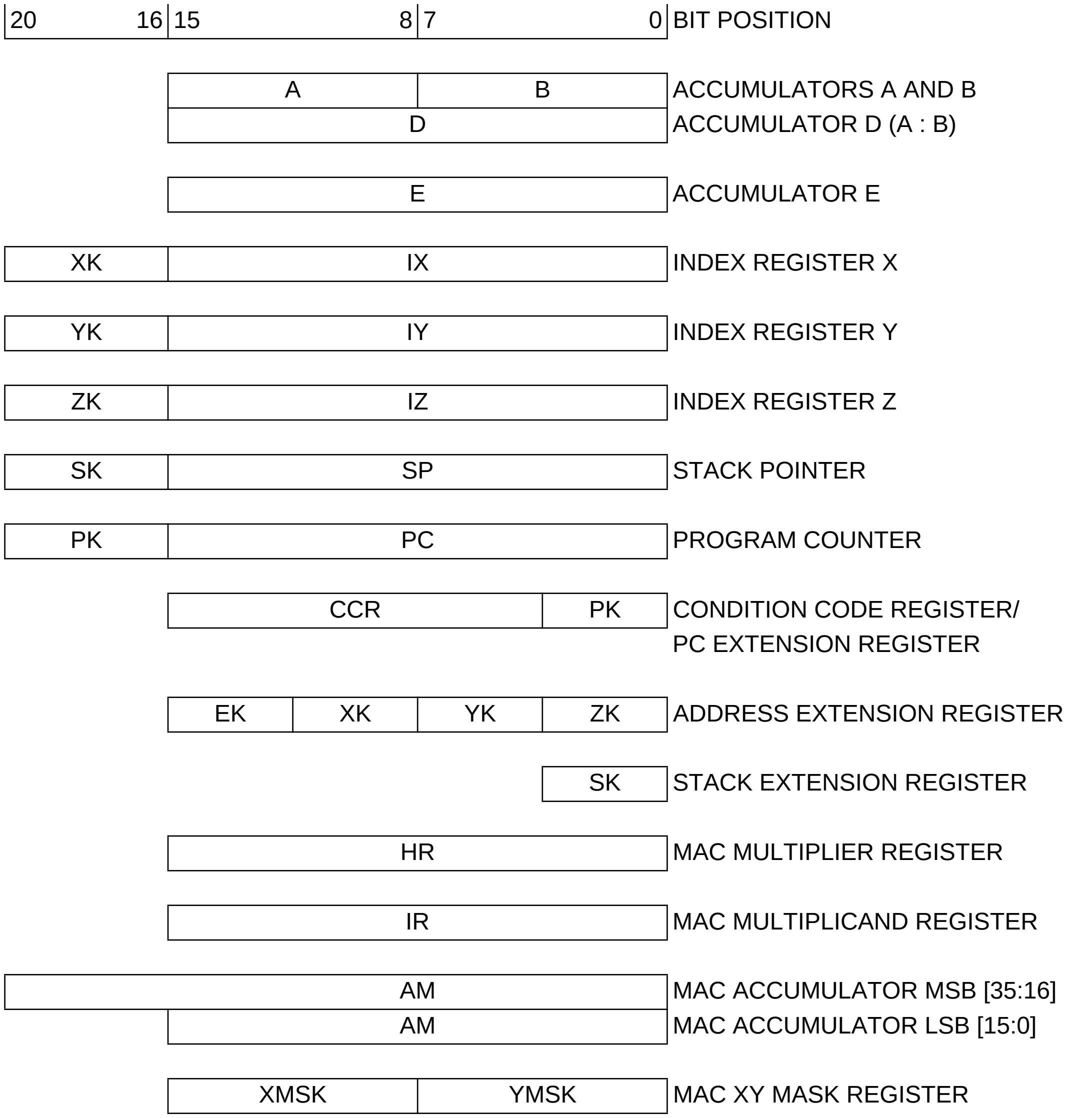


Figure 3-1 CPU16 Register Model

3.2.1 Accumulators

The CPU16 has two 8-bit accumulators (A and B) and one 16-bit accumulator (E). In addition, accumulators A and B can be concatenated into a second 16-bit “double” accumulator (D).

Accumulators A, B, and D are general-purpose registers used to hold operands and results during mathematic and data manipulation operations.

Accumulator E can be used in the same way as accumulator D, and also extends CPU16 capabilities. It allows more data to be held within the CPU16 during operations, simplifies 32-bit arithmetic and digital signal processing, and provides a practical 16-bit accumulator offset indexed addressing mode.

The CPU16 accumulators can perform the same operations as M68HC11 accumulators of the same names, but the CPU16 instruction set provides additional 8-bit, 16-bit, and 32-bit accumulator operations. See **SECTION 5 INSTRUCTION SET** for more information.

3.2.2 Index Registers

The CPU16 has three 16-bit index registers (IX, IY, and IZ). Each index register has an associated 4-bit extension field (XK, YK, and ZK).

Concatenated registers and extension fields provide 20-bit indexed addressing and support data structure functions anywhere in the CPU16 address space.

IX and IY can perform the same operations as M68HC11 registers of the same names, but the CPU16 instruction set provides additional indexed operations.

IZ can perform the same operations as IX and IY, and also provides an additional indexed addressing capability that replaces M68HC11 direct addressing mode. Initial IZ and ZK extension field values are included in the RESET exception vector, so that ZK : IZ can be used as a direct page pointer out of reset. See **SECTION 4 DATA TYPES AND ADDRESSING MODES** and **SECTION 9 EXCEPTION PROCESSING** for more information.

3.2.3 Stack Pointer

The CPU16 stack pointer (SP) is 16 bits wide. An associated 4-bit extension field (SK) provides 20-bit stack addressing.

Stack implementation in the CPU16 is from high to low memory. The stack grows downward as it is filled. SK : SP are decremented each time data is pushed on the stack, and incremented each time data is pulled from the stack.

SK : SP point to the next available stack address, rather than to the address of the latest stack entry. Although the stack pointer is normally incremented or decremented by word address, it is possible to push and pull byte-sized data; however, setting the stack pointer to an odd value causes misalignment, which affects performance. See **SECTION 4 DATA TYPES AND ADDRESSING MODES** and **SECTION 5 INSTRUCTION SET** for more information.

3.2.4 Program Counter

The CPU16 program counter (PC) is 16 bits wide. An associated 4-bit extension field (PK) provides 20-bit program addressing.

CPU16 instructions are fetched from even word boundaries. Bit 0 of the PC always has a value of zero, to assure that instruction fetches are made from word-aligned addresses. See **SECTION 7 INSTRUCTION PROCESS** for more information.

3.2.5 Condition Code Register

The 16-bit condition code register can be divided into two functional blocks. The eight MSB, which correspond to the CCR in the M68HC11, contain the low-power stop control bit and processor status flags. The eight LSB contain the interrupt priority field, the DSP saturation mode control bit, and the program counter address extension field.

Management of interrupt priority in the CPU16 differs considerably from that of the M68HC11. See **SECTION 9 EXCEPTION PROCESSING** for complete information.

Figure 3-2 shows the condition code register. Detailed descriptions of each status indicator and field in the register follow the figure.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			

Figure 3-2 Condition Code Register

S — STOP Enable
0 = Stop clock when LPSTOP instruction is executed
1 = Perform NOP when LPSTOP instruction is executed

MV — Accumulator M Overflow Flag
Set when overflow into AM35 has occurred.

H — Half Carry Flag
Set when a carry from bit 3 in A or B occurs during BCD addition.

EV — Extension Bit Overflow Flag
Set when an overflow into AM31 has occurred.

N — Negative Flag
Set when the MSB of a result register is set.

Z — Zero Flag
Set when all bits of a result register are zero.

V — Overflow Flag
Set when two’s complement overflow occurs as the result of an operation.

C — Carry Flag
Set when carry or borrow occurs during arithmetic operation. Also used during shift and rotate to facilitate multiple word operations.

IP[2:0] — Interrupt Priority Field
The priority value in this field (0 to 7) is used to mask interrupts.

SM — Saturate Mode Bit
When SM is set, if either EV or MV is set, data read from AM using TMER or TMET will be given maximum positive or negative value, depending on the state of the AM sign bit before overflow.

PK[3:0] — Program Counter Address Extension Field

This field is concatenated with the program counter to form a 20-bit address.

3.2.6 Address Extension Register and Address Extension Fields

There are six 4-bit address extension fields. EK, XK, YK, and ZK are contained by the address extension register, PK is part of the CCR, and SK stands alone.

Extension fields are the bank portions of 20-bit concatenated bank : byte addresses used in the CPU16 pseudolinear memory management scheme.

All extension fields except EK correspond directly to a register. XK, YK, and ZK extend registers IX, IY, and IZ; PK extends the PC; and SK extends the SP. EK holds the four MSB of the 20-bit address used by extended addressing mode.

The function of extension fields is discussed in **3.3 Memory Management**.

3.2.7 Multiply and Accumulate Registers

The multiply and accumulate (MAC) registers are part of a CPU submodule that performs repetitive signed fractional multiplication and stores the cumulative result. These operations are part of control-oriented digital signal processing.

There are four MAC registers. Register H contains the 16-bit signed fractional multiplier. Register I contains the 16-bit signed fractional multiplicand. Accumulator M is a specialized 36-bit product accumulation register. XMSK and YMSK contain 8-bit mask values used in modulo addressing.

The CPU16 has a special subset of signal processing instructions that manipulate the MAC registers and perform signal processing calculation. See **SECTION 5 INSTRUCTION SET** and **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

3.3 Memory Management

The CPU16 uses bank switching to provide a 1 Megabyte address space. There are 16 banks within the address space. Each bank is made up of 64 Kbytes addressed from $0000 to $FFFF. Banks are selected by means of address extension fields associated with individual CPU16 registers.

In addition, address space can be split into discrete 1 Megabyte program and data spaces by externally decoding the outputs described in **3.5.1.1 Function Codes**. When this technique is used, instruction fetches and RESET vector fetches access program space, while exception vector fetches (other than RESET), data accesses, and stack accesses are made in data space.

3.3.1 Address Extension

All CPU16 resources that are used to generate addresses are effectively 20 bits wide. These resources include extended index registers, program counter, and stack pointer. All addressing modes use 20-bit addresses.

Twenty-bit addresses are formed from a 16-bit byte address generated by an individual CPU16 register and a 4-bit bank address contained in an associated extension field. The byte address corresponds to ADDR[15:0] and the bank address corresponds to ADDR[19:16].

3.3.2 Extension Fields

The six address extension fields are each used in a different type of access. As shown in **3.2 Register Model**, all but EK are associated with particular CPU16 registers. There are a number of ways to manipulate extension fields and the address map.

3.3.2.1 Using Accumulator B to Modify Extension Fields

EK, XK, YK, ZK, and SK can be examined and modified by using the transfer extension field to B and transfer B to extension field instructions.

Transfer extension field to B instructions (TEKB, TXKB, TYKB, TZKB, and TSKB) copy the designated extension field into the four LSB of accumulator B, where it can be modified. Transfer B to extension field instructions (TBEK, TBXK, TBYK, TBZK, and TBSK) replace the designated extension field with the contents of the four LSB of accumulator B.

3.3.2.2 Using Stack Pointer Transfer to Modify Extension Fields

XK, YK, ZK, and SK can be modified by using the transfer index register to stack pointer and transfer stack pointer to index register instructions.

When the SP is transferred to (TSX, TSY, and TSZ) or from (TXS, TYS, and TZS) an index register, the corresponding address extension field is also transferred. Before the extension field is transferred, it is incremented or decremented if bank overflow occurred as a result of the instruction.

3.3.2.3 Using Index Register Exchange to Modify Extension Fields

XK, YK, and ZK can be modified by using the transfer index register to index register instructions.

When index registers are exchanged (TXY, TXZ, TYX, TYZ, TZX, and TZY), the corresponding address extension field is also exchanged.

3.3.2.4 Stacking Extension Field Values

The push multiple registers (PSHM) instruction can be used to store alternate extension field values on the stack. When bit 5 of the PSHM mask operand is set, the entire address extension register (EK, XK, YK, and ZK values) is pushed onto the stack.

The pull multiple registers (PULM) instruction can be used to replace extension field values. When bit 1 of the PULM mask operand is set, the entire address extension register (EK, XK, YK, and ZK) will be replaced with stacked values.

3.3.2.5 Adding Immediate Data to Registers

XK, YK, ZK, and SK are automatically modified when an AIX, AIY, AIZ, or AIS instruction causes an overflow from the corresponding register. The byte addresses contained in the registers have a range of $0000 to $FFFF. If the operation results in a value below $0000 or above $FFFF, the associated extension field is decremented or incremented by the amount of overflow.

3.3.3 Program Counter Address Extension

The PK field cannot be altered by direct transfer or exchange like other address extension fields, but a number of instructions and addressing modes affect the program counter and its associated extension field.

PK is automatically modified when an operation causes an overflow from the PC. The PC has a range of $0000 to $FFFF. If it is decremented below $0000 or incremented above $FFFF, PK is also incremented or decremented.

3.3.3.1 Effect of Jump Instructions on PK : PC

There are two forms of jump instruction in the CPU16 instruction set. Both use special addressing modes that replace PK : PC with a 20-bit effective address, but do not affect other address extension fields.

JMP causes an unconditional change in program execution. The effective address is placed in PK : PC and execution continues at the new address.

JSR causes a branch to a subroutine. After the contents of the program counter and the condition code register are stacked, the effective address is placed in PK : PC and execution continues at the new address.

See **SECTION 5 INSTRUCTION SET** for detailed information about jump instructions.

3.3.3.2 Effect of Branch Instructions on PK : PC

The CPU16 instruction set includes a number of branch instructions. All add an offset to the program counter when a branch is taken. The size of offset differs, but in all cases, PK is automatically modified when addition of the offset causes PC overflow. The PC has a range of $0000 to $FFFF. If it is decremented below $0000 or incremented above $FFFF, PK is also decremented or incremented. Pipelining affects the actual offset from the instruction. See **SECTION 5 INSTRUCTION SET** for detailed information about branch instructions.

3.3.4 Effective Addresses and Extension Fields

It is important to distinguish address extension field values from effective address values. Effective address calculation is a part of addressing mode operation. Indexed and accumulator offset addressing modes can generate effective addresses that cross bank boundaries — ADDR[19:16] are changed to make an access, but extension field values do not change as a result of the operation. See **SECTION 4 DATA TYPES AND ADDRESSING MODES** for more information. **Table 3-1** summarizes the effects of various operations on address lines and address extension fields.

Table 3-1 Operations that Cross Bank Boundaries

Type of Operation	Extension Field Used	Extension Field Affected	Effect on ADDR[19:16]
Normal PC Increments	PK	PK	Equals new PK
Operand Read Using Indexed Addressing Mode	XK, YK, ZK	None	Used for Effective Address
Operand Write Using Indexed Addressing Mode	XK, YK, ZK	None	Used for Effective Address
Operand Read Using Extended Addressing Mode	EK	None	Used for Effective Address
Operand Write Using Extended Addressing Mode	EK	None	Used for Effective Address
Post-modified Indexed Addressing (XK is modified after use as effective address)	XK	XK	Used for Effective Address
JMP, JSR Instruction	None	PK	Equals new PK
Branch Instructions (Including BSR and LBSR)	PK	PK	Equals new PK
Stack Access	SK	SK	Stack at new SK
AIX, AIY, AIZ, or AIS Instruction	XK, YK, ZK, or SK	XK, YK, ZK, or SK	None
TSX, TSY, or TSZ Instruction	SK	XK, YK, or ZK	None
TXS, TYS, or TZS Instruction	XK, YK, or ZK	SK	None
TXY or TXZ Instruction	XK	YK, ZK	None
TYX or TYZ Instruction	YK	XK, ZK	None
TZX or TZY Instruction	ZK	XK, YK	None

3.4 Intermodule Bus

The intermodule bus is a standardized bus developed to facilitate design of modular microcontrollers. Bus protocols are based on the MC68020 bus. The IMB contains circuitry to support exception processing, address space partitioning, multiple interrupt levels, and vectored interrupts.

Modular microcontroller family modules communicate with one another via the IMB. Although the full IMB supports 24 address and 16 data lines, CPU16 uses only 16 data lines and 20 address lines — ADDR[23:20] are tied to ADDR19 when processor driven.

3.5 External Bus Interface

The external bus interface (EBI) is contained in the system integration module of the modular microcontroller. This section provides a general discussion of EBI capabilities. Refer to the appropriate microcontroller user's manual for detailed information about the bus interface.

The external bus is essentially an extension of the IMB. There are 24 address lines and 16 data lines. ADDR[19:0] are normal address outputs, ADDR[23:20] follow the output state of ADDR19. It provides dynamic sizing between 8- and 16-bit data accesses. A three-line handshaking interface performs bus arbitration.

The EBI transfers information between the MCU and external devices. It supports byte, word, and long-word transfers. Data ports of 8 and 16 bits can be accessed through the use of asynchronous cycles controlled by the data transfer (SIZ1 and SIZ0) and data size acknowledge pins ($\overline{\text{DSACK1}}$ and$\overline{\text{DSACK0}}$). Multiple bus cycles may be required for an operand transfer to an 8-bit port, due to misalignment or to port width smaller than the operand size.

Port width is defined as the maximum number of bits accepted or provided during a bus transfer. External devices must follow the handshake protocol described below.

3.5.1 Bus Control Signals

Control signals indicate the beginning of the cycle, the address space and size of the transfer, and the type of cycle. The selected device controls the length of the cycle. Strobe signals, one for the address bus and another for the data bus, indicate the validity of an address and provide timing information for data. The EBI operates asynchronously for all port widths. A bus cycle is initiated by driving the address, size, function code, and read/write outputs.

3.5.1.1 Function Codes

Function codes are automatically generated by the CPU16. Since the CPU16 always operates in supervisor mode (FC2 = 1) FC1 and FC0 are encoded to select one of four address spaces. One encoding (%00) is reserved. The remaining three spaces are called program space, data space and CPU space. Program and data space are used for instruction and operand accesses. CPU space is used for control information not normally associated with read or write bus cycles, such as interrupt acknowledge cycles, breakpoint acknowledge cycles, and low power stop broadcast cycles. Function codes are valid while address strobe $\overline{\text{AS}}$ is asserted. The following table shows address space encoding.

Table 3-2 Address Space Encoding

FC2	FC1	FC0	Address Space
1	0	0	Reserved
1	0	1	Data Space
1	1	0	Program Space
1	1	1	CPU Space

3.5.1.2 Size Signals

SIZ0 and SIZ1 indicate the number of bytes remaining to be transferred during an operand cycle. They are valid while the $\overline{\text{AS}}$ is asserted. The following table shows SIZ0 and SIZ1 encoding.

Table 3-3 Size Signal Encoding

SIZ1	SIZ0	Transfer Size
0	1	Byte
1	0	Word
1	1	3 Byte
0	0	Long Word

3.5.1.3 Read/Write Signal

R/$\overline{\text{W}}$ determines the direction of the transfer during a bus cycle. This signal changes state, when required, at the beginning of a bus cycle, and is valid while $\overline{\text{AS}}$ is asserted. The signal may remain low for two consecutive write cycles.

3.5.2 Address Bus

Bus signals ADDR[19:0] define the address of the byte (or the most significant byte) to be transferred during a bus cycle. The MCU places the address on the bus at the beginning of a bus cycle. The address is valid while address strobe ($\overline{\text{AS}}$) is asserted.

$\overline{\text{AS}}$ is a timing signal that indicates the validity of an address on the address bus and of many control signals. It is asserted one-half clock after the beginning of a bus cycle.

3.5.3 Data Bus

Bus signals DATA[15:0] comprise a bidirectional, nonmultiplexed parallel bus that transfers data to or from the MCU. A read or write operation can transfer 8 or 16 bits of data in one bus cycle. During a read cycle, the data is latched by the MCU on the last falling edge of the clock for that bus cycle. For a write cycle, all 16 bits of the data bus are driven, regardless of the port width or operand size. The EBI places the data on the data bus one-half clock cycle after $\overline{\text{AS}}$ is asserted in a write cycle.

Data strobe ($\overline{\text{DS}}$) is a timing signal. For a read cycle, the MCU asserts $\overline{\text{DS}}$ to signal an external device to place data on the bus. $\overline{\text{DS}}$ is asserted at the same time as $\overline{\text{AS}}$ during a read cycle. For a write cycle, $\overline{\text{DS}}$ signals an external device that data on the bus is valid. The EBI asserts $\overline{\text{DS}}$ one full clock cycle after the assertion of $\overline{\text{AS}}$ during a write cycle.

3.5.4 Bus Cycle Termination Signals

During bus cycles, external devices assert the data transfer and size acknowledge signals ($\overline{\text{DSACK1}}$ and/or $\overline{\text{DSACK0}}$). During a read cycle, the signals tell the EBI to terminate the bus cycle and to latch data. During a write cycle, the signals indicate that an external device has successfully stored data and that the cycle may terminate. These signals also indicate to the EBI the size of the port for the bus cycle just completed.

The bus error signal ($\overline{\text{BERR}}$) is also a bus cycle termination indicator and can be used in the absence of $\overline{\text{DSACK}}$ to indicate a bus error condition. It can also be asserted in conjunction with $\overline{\text{DSACKx}}$ to indicate a bus error condition, provided it meets the appropriate timing requirements. Simultaneous assertion of $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ is treated in the same way as assertion of $\overline{\text{BERR}}$ alone.

An internal bus monitor can be used to generate the $\overline{\text{BERR}}$ signal for internal and internal-to-external transfers. An external bus master must provide its own $\overline{\text{BERR}}$ generation and drive the $\overline{\text{BERR}}$ pin, since the internal $\overline{\text{BERR}}$ monitor has no information about transfers initiated by an external bus master.

Finally, autovector signal ($\overline{\text{AVEC}}$) can be used to terminate external $\overline{\text{IRQ}}$ pin interrupt acknowledge cycles. $\overline{\text{AVEC}}$ indicates to the EBI that it must internally generate a vector number to locate an interrupt handler routine. If $\overline{\text{AVEC}}$ is continuously asserted, autovectors will be generated for all external interrupt requests. $\overline{\text{AVEC}}$ is ignored during all other bus cycles.

3.5.5 Data Transfer Mechanism

EBI architecture supports byte, word, and long-word operands, allowing access to 8- and 16-bit data ports through the use of asynchronous cycles controlled by the data transfer and size acknowledge inputs ($\overline{\text{DSACK1}}$and $\overline{\text{DSACK0}}$).

3.5.5.1 Dynamic Bus Sizing

The EBI dynamically interprets the port size of the addressed device during each bus cycle, allowing operand transfers to or from 8- and 16-bit ports. During an operand transfer cycle, the slave device signals its port size and indicates completion of the bus cycle to the EBI through the use of the $\overline{\text{DSACKx}}$ inputs, as shown in the following table.

Table 3-4 Effect of $\overline{\text{DSACK}}$ Signals

$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$	Result
1	1	Insert Wait States in Current Bus Cycle
1	0	Complete Cycle — Data Bus Port Size is 8 Bits
0	1	Complete Cycle — Data Bus Port Size is 16 Bits
0	0	Reserved

For example, if the CPU16 is executing an instruction that reads a long-word operand from a 16-bit port, the EBI latches the 16 bits of valid data and runs another bus cycle to obtain the other 16 bits. The operation for an 8-bit port is similar, but requires four read cycles. The addressed device uses the $\overline{\text{DSACK}}$ signals to indicate the port width. For instance, a 16-bit device always returns $\overline{\text{DSACK}}$ for a 16-bit port (regardless of whether the bus cycle is a byte or word operation).

Dynamic bus sizing requires that the portion of the data bus used for a transfer to or from a particular port size be fixed. A 16-bit port must reside on data bus bits [15:0], and an 8-bit port must reside on data bus bits [15:8]. This minimizes the number of bus cycles needed to transfer data and ensures that the EBI transfers valid data.

The EBI always attempts to transfer a maximum amount of data during each bus cycle. For a word operation, it is assumed that the port is 16 bits wide when the bus cycle begins. Operand bytes are designated as shown in **Figure 3-2**. OP0 is the most significant byte of a long-word operand, and OP3 is the least significant byte. The two bytes of a word-length operand are OP0 (most significant) and OP1. The single byte of a byte-length operand is OP0.

Operand	Byte Order 31 24	23 16	15 8	7 0
Long Word	OP0	OP1	OP2	OP3
Three Byte		OP0	OP1	OP2
Word			OP0	OP1
Byte				OP0

Figure 3-3 Operand Byte Order

3.5.5.2 Operand Alignment

Refer to **Table 3-5** for required organization of 8- and 16-bit data ports. A data multiplexer establishes the necessary connections for different combinations of address and data sizes. The multiplexer takes the two bytes of the 16-bit bus and routes them to their required positions. Positioning of bytes is determined by the size and address outputs. SIZ1 and SIZ0 indicate the remaining number of bytes to be transferred during the current bus cycle. The number of bytes transferred is equal to or less than the size indicated by SIZ1 and SIZ0, depending on port width.

ADDR0 also affects data multiplexer operation. During an operand transfer, ADDR[23:1] indicate the word base address of the portion of the operand to be accessed, and ADDR0 indicates the byte offset from the base. **Table 3-5** shows the number of bytes required on the data bus for read cycles. OPn entries are portions of the requested operand that are read or written during a bus cycle and are defined by SIZ1, SIZ0, and ADDR0 for that bus cycle.

Table 3-5 Operand Alignment

Transfer Case	SIZ1	SIZ0	ADDR0	$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$	DATA 15 8	DATA 7 0
Byte to Byte	0	1	X	1	0	OP0	(OP0)
Byte to Word (Even)	0	1	0	0	X	OP0	(OP0)
Byte to Word (Odd)	0	1	1	0	X	(OP0)	OP0
Word to Byte (Aligned)	1	0	0	1	0	OP0	(OP1)
Word to Byte (Misaligned)	1	0	1	1	0	OP0	(OP0)
Word to Word (Aligned)	1	0	0	0	X	OP0	OP1
Word to Word (Misaligned)	1	0	1	0	X	(OP0)	OP0
3 Byte to Byte (Aligned)†	1	1	0	1	0	OP0	(OP1)
3 Byte to Byte (Misaligned)†	1	1	1	1	0	OP0	(OP0)
3 Byte to Word (Aligned)†	1	1	0	0	X	OP0	OP1
3 Byte to Word (Misaligned)†	1	1	1	0	X	(OP0)	OP0
Long Word to Byte (Aligned)	0	0	0	1	0	OP0	(OP1)
Long Word to Byte (Misaligned)*	1	0	1	1	0	OP0	(OP0)
~~Long Word to Word (Aligned)~~	0	0	0	0	X	OP0	OP1
Long Word to Word (Misaligned)*	1	0	1	0	X	(OP0)	OP0

NOTES:

Operands in parentheses are ignored by the CPU16 during read cycles.

*The CPU16 treats misaligned long-word transfers as two misaligned word transfers.

†Three-byte transfer cases occur only as a result of a long word to byte transfer.

3.5.5.3 Misaligned Operands

The value of ADDR0 determines alignment. When ADDR0 = 0, the address is a word and byte boundary. When ADDR0 = 1, the address is a byte boundary only. A byte operand is properly aligned at any address; a word or long-word operand is misaligned at an odd address.

The basic CPU16 operand size is a 16-bit word. The CPU16 fetches instruction words and operands from word boundaries only. The CPU16 performs misaligned data word and long-word transfers. This capability is provided in order to make the CPU16 compatible with the M68HC11.

At most, a bus cycle can transfer a word of data aligned on a word boundary. If data words are misaligned, each byte of the misaligned word is treated as a separate word transfer. If a long-word operand is transferred via a 16-bit port, the most significant operand word is transferred on the first bus cycle and the least significant operand word on a following bus cycle.

SECTION 4 DATA TYPES AND ADDRESSING MODES

This section contains information about CPU16 data types and addressing modes. It is intended to familiarize users with basic processor capabilities.

4.1 Data Types

The CPU16 uses the following types of data:

- Bits
- 4-bit signed integers
- 8-bit (byte) signed and unsigned integers
- 8-bit, 2-digit binary coded decimal numbers
- 16-bit (word) signed and unsigned integers
- 32-bit (long word) signed and unsigned integers
- 16-bit signed fractions
- 32-bit signed fractions
- 36-bit signed fixed-point numbers
- 20-bit effective addresses
- There are 8 bits in a byte, 16 bits in a word. Bit set and clear instructions use both byte and word operands. Bit test instructions use byte operands.

Negative integers are represented in two's-complement form. Four-bit signed integers, packed two to a byte, are used only as X and Y offsets in MAC and RMAC operations. Integers of 32 bits are used only by extended multiply and divide instructions, and by the associated LDED and STED instructions.

Binary coded decimal numbers are packed, two digits per byte. BCD operations use byte operands.

16-bit fractions are used in both fractional multiplication and division, and as multiplicand and multiplier operands in the MAC unit. Bit 15 is the sign bit. An implied radix point lies between bits 15 and 14. There are 15 bits of magnitude — the range of values is –1 ($8000) to 1 – 2^{-15} ($7FFF).

Signed 32-bit fractions are used only by fractional multiplication and division instructions. Bit 31 is the sign bit. An implied radix point lies between bits 31 and 30. There are 31 bits of magnitude — the range of values is –1 ($80000000) to 1 – 2^{-31} ($7FFFFFFF).

Signed 36-bit fixed-point numbers are used only by the MAC unit. Bit 35 is the sign bit. Bits [34:31] are sign extension bits. There is an implied radix point between bits 31 and 30. There are 31 bits of magnitude, but use of the extension bits allows representation of numbers in the range –16 ($800000000) to 15.999999999 ($7FFFFFFFF).

20-bit effective addresses are formed by combining a 16-bit byte address with a 4-bit address extension. See **4.3 Addressing Modes** for more information.

4.2 Memory Organization

Both program and data memory are divided into sixteen 64-Kbyte banks. Addressing is pseudolinear — a 20-bit extended address can access any byte location in the appropriate address space.

A word is composed of two consecutive bytes. A word address is normally an even byte address. Byte 0 of a word has a lower 16-bit address than byte 1. Long words and 32-bit signed fractions consist of two consecutive words, and are normally accessed at the address of byte 0 in the word 0.

Instruction fetches always access word addresses. Word operands are normally accessed at even byte addresses, but may be accessed at odd byte addresses, with a substantial performance penalty.

To be compatible with the M68HC11, misaligned word transfers and misaligned stack accesses are allowed. Transferring a misaligned word requires two successive byte transfer operations.

Figure 4-1 shows how each CPU16 data type is organized in memory. Consecutive even addresses show size and alignment.

Memory/Register Data Types

Address	Type															
$0000	BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
$0002	BYTE0								BYTE1							
$0004	±	X OFFSET			±	Y OFFSET			±	X OFFSET			±	Y OFFSET		
$0006	BCD1				BCD0				BCD1				BCD0			
$0008	WORD 0															
$000A	WORD1															
$000C	MSW LONG WORD 0															
$000E	LSW LONG WORD 0															
$0010	MSW LONG WORD 1															
$0012	LSW LONG WORD 1															
$0014	±	⇐ (Radix Point) 16-BIT SIGNED FRACTION 0														
$0016	±	⇐ (Radix Point) 16-BIT SIGNED FRACTION 1														
$0018	±	⇐ (Radix Point) MSW 32-BIT SIGNED FRACTION 0														
$001A	LSW 32-BIT SIGNED FRACTION 0															0
$001C	±	⇐ (Radix Point) MSW 32-BIT SIGNED FRACTION 1														
$001E	LSW 32-BIT SIGNED FRACTION 1															0

MAC Data Types

35			32	31			16
±	«	«	«	«	⇐ (Radix Point)	MSW 32-BIT SIGNED FRACTION	
				15			0
				LSW 32-BIT SIGNED FRACTION			
				±	⇐ (Radix Point)	16-BIT SIGNED FRACTION	

Address Data Type

19	16	15	0
4-Bit Extension		16-Bit Address	

Figure 4-1 Data Types and Memory Organization

4.3 Addressing Modes

The CPU16 uses nine basic types of addressing. There are one or more addressing modes within each type. **Table 4-1** shows the addressing modes.

Table 4-1 Addressing Modes

Addressing Type	Mode Mnemonic	Description
Accumulator Offset	E, X	Index Register X with Accumulator E offset
	E, Y	Index Register Y with Accumulator E offset
	E, Z	Index Register Z with Accumulator E offset
Extended	EXT	Extended
	EXT20	20-bit Extended
Immediate	IMM8	8-bit Immediate
	IMM16	16-bit Immediate
Indexed 8-Bit	IND8, X	Index Register X with unsigned 8-bit offset
	IND8, Y	Index Register Y with unsigned 8-bit offset
	IND8, Z	Index Register Z with unsigned 8-bit offset
Indexed 16-Bit	IND16, X	Index Register X with signed 16-bit offset
	IND16, Y	Index Register Y with signed 16-bit offset
	IND16, Z	Index Register Z with signed 16-bit offset
Indexed 20-Bit	IND20, X	Index Register X with signed 20-bit offset
	IND20, Y	Index Register Y with signed 20-bit offset
	IND20, Z	Index Register Z with signed 20-bit offset
Inherent	INH	Inherent
Post-modified Index	IXP	Signed 8-bit offset added to Index Register X after effective address is used
Relative	REL8	8-bit relative
	REL16	16-bit relative

All modes generate ADDR[15:0]. This address is combined with ADDR[19:16] from an operand or an extension field to form a 20-bit effective address.

Note

Bank switching is transparent to most instructions. ADDR[19:16] of the effective address are changed to make an access across a page boundary. However, extension field values do not change as a result of effective address computation.

4.3.1 Immediate Addressing Modes

In the immediate modes, an argument is contained in a byte or word immediately following the instruction. For IMM8 and IMM16 modes, the effective address is the address of the argument.

There are three specialized forms of IMM8 addressing.

The AIS, AIX/Y/Z, ADDD and ADDE instructions decrease execution time by sign-extending the 8-bit immediate operand to 16 bits, then adding it to an appropriate register.

The MAC and RMAC instructions use an 8-bit immediate operand to specify two signed 4-bit index register offsets.

The PSHM and PULM instructions use an 8-bit immediate operand to indicate which registers must be pushed to or pulled from the stack.

4.3.2 Extended Addressing Modes

Regular extended mode instructions contain ADDR[15:0] in the word following the opcode. The effective address is formed by concatenating the EK field and the 16-bit byte address. EXT20 mode is used only by JMP and JSR instructions. JMP and JSR instructions contain a complete 20-bit effective address —the operand is zero-extended to 24 bits so that the instruction has an even number of bytes.

4.3.3 Indexed Addressing Modes

In the indexed modes, registers IX, IY, and IZ, together with their associated extension fields, are used to calculate the effective address.

For 8-bit indexed modes an 8-bit unsigned offset contained in the instruction is added to the value contained in an index register and its extension field.

For 16-bit modes, a 16-bit signed offset contained in the instruction is added to the value contained in an index register and its extension field.

For 20-bit modes, a 20-bit signed offset (zero-extended to 24 bits) is added to the value contained in an index register. These modes are used for JMP and JSR instructions only.

4.3.4 Inherent Addressing Mode

Inherent mode instructions use information directly available to the processor to determine the effective address. Operands (if any) are system resources and are thus not fetched from memory.

4.3.5 Accumulator Offset Addressing Mode

Accumulator offset modes form an effective address by sign-extending the content accumulator E to 20 bits, then adding the result to an index register and its associated extension field. This mode allows use of an index register and an accumulator within a loop without corrupting accumulator D.

4.3.6 Relative Addressing Modes

Relative modes are used for branch and long branch instructions. If a branch condition is satisfied, a byte or word signed twos complement offset is added to the concatenated PK field and program counter. The new PK : PC value is the effective address.

4.3.7 Post-Modified Index Addressing Mode

Post-modified index mode is used only by the MOVB and MOVW instructions. A signed 8-bit offset is added to index register X after the effective address formed by XK : IX is used. Post-modified mode provides enhanced block-move capabilities — programmers should carefully consider its effect on pointers.

4.3.8 Use of CPU16 Indexed Mode to Replace M68HC11 Direct Mode

In M68HC11 systems, the direct addressing mode can be used to perform rapid accesses to RAM or I/O mapped into bank 0 ($0000 to $00FF), but the CPU16 uses the first 512 bytes of bank 0 for exception vectors. To provide an enhanced replacement for direct mode, the ZK field and index register Z have been assigned reset initialization vectors — by resetting the ZK field to a chosen page, and using indexed mode addressing, a programmer can access useful data structures anywhere in the address map.

SECTION 5 INSTRUCTION SET

This section contains general information about the instruction set. It is organized into instruction summaries grouped by function. If an instruction has a special purpose, such as aiding indexed operations, it appears in the summary for that function, rather than in a general summary. An instruction that is used for more than one purpose appears in more than one summary. **SECTION 6 INSTRUCTION GLOSSARY** contains detailed information about individual instructions.

5.1 General

The instruction set is based upon that of the M68HC11, but the opcode map has been rearranged to maximize performance with a 16-bit data bus. Most M68HC11 instructions are supported by the CPU16, although they may be executed differently. Much M68HC11 code will run on the CPU16 following reassembly. The user must take into account changed instruction times, the interrupt mask, and the new interrupt stack frame. See **5.13 Comparison of CPU16 and M68HC11 Instruction Sets** for more information.

The CPU16 has a full range of 16-bit arithmetic and logic instructions, including signed and unsigned multiplication and division. A number of instructions support extended addressing and expanded memory space. In addition, there are special instructions related to digital signal processing.

5.2 Data Movement Instructions

The CPU16 has a complete set of 8- and 16-bit data movement instructions, as well as instructions to support 32-bit intermodule bus (IMB) operations. General-purpose load, store, transfer and move instructions facilitate movement of data to and from memory and peripherals. Special purpose instructions enhance indexing, extended addressing, stacking, and digital signal processing.

5.2.1 Load Instructions

Load instructions copy memory content into an accumulator or register. Memory content is not changed by the operation.

There are specialized load instructions for stacking, indexing, extended addressing, and digital signal processing. Refer to the appropriate summary for more information.

Table 5-1 Load Summary

Mnemonic	Function	Operation
LDAA	Load A	(M) ⇒ A
LDAB	Load B	(M) ⇒ B
LDD	Load D	(M : M + 1) ⇒ D
LDE	Load E	(M : M + 1) ⇒ E
LDED	Load Concatenated E and D	(M : M + 1) ⇒ E (M + 2 : M + 3) ⇒ D

5.2.2 Move Instructions

These instructions move data bytes or words from one location to another in memory.

Table 5-2 Move Summary

Mnemonic	Function	Operation
MOVB	Move Byte	$(M_1) \Rightarrow M_2$
MOVW	Move Word	$(M : M + 1_1) \Rightarrow M : M + 1_2$

5.2.3 Store Instructions

Store instructions copy the content of an accumulator or register to memory. Register/ accumulator content is not changed by the operation.

There are specialized store instructions for indexing, extended addressing, and CCR manipulation. Refer to the appropriate summary for more information.

Table 5-3 Store Summary

Mnemonic	Function	Operation
STAA	Store A	(A) ⇒ M
STAB	Store B	(B) ⇒ M
STD	Store D	(D) ⇒ M : M + 1
STE	Store E	(E) ⇒ M : M + 1
STED	Store Concatenated D and E	(E) ⇒ M : M + 1 (D) ⇒ M + 2 : M + 3

5.2.4 Transfer Instructions

These instructions transfer the content of a register or accumulator to another register or accumulator. Content of the source is not changed by the operation.

There are specialized transfer instructions for stacking, indexing, extended addressing, CCR manipulation, and digital signal processing. Refer to the appropriate summary for more information.

Table 5-4 Transfer Summary

Mnemonic	Function	Operation
TAB	Transfer A to B	(A) ⇒ B
TBA	Transfer B to A	(B) ⇒ A
TDE	Transfer D to E	(D)⇒ E
TED	Transfer E to D	(E) ⇒ D

5.2.5 Exchange Instructions

These instructions exchange the contents of pairs of registers or accumulators. There are specialized exchange instructions for indexing. Refer to the appropriate summary for more information.

Table 5-5 Exchange Summary

Mnemonic	Function	Operation
XGAB	Exchange A with B	(A) ⇔ (B)
XGDE	Exchange D with E	(D) ⇔ (E)

5.3 Mathematic Instructions

The CPU16 has a full set of 8- and 16-bit mathematic instructions. There are instructions for signed and unsigned arithmetic, division and multiplication, as well as a complete set of 8- and 16-bit Boolean operators.

Special arithmetic and logic instructions aid stacking operations, indexing, extended addressing, BCD calculation, and condition code register manipulation. There are also dedicated multiply and accumulate unit instructions. Refer to the appropriate instruction summary for more information.

5.3.1 Addition and Subtraction Instructions

Signed and unsigned 8- and 16-bit arithmetic instructions can be performed between registers or between registers and memory. Instructions that also add or subtract the value of the CCR carry bit facilitate multiple precision computation.

Table 5-6 Addition Summary

Mnemonic	Function	Operation
ABA	Add B to A	(A) + (B) ⇒ A
ADCA	Add with Carry to A	(A) + (M) + C ⇒ A
ADCB	Add with Carry to B	(B) + (M) + C ⇒ B
ADCD	Add with Carry to D	(D) + (M : M + 1) + C ⇒ D
ADCE	Add with Carry to E	(E) + (M : M + 1) + C ⇒ E
ADDA	Add to A	(A) + (M) ⇒ A
ADDB	Add to B	(B) + (M) ⇒ B
ADDD	Add to D	(D) + (M : M + 1) ⇒ D
ADDE	Add to E	(E) + (M : M + 1) ⇒ E
ADE	Add D to E	(E) + (D) ⇒ E

Table 5-7 Subtraction Summary

Mnemonic	Function	Operation
SBA	Subtract B from A	(A) – (B) ⇒ A
SBCA	Subtract with Carry from A	(A) – (M) – C ⇒ A
SBCB	Subtract with Carry from B	(B) – (M) – C ⇒ B
SBCD	Subtract with Carry from D	(D) – (M : M + 1) – C ⇒ D
SBCE	Subtract with Carry from E	(E) – (M : M + 1) – C ⇒ E
SDE	Subtract D from E	(E) – (D)⇒ E
SUBA	Subtract from A	(A) – (M) ⇒ A
SUBB	Subtract from B	(B) – (M) ⇒ B
SUBD	Subtract from D	(D) – (M : M + 1) ⇒ D
SUBE	Subtract from E	(E) – (M : M + 1) ⇒ E

The following table shows the type of arithmetic operation performed by each addition and subtraction instruction.

Table 5-8 Arithmetic Operations

Mnemonic	8-Bit	16-Bit	X ± X	X ± M	X ± M ± C
ABA	x		x		
ADCA	x				x
ADCB	x				x
ADCD		x			x
ADCE		x			x
ADDA	x			x	
ADDB	x			x	
ADDD		x		x	
ADDE		x		x	
ADE		x	x		
SBA	x		x		
SBCA	x				x
SBCB	x				x
SBCD		x			x
SBCE		x			x
SDE		x	x		
SUBA	x			x	
SUBB	x			x	
SUBD		x		x	
SUBE		x		x	

5.3.2 Binary Coded Decimal Instructions

To add binary coded decimal operands, use addition instructions that set the half-carry bit in the CCR, then adjust the result with the DAA instruction.

Table 5-9 BCD Summary

Mnemonic	Function	Operation
ABA	Add B to A	(A) + (B) ⇒ A
ADCA	Add with Carry to A	(A) + (M) + C ⇒ A
ADCB	Add with Carry to B	(B) + (M) + C ⇒ B
ADDA	Add to A	(A) + (M) ⇒ A
ADDB	Add to B	(B) + (M) ⇒ B
DAA	Decimal Adjust A	$(A)_{10}$
SXT	Sign Extend B into A	If B7 = 1 then A = $FF else A = $00

The following table shows DAA operation for all legal combinations of input operands. Columns 1 through 4 represent the results of addition operations on BCD operands. The correction factor in column 5 is added to the accumulator to restore the result of an operation on two BCD operands to a valid BCD value, and to set or clear the C bit. All values are hexadecimal.

Table 5-10 DAA Function Summary

1	2	3	4	5	6
Initial C Bit Value	Value of A[7:4]	Initial H Bit Value	Value of A[3:0]	Correction Factor	Corrected C Bit Value
0	0 – 9	0	0 – 9	00	0
0	0 – 8	0	A – F	06	0
0	0 – 9	1	0 – 3	06	0
0	A – F	0	0 – 9	60	1
0	9 – F	0	A – F	66	1
0	A – F	1	0 – 3	66	1
1	0 – 2	0	0 – 9	60	1
1	0 – 2	0	A – F	66	1
1	0 – 3	1	0 – 3	66	1

5.3.3 Compare and Test Instructions

Compare and test instructions perform subtraction between a pair of registers or between a register and memory. The result is not stored, but condition codes are set by the operation. These instructions are generally used to establish conditions for branch instructions.

Table 5-11 Compare and Test Summary

Mnemonic	Function	Operation
CBA	Compare A to B	(A) – (B)
CMPA	Compare A to Memory	(A) – (M)
CMPB	Compare B to Memory	(B) – (M)
CPD	Compare D to Memory	(D) – (M : M + 1)
CPE	Compare E to Memory	(E) – (M : M + 1)
TST	Test for Zero or Minus	(M) – $00
TSTA	Test A for Zero or Minus	(A) – $00
TSTB	Test B for Zero or Minus	(B) – $00
TSTD	Test D for Zero or Minus	(D) – $0000
TSTE	Test E for Zero or Minus	(E) – $0000
TSTW	Test for Zero or Minus Word	(M : M + 1) – $0000

5.3.4 Multiplication and Division Instructions

There are instructions for signed and unsigned 8- and 16-bit multiplication, as well as for signed 16-bit fractional multiplication. Eight-bit multiplication operations have a 16-bit product. Sixteen-bit multiplication operations can have either 16- or 32-bit products.

All division operations have 16-bit divisors, but dividends can be either 16- or 32-bit numbers. Quotients and remainders of all division operations are 16-bit numbers. There are instructions for signed and unsigned division, as well as for fractional division.

Fractional multiplication uses 16-bit operands. Bit 15 is the sign bit. There is an implied radix point between bits 15 and 14. The range of values is –1 ($8000) to 0.999969482 ($7FFF). The MSB of the result is its sign bit, and there is an implied radix point between the sign bit and the rest of the result.

There are special 36-bit signed fractional multiply and accumulate unit instructions to support digital signal processing operations. Refer to the appropriate summary for more information.

Table 5-12 Multiplication and Division Summary

Mnemonic	Function	Operation
EDIV	Extended Unsigned Divide	(E : D) / (IX) Quotient ⇒ IX Remainder ⇒ D
EDIVS	Extended Signed Divide	(E : D) / (IX) Quotient ⇒ IX Remainder ⇒ D
EMUL	Extended Unsigned Multiply	(E) ∗ (D) ⇒ E : D
EMULS	Extended Signed Multiply	(E) ∗ (D) ⇒ E : D
FDIV	Unsigned Fractional Divide	(D) / (IX) ⇒ IX remainder ⇒ D
FMULS	Signed Fractional Multiply	(E) ∗ (D) ⇒ E : D
IDIV	Integer Divide	(D) / (IX) ⇒ IX remainder ⇒ D
MUL	Multiply	(A) ∗ (B) ⇒ D

5.3.5 Decrement and Increment Instructions

These instructions are optimized 8- and 16-bit addition and subtraction operations. They are generally used to implement counters. Because they do not affect the carry bit in the CCR, they are particularly well suited for loop counters in multiple-precision computation routines.

Table 5-13 Decrement and Increment Summary

Mnemonic	Function	Operation
DEC	Decrement Memory	(M) – \$01 ⇒ M
DECA	Decrement A	(A) – \$01 ⇒ A
DECB	Decrement B	(B) – \$01 ⇒ B
DECW	Decrement Memory Word	(M : M + 1) – \$0001 ⇒ M : M + 1
INC	Increment Memory	(M) + \$01 ⇒ M
INCA	Increment A	(A) + \$01 ⇒ A
INCB	Increment B	(B) + \$01 ⇒ B
INCW	Increment Memory Word	(M : M + 1) + \$0001 ⇒ M : M + 1

5.3.6 Clear, Complement, and Negate Instructions

Each of these instructions performs a specific binary operation on a value in an accumulator or in memory. Clear operations set the value to zero, complement operations replace the value with its one's complement, and negate operations replace the value with its two's complement.

Table 5-14 Clear, Complement, and Negate Summary

Mnemonic	Function	Operation
CLR	Clear Memory	\$00 ⇒ M
CLRA	Clear A	\$00 ⇒ A
CLRB	Clear B	\$00 ⇒ B
CLRD	Clear D	\$0000 ⇒ D
CLRE	Clear E	\$0000 ⇒ E
CLRW	Clear Memory Word	\$0000 ⇒ M : M + 1
COM	One's Complement Byte	\$FF – (M) ⇒ M
COMA	One's Complement A	\$FF – (A) ⇒ A
COMB	One's Complement B	\$FF – (B) ⇒ B
COMD	One's Complement D	\$FFFF – (D) ⇒ D
COME	One's Complement E	\$FFFF – (E) ⇒ E
COMW	One's Complement Word	\$FFFF – M : M + 1 ⇒ M : M + 1
NEG	Two's Complement Byte	\$00 – (M) ⇒ M
NEGA	Two's Complement A	\$00 – (A) ⇒ A
NEGB	Two's Complement B	\$00 – (B) ⇒ B
NEGD	Two's Complement D	\$0000 – (D) ⇒ D
NEGE	Two's Complement E	\$0000 – (E) ⇒ E
NEGW	Two's Complement Word	\$0000 – (M : M + 1) ⇒ M : M + 1

5.3.7 Boolean Logic Instructions

Each of these instructions performs the Boolean logic operation represented by the mnemonic. There are 8- and 16-bit versions of each instruction.

There are special forms of logic instructions for stack pointer, program counter, index register, and address extension field manipulation. Refer to the appropriate summary for more information.

Table 5-15 Boolean Logic Summary

Mnemonic	Function	Operation
ANDA	AND A	(A) × (M) ⇒ A
ANDB	AND B	(B) × (M) ⇒ B
ANDD	AND D	(D) × (M : M + 1) ⇒ D
ANDE	AND E	(E) × (M : M + 1) ⇒ E
EORA	Exclusive OR A	(A) ⊕ (M) ⇒ A
EORB	Exclusive OR B	(B) ⊕ (M) ⇒ B
EORD	Exclusive OR D	(D) ⊕ (M : M + 1) ⇒ D
EORE	Exclusive OR E	(E) ⊕ (M : M + 1) ⇒ E
ORAA	OR A	(A) ✢ (M) ⇒ A
ORAB	OR B	(B) ✢ (M) ⇒ B
ORD	OR D	(D) ✢ (M : M + 1) ⇒ D
ORE	OR E	(E) ✢ (M : M + 1) ⇒ E

5.4 Bit Test and Manipulation Instructions

These operations use a mask value to test or change the value of individual bits in an accumulator or in memory. BITA and BITB provide a convenient means of setting condition codes without altering the value of either operand.

Table 5-16 Bit Test and Manipulation Summary

Mnemonic	Function	Operation
BITA	Bit Test A	(A) × (M)
BITB	Bit Test B	(B) × (M)
BCLR	Clear Bit(s)	(M) × ($\overline{\text{Mask}}$) ⇒ M
BCLRW	Clear Bit(s) Word	(M : M + 1) × ($\overline{\text{Mask}}$) ⇒ M : M + 1
BSET	Set Bit(s)	(M) ✢ (Mask) ⇒ M
BSETW	Set Bit(s) Word	(M : M + 1) ✢ (Mask) ⇒ M : M + 1

5.5 Shift and Rotate Instructions

There are shift and rotate commands for all accumulators, for memory bytes, and for memory words. All shift and rotate operations pass the shifted-out bit through the carry bit in the CCR in order to facilitate multiple-byte and multiple-word operations. There are no separate logical left shift operations. Use arithmetic shift left (ASL) for logic shift left (LSL) functions — LSL mnemonics will be assembled as ASL operations.

Special shift commands move multiply and accumulate unit accumulator bits. See **5.10 Digital Signal Processing Instructions** for more information.

Table 5-17 Logic Shift Summary

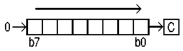
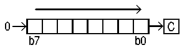
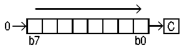
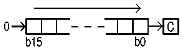
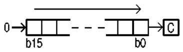
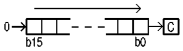
Mnemonic	Function	Operation
LSR	Logic Shift Right	
LSRA	Logic Shift Right A	
LSRB	Logic Shift Right B	
LSRD	Logic Shift Right D	
LSRE	Logic Shift Right E	
LSRW	Logic Shift Right Word	

Table 5-18 Arithmetic Shift Summary

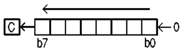
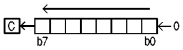
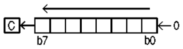
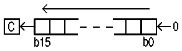
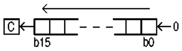
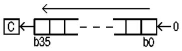
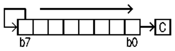
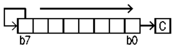
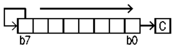
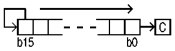
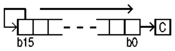
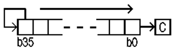
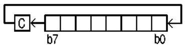
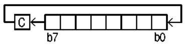
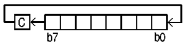
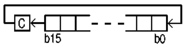
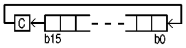
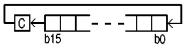
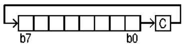
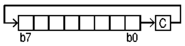
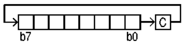
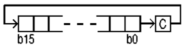
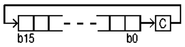
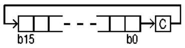
Mnemonic	Function	Operation
ASL (LSL)	Arithmetic Shift Left	
ASLA (LSLA)	Arithmetic Shift Left A	
ASLB (LSLB)	Arithmetic Shift Left B	
ASLD (LSLD)	Arithmetic Shift Left D	
ASLE (LSLE)	Arithmetic Shift Left E	
ASLW (LSLW)	Arithmetic Shift Left Word	
ASR	Arithmetic Shift Right	
ASRA	Arithmetic Shift Right A	
ASRB	Arithmetic Shift Right B	
ASRD	Arithmetic Shift Right D	
ASRE	Arithmetic Shift Right E	
ASRW	Arithmetic Shift Right Word	

Table 5-19 Rotate Summary

Mnemonic	Function	Operation
ROL	Rotate Left	
ROLA	Rotate Left A	
ROLB	Rotate Left B	
ROLD	Rotate Left D	
ROLE	Rotate Left E	
ROLW	Rotate Left Word	
ROR	Rotate Right	
RORA	Rotate Right A	
RORB	Rotate Right B	
RORD	Rotate Right D	
RORE	Rotate Right E	
RORW	Rotate Right Word	

5.6 Program Control Instructions

Program control instructions affect the sequence of instruction execution.

Branch instructions cause sequence to change when specific conditions exist. The CPU16 has short, long, and bit-condition branches.

Jump instructions cause immediate changes in sequence. The CPU16 has a true 20-bit address jump instruction.

Subroutine instructions optimize the process of temporarily transferring control to a segment of code that performs a particular task. The CPU16 can branch or jump to subroutines.

Interrupt instructions handle immediate transfer of control to a routine that performs a critical task. Software interrupts are a type of exception. **SECTION 9 EXCEPTION PROCESSING** covers interrupt exception processing in detail.

5.6.1 Short Branch Instructions

Short branch instructions operate as follows. When a specified condition is met, a signed 8-bit offset is added to the value in the program counter. If addition causes the value in the PC to be greater than $FFFF or less than $0000, the PK extension field is incremented or decremented. Program execution continues at the new extended address.

Short branch instructions can be classified by the type of condition that must be satisfied in order for a branch to be taken. Some instructions belong to more than one classification.

Unary branch instructions always execute.

Simple branches are taken when a specific bit in the condition code register is in a specific state as a result of a previous operation.

Unsigned conditional branches are taken when comparison or test of unsigned quantities results in a specific combination of condition code register bits.

Signed branches are taken when comparison or test of signed quantities results in a specific combination of condition code register bits.

Table 5-20 Short Branch Summary

Mnemonic	Opcode	Equation	Condition
BRA	B0	1 = 1	True
BRN	B1	1 = 0	False
Simple Branches			
Mnemonic	**Opcode**	**Equation**	**Condition**
BCC	B4	C = 0	Equation
BCS	B5	C = 1	Equation
BEQ	B7	Z = 1	Equation
BMI	BB	N = 1	Equation
BNE	B6	Z = 0	Equation
BPL	BA	N = 0	Equation
BVC	B8	V = 0	Equation
BVS	B9	V = 1	Equation
Unsigned Branches			
Mnemonic	**Opcode**	**Equation**	**Condition**
BCC	B4	C = 0	(X) ≥ (M)
BCS	B5	C = 1	(X) < (M)
BEQ	B7	Z = 1	(X) = (M)
BHI	B2	C ✛ Z = 0	(X) > (M)
BLS	B3	C ✛ Z = 1	(X) ≤ (M)
BNE	B6	Z = 0	(X) ≠ (M)

Table 5-20 Short Branch Summary (Continued)

Signed Branches			
Mnemonic	**Opcode**	**Equation**	**Condition**
BEQ	B7	$Z = 1$	$(X) = (M)$
BGE	BC	$N \oplus V = 0$	$(X) \geq (M)$
BGT	BE	$Z + (N \oplus V) = 0$	$(X) > (M)$
BLE	BF	$Z + (N \oplus V) = 1$	$(X) \leq (M)$
BLT	BD	$N \oplus V = 1$	$(X) < (M)$
BNE	B6	$Z = 0$	$(X) \neq (M)$

Note

The numeric range of short branch offset values is $80 (–128) to $7F (127), but actual displacement from the instruction differs from the range for two reasons.

First, PC values are automatically aligned to word boundaries. Only even offsets are valid — an odd offset value is rounded down. Maximum positive offset is $7E.

Second, instruction pipelining affects the value in the PC at the time an instruction executes. The value to which the offset is added is the address of the instruction plus $0006. At maximum positive offset ($7E), displacement from the branch instruction is 132. At maximum negative offset ($80), displacement is –122.

5.6.2 Long Branch Instructions

Long branch instructions operate as follows. When a specified condition is met, a signed 16-bit offset is added to the value in the program counter. If addition causes the value in the PC to be greater than $FFFF or less than $0000, the PK extension field is incremented or decremented. Program execution continues at the new extended address. Long branches are used when large displacements between decision-making steps are necessary.

Long branch instructions can be classified by the type of condition that must be satisfied in order for a branch to be taken. Some instructions belong to more than one classification.

Unary branch instructions always execute.

Simple branches are taken when a specific bit in the condition code register is in a specific state as a result of a previous operation.

Unsigned branches are taken when comparison or test of unsigned quantities results in a specific combination of condition code register bits.

Signed branches are taken when comparison or test of signed quantities results in a specific combination of condition code register bits.

Table 5-21 Long Branch Instructions

Unary Branches			
Mnemonic	**Opcode**	**Equation**	**Condition**
LBRA	3780	1 = 1	True
LBRN	3781	1 = 0	False
Simple Branches			
Mnemonic	**Opcode**	**Equation**	**Condition**
LBCC	3784	C = 0	Equation
LBCS	3785	C = 1	Equation
LBEQ	3787	Z = 1	Equation
LBEV	3791	EV = 1	Equation
LBMI	378B	N = 1	Equation
LBMV	3790	MV = 1	Equation
LBNE	3786	Z = 0	Equation
LBPL	378A	N = 0	Equation
LBVC	3788	V = 0	Equation
LBVS	3789	V = 1	Equation
Unsigned Branches			
Mnemonic	**Opcode**	**Equation**	**Condition**
LBCC	3784	C = 0	(X) ≥ (M)
LBCS	3785	C = 1	(X) < (M)
LBEQ	3787	Z = 1	(X) = (M)
LBHI	3782	C ✛ Z = 0	(X) > (M)
LBLS	3783	C ✛ Z = 1	(X) ≤ (M)
LBNE	3786	Z = 0	(X) ≠ (M)
Signed Branches			
Mnemonic	**Opcode**	**Equation**	**Condition**
LBEQ	3787	Z = 1	(X) = (M)
LBGE	378C	N ⊕ V = 0	(X) ≥ (M)
LBGT	378E	Z ✛ (N ⊕ V) = 0	(X) > (M)
LBLE	378F	Z ✛ (N ⊕ V) = 1	(X) ≤ (M)
LBLT	378D	N ⊕ V = 1	(X) < (M)
LBNE	3786	Z = 0	(X) ≠ (M)

Note

The numeric range of long branch offset values is $8000 (–32768) to $7FFF (32767), but actual displacement from the instruction differs from the range for two reasons.

First, PC values are automatically aligned to word boundaries. Only even offsets are valid — an odd offset value will be rounded down. Maximum positive offset is $7FFE.

Second, instruction pipelining affects the value in the PC at the time an instruction executes. The value to which the offset is added is the address of the instruction plus $0006. At maximum positive offset ($7FFE), displacement from the instruction is 32772. At maximum negative offset ($8000), displacement is –32762.

5.6.3 Bit Condition Branch Instructions

Bit condition branches are taken when specific bits in a memory byte are in a specific state. A mask operand is used to test a memory location pointed to by a 20-bit indexed or extended effective address. If the bits in memory match the mask, an 8- or 16-bit signed relative offset is added to the current value of the program counter. If addition causes the value in the PC to be greater than $FFFF or less than $0000, the PK extension field is incremented or decremented. Program execution continues at the new extended address.

Table 5-22 Bit Condition Branch Summary

Mnemonic	Addressing Mode	Opcode	Equation
BRCLR	IND8, X	CB	(M) • (Mask) = 0
	IND8, Y	DB	
	IND8, Z	EB	
	IND16, X	0A	
	IND16, Y	1A	
	IND16, Z	2A	
	EXT	3A	
BRSET	IND8, X	8B	$(\overline{M})$ • (Mask) = 0
	IND8, Y	9B	
	IND8, Z	AB	
	IND16, X	0B	
	IND16, Y	1B	
	IND16, Z	2B	
	EXT	3B	

Note

The numeric range of 8-bit offset values is $80 (–128) to $7F (127), and the numeric range of 16-bit offset values is $8000 (–32768) to $7FFF (32767), but actual displacement from the branch instruction differs from the range, for two reasons.

First, PC values are automatically aligned to word boundaries. Only even offsets are valid — an odd offset value is rounded down. Maximum positive 8-bit offset is $7E; maximum positive 16-bit offset is $7FFE.

Second, instruction pipelining affects the value in the PC at the time an instruction executes. The value to which the offset is added is the address of the instruction plus $0006. Maximum positive ($7E) and negative ($80) 8-bit offsets correspond to displacements of 132 and

–122 from the branch instruction. Maximum positive ($7FFE) and negative ($8000) 16-bit offsets correspond to displacements of 32772 and –32762.

5.6.4 Jump Instruction

The CPU16 JMP instruction uses 20-bit addressing, so that control can be passed to any address in the memory map. It should be noted that BRA and LBRA execute in fewer cycles than the indexed forms of JMP.

Table 5-23 Jump Summary

Mnemonic	Function	Operation
JMP	Jump	20-bit Address ⇒ PK : PC

5.6.5 Subroutine Instructions

Subroutines can be called by short (BSR) or long (LBSR) branches, or by a jump (JSR). A single instruction, RTS returns control to the calling routine.

All three types of calling instructions stack return PC and CCR values prior to transferring control to a subroutine. Stacking the CCR also saves the PK extension field. Other resources can be saved by means of the PSHM instruction, if necessary.

Table 5-24 Subroutine Summary

Mnemonic	Function	Operation
BSR	Branch to Subroutine	(PK : PC) – 2 ⇒ PK : PC Push (PC) (SK : SP) – 2 ⇒ SK : SP Push (CCR) (SK : SP) – 2 ⇒ SK : SP (PK : PC) + Offset ⇒ PK : PC
JSR	Jump to Subroutine	Push (PC) (SK : SP) – 2 ⇒ SK : SP Push (CCR) (SK : SP) – 2 ⇒ SK : SP 20-bit Address ⇒ PK : PC
LBSR	Long Branch to Subroutine	Push (PC) (SK : SP) – 2 ⇒ SK : SP Push (CCR) (SK : SP) – 2 ⇒ SK : SP (PK : PC) + Offset ⇒ PK : PC
RTS	Return from Subroutine	(SK : SP) + 2 ⇒ SK : SP Pull PK (SK : SP) + 2 ⇒ SK : SP Pull PC (PK : PC) – 2 ⇒ PK : PC

Note

Instruction pipelining affects the operation of BSR. When a subroutine is called, PK : PC contain the address of the calling instruction plus $0006. LBSR and JSR stack this value, but BSR must adjust it prior to stacking.

LBSR and JSR are 4-byte instructions. For program execution to resume at the instruction immediately following them, RTS must subtract $0002 from the stacked PK : PC value.

BSR is a 2-byte instruction. BSR subtracts $0002 from the stacked value prior to stacking so that RTS will work correctly.

5.6.6 Interrupt Instructions

The SWI instruction initiates synchronous exception processing. First, return PC and CCR values are stacked (stacking the CCR saves the PK extension field). After return values are stacked, the PK field is cleared, and the PC is loaded with exception vector 6 (content of address $000C).

The RTI instruction is used to terminate all exception handlers, including interrupt service routines. It causes normal execution to resume with the instruction following the last instruction that executed prior to interrupt. See **SECTION 9 EXCEPTION PROCESSING** for more information.

Table 5-25 Interrupt Summary

Mnemonic	Function	Operation
RTI	Return from Interrupt	(SK : SP) + 2 ⇒ SK : SP Pull CCR (SK : SP) + 2 ⇒ SK : SP Pull PC (PK : PC) – 6 ⇒ PK : PC
SWI	Software Interrupt	(PK : PC) + 2 ⇒ PK : PC Push (PC) (SK : SP) – 2 ⇒ SK : SP Push (CCR) (SK : SP) – 2 ⇒ SK : SP $0 ⇒ PK SWI Vector ⇒ PC

Note

Instruction pipelining affects the operation of SWI. When an interrupt occurs, PK : PC contain the address of the interrupted instruction plus $0006. This value is stacked during asynchronous exception processing, but synchronous exceptions, such as SWI, must adjust the stacked value so that RTI can work correctly.

For program execution to resume with the interrupted instruction following an asynchronous interrupt, RTI must subtract $0006 from the stacked PK : PC value.

Synchronous interrupts allow an interrupted instruction to finish execution before exception processing begins. The SWI instruction must add $0002 prior to stacking in order for execution to resume correctly.

5.7 Indexing and Address Extension Instructions

The CPU16 has a complete set of instructions that enable a user to take full advantage of 20-bit pseudolinear addressing. These instructions use specialized forms of mathematic and data transfer instructions to perform index register manipulation and extension field manipulation.

5.7.1 Indexing Instructions

Indexing instructions perform 8- and 16-bit operations on the three index registers and accumulators, other registers, or memory. Index addition and transfer instructions also affect the associated extension field.

Table 5-26 Indexing Summary

Addition Instructions		
Mnemonic	**Function**	**Operation**
ABX	Add B to IX	(XK : IX) + (000 : B) ⇒ XK : IX
ABY	Add B to IY	(YK : IY) + (000 : B) ⇒ YK : IY
ABZ	Add B to IZ	(ZK : Z) + (000 : B) ⇒ ZK : IZ
ADX	Add D to IX	(XK : IX) + (« D) ⇒ XK : IX
ADY	Add D to IY	(YK : IY) + (« D) ⇒ YK : IY
ADZ	Add D to IZ	(ZK : IZ) + (« D) ⇒ ZK : IZ
AEX	Add E to IX	(XK : IX) + (« D)⇒ XK : IX
AEY	Add E to IY	(YK : IY) + (« E) ⇒ YK : IY
AEZ	Add E to IZ	(ZK : IZ) + (« E) ⇒ ZK : IZ
AIX	Add Immediate Value to IX	XK : IX + (« IMM8/16) ⇒ XK : IX
AIY	Add Immediate Value to IY	YK : IY + (« IMM8/16) ⇒ YK : IY
AIZ	Add Immediate Value to IZ	ZK : IZ + (« IMM8/16) ⇒ ZK : IZ
Compare Instructions		
Mnemonic	**Function**	**Operation**
CPX	Compare IX to Memory	(IX) – (M : M + 1)
CPY	Compare IY to Memory	(IY) – (M : M + 1)
CPZ	Compare IZ to Memory	(IZ) – (M : M + 1)
Load Instructions		
Mnemonic	**Function**	**Operation**
LDX	Load IX	(M : M + 1) ⇒ IX
LDY	Load IY	(M : M + 1) ⇒ IY
LDZ	Load IZ	(M : M + 1) ⇒ IZ
Store Instructions		
Mnemonic	**Function**	**Operation**
STX	Store IX	(IX) ⇒ M : M + 1
STY	Store IY	(IY) ⇒ M : M + 1
STZ	Store IZ	(IZ) ⇒ M : M + 1

Table 5-26 Indexing Summary (Continued)

Transfer Instructions		
Mnemonic	**Function**	**Operation**
TSX	Transfer SP to IX	(SK : SP) + 2 ⇒ XK : IX
TSY	Transfer SP to IY	(SK : SP) + 2 ⇒ YK : IY
TSZ	Transfer SP to IZ	(SK : SP) + 2 ⇒ ZK : IZ
TXS	Transfer IX to SP	(XK : IX) – 2 ⇒ SK : SP
TXY	Transfer IX to IY	(XK : IX) ⇒ YK : IY
TXZ	Transfer IX to IZ	(XK : IX) ⇒ ZK : IZ
TYS	Transfer IY to SP	(YK : IY) – 2 ⇒ SK : SP
TYX	Transfer IY to IX	(YK : IY) ⇒ XK : IX
TYZ	Transfer IY to IZ	(YK : IY) ⇒ ZK : IZ
TZS	Transfer IZ to SP	(ZK : IZ) – 2 ⇒ SK : SP
TZX	Transfer IZ to IX	(ZK : IZ) ⇒ XK : IX
TZY	Transfer IZ to IY	(ZK : IZ) ⇒ ZK : IY
Exchange Instructions		
Mnemonic	**Function**	**Operation**
XGDX	Exchange D with IX	(D) ⇔ (IX)
XGDY	Exchange D with IY	(D) ⇔ (IY)
XGDZ	Exchange D with IZ	(D) ⇔ (IZ)
XGEX	Exchange E with IX	(E) ⇔ (IX)
XGEY	Exchange E with IY	(E) ⇔ (IY)
XGEZ	Exchange E with IZ	(E) ⇔ (IZ)

5.7.2 Address Extension Instructions

Address extension instructions transfer extension field contents to or from accumulator B. Other types of operations can be performed on the extension field value while it is in the accumulator.

Table 5-27 Address Extension Summary

Mnemonic	Function	Operation
TBEK	Transfer B to EK	(B) ⇒ EK
TBSK	Transfer B to SK	(B) ⇒ SK
TBXK	Transfer B to XK	(B) ⇒ XK
TBYK	Transfer B to YK	(B) ⇒ YK
TBZK	Transfer B to ZK	(B) ⇒ ZK
TEKB	Transfer EK to B	$0 ⇒ B[7:4] (EK) ⇒ B[3:0]
TSKB	Transfer SK to B	(SK) ⇒ B[3:0] $0 ⇒ B[7:4]
TXKB	Transfer XK to B	$0 ⇒ B[7:4] (XK) ⇒ B[3:0]
TYKB	Transfer YK to B	$0 ⇒ B[7:4] (YK) ⇒ B[3:0]
TZKB	Transfer ZK to B	$0 ⇒ B[7:4] (ZK) ⇒ B[3:0]

5.8 Stacking Instructions

There are two types of stacking instructions. Stack pointer instructions use specialized forms of mathematic and data transfer instructions to perform stack pointer manipulation. Stack operation instructions save information on and retrieve information from the system stack.

Table 5-28 Stacking Summary

Stack Pointer Instructions		
Mnemonic	**Function**	**Operation**
AIS	Add Immediate Data to SP	SK : SP + (« IMM16) ⇒ SK : SP
CPS	Compare SP to Memory	(SP) – (M : M + 1)
LDS	Load SP	(M : M + 1) ⇒ SP
STS	Store SP	(SP) ⇒ M : M + 1
TSX	Transfer SP to IX	(SK : SP) + 2 ⇒ XK : IX
TSY	Transfer SP to IY	(SK : SP) + 2 ⇒ YK : IY
TSZ	Transfer SP to IZ	(SK : SP) + 2 ⇒ ZK : IZ
TXS	Transfer IX to SP	(XK : IX) – 2 ⇒ SK : SP
TYS	Transfer IY to SP	(YK : IY) – 2 ⇒ SK : SP
TZS	Transfer IZ to SP	(ZK : IZ) – 2 ⇒ SK : SP
Stack Operation Instructions		
Mnemonic	**Function**	**Operation**
PSHA	Push A	(SK : SP) + 1 ⇒ SK : SP Push (A) (SK : SP) – 2 ⇒ SK : SP
PSHB	Push B	(SK : SP) + 1 ⇒ SK : SP Push (B) (SK : SP) – 2 ⇒ SK : SP
PSHM	Push Multiple Registers Mask bits: 0 = D 1 = E 2 = IX 3 = IY 4 = IZ 5 = K 6 = CCR 7 = (reserved)	For mask bits 0 to 6 : If mask bit set Push register (SK : SP) – 2 ⇒ SK : SP
PULA	Pull A	(SK : SP) + 2 ⇒ SK : SP Pull (A) (SK : SP) – 1 ⇒ SK : SP
PULB	Pull B	(SK : SP) + 2 ⇒ SK : SP Pull (B) (SK : SP) – 1 ⇒ SK : SP
PULM	Pull Multiple Registers Mask bits: 0 = CCR[15:4] 1 = K 2 = IZ 3 = IY 4 = IX 5 = E 6 = D 7 = (reserved)	For mask bits 0 to 7: If mask bit set (SK : SP) + 2 ⇒ SK : SP Pull register

5.9 Condition Code Instructions

Condition code instructions use specialized forms of mathematic and data transfer instructions to perform condition code register manipulation. Interrupts are not acknowledged until after the instruction following ANDP, ORP, TAP, and TDP has executed. Refer to **5.11 Stop and Wait Instructions** for more information.

Table 5-29 Condition Code Summary

Mnemonic	Function	Operation
ANDP	AND CCR	(CCR) ¥ IMM16 ⇒ CCR[15:4]
ORP	OR CCR	(CCR) ; IMM16 ⇒ CCR[15:4]
TAP	Transfer A to CCR	(A[7:0]) ⇒ CCR[15:8]
TDP	Transfer D to CCR	(D) ⇒ CCR[15:4]
TPA	Transfer CCR MSB to A	(CCR[15:8]) ⇒ A
TPD	Transfer CCR to D	(CCR) ⇒ D

5.10 Digital Signal Processing Instructions

DSP instructions use the CPU16 multiply and accumulate unit to implement digital filters and other signal processing functions. Other instructions, notably those that operate on concatenated E and D accumulators, are also used. See **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

Table 5-30 DSP Summary

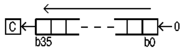
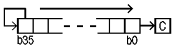
Mnemonic	Function	Operation
ACE	Add E to AM[31:15]	(AM[31:15]) + (E) ⇒ AM
ACED	Add concatenated E and D to AM	(E : D) + (AM) ⇒ AM
ASLM	Arithmetic Shift Left AM	
ASRM	Arithmetic Shift Right AM	
CLRM	Clear AM	$000000000 ⇒ AM[35:0]
LDHI	Initialize HR and IR	$(M : M + 1)_X \Rightarrow HR$ $(M : M + 1)_Y \Rightarrow IR$
MAC	Multiply and Accumulate Signed 16-Bit Fractions	(HR) ∗ (IR) ⇒ E : D (AM) + (E : D) ⇒ AM Qualified (IX) ⇒ IX Qualified (IY) ⇒ IY (HR) ⇒ IZ $(M : M + 1)_X \Rightarrow HR$ $(M : M + 1)_Y \Rightarrow IR$
PSHMAC	Push MAC State	MAC Registers ⇒ Stack
PULMAC	Pull MAC State	Stack ⇒ MAC Registers

Table 5-30 DSP Summary (Continued)

Mnemonic	Function	Operation
RMAC	Repeating Multiply and Accumulate Signed 16-Bit Fractions	Repeat until (E) < 0 (AM) + (H) ∗ (I) ⇒ AM Qualified (IX) ⇒ IX; Qualified (IY) ⇒ IY; $(M : M + 1)_X \Rightarrow H$; $(M : M + 1)_Y \Rightarrow I$ (E) – 1 ⇒ E
TDMSK	Transfer D to XMSK : YMSK	(D[15:8]) ⇒ X MASK (D[7:0]) ⇒ Y MASK
TEDM	Transfer E and D to AM[31:0] Sign Extend AM	(D) ⇒ AM[15:0] (E) ⇒ AM[31:16] AM[32:35] = AM31
TEM	Transfer E to AM[31:16] Sign Extend AM Clear AM LSB	(E) ⇒ AM[31:16] $00 ⇒ AM[15:0] AM[32:35] = AM31
TMER	Transfer AM to E Rounded	Rounded (AM) ⇒ Temp If (SM • (EV ; MV)) then Saturation ⇒ E else Temp[31:16] ⇒ E
TMET	Transfer AM to E Truncated	If (SM • (EV ; MV)) then Saturation ⇒ E else AM[31:16] ⇒ E
TMXED	Transfer AM to IX : E : D	AM[35:32] ⇒ IX[3:0] AM35 ⇒ IX[15:4] AM[31:16] ⇒ E AM[15:0] ⇒ D

5.11 Stop and Wait Instructions

There are two instructions that put the CPU16 in an inactive state. Both require that either an interrupt or a reset exception occurs before normal execution of instructions resumes. However, each operates differently.

LPSTOP minimizes microcontroller power consumption. The CPU16 initiates a stop, but it and other controller modules are deactivated by the microcontroller system integration module. Reactivation is also handled by the integration module. The interrupt priority field from the CPU16 condition code register is copied into the integration module external bus interface, then the system clock to the processor is stopped. When a reset or an interrupt of higher priority than the IP value occurs, the integration module activates the CPU16, and the appropriate exception processing sequence begins.

WAI idles the CPU16, but does not affect operation of other microcontroller modules. The IP field is not copied to the integration module. System clocks continue to run. The processor waits until a reset or an interrupt of higher priority than the IP value occurs, then begins the appropriate exception processing sequence.

Because the system integration module does not restart the CPU16, interrupts are acknowledged more quickly following WAI than following LPSTOP. See **SECTION 9 EXCEPTION PROCESSING** for more information.

To make certain that conditions for termination of LPSTOP and WAI are correct, interrupts are not recognized until after the instruction following ANDP, ORP, TAP, and TDP executes. This prevents interrupt exception processing during the period after the mask changes but before the following instruction executes.

Table 5-31 Stop and Wait Summary

Mnemonic	Function	Operation
LPSTOP	Low Power Stop	If $\overline{S}$ then STOP else NOP
WAI	Wait for Interrupt	WAIT

5.12 Background Mode and Null Operations

Background debug mode is a special CPU16 operating mode that is used for system development and debugging. Executing BGND when BDM is enabled puts the CPU16 in this mode. For complete information refer to **SECTION 10 DEVELOPMENT SUPPORT**.

Null operations are often used to replace other instructions during software debugging. Replacing conditional branch instructions with BRN, for instance, permits testing a decision-making routine without actually taking the branches.

Table 5-32 Background Mode and Null Operations

Mnemonic	Function	Operation

BGND	Enter Background Debugging Mode	If BDM enabled
		enter BDM;
		else, illegal instruction

BRN	Branch Never	If 1 = 0, branch
LBRN	Long Branch Never	If 1 = 0, branch
NOP	Null operation	—

5.13 Comparison of CPU16 and M68HC11 Instruction Sets

Most M68HC11 instructions are a source-code compatible subset of the CPU16 instruction set. However, certain M68HC11 instructions have been replaced by functionally equivalent CPU16 instructions, and some M68HC11 instructions operate differently in the CPU16. **APPENDIX A COMPARISON OF CPU16/M68HC11 CPU ASSEMBLY LANGUAGE** gives detailed information.

Table 5-33 shows M68HC11 instructions that have either been replaced by CPU16 instructions or that operate differently in the CPU16. Replacement instructions are not identical to M68HC11 instructions; M68HC11 code must be altered to establish proper preconditions.

All CPU16 instruction cycle counts and execution times differ from those of the M68HC11. **SECTION 6 INSTRUCTION GLOSSARY** gives information on instruction cycles. See **SECTION 8 INSTRUCTION TIMING** for information regarding calculation of instruction cycle times.

Table 5-33 CPU16 Implementation of M68HC11 Instructions

M68HC11 Instruction	M68HC16 Implementation
BHS	Replaced by BCC
BLO	Replaced by BCS
BSR	Generates a different stack frame
CLC	Replaced by ANDP
CLI	Replaced by ANDP
CLV	Replaced by ANDP
DES	Replaced by AIS
DEX	Replaced by AIX
DEY	Replaced by AIY
INS	Replaced by AIS
INX	Replaced by AIX
INY	Replaced by AIY
JMP	IND8 addressing modes replaced by IND20 and EXT modes
JSR	IND8 addressing modes replaced by IND20 and EXT modes Generates a different stack frame
LSL, LSLD	Use ASL instructions*
PSHX	Replaced by PSHM
PSHY	Replaced by PSHM
PULX	Replaced by PULM
PULY	Replaced by PULM
RTI	Reloads PC and CCR only
RTS	Uses two-word stack frame
SEC	Replaced by ORP
SEI	Replaced by ORP
SEV	Replaced by ORP
STOP	Replaced by LPSTOP
TAP	CPU16 CCR bits differ from M68HC11 CPU16 interrupt priority scheme differs from M68HC11
TPA	CPU16 CCR bits differ from M68HC11 CPU16 interrupt priority scheme differs from M68HC11
TSX	Adds two to SK : SP before transfer to XK : IX
TSY	Adds two to SK : SP before transfer to YK : IY
TXS	Subtracts two from XK : IX before transfer to SK : SP
TXY	Transfers XK field to YK field
TYS	Subtracts two from YK : IY before transfer to SK : SP
TYX	Transfers YK field to XK field
WAI	Waits indefinitely for interrupt or reset Generates a different stack frame

*Motorola assemblers will automatically translate LSL mnemonics

SECTION 6 INSTRUCTION GLOSSARY

The instruction glossary presents detailed information concerning each CPU16 instruction in concise form. **6.1 Assembler Syntax** shows standard assembler syntax formats. **6.2 Instructions** contains the glossary pages. **6.3 Condition Code Evaluation** lists Boolean expressions used to determine the effect of instructions on condition codes. **6.4 Instruction Set Summary** is a quick reference to the instruction set.

6.1 Assembler Syntax

Addressing mode determines standard assembler syntax. **Table 6-1** shows the standard formats. Bit set and clear instructions, bit condition branch instructions, jump instructions, multiply and accumulate instructions, move instructions and register stacking instructions have special syntax. Information on syntax is given on the appropriate glossary page. **APPENDIX B MOTOROLA ASSEMBLER SYNTAX** is a detailed syntax reference.

Table 6-1 Standard Assembler Formats

Addressing Mode	Instruction Mnemonic	E,Index Register Symbol
Extended	Instruction Mnemonic	Address Extension Operand
Immediate	Instruction Mnemonic	#Operand
Indexed	Instruction Mnemonic	Offset Operand,Index Register Symbol
Inherent	Instruction Mnemonic	
Relative	Instruction Mnemonic	Displacement

6.2 Instructions

Each instruction is listed alphabetically by mnemonic. Each listing contains complete information about instruction format, operation, and the effect an operation has on the condition code register.

The number of system clock cycles required to execute each instruction is also shown. Cycle counts are based on bus accesses that require two system clock cycles each, a 16-bit data bus, and aligned access. Cycle counts include system clock cycles required for prefetch, operand access, and internal operation. See **SECTION 8 INSTRUCTION TIMING** for more information.

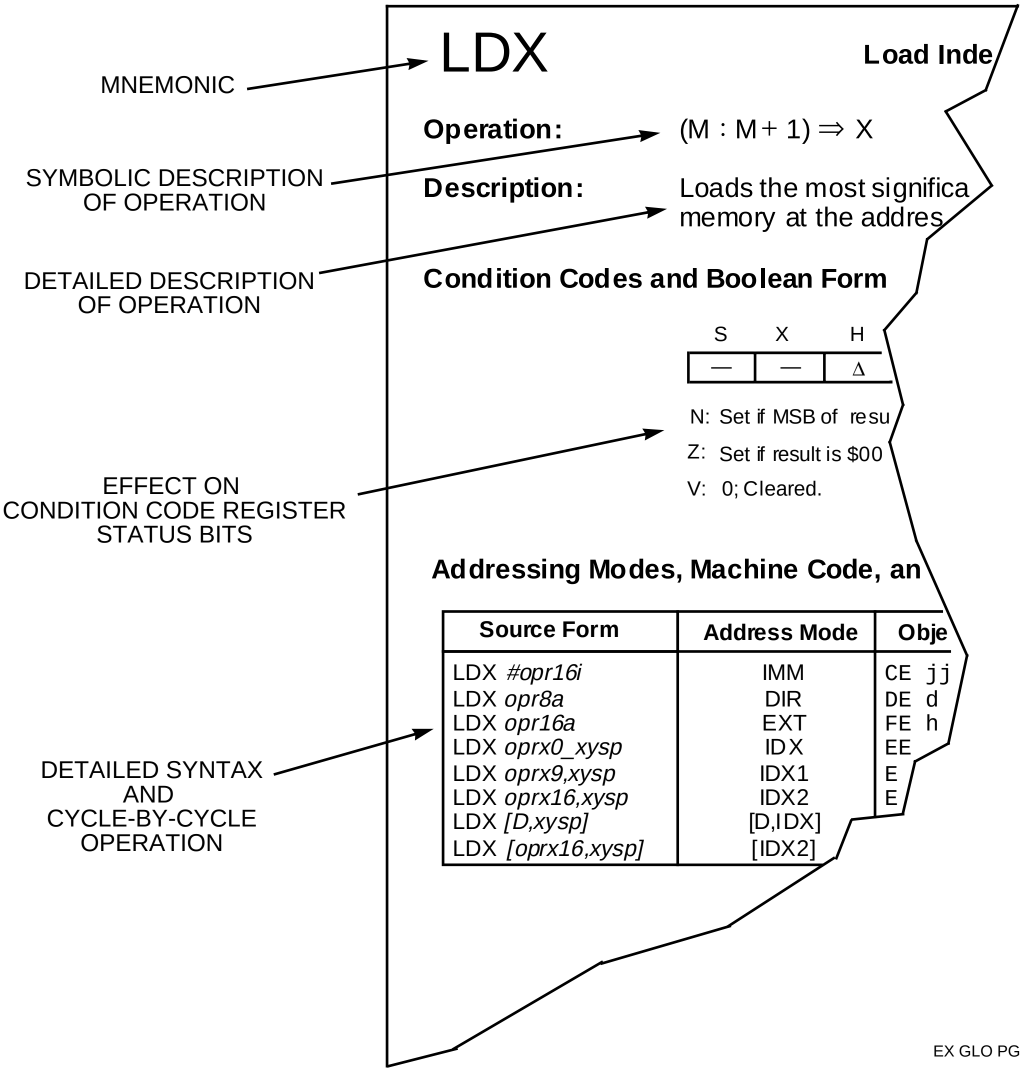


Figure 6-1 Typical Instruction Glossary Entry

ABA Add B to A ABA

Operation: $(A) + (B) \Rightarrow A$

Description: Adds the content of accumulator B to the content of accumulator A, then places the result in accumulator A. Content of accumulator B does not change. The ABA operation affects the CCR H bit, which makes it useful for BCD arithmetic (see DAA for more information).

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	5	4	3	0
S	MV	H	EV	N	Z	V	C	IP		SM	PK	
–	–	∆	–	∆	∆	∆	∆	–		–	–	

S: Not affected.
MV: Not affected.
H: Set if there is a carry from bit 3 during addition; else cleared.
EV: Not affected.
N: Set if A7 is set by operation; else cleared.
Z: Set if (A) = $00 as a result of operation; else cleared.
V: Set if two’s complement overflow occurs as a result of the operation; else cleared.
C: Set if there is a carry from A during operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	370B	—	2

ABX Add B to IX ABX

Operation: (XK : IX) + (000 : B) ⇒ XK : IX

Description: Adds the zero-extended content of accumulator B to the content of index register X, then places the result in index register X. Content of accumulator B does not change. If IX overflows as a result of the operation, the XK is incremented or decremented.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	–	–	–	–	–			–	–			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Not affected.
Z:	Not affected.
V:	Not affected.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	374F	—	2

ABY Add B to IY ABY

Operation: (YK : IY) + (000 : B) ⇒ YK : IY

Description: Adds the zero-extended content of accumulator B to the content of index register Y, then places the result in index register Y. Content of accumulator B does not change. If IY overflows as a result of the operation, the YK is incremented or decremented.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	375F	—	2

ABZ Add B to IZ ABZ

Operation: $(ZK : IZ) + (000 : B) \Rightarrow ZK : IZ$

Description: Adds the zero-extended content of accumulator B to the content of index register Z, then places the result in index register Z. Content of accumulator B does not change. If IZ overflows as a result of the operation, the ZK is incremented or decremented.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	376F	—	2

ACE Add E to AM ACE

Operation: (AM[31:16]) + (E) ⇒ AM

Description: Adds the content of accumulator E to bits 31 to 16 of accumulator M, then places the result in accumulator M. Bits 15 to 0 of accumulator M are not affected. The value in E is assumed to be a 16-bit signed fraction. See **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
–	Δ	–	Δ	–	–	–	–		–		–		–		

S:	Not affected.
MV:	Set if overflow into AM35 occurs during addition; else not affected.
H:	Not affected.
EV:	Set if overflow into AM[34:31] occurs during addition; else cleared.
N:	Not affected.
Z:	Not affected.
V:	Not affected.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3722	—	2

ACED Add E : D to AM ACED

Operation: (AM) + (E : D) ⇒ AM

Description: The concatenated contents of accumulators E and D are added to accumulator M. The value in the concatenated registers is assumed to be a 32-bit signed fraction. See **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7 5	4	3 0
S	MV	H	EV	N	Z	V	C	IP	SM	PK
–	Δ	–	Δ	–	–	–	–	–	–	–

S: Not affected.
MV: Set if overflow into AM35 occurs as a result of addition; else cleared.
H: Not affected.
EV: Set if overflow into AM[34:31] occurs as a result of addition; else cleared.
N: Not affected.
Z: Not affected.
V: Not affected.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3723	—	4

ADCA — Add with Carry to A — ADCA

Operation: (A) + (M) + C ⇒ A

Description: Adds the value of the CCR carry bit to the sum of the content of accumulator A and a memory byte, then places the result in accumulator A. Memory content is not affected. ADCA operation affects the CCR H bit, which makes it useful for BCD arithmetic.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	Δ	–	Δ	Δ	Δ	Δ	–			–	–			

S:	Not affected.
MV:	Not affected.
H:	Set if there is a carry from bit 3 during addition; else cleared.
EV:	Not affected.
N:	Set if A7 is set by operation; else cleared.
Z:	Set if (A) = $00 as a result of operation; else cleared.
V:	Set if two’s complement overflow occurs as a result of the operation; else cleared.
C:	Set if there is a carry from A during operation; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	43	ff	6
IND8, Y	53	ff	6
IND8, Z	63	ff	6
IMM8	73	ii	2
IND16, X	1743	gggg	6
IND16, Y	1753	gggg	6
IND16, Z	1763	gggg	6
EXT	1773	hhll	6
E, X	2743	—	6
E, Y	2753	—	6
E, Z	2763	—	6

ADCB Add with Carry to B ADCB

Operation: (B) + (M) + C ⇒ B

Description: Adds the value of the CCR carry bit to the sum of the content of accumulator B and a memory byte, then places the result in accumulator B. Memory content is not affected. ADCB operation affects the CCR H bit, which makes it useful for BCD arithmetic.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
–	–	∆	–	∆	∆	∆	∆		–		–		–		

S:	Not affected.
MV:	Not affected.
H:	Set if there is a carry from bit 3 during addition; else cleared.
EV:	Not affected.
N:	Set if B7 is set by operation; else cleared.
Z:	Set if B = $00 as a result of operation; else cleared.
V:	Set if two’s complement overflow occurs as a result of the operation; else cleared.
C:	Set if there is a carry from B during operation; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	C3	ff	6
IND8, Y	D3	ff	6
IND8, Z	E3	ff	6
IMM8	F3	ii	2
IND16, X	17C3	gggg	6
IND16, Y	17D3	gggg	6
IND16, Z	17E3	gggg	6
EXT	17F3	hhll	6
E, X	27C3	—	6
E, Y	27D3	—	6
E, Z	27E3	—	6

ADCD Add with Carry to D ADCD

Operation: (D) + (M : M + 1) + C ⇒ D

Description: Adds the value of the CCR carry bit to the sum of the content of accumulator D and a memory word, then places the result in accumulator D. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
–	–	–	–	∆	∆	∆	∆		–		–		–		

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if D15 is set by operation; else cleared.
Z:	Set if (D) = $0000 as a result of operation; else cleared.
V:	Set if two’s complement overflow occurs as a result of the operation; else cleared.
C:	Set if there is a carry from D during operation; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	83	ff	6
IND8, Y	93	ff	6
IND8, Z	A3	ff	6
IMM16	37B3	jjkk	4
IND16, X	37C3	gggg	6
IND16, Y	37D3	gggg	6
IND16, Z	37E3	gggg	6
EXT	37F3	hhll	6
E, X	2783	—	6
E, Y	2793	—	6
E, Z	27A3	—	6

ADCE Add with Carry to E ADCE

Operation: (E) + (M : M + 1) + C ⇒ E

Description: Adds the value of the CCR carry bit to the sum of the content of accumulator E and a memory word, then places the result in accumulator E. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
–	–	–	–	∆	∆	∆	∆		–		–		–		

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if E15 is set by operation; else cleared.
Z: Set if (E) = $0000 as a result of operation; else cleared.
V: Set if two’s complement overflow occurs as a result of the operation; else cleared.
C: Set if there is a carry from E during operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IMM16	3733	jjkk	4
IND16, X	3743	gggg	6
IND16, Y	3753	gggg	6
IND16, Z	3763	gggg	6
EXT	3773	hhll	6

ADDA Add to A ADDA

Operation: (A) + (M) ⇒ A

Description: Adds a memory byte to the content of accumulator A, then places the result in accumulator A. Memory content is not affected. ADDA affects the CCR H bit . It is used for BCD arithmetic.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	Δ	–	Δ	Δ	Δ	Δ	–			–	–			

S: Not affected.
MV: Not affected.
H: Set if operation requires a carry from A3; else cleared.
EV: Not affected.
N: Set if A7 is set by operation; else cleared.
Z: Set if (A) = $00 as a result of operation; else cleared.
V: Set if two’s complement overflow occurs as a result of the operation; else cleared.
C: Set if there is a carry from A during operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	41	ff	6
IND8, Y	51	ff	6
IND8, Z	61	ff	6
IMM8	71	ii	2
IND16, X	1741	gggg	6
IND16, Y	1751	gggg	6
IND16, Z	1761	gggg	6
EXT	1771	hhll	6
E, X	2741	—	6
E, Y	2751	—	6
E, Z	2761	—	6

ADDB Add to B ADDB

Operation: (B) + (M) ⇒ B

Description: Adds a memory byte to the content of accumulator B, then places the result in accumulator B. Memory content is not affected. ADDB affects the CCR H bit — it is used for BCD arithmetic.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
–	–	∆	–	∆	∆	∆	∆		–		–		–		

S: Not affected.
MV: Not affected.
H: Set if operation requires a carry from B3; else cleared.
EV: Not affected.
N: Set if B7 is set by operation; else cleared.
Z: Set if (B) = $00 as a result of operation; else cleared.
V: Set if two’s complement overflow occurs as a result of the operation; else cleared.
C: Set if there is a carry from B during operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	C1	ff	6
IND8, Y	D1	ff	6
IND8, Z	E1	ff	6
IMM8	F1	ii	2
IND16, X	17C1	gggg	6
IND16, Y	17D1	gggg	6
IND16, Z	17E1	gggg	6
EXT	17F1	hhll	6
E, X	27C1	—	6
E, Y	27D1	—	6
E, Z	27E1	—	6

ADDD Add to D ADDD

Operation: (D) + (M : M + 1) ⇒ D

Description: Adds a memory word to the content of accumulator D, then places the result in accumulator D. Memory content is not affected.

Syntax: Standard

Condition Code Register:

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>S</td><td>MV</td><td>H</td><td>EV</td><td>N</td><td>Z</td><td>V</td><td>C</td><td colspan="3">IP</td><td>SM</td><td colspan="4">PK</td></tr>
<tr><td>—</td><td>—</td><td>—</td><td>—</td><td>∆</td><td>∆</td><td>∆</td><td>∆</td><td colspan="3">—</td><td>—</td><td colspan="4">—</td></tr>
</table>

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if D15 is set by operation; else cleared.
Z: Set if (D) = $0000 as a result of operation; else cleared.
V: Set if two’s complement overflow occurs as a result of the operation; else cleared.
C: Set if there is a carry from D during operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	81	ff	6
IND8, Y	91	ff	6
IND8, Z	A1	ff	6
IMM8	FC	ii	2
IMM16	37B1	jjkk	4
IND16, X	37C1	gggg	6
IND16, Y	37D1	gggg	6
IND16, Z	37E1	gggg	6
EXT	37F1	hhll	6
E, X	2781	—	6
E, Y	2791	—	6
E, Z	27A1	—	6

ADDE Add to E ADDE

Operation: (E) + (M : M + 1) ⇒ E

Description: Adds a memory word to the content of accumulator E, then places the result in accumulator E. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	Δ	Δ	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if E15 is set by operation; else cleared.
Z: Set if (E) = $0000 as a result of operation; else cleared.
V: Set if two’s complement overflow occurs as a result of the operation; else cleared.
C: Set if there is a carry from E during operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IMM8	7C	ii	2
IMM16	3731	jjkk	4
IND16, X	3741	gggg	6
IND16, Y	3751	gggg	6
IND16, Z	3761	gggg	6
EXT	3771	hhll	6

ADE Add D to E ADE

Operation: (E) + (D) ⇒ E

Description: Adds the content of accumulator D to the content of accumulator E, then places the result in accumulator E. Content of accumulator D is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7 5	4	3 0
S	MV	H	EV	N	Z	V	C	IP	SM	PK
–	–	–	–	Δ	Δ	Δ	Δ	–	–	–

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if E15 is set by operation; else cleared.
Z: Set if (E) = $0000 as a result of operation; else cleared.
V: Set if two’s complement overflow occurs as a result of the operation; else cleared.
C: Set if there is a carry from E during operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	2778	—	2

ADX Add D to IX ADX

Operation: (XK : IX) + (20 « D) ⇒ XK : IX

Description: Sign-extends the content of accumulator D to 20 bits, then adds it to the content of concatenated XK and IX. Content of accumulator D does not change.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	37CD	—	2

ADY Add D to IY ADY

Operation: (YK : IY) + (20 « D) ⇒ YK : IY

Description: Sign-extends the content of accumulator D to 20 bits, then adds it to the content of concatenated YK and IY. Content of accumulator D does not change.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	37DD	—	2

ADZ Add D to IZ ADZ

Operation: (ZK : IZ) + (20 « D) ⇒ ZK : IZ

Description: Sign-extends the content of accumulator D to 20 bits, then adds it to the content of concatenated ZK and IZ. Content of accumulator D does not change.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	37ED	—	2

AEX Add E to IX AEX

Operation: (XK : IX) + (20 « E) ⇒ XK : IX

Description: Sign-extends the content of accumulator E to 20 bits, then adds it to the content of concatenated XK and IX. Content of accumulator E does not change.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	374D	—	2

AEY Add E to IY AEY

Operation: (YK : IY) + (20 « E) ⇒ YK : IY

Description: Sign-extends the content of accumulator E to 20 bits, then adds it to the content of concatenated YK and IY. Content of accumulator E does not change.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	375D	—	2

AEZ Add E to IZ AEZ

Operation: (ZK : IZ) + (20 « E) ⇒ ZK : IZ

Description: Sign-extends the content of accumulator E to 20 bits, then adds it to the content of concatenated ZK and IZ. Content of accumulator E does not change.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	376D	—	2

AIS Add Immediate Value to Stack Pointer AIS

Operation: (SK : SP) + (20 « IMM)⇒ SK : SP

Description: Adds a 20-bit value to concatenated SK and SP. The 20-bit value is formed by sign-extending an 8-bit or 16-bit signed immediate operand.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IMM8	3F	ii	2
IMM16	373F	jjkk	4

AIX — Add Immediate Value to IX — AIX

Operation: (XK : IX) + (20 « IMM) ⇒ XK : IX

Description: Adds a 20-bit value to the concatenated XK and IX. The 20-bit value is formed by sign-extending an 8-bit or 16-bit signed immediate operand.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	–	Δ	–	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Not affected.
Z: Set if (IX) = $0000 as a result of operation; else cleared.
V: Not affected.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IMM8	3C	ii	2
IMM16	373C	jjkk	4

AIY Add Immediate Value to IY AIY

Operation: (YK : IY) + (20 « IMM) ⇒ YK : IY

Description: Adds a 20-bit value to the concatenated YK and IY. The 20-bit value is formed by sign-extending an 8-bit or 16-bit signed immediate operand.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM			PK	
–	–	–	–	–	∆	–	–		–		–			–	

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Not affected.
Z: Set if (IY) = $0000 as a result of operation; else cleared.
V: Not affected.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IMM8	3D	ii	2
IMM16	373D	jjkk	4

AIZ Add Immediate Value to IZ AIZ

Operation: (ZK : IZ) + (20 « IMM) ⇒ ZK : IZ

Description: Adds a 20-bit value to the concatenated ZK and IZ. The 20-bit value is formed by sign-extending an 8-bit or 16-bit signed immediate operand.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	—	Δ	—	—	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Not affected.
Z: Set if (IZ) = $0000 as a result of operation; else cleared.
V: Not affected.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IMM8	3E	ii	2
IMM16	373E	jjkk	4

ANDA AND A ANDA

Operation: $(A) \leq (M) \Rightarrow A$

Description: Performs AND between the content of accumulator A and a memory byte, then places the result in accumulator A. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	0	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 is set by operation; else cleared.
Z: Set if (A) = $00 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	46	ff	6
IND8, Y	56	ff	6
IND8, Z	66	ff	6
IMM8	76	ii	2
IND16, X	1746	gggg	6
IND16, Y	1756	gggg	6
IND16, Z	1766	gggg	6
EXT	1776	hhll	6
E, X	2746	—	6
E, Y	2756	—	6
E, Z	2766	—	6

ANDB AND B ANDB

Operation: $(B) \le (M) \Rightarrow B$

Description: Performs AND between the content of accumulator B and a memory byte, then places the result in accumulator B. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	∆	∆	0	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if B7 is set by operation; else cleared.
Z: Set if (B) = $00 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	C6	ff	6
IND8, Y	D6	ff	6
IND8, Z	E6	ff	6
IMM8	F6	ii	2
IND16, X	17C6	gggg	6
IND16, Y	17D6	gggg	6
IND16, Z	17E6	gggg	6
EXT	17F6	hhll	6
E, X	27C6	—	6
E, Y	27D6	—	6
E, Z	27E6	—	6

ANDD AND D ANDD

Operation: $(D) \leq (M : M + 1) \Rightarrow D$

Description: Performs AND between the content of accumulator D and a memory word, then places the result in accumulator D. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	0	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if D is set by operation; else cleared.
Z: Set if (D) = $0000 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	86	ff	6
IND8, Y	96	ff	6
IND8, Z	A6	ff	6
IMM16	37B6	jjkk	4
IND16, X	37C6	gggg	6
IND16, Y	37D6	gggg	6
IND16, Z	37E6	gggg	6
EXT	37F6	hhll	6
E, X	2786	—	6
E, Y	2796	—	6
E, Z	27A6	—	6

ANDE AND E ANDE

Operation: (E) ≤ (M : M + 1) ⇒ E

Description: Performs AND between the content of accumulator E and a memory word, then places the result in accumulator E. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	0	–	–			–	–			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if E15 is set by operation; else cleared.
Z:	Set if (E) = $0000 as a result of operation; else cleared.
V:	Cleared.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IMM16	3736	jjkk	4
IND16, X	3746	gggg	6
IND16, Y	3756	gggg	6
IND16, Z	3766	gggg	6
EXT	3776	hhll	6

ANDP AND Condition Code Register ANDP

Operation: (CCR) ≤ IMM16 ⇒ CCR

Description: Performs AND between the content of the condition code register and an unsigned immediate operand, then replaces the content of the CCR with the result.

To make certain that conditions for termination of LPSTOP and WAI are correct, interrupts are not recognized until after the instruction following ANDP executes. This prevents interrupt exception processing during the period after the mask changes but before the following instruction executes.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ			Δ	–			

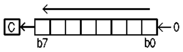
S: Cleared if bit 15 of operand = 0; else unchanged.
MV: Cleared if bit 14 of operand = 0; else unchanged.
H: Cleared if bit 13 of operand = 0; else unchanged.
EV: Cleared if bit 12 of operand = 0; else unchanged.
N: Cleared if bit 11 of operand = 0; else unchanged.
Z: Cleared if bit 10 of operand = 0; else unchanged.
V: Cleared if bit 9 of operand = 0; else unchanged.
C: Cleared if bit 8 of operand = 0; else unchanged.
IP: Each bit in field cleared if corresponding bit [7:5] of operand = 0; else unchanged.
SM: Cleared if bit 4 of operand = 0; else unchanged.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IMM16	373A	jjkk	4

ASL Arithmetic Shift Left ASL

Operation:



Description: Shifts all eight bits of a memory byte one place to the left. Bit 7 is transferred to the CCR C bit. Bit 0 is loaded with a zero.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	Δ	Δ	—			—	—			

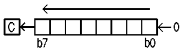
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M7 = 1 as a result of operation; else cleared.
Z: Set if (M) = $00 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if M7 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	04	ff	8
IND8, Y	14	ff	8
IND8, Z	24	ff	8
IND16, X	1704	gggg	8
IND16, Y	1714	gggg	8
IND16, Z	1724	gggg	8
EXT	1734	hhll	8

ASLA Arithmetic Shift Left A ASLA

Operation:



Description: Shifts all eight bits of accumulator A one place to the left. Bit 7 is transferred to the CCR C bit. Bit 0 is loaded with a zero.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	Δ	Δ	—			—	—			

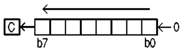
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 = 1 as a result of operation; else cleared.
Z: Set if (A) = $00 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if A7 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3704	—	2

ASLB Arithmetic Shift Left B ASLB

Operation:



Description: Shifts all eight bits of accumulator B one place to the left. Bit 7 is transferred to the CCR C bit. Bit 0 is loaded with a zero.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM			PK	
—	—	—	—	∆	∆	∆	∆		—		—			—	

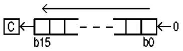
S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if B7 = 1 as a result of operation; else cleared.
Z:	Set if (B) = $00 as a result of operation; else cleared.
V:	Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C:	Set if B7 = 1 before operation; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3714	—	2

ASLD Arithmetic Shift Left D ASLD

Operation:



Description: Shifts all sixteen bits of accumulator D one place to the left. Bit 15 is transferred to the CCR C bit. Bit 0 is loaded with a zero.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	Δ	Δ	–			–	–			

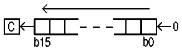
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if D15 = 1 as a result of operation; else cleared.
Z: Set if (D) = $0000 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if D15 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27F4	—	2

ASLE Arithmetic Shift Left E ASLE

Operation:



Description: Shifts all sixteen bits of accumulator E one place to the left. Bit 15 is transferred to the CCR C bit. Bit 0 is loaded with a zero.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	Δ	Δ	—			—	—			

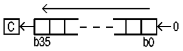
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if E15 = 1 as a result of operation; else cleared.
Z: Set if (E) = $0000 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if E15 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	2774	—	2

ASLM Arithmetic Shift Left AM ASLM

Operation:



Description: Shifts all 36 bits of accumulator M one place to the left. Bit 35 is transferred to the CCR C bit. Bit 0 is loaded with a zero. See **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	Δ	—	Δ	Δ	—	—	Δ	—			—	—			

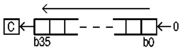
S: Not affected.
MV: Set if AM[35] has changed state as a result of operation; else unchanged.
H: Not affected.
EV: Cleared if AM[34:31] = $0000 or $1111 as a result of operation; else set.
N: Set if M35 = 1 as a result of operation; else cleared.
Z: Not affected.
V: Not affected.
C: Set if AM35 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27B6	—	4

ASLW Arithmetic Shift Left Word ASLW

Operation:



Description: Shifts all sixteen bits of memory word one place to the left. Bit 15 is transferred to the CCR C bit. Bit 0 is loaded with a zero.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	Δ	Δ	–			–	–			

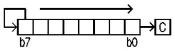
S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if M : M + 1[15] = 1 as a result of operation; else cleared.
Z:	Set if (M : M + 1) = $0000 as a result of operation; else cleared.
V:	Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C:	Set if M : M + 1[15] = 1 before operation; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND16, X	2704	gggg	8
IND16, Y	2714	gggg	8
IND16, Z	2724	gggg	8
EXT	2734	hhll	8

ASR Arithmetic Shift Right ASR

Operation:



Description: Shifts all eight bits of a memory byte one place to the right. Bit 7 is held constant. Bit 0 is transferred to the CCR C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	Δ	Δ	—			—	—			

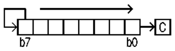
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M7 set as a result of operation; else cleared.
Z: Set if (M) = \$00 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if M0 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	0D	ff	8
IND8, Y	1D	ff	8
IND8, Z	2D	ff	8
IND16, X	170D	gggg	8
IND16, Y	171D	gggg	8
IND16, Z	172D	gggg	8
EXT	173D	hhll	8

ASRA Arithmetic Shift Right A ASRA

Operation:



Description: Shifts all eight bits of accumulator A one place to the right. Bit 7 is held constant. Bit 0 is transferred to the CCR C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	Δ	Δ	–			–	–			

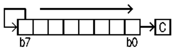
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 = 1 as a result of operation; else cleared.
Z: Set if (A) = $00; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if A0 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	370D	—	2

ASRB Arithmetic Shift Right B ASRB

Operation:



Description: Shifts all eight bits of accumulator B one place to the right. Bit 7 is held constant. Bit 0 is transferred to the CCR C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	Δ	Δ	–			–	–			

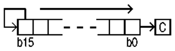
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if B7 = 1 as a result of operation; else cleared.
Z: Set if (B) = $00 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if B0 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	371D	—	2

ASRD Arithmetic Shift Right D ASRD

Operation:



Description: Shifts all sixteen bits of accumulator D one place to the right. Bit 15 is held constant. Bit 0 is transferred to the CCR C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
–	–	–	–	∆	∆	∆	∆		–		–		–		

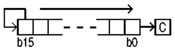
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if D15 = 1 as a result of operation; else cleared.
Z: Set if (D) = $0000 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if D0 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27FD	—	2

ASRE Arithmetic Shift Right E ASRE

Operation:



Description: Shifts all sixteen bits of accumulator E one place to the right. Bit 15 is held constant. Bit 0 is transferred to the CCR C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	Δ	Δ	–			–	–			

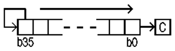
S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if E15 = 1 as a result of operation; else cleared.
Z:	Set if (E) = $0000 as a result of operation; else cleared.
V:	Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C:	Set if E0 = 1 before operation; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	277D	—	2

ASRM Arithmetic Shift Right AM ASRM

Operation:



Description: Shifts all 36 bits of accumulator M one place to the right. Bit 35 is held constant. Bit 0 is transferred to the CCR C bit. See **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	∆	∆	–	–	∆	–			–	–			

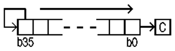
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Cleared if AM[34:31] = \$0000 or \$1111 as a result of operation; else set.
N: Set if AM35 = 1 as a result of operation; else cleared.
Z: Not affected.
V: Not affected.
C: Set if AM0 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27BA	—	4

ASRW Arithmetic Shift Right Word ASRW

Operation:



Description: Shifts all sixteen bits of a memory word one place to the right. Bit 15 is held constant. Bit 0 is transferred to the CCR C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	∆	∆	∆	∆	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M : M + 1[15] = 1 as a result of operation; else cleared.
Z: Set if (M : M + 1) = $0000 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if M : M + 1[0] = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND16, X	270D	gggg	8
IND16, Y	271D	gggg	8
IND16, Z	272D	gggg	8
EXT	273D	hhll	8

BCC Branch If Carry Clear BCC

Operation: If C = 0, then (PK : PC) + Offset ⇒ PK : PC

Description: Causes a program branch if the CCR carry bit has a value of zero. An 8-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement simple or unsigned conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	B4	rr	6, 2

Table 6-2 Branch Instruction Summary (8-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
BCC	B4	C = 0	Simple, Unsigned	BCS
BCS	B5	C = 1	Simple, Unsigned	BCC
BEQ	B7	Z = 1	Simple, Unsigned, Signed	BNE
BGE	BC	N ⊕ V = 0	Signed	BLT
BGT	BE	Z ✢ (N ⊕ V) = 0	Signed	BLE
BHI	B2	C ✢ Z = 0	Unsigned	BLS
BLE	BF	Z ✢ (N ⊕ V) = 1	Signed	BGT
BLS	B3	C ✢ Z = 1	Unsigned	BHI
BLT	BD	N ⊕ V = 1	Signed	BGE
BMI	BB	N = 1	Simple	BPL
BNE	B6	Z = 0	Simple, Unsigned, Signed	BEQ
BPL	BA	N = 0	Simple	BMI
BRA	B0	1	Unary	BRN
BRN	B1	0	Unary	BRA
BVC	B8	V = 0	Simple	BVS
BVS	B9	V = 1	Simple	BVC

BCLR Clear Bits BCLR

Operation: $(M) \leq (\overline{\text{Mask}}) \Rightarrow M$

Description: Performs AND between a memory byte and the complement of a mask byte. Bits in the mask are set to clear corresponding bits in memory. Other bits in the memory byte are unchanged. The location of the mask differs for 8- and 16-bit addressing modes.

Syntax: BCLR address operand, [register symbol,] #mask

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	0	–	–			–	–			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if M7 = 1 as a result of operation; else cleared.
Z:	Set if (M) = $00 as a result of operation; else cleared.
V:	Cleared.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Mask	Operand	Cycles
IND8, X	1708	mm	ff	8
IND8, Y	1718	mm	ff	8
IND8, Z	1728	mm	ff	8
IND16, X	08	mm	gggg	8
IND16, Y	18	mm	gggg	8
IND16, Z	28	mm	gggg	8
EXT	38	mm	hhll	8

BCLRW Clear Bits in a Word BCLRW

Operation: $(M : M + 1) \leq (\overline{\text{Mask}}) \Rightarrow M : M + 1$

Description: Performs AND between a memory word and the complement of a mask word. Bits in the mask are set to clear corresponding bits in memory. Other bits in the memory word are unchanged.

Syntax: BCLRW Address Operand, [Index Register Symbol,] #Mask

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	0	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M15 = 1 as a result of operation; else cleared.
Z: Set if (M : M + 1) = $0000 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Mask	Cycles
IND16, X	2708	gggg	mmmm	10
IND16, Y	2718	gggg	mmmm	10
IND16, Z	2728	gggg	mmmm	10
EXT	2738	hhll	mmmm	10

BCS Branch If Carry Set BCS

Operation: If C = 1, then (PK : PC) + Offset ⇒ PK : PC

Description: Causes a program branch if the CCR carry bit has a value of one. An 8-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement simple or unsigned conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	B5	rr	6, 2

Table 6-3 Branch Instruction Summary (8-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
BCC	B4	$C = 0$	Simple, Unsigned	BCS
BCS	B5	$C = 1$	Simple, Unsigned	BCC
BEQ	B7	$Z = 1$	Simple, Unsigned, Signed	BNE
BGE	BC	$N \oplus V = 0$	Signed	BLT
BGT	BE	$Z + (N \oplus V) = 0$	Signed	BLE
BHI	B2	$C + Z = 0$	Unsigned	BLS
BLE	BF	$Z + (N \oplus V) = 1$	Signed	BGT
BLS	B3	$C + Z = 1$	Unsigned	BHI
BLT	BD	$N \oplus V = 1$	Signed	BGE
BMI	BB	$N = 1$	Simple	BPL
BNE	B6	$Z = 0$	Simple, Unsigned, Signed	BEQ
BPL	BA	$N = 0$	Simple	BMI
BRA	B0	1	Unary	BRN
BRN	B1	0	Unary	BRA
BVC	B8	$V = 0$	Simple	BVS
BVS	B9	$V = 1$	Simple	BVC

BEQ Branch If Equal to Zero BEQ

Operation: If Z = 1, then (PK : PC) + Offset ⇒ PK : PC

Description: Causes a program branch if the CCR zero bit has a value of one. An 8-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement simple, signed, or unsigned conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	B7	rr	6, 2

Table 6-4 Branch Instruction Summary (8-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
BCC	B4	C = 0	Simple, Unsigned	BCS
BCS	B5	C = 1	Simple, Unsigned	BCC
BEQ	B7	Z = 1	Simple, Unsigned, Signed	BNE
BGE	BC	N ⊕ V = 0	Signed	BLT
BGT	BE	Z ✢ (N ⊕ V) = 0	Signed	BLE
BHI	B2	C ✢ Z = 0	Unsigned	BLS
BLE	BF	Z ✢ (N ⊕ V) = 1	Signed	BGT
BLS	B3	C ✢ Z = 1	Unsigned	BHI
BLT	BD	N ⊕ V = 1	Signed	BGE
BMI	BB	N = 1	Simple	BPL
BNE	B6	Z = 0	Simple, Unsigned, Signed	BEQ
BPL	BA	N = 0	Simple	BMI
BRA	B0	1	Unary	BRN
BRN	B1	0	Unary	BRA
BVC	B8	V = 0	Simple	BVS
BVS	B9	V = 1	Simple	BVC

BGE Branch If Greater than or Equal to Zero BGE

Operation: If $N \oplus V = 0$, then $(PK : PC) + \text{Offset} \Rightarrow PK : PC$

Description: Causes a program branch if the CCR negative and overflow bits both have a value of zero or both have a value of one. An 8-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement signed conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	BC	rr	6, 2

Table 6-5 Branch Instruction Summary (8-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
BCC	B4	$C = 0$	Simple, Unsigned	BCS
BCS	B5	$C = 1$	Simple, Unsigned	BCC
BEQ	B7	$Z = 1$	Simple, Unsigned, Signed	BNE
BGE	BC	$N \oplus V = 0$	Signed	BLT
BGT	BE	$Z + (N \oplus V) = 0$	Signed	BLE
BHI	B2	$C + Z = 0$	Unsigned	BLS
BLE	BF	$Z + (N \oplus V) = 1$	Signed	BGT
BLS	B3	$C + Z = 1$	Unsigned	BHI
BLT	BD	$N \oplus V = 1$	Signed	BGE
BMI	BB	$N = 1$	Simple	BPL
BNE	B6	$Z = 0$	Simple, Unsigned, Signed	BEQ
BPL	BA	$N = 0$	Simple	BMI
BRA	B0	1	Unary	BRN
BRN	B1	0	Unary	BRA
BVC	B8	$V = 0$	Simple	BVS
BVS	B9	$V = 1$	Simple	BVC

BGND Enter Background Debug Mode BGND

Operation: If background debug mode is enabled, begin debug; else, illegal instruction trap

Description: Background debug mode is an operating mode in which the CPU16 microcode performs debugging functions. To prevent accidental entry, a specific method of enabling BDM is used. If BDM has been correctly enabled, executing BGND will cause the CPU16 to suspend normal operation. If BDM has not been correctly enabled, an illegal instruction exception is generated. See **SECTION 9 EXCEPTION PROCESSING** for more information.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	37A6	—	N/A

BGT Branch If Greater than Zero BGT

Operation: If Z ✢ (N ⊕ V) = 0, then (PK : PC) + Offset ⇒ PK : PC

Description: Causes a program branch if the CCR negative and overflow bits both have a value of zero or both have a value of one, and the CCR zero bit has a value of zero. An 8-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement signed conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	BE	rr	6, 2

Table 6-6 Branch Instruction Summary (8-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
BCC	B4	C = 0	Simple, Unsigned	BCS
BCS	B5	C = 1	Simple, Unsigned	BCC
BEQ	B7	Z = 1	Simple, Unsigned, Signed	BNE
BGE	BC	N ⊕ V = 0	Signed	BLT
BGT	BE	Z ✢ (N ⊕ V) = 0	Signed	BLE
BHI	B2	C ✢ Z = 0	Unsigned	BLS
BLE	BF	Z ✢ (N ⊕ V) = 1	Signed	BGT
BLS	B3	C ✢ Z = 1	Unsigned	BHI
BLT	BD	N ⊕ V = 1	Signed	BGE
BMI	BB	N = 1	Simple	BPL
BNE	B6	Z = 0	Simple, Unsigned, Signed	BEQ
BPL	BA	N = 0	Simple	BMI
BRA	B0	1	Unary	BRN
BRN	B1	0	Unary	BRA
BVC	B8	V = 0	Simple	BVS
BVS	B9	V = 1	Simple	BVC

BHI **Branch If Higher** BHI

Operation: If C ✢ Z = 0, then (PK : PC) + Offset ⇒ PK : PC

Description: Causes a program branch if the CCR carry and zero bits both have a value of zero. An 8-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement unsigned conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	B2	rr	6, 2

Table 6-7 Branch Instruction Summary (8-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
BCC	B4	C = 0	Simple, Unsigned	BCS
BCS	B5	C = 1	Simple, Unsigned	BCC
BEQ	B7	Z = 1	Simple, Unsigned, Signed	BNE
BGE	BC	N ⊕ V = 0	Signed	BLT
BGT	BE	Z ✢ (N ⊕ V) = 0	Signed	BLE
BHI	B2	C ✢ Z = 0	Unsigned	BLS
BLE	BF	Z ✢ (N ⊕ V) = 1	Signed	BGT
BLS	B3	C ✢ Z = 1	Unsigned	BHI
BLT	BD	N ⊕ V = 1	Signed	BGE
BMI	BB	N = 1	Simple	BPL
BNE	B6	Z = 0	Simple, Unsigned, Signed	BEQ
BPL	BA	N = 0	Simple	BMI
BRA	B0	1	Unary	BRN
BRN	B1	0	Unary	BRA
BVC	B8	V = 0	Simple	BVS
BVS	B9	V = 1	Simple	BVC

BITA Bit Test A BITA

Operation: (A) ≤ (M)

Description: Performs AND between the content of accumulator A and corresponding bits in a memory byte. Condition codes are set, but neither accumulator content nor memory content is changed.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	∆	∆	0	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 ≤ M7 = 1; else cleared.
Z: Set if (A) ≤ (M) = $00; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	49	ff	6
IND8, Y	59	ff	6
IND8, Z	69	ff	6
IMM8	79	ii	2
IND16, X	1749	gggg	6
IND16, Y	1759	gggg	6
IND16, Z	1769	gggg	6
EXT	1779	hhll	6
E, X	2749	—	6
E, Y	2759	—	6
E, Z	2769	—	6

BITB Bit Test B BITB

Operation: (B) ≤ (M)

Description: Performs AND between the content of accumulator B and corresponding bits in a memory byte. Condition codes are set, but neither accumulator content nor memory content is changed.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	0	–	–			–	–			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if B7 ≤ M7 = 1; else cleared.
Z:	Set if (B) ≤ (M) = $00; else cleared.
V:	Cleared.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	C9	ff	6
IND8, Y	D9	ff	6
IND8, Z	E9	ff	6
IMM8	F9	ii	2
IND16, X	17C9	gggg	6
IND16, Y	17D9	gggg	6
IND16, Z	17E9	gggg	6
EXT	17F9	hhll	6
E, X	27C9	—	6
E, Y	27D9	—	6
E, Z	27E9	—	6

BLE Branch If Less than or Equal to Zero BLE

Operation: If Z ✢ (N ⊕ V) = 1, then (PK : PC) + Offset ⇒ PK : PC

Description: Causes a program branch if either the CCR negative bit or overflow bit has a value of one, or the CCR zero bit has a value of one. An 8-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement signed conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	BF	rr	6, 2

Table 6-8 Branch Instruction Summary (8-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
BCC	B4	C = 0	Simple, Unsigned	BCS
BCS	B5	C = 1	Simple, Unsigned	BCC
BEQ	B7	Z = 1	Simple, Unsigned, Signed	BNE
BGE	BC	N ⊕ V = 0	Signed	BLT
BGT	BE	Z ✢ (N ⊕ V) = 0	Signed	BLE
BHI	B2	C ✢ Z = 0	Unsigned	BLS
BLE	BF	Z ✢ (N ⊕ V) = 1	Signed	BGT
BLS	B3	C ✢ Z = 1	Unsigned	BHI
BLT	BD	N ⊕ V = 1	Signed	BGE
BMI	BB	N = 1	Simple	BPL
BNE	B6	Z = 0	Simple, Unsigned, Signed	BEQ
BPL	BA	N = 0	Simple	BMI
BRA	B0	1	Unary	BRN
BRN	B1	0	Unary	BRA
BVC	B8	V = 0	Simple	BVS
BVS	B9	V = 1	Simple	BVC

BLS Branch If Lower or Same BLS

Operation: If $C + Z = 1$, then $(PK : PC) + \text{Offset} \Rightarrow PK : PC$

Description: Causes a program branch if either or both the CCR carry and zero bits have a value of one. An 8-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement unsigned conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	B3	rr	6, 2

Table 6-9 Branch Instruction Summary (8-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
BCC	B4	$C = 0$	Simple, Unsigned	BCS
BCS	B5	$C = 1$	Simple, Unsigned	BCC
BEQ	B7	$Z = 1$	Simple, Unsigned, Signed	BNE
BGE	BC	$N \oplus V = 0$	Signed	BLT
BGT	BE	$Z + (N \oplus V) = 0$	Signed	BLE
BHI	B2	$C + Z = 0$	Unsigned	BLS
BLE	BF	$Z + (N \oplus V) = 1$	Signed	BGT
BLS	B3	$C + Z = 1$	Unsigned	BHI
BLT	BD	$N \oplus V = 1$	Signed	BGE
BMI	BB	$N = 1$	Simple	BPL
BNE	B6	$Z = 0$	Simple, Unsigned, Signed	BEQ
BPL	BA	$N = 0$	Simple	BMI
BRA	B0	1	Unary	BRN
BRN	B1	0	Unary	BRA
BVC	B8	$V = 0$	Simple	BVS
BVS	B9	$V = 1$	Simple	BVC

BLT Branch If Less than Zero BLT

Operation: If $N \oplus V = 1$, then $(PK : PC) + \text{Offset} \Rightarrow PK : PC$

Description: Causes a program branch if either of the CCR negative or overflow bits has a value of one. An 8-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement signed conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	BD	rr	6, 2

Table 6-10 Branch Instruction Summary (8-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
BCC	B4	C = 0	Simple, Unsigned	BCS
BCS	B5	C = 1	Simple, Unsigned	BCC
BEQ	B7	Z = 1	Simple, Unsigned, Signed	BNE
BGE	BC	N ⊕ V = 0	Signed	BLT
BGT	BE	Z ✢ (N ⊕ V) = 0	Signed	BLE
BHI	B2	C ✢ Z = 0	Unsigned	BLS
BLE	BF	Z ✢ (N ⊕ V) = 1	Signed	BGT
BLS	B3	C ✢ Z = 1	Unsigned	BHI
BLT	BD	N ⊕ V = 1	Signed	BGE
BMI	BB	N = 1	Simple	BPL
BNE	B6	Z = 0	Simple, Unsigned, Signed	BEQ
BPL	BA	N = 0	Simple	BMI
BRA	B0	1	Unary	BRN
BRN	B1	0	Unary	BRA
BVC	B8	V = 0	Simple	BVS
BVS	B9	V = 1	Simple	BVC

BMI — Branch If Minus — BMI

Operation: If N = 1, then (PK : PC) + Offset ⇒ PK : PC

Description: Causes a program branch if the CCR negative bit has a value of one. An 8-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement simple conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	BB	rr	6, 2

Table 6-11 Branch Instruction Summary (8-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
BCC	B4	C = 0	Simple, Unsigned	BCS
BCS	B5	C = 1	Simple, Unsigned	BCC
BEQ	B7	Z = 1	Simple, Unsigned, Signed	BNE
BGE	BC	N ⊕ V = 0	Signed	BLT
BGT	BE	Z ✛ (N ⊕ V) = 0	Signed	BLE
BHI	B2	C ✛ Z = 0	Unsigned	BLS
BLE	BF	Z ✛ (N ⊕ V) = 1	Signed	BGT
BLS	B3	C ✛ Z = 1	Unsigned	BHI
BLT	BD	N ⊕ V = 1	Signed	BGE
BMI	BB	N = 1	Simple	BPL
BNE	B6	Z = 0	Simple, Unsigned, Signed	BEQ
BPL	BA	N = 0	Simple	BMI
BRA	B0	1	Unary	BRN
BRN	B1	0	Unary	BRA
BVC	B8	V = 0	Simple	BVS
BVS	B9	V = 1	Simple	BVC

BNE Branch If Not Equal to Zero BNE

Operation: If $Z = 0$, then $(PK : PC) + \text{Offset} \Rightarrow PK : PC$

Description: Causes a program branch if the CCR zero bit has a value of zero. An 8-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement simple, signed, and unsigned conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	B6	rr	6, 2

Table 6-12 Branch Instruction Summary (8-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
BCC	B4	$C = 0$	Simple, Unsigned	BCS
BCS	B5	$C = 1$	Simple, Unsigned	BCC
BEQ	B7	$Z = 1$	Simple, Unsigned, Signed	BNE
BGE	BC	$N \oplus V = 0$	Signed	BLT
BGT	BE	$Z + (N \oplus V) = 0$	Signed	BLE
BHI	B2	$C + Z = 0$	Unsigned	BLS
BLE	BF	$Z + (N \oplus V) = 1$	Signed	BGT
BLS	B3	$C + Z = 1$	Unsigned	BHI
BLT	BD	$N \oplus V = 1$	Signed	BGE
BMI	BB	$N = 1$	Simple	BPL
BNE	B6	$Z = 0$	Simple, Unsigned, Signed	BEQ
BPL	BA	$N = 0$	Simple	BMI
BRA	B0	1	Unary	BRN
BRN	B1	0	Unary	BRA
BVC	B8	$V = 0$	Simple	BVS
BVS	B9	$V = 1$	Simple	BVC

BPL Branch If Plus BPL

Operation: If $N = 0$, then $(PK : PC) + \text{Offset} \Rightarrow PK : PC$

Description: Causes a program branch if the CCR negative bit has a value of zero. An 8-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement simple conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	BA	rr	6, 2

Table 6-13 Branch Instruction Summary (8-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
BCC	B4	$C = 0$	Simple, Unsigned	BCS
BCS	B5	$C = 1$	Simple, Unsigned	BCC
BEQ	B7	$Z = 1$	Simple, Unsigned, Signed	BNE
BGE	BC	$N \oplus V = 0$	Signed	BLT
BGT	BE	$Z ✣ (N \oplus V) = 0$	Signed	BLE
BHI	B2	$C ✣ Z = 0$	Unsigned	BLS
BLE	BF	$Z ✣ (N \oplus V) = 1$	Signed	BGT
BLS	B3	$C ✣ Z = 1$	Unsigned	BHI
BLT	BD	$N \oplus V = 1$	Signed	BGE
BMI	BB	$N = 1$	Simple	BPL
BNE	B6	$Z = 0$	Simple, Unsigned, Signed	BEQ
BPL	BA	$N = 0$	Simple	BMI
BRA	B0	1	Unary	BRN
BRN	B1	0	Unary	BRA
BVC	B8	$V = 0$	Simple	BVS
BVS	B9	$V = 1$	Simple	BVC

BRA Branch Always BRA

Operation: (PK : PC) + Offset ⇒ PK : PC

Description: Always branches. An 8-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	B0	rr	6

Table 6-14 Branch Instruction Summary (8-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
BCC	B4	C = 0	Simple, Unsigned	BCS
BCS	B5	C = 1	Simple, Unsigned	BCC
BEQ	B7	Z = 1	Simple, Unsigned, Signed	BNE
BGE	BC	N ⊕ V = 0	Signed	BLT
BGT	BE	Z ✢ (N ⊕ V) = 0	Signed	BLE
BHI	B2	C ✢ Z = 0	Unsigned	BLS
BLE	BF	Z ✢ (N ⊕ V) = 1	Signed	BGT
BLS	B3	C ✢ Z = 1	Unsigned	BHI
BLT	BD	N ⊕ V = 1	Signed	BGE
BMI	BB	N = 1	Simple	BPL
BNE	B6	Z = 0	Simple, Unsigned, Signed	BEQ
BPL	BA	N = 0	Simple	BMI
BRA	B0	1	Unary	BRN
BRN	B1	0	Unary	BRA
BVC	B8	V = 0	Simple	BVS
BVS	B9	V = 1	Simple	BVC

BRCLR Branch if Bits Clear BRCLR

Operation: If (M) ≤ (Mask) = 0, (PK : PC) + Offset ⇒ PK : PC

Description: Causes a program branch when specified bits in memory have values of zero. Performs AND between a memory byte and a mask byte. The memory byte is pointed to by a 20-bit indexed or extended effective address.

If a mask bit has a value of one, the corresponding memory bit must have a value of zero. When the result of the operation is zero, an 8- or 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented.

Syntax: BRCLR address operand, [register symbol,] #mask, displacement

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Mask	Addr Operand	Branch Offset	Cycles
IND8, X	CB	mm	ff	rr	10, 12
IND8, Y	DB	mm	ff	rr	10, 12
IND8, Z	EB	mm	ff	rr	10, 12
IND16, X	0A	mm	gggg	rrrr	10, 14
IND16, Y	1A	mm	gggg	rrrr	10, 14
IND16, Z	2A	mm	gggg	rrrr	10, 14
EXT	3A	mm	hhll	rrrr	10, 14

BRN Branch Never BRN

Operation: $(PK : PC) + 2 \Rightarrow PK : PC$

Description: Never branches. This instruction is effectively a NOP that requires two cycles to execute. When the operation causes PC overflow, the PK field is incremented or decremented.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	B1	rr	2

Table 6-15 Branch Instruction Summary (8-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
BCC	B4	$C = 0$	Simple, Unsigned	BCS
BCS	B5	$C = 1$	Simple, Unsigned	BCC
BEQ	B7	$Z = 1$	Simple, Unsigned, Signed	BNE
BGE	BC	$N \oplus V = 0$	Signed	BLT
BGT	BE	$Z + (N \oplus V) = 0$	Signed	BLE
BHI	B2	$C + Z = 0$	Unsigned	BLS
BLE	BF	$Z + (N \oplus V) = 1$	Signed	BGT
BLS	B3	$C + Z = 1$	Unsigned	BHI
BLT	BD	$N \oplus V = 1$	Signed	BGE
BMI	BB	$N = 1$	Simple	BPL
BNE	B6	$Z = 0$	Simple, Unsigned, Signed	BEQ
BPL	BA	$N = 0$	Simple	BMI
BRA	B0	1	Unary	BRN
BRN	B1	0	Unary	BRA
BVC	B8	$V = 0$	Simple	BVS
BVS	B9	$V = 1$	Simple	BVC

BRSET Branch if Bits Set BRSET

Operation: If $(\overline{M}) \leq (\text{Mask}) = 0$, $(PC) + \text{Offset} \Rightarrow PK : PC$

Description: Causes a program branch when specified bits in memory have values of one. Performs AND between the complement of memory byte and a mask byte. The memory byte is pointed to by a 20-bit indexed or extended effective address.

If a mask bit has a value of one, the corresponding (uncomplemented) memory bit must have a value of one. When the result of the operation is zero, an 8- or 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented.

Syntax: BRSET address operand, [register symbol,] #mask, displacement

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Mask	Addr Operand	Branch Offset	Cycles
IND8, X	8B	mm	ff	rr	10, 12
IND8, Y	9B	mm	ff	rr	10, 12
IND8, Z	AB	mm	ff	rr	10, 12
IND16, X	0B	mm	gggg	rrrr	10, 14
IND16, Y	1B	mm	gggg	rrrr	10, 14
IND16, Z	2B	mm	gggg	rrrr	10, 14
EXT	3B	mm	hhll	rrrr	10, 14

BSET Set Bits in a Byte BSET

Operation: (M) ✢ (MASK) ⇒ M

Description: Performs OR between a memory byte and a mask byte. Bits in the mask are set to set corresponding bits in memory. Other bits in the memory word are unchanged. The location of the mask differs for 8- and 16-bit addressing modes.

Syntax: BSET address operand, [register symbol,] #mask

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
–	–	–	–	∆	∆	0	–		–		–		–		

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if M7 = 1 as a result of operation; else cleared.
Z:	Set if (M) = $00 as a result of operation; else cleared.
V:	Cleared.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Mask	Operand	Cycles
IND8, X	1709	mm	ff	8

BSETW Set Bits in a Word BSETW

Operation: (M : M + 1) ✛ (Mask) ⇒ M : M + 1

Description: Performs OR between a memory word and a mask word. Set bits in the mask to set corresponding bits in memory. Other bits in the memory word are unchanged.

Syntax: BSETW address operand, [register symbol,] #mask

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	∆	∆	0	—	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if M15 = 1 as a result of operation; else cleared.
Z:	Set if (M : M + 1) = $0000 as a result of operation; else cleared.
V:	Cleared.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Mask	Cycles
IND16, X	2709	gggg	mmmm	10
IND16, Y	2719	gggg	mmmm	10
IND16, Z	2729	gggg	mmmm	10
EXT	2739	hhll	mmmm	10

BSR Branch to Subroutine BSR

Operation:

(PK : PC) – \$0002 ⇒ PK : PC
Push (PC)
(SK : SP) – \$0002 ⇒ SK : SP
Push (CCR)
(SK : SP) – \$0002 ⇒ SK : SP
(PK : PC) + Offset ⇒ PK : PC

Description:

Saves current program address and status, then branches to a subroutine. PK : PC are adjusted so that program execution will resume correctly after return from subroutine.

The program counter is stacked, then the condition code register is stacked (PK field as well as condition code bits and interrupt priority mask). An 8-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	36	rr	10

BVC Branch If Overflow Clear BVC

Operation: If V = 0, then (PK : PC) + Offset ⇒ PK : PC

Description: Causes a program branch if the CCR overflow bit has a value of zero. An 8-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement simple, signed, and unsigned conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	B8	rr	6, 2

Table 6-16 Branch Instruction Summary (8-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
BCC	B4	C = 0	Simple, Unsigned	BCS
BCS	B5	C = 1	Simple, Unsigned	BCC
BEQ	B7	Z = 1	Simple, Unsigned, Signed	BNE
BGE	BC	N ⊕ V = 0	Signed	BLT
BGT	BE	Z ✢ (N ⊕ V) = 0	Signed	BLE
BHI	B2	C ✢ Z = 0	Unsigned	BLS
BLE	BF	Z ✢ (N ⊕ V) = 1	Signed	BGT
BLS	B3	C ✢ Z = 1	Unsigned	BHI
BLT	BD	N ⊕ V = 1	Signed	BGE
BMI	BB	N = 1	Simple	BPL
BNE	B6	Z = 0	Simple, Unsigned, Signed	BEQ
BPL	BA	N = 0	Simple	BMI
BRA	B0	1	Unary	BRN
BRN	B1	0	Unary	BRA
BVC	B8	V = 0	Simple	BVS
BVS	B9	V = 1	Simple	BVC

BVS Branch If Overflow Set BVS

Operation: If $V = 1$, then $(PK : PC) + \text{Offset} \Rightarrow PK : PC$

Description: Causes a program branch if the CCR overflow bit has a value of one. An 8-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement simple, signed, and unsigned conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL8	B9	rr	6, 2

Table 6-17 Branch Instruction Summary (8-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
BCC	B4	$C = 0$	Simple, Unsigned	BCS
BCS	B5	$C = 1$	Simple, Unsigned	BCC
BEQ	B7	$Z = 1$	Simple, Unsigned, Signed	BNE
BGE	BC	$N \oplus V = 0$	Signed	BLT
BGT	BE	$Z + (N \oplus V) = 0$	Signed	BLE
BHI	B2	$C + Z = 0$	Unsigned	BLS
BLE	BF	$Z + (N \oplus V) = 1$	Signed	BGT
BLS	B3	$C + Z = 1$	Unsigned	BHI
BLT	BD	$N \oplus V = 1$	Signed	BGE
BMI	BB	$N = 1$	Simple	BPL
BNE	B6	$Z = 0$	Simple, Unsigned, Signed	BEQ
BPL	BA	$N = 0$	Simple	BMI
BRA	B0	1	Unary	BRN
BRN	B1	0	Unary	BRA
BVC	B8	$V = 0$	Simple	BVS
BVS	B9	$V = 1$	Simple	BVC

CBA Compare B to A CBA

Operation: (A) – (B)

Description: Subtracts the content of accumulator B from the content of accumulator A and sets appropriate condition code register bits. The contents of the accumulators are not changed by the operation, and no result is stored.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	∆	∆	∆	∆	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if R7 = 1 as a result of operation; else cleared.
Z: Set if (A) – (B) = $00; else cleared.
V: Set if operation causes two’s complement overflow; else cleared.
C: Set if operation requires a borrow; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	371B	—	2

CLR Clear a Byte in Memory CLR

Operation: $00 ⇒ M

Description: Content of a memory byte is cleared to zero.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	0	1	0	0	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Cleared.
Z:	Set.
V:	Cleared.
C:	Cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	05	ff	4
IND8, Y	15	ff	4
IND8, Z	25	ff	4
IND16, X	1705	gggg	6
IND16, Y	1715	gggg	6
IND16, Z	1725	gggg	6
EXT	1735	hhll	6

CLRA Clear A CLRA

Operation: $00 ⇒ A

Description: Content of accumulator A is cleared to zero.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	0	1	0	0	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Cleared.
Z:	Set.
V:	Cleared.
C:	Cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3705	—	2

CLRB Clear B CLRB

Operation: $00 ⇒ B

Description: Content of accumulator B is cleared to zero.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	0	1	0	0	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Cleared.
Z: Set.
V: Cleared.
C: Cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3715	—	2

CLRD Clear D CLRD

Operation: $0000 ⇒ D

Description: Content of accumulator D is cleared to zero.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	0	1	0	0	–			–	–			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Cleared.
Z:	Set.
V:	Cleared.
C:	Cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27F5	—	2

CLRE Clear E CLRE

Operation: $0000 ⇒ E

Description: Content of accumulator E is cleared to zero.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	0	1	0	0	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Cleared.
Z:	Set.
V:	Cleared.
C:	Cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	2775	—	2

CLRM Clear AM CLRM

Operation: $000000000 ⇒ AM[35:0]

Description: Content of MAC accumulator is cleared to zero. See **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	0	–	0	–	–	–	–	–			–	–			

S:	Not affected.
MV:	Cleared.
H:	Not affected.
EV:	Cleared.
N:	Not affected.
Z:	Not affected.
V:	Not affected.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27B7	—	2

CLRW Clear a Word in Memory CLRW

Operation: $0000 ⇒ M : M + 1

Description: Content of a memory word is cleared to zero.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	0	1	0	0	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Cleared.
Z:	Set.
V:	Cleared.
C:	Cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND16, X	2705	gggg	6
IND16, Y	2715	gggg	6
IND16, Z	2725	gggg	6
EXT	2735	hhll	6

Compare A

Operation: (A) – (M)

Description: Subtracts content of a memory byte from content of accumulator A and sets condition code register bits. Accumulator and memory contents are not changed, and no result is stored.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
–	–	–	–	∆	∆	∆	∆		–		–		–		

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if R7 = 1 as a result of operation; else cleared.
Z:	Set if (A) – (M) = $00; else cleared.
V:	Set if operation causes two’s complement overflow; else cleared.
C:	Set if operation requires a borrow; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	48	ff	6
IND8, Y	58	ff	6
IND8, Z	68	ff	6
IMM8	78	ii	2
IND16, X	1748	gggg	6
IND16, Y	1758	gggg	6

CMPB Compare B CMPB

Operation: (B) – (M)

Description: Subtracts content of a memory byte from content of accumulator B and sets condition code register bits. Accumulator and memory contents are not changed, and no result is stored.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
–	–	–	–	Δ	Δ	Δ	Δ		–		–		–		

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if R7 = 1 as a result of operation; else cleared.
Z: Set if (B) – (M) = $00; else cleared.
V: Set if operation causes two's complement overflow; else cleared.
C: Set if operation requires a borrow; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	C8	ff	6
IND8, Y	D8	ff	6
IND8, Z	E8	ff	6
IMM8	F8	ii	2
IND16, X	17C8	gggg	6
IND16, Y	17D8	gggg	6
IND16, Z	17E8	gggg	6
EXT	17F8	hhll	6
E, X	27C8	—	6
E, Y	27D8	—	6
E, Z	27E8	—	6

COM One's Complement Byte COM

Operation: $\$FF - (M) \Rightarrow M$, or $\overline{M} \Rightarrow M$

Description: Replaces content of a memory byte with its one's complement. Only BEQ and BNE branches will perform consistently immediately after COM on unsigned values. All signed branches are available after COM on two's complement values.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
–	–	–	–	Δ	Δ	0	1		–		–		–		

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M7 is set; else cleared.
Z: Set if (M) = $00 as a result of operation; else cleared.
V: Cleared.
C: Set.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	00	ff	8
IND8, Y	10	ff	8
IND8, Z	20	ff	8
IND16, X	1700	gggg	8
IND16, Y	1710	gggg	8
IND16, Z	1720	gggg	8
EXT	1730	hhll	8

COMA One's Complement A COMA

Operation: $FF – (A) ⇒ A, or $\overline{\text{M}}$ ⇒ A

Description: Replaces content of accumulator A with its one's complement. Only BEQ and BNE branches will perform consistently immediately after COMA on an unsigned value. All signed branches are available after COMA on a two's complement value.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	∆	∆	0	1	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 = 1 as a result of operation; else cleared.
Z: Set if (A) = $00 as a result of operation; else cleared.
V: Cleared.
C: Set.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3700	—	2

COMB One's Complement B COMB

Operation: $\$FF - (B) \Rightarrow B$, or $\overline{B} \Rightarrow B$

Description: Replaces content of accumulator B with its one's complement. Only BEQ and BNE branches will perform consistently immediately after COMB on an unsigned value. All signed branches are available after COMB on a two's complement value.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	0	1	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if B7 = 1 as a result of operation; else cleared.
Z: Set if (B) = $00 as a result of operation; else cleared.
V: Cleared.
C: Set.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3710	—	2

COMD One's Complement D COMD

Operation: $\$FFFF - (D) \Rightarrow D$, or $\overline{D} \Rightarrow D$

Description: Replaces content of accumulator D with its one's complement. Only BEQ and BNE branches will perform consistently immediately after COMD on an unsigned value. All signed branches are available after COMD on a two's complement value.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	0	1	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if D15 = 1 as a result of operation; else cleared.
Z: Set if (D) = \$0000 as a result of operation; else cleared.
V: Cleared.
C: Set.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27F0	—	2

COME One's Complement E COME

Operation: $\$FFFF - (E) \Rightarrow E$, or $\overline{E} \Rightarrow E$

Description: Replaces content of accumulator E with its one's complement. Only BEQ and BNE branches will perform consistently immediately after COME on an unsigned value. All signed branches are available after COME on a two's complement value.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	∆	∆	0	1	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if E15 = 1 as a result of operation; else cleared.
Z: Set if (E) = \$0000 as a result of operation; else cleared.
V: Cleared.
C: Set.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	2770	—	2

COMW One's Complement Word COMW

Operation: $\$FFFF - (M : M + 1) \Rightarrow M : M + 1$, or
$(\overline{M : M} + 1) \Rightarrow M : M + 1$

Description: Replaces content of a memory word with its one's complement. Only BEQ and BNE branches will perform consistently immediately after COMW on unsigned values. All signed branches are available after COMW on two's complement values.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	∆	∆	0	1	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M15 is set; else cleared.
Z: Set if (M : M + 1) = $0000 as a result of operation; else cleared.
V: Cleared.
C: Set.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND16, X	2700	gggg	8
IND16, Y	2710	gggg	8
IND16, Z	2720	gggg	8
EXT	2730	hhll	8

CPD Compare D CPD

Operation: (D) – (M : M + 1)

Description: Subtracts content of a memory word from content of accumulator D and sets condition code register bits. Accumulator and memory contents are not changed, and no result is stored.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
–	–	–	–	Δ	Δ	Δ	Δ		–		–		–		

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if R15 = 1 as a result of operation; else cleared.
Z: Set if (D) – (M) = $0000; else cleared.
V: Set if operation causes two’s complement overflow; else cleared.
C: Set if operation requires a borrow; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	88	ff	6
IND8, Y	98	ff	6
IND8, Z	A8	ff	6
IMM16	37B8	jjkk	4
IND16, X	37C8	gggg	6
IND16, Y	37D8	gggg	6
IND16, Z	37E8	gggg	6
EXT	37F8	hhll	6
E, X	2788	—	6
E, Y	2798	—	6
E, Z	27A8	—	6

CPE Compare E CPE

Operation: (E) – (M : M + 1)

Description: Subtracts content of a memory word from content of accumulator E and sets condition code register bits. Accumulator and memory contents are not changed, and no result is stored.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	∆	∆	∆	∆	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if R15 = 1 as a result of operation; else cleared.
Z: Set if (E) – (M) = $0000; else cleared.
V: Set if operation causes two's complement overflow; else cleared.
C: Set if operation requires a borrow; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IMM16	3738	jjkk	4
IND16, X	3748	gggg	6
IND16, Y	3758	gggg	6
IND16, Z	3768	gggg	6
EXT	3778	hhll	6

CPS Compare Stack Pointer CPS

Operation: (SP) – (M : M + 1)

Description: Subtracts content of a memory word from content of the stack pointer and sets condition code register bits. SP and memory contents are not changed, and no result is stored.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	∆	∆	∆	∆	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if R15 = 1 as a result of operation; else cleared.
Z: Set if (SP) – (M) = $0000; else cleared.
V: Set if operation causes two’s complement overflow; else cleared.
C: Set if operation requires a borrow; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	4F	ff	6
IND8, Y	5F	ff	6
IND8, Z	6F	ff	6
IMM16	377F	jjkk	4
IND16, X	174F	gggg	6
IND16, Y	175F	gggg	6
IND16, Z	176F	gggg	6
EXT	177F	hhll	6

CPX Compare IX CPX

Operation: (IX) – (M : M + 1)

Description: Subtracts content of a memory word from content of index register X and sets condition code register bits. IX and memory contents are not changed, and no result is stored.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	Δ	Δ	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if R15 = 1 as a result of operation; else cleared.
Z: Set if (IX) – (M) = $0000; else cleared.
V: Set if operation causes two's complement overflow; else cleared.
C: Set if operation requires a borrow; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	4C	ff	6
IND8, Y	5C	ff	6
IND8, Z	6C	ff	6
IMM16	377C	jjkk	4
IND16, X	174C	gggg	6
IND16, Y	175C	gggg	6
IND16, Z	176C	gggg	6
EXT	177C	hhll	6

CPY Compare IY CPY

Operation: (IY) – (M : M + 1)

Description: Subtracts content of a memory word from content of index register Y and sets condition code register bits. IY and memory contents are not changed, and no result is stored.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7 6 5	4	3 2 1 0
S	MV	H	EV	N	Z	V	C	IP	SM	PK
–	–	–	–	∆	∆	∆	∆	–	–	–

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if R15 = 1 as a result of operation; else cleared.
Z:	Set if (IY) – (M) = $0000; else cleared.
V:	Set if operation causes two’s complement overflow; else cleared.
C:	Set if operation requires a borrow; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	4D	ff	6
IND8, Y	5D	ff	6
IND8, Z	6D	ff	6
IMM16	377D	jjkk	4
IND16, X	174D	gggg	6
IND16, Y	175D	gggg	6
IND16, Z	176D	gggg	6
EXT	177D	hhll	6

CPZ Compare IZ CPZ

Operation: (IZ) – (M : M + 1)

Description: Subtracts content of a memory word from content of index register Z and sets condition code register bits. IZ and memory contents are not changed, and no result is stored.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM			PK	
—	—	—	—	Δ	Δ	Δ	Δ		—		—			—	

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if R15 = 1 as a result of operation; else cleared.
Z: Set if (IZ) – (M) = $0000; else cleared.
V: Set if operation causes two's complement overflow; else cleared.
C: Set if operation requires a borrow; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	4E	ff	6
IND8, Y	5E	ff	6
IND8, Z	6E	ff	6
IMM16	377E	jjkk	4
IND16, X	174E	gggg	6
IND16, Y	175E	gggg	6
IND16, Z	176E	gggg	6
EXT	177E	hhll	6

DAA Decimal Adjust A DAA

Operation: $(A)_{10}$

Description: Adjusts the content of accumulator A and the state of the CCR carry bit after binary-coded decimal operations, so that there is a correct BCD sum and an accurate carry indication. The state of the CCR half carry bit affects operation. **Table 6-18** shows details of operation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	U	Δ	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 = 1 as a result of operation; else cleared.
Z: Set if (A) = $00 as a result of operation; else cleared.
V: Undefined.
C: See **Table 6-18**.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3721	—	2

Table 6-18 DAA Function Summary

1	2	3	4	5	6
Initial C Bit Value	Value of A[7:4]	Initial H Bit Value	Value of A[3:0]	Correction Factor	Corrected C Bit Value
0	0 – 9	0	0 – 9	00	0
0	0 – 8	0	A – F	06	0
0	0 – 9	1	0 – 3	06	0
0	A – F	0	0 – 9	60	1
0	9 – F	0	A – F	66	1
0	A – F	1	0 – 3	66	1
1	0 – 2	0	0 – 9	60	1
1	0 – 2	0	A – F	66	1
1	0 – 3	1	0 – 3	66	1

The table shows DAA operation for all legal combinations of input operands. Columns 1 through 4 represent the results of ABA, ADC, or ADD operations on BCD operands. The correction factor in column 5 is added to the accumulator to restore the result of an operation on two BCD operands to a valid BCD value, and to set or clear the C bit. All values are in hexadecimal.

DEC Decrement Byte DEC

Operation: (M) – $01 ⇒ M

Description: Subtracts $01 from the content of a memory byte. Only BEQ and BNE branches will perform consistently immediately after DEC on unsigned values. All signed branches are available after DEC on two's complement values. Because DEC does not affect the C bit in the condition code register, it can be used to implement a loop counter in multiple-precision computation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	Δ	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M7 = 1 as a result of operation; else cleared.
Z: Set if (M) = $00 as a result of operation; else cleared.
V: Set if (M) = $80 before operation (operation causes two's complement overflow); else cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	01	ff	8
IND8, Y	11	ff	8
IND8, Z	21	ff	8
IND16, X	1701	gggg	8
IND16, Y	1711	gggg	8
IND16, Z	1721	gggg	8
EXT	1731	hhll	8

DECA Decrement A DECA

Operation: (A) – $01 ⇒ A

Description: Subtracts $01 from the content of accumulator A. Only BEQ and BNE branches will perform consistently immediately after DECA on unsigned values. All signed branches are available after DECA on two’s complement values. Because DECA does not affect the C bit in the condition code register, it can be used to implement a loop counter in multiple-precision computation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	Δ	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 = 1 as a result of operation; else cleared.
Z: Set if (A) = $00 as a result of operation; else cleared.
V: Set if (A) = $80 before operation (operation causes two’s complement overflow); else cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3701	—	2

DECW Decrement Word DECW

Operation: (M : M + 1) – $0001 ⇒ M : M + 1

Description: Subtracts $0001 from the content of a memory word. Only BEQ and BNE branches will perform consistently immediately after DECW on unsigned values. All signed branches are available after DECW on two's complement values. Because DECW does not affect the C bit in the condition code register, it can be used to implement a loop counter in multiple-precision computation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	Δ	—	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M : M + 1[15] = 1 as a result of operation; else cleared.
Z: Set if (M : M + 1) = $0000 as a result of operation; else cleared.
V: Set if (M : M + 1) = $8000 before operation (operation causes two's complement overflow); else cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND16, X	2701	gggg	8
IND16, Y	2711	gggg	8
IND16, Z	2721	gggg	8
EXT	2731	hhll	8

EDIV Extended Unsigned Integer Divide EDIV

Operation: (E : D) / (IX) ⇒ IX
Remainder ⇒ D

Description: Divides a 32-bit unsigned dividend contained in concatenated accumulators E and D by a 16-bit divisor contained in index register X. The quotient is placed in IX and the remainder in D. There is an implied radix point to the right of the quotient (IX0). An implied radix point is assumed to occupy the same position in both dividend and divisor.

The states of condition code register bits N, Z, V, and C are undefined after division by zero, but accumulator contents are not changed. Division by zero causes an exception. See **SECTION 9 EXCEPTION PROCESSING** for more information. The states of the N, Z, and C bits are also undefined after overflow.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM			PK	
–	–	–	–	∆	∆	∆	∆		–		–			–	

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if IX15 = 1 as a result of operation; else cleared. Undefined after overflow or division by zero.
Z: Set if (IX) = $0000 as a result of operation; else cleared. Undefined after overflow or division by zero.
V: Set if (IX) > $FFFF as a result of operation; else cleared. Undefined after division by zero.
C: Set if 2 ∗ Remainder ≥ Divisor; else cleared. Undefined after overflow or division by zero.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3728	—	24

EDIVS Extended Signed Integer Divide EDIVS

Operation: (E : D) / (IX) ⇒ IX
Remainder ⇒ D

Description: Divides a 32-bit signed dividend contained in concatenated accumulators E and D by a 16-bit divisor contained in index register X. The quotient is placed in IX and the remainder in D. There is an implied radix point to the right of IX0. Implied radix points in dividend and divisor must occupy the same bit position.

The states of condition code register bits N, Z, and C are undefined after overflow. The states of bits N, Z, V, and C are undefined after division by zero, but accumulator contents are not changed. Division by zero causes an exception. See **SECTION 9 EXCEPTION PROCESSING** for more information.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM			PK	
–	–	–	–	Δ	Δ	Δ	Δ		–		–			–	

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if IX15 = 1 as a result of operation; else cleared. Undefined after overflow or division by zero.
Z: Set if (IX) = $0000 as a result of operation; else cleared. Undefined after overflow or division by zero.
V: Set if (IX) > $7FFF for a positive quotient or if (IX) > $8000 for a negative quotient as a result of operation; else cleared. Undefined after division by zero.
C: Set if $|2 * \text{Remainder}| \geq |\text{Divisor}|$; else cleared. Undefined after overflow or division by zero.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3729	—	38

EMUL Extended Unsigned Multiply EMUL

Operation: $(E) * (D) \Rightarrow E : D$

Description: Multiplies a 16-bit unsigned multiplicand contained in accumulator E by a 16-bit unsigned multiplier contained in accumulator D, then places the product in concatenated accumulators E and D. The CCR carry bit can be used to round the high word of the product — execute EMUL, then ADCE #0.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	–	Δ	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if E15 = 1 as a result of operation; else cleared.
Z: Set if (E : D) = $00000000 as a result of operation; else cleared.
V: Not affected.
C: Set if D15 = 1 as a result of operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3725	—	10

EMULS Extended Signed Multiply EMULS

Operation: (E) ∗ (D) ⇒ E : D

Description: Multiplies a 16-bit signed multiplicand contained in accumulator E by a 16-bit signed multiplier contained in accumulator D, then places the product in concatenated accumulators E and D. The CCR carry bit can be used to round the high word of the product — execute EMULS, then ADCE #0.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
–	–	–	–	Δ	Δ	–	Δ		–		–		–		

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if E15 = 1 as a result of operation; else cleared.
Z:	Set if (E : D) = $00000000 as a result of operation; else cleared.
V:	Not affected.
C:	Set if D15 = 1 as a result of operation; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3726	—	8

EORA Exclusive OR A EORA

Operation: (A) ⊕ (M) ⇒ A

Description: Performs EOR between the content of accumulator A and a memory byte, then places the result in accumulator A. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	0	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 is set by operation; else cleared.
Z: Set if (A) = $00 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	44	ff	6
IND8, Y	54	ff	6
IND8, Z	64	ff	6
IMM8	74	ii	2
IND16, X	1744	gggg	6
IND16, Y	1754	gggg	6
IND16, Z	1764	gggg	6
EXT	1774	hhll	6
E, X	2744	—	6
E, Y	2754	—	6
E, Z	2764	—	6

EORB Exclusive OR B EORB

Operation: (B) ⊕ (M) ⇒ B

Description: Performs EOR between the content of accumulator B and a memory byte, then places the result in accumulator B. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	∆	∆	0	–	–			–	–			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if B7 is set by operation; else cleared.
Z:	Set if (B) = $00 as a result of operation; else cleared.
V:	Cleared.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	C4	ff	6
IND8, Y	D4	ff	6
IND8, Z	E4	ff	6
IMM8	F4	ii	2
IND16, X	17C4	gggg	6
IND16, Y	17D4	gggg	6
IND16, Z	17E4	gggg	6
EXT	17F4	hhll	6
E, X	27C4	—	6
E, Y	27D4	—	6
E, Z	27E4	—	6

EORD

Exclusive OR D

Operation: (D) ⊕ (M : M + 1) ⇒ D

Description: Performs EOR between the content of accumulator D and a memory word, then places the result in accumulator D. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	∆	∆	0	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if D15 is set by operation; else cleared.
Z: Set if (D) = $0000 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	84	ff	6
IND8, Y	94	ff	6
IND8, Z	A4	ff	6
IMM16	37B4	jjkk	4
IND16, X	37C4	gggg	6
IND16, Y	37D4	gggg	6
IND16, Z	37E4	gggg	6
EXT	37F4	hhll	6
E, X	2784	—	6
E, Y	2794	—	6
E, Z	27A4	—	6

EORE Exclusive OR E EORE

Operation: $(E) \oplus (M : M + 1) \Rightarrow E$

Description: Performs EOR between the content of accumulator E and a memory word, then places the result in accumulator E. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	∆	∆	0	—	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if E15 is set by operation; else cleared.
Z: Set if (E) = $0000 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IMM16	3734	jjkk	4
IND16, X	3744	gggg	6
IND16, Y	3754	gggg	6
IND16, Z	3764	gggg	6
EXT	3774	hhll	6

FDIV Unsigned Fractional Divide FDIV

Operation: (D) / (IX) ⇒ IX
Remainder ⇒ D

Description: Divides a 16-bit unsigned dividend contained in accumulator D by a 16-bit unsigned divisor contained in index register X. The quotient is placed in IX and the remainder is placed in D.

There is an implied radix point to the left of the quotient (IX15). An implied radix point is assumed to occupy the same position in both dividend and divisor. If the dividend is greater than or equal to the divisor, or if the divisor is equal to zero, (IX) is set to $FFFF and (D) is indeterminate. To maintain compatibility with the M68HC11, no exception is generated on overflow or division by zero.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
—	—	—	—	—	∆	∆	∆		—		—		—		

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Not affected.
Z: Set if (IX) = $0000 as a result of operation; else cleared.
V: Set if (IX) ≤ (D) before operation; else cleared.
C: Set if (IX) = $0000 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	372B	—	22

FMULS Signed Fractional Multiply FMULS

Operation: $(E) * (D) \Rightarrow E : D[31:1]$
$0 \Rightarrow E : D[0]$

Description: Multiplies a 16-bit signed fractional multiplicand contained in accumulator E by a 16-bit signed fractional multiplier contained in accumulator D. The implied radix points are between bits 15 and 14 of the accumulators. The product is left-shifted one place to align the radix point between bits 31 and 30, then placed in bits 31 to 1 of concatenated accumulators E and D. D0 is cleared. The CCR carry bit can be used to round the high word of the product — execute FMULS, then ADCE #0.

When both accumulators contain $8000 (–1), the product is $80000000 (–1.0) and the CCR V bit is set.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	Δ	Δ	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if E15 = 1 as a result of operation; else cleared.
Z: Set if (E : D) = $00000000 as a result of operation; else cleared.
V: Set when operation is $(–1)^2$; else cleared.
C: Set if D15 = 1 as a result of operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3727	—	8

IDIV Integer Divide IDIV

Operation: (D) / (IX) ⇒ IX
Remainder ⇒ D

Description: Divides a 16-bit unsigned dividend contained in accumulator D by a 16-bit unsigned divisor contained in index register X. The quotient is placed in IX and the remainder is placed in D.

There is an implied radix point to the right of the quotient (IX0). An implied radix point is assumed to occupy the same position in both dividend and divisor. If the divisor is equal to zero, (IX) is set to $FFFF and (D) is indeterminate. To maintain compatibility with the M68HC11, no exception is generated on division by zero.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	–	∆	0	∆	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Not affected.
Z: Set if (IX) = $0000 as a result of operation; else cleared.
V: Cleared.
C: Set if (IX) = $0000 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	372A	—	22

INC Increment Byte INC

Operation: (M) + $01 ⇒ M

Description: Adds $01 to the content of a memory byte. Only BEQ and BNE branches will perform consistently immediately after INC on unsigned values. All signed branches are available after INC on two's complement values. Because INC does not affect the C bit in the condition code register, it can be used to implement a loop counter in multiple-precision computation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	Δ	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M7 = 1 as a result of operation; else cleared.
Z: Set if (M) = $00 as a result of operation; else cleared.
V: Set if (M) = $7F before operation (operation causes two's complement overflow); else cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	03	ff	8
IND8, Y	13	ff	8
IND8, Z	23	ff	8
IND16, X	1703	gggg	8
IND16, Y	1713	gggg	8
IND16, Z	1723	gggg	8
EXT	1733	hhll	8

INCA Increment A INCA

Operation: (A) + $01 ⇒ A

Description: Adds $01 to the content of accumulator A. Only BEQ and BNE branches will perform consistently immediately after INCA on unsigned values. All signed branches are available after INCA on two's complement values. Because INCA does not affect the C bit in the condition code register, it can be used to implement a loop counter in multiple-precision computation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	Δ	—	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 = 1 as a result of operation; else cleared.
Z: Set if (A) = $00 as a result of operation; else cleared.
V: Set if (A) = $7F before operation (operation causes two's complement overflow); else cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3703	—	2

INCB Increment B INCB

Operation: (B) + $01 ⇒ B

Description: Adds $01 to the content of accumulator B. Only BEQ and BNE branches will perform consistently immediately after INCB on unsigned values. All signed branches are available after INCB on two's complement values. Because INCB does not affect the C bit in the condition code register, it can be used to implement a loop counter in multiple-precision computation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	Δ	–	–			–	–			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if B7 = 1 as a result of operation; else cleared.
Z:	Set if (B) = $00 as a result of operation; else cleared.
V:	Set if (B) = $7F before operation (operation causes two's complement overflow); else cleared.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3713	—	2

INCW Increment Word INCW

Operation: (M : M + 1) + $0001 ⇒ M : M + 1

Description: Adds $0001 to the content of a memory word. Only BEQ and BNE branches will perform consistently immediately after INCW on unsigned values. All signed branches are available after INCW on two’s complement values. Because INCW does not affect the C bit in the condition code register, it can be used to implement a loop counter in multiple-precision computation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	∆	∆	∆	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M : M + 1[15] = 1 as a result of operation; else cleared.
Z: Set if (M : M + 1) = $0000 as a result of operation; else cleared.
V: Set if (M : M + 1) = $7FFF before operation (operation causes two’s complement overflow); else cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND16, X	2703	gggg	8
IND16, Y	2713	gggg	8
IND16, Z	2723	gggg	8
EXT	2733	hhll	8

JMP **Jump** JMP

Operation: Effective Address ⇒ PK : PC

Description:

JSR Jump to Subroutine JSR

Operation: Push (PC)
(SK : SP) – \$0002 ⇒ SK : SP
Push (CCR)
(SK : SP) – \$0002 ⇒ SK : SP
Effective Address ⇒ PK : PC

Description: Causes a branch to a subroutine. After the current content of the program counter and the condition code register are stacked, a 20-bit effective address is placed in the concatenated program counter extension field and program counter. The next instruction is fetched from the new address. The effective address can be generated in two ways:

1. Effective Address = Extension: 16-bit Extended Address

   When extended addressing mode is employed, the effective address is formed by a zero-extended 4-bit right-justified address extension and a 16-bit extended address that are both contained in the instruction. The EK field is not changed.

2. Effective Address = \$0 : (index register) + 0 : 20-bit Offset

   When indexed addressing mode is employed, the effective address is calculated by adding a zero-extended 20-bit signed offset to the zero-extended content of an index register. The associated extension field is not changed.

Syntax: JSR (effective address)

JSR (offset)

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
EXT20	FA	zb hh ll	10
IND20, X	89	zg gggg	12
IND20, Y	99	zg gggg	12
IND20, Z	A9	zg gggg	12

LBCC Long Branch If Carry Clear LBCC

Operation: If $C = 0$, then $(PK : PC) + \text{Offset} \Rightarrow PK : PC$

Description: Causes a long program branch if the CCR carry bit has a value of zero. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement simple or unsigned conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	3784	rrrr	6, 4

Table 6-19 Branch Instruction Summary (16-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
LBCC	3784	$C = 0$	Simple, Unsigned	LBCS
LBCS	3785	$C = 1$	Simple, Unsigned	LBCC
LBEQ	3787	$Z = 1$	Simple, Unsigned, Signed	LBNE
LBGE	378C	$N \oplus V = 0$	Signed	LBLT
LBGT	378E	$Z + (N \oplus V) = 0$	Signed	LBLE
LBHI	3782	$C + Z = 0$	Unsigned	LBLS
LBLE	378F	$Z + (N \oplus V) = 1$	Signed	LBGT
LBLS	3783	$C + Z = 1$	Unsigned	LBHI
LBLT	378D	$N \oplus V = 1$	Signed	LBGE
LBMI	378B	$N = 1$	Simple	LBPL
LBNE	3786	$Z = 0$	Simple, Unsigned, Signed	LBEQ
LBPL	378A	$N = 0$	Simple	LBMI
LBRA	3780	1	Unary	LBRN
LBRN	3781	0	Unary	LBRA
LBVC	3788	$V = 0$	Simple	LBVS
LBVS	3789	$V = 1$	Simple	LBVC

LBCS Long Branch If Carry Set LBCS

Operation: If $C = 1$, then $(PK : PC) + \text{Offset} \Rightarrow PK : PC$

Description: Causes a long program branch if the CCR carry bit has a value of one. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement simple or unsigned conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	3785	rrrr	6, 4

Table 6-20 Branch Instruction Summary (16-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
LBCC	3784	$C = 0$	Simple, Unsigned	LBCS
LBCS	3785	$C = 1$	Simple, Unsigned	LBCC
LBEQ	3787	$Z = 1$	Simple, Unsigned, Signed	LBNE
LBGE	378C	$N \oplus V = 0$	Signed	LBLT
LBGT	378E	Z ✢ (N ⊕ V) = 0	Signed	LBLE
LBHI	3782	C ✢ Z = 0	Unsigned	LBLS
LBLE	378F	Z ✢ (N ⊕ V) = 1	Signed	LBGT
LBLS	3783	C ✢ Z = 1	Unsigned	LBHI
LBLT	378D	$N \oplus V = 1$	Signed	LBGE
LBMI	378B	$N = 1$	Simple	LBPL
LBNE	3786	$Z = 0$	Simple, Unsigned, Signed	LBEQ
LBPL	378A	$N = 0$	Simple	LBMI
LBRA	3780	1	Unary	LBRN
LBRN	3781	0	Unary	LBRA
LBVC	3788	$V = 0$	Simple	LBVS
LBVS	3789	$V = 1$	Simple	LBVC

LBEQ Long Branch If Equal to Zero LBEQ

Operation: If $Z = 1$, then $(PK : PC) + \text{Offset} \Rightarrow PK : PC$

Description: Causes a long program branch if the CCR zero bit has a value of one. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement simple, signed, or unsigned conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	3787	rrrr	6, 4

Table 6-21 Branch Instruction Summary (16-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
LBCC	3784	$C = 0$	Simple, Unsigned	LBCS
LBCS	3785	$C = 1$	Simple, Unsigned	LBCC
LBEQ	3787	$Z = 1$	Simple, Unsigned, Signed	LBNE
LBGE	378C	$N \oplus V = 0$	Signed	LBLT
LBGT	378E	$Z + (N \oplus V) = 0$	Signed	LBLE
LBHI	3782	$C + Z = 0$	Unsigned	LBLS
LBLE	378F	$Z + (N \oplus V) = 1$	Signed	LBGT
LBLS	3783	$C + Z = 1$	Unsigned	LBHI
LBLT	378D	$N \oplus V = 1$	Signed	LBGE
LBMI	378B	$N = 1$	Simple	LBPL
LBNE	3786	$Z = 0$	Simple, Unsigned, Signed	LBEQ
LBPL	378A	$N = 0$	Simple	LBMI
LBRA	3780	1	Unary	LBRN
LBRN	3781	0	Unary	LBRA
LBVC	3788	$V = 0$	Simple	LBVS
LBVS	3789	$V = 1$	Simple	LBVC

LBEV Long Branch If EV Set LBEV

Operation: If EV = 1, then (PK : PC) + Offset ⇒ PK : PC

Description: Causes a long program branch if the EV bit in the condition code register has a value of one. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. See **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	3791	rrrr	6, 4

LBGE Long Branch If Greater than or Equal to Zero LBGE

Operation: If $N \oplus V = 0$, then $(PK : PC) + \text{Offset} \Rightarrow PK : PC$

Description: Causes a long program branch if the CCR negative and overflow bits both have a value of zero or both have a value of one. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement signed conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	378C	rrrr	6, 4

Table 6-22 Branch Instruction Summary (16-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
LBCC	3784	C = 0	Simple, Unsigned	LBCS
LBCS	3785	C = 1	Simple, Unsigned	LBCC
LBEQ	3787	Z = 1	Simple, Unsigned, Signed	LBNE
LBGE	378C	N ⊕ V = 0	Signed	LBLT
LBGT	378E	Z ✢ (N ⊕ V) = 0	Signed	LBLE
LBHI	3782	C ✢ Z = 0	Unsigned	LBLS
LBLE	378F	Z ✢ (N ⊕ V) = 1	Signed	LBGT
LBLS	3783	C ✢ Z = 1	Unsigned	LBHI
LBLT	378D	N ⊕ V = 1	Signed	LBGE
LBMI	378B	N = 1	Simple	LBPL
LBNE	3786	Z = 0	Simple, Unsigned, Signed	LBEQ
LBPL	378A	N = 0	Simple	LBMI
LBRA	3780	1	Unary	LBRN
LBRN	3781	0	Unary	LBRA
LBVC	3788	V = 0	Simple	LBVS
LBVS	3789	V = 1	Simple	LBVC

LBGT Long Branch If Greater than Zero LBGT

Operation: If Z ✢ (N ⊕ V) = 0, then (PK : PC) + Offset ⇒ PK : PC

Description: Causes a long program branch if the CCR negative and overflow bits both have a value of zero or both have a value of one, and the CCR zero bit has a value of zero. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement signed conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	378E	rrrr	6, 4

Table 6-23 Branch Instruction Summary (16-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
LBCC	3784	C = 0	Simple, Unsigned	LBCS
LBCS	3785	C = 1	Simple, Unsigned	LBCC
LBEQ	3787	Z = 1	Simple, Unsigned, Signed	LBNE
LBGE	378C	N ⊕ V = 0	Signed	LBLT
LBGT	378E	Z ✢ (N ⊕ V) = 0	Signed	LBLE
LBHI	3782	C ✢ Z = 0	Unsigned	LBLS
LBLE	378F	Z ✢ (N ⊕ V) = 1	Signed	LBGT
LBLS	3783	C ✢ Z = 1	Unsigned	LBHI
LBLT	378D	N ⊕ V = 1	Signed	LBGE
LBMI	378B	N = 1	Simple	LBPL
LBNE	3786	Z = 0	Simple, Unsigned, Signed	LBEQ
LBPL	378A	N = 0	Simple	LBMI
LBRA	3780	1	Unary	LBRN
LBRN	3781	0	Unary	LBRA
LBVC	3788	V = 0	Simple	LBVS
LBVS	3789	V = 1	Simple	LBVC

LBHI Long Branch If Higher LBHI

Operation: If C ✣ Z = 0, then (PK : PC) + Offset ⇒ PK : PC

Description: Causes a long program branch if the CCR carry and zero bits both have a value of zero. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement unsigned conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	3782	rrrr	6, 4

Table 6-24 Branch Instruction Summary (16-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
LBCC	3784	C = 0	Simple, Unsigned	LBCS
LBCS	3785	C = 1	Simple, Unsigned	LBCC
LBEQ	3787	Z = 1	Simple, Unsigned, Signed	LBNE
LBGE	378C	N ⊕ V = 0	Signed	LBLT
LBGT	378E	Z ✣ (N ⊕ V) = 0	Signed	LBLE
LBHI	3782	C ✣ Z = 0	Unsigned	LBLS
LBLE	378F	Z ✣ (N ⊕ V) = 1	Signed	LBGT
LBLS	3783	C ✣ Z = 1	Unsigned	LBHI
LBLT	378D	N ⊕ V = 1	Signed	LBGE
LBMI	378B	N = 1	Simple	LBPL
LBNE	3786	Z = 0	Simple, Unsigned, Signed	LBEQ
LBPL	378A	N = 0	Simple	LBMI
LBRA	3780	1	Unary	LBRN
LBRN	3781	0	Unary	LBRA
LBVC	3788	V = 0	Simple	LBVS
LBVS	3789	V = 1	Simple	LBVC

LBLE Long Branch If Less than or Equal to Zero LBLE

Operation: If Z ✢ (N ⊕ V) = 1, then (PK : PC) + Offset ⇒ PK : PC

Description: Causes a long program branch if either the CCR negative bit or overflow bit has a value of one, or the CCR zero bit has a value of one. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement signed conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	378F	rrrr	6, 4

Table 6-25 Branch Instruction Summary (16-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
LBCC	3784	C = 0	Simple, Unsigned	LBCS
LBCS	3785	C = 1	Simple, Unsigned	LBCC
LBEQ	3787	Z = 1	Simple, Unsigned, Signed	LBNE
LBGE	378C	N ⊕ V = 0	Signed	LBLT
LBGT	378E	Z ✢ (N ⊕ V) = 0	Signed	LBLE
LBHI	3782	C ✢ Z = 0	Unsigned	LBLS
LBLE	378F	Z ✢ (N ⊕ V) = 1	Signed	LBGT
LBLS	3783	C ✢ Z = 1	Unsigned	LBHI
LBLT	378D	N ⊕ V = 1	Signed	LBGE
LBMI	378B	N = 1	Simple	LBPL
LBNE	3786	Z = 0	Simple, Unsigned, Signed	LBEQ
LBPL	378A	N = 0	Simple	LBMI
LBRA	3780	1	Unary	LBRN
LBRN	3781	0	Unary	LBRA
LBVC	3788	V = 0	Simple	LBVS
LBVS	3789	V = 1	Simple	LBVC

LBLS Long Branch If Lower or Same LBLS

Operation: If C ✣ Z = 1, then (PK : PC) + Offset ⇒ PK : PC

Description: Causes a long program branch if either or both the CCR carry and zero bits have a value of one. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement unsigned conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	3783	rrrr	6, 4

Table 6-26 Branch Instruction Summary (16-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
LBCC	3784	C = 0	Simple, Unsigned	LBCS
LBCS	3785	C = 1	Simple, Unsigned	LBCC
LBEQ	3787	Z = 1	Simple, Unsigned, Signed	LBNE
LBGE	378C	N ⊕ V = 0	Signed	LBLT
LBGT	378E	Z ✣ (N ⊕ V) = 0	Signed	LBLE
LBHI	3782	C ✣ Z = 0	Unsigned	LBLS
LBLE	378F	Z ✣ (N ⊕ V) = 1	Signed	LBGT
LBLS	3783	C ✣ Z = 1	Unsigned	LBHI
LBLT	378D	N ⊕ V = 1	Signed	LBGE
LBMI	378B	N = 1	Simple	LBPL
LBNE	3786	Z = 0	Simple, Unsigned, Signed	LBEQ
LBPL	378A	N = 0	Simple	LBMI
LBRA	3780	1	Unary	LBRN
LBRN	3781	0	Unary	LBRA
LBVC	3788	V = 0	Simple	LBVS
LBVS	3789	V = 1	Simple	LBVC

LBLT Long Branch If Less than Zero LBLT

Operation: If $N \oplus V = 1$, then $(PK : PC) + \text{Offset} \Rightarrow PK : PC$

Description: Causes a long program branch if either the CCR negative or overflow bits has a value of one. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement signed conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	378D	rrrr	6, 4

Table 6-27 Branch Instruction Summary (16-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
LBCC	3784	$C = 0$	Simple, Unsigned	LBCS
LBCS	3785	$C = 1$	Simple, Unsigned	LBCC
LBEQ	3787	$Z = 1$	Simple, Unsigned, Signed	LBNE
LBGE	378C	$N \oplus V = 0$	Signed	LBLT
LBGT	378E	Z ✛ (N ⊕ V) = 0	Signed	LBLE
LBHI	3782	C ✛ Z = 0	Unsigned	LBLS
LBLE	378F	Z ✛ (N ⊕ V) = 1	Signed	LBGT
LBLS	3783	C ✛ Z = 1	Unsigned	LBHI
LBLT	378D	$N \oplus V = 1$	Signed	LBGE
LBMI	378B	$N = 1$	Simple	LBPL
LBNE	3786	$Z = 0$	Simple, Unsigned, Signed	LBEQ
LBPL	378A	$N = 0$	Simple	LBMI
LBRA	3780	1	Unary	LBRN
LBRN	3781	0	Unary	LBRA
LBVC	3788	$V = 0$	Simple	LBVS
LBVS	3789	$V = 1$	Simple	LBVC

LBMI Long Branch If Minus LBMI

Operation: If N = 1, then (PK : PC) + Offset ⇒ PK : PC

Description: Causes a long program branch if the CCR negative bit has a value of one. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement simple conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	378B	rrrr	6, 4

Table 6-28 Branch Instruction Summary (16-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
LBCC	3784	C = 0	Simple, Unsigned	LBCS
LBCS	3785	C = 1	Simple, Unsigned	LBCC
LBEQ	3787	Z = 1	Simple, Unsigned, Signed	LBNE
LBGE	378C	$N \oplus V = 0$	Signed	LBLT
LBGT	378E	$Z + (N \oplus V) = 0$	Signed	LBLE
LBHI	3782	$C + Z = 0$	Unsigned	LBLS
LBLE	378F	$Z + (N \oplus V) = 1$	Signed	LBGT
LBLS	3783	$C + Z = 1$	Unsigned	LBHI
LBLT	378D	$N \oplus V = 1$	Signed	LBGE
LBMI	378B	N = 1	Simple	LBPL
LBNE	3786	Z = 0	Simple, Unsigned, Signed	LBEQ
LBPL	378A	N = 0	Simple	LBMI
LBRA	3780	1	Unary	LBRN
LBRN	3781	0	Unary	LBRA
LBVC	3788	V = 0	Simple	LBVS
LBVS	3789	V = 1	Simple	LBVC

LBMV Long Branch If MV Set LBMV

Operation: If MV = 1, then (PK : PC) + Offset ⇒ PK : PC

Description: Causes a long program branch if the MV bit in the condition code register has a value of one. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. See **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	3790	rrrr	6, 4

LBNE Long Branch If Not Equal to Zero LBNE

Operation: If Z = 0, then (PK : PC) + Offset ⇒ PK : PC

Description: Causes a long program branch if the CCR zero bit has a value of zero. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement simple, signed, and unsigned conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	3786	rrrr	6, 4

Table 6-29 Branch Instruction Summary (16-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
LBCC	3784	C = 0	Simple, Unsigned	LBCS
LBCS	3785	C = 1	Simple, Unsigned	LBCC
LBEQ	3787	Z = 1	Simple, Unsigned, Signed	LBNE
LBGE	378C	N ⊕ V = 0	Signed	LBLT
LBGT	378E	Z ✢ (N ⊕ V) = 0	Signed	LBLE
LBHI	3782	C ✢ Z = 0	Unsigned	LBLS
LBLE	378F	Z ✢ (N ⊕ V) = 1	Signed	LBGT
LBLS	3783	C ✢ Z = 1	Unsigned	LBHI
LBLT	378D	N ⊕ V = 1	Signed	LBGE
LBMI	378B	N = 1	Simple	LBPL
LBNE	3786	Z = 0	Simple, Unsigned, Signed	LBEQ
LBPL	378A	N = 0	Simple	LBMI
LBRA	3780	1	Unary	LBRN
LBRN	3781	0	Unary	LBRA
LBVC	3788	V = 0	Simple	LBVS
LBVS	3789	V = 1	Simple	LBVC

LBPL Long Branch If Plus LBPL

Operation: If $N = 0$, then $(PK : PC) + \text{Offset} \Rightarrow PK : PC$

Description: Causes a long program branch if the CCR negative bit has a value of zero. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement simple conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	378A	rrrr	6, 4

Table 6-30 Branch Instruction Summary (16-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
LBCC	3784	$C = 0$	Simple, Unsigned	LBCS
LBCS	3785	$C = 1$	Simple, Unsigned	LBCC
LBEQ	3787	$Z = 1$	Simple, Unsigned, Signed	LBNE
LBGE	378C	$N \oplus V = 0$	Signed	LBLT
LBGT	378E	$Z + (N \oplus V) = 0$	Signed	LBLE
LBHI	3782	$C + Z = 0$	Unsigned	LBLS
LBLE	378F	$Z + (N \oplus V) = 1$	Signed	LBGT
LBLS	3783	$C + Z = 1$	Unsigned	LBHI
LBLT	378D	$N \oplus V = 1$	Signed	LBGE
LBMI	378B	$N = 1$	Simple	LBPL
LBNE	3786	$Z = 0$	Simple, Unsigned, Signed	LBEQ
LBPL	378A	$N = 0$	Simple	LBMI
LBRA	3780	1	Unary	LBRN
LBRN	3781	0	Unary	LBRA
LBVC	3788	$V = 0$	Simple	LBVS
LBVS	3789	$V = 1$	Simple	LBVC

LBRA Long Branch Always LBRA

Operation: $(PK : PC) + \text{Offset} \Rightarrow PK : PC$

Description: Causes a long program branch. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	3780	rrrr	6

Table 6-31 Branch Instruction Summary (16-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
LBCC	3784	C = 0	Simple, Unsigned	LBCS
LBCS	3785	C = 1	Simple, Unsigned	LBCC
LBEQ	3787	Z = 1	Simple, Unsigned, Signed	LBNE
LBGE	378C	N ⊕ V = 0	Signed	LBLT
LBGT	378E	Z ✢ (N ⊕ V) = 0	Signed	LBLE
LBHI	3782	C ✢ Z = 0	Unsigned	LBLS
LBLE	378F	Z ✢ (N ⊕ V) = 1	Signed	LBGT
LBLS	3783	C ✢ Z = 1	Unsigned	LBHI
LBLT	378D	N ⊕ V = 1	Signed	LBGE
LBMI	378B	N = 1	Simple	LBPL
LBNE	3786	Z = 0	Simple, Unsigned, Signed	LBEQ
LBPL	378A	N = 0	Simple	LBMI
LBRA	3780	1	Unary	LBRN
LBRN	3781	0	Unary	LBRA
LBVC	3788	V = 0	Simple	LBVS
LBVS	3789	V = 1	Simple	LBVC

LBRN Long Branch Never LBRN

Operation: $(PK : PC) + 4 \Rightarrow PK : PC$

Description: Never branches. This instruction is effectively a NOP that requires three cycles to execute. When the operation causes PC overflow, the PK field is incremented or decremented.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	3781	rrrr	6

Table 6-32 Branch Instruction Summary (16-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
LBCC	3784	$C = 0$	Simple, Unsigned	LBCS
LBCS	3785	$C = 1$	Simple, Unsigned	LBCC
LBEQ	3787	$Z = 1$	Simple, Unsigned, Signed	LBNE
LBGE	378C	$N \oplus V = 0$	Signed	LBLT
LBGT	378E	$Z + (N \oplus V) = 0$	Signed	LBLE
LBHI	3782	$C + Z = 0$	Unsigned	LBLS
LBLE	378F	$Z + (N \oplus V) = 1$	Signed	LBGT
LBLS	3783	$C + Z = 1$	Unsigned	LBHI
LBLT	378D	$N \oplus V = 1$	Signed	LBGE
LBMI	378B	$N = 1$	Simple	LBPL
LBNE	3786	$Z = 0$	Simple, Unsigned, Signed	LBEQ
LBPL	378A	$N = 0$	Simple	LBMI
LBRA	3780	1	Unary	LBRN
LBRN	3781	0	Unary	LBRA
LBVC	3788	$V = 0$	Simple	LBVS
LBVS	3789	$V = 1$	Simple	LBVC

LBSR **Long Branch to Subroutine** LBSR

Operation:

Push (PC)
$(SK : SP) - \$0002 \Rightarrow SK : SP$
Push (CCR)
$(SK : SP) - \$0002 \Rightarrow SK : SP$
$(PK : PC) + \text{Offset} \Rightarrow PK : PC$

Description: Saves current address and flags, then branches to a subroutine. The current value of the program counter is stacked, then the condition code register is stacked (which preserves the PK field as well as condition code bits and the interrupt priority mask). A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	27F9	rrrr	10

LBVC Long Branch If Overflow Clear LBVC

Operation: If $V = 0$, then $(PK : PC) + \text{Offset} \Rightarrow PK : PC$

Description: Causes a long program branch if the CCR overflow bit has a value of zero. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement simple, signed, and unsigned conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	3788	rrrr	6, 4

Table 6-33 Branch Instruction Summary (16-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
LBCC	3784	$C = 0$	Simple, Unsigned	LBCS
LBCS	3785	$C = 1$	Simple, Unsigned	LBCC
LBEQ	3787	$Z = 1$	Simple, Unsigned, Signed	LBNE
LBGE	378C	$N \oplus V = 0$	Signed	LBLT
LBGT	378E	$Z + (N \oplus V) = 0$	Signed	LBLE
LBHI	3782	$C + Z = 0$	Unsigned	LBLS
LBLE	378F	$Z + (N \oplus V) = 1$	Signed	LBGT
LBLS	3783	$C + Z = 1$	Unsigned	LBHI
LBLT	378D	$N \oplus V = 1$	Signed	LBGE
LBMI	378B	$N = 1$	Simple	LBPL
LBNE	3786	$Z = 0$	Simple, Unsigned, Signed	LBEQ
LBPL	378A	$N = 0$	Simple	LBMI
LBRA	3780	1	Unary	LBRN
LBRN	3781	0	Unary	LBRA
LBVC	3788	$V = 0$	Simple	LBVS
LBVS	3789	$V = 1$	Simple	LBVC

LBVS Long Branch If Overflow Set LBVS

Operation: If $V = 1$, then $(PK : PC) + \text{Offset} \Rightarrow PK : PC$

Description: Causes a long program branch if the CCR overflow bit has a value of one. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented. Used to implement simple, signed, and unsigned conditional branches.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
REL16	3789	rrrr	6, 4

Table 6-34 Branch Instruction Summary (16-Bit Offset)

Mnemonic	Opcode	Equation	Type	Complement
LBCC	3784	$C = 0$	Simple, Unsigned	LBCS
LBCS	3785	$C = 1$	Simple, Unsigned	LBCC
LBEQ	3787	$Z = 1$	Simple, Unsigned, Signed	LBNE
LBGE	378C	$N \oplus V = 0$	Signed	LBLT
LBGT	378E	$Z + (N \oplus V) = 0$	Signed	LBLE
LBHI	3782	$C + Z = 0$	Unsigned	LBLS
LBLE	378F	$Z + (N \oplus V) = 1$	Signed	LBGT
LBLS	3783	$C + Z = 1$	Unsigned	LBHI
LBLT	378D	$N \oplus V = 1$	Signed	LBGE
LBMI	378B	$N = 1$	Simple	LBPL
LBNE	3786	$Z = 0$	Simple, Unsigned, Signed	LBEQ
LBPL	378A	$N = 0$	Simple	LBMI
LBRA	3780	1	Unary	LBRN
LBRN	3781	0	Unary	LBRA
LBVC	3788	$V = 0$	Simple	LBVS
LBVS	3789	$V = 1$	Simple	LBVC

LDAA Load A LDAA

Operation: (M) ⇒ A

Description: Loads the content of a memory byte into accumulator A. Memory content is not changed by the operation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM			PK	
—	—	—	—	Δ	Δ	0	—		—		—			—	

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if A7 = 1 as a result of operation; else cleared.
Z:	Set if (A) = $00 as a result of operation; else cleared.
V:	Cleared.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	45	ff	6
IND8, Y	55	ff	6
IND8, Z	65	ff	6
IMM8	75	ii	2
IND16, X	1745	gggg	6
IND16, Y	1755	gggg	6
IND16, Z	1765	gggg	6
EXT	1775	hhll	6
E, X	2745	—	6
E, Y	2755	—	6
E, Z	2765	—	6

LDAB Load B LDAB

Operation: (M) ⇒ B

Description: Loads the content of a memory byte into accumulator B. Memory content is not changed by the operation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	0	—	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if B7 = 1 as a result of operation; else cleared.
Z:	Set if (B) = $00 as a result of operation; else cleared.
V:	Cleared.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	C5	ff	6
IND8, Y	D5	ff	6
IND8, Z	E5	ff	6
IMM8	F5	ii	2
IND16, X	17C5	gggg	6
IND16, Y	17D5	gggg	6
IND16, Z	17E5	gggg	6
EXT	17F5	hhll	6
E, X	27C5	—	6
E, Y	27D5	—	6
E, Z	27E5	—	6

LDD Load D LDD

Operation: (M : M + 1) ⇒ D

Description: Loads the content of a memory word into accumulator D. Memory content is not changed by the operation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	0	—	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if D15 = 1 as a result of operation; else cleared.
Z:	Set if (D) = $0000 as a result of operation; else cleared.
V:	Cleared.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	85	ff	6
IND8, Y	95	ff	6
IND8, Z	A5	ff	6
IMM16	37B5	jjkk	4
IND16, X	37C5	gggg	6
IND16, Y	37D5	gggg	6
IND16, Z	37E5	gggg	6
EXT	37F5	hhll	6
E, X	2785	—	6
E, Y	2795	—	6
E, Z	27A5	—	6

LDE Load E LDE

Operation: $(M : M + 1) \Rightarrow E$

Description: Loads the content of a memory word into accumulator E. Memory content is not changed by the operation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	0	—	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if E15 = 1 as a result of operation; else cleared.
Z: Set if (E) = $0000 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IMM16	3735	jjkk	4
IND16, X	3745	gggg	6
IND16, Y	3755	gggg	6
IND16, Z	3765	gggg	6
EXT	3775	hhll	6

LDED Load Concatenated E and D LDED

Operation: $(M : M + 1) \Rightarrow E$
$(M + 2 : M + 3) \Rightarrow D$

Description: Loads four successive bytes of memory into concatenated accumulators E and D. Used to transfer long word operands and 32-bit signed fractions from memory. Can also be used to transfer 32-bit words from IMB peripherals. Misaligned long transfers are converted into two misaligned word transfers.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
EXT	2771	hhll	8

LDHI Load MAC Registers H and I LDHI

Operation: $(M : M + 1)_X \Rightarrow HR$
$(M : M + 1)_Y \Rightarrow IR$

Description: Initializes MAC registers H and I. HR is loaded with a memory word located at address (XK : IX). IR is loaded with a memory word located at address (YK : IY). Memory content is not changed by the operation. See **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
EXT	27B0	—	8

LDS — Load Stack Pointer — LDS

Operation: (M : M + 1) ⇒ SP

Description: Loads the content of a memory word into the stack pointer. Memory content is not changed by the operation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	0	—	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if SP15 = 1 as a result of operation; else cleared.
Z: Set if (SP) = $0000 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	CF	ff	6
IND8, Y	DF	ff	6
IND8, Z	EF	ff	6
IMM16	37BF	jjkk	4
IND16, X	17CF	gggg	6
IND16, Y	17DF	gggg	6
IND16, Z	17EF	gggg	6
EXT	17FF	hhll	6

LDX Load IX LDX

Operation: (M : M + 1) ⇒ IX

Description: Loads the content of a memory word into index register X. Memory content is not changed by the operation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	0	—	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if IX15 = 1 as a result of operation; else cleared.
Z: Set if (IX) = $0000 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	CC	ff	6
IND8, Y	DC	ff	6
IND8, Z	EC	ff	6
IMM16	37BC	jjkk	4
IND16, X	17CC	gggg	6
IND16, Y	17DC	gggg	6
IND16, Z	17EC	gggg	6
EXT	17FC	hhll	6

LDY Load IY LDY

Operation: (M : M + 1) ⇒ IY

Description: Loads the content of a memory word into index register Y. Memory content is not changed by the operation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	0	—	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if IY15 = 1 as a result of operation; else cleared.
Z: Set if (IY) = $0000 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	CD	ff	6
IND8, Y	DD	ff	6
IND8, Z	ED	ff	6
IMM16	37BD	jjkk	4
IND16, X	17CD	gggg	6
IND16, Y	17DD	gggg	6
IND16, Z	17ED	gggg	6
EXT	17FD	hhll	6

LDZ Load IZ LDZ

Operation: (M : M + 1) ⇒ IZ

Description: Loads the content of a memory word into index register Z. Memory content is not changed by the operation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
—	—	—	—	Δ	Δ	0	—		—		—		—		

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if IZ15 = 1 as a result of operation; else cleared.
Z: Set if (IZ) = $0000 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	CE	ff	6
IND8, Y	DE	ff	6
IND8, Z	EE	ff	6
IMM16	37BE	jjkk	4
IND16, X	17CE	gggg	6
IND16, Y	17DE	gggg	6
IND16, Z	17EE	gggg	6
EXT	17FE	hhll	6

LPSTOP Low Power Stop LPSTOP

Operation: If $\overline{S}$, then enter low-power mode
Else NOP

Description: Operation is controlled by the S bit in the CCR. If S = 0 when LPSTOP is executed, the IP field from the condition code register is copied into an external bus interface, and the system clock input to the CPU is disabled. If S = 1, LPSTOP operates in the same way as a 4-cycle NOP.

Normal execution of instructions can resume in one of two ways. If a reset occurs, a reset exception is generated. If an interrupt request of higher priority than the copied IP value is received, an interrupt exception is generated. See **SECTION 9 EXCEPTION PROCESSING** for more information.

Syntax: Standard

Condition Code Register: Not affected.

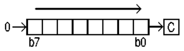
Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27F1	—	4, 20

Cycle times are for S = 1, S = 0 respectively.

LSR Logic Shift Right LSR

Operation:



Description: Shifts all eight bits of a memory byte one place to the right. Bit 7 is cleared. Bit 0 is transferred to the CCR C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	0	Δ	Δ	Δ	—			—	—			

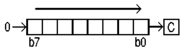
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Cleared.
Z: Set if (M) = $00 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if M0 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	0F	ff	8
IND8, Y	1F	ff	8
IND8, Z	2F	ff	8
IND16, X	170F	gggg	8
IND16, Y	171F	gggg	8
IND16, Z	172F	gggg	8
EXT	173F	hhll	8

LSRA Logic Shift Right A LSRA

Operation:



Description: Shifts all eight bits of accumulator A one place to the right. Bit 7 is cleared. Bit 0 is transferred to the CCR C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	0	Δ	Δ	Δ	—			—	—			

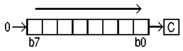
S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Cleared.
Z:	Set if (A) = $00; else cleared.
V:	Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C:	Set if A0 = 1 before operation; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	370F	—	2

LSRB Logic Shift Right B LSRB

Operation:



Description: Shifts all eight bits of accumulator B one place to the right. Bit 7 is cleared. Bit 0 is transferred to the CCR C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	0	∆	∆	∆	—			—	—			

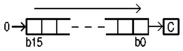
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Cleared.
Z: Set if (B) = $00 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if B0 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	371F	—	2

LSRD Logic Shift Right D LSRD

Operation:



Description: Shifts all sixteen bits of accumulator D one place to the right. Bit 15 is cleared. Bit 0 is transferred to the CCR C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	0	∆	∆	∆	—			—	—			

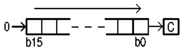
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Cleared.
Z: Set if (D) = $0000 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if D0 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27FF	—	2

LSRE Logic Shift Right E LSRE

Operation:



Description: Shifts all sixteen bits of accumulator E one place to the right. Bit 15 is cleared. Bit 0 is transferred to the CCR C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	0	Δ	Δ	Δ	—			—	—			

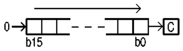
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Cleared.
Z: Set if (E) = $0000 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if E0 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	277F	—	2

LSRW Logic Shift Right Word LSRW

Operation:



Description: Shifts all sixteen bits of a memory word one place to the right. Bit 15 is cleared. Bit 0 is transferred to the CCR C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	0	Δ	Δ	Δ	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Cleared.
Z:	Set if (M : M + 1) = $0000 as a result of operation; else cleared.
V:	Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C:	Set if M : M + 1[0] = 1 before operation; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND16, X	270F	gggg	8
IND16, Y	271F	gggg	8
IND16, Z	272F	gggg	8
EXT	273F	hhll	8

Multiply and Accumulate

Operation:

$(HR) * (IR) \Rightarrow E : D$

$(AM) + (E : D) \Rightarrow AM$

$((IX) \leq \overline{X\ MASK}) \dotplus ((IX) + xo) \leq X\ MASK) \Rightarrow IX$

$((IY) \leq \overline{Y\ MASK}) \dotplus ((IY) + yo) \leq Y\ MASK) \Rightarrow IY$

$(HR) \Rightarrow IZ$

$(M : M + 1)_X \Rightarrow HR$

$(M : M + 1)_Y \Rightarrow IR$

Description:

Multiplies a 16-bit signed fractional multiplicand in MAC register I by a 16-bit signed fractional multiplier in MAC register H. There are implied radix points between bits 15 and 14 of the registers. The product is left-shifted one place to align the radix point between bits 31 and 30, then placed in bits 31:1 of concatenated accumulators E and D. D0 is cleared. The aligned product is then added to the content of AM.

As multiply and accumulate operations take place, 4-bit offsets xo and yo are sign-extended to 16 bits and used with X and Y masks to qualify the X and Y index registers.

Writing a non-zero value into a mask register prior to MAC execution enables modulo addressing. The TDMSK instruction writes mask values. When a mask contains $0, modulo addressing is disabled, and the sign-extended offset is added to the content of the corresponding index register.

After accumulation, the content of HR is transferred to IZ, then a word at the address pointed to by XK : IX is loaded into HR, and a word at the address pointed to by YK : IY is loaded into IR. The fractional product remains in concatenated E and D.

When both registers contain $8000 (–1), a value of $80000000 (1.0 in 36-bit format) is accumulated, (E : D) is $80000000 (–1 in 32-bit format), and the V bit in the condition code register is set. See **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

MAC Multiply and Accumulate MAC

Syntax: MAC xo, yo

Condition Code Register:

15	14	13	12	11	10	9	8	7 6 5	4	3 2 1 0
S	MV	H	EV	N	Z	V	C	IP	SM	PK
—	∆	—	∆	—	—	∆	—	—	—	—

S: Not affected.
MV: Set if overflow into AM35 occurs as a result of addition; else not affected.
H: Not affected.
EV: Set if overflow into AM[34:31] occurs as a result of addition; else cleared.
N: Not affected.
Z: Not affected.
V: Set if operation is (–1)^{2}; else cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Offset	Cycles
IMM8	7B	xoyo	12

MOVB Move Byte MOVB

Operation: $(M_1) \Rightarrow M_2$

Description: Moves a byte of data from a source address to a destination address. Data is examined as it is moved, and condition codes are set. Source data is not changed. A combination of source and destination addressing modes is used. Extended addressing can be used to specify source, destination, or both. A special form of indexed addressing, in which an 8-bit signed offset is added to the content of index register X after the move is complete, can be used to specify source or destination. If addition causes IX to overflow, the XK field is incremented or decremented.

Syntax: MOVB Source Offset Operand, X, Destination Address Operand
MOVB Source Address Operand, Destination Offset Operand,
XMOVB Source Address Operand, Destination Address Operand

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	0	—	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if MSB of source data = 1; else cleared.
Z: Set if source data = $00; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Offset	Addr Operand	Cycles
IXP to EXT	30	ff	hh ll	8
EXT to IXP	32	ff	hh ll	8
EXT to EXT	37FE	—	hhll hhll	10

MOVW Move Word MOVW

Operation: $(M : M + 1_1) \Rightarrow M : M + 1_2$

Description: Moves a data word from a source address to a destination address. Data is examined as it is moved, and condition codes are set. Source data is not changed. A combination of source and destination addressing modes is used. Extended addressing can be used to specify source, destination, or both. A special form of indexed addressing, in which an 8-bit signed offset is added to the content of index register X after the move is complete, can be used to specify source or destination only. If addition causes IX to overflow, the XK field is incremented or decremented.

Syntax:
MOVB Source Offset Operand, X, Destination Address Operand
MOVB Source Address Operand, Destination Offset Operand,
XMOVB Source Address Operand, Destination Address Operand

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	0	—	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if MSB of source data = 1; else cleared.
Z:	Set if source data = $0000; else cleared.
V:	Cleared.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Offset	Operand	Cycles
IXP to EXT	31	ff	hhll	8
EXT to IXP	33	ff	hhll	8
EXT to EXT	37FF	—	hhll hhll	10

MUL Unsigned Multiply MUL

Operation: (A) ∗ (B) ⇒ D

Description: Multiplies an 8-bit unsigned multiplicand contained in accumulator A by an 8-bit unsigned multiplier contained in accumulator B, then places the product in accumulator D. Unsigned multiply can be used to perform multiple-precision operations. The CCR Carry bit can be used to round the high byte of the product — execute MUL, then ADCA #0.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	—	—	—	∆	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Not affected.
Z:	Not affected.
V:	Not affected.
C:	Set if D7 = 1 as a result of operation; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3724	—	10

NEG Negate Byte NEG

Operation: $00 – (M) ⇒ M

Description: Replaces the content of a memory byte with its two’s complement. A value of $80 will not be changed.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	Δ	Δ	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if M7 = 1 as a result of operation; else cleared.
Z:	Set if (M) = $00 as a result of operation; else cleared.
V:	Set if (M) = $80 after operation (two’s complement overflow); else cleared.
C:	Cleared if (M) = $00 before operation; else set.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	02	ff	8
IND8, Y	12	ff	8
IND8, Z	22	ff	8
IND16, X	1702	gggg	8
IND16, Y	1712	gggg	8
IND16, Z	1722	gggg	8
EXT	1732	hhll	8

NEGA Negate A NEGA

Operation: $00 – (A) ⇒ A

Description: Replaces the content of accumulator A with its two’s complement. A value of $80 will not be changed.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	Δ	Δ	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if A7 = 1 as a result of operation; else cleared.
Z:	Set if (A) = $00 as a result of operation; else cleared.
V:	Set if (A) = $80 after operation (two’s complement overflow); else cleared.
C:	Cleared if (A) = $00 before operation; else set.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3702	—	2

NEGB Negate B NEGB

Operation: $00 – (B) ⇒ B

Description: Replaces the content of accumulator B with its two's complement. A value of $80 will not be changed.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	Δ	Δ	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if B7 = 1 as a result of operation; else cleared.
Z: Set if (B) = $00 as a result of operation; else cleared.
V: Set if (B) = $80 after operation (two's complement overflow); else cleared.
C: Cleared if (B) = $00 before operation; else set.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3712	—	2

NEGD Negate D NEGD

Operation: $0000 – (D) ⇒ D

Description: Replaces the content of accumulator D with its two’s complement. A value of $8000 will not be changed.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	Δ	Δ	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if D15 = 1 as a result of operation; else cleared.
Z:	Set if (D) = $0000 as a result of operation; else cleared.
V:	Set if (D) = $8000 after operation (two’s complement overflow); else cleared.
C:	Cleared if (D) = $0000 before operation; else set.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27F2	—	2

NEGE Negate E NEGE

Operation: $0000 – (E) ⇒ E

Description: Replaces the content of accumulator E with its two’s complement. A value of $8000 will not be changed.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
–	–	–	–	Δ	Δ	Δ	Δ		–		–		–		

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if E15 = 1 as a result of operation; else cleared.
Z: Set if (E) = $0000 as a result of operation; else cleared.
V: Set if (E) = $8000 after operation (two’s complement overflow); else cleared.
C: Cleared if (E) = $0000 before operation; else set.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	2772	—	2

NEGW Negate Word NEGW

Operation: $0000 – (M : M + 1) ⇒ M : M + 1

Description: Replaces the content of a memory word with its two's complement. A value of $8000 will not be changed.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM			PK	
–	–	–	–	∆	∆	∆	∆		–		–			–	

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if M : M + 1[15] = 1 as a result of operation; else cleared.
Z:	Set if (M : M + 1) = $0000 as a result of operation; else cleared.
V:	Set if (M : M + 1) = $8000 after operation (two's complement overflow); else cleared.
C:	Cleared if (M : M + 1) = $0000 before operation; else set.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND16, X	2702	gggg	8
IND16, Y	2712	gggg	8
IND16, Z	2722	gggg	8
EXT	2732	hhll	8

NOP Null Operation NOP

Operation: None

Description: Causes program counter to be incremented, but has no other effect. Often used to temporarily replace other instructions during debug, so that execution continues with a routine disabled. Can be used to produce a time delay based on CPU clock frequency, although this practice makes programs system-specific.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	274C	—	2

ORAA OR A ORAA

Operation: $(A) + (M) \Rightarrow A$

Description: Performs inclusive OR between the content of accumulator A and a memory byte, then places the result in accumulator A. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	0	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 is set by operation; else cleared.
Z: Set if (A) = $00 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	47	ff	6
IND8, Y	57	ff	6
IND8, Z	67	ff	6
IMM8	77	ii	2
IND16, X	1747	gggg	6
IND16, Y	1757	gggg	6
IND16, Z	1767	gggg	6
EXT	1777	hhll	6
E, X	2747	—	6
E, Y	2757	—	6
E, Z	2767	—	6

ORAB OR B ORAB

Operation: $(B) \dotplus (M) \Rightarrow B$

Description: Performs inclusive OR between the content of accumulator B and a memory byte, then places the result in accumulator B. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	0	—	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if B7 is set by operation; else cleared.
Z: Set if (B) = $00 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	C7	ff	6
IND8, Y	D7	ff	6
IND8, Z	E7	ff	6
IMM8	F7	ii	2
IND16, X	17C7	gggg	6
IND16, Y	17D7	gggg	6
IND16, Z	17E7	gggg	6
EXT	17F7	hhll	6
E, X	27C7	—	6
E, Y	27D7	—	6
E, Z	27E7	—	6

ORD OR D ORD

Operation: $(D) + (M : M + 1) \Rightarrow D$

Description: Performs inclusive OR between the content of accumulator D and a memory word, then places the result in accumulator D. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	∆	∆	0	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if D is set by operation; else cleared.
Z: Set if (D) = $0000 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	87	ff	6
IND8, Y	97	ff	6
IND8, Z	A7	ff	6
IMM16	37B7	jjkk	4
IND16, X	37C7	gggg	6
IND16, Y	37D7	gggg	6
IND16, Z	37E7	gggg	6
EXT	37F7	hhll	6
E, X	2787	—	6
E, Y	2797	—	6
E, Z	27A7	—	6

ORE OR E ORE

Operation: $(E) \dotplus (M : M + 1) \Rightarrow E$

Description: Performs inclusive OR between the content of accumulator E and a memory word, then places the result in accumulator E. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	0	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if E15 is set by operation; else cleared.
Z: Set if (E) = $0000 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IMM16	3737	jjkk	4
IND16, X	3747	gggg	6
IND16, Y	3757	gggg	6
IND16, Z	3767	gggg	6
EXT	3777	hhll	6

ORP OR Condition Code Register ORP

Operation: (CCR) ∔ IMM16 ⇒ CCR

Description: Performs inclusive OR between the content of the condition code register and a 16-bit unsigned immediate operand, then replaces the content of the CCR with the result.

To make certain that conditions for termination of LPSTOP and WAI are correct, interrupts are not recognized until after the instruction following ORP executes. This prevents interrupt exception processing during the period after the mask changes but before the following instruction executes.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ			Δ	–			

S: Set if bit 15 of operand = 1; else unchanged.
MV: Set if bit 14 of operand = 1; else unchanged.
H: Set if bit 13 of operand = 1; else unchanged.
EV: Set if bit 12 of operand = 1; else unchanged.
N: Set if bit 11 of operand = 1; else unchanged.
Z: Set if bit 10 of operand = 1; else unchanged.
V: Set if bit 9 of operand = 1; else unchanged.
C: Set if bit 8 of operand = 1; else unchanged.
IP: Each bit in field set if corresponding bit [7:5] of operand = 1; else unchanged.
SM: Set if bit 4 of operand = 1; else unchanged.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IMM16	373B	jjkk	4

PSHA Push A PSHA

Operation: $(SK : SP) + \$0001 \Rightarrow SK : SP$
Push (A)
$(SK : SP) - \$0002 \Rightarrow SK : SP$

Description: Increments (SK : SP) by one, stores the content of accumulator A at that address, then decrements (SK : SP) by two. If the SP overflows as a result of the operation, the SK field is incremented or decremented.

Pushing byte data to the stack can misalign the stack pointer and degrade performance. See **SECTION 8 INSTRUCTION TIMING** for more information.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3708	—	4

PSHB Push B PSHB

Operation: (SK : SP) + $0001 ⇒ SK : SP
Push (B)
(SK : SP) – $0002 ⇒ SK : SP

Description: Increments (SK : SP) by one, stores the content of accumulator B at that address, then decrements (SK : SP) by two. If the SP overflows as a result of the operation, the SK field is incremented or decremented.

Pushing byte data to the stack can misalign the stack pointer and degrade performance. See **SECTION 8 INSTRUCTION TIMING** for more information.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3718	—	4

PSHM Push Multiple Registers PSHM

Operation:

```
For mask bits 0 to 7
If bit set
push corresponding register
(SK : SP) – $0002 ⇒ SK : SP
Next
```

Mask bits:
0 = accumulator D
1 = accumulator E
2 = index register X
3 = index register Y
4 = index register Z
5 = extension register
6 = condition code register
7 = (Reserved)

Description: Stores contents of selected registers on the system stack. Registers are designated by setting bits in a mask byte. The PULM instruction restores registers from the stack. PUSHM mask order is the reverse of PULM mask order. If SP overflow occurs as a result of operation, the SK field is decremented.

Stacking into the highest available memory address causes the PULM instruction to attempt a prefetch from inaccessible memory. Pushing to an odd SK : SP can degrade performance. See **SECTION 8 INSTRUCTION TIMING** for more information.

Syntax: PSHM (mask)

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Mask	Cycles
IMM8	34	ii	4 + 2N*

*N = Number of registers to be pushed.

PSHMAC Push MAC Registers PSHMAC

Operation: Stack registers in sequence shown, beginning at address pointed to by stack pointer.

		15	14	8	7	3	0
Start	(SP)	H REGISTER					
	(SP) + $0002	I REGISTER					
	(SP) + $0004	ACCUMULATOR M[15:0]					
	(SP) + $0006	ACCUMULATOR M[31:16]					
	(SP) + $0008	SL	RESERVED			AM[35:32]	
End	(SP) + $000A	IX ADDRESS MASK			IY ADDRESS MASK		

Description: Stores multiply and accumulate unit internal state on the system stack. The SP is decremented after each save operation (stack grows downward in memory). If SP overflow occurs as a result of operation, the SK field is decremented. See **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27B8	—	14

PULA Pull A PULA

Operation: (SK : SP) + $0002 ⇒ SK : SP
Pull (A)
(SK : SP) – $0001 ⇒ SK : SP

Description: Increments (SK : SP) by two, restores the content of accumulator A from that address, then decrements (SK : SP) by one. If the SP overflows as a result of the operation, the SK field is incremented or decremented.

Pulling byte data from the stack can misalign the stack pointer and degrade performance. See **SECTION 8 INSTRUCTION TIMING** for more information.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3709	—	6

PULB Pull B PULB

Operation:

(SK : SP) + $0002 ⇒ SK : SP
Pull (B)
(SK : SP) – $0001 ⇒ SK : SP

Description: Increments (SK : SP) by two, restores the content of accumulator B from that address, then decrements (SK : SP) by one. If the SP overflows as a result of the operation, the SK field is incremented or decremented.

Pulling byte data from the stack can misalign the stack pointer and degrade performance. See **SECTION 8 INSTRUCTION TIMING** for more information.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3719	—	6

PULM Pull Multiple Registers PULM

Operation:

```
For mask bits 0 to 7
    If bit set
        (SK : SP) + $0002 ⇒ SK : SP
        Pull corresponding register
Next
```

Mask bits:

0 = condition code register
1 = extension register
2 = index register Z
3 = index register Y
4 = index register X
5 = accumulator E
6 = accumulator D
7 = (Reserved)

Description: Restores contents of registers stacked by a PSHM instruction. Registers are designated by setting bits in a mask byte. PULM mask order is the reverse of PSHM mask order. If SP overflow occurs as a result of operation, the SK field is incremented.

PULM prefetches a stacked word on each iteration. If SP points to the highest available stack address after the last register has been restored, the prefetch will attempt to read inaccessible memory. Pulling from an odd SK : SP can degrade performance. See **SECTION 8 INSTRUCTION TIMING** for more information.

Syntax: PULM (mask)

Condition Code Register:

Set according to CCR pulled from stack. Not affected unless CCR is pulled.

Instruction Format:

Addressing Mode	Opcode	Mask	Cycles
IMM8	35	ii	4+ 2 (N + 1)*

*N = Number of registers to be pulled.

PULMAC Pull MAC Registers PULMAC

Operation: Restore registers in sequence shown, beginning at address pointed to by stack pointer.

		15	14 8	7	3 0
End	(SP) + $000C	IX ADDRESS MASK		IY ADDRESS MASK	
	(SP) + $000A	SL	RESERVED		AM[35:32]
	(SP) + $0008	ACCUMULATOR M[31:16]			
	(SP) + $0006	ACCUMULATOR M[15:0]			
	(SP) + $0004	I REGISTER			
	(SP) + $0002	H REGISTER			
Start	(SP)	(Top of Stack)			

Description: Restores multiply and accumulate unit internal state from the system stack. The SP is incremented after each restoration (stack shrinks upward in memory). If SP overflow occurs as a result of operation, the SK field is incremented. See **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27B9	—	16

RMAC Repeating Multiply and Accumulate RMAC

Operation:

Repeat:
$(AM) + ((HR) * (IR)) \Rightarrow AM$
$((IX) \leq \overline{X\ MASK}) \text{⊹} ((IX) + xo) \leq X\ MASK) \Rightarrow IX$
$((IY) \leq \overline{Y\ MASK}) \text{⊹} ((IY) + yo) \leq Y\ MASK) \Rightarrow IY$
$(M : M + 1)_X \Rightarrow HR$
$(M : M + 1)_Y \Rightarrow IR$
$(E) - \$0001 \Rightarrow E$Until $(E) < \$0000$

Description:

Performs repeated multiplication of 16-bit signed fractional multiplicands in MAC register I by 16-bit signed fractional multipliers in MAC register H. Each product is added to the content of accumulator M. Accumulator D is used for temporary storage during multiplication. A 16-bit signed integer in accumulator E determines the number of repetitions.

There are implied radix points between bits 15 and 14 of HR and IR. Each product is left-shifted one place to align the radix point between bits 31 and 30 before addition to AM.

As multiply and accumulate operations take place, 4-bit offsets xo and yo are sign-extended to 16 bits and used with X and Y masks to qualify the X and Y index registers.

Writing a non-zero value into a mask register prior to RMAC execution enables modulo addressing. The TDMSK instruction writes mask values. When a mask contains $0, modulo addressing is disabled, and the sign-extended offset is added to the content of the corresponding index register.

After accumulation, a word pointed to by XK : IX is loaded into HR, and a word pointed to by YK : IY is loaded into IR, then the value in E is decremented and tested. After execution, content of E is indeterminate.

RMAC always iterates at least once, even when executed with a zero or negative value in E. Since the value in E is decremented, then tested, loading E with $8000 results in 32,769 iterations.

If HR and IR both contain $8000 (–1), a value of $80000000 (1.0 in 36-bit format) is accumulated, but no condition code is set.

RMAC execution is suspended during asynchronous exceptions. Operation resumes when RTI is executed. All registers used by RMAC must be restored prior to RTI. See **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

Syntax: RMAC xo, yo

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	∆	—	∆	—	—	—	—	—			—	—			

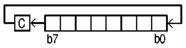
S: Not affected.
MV: Set if overflow into AM35 occurs as a result of addition; else not affected.
H: Not affected.
EV: Set if overflow into AM[34:31] occurs as a result of addition; else cleared.
N: Not affected.
Z: Not affected.
V: Not affected.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Offset	Cycles
IMM8	FB	xoyo	6 + 12 per iteration

ROL Rotate Left Byte ROL

Operation:



Description: Rotates all eight bits of a memory byte one place to the left. Bit 0 is loaded from the CCR carry bit. Bit 7 is transferred to the C bit.

Rotation through the C bit aids shifting and rotating multiple bytes. For example, use the sequence ASL Byte0, ROL Byte1, ROL Byte2 to shift a 24-bit value contained in bytes 0 to 2 left one bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	Δ	—	Δ	—	—	—	—	—			—	—			

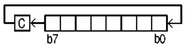
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M7 = 1 as a result of operation; else cleared.
Z: Set if (M) = $00 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if M7 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	0C	ff	8
IND8, Y	1C	ff	8
IND8, Z	2C	ff	8
IND16, X	170C	gggg	8
IND16, Y	171C	gggg	8
IND16, Z	172C	gggg	8
EXT	173C	hhll	8

ROLA Rotate Left A ROLA

Operation:



Description: Rotates all eight bits of accumulator A one place to the left. Bit 0 is loaded from the CCR carry bit. Bit 7 is transferred to the C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	Δ	Δ	–			–	–			

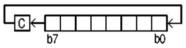
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 = 1 as a result of operation; else cleared.
Z: Set if (A) = $00 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if A7 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	370C	—	2

ROLB Rotate Left B ROLB

Operation:



Description: Rotates all eight bits of accumulator B one place to the left. Bit 0 is loaded from the CCR carry bit. Bit 7 is transferred to the C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	Δ	Δ	–			–	–			

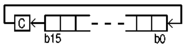
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if B7 = 1 as a result of operation; else cleared.
Z: Set if (B) = $00 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if B7 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	371C	—	2

ROLD Rotate Left D ROLD

Operation:



Description: Rotates all sixteen bits of accumulator D one place to the left. Bit 0 is loaded from the CCR carry bit. Bit 15 is transferred to the C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	Δ	Δ	–			–	–			

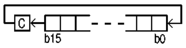
S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if D15 = 1 as a result of operation; else cleared.
Z:	Set if (D) = $0000 as a result of operation; else cleared.
V:	Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C:	Set if D15 = 1 before operation; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27FC	—	2

ROLE Rotate Left E ROLE

Operation:



Description: Rotates all sixteen bits of accumulator E one place to the left. Bit 0 is loaded from the CCR carry bit. Bit 15 is transferred to the C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	∆	∆	∆	∆	–			–	–			

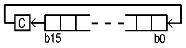
S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if E15 = 1 as a result of operation; else cleared.
Z:	Set if (E) = $0000 as a result of operation; else cleared.
V:	Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C:	Set if E15 = 1 before operation; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	277C	—	2

ROLW Rotate Left Word ROLW

Operation:



Description: Rotates all sixteen bits of a memory word one place to the left. Bit 0 is loaded from the CCR carry bit. Bit 15 is transferred to the C bit.

Rotation through the C bit aids shifting and rotating multiple words. For example, use the sequence ASLW Word0, ROLW Word1, ROLW Word2 to shift a 48-bit value contained in words 0 to 2 left one bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	∆	∆	∆	∆	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M : M + 1[15] = 1 as a result of operation; else cleared.
Z: Set if (M : M + 1) = $0000 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if M : M + 1[15] = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND16, X	270C	gggg	8
IND16, Y	271C	gggg	8
IND16, Z	272C	gggg	8
EXT	273C	hhll	8

ROR Rotate Right Byte ROR

Operation:

Description: Rotates all eight bits of a memory byte one place to the right. Bit 7 is loaded from the CCR C bit. Bit 0 is transferred to the C bit.

Rotation through the C bit aids shifting and rotating multiple words. For example, use the sequence LSR Byte2, ROR Byte1, ROR Byte0 to shift a 24-bit value contained in bytes 0 to 2 right one bit. Replace LSR with ASR to maintain the value of a sign bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	∆	∆	∆	∆	—			—	—			

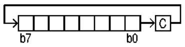
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M7 set as a result of operation; else cleared.
Z: Set if (M) = $00 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if M0 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	0E	ff	8
IND8, Y	1E	ff	8
IND8, Z	2E	ff	8
IND16, X	170E	gggg	8
IND16, Y	171E	gggg	8
IND16, Z	172E	gggg	8
EXT	173E	hhll	8

RORA Rotate Right A RORA

Operation:



Description: Rotates all eight bits of accumulator A one place to the right. Bit 7 is loaded from the CCR C bit. Bit 0 is transferred to the C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM			PK	
–	–	–	–	Δ	Δ	Δ	Δ		–		–			–	

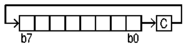
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 = 1 as a result of operation; else cleared.
Z: Set if (A) = $00; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if A0 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	370E	—	2

RORB Rotate Right B RORB

Operation:



Description: Rotates all eight bits of accumulator B one place to the right. Bit 7 is loaded from the CCR C bit. Bit 0 is transferred to the C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM			PK	
–	–	–	–	Δ	Δ	Δ	Δ		–		–			–	

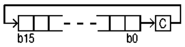
S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if B7 = 1 as a result of operation; else cleared.
Z:	Set if (B) = $00 as a result of operation; else cleared.
V:	Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C:	Set if B0 = 1 before operation; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	371E	—	2

RORD Rotate Right D RORD

Operation:



Description: Rotates all sixteen bits of accumulator D one place to the right. Bit 15 is loaded from the CCR C bit. Bit 0 is transferred to the C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	Δ	Δ	–			–	–			

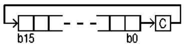
S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if D15 = 1 as a result of operation; else cleared.
Z: Set if (D) = $0000 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if D0 = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27FE	—	2

RORE Rotate Right E RORE

Operation:



Description: Rotates all sixteen bits of accumulator E one place to the right. Bit 15 is loaded from the CCR C bit. Bit 0 is transferred to the C bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	Δ	Δ	–			–	–			

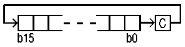
S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if E15 = 1 as a result of operation; else cleared.
Z:	Set if (E) = $0000 as a result of operation; else cleared.
V:	Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C:	Set if E0 = 1 before operation; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	277E	—	2

RORW Rotate Right Word RORW

Operation:



Description: Rotates all sixteen bits of a memory word one place to the right. Bit 15 is loaded from the CCR C bit. Bit 0 is transferred to the C bit.

Rotation through the C bit aids shifting and rotating multiple words. For example, use the sequence LSRW Word2, RORW Word1, RORW Word0 to shift a 48-bit value contained in words 0 to 2 right one bit. Replace LSRW with ASRW to maintain value of a sign bit.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	Δ	Δ	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M : M + 1[15] = 1 as a result of operation; else cleared.
Z: Set if (M : M + 1) = $0000 as a result of operation; else cleared.
V: Set if (N is set and C is clear) or (N is clear and C is set) as a result of operation; else cleared.
C: Set if M : M + 1[0] = 1 before operation; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND16, X	270E	gggg	8
IND16, Y	271E	gggg	8
IND16, Z	272E	gggg	8
EXT	273E	hhll	8

RTI Return From Interrupt RTI

Operation:

(SK : SP) + 2 ⇒ SK : SP
Pull CCR
(SK : SP) + 2 ⇒ SK : SP
Pull PC(PK : PC) – 6 ⇒ PK : PC

Description:

Causes normal program execution to resume after an interrupt, or any exception other than reset. The condition code register and program counter are restored from the system stack. When the CCR is pulled, the PK field is restored, so that execution resumes on the proper page after the PC is pulled.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
∆	∆	∆	∆	∆	∆	∆	∆	∆			∆	∆			

S: Set or cleared according to CCR restored from stack.
MV: Set or cleared according to CCR restored from stack.
H: Set or cleared according to CCR restored from stack.
EV: Set or cleared according to CCR restored from stack.
N: Set or cleared according to CCR restored from stack.
Z: Set or cleared according to CCR restored from stack.
V: Set or cleared according to CCR restored from stack.
C: Set or cleared according to CCR restored from stack.
IP: Value changes according to CCR restored from stack.
SM: Set or cleared according to CCR restored from stack.
PK: Value changes according to CCR restored from stack.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	2777	—	12

RTS — Return From Subroutine — RTS

Operation:

$(\mathrm{SK} : \mathrm{SP}) + 2 \Rightarrow \mathrm{SK} : \mathrm{SP}$
Pull PK
$(\mathrm{SK} : \mathrm{SP}) + 2 \Rightarrow \mathrm{SK} : \mathrm{SP}$
Pull PC
$(\mathrm{PK} : \mathrm{PC}) - 2 \Rightarrow \mathrm{PK} : \mathrm{PC}$

Description: Returns control to a routine that executed JSR. The PK field and program counter are restored from the system stack, so that execution resumes on the proper page. Use PSHM/PULM to conserve other program resources.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	–	–	–	–	–			–	Δ			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Not affected.
Z:	Not affected.
V:	Not affected.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Value changes to that of PK restored from stack.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27F7	—	12

SBA Subtract B from A SBA

Operation: (A) – (B) ⇒ A

Description: Subtracts the content of accumulator B from the content of accumulator A, then places the result in accumulator A. Content of accumulator B does not change. The CCR C bit represents a borrow for subtraction.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7 5	4	3 0
S	MV	H	EV	N	Z	V	C	IP	SM	PK
–	–	–	–	∆	∆	∆	∆	–	–	–

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 is set by operation; else cleared.
Z: Set if (A) = $00 as a result of operation; else cleared.
V: Set if two’s complement overflow occurs as a result of the operation; else cleared.
C: Set if $|(A)| < |(B)|$; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	370A	—	2

SBCA Subtract with Carry from A SBCA

Operation: (A) – (M) – C ⇒ A

Description: Subtracts the content of a memory byte minus the value of the C bit from the content of accumulator A, then places the result in accumulator A. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7 5	4	3 0
S	MV	H	EV	N	Z	V	C	IP	SM	PK
–	–	–	–	Δ	Δ	Δ	Δ	–	–	–

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 is set by operation; else cleared.
Z: Set if (A) = $00 as a result of operation; else cleared.
V: Set if two’s complement overflow occurs as a result of the operation; else cleared.
C: Set if $|(A)| < |(M) + C|$; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	42	ff	6
IND8, Y	52	ff	6
IND8, Z	62	ff	6
IMM8	72	ii	2
IND16, X	1742	gggg	6
IND16, Y	1752	gggg	6
IND16, Z	1762	gggg	6
EXT	1772	hhll	6
E, X	2742	—	6
E, Y	2752	—	6
E, Z	2762	—	6

SBCB Subtract with Carry from B SBCB

Operation: (B) – (M) – C ⇒ B

Description: Subtracts the content of a memory byte minus the value of the C bit from the content of accumulator B, then places the result in accumulator B. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7 5	4	3 0
S	MV	H	EV	N	Z	V	C	IP	SM	PK
–	–	–	–	∆	∆	∆	∆	–	–	–

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if B7 is set by operation; else cleared.
Z: Set if (B) = $00 as a result of operation; else cleared.
V: Set if two’s complement overflow occurs as a result of the operation; else cleared.
C: Set if $|(B)| < |(M) + C|$; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	C2	ff	6
IND8, Y	D2	ff	6
IND8, Z	E2	ff	6
IMM8	F2	ii	2
IND16, X	17C2	gggg	6
IND16, Y	17D2	gggg	6
IND16, Z	17E2	gggg	6
EXT	17F2	hhll	6
E, X	27C2	—	6
E, Y	27D2	—	6
E, Z	27E2	—	6

SBCD Subtract with Carry from D SBCD

Operation: (D) – (M : M + 1) – C ⇒ D

Description: Subtracts the content of a memory word minus the value of the C bit from the content of accumulator D, then places the result in accumulator D. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	∆	∆	∆	∆	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if D15 is set by operation; else cleared.
Z: Set if (D) = $0000 as a result of operation; else cleared.
V: Set if two’s complement overflow occurs as a result of operation; else cleared.
C: Set if $|(D)| < |(M : M + 1) + C|$; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	82	ff	6
IND8, Y	92	ff	6
IND8, Z	A2	ff	6
IMM16	37B2	jjkk	4
IND16, X	37C2	gggg	6
IND16, Y	37D2	gggg	6
IND16, Z	37E2	gggg	6
EXT	37F2	hhll	6
E, X	2782	—	6
E, Y	2792	—	6
E, Z	27A2	—	6

STAA Store A STAA

Operation: (A) ⇒ M

Description: Stores content of accumulator A in a memory byte. Content of accumulator is unchanged.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM			PK	
—	—	—	—	∆	∆	0	—		—		—			—	

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M7 is set as a result of operation; else cleared.
Z: Set if (M) = $00 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	4A	ff	4
IND8, Y	5A	ff	4
IND8, Z	6A	ff	4
IND16, X	174A	gggg	6
IND16, Y	175A	gggg	6
IND16, Z	176A	gggg	6
EXT	177A	hhll	6
E, X	274A	—	4
E, Y	275A	—	4
E, Z	276A	—	4

STAB Store B STAB

Operation: (B) ⇒ M

Description: Stores content of accumulator B in a memory byte. Content of accumulator is unchanged.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	∆	∆	0	—	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if M7 is set as a result of operation; else cleared.
Z:	Set if (M) = $00 as a result of operation; else cleared.
V:	Cleared.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	CA	ff	4
IND8, Y	DA	ff	4
IND8, Z	EA	ff	4
IND16, X	17CA	gggg	6
IND16, Y	17DA	gggg	6
IND16, Z	17EA	gggg	6
EXT	17FA	hhll	6
E, X	27CA	—	4
E, Y	27DA	—	4
E, Z	27EA	—	4

STD Store D STD

Operation: (D) ⇒ M : M + 1

Description: Stores content of accumulator D in a memory word. Content of accumulator is unchanged.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	∆	∆	0	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M : M + 1[15] is set as a result of operation; else cleared.
Z: Set if (M : M + 1) = $00 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	8A	ff	4
IND8, Y	9A	ff	4
IND8, Z	AA	ff	4
IND16, X	37CA	gggg	6
IND16, Y	37DA	gggg	6
IND16, Z	37EA	gggg	6
EXT	37FA	hhll	6
E, X	278A	—	6
E, Y	279A	—	6
E, Z	27AA	—	6

STE Store E STE

Operation: (E) ⇒ M : M + 1

Description: Stores content of accumulator E in a memory word. Content of accumulator is unchanged.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	0	—	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if M : M + 1[15] is set as a result of operation; else cleared.
Z:	Set if (M : M + 1) = $00 as a result of operation; else cleared.
V:	Cleared.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND16, X	374A	gggg	6
IND16, Y	375A	gggg	6
IND16, Z	376A	gggg	6
EXT	377A	hhll	6

STED Store Concatenated E and D STED

Operation: (E) ⇒ (M : M + 1)
(D) ⇒ (M + 2 : M + 3)

Description: Stores concatenated accumulators E and D into four successive bytes of memory. Used to transfer long-word and 32-bit fractional operands to memory. Can also be used to perform coherent long word transfers to IMB peripherals. Misaligned long word transfers are converted into two misaligned word transfers.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
EXT	2773	hhll	8

STS Store Stack Pointer STS

Operation: (SP) ⇒ M : M + 1

Description: Stores content of stack pointer in a memory word. Content of pointer is unchanged.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	0	—	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M : M + 1[15] is set as a result of operation; else cleared.
Z: Set if (M : M + 1) = $00 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	8F	ff	4
IND8, Y	9F	ff	4
IND8, Z	AF	ff	4
IND16, X	178F	gggg	6
IND16, Y	179F	gggg	6
IND16, Z	17AF	gggg	6
EXT	17BF	hhll	6

STX Store IX STX

Operation: (IX) ⇒ M : M + 1

Description: Stores content of index register X in a memory word. Content of register is unchanged.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	0	–	–			–	–			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if M : M + 1[15] is set as a result of operation; else cleared.
Z:	Set if (M : M + 1) = $00 as a result of operation; else cleared.
V:	Cleared.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	8C	ff	4
IND8, Y	9C	ff	4
IND8, Z	AC	ff	4
IND16, X	178C	gggg	6
IND16, Y	179C	gggg	6
IND16, Z	17AC	gggg	6
EXT	17BC	hhll	6

STY Store IY STY

Operation: (IY) ⇒ M : M + 1

Description: Stores content of index register Y in a memory word. Content of register is unchanged.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	∆	∆	0	—	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M : M + 1[15] is set as a result of operation; else cleared.
Z: Set if (M : M + 1) = $00 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	8D	ff	4
IND8, Y	9D	ff	4
IND8, Z	AD	ff	4
IND16, X	178D	gggg	6
IND16, Y	179D	gggg	6
IND16, Z	17AD	gggg	6
EXT	17BD	hhll	6

STZ Store IZ STZ

Operation: (IZ) ⇒ M : M + 1

Description: Stores content of index register Z in a memory word. Content of register is unchanged.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	∆	∆	0	—	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if M : M + 1[15] is set as a result of operation; else cleared.
Z:	Set if (M : M + 1) = $00 as a result of operation; else cleared.
V:	Cleared.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	8E	ff	4
IND8, Y	9E	ff	4
IND8, Z	AE	ff	4
IND16, X	178E	gggg	6
IND16, Y	179E	gggg	6
IND16, Z	17AE	gggg	6
EXT	17BE	hhll	6

SUBA Subtract from A SUBA

Operation: (A) – (M) ⇒ A

Description: Subtracts the content of a memory byte from the content of accumulator A, then places the result in accumulator A. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7 5	4	3 0
S	MV	H	EV	N	Z	V	C	IP	SM	PK
–	–	–	–	∆	∆	∆	∆	–	–	–

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 is set by operation; else cleared.
Z: Set if (A) = $00 as a result of operation; else cleared.
V: Set if two’s complement overflow occurs as a result of the operation; else cleared.
C: Set if $|(A)| < |(M)|$; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	40	ff	6
IND8, Y	50	ff	6
IND8, Z	60	ff	6
IMM8	70	ii	2
IND16, X	1740	gggg	6
IND16, Y	1750	gggg	6
IND16, Z	1760	gggg	6
EXT	1770	hhll	6
E, X	2740	—	6
E, Y	2750	—	6
E, Z	2760	—	6

SUBB Subtract from B SUBB

Operation: (B) – (M) ⇒ B

Description: Subtracts the content of a memory byte from the content of accumulator B, then places the result in accumulator B. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7 5	4	3 0
S	MV	H	EV	N	Z	V	C	IP	SM	PK
—	—	—	—	∆	∆	∆	∆	—	—	—

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if B7 is set by operation; else cleared.
Z:	Set if (B) = $00 as a result of operation; else cleared.
V:	Set if two’s complement overflow occurs as a result of the operation; else cleared.
C:	Set if $\lvert(B)\rvert < \lvert(M)\rvert$; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	C0	ff	6
IND8, Y	D0	ff	6
IND8, Z	E0	ff	6
IMM8	F0	ii	2
IND16, X	17C0	gggg	6
IND16, Y	17D0	gggg	6
IND16, Z	17E0	gggg	6
EXT	17F0	hhll	6
E, X	27C0	—	6
E, Y	27D0	—	6
E, Z	27E0	—	6

SUBD Subtract from D SUBD

Operation: (D) – (M : M + 1) ⇒ D

Description: Subtracts the content of a memory word from the content of accumulator D, then places the result in accumulator D. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	Δ	Δ	—			—	—			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if D15 is set by operation; else cleared.
Z:	Set if (D) = $0000 as a result of operation; else cleared.
V:	Set if two's complement overflow occurs as a result of operation; else cleared.
C:	Set if $\lvert(D)\rvert < \lvert(M : M + 1)\rvert$; else cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	80	ff	6
IND8, Y	90	ff	6
IND8, Z	A0	ff	6
IMM16	37B0	jjkk	4
IND16, X	37C0	gggg	6
IND16, Y	37D0	gggg	6
IND16, Z	37E0	gggg	6
EXT	37F0	hhll	6
E, X	2780	—	6
E, Y	2790	—	6
E, Z	27A0	—	6

SUBE Subtract from E SUBE

Operation: $(E) - (M : M + 1) \Rightarrow E$

Description: Subtracts the content of a memory word from the content of accumulator E, then places the result in accumulator E. Memory content is not affected.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	Δ	Δ	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if E15 is set by operation; else cleared.
Z: Set if (E) = $0000 as a result of operation; else cleared.
V: Set if two's complement overflow occurs as a result of the operation; else cleared.
C: Set if $|(E)| < |(M : M + 1)|$; else cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IMM16	3730	jjkk	4
IND16, X	3740	gggg	6
IND16, Y	3750	gggg	6
IND16, Z	3760	gggg	6
EXT	3770	hhll	6

SWI Software Interrupt SWI

Operation:

(PK : PC) + \$0002 ⇒ PK : PC
Push (PC)
(SK : SP) – \$0002 ⇒ SK : SP
Push (CCR)
(SK : SP) – \$0002 ⇒ SK : SP
\$0 ⇒ PK
(SWI Vector) ⇒ PC

Description: Causes an internally generated interrupt exception. Current program counter and condition code register (including the PK field) are saved on the system stack, then PK is cleared and the PC is loaded with exception vector 6 (content of address \$000C). See **SECTION 9 EXCEPTION PROCESSING** for more information.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	—	—	—	—	—			—	0			

S:	Not Affected.
MV:	Not Affected.
H:	Not Affected.
EV:	Not Affected.
N:	Not Affected.
Z:	Not Affected.
V:	Not Affected.
C:	Not Affected.
IP:	Not Affected.
SM:	Not Affected.
PK:	Cleared.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3720	—	16

SXT Sign Extend B into A SXT

Operation: If B7 = 1
then $FF ⇒ A
else $00 ⇒ A

Description: Extends an 8-bit two’s complement value contained in accumulator B into a 16-bit two’s complement value in accumulator D.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	∆	∆	–	–	–			–	–			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if A7 = 1 as a result of operation; else cleared.
Z:	Set if (A) = $00 as a result of operation; else cleared.
V:	Not affected.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27F8	—	2

TAB Transfer A to B TAB

Operation: (A) ⇒ B

Description: Replaces the content of accumulator B with the content of accumulator A. Content of A is not changed.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	0	—	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if B7 = 1 as a result of operation; else cleared.
Z: Set if (B) = $00 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3717	—	2

TAP Transfer A to Condition Code Register TAP

Operation: (A) ⇒ CCR[15:8]

Description: Replaces bits 15 to 8 of the condition code register with the content of accumulator A. Content of A is not changed.

To make certain that conditions for termination of LPSTOP and WAI are correct, interrupts are not recognized until after the instruction following TAP executes. This prevents interrupt exception processing during the period after the mask changes but before the following instruction executes.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	–			–	–			

S: Set or cleared according to content of A.
MV: Set or cleared according to content of A.
H: Set or cleared according to content of A.
EV: Set or cleared according to content of A.
N: Set or cleared according to content of A.
Z: Set or cleared according to content of A.
V: Set or cleared according to content of A.
C: Set or cleared according to content of A.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	37FD	—	4

TBA Transfer B to A TBA

Operation: (B) ⇒ A

Description: Replaces the content of accumulator A with the content of accumulator B. Content of B is not changed.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	0	—	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 = 1 as a result of operation; else cleared.
Z: Set if (A) = $00 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3707	—	2

TBEK Transfer B to EK TBEK

Operation: (B[3:0]) ⇒ EK

Description: Replaces the content of the EK field with the content of bits 0 to 3 of accumulator B. Bits 4 to 7 are ignored. Content of B is not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27FA	—	2

TBSK Transfer B to SK TBSK

Operation: (B[3:0]) ⇒ SK

Description: Replaces the content of the SK field with the content of bits 0 to 3 of accumulator B. Bits 4 to 7 are ignored. Content of B is not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	379F	—	2

TBXK Transfer B to XK TBXK

Operation: (B[3:0]) ⇒ XK

Description: Replaces the content of the XK field with the content of bits 0 to 3 of accumulator B. Bits 4 to 7 are ignored. Content of B is not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	379C	—	2

TBYK Transfer B to YK TBYK

Operation: (B[3:0]) ⇒ YK

Description: Replaces the content of the YK field with the content of bits 0 to 3 of accumulator B. Bits 4 to 7 are ignored. Content of B is not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	379D	—	2

TBZK Transfer B to ZK TBZK

Operation: (B[3:0]) ⇒ ZK

Description: Replaces the content of the ZK field with the content of bits 0 to 3 of accumulator B. Bits 4 to 7 are ignored. Content of B is not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	379E	—	2

TDE Transfer D to E TDE

Operation: (D) ⇒ E

Description: Replaces the content of accumulator E with the content of accumulator D. Content of D is not changed.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	0	—	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if E15 = 1 as a result of operation; else cleared.
Z: Set if (E) = $0000 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	277B	—	2

TDMSK Transfer D to XMSK:YMSK TDMSK

Operation: (D[15:8]) ⇒ XMSK
(D[7:0]) ⇒ YMSK

Description: Replaces the content of the MAC X and Y masks with the content of accumulator D. Content of D is not changed. Masks are used to implement modulo buffers. See **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	372F	—	2

TDP Transfer D to Condition Code Register TDP

Operation: (D) ⇒ CCR[15:4]

Description: Replaces bits 15 to 4 of the condition code register with the content of accumulator D. Content of D is not changed.

To make certain that conditions for termination of LPSTOP and WAI are correct, interrupts are not recognized until after the instruction following TDP executes. This prevents interrupt exception processing during the period after the mask changes but before the following instruction executes.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ			Δ	—			

S: Set or cleared according to content of D.
MV: Set or cleared according to content of D.
H: Set or cleared according to content of D.
EV: Set or cleared according to content of D.
N: Set or cleared according to content of D.
Z: Set or cleared according to content of D.
V: Set or cleared according to content of D.
C: Set or cleared according to content of D.
IP: Set or cleared according to content of D.
SM: Set or cleared according to content of D.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	372D	—	4

TED Transfer E to D TED

Operation: (E) ⇒ D

Description: Replaces the content of accumulator D with the content of accumulator E. Content of E is not changed.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	0	—	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if D15 = 1 as a result of operation; else cleared.
Z: Set if (D) = $0000 as a result of operation; else cleared.
V: Cleared.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27FB	—	2

TEKB Transfer EK to B TEKB

Operation: (EK) ⇒ B[3:0]
$0 ⇒ B[7:4]

Description: Replaces bits 0 to 3 of accumulator B with the content of the EK field. Bits 4 to 7 of B are cleared. Content of EK is not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27BB	—	2

TEM Transfer E to AM TEM

Operation:

$(E) \Rightarrow AM[31:16]$
$\$00 \Rightarrow AM[15:0]$
$AM[35:32] = AM31$

Description: Replaces bits 31 to 16 of the MAC accumulator with the content of accumulator E. AM[15:0] are cleared. AM[35:32] reflect the state of bit 31. Content of E is not changed.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
—	0	—	0	—	—	—	—		—		—		—		

S:	Not affected.
MV:	Cleared.
H:	Not affected.
EV:	Cleared.
N:	Not affected.
Z:	Not affected.
V:	Not affected.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27B2	—	4

TMER Transfer Rounded AM to E TMER

Operation:

Rounded (AM) ⇒ Temp
If (SM • (EV ÷ MV))
    then Saturation Value ⇒ E
else Temp ⇒ E

Description: The content of the MAC accumulator is rounded and transferred to temporary storage. If the saturation mode bit in the CCR is set and overflow occurs, a saturation value is transferred to accumulator E. Otherwise, the rounded value is transferred to accumulator E. TMER uses convergent rounding. Refer to **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	Δ	–	Δ	Δ	Δ	–	–	–			–	–			

S:	Not affected.
MV:	Set if overflow into AM35 occurs as a result of rounding; else not affected.
H:	Not affected.
EV:	Set if overflow into AM[34:31] occurs as a result of rounding; else not affected.
N:	Set if E15 = 1 as a result of operation; else cleared.
Z:	Set if (E) = $00 as a result of operation; else cleared.
V:	Not affected.
C:	Not affected.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27B4	—	6

TMET Transfer Truncated AM to E TMET

Operation:

```
If (SM ≤ (EV + MV))
    then Saturation Value ⇒ E
else AM[31:16] ⇒ E
```

Description: If the saturation mode bit in the CCR is set and overflow has occurred, a saturation value is transferred to accumulator E. Otherwise, AM[31:16] are transferred to accumulator E. Refer to **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information on overflow and data saturation.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	–	–	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if E15 = 1 as a result of operation; else cleared.
Z: Set if (E) = $00 as a result of operation; else cleared.
V: Not affected.
C: Not affected.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27B5	—	2

TMXED Transfer AM to IX : E : D TMXED

Operation:

AM[35:32] ⇒ IX[3:0]
AM35 ⇒ IX[15:4]
AM[31:16] ⇒ E
AM[15:0] ⇒ D

Description: Transfers content of the MAC accumulator to index register X, accumulator E, and accumulator D. See **SECTION 11 DIGITAL SIGNAL PROCESSING** for more information.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27B3	—	6

TPA Transfer Condition Code Register to A TPA

Operation: (CCR[15:8]) ⇒ A

Description: Replaces the content of accumulator A with bits 15 to 8 of the condition code register. Content of CCR is not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	37FC	—	2

TPD Transfer Condition Code Register to D TPD

Operation: (CCR) ⇒ D

Description: Replaces the content of accumulator D with the content of the condition code register. Content of CCR is not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	372C	—	2

TSKB Transfer SK to B TSKB

Operation: (SK) ⇒ B[3:0]$0 ⇒ B[7:4]

Description: Replaces bits 0 to 3 of accumulator B with the content of the SK field. Bits 4 to 7 of B are cleared. Content of SK is not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	37AF	—	2

TST Test Byte TST

Operation: (M) – $00

Description: Subtracts $00 from the content of a memory byte and sets bits in the condition code register accordingly. The operation does not change memory content.

TST has minimal utility with unsigned values. BLO and BLS, for example, will not function because no unsigned value is less than zero. BHI will function the same as BNE, which is preferred.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
—	—	—	—	Δ	Δ	0	0	—			—	—			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if M7 = 1 as a result of operation; else cleared.
Z: Set if (M) = $00 as a result of operation; else cleared.
V: Cleared.
C: Cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND8, X	06	ff	6
IND8, Y	16	ff	6
IND8, Z	26	ff	6
IND16, X	1706	gggg	6
IND16, Y	1716	gggg	6
IND16, Z	1726	gggg	6
EXT	1736	hhll	6

TSTA Test A TSTA

Operation: (A) – $00

Description: Subtracts $00 from the content of accumulator A and sets bits in the condition code register accordingly. The operation does not change accumulator content.

TSTA has minimal utility with unsigned values. BLO and BLS, for example, will not function because no unsigned value is less than zero. BHI will function the same as BNE, which is preferred.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	0	0	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if A7 = 1 as a result of operation; else cleared.
Z: Set if (A) = $00 as a result of operation; else cleared.
V: Cleared.
C: Cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3706	—	2

TSTB Test B TSTB

Operation: (B) – $00

Description: Subtracts $00 from the content of accumulator B and sets bits in the condition code register accordingly. The operation does not change accumulator content.

TSTB has minimal utility with unsigned values. BLO and BLS, for example, will not function because no unsigned value is less than zero. BHI will function the same as BNE, which is preferred.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		
–	–	–	–	Δ	Δ	0	0		–		–		–		

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if B7 = 1 as a result of operation; else cleared.
Z: Set if (B) = $00 as a result of operation; else cleared.
V: Cleared.
C: Cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	3716	—	2

TSTD Test D TSTD

Operation: (D) – $0000

Description: Subtracts $0000 from the content of accumulator D and sets bits in the condition code register accordingly. The operation does not change accumulator content.

TSTD provides minimum information to subsequent instructions when unsigned values are tested. BLO and BLS, for example, have no utility because no unsigned value is less than zero. BHI will function the same as BNE, which is preferred. All signed branch instructions are available after test of signed values.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	0	0	–			–	–			

S: Not affected.
MV: Not affected.
H: Not affected.
EV: Not affected.
N: Set if D15 = 1 as a result of operation; else cleared.
Z: Set if (D) = $0000 as a result of operation; else cleared.
V: Cleared.
C: Cleared.
IP: Not affected.
SM: Not affected.
PK: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27F6	—	2

TSTE Test E TSTE

Operation: (E) – $0000

Description: Subtracts $0000 from the content of accumulator E and sets the bits in the condition code register accordingly. The operation does not change accumulator content.

TSTE provides minimum information to subsequent instructions when unsigned values are tested. BLO and BLS, for example, have no utility because no unsigned value is less than zero. BHI will function the same as BNE, which is preferred. All signed branch instructions are available after test of signed values.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	∆	∆	0	0	–			–	–			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if E15 = 1 as a result of operation; else cleared.
Z:	Set if (E) = $0000 as a result of operation; else cleared.
V:	Cleared.
C:	Cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	2776	—	2

TSTW Test Word TSTW

Operation: (M : M + 1) – $0000

Description: Subtracts $0000 from the content of a memory word and sets the bits in the condition code register accordingly. The operation does not change memory content.

TSTW provides minimum information to subsequent instructions when unsigned values are tested. BLO and BLS, for example, have no utility because no unsigned value is less than zero. BHI will function the same as BNE, which is preferred. All signed branch instructions are available after test of signed values.

Syntax: Standard

Condition Code Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
–	–	–	–	Δ	Δ	0	0	–			–	–			

S:	Not affected.
MV:	Not affected.
H:	Not affected.
EV:	Not affected.
N:	Set if M : M + 1[15] = 1 as a result of operation; else cleared.
Z:	Set if (M : M + 1) = $0000 as a result of operation; else cleared.
V:	Cleared.
C:	Cleared.
IP:	Not affected.
SM:	Not affected.
PK:	Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
IND16, X	2706	gggg	6
IND16, Y	2716	gggg	6
IND16, Z	2726	gggg	6
EXT	2736	hhll	6

TSX Transfer SP to IX TSX

Operation: (SK : SP) + $0002 ⇒ XK : IX

Description: Replaces the contents of the XK field and index register X with the contents of the SK field and the stack pointer plus two. Contents of SK and SP are not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	274F	—	2

TSY Transfer SP to IY TSY

Operation: (SK : SP) + $0002 ⇒ YK : IY

Description: Replaces the contents of the YK field and index register Y with the contents of the SK field and the stack pointer plus two. Contents of SK and SP are not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	275F	—	2

TSZ Transfer SP to IZ TSZ

Operation: (SK : SP) + \$0002 ⇒ ZK : IZ

Description: Replaces the contents of the ZK field and index register Z with the contents of the SK field and the stack pointer plus two. Contents of SK and SP are not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	276F	—	2

TXKB Transfer XK to B TXKB

Operation: (XK) ⇒ B[3:0]$0 ⇒ B[7:4]

Description: Replaces bits 0 to 3 of accumulator B with the content of the XK field. Bits 4 to 7 of B are cleared. Content of XK is not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	37AC	—	2

TXS Transfer IX to SP TXS

Operation: (XK : IX) – $0002 ⇒ SK : SP

Description: Replaces the content of the SK field and the stack pointer with the content of the XK field and index register X minus two. Content of XK and IX are not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	374E	—	2

TXY Transfer IX to IY TXY

Operation: (XK : IX) ⇒ YK : IY

Description: Replaces the content of the YK field and index register Y with the content of the XK field and index register X. Content of XK and IX are not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	275C	—	2

TXZ Transfer IX to IZ TXZ

Operation: (XK : IX) ⇒ ZK : IZ

Description: Replaces the content of the ZK field and index register Z with the content of the XK field and index register X. Content of XK and IX are not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	276C	—	2

TYKB Transfer YK to B TYKB

Operation: (YK) ⇒ B[3:0]$0 ⇒ B[7:4]

Description: Replaces bits 0 to 3 of accumulator B with the content of the YK field. Bits 4 to 7 of B are cleared. Content of YK is not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	37AD	—	2

TYS Transfer IY to SP TYS

Operation: (YK : IY) – $0002 ⇒ SK : SP

Description: Replaces the content of the SK field and the stack pointer with the content of the YK field and index register Y minus two. Content of YK and IY are not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	375E	—	2

TYX Transfer IY to IX TYX

Operation: (YK : IY) ⇒ XK : IX

Description: Replaces the content of the XK field and index register X with the content of the YK field and index register Y. Content of YK and IY are not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	274D	—	2

TYZ Transfer IY to IZ TYZ

Operation: (YK : IY) ⇒ ZK : IZ

Description: Replaces the content of the ZK field and index register Z with the content of the YK field and index register Y. Content of YK and IY are not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	276D	—	2

TZKB Transfer ZK to B TZKB

Operation: (ZK) ⇒ B[3:0]
$0 ⇒ B[7:4]

Description: Replaces bits 0 to 3 of accumulator B with the content of the ZK field. Bits 4 to 7 of B are cleared. Content of ZK is not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	37AE	—	2

TZS Transfer IZ to SP TZS

Operation: (ZK : IZ) – $0002 ⇒ SK : SP

Description: Replaces the content of the SK field and the stack pointer with the content of the ZK field and index register Z minus two. Content of ZK and IZ are not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	376E	—	2

TZX Transfer IZ to IX TZX

Operation: (ZK : IZ) ⇒ XK : IX

Description: Replaces the content of the XK field and index register X with the content of the ZK field and index register Z. Content of ZK and IZ are not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	274E	—	2

TZY Transfer IZ to IY TZY

Operation: (ZK : IZ) ⇒ YK : IY

Description: Replaces the content of the YK field and index register Y with the content of the ZK field and index register Z. Content of ZK and IZ are not changed.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	275E	—	2

WAI Wait for Interrupt WAI

Operation: WAIT

Description: Internal CPU clocks are stopped, and normal execution of instructions ceases. Instruction execution can resume in one of two ways. If a reset occurs, a reset exception is generated. If an interrupt request of higher priority than the current IP value is received, an Interrupt exception is generated.

Interrupts are acknowledged faster after WAI than after LPSTOP, because IMB clocks continue to run during WAI operation, and the CPU16 does not copy the IP field to the system integration module external bus interface. However, LPSTOP minimizes microcontroller power consumption during inactivity. Refer to **SECTION 9 EXCEPTION PROCESSING** for more information.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	27F3	—	8

XGAB Exchange A and B XGAB

Operation: (A) ⇔ (B)

Description: Exchanges contents of accumulators A and B.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	371A	—	2

XGDE Exchange D and E XGDE

Operation: (D) ⇔ (E)

Description: Exchanges contents of accumulators D and E.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	277A	—	2

XGDX Exchange D and IX XGDX

Operation: (D) ⇔ (IX)

Description: Exchanges contents of accumulator D and index register X.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	37CC	—	2

XGDY Exchange D and IY XGDY

Operation: (D) ⇔ (IY)

Description: Exchanges contents of accumulator D and index register IY.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	37DC	—	2

XGDZ Exchange D and IZ XGDZ

Operation: (D) ⇔ (IZ)

Description: Exchanges contents of accumulator D and index register IZ.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	37EC	—	2

XGEX Exchange E and IX XGEX

Operation: (E) ⇔ (IX)

Description: Exchanges contents of accumulator E and index register X.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	374C	—	2

XGEY Exchange E and IY XGEY

Operation: (E) ⇔ (IY)

Description: Exchanges contents of accumulator E and index register Y.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	375C	—	2

XGEZ Exchange E and IZ XGEZ

Operation: (E) ⇔ (IZ)

Description: Exchanges contents of accumulator E and index register Z.

Syntax: Standard

Condition Code Register: Not affected.

Instruction Format:

Addressing Mode	Opcode	Operand	Cycles
INH	376C	—	2

6 :ondition Code Evaluation

he following table contains F e expressions used to evaluate the effect of an op- ation on conditi tus flags.

ondition Code Evaluation

	Evaluation
	= A3 • B3 ÷ B3 • $\overline{R3}$ ÷ $\overline{R3}$ • A3 = R7 = $\overline{R7}$ • $\overline{R6}$ •... • $\overline{R1}$ • $\overline{R0}$ = A7 • B7 • $\overline{R7}$ ÷ $\overline{A7}$ • $\overline{B7}$ • R7 C = A7 • B7 ÷ B7 • $\overline{R7}$ ÷ $\overline{R7}$ • A7
ACE	EV = [(AM35 ÷...÷ AM31) • ($\overline{AM35}$ ÷... ÷ $\overline{AM31}$)] ÷ MV MV — cannot be represented by a Boolean equation
	= X3 • M3 ÷ M3 • $\overline{R3}$ ÷ $\overline{R3}$ • X3 R7 $\overline{R7}$ • $\overline{R6}$ •... • $\overline{R1}$ • $\overline{R0}$ 7 • M7 • $\overline{R7}$ ÷ $\overline{X7}$ 6–265

Table 6-35 Condition Code Evaluation

Mnemonic	Evaluation

ASLM	$EV = [(AM35 \div \ldots \div AM31) \bullet (\overline{AM35} \div \ldots \div \overline{AM31})] \div MV$
	N = R35
	C = MSB of unshifted accumulator
	MV — cannot be represented by a Boolean equation

ASR
ASRA
ASRB
	N = R7
	$Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	$V = N \oplus C = [N \bullet \overline{C}] \div [\overline{N} \div C]$
	C = LSB of unshifted byte (accumulator)

ASRD
ASRE
ASRW
	N = R15
	$Z = \overline{R15} \bullet \overline{R14} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	$V = N \oplus C = [N \bullet \overline{C}] \div [\overline{N} \div C]$
	C = LSB of unshifted word (accumulator)

ASRM	$EV = [(AM35 \div \ldots \div AM31) \bullet (\overline{AM35} \div \ldots \div \overline{AM31})] \div MV$
	N = R35
	C = LSB of unshifted accumulator

BCLR	N = R7
	$Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	V = 0

BCLRW	N = R15
	$Z = \overline{R15} \bullet \overline{R14} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	V = 0

BITA
BITB
	N = R7
	$Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	V = 0

BSET	N = R7
	$Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	V = 0

CBA	N = R7 $Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$ $V = A7 \bullet \overline{B7} \bullet \overline{R7} \div \overline{A7} \bullet B7 \bullet R7$ $C = \overline{A7} \bullet B7 \div B7 \bullet R7 \div R7 \bullet \overline{A7}$

CLR
CLRA
CLRB
CLRD
CLRE
CLRW
	N = 0
	Z = 1
	V = 0
	C = 0

CLRM	EV = 0
	MV = 0

CMPA CMPB	N = R7 $Z = \overline{R7} \bullet \overline{R6} \bullet .. \bullet \overline{R1} \bullet \overline{R0}$ $V = X7 \bullet \overline{M7} \bullet \overline{R7} \div \overline{X7} \bullet M7 \bullet R7$ $C = \overline{X7} \bullet M7 \div M7 \bullet R7 \div R7 \bullet \overline{X7}$

COM
COMA
COMB
	N = R7
	$Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	V = 0
	C = 1

COMD
COME
COMW
	N = R15
	$Z = \overline{R15} \bullet \overline{R14} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	V = 0
	C = 1

Table 6-35 Condition Code Evaluation

Mnemonic	Evaluation
CPD CPE CPS CPX CPY CPZ	$N = R15$ $Z = \overline{R15} \bullet \overline{R14} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$ $V = X15 \bullet \overline{M15} \bullet \overline{R15} \div \overline{X15} \bullet M15 \bullet R15$ $C = \overline{X15} \bullet M15 \div M15 \bullet R15 \div R15 \bullet \overline{X15}$
DAA	$N = R7$ $Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$ $V = U$ C = Determined by adjustment
DEC DECA DECB	$N = R7$ $Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$ $V = \overline{R7} \bullet R6 \bullet \ldots \bullet R1 \bullet R0$
DECW	$N = R15$ $Z = \overline{R15} \bullet \overline{R14} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$ $V = \overline{R15} \bullet R14 \bullet \ldots \bullet R1 \bullet R0$

EDIV
EDIVS
	$N = R15$
	$Z = \overline{R15} \bullet \overline{R14} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	V = 1 if R > \$FFFF
	C = 1 if $\lvert 2 * \text{Remainder} \rvert \geq \lvert \text{Divisor} \rvert$

EORA EORB	$N = R7$ $Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$ $V = 0$
EORD EORE	$N = R15$ $Z = \overline{R15} \bullet \overline{R14} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$ $V = 0$
FDIV	$Z = \overline{R15} \bullet \overline{R14} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$ V = 1, if (IX) • (D) $C = \overline{IX15} \bullet \overline{IX14} \bullet \ldots \bullet \overline{IX1} \bullet \overline{IX0}$
FMULS	$N = R31\ (E15)$ $Z = \overline{R31} \bullet \overline{R30} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$ $V = (D15 \bullet (\overline{D14} \bullet \overline{D13} \bullet \ldots \bullet \overline{D1} \bullet \overline{D0})) \bullet$ $(E15 \bullet (\overline{E14} \bullet \overline{E13} \bullet \ldots \bullet \overline{E1} \bullet \overline{E0}))$ $C = R15\ (D15)$
IDIV	$Z = \overline{R15} \bullet \overline{R14} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$ $V = 0$ $C = \overline{IX15} \bullet \overline{IX14} \bullet \ldots \bullet \overline{IX1} \bullet \overline{IX0}$
INC INCA INCB	$N = R7$ $Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$ $V = R7 \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
INCW	$N = R15$ $Z = \overline{R15} \bullet \overline{R14} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$ $V = R15 \bullet \overline{R14} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
LDAA LDAB	$N = R7$ $Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$ $V = 0$
LDD LDE LDS LDX LDY LDZ	$N = R15$ $Z = \overline{R15} \bullet \overline{R14} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$ $V = 0$

Table 6-35 Condition Code Evaluation

Mnemonic	Evaluation

LSR
LSRA
LSRB
	N = 0
	$Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	$V = [N \bullet \overline{C}] \div [\overline{N} \bullet C]$
	C = MSB of unshifted byte (accumulator)

LSRD
LSRE
LSRW
	N = 0
	$Z = \overline{R15} \bullet \overline{R14} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	$V = [N \bullet \overline{C}] \div [\overline{N} \bullet C]$
	C = MSB of unshifted word (accumulator)

MAC	$EV = [(AM35 \div \ldots \div AM31) \bullet (\overline{AM35} \div \ldots \div \overline{AM31})] \div MV$ $V = (H15 \bullet (\overline{H14} \bullet \ldots \bullet \overline{H0})) \bullet (I15 \bullet (\overline{I14} \bullet \ldots \bullet \overline{I0}))$ MV — cannot be represented by a Boolean equation
MOVB	N = MSB of source data $Z = S7 \bullet S6 \bullet \ldots \bullet S1 \bullet S0$
MOVW	N = MSB of source data $Z = S15 \bullet S14 \bullet \ldots \bullet S1 \bullet S0$
MUL	C = R7 (D7)

ORAA
ORAB
	N = R7
	$Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	V = 0

ORD
ORE
	N = R15
	$Z = \overline{R15} \bullet \overline{R14} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	V = 0

ORP	CCR[15:4] changed by OR with 16-bit immediate data, CCR[3:0] not affected.
PULM	Entire CCR changed if a stacked CCR is pulled.
RMAC	$EV = [(AM35 \div \ldots \div AM31) \bullet (\overline{AM35} \div \ldots \div \overline{AM31})] \div MV$ $V = (H15 \bullet (\overline{H14} \bullet \ldots \bullet \overline{H0})) \bullet (I15 \bullet (\overline{I14} \bullet \ldots \bullet \overline{I0}))$ MV — cannot be represented by a Boolean equation

ROL
ROLA
ROLB
	N = R7
	$Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	$V = N \oplus C = [N \bullet \overline{C}] \div [\overline{N} \div C]$
	C = MSB of unshifted byte (accumulator)

ROLD
ROLE
ROLW
	N = R15
	$Z = \overline{R15} \bullet \overline{R14} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	$V = N \oplus C = [N \bullet \overline{C}] \div [\overline{N} \div C]$
	C = MSB of unshifted word (accumulator)

ROR
RORA
RORB
	N = R7
	$Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	$V = N \oplus C = [N \bullet \overline{C}] \div [\overline{N} \div C]$
	C = MSB of unshifted byte (accumulator)

RORD
RORE
RORW
	N = R15
	$Z = \overline{R15} \bullet \overline{R14} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$
	$V = N \oplus C = [N \bullet \overline{C}] \div [\overline{N} \div C]$
	C = MSB of unshifted word (accumulator)

RTI	Entire CCR changed when stacked CCR is pulled.
SBA	N = R7 $Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$ $V = A7 \bullet B7 \bullet \overline{R7} \div \overline{A7} \bullet \overline{B7} \bullet R7$ $C = \overline{A7} \bullet B7 \div B7 \bullet R7 \div R7 \bullet \overline{A7}$
SBCA SBCB	N = R7 $Z = \overline{R7} \bullet \overline{R6} \bullet \ldots \bullet \overline{R1} \bullet \overline{R0}$ $V = X7 \bullet \overline{M7} \bullet \overline{R7} \div \overline{X7} \bullet M7 \bullet R7$ $C = \overline{X7} \bullet M7 \div M7 \bullet R7 \div R7 \bullet \overline{X7}$

Table 6-35 Condition Code Evaluation

Mnemonic	Evaluation
SBCD SBCE	$N = R15$ $Z = \overline{R15} \bullet \overline{R14} \bullet ... \bullet \overline{R1} \bullet \overline{R0}$ $V = X15 \bullet \overline{M15} \bullet \overline{R15} \div \overline{X15} \bullet M15 \bullet R15$ $C = \overline{X15} \bullet M15 \div \overline{X15} \bullet R15 \div M15 \bullet R15$
SDE	$N = R15$ $Z = \overline{R15} \bullet \overline{R14} \bullet ... \bullet \overline{R1} \bullet \overline{R0}$ $V = E15 \bullet \overline{D15} \bullet \overline{R15} \div \overline{E15} \bullet D15 \bullet R15$ $C = \overline{E15} \bullet D15 \div \overline{E15} \bullet R15 \div D15 \bullet R15$
STAA STAB	$N = R7$ $Z = \overline{R7} \bullet \overline{R6} \bullet ... \bullet \overline{R1} \bullet \overline{R0}$ $V = 0$
STD STE STS STX STY STZ	$N = R15$ $Z = \overline{R15} \bullet \overline{R14} \bullet ... \bullet \overline{R1} \bullet \overline{R0}$ $V = 0$
SUBA SUBB	$N = R7$ $Z = \overline{R7} \bullet \overline{R6} \bullet ... \bullet \overline{R1} \bullet \overline{R0}$ $V = X7 \bullet \overline{M7} \bullet \overline{R7} \div \overline{X7} \bullet M7 \bullet R7$ $C = \overline{X7} \bullet M7 \div M7 \bullet R7 \div R7 \bullet \overline{X7}$
SUBD SUBE	$N = R15$ $Z = \overline{R15} \bullet \overline{R14} \bullet ... \bullet \overline{R1} \bullet \overline{R0}$ $V = X15 \bullet \overline{M15} \bullet \overline{R15} \div \overline{X15} \bullet M15 \bullet R15$ $C = \overline{X15} \bullet M15 \div \overline{X15} \bullet R15 \div M15 \bullet R15$
SXT	$N = R15$ $Z = \overline{R15} \bullet \overline{R14} \bullet ... \bullet \overline{R1} \bullet \overline{R0}$
TAB TBA	$N = R7$ $Z = \overline{R7} \bullet \overline{R6} \bullet ... \bullet \overline{R1} \bullet \overline{R0}$ $V = 0$
TAP	CCR[15:8] replaced by content of Accumulator A. CCR[7:0] not affected.
TDE TED	$N = R15$ $Z = \overline{R15} \bullet \overline{R14} \bullet ... \bullet \overline{R1} \bullet \overline{R0}$ $V = 0$
TDP	CCR[15:4] replaced by content of Accumulator D. CCR[3:0] not affected.
TEDM TEM	$EV = 0$ $MV = 0$
TMER	$EV = [(AM35 \div ... \div AM31) \bullet (\overline{AM35} \div ... \div \overline{AM31})] \div MV$ MV not representable with Boolean equation
TMET	$N = R15$ $Z = \overline{R15} \bullet \overline{R14} \bullet ... \bullet \overline{R1} \bullet \overline{R0}$
TST TSTA TSTB	$N = R7$ $Z = \overline{R7} \bullet \overline{R6} \bullet ... \bullet \overline{R1} \bullet \overline{R0}$ $V = 0$ $C = 0$
TSTD TSTE TSTW	$N = R15$ $Z = \overline{R15} \bullet \overline{R14} \bullet ... \bullet \overline{R1} \bullet \overline{R0}$ $V = 0$ $C = 0$

6.4 Instruction Set Summary

The following table is a summary of the CPU16 instruction set. Because it is only affected by a few instructions, the LSB of the condition code register is not shown in the table — instructions that affect the interrupt mask and PK field are noted.

Table 6-36 Instruction Set Summary

Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
ABA	Add B to A	(A) + (B) ⇒ A	INH	370B	—	2	—	—	Δ	—	Δ	Δ	Δ	Δ
ABX	Add B to IX	(XK : IX) + (000 : B) ⇒ XK : IX	INH	374F	—	2	—	—	—	—	—	—	—	—
ABY	Add B to IY	(YK : IY) + (000 : B) ⇒ YK : IY	INH	375F	—	2	—	—	—	—	—	—	—	—
ABZ	Add B to IZ	(ZK : IZ) + (000 : B) ⇒ ZK : IZ	INH	376F	—	2	—	—	—	—	—	—	—	—
ACE	Add E to AM	(AM[31:16]) + (E) ⇒ AM	INH	3722	—	2	—	Δ	—	Δ	—	—	—	—
ACED	Add E : D to AM	(AM) + (E : D) ⇒ AM	INH	3723	—	4	—	Δ	—	Δ	—	—	—	—
ADCA	Add with Carry to A	(A) + (M) + C ⇒ A	IND8, X	43	ff	6	—	—	Δ	—	Δ	Δ	Δ	Δ
			IND8, Y	53	ff	6								
			IND8, Z	63	ff	6								
			IMM8	73	ii	2								
			IND16, X	1743	gggg	6								
			IND16, Y	1753	gggg	6								
			IND16, Z	1763	gggg	6								
			EXT	1773	hh ll	6								
			E, X	2743	—	6								
			E, Y	2753	—	6								
			E, Z	2763	—	6								
ADCB	Add with Carry to B	(B) + (M) + C ⇒ B	IND8, X	C3	ff	6	—	—	Δ	—	Δ	Δ	Δ	Δ
			IND8, Y	D3	ff	6								
			IND8, Z	E3	ff	6								
			IMM8	F3	ii	2								
			IND16, X	17C3	gggg	6								
			IND16, Y	17D3	gggg	6								
			IND16, Z	17E3	gggg	6								
			EXT	17F3	hh ll	6								
			E, X	27C3	—	6								
			E, Y	27D3	—	6								
			E, Z	27E3	—	6								
ADCD	Add with Carry to D	(D) + (M : M + 1) + C ⇒ D	IND8, X	83	ff	6	—	—	—	—	Δ	Δ	Δ	Δ
			IND8, Y	93	ff	6								
			IND8, Z	A3	ff	6								
			IMM16	37B3	jj kk	4								
			IND16, X	37C3	gggg	6								
			IND16, Y	37D3	gggg	6								
			IND16, Z	37E3	gggg	6								
			EXT	37F3	hh ll	6								
			E, X	2783	—	6								
			E, Y	2793	—	6								
			E, Z	27A3	—	6								
ADCE	Add with Carry to E	(E) + (M : M + 1) + C ⇒ E	IMM16	3733	jj kk	4	—	—	—	—	Δ	Δ	Δ	Δ
			IND16, X	3743	gggg	6								
			IND16, Y	3753	gggg	6								
			IND16, Z	3763	gggg	6								
			EXT	3773	hh ll	6								
ADDA	Add to A	(A) + (M) ⇒ A	IND8, X	41	ff	6	—	—	Δ	—	Δ	Δ	Δ	Δ
			IND8, Y	51	ff	6								
			IND8, Z	61	ff	6								
			IMM8	71	ii	2								
			IND16, X	1741	gggg	6								
			IND16, Y	1751	gggg	6								
			IND16, Z	1761	gggg	6								
			EXT	1771	hh ll	6								
			E, X	2741	—	6								
			E, Y	2751	—	6								
			E, Z	2761	—	6								

Table 6-36 Instruction Set Summary (Continued)

Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
ADDB	Add to B	(B) + (M) ⇒ B	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	C1 D1 E1 F1 17C1 17D1 17E1 17F1 27C1 27D1 27E1	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	Δ	—	Δ	Δ	Δ	Δ
ADDD	Add to D	(D) + (M : M + 1) ⇒ D	IND8, X IND8, Y IND8, Z IMM8 IMM16 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	81 91 A1 FC 37B1 37C1 37D1 37E1 37F1 2781 2791 27A1	ff ff ff ii jj kk gggg gggg gggg hh ll — — —	6 6 6 2 4 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
ADDE	Add to E	(E) + (M : M + 1) ⇒ E	IMM8 IMM16 IND16, X IND16, Y IND16, Z EXT	7C 3731 3741 3751 3761 3771	ii jj kk gggg gggg gggg hh ll	2 4 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
ADE	Add D to E	(E) + (D) ⇒ E	INH	2778	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ADX	Add D to IX	(XK : IX) + («D) ⇒ XK : IX	INH	37CD	—	2	—	—	—	—	—	—	—	—
ADY	Add D to IY	(YK : IY) + («D) ⇒ YK : IY	INH	37DD	—	2	—	—	—	—	—	—	—	—
ADZ	Add D to IZ	(ZK : IZ) + («D) ⇒ ZK : IZ	INH	37ED	—	2	—	—	—	—	—	—	—	—
AEX	Add E to IX	(XK : IX) + («E) ⇒ XK : IX	INH	374D	—	2	—	—	—	—	—	—	—	—
AEY	Add E to IY	(YK : IY) + («E) ⇒ YK : IY	INH	375D	—	2	—	—	—	—	—	—	—	—
AEZ	Add E to IZ	(ZK : IZ) + («E) ⇒ ZK : IZ	INH	376D	—	2	—	—	—	—	—	—	—	—
AIS	Add Immediate Data to Stack Pointer	(SK : SP) + (20 « IMM) ⇒ SK : SP	IMM8 IMM16	3F 373F	ii jj kk	2 4	—	—	—	—	—	—	—	—
AIX	Add Immediate Value to IX	(XK : IX) + (20 « IMM) ⇒ XK : IX	IMM8 IMM16	3C 373C	ii jj kk	2 4	—	—	—	—	—	Δ	—	—
AIY	Add Immediate Value to IY	(YK : IY) + (20 « IMM) ⇒ YK : IY	IMM8 IMM16	3D 373D	ii jj kk	2 4	—	—	—	—	—	Δ	—	—
AIZ	Add Immediate Value to IZ	(ZK : IZ) + (20 « IMM) ⇒ ZK : IZ	IMM8 IMM16	3E 373E	ii jj kk	2 4	—	—	—	—	—	Δ	—	—
ANDA	AND A	(A) • (M) ⇒ A	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	46 56 66 76 1746 1756 1766 1776 2746 2756 2766	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
ANDB	AND B	(B) • (M) ⇒ B	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	C6 D6 E6 F6 17C6 17D6 17E6 17F6 27C6 27D6 27E6	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—

Table 6-36 Instruction Set Summary (Continued)

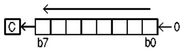
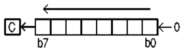
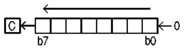
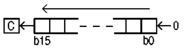
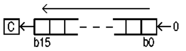
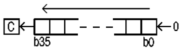
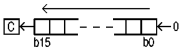
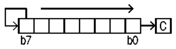
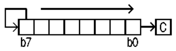
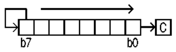
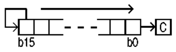
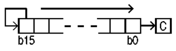
Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
ANDD	AND D	(D) • (M : M + 1) ⇒ D	IND8, X	86	ff	6	—	—	—	—	Δ	Δ	0	—
			IND8, Y	96	ff	6								
			IND8, Z	A6	ff	6								
			IMM16	37B6	jj kk	4								
			IND16, X	37C6	gggg	6								
			IND16, Y	37D6	gggg	6								
			IND16, Z	37E6	gggg	6								
			EXT	37F6	hh ll	6								
			E, X	2786	—	6								
			E, Y	2796	—	6								
			E, Z	27A6	—	6								
ANDE	AND E	(E) • (M : M + 1) ⇒ E	IMM16	3736	jj kk	4	—	—	—	—	Δ	Δ	0	—
			IND16, X	3746	gggg	6								
			IND16, Y	3756	gggg	6								
			IND16, Z	3766	gggg	6								
			EXT	3776	hh ll	6								
ANDP[1]	AND CCR	(CCR) • IMM16⇒ CCR	IMM16	373A	jj kk	4	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ
ASL	Arithmetic Shift Left		IND8, X	04	ff	8	—	—	—	—	Δ	Δ	Δ	Δ
			IND8, Y	14	ff	8								
			IND8, Z	24	ff	8								
			IND16, X	1704	gggg	8								
			IND16, Y	1714	gggg	8								
			IND16, Z	1724	gggg	8								
			EXT	1734	hh ll	8								
ASLA	Arithmetic Shift Left A		INH	3704	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASLB	Arithmetic Shift Left B		INH	3714	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASLD	Arithmetic Shift Left D		INH	27F4	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASLE	Arithmetic Shift Left E		INH	2774	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASLM	Arithmetic Shift Left AM		INH	27B6	—	4	—	Δ	—	Δ	Δ	—	—	Δ
ASLW	Arithmetic Shift Left Word		IND16, X	2704	gggg	8	—	—	—	—	Δ	Δ	Δ	Δ
			IND16, Y	2714	gggg	8								
			IND16, Z	2724	gggg	8								
			EXT	2734	hh ll	8								
ASR	Arithmetic Shift Right		IND8, X	0D	ff	8	—	—	—	—	Δ	Δ	Δ	Δ
			IND8, Y	1D	ff	8								
			IND8, Z	2D	ff	8								
			IND16, X	170D	gggg	8								
			IND16, Y	171D	gggg	8								
			IND16, Z	172D	gggg	8								
			EXT	173D	hh ll	8								
ASRA	Arithmetic Shift Right A		INH	370D	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASRB	Arithmetic Shift Right B		INH	371D	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASRD	Arithmetic Shift Right D		INH	27FD	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASRE	Arithmetic Shift Right E		INH	277D	—	2	—	—	—	—	Δ	Δ	Δ	Δ

Table 6-36 Instruction Set Summary (Continued)

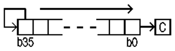
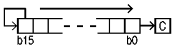
Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
ASRM	Arithmetic Shift Right AM		INH	27BA	—	4	—	—	—	Δ	Δ	—	—	Δ
ASRW	Arithmetic Shift Right Word		IND16, X IND16, Y IND16, Z EXT	270D 271D 272D 273D	gggg gggg gggg hh ll	8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
BCC^2	Branch if Carry Clear	If C = 0, branch	REL8	B4	rr	6, 2	—	—	—	—	—	—	—	—
BCLR	Clear Bit(s)	(M) • ($\overline{\text{Mask}}$) ⇒ M	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	1708 1718 1728 08 18 28 38	mm ff mm ff mm ff mm gggg mm gggg mm gggg mm hh ll	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	0	—
BCLRW	Clear Bit(s) in a Word	(M : M + 1) • ($\overline{\text{Mask}}$) ⇒ M : M + 1	IND16, X IND16, Y IND16, Z EXT	2708 2718 2728 2738	gggg mmmm gggg mmmm gggg mmmm hh ll mmmm	10 10 10 10	—	—	—	—	Δ	Δ	0	—
BCS^2	Branch if Carry Set	If C = 1, branch	REL8	B5	rr	6, 2	—	—	—	—	—	—	—	—
BEQ^2	Branch if Equal	If Z = 1, branch	REL8	B7	rr	6, 2	—	—	—	—	—	—	—	—
BGE^2	Branch if Greater Than or Equal to Zero	If N ⊕ V = 0, branch	REL8	BC	rr	6, 2	—	—	—	—	—	—	—	—
BGND	Enter Background Debug Mode	If BDM enabled, begin debug; else, illegal instruction trap	INH	37A6	—	—	—	—	—	—	—	—	—	—
BGT^2	Branch if Greater Than Zero	If Z ✢ (N ⊕ V) = 0, branch	REL8	BE	rr	6, 2	—	—	—	—	—	—	—	—
BHI^2	Branch if Higher	If C ✢ Z = 0, branch	REL8	B2	rr	6, 2	—	—	—	—	—	—	—	—
BITA	Bit Test A	(A) • (M)	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	49 59 69 79 1749 1759 1769 1779 2749 2759 2769	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
BITB	Bit Test B	(B) • (M)	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	C9 D9 E9 F9 17C9 17D9 17E9 17F9 27C9 27D9 27E9	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
BLE^2	Branch if Less Than or Equal to Zero	If Z ✢ (N ⊕ V) = 1, branch	REL8	BF	rr	6, 2	—	—	—	—	—	—	—	—
BLS^2	Branch if Lower or Same	If C ✢ Z = 1, branch	REL8	B3	rr	6, 2	—	—	—	—	—	—	—	—
BLT^2	Branch if Less Than Zero	If N ⊕ V = 1, branch	REL8	BD	rr	6, 2	—	—	—	—	—	—	—	—
BMI^2	Branch if Minus	If N = 1, branch	REL8	BB	rr	6, 2	—	—	—	—	—	—	—	—
BNE^2	Branch if Not Equal	If Z = 0, branch	REL8	B6	rr	6, 2	—	—	—	—	—	—	—	—
BPL^2	Branch if Plus	If N = 0, branch	REL8	BA	rr	6, 2	—	—	—	—	—	—	—	—

Table 6-36 Instruction Set Summary (Continued)

Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
BRA	Branch Always	If 1 = 1, branch	REL8	B0	rr	6	—	—	—	—	—	—	—	—
BRCLR[2]	Branch if Bit(s) Clear	If (M) • (Mask) = 0, branch	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	CB DB EB 0A 1A 2A 3A	mm ff rr mm ff rr mm ff rr mm gggg rrrr mm gggg rrrr mm gggg rrrr mm hh ll rrrr	10, 12 10, 12 10, 12 10, 14 10, 14 10, 14 10, 14	—	—	—	—	—	—	—	—
BRN	Branch Never	If 1 = 0, branch	REL8	B1	rr	2	—	—	—	—	—	—	—	—
BRSET[2]	Branch if Bit(s) Set	If $(\overline{M})$ • (Mask) = 0, branch	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	8B 9B AB 0B 1B 2B 3B	mm ff rr mm ff rr mm ff rr mm gggg rrrr mm gggg rrrr mm gggg rrrr mm hh ll rrrr	10, 12 10, 12 10, 12 10, 14 10, 14 10, 14 10, 14	—	—	—	—	—	—	—	—
BSET	Set Bit(s)	(M) ✢ (Mask) ⇒ M	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	1709 1719 1729 09 19 29 39	mm ff mm ff mm ff mm gggg mm gggg mm gggg mm hh ll	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	0	Δ
BSETW	Set Bit(s) in Word	(M : M + 1) ✢ (Mask) ⇒ M : M + 1	IND16, X IND16, Y IND16, Z EXT	2709 2719 2729 2739	gggg mmmm gggg mmmm gggg mmmm hh ll mmmm	10 10 10 10	—	—	—	—	Δ	Δ	0	Δ
BSR	Branch to Subroutine	(PK : PC) - 2 ⇒ PK : PC Push (PC) (SK : SP) - 2 ⇒ SK : SP Push (CCR) (SK : SP) - 2 ⇒ SK : SP (PK : PC) + Offset ⇒ PK : PC	REL8	36	rr	10	—	—	—	—	—	—	—	—
BVC[2]	Branch if Overflow Clear	If V = 0, branch	REL8	B8	rr	6, 2	—	—	—	—	—	—	—	—
BVS[2]	Branch if Overflow Set	If V = 1, branch	REL8	B9	rr	6, 2	—	—	—	—	—	—	—	—
CBA	Compare A to B	(A) – (B)	INH	371B	—	2	—	—	—	—	Δ	Δ	Δ	Δ
CLR	Clear a Byte in Memory	$00 ⇒ M	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	05 15 25 1705 1715 1725 1735	ff ff ff gggg gggg gggg hh ll	4 4 4 6 6 6 6	—	—	—	—	0	1	0	0
CLRA	Clear A	$00 ⇒ A	INH	3705	—	2	—	—	—	—	0	1	0	0
CLRB	Clear B	$00 ⇒ B	INH	3715	—	2	—	—	—	—	0	1	0	0
CLRD	Clear D	$0000 ⇒ D	INH	27F5	—	2	—	—	—	—	0	1	0	0
CLRE	Clear E	$0000 ⇒ E	INH	2775	—	2	—	—	—	—	0	1	0	0
CLRM	Clear AM	$000000000 ⇒ AM[35:0]	INH	27B7	—	2	—	0	—	0	—	—	—	—
CLRW	Clear a Word in Memory	$0000 ⇒ M : M + 1	IND16, X IND16, Y IND16, Z EXT	2705 2715 2725 2735	gggg gggg gggg hh ll	6 6 6 6	—	—	—	—	0	1	0	0

Table 6-36 Instruction Set Summary (Continued)

Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
CMPA	Compare A to Memory	(A) – (M)	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	48 58 68 78 1748 1758 1768 1778 2748 2758 2768	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
CMPB	Compare B to Memory	(B) – (M)	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	C8 D8 E8 F8 17C8 17D8 17E8 17F8 27C8 27D8 27E8	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
COM	One's Complement	$FF – (M) ⇒ M, or M̄ ⇒ M	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	00 10 20 1700 1710 1720 1730	ff ff ff gggg gggg gggg hh ll	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	0	1
COMA	One's Complement A	$FF – (A) ⇒ A, or M̄ ⇒ A	INH	3700	—	2	—	—	—	—	Δ	Δ	0	1
COMB	One's Complement B	$FF – (B) ⇒ B, or B̄ ⇒ B	INH	3710	—	2	—	—	—	—	Δ	Δ	0	1
COMD	One's Complement D	$FFFF – (D) ⇒ D, or D̄ ⇒ D	INH	27F0	—	2	—	—	—	—	Δ	Δ	0	1
COME	One's Complement E	$FFFF – (E) ⇒ E, or Ē ⇒ E	INH	2770	—	2	—	—	—	—	Δ	Δ	0	1
COMW	One's Complement Word	$FFFF – M : M + 1 ⇒ M : M + 1, or (M̄ : M̄ + 1̄) ⇒ M : M + 1	IND16, X IND16, Y IND16, Z EXT	2700 2710 2720 2730	gggg gggg gggg hh ll	8 8 8 8	—	—	—	—	Δ	Δ	0	1
CPD	Compare D to Memory	(D) – (M : M + 1)	IND8, X IND8, Y IND8, Z IMM16 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	88 98 A8 37B8 37C8 37D8 37E8 37F8 2788 2798 27A8	ff ff ff jj kk gggg gggg gggg hh ll — — —	6 6 6 4 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
CPE	Compare E to Memory	(E) – (M : M + 1)	IMM16 IND16, X IND16, Y IND16, Z EXT	3738 3748 3758 3768 3778	jjkk gggg gggg gggg hhll	4 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
CPS	Compare Stack Pointer to Memory	(SP) – (M : M + 1)	IND8, X IND8, Y IND8, Z IMM16 IND16, X IND16, Y IND16, Z EXT	4F 5F 6F 377F 174F 175F 176F 177F	ff ff ff jj kk gggg gggg gggg hh ll	6 6 6 4 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ

Table 6-36 Instruction Set Summary (Continued)

Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
CPX	Compare IX to Memory	(IX) – (M : M + 1)	IND8, X IND8, Y IND8, Z IMM16 IND16, X IND16, Y IND16, Z EXT	4C 5C 6C 377C 174C 175C 176C 177C	ff ff ff jj kk gggg gggg gggg hh ll	6 6 6 4 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
CPY	Compare IY to Memory	(IY) – (M : M + 1)	IND8, X IND8, Y IND8, Z IMM16 IND16, X IND16, Y IND16, Z EXT	4D 5D 6D 377D 174D 175D 176D 177D	ff ff ff jj kk gggg gggg gggg hh ll	6 6 6 4 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
CPZ	Compare IZ to Memory	(IZ) – (M : M + 1)	IND8, X IND8, Y IND8, Z IMM16 IND16, X IND16, Y IND16, Z EXT	4E 5E 6E 377E 174E 175E 176E 177E	ff ff ff jj kk gggg gggg gggg hh ll	6 6 6 4 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
DAA	Decimal Adjust A	$(A)_{10}$	INH	3721	—	2	—	—	—	—	Δ	Δ	U	Δ
DEC	Decrement Memory	(M) – $01 ⇒ M	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	01 11 21 1701 1711 1721 1731	ff ff ff gggg gggg gggg hh ll	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	Δ	—
DECA	Decrement A	(A) – $01 ⇒ A	INH	3701	—	2	—	—	—	—	Δ	Δ	Δ	—
DECB	Decrement B	(B) – $01 ⇒ B	INH	3711	—	2	—	—	—	—	Δ	Δ	Δ	—
DECW	Decrement Memory Word	(M : M + 1) – $0001 ⇒ M : M + 1	IND16, X IND16, Y IND16, Z EXT	2701 2711 2721 2731	gggg gggg gggg hh ll	8 8 8 8	—	—	—	—	Δ	Δ	Δ	—
EDIV	Extended Unsigned Integer Divide	(E : D) / (IX) Quotient ⇒ IX Remainder ⇒ D	INH	3728	—	24	—	—	—	—	Δ	Δ	Δ	Δ
EDIVS	Extended Signed Integer Divide	(E : D) / (IX) Quotient ⇒ IX Remainder ⇒ D	INH	3729	—	38	—	—	—	—	Δ	Δ	Δ	Δ
EMUL	Extended Unsigned Multiply	(E) ∗ (D) ⇒ E : D	INH	3725	—	10	—	—	—	—	Δ	Δ	—	Δ
EMULS	Extended Signed Multiply	(E) ∗ (D) ⇒ E : D	INH	3726	—	8	—	—	—	—	Δ	Δ	—	Δ
EORA	Exclusive OR A	(A) ⊕ (M) ⇒ A	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	44 54 64 74 1744 1754 1764 1774 2744 2754 2764	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—

Table 6-36 Instruction Set Summary (Continued)

Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
EORB	Exclusive OR B	(B) ⊕ (M) ⇒ B	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	C4 D4 E4 F4 17C4 17D4 17E4 17F4 27C4 27D4 27E4	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
EORD	Exclusive OR D	(D) ⊕ (M : M + 1) ⇒ D	IND8, X IND8, Y IND8, Z IMM16 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	84 94 A4 37B4 37C4 37D4 37E4 37F4 2784 2794 27A4	ff ff ff jj kk gggg gggg gggg hh ll — — —	6 6 6 4 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
EORE	Exclusive OR E	(E) ⊕ (M : M + 1) ⇒ E	IMM16 IND16, X IND16, Y IND16, Z EXT	3734 3744 3754 3764 3774	jj kk gggg gggg gggg hh ll	4 6 6 6 6	—	—	—	—	Δ	Δ	0	—
FDIV	Fractional Unsigned Divide	(D) / (IX) ⇒ IX Remainder ⇒ D	INH	372B	—	22	—	—	—	—	—	Δ	Δ	Δ
FMULS	Fractional Signed Multiply	(E) ∗ (D) ⇒ E : D[31:1] 0 ⇒ D[0]	INH	3727	—	8	—	—	—	—	Δ	Δ	Δ	Δ
IDIV	Integer Divide	(D) / (IX) ⇒ IX Remainder ⇒ D	INH	372A	—	22	—	—	—	—	—	Δ	0	Δ
INC	Increment Memory	(M) + $01 ⇒ M	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	03 13 23 1703 1713 1723 1733	ff ff ff gggg gggg gggg hh ll	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	Δ	—
INCA	Increment A	(A) + $01 ⇒ A	INH	3703	—	2	—	—	—	—	Δ	Δ	Δ	—
INCB	Increment B	(B) + $01 ⇒ B	INH	3713	—	2	—	—	—	—	Δ	Δ	Δ	—
INCW	Increment Memory Word	(M : M + 1) + $0001 ⇒ M : M + 1	IND16, X IND16, Y IND16, Z EXT	2703 2713 2723 2733	gggg gggg gggg hh ll	8 8 8 8	—	—	—	—	Δ	Δ	Δ	—
JMP	Jump	〈ea〉 ⇒ PK : PC	EXT20 IND20, X IND20, Y IND20, Z	7A 4B 5B 6B	zb hh ll zg gggg zg gggg zg gggg	6 8 8 8	—	—	—	—	—	—	—	—
JSR	Jump to Subroutine	Push (PC) (SK : SP) – $0002 ⇒ SK : SP Push (CCR) (SK : SP) – $0002 ⇒ SK : SP 〈ea〉 ⇒ PK : PC	EXT20 IND20, X IND20, Y IND20, Z	FA 89 99 A9	zb hh ll zg gggg zg gggg zg gggg	10 12 12 12	—	—	—	—	—	—	—	—
LBCC[2]	Long Branch if Carry Clear	If C = 0, branch	REL16	3784	rrrr	6, 4	—	—	—	—	—	—	—	—
LBCS[2]	Long Branch if Carry Set	If C = 1, branch	REL16	3785	rrrr	6, 4	—	—	—	—	—	—	—	—
LBEQ[2]	Long Branch if Equal to Zero	If Z = 1, branch	REL16	3787	rrrr	6, 4	—	—	—	—	—	—	—	—
LBEV[2]	Long Branch if EV Set	If EV = 1, branch	REL16	3791	rrrr	6, 4	—	—	—	—	—	—	—	—
LBGE[2]	Long Branch if Greater Than or Equal to Zero	If N ⊕ V = 0, branch	REL16	378C	rrrr	6, 4	—	—	—	—	—	—	—	—
LBGT[2]	Long Branch if Greater Than Zero	If Z ✛ (N ⊕ V) = 0, branch	REL16	378E	rrrr	6, 4	—	—	—	—	—	—	—	—

Table 6-36 Instruction Set Summary (Continued)

Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
LBHI [2]	Long Branch if Higher	If C ✢ Z = 0, branch	REL16	3782	rrrr	6, 4	—	—	—	—	—	—	—	—
LBLE[2]	Long Branch if Less Than or Equal to Zero	If Z ✢ (N ⊕ V) = 1, branch	REL16	378F	rrrr	6, 4	—	—	—	—	—	—	—	—
LBLS[2]	Long Branch if Lower or Same	If C ✢ Z = 1, branch	REL16	3783	rrrr	6, 4	—	—	—	—	—	—	—	—
LBLT[2]	Long Branch if Less Than Zero	If N ⊕ V = 1, branch	REL16	378D	rrrr	6, 4	—	—	—	—	—	—	—	—
LBMI[2]	Long Branch if Minus	If N = 1, branch	REL16	378B	rrrr	6, 4	—	—	—	—	—	—	—	—
LBMV[2]	Long Branch if MV Set	If MV = 1, branch	REL16	3790	rrrr	6, 4	—	—	—	—	—	—	—	—
LBNE[2]	Long Branch if Not Equal to Zero	If Z = 0, branch	REL16	3786	rrrr	6, 4	—	—	—	—	—	—	—	—
LBPL[2]	Long Branch if Plus	If N = 0, branch	REL16	378A	rrrr	6, 4	—	—	—	—	—	—	—	—
LBRA	Long Branch Always	If 1 = 1, branch	REL16	3780	rrrr	6	—	—	—	—	—	—	—	—
LBRN	Long Branch Never	If 1 = 0, branch	REL16	3781	rrrr	6	—	—	—	—	—	—	—	—
LBSR	Long Branch to Subroutine	Push (PC) (SK : SP) – 2 ⇒ SK : SP Push (CCR) (SK : SP) – 2 ⇒ SK : SP (PK : PC) + Offset ⇒ PK : PC	REL16	27F9	rrrr	10	—	—	—	—	—	—	—	—
LBVC[2]	Long Branch if Overflow Clear	If V = 0, branch	REL16	3788	rrrr	6, 4	—	—	—	—	—	—	—	—
LBVS[2]	Long Branch if Overflow Set	If V = 1, branch	REL16	3789	rrrr	6, 4	—	—	—	—	—	—	—	—
LDAA	Load A	(M) ⇒ A	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	45 55 65 75 1745 1755 1765 1775 2745 2755 2765	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
LDAB	Load B	(M) ⇒ B	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	C5 D5 E5 F5 17C5 17D5 17E5 17F5 27C5 27D5 27E5	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	Δ
LDD	Load D	(M : M + 1) ⇒ D	IND8, X IND8, Y IND8, Z IMM16 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	85 95 A5 37B5 37C5 37D5 37E5 37F5 2785 2795 27A5	ff ff ff jj kk gggg gggg gggg hh ll — — —	6 6 6 4 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
LDE	Load E	(M : M + 1) ⇒ E	IMM16 IND16, X IND16, Y IND16, Z EXT	3735 3745 3755 3765 3775	jj kk gggg gggg gggg hh ll	4 6 6 6 6	—	—	—	—	Δ	Δ	0	—
LDED	Load Concatenated E and D	(M : M + 1) ⇒ E (M + 2 : M + 3) ⇒ D	EXT	2771	hh ll	8	—	—	—	—	—	—	—	—

Table 6-36 Instruction Set Summary (Continued)

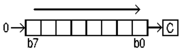
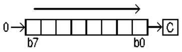
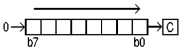
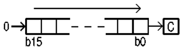
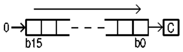
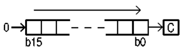
Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
LDHI	Initialize H and I	$(M : M + 1)_X \Rightarrow H\ R$ $(M : M + 1)_Y \Rightarrow I\ R$	INH	27B0	—	8	—	—	—	—	—	—	—	—
LDS	Load SP	$(M : M + 1) \Rightarrow SP$	IND8, X	CF	ff	6	—	—	—	—	Δ	Δ	0	—
			IND8, Y	DF	ff	6								
			IND8, Z	EF	ff	6								
			IND16, X	17CF	gggg	6								
			IND16, Y	17DF	gggg	6								
			IND16, Z	17EF	gggg	6								
			EXT	17FF	hh ll	6								
			IMM16	37BF	jj kk	4								
LDX	Load IX	$(M : M + 1) \Rightarrow IX$	IND8, X	CC	ff	6	—	—	—	—	Δ	Δ	0	—
			IND8, Y	DC	ff	6								
			IND8, Z	EC	ff	6								
			IMM16	37BC	jj kk	4								
			IND16, X	17CC	gggg	6								
			IND16, Y	17DC	gggg	6								
			IND16, Z	17EC	gggg	6								
			EXT	17FC	hh ll	6								
LDY	Load IY	$(M : M + 1) \Rightarrow IY$	IND8, X	CD	ff	6	—	—	—	—	Δ	Δ	0	—
			IND8, Y	DD	ff	6								
			IND8, Z	ED	ff	6								
			IMM16	37BD	jj kk	4								
			IND16, X	17CD	gggg	6								
			IND16, Y	17DD	gggg	6								
			IND16, Z	17ED	gggg	6								
			EXT	17FD	hh ll	6								
LDZ	Load IZ	$(M : M + 1) \Rightarrow IZ$	IND8, X	CE	ff	6	—	—	—	—	Δ	Δ	0	—
			IND8, Y	DE	ff	6								
			IND8, Z	EE	ff	6								
			IMM16	37BE	jj kk	4								
			IND16, X	17CE	gggg	6								
			IND16, Y	17DE	gggg	6								
			IND16, Z	17EE	gggg	6								
			EXT	17FE	hh ll	6								
LPSTOP	Low Power Stop	If $\overline{S}$ then STOP else NOP	INH	27F1	—	4, 20	—	—	—	—	—	—	—	—
LSR	Logical Shift Right		IND8, X	0F	ff	8	—	—	—	—	0	Δ	Δ	Δ
			IND8, Y	1F	ff	8								
			IND8, Z	2F	ff	8								
			IND16, X	170F	gggg	8								
			IND16, Y	171F	gggg	8								
			IND16, Z	172F	gggg	8								
			EXT	173F	hh ll	8								
LSRA	Logical Shift Right A		INH	370F	—	2	—	—	—	—	0	Δ	Δ	Δ
LSRB	Logical Shift Right B		INH	371F	—	2	—	—	—	—	0	Δ	Δ	Δ
LSRD	Logical Shift Right D		INH	27FF	—	2	—	—	—	—	0	Δ	Δ	Δ
LSRE	Logical Shift Right E		INH	277F	—	2	—	—	—	—	0	Δ	Δ	Δ
LSRW	Logical Shift Right Word		IND16, X	270F	gggg	8	—	—	—	—	0	Δ	Δ	Δ
			IND16, Y	271F	gggg	8								
			IND16, Z	272F	gggg	8								
			EXT	273F	hh ll	8								

Table 6-36 Instruction Set Summary (Continued)

Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
MAC	Multiply and Accumulate Signed 16-Bit Fractions	(HR) ∗ (IR) ⇒ E : D (AM) + (E : D) ⇒ AM Qualified (IX) ⇒ IX Qualified (IY) ⇒ IY (HR) ⇒ IZ (M : M + 1)$_X$ ⇒ HR (M : M + 1)$_Y$ ⇒ IR	IMM8	7B	xoyo	12	—	Δ	—	Δ	—	—	Δ	—
MOVB	Move Byte	(M$_1$) ⇒ M$_2$	IXP to EXT EXT to IXP EXT to EXT	30 32 37FE	ff hh ll ff hh ll hh ll hh ll	8 8 10	—	—	—	—	Δ	Δ	0	—
MOVW	Move Word	(M : M + 1$_1$) ⇒ M : M + 1$_2$	IXP to EXT EXT to IXP EXT to EXT	31 33 37FF	ff hh ll ff hh ll hh ll hh ll	8 8 10	—	—	—	—	Δ	Δ	0	—
MUL	Multiply	(A) ∗ (B) ⇒ D	INH	3724	—	10	—	—	—	—	—	—	—	Δ
NEG	Negate Memory	$00 – (M) ⇒ M	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	02 12 22 1702 1712 1722 1732	ff ff ff gggg gggg gggg hh ll	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
NEGA	Negate A	$00 – (A) ⇒ A	INH	3702	—	2	—	—	—	—	Δ	Δ	Δ	Δ
NEGB	Negate B	$00 – (B) ⇒ B	INH	3712	—	2	—	—	—	—	Δ	Δ	Δ	Δ
NEGD	Negate D	$0000 – (D) ⇒ D	INH	27F2	—	2	—	—	—	—	Δ	Δ	Δ	Δ
NEGE	Negate E	$0000 – (E) ⇒ E	INH	2772	—	2	—	—	—	—	Δ	Δ	Δ	Δ
NEGW	Negate Memory Word	$0000 – (M : M + 1) ⇒ M : M + 1	IND16, X IND16, Y IND16, Z EXT	2702 2712 2722 2732	gggg gggg gggg hh ll	8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
NOP	Null Operation	—	INH	274C	—	2	—	—	—	—	—	—	—	—
ORAA	OR A	(A) ✛ (M) ⇒ A	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	47 57 67 77 1747 1757 1767 1777 2747 2757 2767	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
ORAB	OR B	(B) ✛ (M) ⇒ B	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	C7 D7 E7 F7 17C7 17D7 17E7 17F7 27C7 27D7 27E7	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
ORD	OR D	(D) ✛ (M : M + 1) ⇒ D	IND8, X IND8, Y IND8, Z IMM16 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	87 97 A7 37B7 37C7 37D7 37E7 37F7 2787 2797 27A7	ff ff ff jj kk gggg gggg gggg hh ll — — —	6 6 6 4 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—

Table 6-36 Instruction Set Summary (Continued)

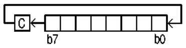
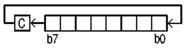
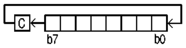
Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
ORE	OR E	(E) ✢ (M : M + 1) ⇒ E	IMM16 IND16, X IND16, Y IND16, Z EXT	3737 3747 3757 3767 3777	jj kk gggg gggg gggg hh ll	4 6 6 6 6	—	—	—	—	Δ	Δ	0	—
ORP [1]	OR Condition Code Register	(CCR) ✢ IMM16 ⇒ CCR	IMM16	373B	jj kk	4	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ
PSHA	Push A	(SK : SP) + $0001 ⇒ SK : SP Push (A) (SK : SP) – $0002 ⇒ SK : SP	INH	3708	—	4	—	—	—	—	—	—	—	—
PSHB	Push B	(SK : SP) + $0001 ⇒ SK : SP Push (B) (SK : SP) – $0002 ⇒ SK : SP	INH	3718	—	4	—	—	—	—	—	—	—	—
PSHM	Push Multiple Registers Mask bits: 0 = D 1 = E 2 = IX 3 = IY 4 = IZ 5 = K 6 = CCR 7 = (Reserved)	For mask bits 0 to 7: If mask bit set Push register (SK : SP) – 2 ⇒ SK : SP	IMM8	34	ii	4 + 2N N = number of registers pushed	—	—	—	—	—	—	—	—
PSHMAC	Push MAC Registers	MAC Registers ⇒ Stack	INH	27B8	—	14	—	—	—	—	—	—	—	—
PULA	Pull A	(SK : SP) + $0002 ⇒ SK : SP Pull (A) (SK : SP) – $0001 ⇒ SK : SP	INH	3709	—	6	—	—	—	—	—	—	—	—
PULB	Pull B	(SK : SP) + $0002 ⇒ SK : SP Pull (B) (SK : SP) – $0001 ⇒ SK : SP	INH	3719	—	6	—	—	—	—	—	—	—	—
PULM[1]	Pull Multiple Registers Mask bits: 0 = CCR[15:4] 1 = K 2 = IZ 3 = IY 4 = IX 5 = E 6 = D 7 = (Reserved)	For mask bits 0 to 7: If mask bit set (SK : SP) + 2 ⇒ SK : SP Pull register	IMM8	35	ii	4+2(N+1) N = number of registers pulled	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ
PULMAC	Pull MAC State	Stack ⇒ MAC Registers	INH	27B9	—	16	—	—	—	—	—	—	—	—
RMAC	Repeating Multiply and Accumulate Signed 16-Bit Fractions	Repeat until (E) < 0 (AM) + (H) ∗ (I) ⇒ AM Qualified (IX) ⇒ IX; Qualified (IY) ⇒ IY; $(M : M + 1)_X \Rightarrow H$; $(M : M + 1)_Y \Rightarrow I$ (E) – 1 ⇒ E Until (E) < $0000	IMM8	FB	xoyo	6 + 12 per iteration	—	Δ	—	Δ	—	—	—	—
ROL	Rotate Left		IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	0C 1C 2C 170C 171C 172C 173C	ff ff ff gggg gggg gggg hh ll	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
ROLA	Rotate Left A		INH	370C	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ROLB	Rotate Left B		INH	371C	—	2	—	—	—	—	Δ	Δ	Δ	Δ

Table 6-36 Instruction Set Summary (Continued)

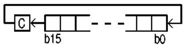
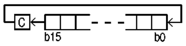
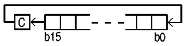
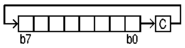
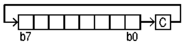
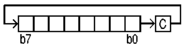
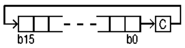
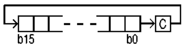
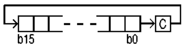
Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
ROLD	Rotate Left D		INH	27FC	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ROLE	Rotate Left E		INH	277C	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ROLW	Rotate Left Word		IND16, X IND16, Y IND16, Z EXT	270C 271C 272C 273C	gggg gggg gggg hh ll	8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
ROR	Rotate Right Byte		IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	0E 1E 2E 170E 171E 172E 173E	ff ff ff gggg gggg gggg hh ll	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
RORA	Rotate Right A		INH	370E	—	2	—	—	—	—	Δ	Δ	Δ	Δ
RORB	Rotate Right B		INH	371E	—	2	—	—	—	—	Δ	Δ	Δ	Δ
RORD	Rotate Right D		INH	27FE	—	2	—	—	—	—	Δ	Δ	Δ	Δ
RORE	Rotate Right E		INH	277E	—	2	—	—	—	—	Δ	Δ	Δ	Δ
RORW	Rotate Right Word		IND16, X IND16, Y IND16, Z EXT	270E 271E 272E 273E	gggg gggg gggg hh ll	8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
RTI[3]	Return from Interrupt	(SK : SP) + 2 ⇒ SK : SP Pull CCR (SK : SP) + 2 ⇒ SK : SP Pull PC (PK : PC) – 6 ⇒ PK : PC	INH	2777	—	12	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ
RTS[4]	Return from Subroutine	(SK : SP) + 2 ⇒ SK : SP Pull PK (SK : SP) + 2 ⇒ SK : SP Pull PC (PK : PC) – 2 ⇒ PK : PC	INH	27F7	—	12	—	—	—	—	—	—	—	—
SBA	Subtract B from A	(A) – (B) ⇒ A	INH	370A	—	2	—	—	—	—	Δ	Δ	Δ	Δ
SBCA	Subtract with Carry from A	(A) – (M) – C ⇒ A	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	42 52 62 72 1742 1752 1762 1772 2742 2752 2762	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ

Table 6-36 Instruction Set Summary (Continued)

Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
SBCB	Subtract with Carry from B	(B) – (M) – C ⇒ B	IND8, X	C2	ff	6	—	—	—	—	Δ	Δ	Δ	Δ
			IND8, Y	D2	ff	6								
			IND8, Z	E2	ff	6								
			IMM8	F2	ii	2								
			IND16, X	17C2	gggg	6								
			IND16, Y	17D2	gggg	6								
			IND16, Z	17E2	gggg	6								
			EXT	17F2	hh ll	6								
			E, X	27C2	—	6								
			E, Y	27D2	—	6								
			E, Z	27E2	—	6								
SBCD	Subtract with Carry from D	(D) – (M : M + 1) – C ⇒ D	IND8, X	82	ff	6	—	—	—	—	Δ	Δ	Δ	Δ
			IND8, Y	92	ff	6								
			IND8, Z	A2	ff	6								
			IMM16	37B2	jj kk	4								
			IND16, X	37C2	gggg	6								
			IND16, Y	37D2	gggg	6								
			IND16, Z	37E2	gggg	6								
			EXT	37F2	hh ll	6								
			E, X	2782	—	6								
			E, Y	2792	—	6								
			E, Z	27A2	—	6								
SBCE	Subtract with Carry from E	(E) – (M : M + 1) – C ⇒ E	IMM16	3732	jj kk	4	—	—	—	—	Δ	Δ	Δ	Δ
			IND16, X	3742	gggg	6								
			IND16, Y	3752	gggg	6								
			IND16, Z	3762	gggg	6								
			EXT	3772	hh ll	6								
SDE	Subtract D from E	(E) – (D)⇒ E	INH	2779	—	2	—	—	—	—	Δ	Δ	Δ	Δ
STAA	Store A	(A) ⇒ M	IND8, X	4A	ff	4	—	—	—	—	Δ	Δ	0	—
			IND8, Y	5A	ff	4								
			IND8, Z	6A	ff	4								
			IND16, X	174A	gggg	6								
			IND16, Y	175A	gggg	6								
			IND16, Z	176A	gggg	6								
			EXT	177A	hh ll	6								
			E, X	274A	—	4								
			E, Y	275A	—	4								
			E, Z	276A	—	4								
STAB	Store B	(B) ⇒ M	IND8, X	CA	ff	4	—	—	—	—	Δ	Δ	0	—
			IND8, Y	DA	ff	4								
			IND8, Z	EA	ff	4								
			IND16, X	17CA	gggg	6								
			IND16, Y	17DA	gggg	6								
			IND16, Z	17EA	gggg	6								
			EXT	17FA	hh ll	6								
			E, X	27CA	—	4								
			E, Y	27DA	—	4								
			E, Z	27EA	—	4								
STD	Store D	(D) ⇒ M : M + 1	IND8, X	8A	ff	4	—	—	—	—	Δ	Δ	0	—
			IND8, Y	9A	ff	4								
			IND8, Z	AA	ff	4								
			IND16, X	37CA	gggg	6								
			IND16, Y	37DA	gggg	6								
			IND16, Z	37EA	gggg	6								
			EXT	37FA	hh ll	6								
			E, X	278A	—	6								
			E, Y	279A	—	6								
			E, Z	27AA	—	6								
STE	Store E	(E) ⇒ M : M + 1	IND16, X	374A	gggg	6	—	—	—	—	Δ	Δ	0	—
			IND16, Y	375A	gggg	6								
			IND16, Z	376A	gggg	6								
			EXT	377A	hh ll	6								
STED	Store Concatenated D and E	(E) ⇒ M : M + 1 (D) ⇒ M + 2 : M + 3	EXT	2773	hh ll	8	—	—	—	—	—	—	—	—

Table 6-36 Instruction Set Summary (Continued)

Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
STS	Store Stack Pointer	(SP) ⇒ M : M + 1	IND8, X	8F	ff	4	—	—	—	—	Δ	Δ	0	—
			IND8, Y	9F	ff	4								
			IND8, Z	AF	ff	4								
			IND16, X	178F	gggg	6								
			IND16, Y	179F	gggg	6								
			IND16, Z	17AF	gggg	6								
			EXT	17BF	hh ll	6								
STX	Store IX	(IX) ⇒ M : M + 1	IND8, X	8C	ff	4	—	—	—	—	Δ	Δ	0	—
			IND8, Y	9C	ff	4								
			IND8, Z	AC	ff	4								
			IND16, X	178C	gggg	6								
			IND16, Y	179C	gggg	6								
			IND16, Z	17AC	gggg	6								
			EXT	17BC	hh ll	6								
STY	Store IY	(IY) ⇒ M : M + 1	IND8, X	8D	ff	4	—	—	—	—	Δ	Δ	0	—
			IND8, Y	9D	ff	4								
			IND8, Z	AD	ff	4								
			IND16, X	178D	gggg	6								
			IND16, Y	179D	gggg	6								
			IND16, Z	17AD	gggg	6								
			EXT	17BD	hh ll	6								
STZ	Store Z	(IZ) ⇒ M : M + 1	IND8, X	8E	ff	4	—	—	—	—	Δ	Δ	0	—
			IND8, Y	9E	ff	4								
			IND8, Z	AE	ff	4								
			IND16, X	178E	gggg	6								
			IND16, Y	179E	gggg	6								
			IND16, Z	17AE	gggg	6								
			EXT	17BE	hh ll	6								
SUBA	Subtract from A	(A) – (M) ⇒ A	IND8, X	40	ff	6	—	—	—	—	Δ	Δ	Δ	Δ
			IND8, Y	50	ff	6								
			IND8, Z	60	ff	6								
			IMM8	70	ii	2								
			IND16, X	1740	gggg	6								
			IND16, Y	1750	gggg	6								
			IND16, Z	1760	gggg	6								
			EXT	1770	hh ll	6								
			E, X	2740	—	6								
			E, Y	2750	—	6								
			E, Z	2760	—	6								
SUBB	Subtract from B	(B) – (M) ⇒ B	IND8, X	C0	ff	6	—	—	—	—	Δ	Δ	Δ	Δ
			IND8, Y	D0	ff	6								
			IND8, Z	E0	ff	6								
			IMM8	F0	ii	2								
			IND16, X	17C0	gggg	6								
			IND16, Y	17D0	gggg	6								
			IND16, Z	17E0	gggg	6								
			EXT	17F0	hh ll	6								
			E, X	27C0	—	6								
			E, Y	27D0	—	6								
			E, Z	27E0	—	6								
SUBD	Subtract from D	(D) – (M : M + 1) ⇒ D	IND8, X	80	ff	6	—	—	—	—	Δ	Δ	Δ	Δ
			IND8, Y	90	ff	6								
			IND8, Z	A0	ff	6								
			IMM16	37B0	jj kk	4								
			IND16, X	37C0	gggg	6								
			IND16, Y	37D0	gggg	6								
			IND16, Z	37E0	gggg	6								
			EXT	37F0	hh ll	6								
			E, X	2780	—	6								
			E, Y	2790	—	6								
			E, Z	27A0	—	6								
SUBE	Subtract from E	(E) – (M : M + 1) ⇒ E	IMM16	3730	jj kk	4	—	—	—	—	Δ	Δ	Δ	Δ
			IND16, X	3740	gggg	6								
			IND16, Y	3750	gggg	6								
			IND16, Z	3760	gggg	6								
			EXT	3770	hh ll	6								

Table 6-36 Instruction Set Summary (Continued)

Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
SWI	Software Interrupt	(PK : PC) + $0002 ⇒ PK : PC Push (PC) (SK : SP) – $0002 ⇒ SK : SP Push (CCR) (SK : SP) – $0002 ⇒ SK : SP $0 ⇒ PK SWI Vector ⇒ PC	INH	3720	—	16	—	—	—	—	—	—	—	—
SXT	Sign Extend B into A	If B7 = 1 then $FF ⇒ A else $00 ⇒ A	INH	27F8	—	2	—	—	—	—	∆	∆	—	—
TAB	Transfer A to B	(A) ⇒ B	INH	3717	—	2	—	—	—	—	∆	∆	0	—
TAP	Transfer A to CCR	(A[7:0]) ⇒ CCR[15:8]	INH	37FD	—	4	∆	∆	∆	∆	∆	∆	∆	∆
TBA	Transfer B to A	(B) ⇒ A	INH	3707	—	2	—	—	—	—	∆	∆	0	—
TBEK	Transfer B to EK	(B[3:0]) ⇒ EK	INH	27FA	—	2	—	—	—	—	—	—	—	—
TBSK	Transfer B to SK	(B[3:0]) ⇒ SK	INH	379F	—	2	—	—	—	—	—	—	—	—
TBXK	Transfer B to XK	(B[3:0]) ⇒ XK	INH	379C	—	2	—	—	—	—	—	—	—	—
TBYK	Transfer B to YK	(B[3:0]) ⇒ YK	INH	379D	—	2	—	—	—	—	—	—	—	—
TBZK	Transfer B to ZK	(B[3:0]) ⇒ ZK	INH	379E	—	2	—	—	—	—	—	—	—	—
TDE	Transfer D to E	(D) ⇒ E	INH	277B	—	2	—	—	—	—	∆	∆	0	—
TDMSK	Transfer D to XMSK : YMSK	(D[15:8]) ⇒ X MASK (D[7:0]) ⇒ Y MASK	INH	372F	—	2	—	—	—	—	—	—	—	—
TDP[1]	Transfer D to CCR	(D) ⇒ CCR[15:4]	INH	372D	—	4	∆	∆	∆	∆	∆	∆	∆	∆
TED	Transfer E to D	(E) ⇒ D	INH	27FB	—	2	—	—	—	—	∆	∆	0	—
TEDM	Transfer E and D to AM[31:0] Sign Extend AM	(E) ⇒ AM[31:16] (D) ⇒ AM[15:0] AM[35:32] = AM31	INH	27B1	—	4	—	0	—	0	—	—	—	—
TEKB	Transfer EK to B	(EK) ⇒ B[3:0] $0 ⇒ B[7:4]	INH	27BB	—	2	—	—	—	—	—	—	—	—
TEM	Transfer E to AM[31:16] Sign Extend AM Clear AM LSB	(E) ⇒ AM[31:16] $00 ⇒ AM[15:0] AM[35:32] = AM31	INH	27B2	—	4	—	0	—	0	—	—	—	—
TMER	Transfer Rounded AM to E	Rounded (AM) ⇒ Temp If (SM • (EV ✢ MV)) then Saturation Value ⇒ E else Temp[31:16] ⇒ E	INH	27B4	—	6	—	∆	—	∆	∆	∆	—	—
TMET	Transfer Truncated AM to E	If (SM • (EV ✢ MV)) then Saturation Value ⇒ E else AM[31:16] ⇒ E	INH	27B5	—	2	—	—	—	—	∆	∆	—	—
TMXED	Transfer AM to IX : E : D	AM[35:32] ⇒ IX[3:0] AM35 ⇒ IX[15:4] AM[31:16] ⇒ E AM[15:0] ⇒ D	INH	27B3	—	6	—	—	—	—	—	—	—	—
TPA	Transfer CCR to A	(CCR[15:8]) ⇒ A	INH	37FC	—	2	—	—	—	—	—	—	—	—
TPD	Transfer CCR to D	(CCR) ⇒ D	INH	372C	—	2	—	—	—	—	—	—	—	—
TSKB	Transfer SK to B	(SK) ⇒ B[3:0] $0 ⇒ B[7:4]	INH	37AF	—	2	—	—	—	—	—	—	—	—
TST	Test Byte Zero or Minus	(M) – $00	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	06 16 26 1706 1716 1726 1736	ff ff ff gggg gggg gggg hh ll	6 6 6 6 6 6 6	—	—	—	—	∆	∆	0	0
TSTA	Test A for Zero or Minus	(A) – $00	INH	3706	—	2	—	—	—	—	∆	∆	0	0
TSTB	Test B for Zero or Minus	(B) – $00	INH	3716	—	2	—	—	—	—	∆	∆	0	0
TSTD	Test D for Zero or Minus	(D) – $0000	INH	27F6	—	2	—	—	—	—	∆	∆	0	0
TSTE	Test E for Zero or Minus	(E) – $0000	INH	2776	—	2	—	—	—	—	∆	∆	0	0

Table 6-36 Instruction Set Summary (Continued)

Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
TSTW	Test for Zero or Minus Word	(M : M + 1) – $0000	IND16, X IND16, Y IND16, Z EXT	2706 2716 2726 2736	gggg gggg gggg hh ll	6 6 6 6	—	—	—	—	∆	∆	0	0
TSX	Transfer SP to X	(SK : SP) + $0002 ⇒ XK : IX	INH	274F	—	2	—	—	—	—	—	—	—	—
TSY	Transfer SP to Y	(SK : SP) + $0002 ⇒ YK : IY	INH	275F	—	2	—	—	—	—	—	—	—	—
TSZ	Transfer SP to Z	(SK : SP) + $0002 ⇒ ZK : IZ	INH	276F	—	2	—	—	—	—	—	—	—	—
TXKB	Transfer XK to B	(XK) ⇒ B[3:0] $0 ⇒ B[7:4]	INH	37AC	—	2	—	—	—	—	—	—	—	—
TXS	Transfer X to SP	(XK : IX) – $0002 ⇒ SK : SP	INH	374E	—	2	—	—	—	—	—	—	—	—
TXY	Transfer X to Y	(XK : IX) ⇒ YK : IY	INH	275C	—	2	—	—	—	—	—	—	—	—
TXZ	Transfer X to Z	(XK : IX) ⇒ ZK : IZ	INH	276C	—	2	—	—	—	—	—	—	—	—
TYKB	Transfer YK to B	(YK) ⇒ B[3:0] $0 ⇒ B[7:4]	INH	37AD	—	2	—	—	—	—	—	—	—	—
TYS	Transfer Y to SP	(YK : IY) – $0002 ⇒ SK : SP	INH	375E	—	2	—	—	—	—	—	—	—	—
TYX	Transfer Y to X	(YK : IY) ⇒ XK : IX	INH	274D	—	2	—	—	—	—	—	—	—	—
TYZ	Transfer Y to Z	(YK : IY) ⇒ ZK : IZ	INH	276D	—	2	—	—	—	—	—	—	—	—
TZKB	Transfer ZK to B	(ZK) ⇒ B[3:0] $0 ⇒ B[7:4]	INH	37AE	—	2	—	—	—	—	—	—	—	—
TZS	Transfer Z to SP	(ZK : IZ) – $0002 ⇒ SK : SP	INH	376E	—	2	—	—	—	—	—	—	—	—
TZX	Transfer Z to X	(ZK : IZ) ⇒ XK : IX	INH	274E	—	2	—	—	—	—	—	—	—	—
TZY	Transfer Z to Y	(ZK : IZ) ⇒ YK : IY	INH	275E	—	2	—	—	—	—	—	—	—	—
WAI	Wait for Interrupt	WAIT	INH	27F3	—	8	—	—	—	—	—	—	—	—
XGAB	Exchange A with B	(A) ⇔ (B)	INH	371A	—	2	—	—	—	—	—	—	—	—
XGDE	Exchange D with E	(D) ⇔ (E)	INH	277A	—	2	—	—	—	—	—	—	—	—
XGDX	Exchange D with IX	(D) ⇔ (IX)	INH	37CC	—	2	—	—	—	—	—	—	—	—
XGDY	Exchange D with IY	(D) ⇔ (IY)	INH	37DC	—	2	—	—	—	—	—	—	—	—
XGDZ	Exchange D with IZ	(D) ⇔ (IZ)	INH	37EC	—	2	—	—	—	—	—	—	—	—
XGEX	Exchange E with IX	(E) ⇔ (IX)	INH	374C	—	2	—	—	—	—	—	—	—	—
XGEY	Exchange E with IY	(E) ⇔ (IY)	INH	375C	—	2	—	—	—	—	—	—	—	—
XGEZ	Exchange E with IZ	(E) ⇔ (IZ)	INH	376C	—	2	—	—	—	—	—	—	—	—

NOTES:

1. CCR[15:4] change according to the results of the operation. The PK field is not affected.
2. Cycle times for conditional branches are shown in “taken, not taken” order.
3. CCR[15:0] change according to the copy of the CCR pulled from the stack.
4. PK field changes according to the state pulled from the stack. The rest of the CCR is not affected.

SECTION 7 INSTRUCTION PROCESS

This section explains how the CPU16 fetches and executes instructions. Topics include instruction format, pipelining, and changes in program flow. Other forms of the instruction process are covered in **SECTION 9 EXCEPTION PROCESSING** and **SECTION 11 DIGITAL SIGNAL PROCESSING**. See **SECTION 5 INSTRUCTION SET** and **SECTION 6 INSTRUCTION GLOSSARY** for detailed information concerning instructions.

7.1 Instruction Format

CPU16 instructions consist of an 8-bit opcode, which may be preceded by an 8-bit prebyte and/or followed by one or more operands.

Opcodes are mapped in four 256-instruction pages. Page 0 opcodes stand alone, but page 1, 2, and 3 opcodes are pointed to by a prebyte code on page 0. The prebytes are $17 (page 1), $27 (page 2), and $37 (page 3).

Operands can be four bits, eight bits, or sixteen bits in length. However, because the CPU16 fetches 16-bit instruction words from even byte boundaries, each instruction must contain an even number of bytes.

Operands are organized as bytes, words, or a combination of bytes and words. Four-bit operands are either zero-extended to eight bits, or packed two to a byte. The largest instructions are six bytes in length. Size, order, and function of operands are evaluated when an instruction is decoded.

A page 0 opcode and an 8-bit operand can be fetched simultaneously. Instructions that use 8-bit indexed, immediate, and relative addressing modes have this form — code written with these instructions is very compact.

Table 7-1 shows basic CPU16 instruction formats. **Table 7-2**, **Table 7-3**, **Table 7-4**, and **Table 7-5** show instructions in opcode order by page.

Table 7-1 Basic Instruction Formats

8-Bit Opcode with 8-Bit Operand

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode								Operand							

8-Bit Opcode with 4-Bit Index Extensions

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode								X Extension				Y Extension			

8-Bit Opcode, Argument(s)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode								Operand							
Operand(s)															
Operand(s)															

8-Bit Opcode with 8-Bit Prebyte, No Argument

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Prebyte								Opcode							

8-Bit Opcode with 8-Bit Prebyte, Argument(s)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Prebyte								Opcode							
Operand(s)															
Operand(s)															

8-Bit Opcode with 20-Bit Argument

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode								$0				Extension			
Operand															

7.2 Execution Model

This description builds up a conceptual model of the mechanism the CPU16 uses to fetch and execute instructions. The functional divisions in the model do not necessarily correspond to distinct architectural subunits of the microprocessor. **SECTION 10 DEVELOPMENT SUPPORT** expands the model to include the concept of deterministic opcode tracking.

As shown in **Figure 7-1**, there are three functional blocks involved in fetching, decoding, and executing instructions. These are the microsequencer, the instruction pipeline, and the execution unit. These elements function concurrently; at any given time, all three may be active.

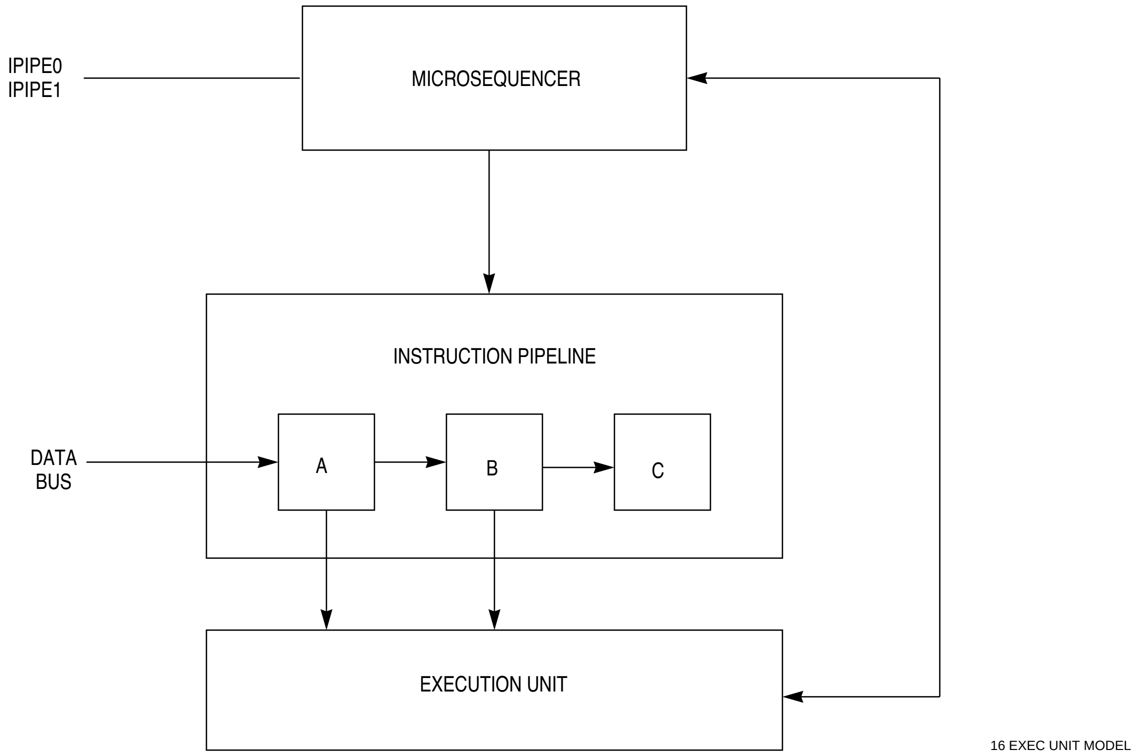


Figure 7-1 Instruction Execution Model

7.2.1 Microsequencer

The microsequencer controls the order in which instructions are fetched, advanced through the pipeline, and executed. It increments the program counter and generates multiplexed external tracking signals IPIPE0 and IPIPE1 from internal signals that control execution sequence.

7.2.2 Instruction Pipeline

The pipeline is a three stage FIFO that holds instructions while they are decoded and executed. Depending upon instruction size, as many as three instructions can be in the pipeline at one time (single-word instructions, one held in stage C, one being executed in stage B, and one latched in stage A).

7.2.3 Execution Unit

The execution unit evaluates opcodes, interfaces with the microsequencer to advance instructions through the pipeline, and performs instruction operations.

7.3 Execution Process

Fetched opcodes are latched into stage A, then advanced to stage B. Opcodes are evaluated in stage B. The execution unit can access operands in either stage A or stage B (stage B accesses are limited to 8-bit operands). When execution is complete, opcodes are moved from stage B to stage C, where they remain until the next instruction is complete.

A prefetch mechanism in the microsequencer reads instruction words from memory and increments the program counter. When instruction execution begins, the program counter points to an address six bytes after the address of the first word of the instruction being executed.

The number of machine cycles necessary to complete an execution sequence varies according to the complexity of the instruction. **SECTION 8 INSTRUCTION TIMING** gives detailed information concerning execution time calculation.

7.3.1 Detailed Process

The following description divides execution processing into discrete steps in order to describe it fully. Events in the steps are often concurrent. Refer to **SECTION 10 DEVELOPMENT SUPPORT** for information concerning signals used to track the sequence of execution. Relative PC values are given to aid following instructions through the pipeline.

A. PK : PC points to the first word address (FWA) of the instruction to be executed (PK : PC = FWA + $0000).
B. The microsequencer initiates a read from the memory address pointed to by PK : PC, signals pipeline stage A to latch the word (FWA + $0000) read from memory, then increments PK : PC (PK : PC = FWA + $0002).
C. The latched word contains either an 8-bit prebyte and an 8-bit opcode or an 8-bit opcode and an 8-bit operand. The microsequencer advances (FWA + $0000) to stage B, prefetches (FWA + $0002) from the data bus, and increments PK : PC (PK : PC = FWA + $0004).
D. Stage A now contains (FWA + $0002) and stage B contains (FWA + $0000). The execution unit determines what operations must be performed and the character of the operands needed to perform them. The microsequencer initiates a prefetch from FWA + $0004 and increments PK : PC (PK : PC = FWA + $0006). Subsequent execution depends upon instruction format.
    1. 8-bit opcode with 8-bit operand — The execution unit reads the operand and signals that execution has begun. The instruction executes, the content of stage B advances to stage C, the content of stage A advances to stage B, and (FWA + $0004) is latched into stage A.
    2. 16-bit opcode with no argument — The execution unit signals that execution has begun. The instruction executes, the content of stage B advances to stage C, the content of stage A advances to stage B, and (FWA + $0004) is latched into stage A.

3. 8-bit opcode with 20-bit argument **—** The execution unit reads the operand byte from stage B and the operand word from stage A, then signals that execution has begun. The instruction executes, the content of stage B advances to stage C, and (FWA + $0004) is latched into stage A.
4. 8-bit opcode with argument **—** The execution unit determines the number of operands needed, reads an operand byte from stage B and an operand word from stage A, then signals that execution has begun. The instruction executes, the content of stage B advances to stage C, and (FWA + $0004) is latched into stage A — this word can be either the third word of the current instruction or the first word of a new instruction.
5. 16-bit opcode with argument **—** The execution unit determines the number of operand words needed, reads the first operand word from stage A, then signals that execution has begun. The instruction executes, the content of stage B advances to stage C, and (FWA + $0004) is latched into stage A — this word can be either the third word of the current instruction or the first word of a new instruction.

E. At this point PK : PC = $0006, and the process repeats, but entry points differ for instructions of different lengths:
   1. One-word instructions — Stage B contains a new opcode for the execution unit to evaluate, and process repeats from D.
   2. Two-word instructions — Stage A contains a new opcode, and process repeats from C.
   3. Three-word instructions — Stages A and B contain operands from the instruction just completed, and process repeats from B.

Note

Due to the action of the prefetch mechanism, it is necessary to leave a two-word buffer at the end of program space. The last word of an instruction must be located at End of Memory – $0004.

The microsequencer always prefetches two words past the first word address of an instruction while that instruction is executing.

If an instruction is placed in either of the two highest available word addresses, these fetches may attempt access to addresses that do not exist — these attempts can cause bus errors.

7.3.2 Changes in Program Flow

When program flow changes, instructions are fetched from a new address. Before execution can begin at the new address, instructions and operands from the previous instruction stream must be removed from the pipeline. If a change in flow is temporary, a return address must be stored, so that execution of the original instruction stream can resume after the change in flow.

At the time an instruction that causes a change in program flow executes, PK : PC point to FWA + $0006. During execution of an instruction that causes a change of flow, PK : PC is loaded with the FWA of the new instruction stream. However, stages A and B still contain words from the old instruction stream. Process steps A through C must be performed prior to execution from the new instruction stream.

7.3.2.1 Jumps

Jump instructions cause an immediate, unconditional change in program flow. The CPU16 jump instruction uses 20-bit extended and indexed addressing modes. It consists of an 8-bit opcode with a 20-bit argument.

7.3.2.2 Branches

Branch instructions cause a change in program flow when a specific precondition is met. The CPU16 supports 8-bit relative displacement (short), and 16-bit relative displacement (long) branch instructions, as well as specialized bit condition branches that use indexed addressing modes.

Short branch instructions consist of an 8-bit opcode and an 8-bit operand contained in one word. Long branch instructions consist of an 8-bit prebyte and an 8-bit opcode in one word, followed by an operand word. Bit condition branches consist of an 8-bit opcode and an 8-bit operand in one word, followed by one or two operand words.

At the time a branch instruction is executed, PK : PC point to an address equal to the address of the instruction plus $0006. The range of displacement for each type of branch is relative to this value, not to the address of the instruction. In addition, because prefetches are automatically aligned to word boundaries, only even offsets are valid — an odd offset value is rounded down.

The numeric range of short branch and 8-bit indexed offset values is $80 (–128) to $7F (127). Due to word-alignment, maximum positive offset is $7E. At maximum positive offset, displacement from the branch instruction is 132. At maximum negative offset ($80), displacement is –122.

The numeric range of long branch and 16-bit indexed offset values is $8000 (–32768) to $7FFF (32767). Due to word-alignment, maximum positive offset is $7FFE. At maximum positive offset, displacement from the instruction is 32772. At maximum negative offset ($8000), displacement is –32762.

7.3.2.3 Subroutines

Subroutine instructions optimize the process of temporarily executing instructions from another instruction stream, usually to perform a particular task. The CPU16 can branch or jump to subroutines. A single instruction returns to the original instruction stream.

Subroutines can be called by short (BSR) or long (LBSR) branches, or by a jump (JSR). The RTS instruction returns control to the calling routine. BSR consists of an 8-bit opcode with an 8-bit operand. LBSR consists of an 8-bit prebyte and an 8-bit opcode in one word, followed by an operand word. JSR consists of an 8-bit opcode with a 20-bit argument. RTS consists of an 8-bit prebyte and an 8-bit opcode in one word.

When a subroutine instruction is executed, PK : PC contain the address of the calling instruction plus $0006. All three calling instructions stack return PK : PC values prior to processing instructions from the new instruction stream. In order for RTS to work with all three calling instructions, however, the value stacked by BSR must be adjusted.

LBSR and JSR are two-word instructions. In order for program execution to resume with the instruction immediately following them, RTS must subtract $0002 from the stacked PK : PC value. BSR is a one-word instruction — it subtracts $0002 from PK : PC prior to stacking so that execution will resume correctly after RTS.

7.3.2.4 Interrupts

An interrupt routine usually performs a critical task, then returns control to the interrupted instruction stream. Interrupts are a type of exception, and are thus subject to special rules regarding execution process. **SECTION 9 EXCEPTION PROCESSING** covers interrupt exception processing in detail. This discussion is limited to the effects of SWI (software interrupt) and RTI (return from interrupt) instructions.

Both SWI and RTI consist of an 8-bit prebyte and an 8-bit opcode in one word. SWI initiates synchronous exception processing. RTI causes execution to resume with the instruction following the last instruction that completed execution prior to interrupt.

Asynchronous interrupts are serviced at instruction boundaries. PK : PC + $0006 for the following instruction is stacked, and exception processing begins. In order to resume execution with the correct instruction, RTI subtracts $0006 from the stacked value.

Interrupt exception processing is included in the SWI instruction definition. The PK : PC value at the time of execution is the first word address of SWI plus $0006. If this value were stacked, RTI would cause SWI to execute again. In order to resume execution with the instruction following SWI, $0002 is added to the PK : PC value prior to stacking.

Table 7-2 Page 0 Opcodes

Opcode	Mnemonic	Mode	Opcode	Mnemonic	Mode
00	COM	IND8, X	20	COM	IND8, Z
01	DEC	IND8, X	21	DEC	IND8, Z
02	NEG	IND8, X	22	NEG	IND8, Z
03	INC	IND8, X	23	INC	IND8, Z
04	ASL	IND8, X	24	ASL	IND8, Z
05	CLR	IND8, X	25	CLR	IND8, Z
06	TST	IND8, X	26	TST	IND8, Z
07	—	—	27	PREBYTE	PAGE 2
08	BCLR	IND16, X	28	BCLR	IND16, Z
09	BSET	IND16, X	29	BSET	IND16, Z
0A	BRCLR	IND16, X	2A	BRCLR	IND16, Z
0B	BRSET	IND16, X	2B	BRSET	IND16, Z
0C	ROL	IND8, X	2C	ROL	IND8, Z
0D	ASR	IND8, X	2D	ASR	IND8, Z
0E	ROR	IND8, X	2E	ROR	IND8, Z
0F	LSR	IND8, X	2F	LSR	IND8, Z
10	COM	IND8, Y	30	MOVB	IXP to EXT
11	DEC	IND8, Y	31	MOVW	IXP to EXT
12	NEG	IND8, Y	32	MOVB	EXT to IXP
13	INC	IND8, Y	33	MOVW	EXT to IXP
14	ASL	IND8, Y	34	PSHM	INH
15	CLR	IND8, Y	35	PULM	INH
16	TST	IND8, Y	36	BSR	REL8
17	PREBYTE	PAGE 1	37	PREBYTE	PAGE 3
18	BCLR	IND16, Y	38	BCLR	EXT
19	BSET	IND16, Y	39	BSET	EXT
1A	BRCLR	IND16, Y	3A	BRCLR	EXT
1B	BRSET	IND16, Y	3B	BRSET	EXT
1C	ROL	IND8, Y	3C	AIX	IMM8
1D	ASR	IND8, Y	3D	AIY	IMM8
1E	ROR	IND8, Y	3E	AIZ	IMM8
1F	LSR	IND8, Y	3F	AIS	IMM8
40	SUBA	IND8, X	60	SUBA	IND8, Z
41	ADDA	IND8, X	61	ADDA	IND8, Z
42	SBCA	IND8, X	62	SBCA	IND8, Z
43	ADCA	IND8, X	63	ADCA	IND8, Z
44	EORA	IND8, X	64	EORA	IND8, Z
45	LDAA	IND8, X	65	LDAA	IND8, Z
46	ANDA	IND8, X	66	ANDA	IND8, Z
47	ORAA	IND8, X	67	ORAA	IND8, Z
48	CMPA	IND8, X	68	CMPA	IND8, Z
49	BITA	IND8, X	69	BITA	IND8, Z
4A	STAA	IND8, X	6A	STAA	IND8, Z
4B	JMP	IND20, X	6B	JMP	IND20, Z
4C	CPX	IND8, X	6C	CPX	IND8, Z
4D	CPY	IND8, X	6D	CPY	IND8, Z
4E	CPZ	IND8, X	6E	CPZ	IND8, Z
4F	CPS	IND8, X	6F	CPS	IND8, Z

Table 7-2 Page 0 Opcodes (Continued)

Opcode	Mnemonic	Mode	Opcode	Mnemonic	Mode
50	SUBA	IND8, Y	70	SUBA	IMM8
51	ADDA	IND8, Y	71	ADDA	IMM8
52	SBCA	IND8, Y	72	SBCA	IMM8
53	ADCA	IND8, Y	73	ADCA	IMM8
54	EORA	IND8, Y	74	EORA	IMM8
55	LDAA	IND8, Y	75	LDAA	IMM8
56	ANDA	IND8, Y	76	ANDA	IMM8
57	ORAA	IND8, Y	77	ORAA	IMM8
58	CMPA	IND8, Y	78	CMPA	IMM8
59	BITA	IND8, Y	79	BITA	IMM8
5A	STAA	IND8, Y	7A	JMP	EXT
5B	JMP	IND20, Y	7B	MAC	IMM8
5C	CPX	IND8, Y	7C	ADDE	IMM8
5D	CPY	IND8, Y	7D	—	—
5E	CPZ	IND8, Y	7E	—	—
5F	CPS	IND8, Y	7F	—	—
80	SUBD	IND8, X	A0	SUBD	IND8, Z
81	ADDD	IND8, X	A1	ADDD	IND8, Z
82	SBCD	IND8, X	A2	SBCD	IND8, Z
83	ADCD	IND8, X	A3	ADCD	IND8, Z
84	EORD	IND8, X	A4	EORD	IND8, Z
85	LDD	IND8, X	A5	LDD	IND8, Z
86	ANDD	IND8, X	A6	ANDD	IND8, Z
87	ORD	IND8, X	A7	ORD	IND8, Z
88	CPD	IND8, X	A8	CPD	IND8, Z
89	JSR	IND20, X	A9	JSR	IND20, Z
8A	STD	IND8, X	AA	STD	IND8, Z
8B	BRSET	IND8, X	AB	BRSET	IND8, Z
8C	STX	IND8, X	AC	STX	IND8, Z
8D	STY	IND8, X	AD	STY	IND8, Z
8E	STZ	IND8, X	AE	STZ	IND8, Z
8F	STS	IND8, X	AF	STS	IND8, Z
90	SUBD	IND8, Y	B0	BRA	REL8
91	ADDD	IND8, Y	B1	BRN	REL8
92	SBCD	IND8, Y	B2	BHI	REL8
93	ADCD	IND8, Y	B3	BLS	REL8
94	EORD	IND8, Y	B4	BCC	REL8
95	LDD	IND8, Y	B5	BCS	REL8
96	ANDD	IND8, Y	B6	BNE	REL8
97	ORD	IND8, Y	B7	BEQ	REL8
98	CPD	IND8, Y	B8	BVC	REL8
99	JSR	IND20, Y	B9	BVS	REL8
9A	STD	IND8, Y	BA	BPL	REL8
9B	BRSET	IND8, Y	BB	BMI	REL8
9C	STX	IND8, Y	BC	BGE	REL8
9D	STY	IND8, Y	BD	BLT	REL8
9E	STZ	IND8, Y	BE	BGT	REL8
9F	STS	IND8, Y	BF	BLE	REL8

Table 7-2 Page 0 Opcodes (Continued)

Opcode	Mnemonic	Mode	Opcode	Mnemonic	Mode
C0	SUBB	IND8, X	E0	SUBB	IND8, Z
C1	ADDB	IND8, X	E1	ADDB	IND8, Z
C2	SBCB	IND8, X	E2	SBCB	IND8, Z
C3	ADCB	IND8, X	E3	ADCB	IND8, Z
C4	EORB	IND8, X	E4	EORB	IND8, Z
C5	LDAB	IND8, X	E5	LDAB	IND8, Z
C6	ANDB	IND8, X	E6	ANDB	IND8, Z
C7	ORAB	IND8, X	E7	ORAB	IND8, Z
C8	CMPB	IND8, X	E8	CMPB	IND8, Z
C9	BITB	IND8, X	E9	BITB	IND8, Z
CA	STAB	IND8, X	EA	STAB	IND8, Z
CB	BRCLR	IND8, X	EB	BRCLR	IND8, Z
CC	LDX	IND8, X	EC	LDX	IND8, Z
CD	LDY	IND8, X	ED	LDY	IND8, Z
CE	LDZ	IND8, X	EE	LDZ	IND8, Z
CF	LDS	IND8, X	EF	LDS	IND8, Z
D0	SUBB	IND8, Y	F0	SUBB	IMM8
D1	ADDB	IND8, Y	F1	ADDB	IMM8
D2	SBCB	IND8, Y	F2	SBCB	IMM8
D3	ADCB	IND8, Y	F3	ADCB	IMM8
D4	EORB	IND8, Y	F4	EORB	IMM8
D5	LDAB	IND8, Y	F5	LDAB	IMM8
D6	ANDB	IND8, Y	F6	ANDB	IMM8
D7	ORAB	IND8, Y	F7	ORAB	IMM8
D8	CMPB	IND8, Y	F8	CMPB	IMM8
D9	BITB	IND8, Y	F9	BITB	IMM8
DA	STAB	IND8, Y	FA	JSR	EXT
DB	BRCLR	IND8, Y	FB	RMAC	IMM8
DC	LDX	IND8, Y	FC	ADDD	IMM8
DD	LDY	IND8, Y	FD	—	—
DE	LDZ	IND8, Y	FE	—	—
DF	LDS	IND8, Y	FF	—	—

Table 7-3 Page 1 Opcodes

Opcode	Mnemonic	Mode	Opcode	Mnemonic	Mode
1700	COM	IND16, X	1720	COM	IND16, Z
1701	DEC	IND16, X	1721	DEC	IND16, Z
1702	NEG	IND16, X	1722	NEG	IND16, Z
1703	INC	IND16, X	1723	INC	IND16, Z
1704	ASL	IND16, X	1724	ASL	IND16, Z
1705	CLR	IND16, X	1725	CLR	IND16, Z
1706	TST	IND16, X	1726	TST	IND16, Z
1707	—	—	1727	—	—
1708	BCLR	IND8, X	1728	BCLR	IND8, Z
1709	BSET	IND8, X	1729	BSET	IND8, Z
170A	—	—	172A	—	—
170B	—	—	172B	—	—
170C	ROL	IND16, X	172C	ROL	IND16, Z
170D	ASR	IND16, X	172D	ASR	IND16, Z
170E	ROR	IND16, X	172E	ROR	IND16, Z
170F	LSR	IND16, X	172F	LSR	IND16, Z
1710	COM	IND16, Y	1730	COM	EXT
1711	DEC	IND16, Y	1731	DEC	EXT
1712	NEG	IND16, Y	1732	NEG	EXT
1713	INC	IND16, Y	1733	INC	EXT
1714	ASL	IND16, Y	1734	ASL	EXT
1715	CLR	IND16, Y	1735	CLR	EXT
1716	TST	IND16, Y	1736	TST	EXT
1717	—	—	1737	—	—
1718	BCLR	IND8, Y	1738	—	—
1719	BSET	IND8, Y	1739	—	—
171A	—	—	173A	—	—
171B	—	—	173B	—	—
171C	ROL	IND16, Y	173C	ROL	EXT
171D	ASR	IND16, Y	173D	ASR	EXT
171E	ROR	IND16, Y	173E	ROR	EXT
171F	LSR	IND16, Y	173F	LSR	EXT
1740	SUBA	IND16, X	1760	SUBA	IND16, Z
1741	ADDA	IND16, X	1761	ADDA	IND16, Z
1742	SBCA	IND16, X	1762	SBCA	IND16, Z
1743	ADCA	IND16, X	1763	ADCA	IND16, Z
1744	EORA	IND16, X	1764	EORA	IND16, Z
1745	LDAA	IND16, X	1765	LDAA	IND16, Z
1746	ANDA	IND16, X	1766	ANDA	IND16, Z
1747	ORAA	IND16, X	1767	ORAA	IND16, Z
1748	CMPA	IND16, X	1768	CMPA	IND16, Z
1749	BITA	IND16, X	1769	BITA	IND16, Z
174A	STAA	IND16, X	176A	STAA	IND16, Z
174B	—	—	176B	—	—
174C	CPX	IND16, X	176C	CPX	IND16, Z
174D	CPY	IND16, X	176D	CPY	IND16, Z
174E	CPZ	IND16, X	176E	CPZ	IND16, Z
174F	CPS	IND16, X	176F	CPS	IND16, Z

Table 7-3 Page 1 Opcodes (Continued)

Opcode	Mnemonic	Mode	Opcode	Mnemonic	Mode
1750	SUBA	IND16, Y	1770	SUBA	EXT
1751	ADDA	IND16, Y	1771	ADDA	EXT
1752	SBCA	IND16, Y	1772	SBCA	EXT
1753	ADCA	IND16, Y	1773	ADCA	EXT
1754	EORA	IND16, Y	1774	EORA	EXT
1755	LDAA	IND16, Y	1775	LDAA	EXT
1756	ANDA	IND16, Y	1776	ANDA	EXT
1757	ORAA	IND16, Y	1777	ORAA	EXT
1758	CMPA	IND16, Y	1778	CMPA	EXT
1759	BITA	IND16, Y	1779	BITA	EXT
175A	STAA	IND16, Y	177A	STAA	EXT
175B	—	—	177B	—	—
175C	CPX	IND16, Y	177C	CPX	EXT
175D	CPY	IND16, Y	177D	CPY	EXT
175E	CPZ	IND16, Y	177E	CPZ	EXT
175F	CPS	IND16, Y	177F	CPS	EXT
1780	—	—	17A0	—	—
1781	—	—	17A1	—	—
1782	—	—	17A2	—	—
1783	—	—	17A3	—	—
1784	—	—	17A4	—	—
1785	—	—	17A5	—	—
1786	—	—	17A6	—	—
1787	—	—	17A7	—	—
1788	—	—	17A8	—	—
1789	—	—	17A9	—	—
178A	—	—	17AA	—	—
178B	—	—	17AB	—	—
178C	STX	IND16, X	17AC	STX	IND16, Z
178D	STY	IND16, X	17AD	STY	IND16, Z
178E	STZ	IND16, X	17AE	STZ	IND16, Z
178F	STS	IND16, X	17AF	STS	IND16, Z
1790	—	—	17B0	—	—
1791	—	—	17B1	—	—
1792	—	—	17B2	—	—
1793	—	—	17B3	—	—
1794	—	—	17B4	—	—
1795	—	—	17B5	—	—
1796	—	—	17B6	—	—
1797	—	—	17B7	—	—
1798	—	—	17B8	—	—
1799	—	—	17B9	—	—
179A	—	—	17BA	—	—
179B	—	—	17BB	—	—
179C	STX	IND16, Y	17BC	STX	EXT
179D	STY	IND16, Y	17BD	STY	EXT
179E	STZ	IND16, Y	17BE	STZ	EXT
179F	STS	IND16, Y	17BF	STS	EXT

Table 7-3 Page 1 Opcodes (Continued)

Opcode	Mnemonic	Mode	Opcode	Mnemonic	Mode
17C0	SUBB	IND16, X	17E0	SUBB	IND16, Z
17C1	ADDB	IND16, X	17E1	ADDB	IND16, Z
17C2	SBCB	IND16, X	17E2	SBCB	IND16, Z
17C3	ADCB	IND16, X	17E3	ADCB	IND16, Z
17C4	EORB	IND16, X	17E4	EORB	IND16, Z
17C5	LDAB	IND16, X	17E5	LDAB	IND16, Z
17C6	ANDB	IND16, X	17E6	ANDB	IND16, Z
17C7	ORAB	IND16, X	17E7	ORAB	IND16, Z
17C8	CMPB	IND16, X	17E8	CMPB	IND16, Z
17C9	BITB	IND16, X	17E9	BITB	IND16, Z
17CA	STAB	IND16, X	17EA	STAB	IND16, Z
17CB	—	—	17EB	—	—
17CC	LDX	IND16, X	17EC	LDX	IND16, Z
17CD	LDY	IND16, X	17ED	LDY	IND16, Z
17CE	LDZ	IND16, X	17EE	LDZ	IND16, Z
17CF	LDS	IND16, X	17EF	LDS	IND16, Z
17D0	SUBB	IND16, Y	17F0	SUBB	EXT
17D1	ADDB	IND16, Y	17F1	ADDB	EXT
17D2	SBCB	IND16, Y	17F2	SBCB	EXT
17D3	ADCB	IND16, Y	17F3	ADCB	EXT
17D4	EORB	IND16, Y	17F4	EORB	EXT
17D5	LDAB	IND16, Y	17F5	LDAB	EXT
17D6	ANDB	IND16, Y	17F6	ANDB	EXT
17D7	ORAB	IND16, Y	17F7	ORAB	EXT
17D8	CMPB	IND16, Y	17F8	CMPB	EXT
17D9	BITB	IND16, Y	17F9	BITB	EXT
17DA	STAB	IND16, Y	17FA	STAB	EXT
17DB	—	—	17FB	—	—
17DC	LDX	IND16, Y	17FC	LDX	EXT
17DD	LDY	IND16, Y	17FD	LDY	EXT
17DE	LDZ	IND16, Y	17FE	LDZ	EXT
17DF	LDS	IND16, Y	17FF	LDS	EXT

Table 7-4 Page 2 Opcodes

Opcode	Mnemonic	Mode	Opcode	Mnemonic	Mode
2700	COMW	IND16, X	2720	COMW	IND16, Z
2701	DECW	IND16, X	2721	DECW	IND16, Z
2702	NEGW	IND16, X	2722	NEGW	IND16, Z
2703	INCW	IND16, X	2723	INCW	IND16, Z
2704	ASLW	IND16, X	2724	ASLW	IND16, Z
2705	CLRW	IND16, X	2725	CLRW	IND16, Z
2706	TSTW	IND16, X	2726	TSTW	IND16, Z
2707	—	—	2727	—	—
2708	BCLRW	IND16, X	2728	BCLRW	IND16, Z
2709	BSETW	IND16, X	2729	BSETW	IND16, Z
270A	—	—	272A	—	—
270B	—	—	272B	—	—
270C	ROLW	IND16, X	272C	ROLW	IND16, Z
270D	ASRW	IND16, X	272D	ASRW	IND16, Z
270E	RORW	IND16, X	272E	RORW	IND16, Z
270F	LSRW	IND16, X	272F	LSRW	IND16, Z
2710	COMW	IND16, Y	2730	COMW	EXT
2711	DECW	IND16, Y	2731	DECW	EXT
2712	NEGW	IND16, Y	2732	NEGW	EXT
2713	INCW	IND16, Y	2733	INCW	EXT
2714	ASLW	IND16, Y	2734	ASLW	EXT
2715	CLRW	IND16, Y	2735	CLRW	EXT
2716	TSTW	IND16, Y	2736	TSTW	EXT
2717	—	—	2737	—	—
2718	BCLRW	IND16, Y	2738	BCLRW	EXT
2719	BSETW	IND16, Y	2739	BSETW	EXT
271A	—	—	273A	—	—
271B	—	—	273B	—	—
271C	ROLW	IND16, Y	273C	ROLW	EXT
271D	ASRW	IND16, Y	273D	ASRW	EXT
271E	RORW	IND16, Y	273E	RORW	EXT
271F	LSRW	IND16, Y	273F	LSRW	EXT
2740	SUBA	E, X	2760	SUBA	E, Z
2741	ADDA	E, X	2761	ADDA	E, Z
2742	SBCA	E, X	2762	SBCA	E, Z
2743	ADCA	E, X	2763	ADCA	E, Z
2744	EORA	E, X	2764	EORA	E, Z
2745	LDAA	E, X	2765	LDAA	E, Z
2746	ANDA	E, X	2766	ANDA	E, Z
2747	ORAA	E, X	2767	ORAA	E, Z
2748	CMPA	E, X	2768	CMPA	E, Z
2749	BITA	E, X	2769	BITA	E, Z
274A	STAA	E, X	276A	STAA	E, Z
274B	—	—	276B	—	—
274C	NOP	INH	276C	TXZ	INH
274D	TYX	INH	276D	TYZ	INH
274E	TZX	INH	276E	—	—
274F	TSX	INH	276F	TSZ	INH

Table 7-4 Page 2 Opcodes (Continued)

Opcode	Mnemonic	Mode	Opcode	Mnemonic	Mode
2750	SUBA	E, Y	2770	COME	INH
2751	ADDA	E, Y	2771	LDED	EXT
2752	SBCA	E, Y	2772	NEGE	INH
2753	ADCA	E, Y	2773	STED	EXT
2754	EORA	E, Y	2774	ASLE	INH
2755	LDAA	E, Y	2775	CLRE	INH
2756	ANDA	E, Y	2776	TSTE	INH
2757	ORAA	E, Y	2777	RTI	INH
2758	CMPA	E, Y	2778	ADE	INH
2759	BITA	E, Y	2779	SDE	INH
275A	STAA	E, Y	277A	XGDE	INH
275B	—	—	277B	TDE	INH
275C	TXY	INH	277C	ROLE	INH
275D	—	—	277D	ASRE	INH
275E	TZY	INH	277E	RORE	INH
275F	TSY	INH	277F	LSRE	INH
2780	SUBD	E, X	27A0	SUBD	E, Z
2781	ADDD	E, X	27A1	ADDD	E, Z
2782	SBCD	E, X	27A2	SBCD	E, Z
2783	ADCD	E, X	27A3	ADCD	E, Z
2784	EORD	E, X	27A4	EORD	E, Z
2785	LDD	E, X	27A5	LDD	E, Z
2786	ANDD	E, X	27A6	ANDD	E, Z
2787	ORD	E, X	27A7	ORD	E, Z
2788	CPD	E, X	27A8	CPD	E, Z
2789	—	—	27A9	—	—
278A	STD	E, X	27AA	STD	E, Z
278B	—	—	27AB	—	—
278C	—	—	27AC	—	—
278D	—	—	27AD	—	—
278E	—	—	27AE	—	—
278F	—	—	27AF	—	—
2790	SUBD	E, Y	27B0	LDHI	EXT
2791	ADDD	E, Y	27B1	TEDM	INH
2792	SBCD	E, Y	27B2	TEM	INH
2793	ADCD	E, Y	27B3	TMXED	INH
2794	EORD	E, Y	27B4	TMER	INH
2795	LDD	E, Y	27B5	TMET	INH
2796	ANDD	E, Y	27B6	ASLM	INH
2797	ORD	E, Y	27B7	CLRM	INH
2798	CPD	E, Y	27B8	PSHMAC	INH
2799	—	—	27B9	PULMAC	INH
279A	STD	E, Y	27BA	ASRM	INH
279B	—	—	27BB	TEKB	INH
279C	—	—	27BC	—	—
279D	—	—	27BD	—	—
279E	—	—	27BE	—	—
279F	—	—	27BF	—	—

Table 7-4 Page 2 Opcodes (Continued)

Opcode	Mnemonic	Mode	Opcode	Mnemonic	Mode
27C0	SUBB	E, X	27E0	SUBB	E, Z
27C1	ADDB	E, X	27E1	ADDB	E, Z
27C2	SBCB	E, X	27E2	SBCB	E, Z
27C3	ADCB	E, X	27E3	ADCB	E, Z
27C4	EORB	E, X	27E4	EORB	E, Z
27C5	LDAB	E, X	27E5	LDAB	E, Z
27C6	ANDB	E, X	27E6	ANDB	E, Z
27C7	ORAB	E, X	27E7	ORAB	E, Z
27C8	CMPB	E, X	27E8	CMPB	E, Z
27C9	BITB	E, X	27E9	BITB	E, Z
27CA	STAB	E, X	27EA	STAB	E, Z
27CB	—	—	27EB	—	—
27CC	—	—	27EC	—	—
27CD	—	—	27ED	—	—
27CE	—	—	27EE	—	—
27CF	—	—	27EF	—	—
27D0	SUBB	E, Y	27F0	COMD	INH
27D1	ADDB	E, Y	27F1	LPSTOP	INH
27D2	SBCB	E, Y	27F2	NEGD	INH
27D3	ADCB	E, Y	27F3	WAI	INH
27D4	EORB	E, Y	27F4	ASLD	INH
27D5	LDAB	E, Y	27F5	CLRD	INH
27D6	ANDB	E, Y	27F6	TSTD	INH
27D7	ORAB	E, Y	27F7	RTS	INH
27D8	CMPB	E, Y	27F8	SXT	INH
27D9	BITB	E, Y	27F9	LBSR	REL16
27DA	STAB	E, Y	27FA	TBEK	INH
27DB	—	—	27FB	TED	INH
27DC	—	—	27FC	ROLD	INH
27DD	—	—	27FD	ASRD	INH
27DE	—	—	27FE	RORD	INH
27DF	—	—	27FF	LSRD	INH

Table 7-5 Page 3 Opcodes

Opcode	Mnemonic	Mode	Opcode	Mnemonic	Mode
3700	COMA	INH	3720	SWI	INH
3701	DECA	INH	3721	DAA	INH
3702	NEGA	INH	3722	ACE	INH
3703	INCA	INH	3723	ACED	INH
3704	ASLA	INH	3724	MUL	INH
3705	CLRA	INH	3725	EMUL	INH
3706	TSTA	INH	3726	EMULS	INH
3707	TBA	INH	3727	FMULS	INH
3708	PSHA	INH	3728	EDIV	INH
3709	PULA	INH	3729	EDIVS	INH
370A	SBA	INH	372A	IDIV	INH
370B	ABA	INH	372B	FDIV	INH
370C	ROLA	INH	372C	TPD	INH
370D	ASRA	INH	372D	TDP	INH
370E	RORA	INH	372E	—	—
370F	LSRA	INH	372F	TDMSK	INH
3710	COMB	INH	3730	SUBE	IMM16
3711	DECB	INH	3731	ADDE	IMM16
3712	NEGB	INH	3732	SBCE	IMM16
3713	INCB	INH	3733	ADCE	IMM16
3714	ASLB	INH	3734	EORE	IMM16
3715	CLRB	INH	3735	LDE	IMM16
3716	TSTB	INH	3736	ANDE	IMM16
3717	TAB	INH	3737	ORE	IMM16
3718	PSHB	INH	3738	CPE	IMM16
3719	PULB	INH	3739	—	—
371A	XGAB	INH	373A	ANDP	IMM16
371B	CBA	INH	373B	ORP	IMM16
371C	ROLB	INH	373C	AIX	IMM16
371D	ASRB	INH	373D	AIY	IMM16
371E	RORB	INH	373E	AIZ	IMM16
371F	LSRB	INH	373F	AIS	IMM16
3740	SUBE	IND16, X	3760	SUBE	IND16, Z
3741	ADDE	IND16, X	3761	ADDE	IND16, Z
3742	SBCE	IND16, X	3762	SBCE	IND16, Z
3743	ADCE	IND16, X	3763	ADCE	IND16, Z
3744	EORE	IND16, X	3764	EORE	IND16, Z
3745	LDE	IND16, X	3765	LDE	IND16, Z
3746	ANDE	IND16, X	3766	ANDE	IND16, Z
3747	ORE	IND16, X	3767	ORE	IND16, Z
3748	CPE	IND16, X	3768	CPE	IND16, Z
3749	—	—	3769	—	—
374B	—	—	376A	STE	IND16, Z
374A	STE	IND16, X	376B	—	—
374C	XGEX	INH	376C	XGEZ	INH
374D	AEX	INH	376D	AEZ	INH
374E	TXS	INH	376E	TZS	INH
374F	ABX	INH	376F	ABZ	INH

Table 7-5 Page 3 Opcodes (Continued)

Opcode	Mnemonic	Mode	Opcode	Mnemonic	Mode
3750	SUBE	IND16, Y	3770	SUBE	EXT
3751	ADDE	IND16, Y	3771	ADDE	EXT
3752	SBCE	IND16, Y	3772	SBCE	EXT
3753	ADCE	IND16, Y	3773	ADCE	EXT
3754	EORE	IND16, Y	3774	EORE	EXT
3755	LDE	IND16, Y	3775	LDE	EXT
3756	ANDE	IND16, Y	3776	ANDE	EXT
3757	ORE	IND16, Y	3777	ORE	EXT
3758	CPE	IND16, Y	3778	CPE	EXT
3759	—	—	3779	—	—
375A	STE	IND16, Y	377A	STE	EXT
375B	—	—	377B	—	—
375C	XGEY	INH	377C	CPX	IMM16
375D	AEY	INH	377D	CPY	IMM16
375E	TYS	INH	377E	CPZ	IMM16
375F	ABY	INH	377F	CPS	IMM16
3780	LBRA	REL16	37A0	—	—
3781	LBRN	REL16	37A1	—	—
3782	LBHI	REL16	37A2	—	—
3783	LBLS	REL16	37A3	—	—
3784	LBCC	REL16	37A4	—	—
3785	LBCS	REL16	37A5	—	—
3786	LBNE	REL16	37A6	BGND	INH
3787	LBEQ	REL16	37A7	—	—
3788	LBVC	REL16	37A8	—	—
3789	LBVS	REL16	37A9	—	—
378A	LBPL	REL16	37AA	—	—
378B	LBMI	REL16	37AB	—	—
378C	LBGE	REL16	37AC	TXKB	INH
378D	LBLT	REL16	37AD	TYKB	INH
378E	LBGT	REL16	37AE	TZKB	INH
378F	LBLE	REL16	37AF	TSKB	INH
3790	LBMV	REL16	37B0	SUBD	IMM16
3791	LBEV	REL16	37B1	ADDD	IMM16
3792	—	—	37B2	SBCD	IMM16
3793	—	—	37B3	ADCD	IMM16
3794	—	—	37B4	EORD	IMM16
3795	—	—	37B5	LDD	IMM16
3796	—	—	37B6	ANDD	IMM16
3797	—	—	37B7	ORD	IMM16
3798	—	—	37B8	CPD	IMM16
3799	—	—	37B9	—	—
379A	—	—	37BA	—	—
379B	—	—	37BA	—	—
379C	TBXK	INH	37BC	LDX	IMM16
379D	TBYK	INH	37BD	LDY	IMM16
379E	TBZK	INH	37BE	LDZ	IMM16
379F	TBSK	INH	37BF	LDS	IMM16

Table 7-5 Page 3 Opcodes (Continued)

Opcode	Mnemonic	Mode	Opcode	Mnemonic	Mode
37C0	SUBD	IND16, X	37E0	SUBD	IND16, Z
37C1	ADDD	IND16, X	37E1	ADDD	IND16, Z
37C2	SBCD	IND16, X	37E2	SBCD	IND16, Z
37C3	ADCD	IND16, X	37E3	ADCD	IND16, Z
37C4	EORD	IND16, X	37E4	EORD	IND16, Z
37C5	LDD	IND16, X	37E5	LDD	IND16, Z
37C6	ANDD	IND16, X	37E6	ANDD	IND16, Z
37C7	ORD	IND16, X	37E7	ORD	IND16, Z
37C8	CPD	IND16, X	37E8	CPD	IND16, Z
37C9	—	—	37E9	—	—
37CA	STD	IND16, X	37EA	STD	IND16, Z
37CB	—	—	37EB	—	—
37CC	XGDX	INH	37EC	XGDZ	INH
37CD	ADX	INH	37ED	ADZ	INH
37CE	—	—	37EE	—	—
37CF	—	—	37EF	—	—
37D0	SUBD	IND16, Y	37F0	SUBD	EXT
37D1	ADDD	IND16, Y	37F1	ADDD	EXT
37D2	SBCD	IND16, Y	37F2	SBCD	EXT
37D3	ADCD	IND16, Y	37F3	ADCD	EXT
37D4	EORD	IND16, Y	37F4	EORD	EXT
37D5	LDD	IND16, Y	37F5	LDD	EXT
37D6	ANDD	IND16, Y	37F6	ANDD	EXT
37D7	ORD	IND16, Y	37F7	ORD	EXT
37D8	CPD	IND16, Y	37F8	CPD	EXT
37D9	—	—	37F9	—	—
37DA	STD	IND16, Y	37FA	STD	EXT
37DB	—	—	37FB	—	—
37DC	XGDY	INH	37FC	TPA	INH
37DD	ADY	INH	37FD	TAP	INH
37DE	—	—	37FE	MOVB	EXT to EXT
37DF	—	—	37FF	MOVW	EXT to EXT

SECTION 8 INSTRUCTION TIMING

This section gives detailed information concerning calculating the amount of time required to execute instructions.

8.1 Execution Time Components

CPU16 instruction execution time has three components:

Bus cycles required to prefetch the next instruction.
Bus cycles required for operand accesses.
Clock cycles required for internal operations.

Each bus cycle requires a minimum of two system clock cycles. If the time required to access an external device exceeds two system clock cycles, bus cycles must be longer. However, all bus cycles must be made up of an integer number of clock cycles. CPU16 internal operations always require an integer multiple of two system clock cycles.

NOTE

To avoid confusion between bus cycles and system clock cycles, this discussion subsequently refers to the time required by system clock cycles, or clock periods, rather than to the clock cycles themselves.

Dynamic bus sizing affects bus cycle time. The CPU16 is a component of a modular microcontroller. Modules in the system communicate via a standardized intermodule bus and access external devices via an external bus interface. The microcontroller system integration module manages all accesses in order to make more efficient use of common resources. See **SECTION 3 SYSTEM RESOURCES** for more information.

The CPU16 does not execute more than one instruction at a time. The total time required to execute a particular instruction stream can be calculated by summing the individual execution times of each instruction in the stream.

Total execution time is calculated using the expression:

$$(CL_T) = (CL_P) + (CL_O) + (CL_I)$$

Where:

(CL_T) = Total clock periods per instruction
(CL_I) = Clock periods used for internal operation
(CL_P) = Clock periods used for program access
(CL_O) = Clock periods used for operand access

CL_T is the value provided in the instruction glossary pages.

8.2 Program and Operand Access Time

The number of bus cycles required by a prefetch or an operand access generally depends upon three factors:

Data bus width (8- or 16-bit). Access size (byte, word, or long-word). Access alignment (aligned or misaligned with even byte boundaries).

Prefetches are always word-sized, and are always aligned with even byte boundaries. Operand accesses vary in size and alignment. **Table 8-1** shows the number of bus cycles required by accesses of various sizes and alignments.

Table 8-1 Access Bus Cycles

Access Size	8-Bit Data Bus	16-Bit Data Bus Aligned	16-Bit Data Bus Misaligned
Byte	1	1	—
Word	2	1	2
Long-word	4	2	4

8.2.1 Program Accesses

For all instructions except those that cause a change in program flow, there is one prefetch access per instruction word. These accesses keep the instruction pipeline full. Once the number of prefetches is determined, the number of bus cycles can be found in **Table 8-1**.

Instructions that cause changes in program flow also have various forms of operand access. See **8.2.2.3 Change-of-Flow Instructions** for complete information on prefetch access and operand access.

8.2.2 Operand Accesses

The number of operand accesses per instruction is not fixed. Most instructions follow a regular pattern, but there are several variant types. Immediate operands are considered to be part of the instruction — immediate operand access time is considered to be a prefetch access.

8.2.2.1 Regular Instructions

Regular instructions require one operand access per operand. Determine the number of byte and/or word operands, then use **Table 8-1** to determine the number of cycles.

8.2.2.2 Read-Modify-Write Instructions

Read-modify-write instructions, which include the byte and word forms of ASL, ASR, BCLR, BSET, COM, DEC, LSR, NEG, ROL, and ROR, require two accesses per memory operand. The first access is needed to read the operand, and the second access is needed to write it back after modification. Determine the number and size of operands, multiply by two (the mask used in bit clear and set instructions is considered to be an immediate operand), then use **Table 8-1** to determine the number of cycles.

8.2.2.3 Change-of-Flow Instructions

Operand access for change of flow instructions varies according to type. Unary branches, conditional branches, and jumps have no operand access. Bit-condition branches must make one memory access in order to perform masking. Subroutine and interrupt instructions must make stack accesses.

In addition, when an instruction that can cause a change in flow executes, no prefetch is made until after the precondition for change of flow is evaluated.

There are two evaluation cases:

> If the instruction causes an unconditional change, or meets a specific precondition for change, the program counter is loaded with the first address of a new instruction stream, and the pipeline is filled with new instructions.
>
> If the instruction does not meet a specific precondition (preconditions of unary branches are always true or always false), prefetch is made and execution of the old instruction stream resumes.

Table 8-2 shows the number of program and operand access cycles for each instruction that causes a change in program flow.

Table 8-2 Change-of-Flow Instruction Timing

Instruction	Operand Access	Program Access	Comment
BRA	0	3	Unary branch (1 = 1)
BRN	0	1	Unary branch (1 = 0)
Short Branches	0	3/1	Conditional branches
LBRA	0	3	Unary branch (1 = 1)
LBRN	0	2	Unary branch (1 = 0)
Long Branches	0	3/2	Conditional branches
BRCLR	1	4/3	Bit-condition branch, IND8 addressing mode
BRCLR	1	5/3	Bit-condition branch, EXT, IND16 addressing modes
BRSET	1	4/3	Bit-condition branch, IND8 addressing mode
BRSET	1	5/3	Bit-condition branch, EXT, IND16 addressing modes
JMP	0	3	Unconditional
JSR	2	3	Operand accesses include stack access
BSR	2	3	Operand accesses include stack access
LBSR	2	3	Operand accesses include stack access
RTS	2	3	Operand accesses include stack access
SWI	3	3	Operand accesses include stack access and vector fetch
RTI	2	3	Operand accesses include stack access

In program access values for conditional branches, the first value is for branch taken, the second value is for branch not taken.

8.2.2.4 Stack Manipulation Instructions

Aligned stack manipulation instructions comply with normal program access constraints, but have extra operand access cycles for stacking operations. Treat misaligned stacking operations as byte transfers on a misaligned 16-bit bus.

Table 8-3 shows program and operand access cycles for each instruction.

Table 8-3 Stack Manipulation Timing

Instruction	Operand Access	Program Access	Comment
PSHA/PSHB	1	1	Byte operation
PULA/PULB	1	1	Byte operation
PSHM	N	1	N = Number of registers pushed
PULM	N + 1	1	N = Number of registers pulled*
PSHMAC/PULMAC	6	1	Stacks/retrieves all MAC registers

*The last operand read from the stack is ignored

8.2.2.5 Stop and Wait Instructions

Stop and wait instructions have normal program access cycles, but differ from regular instructions in number of operand accesses. If LPSTOP is executed at a time when the CCR S bit is equal to zero, it must make one operand access to store the CCR IP field. WAI performs one prefetch access to establish a PC value that insures proper stacking and return from interrupt.

Table 8-4 shows program and operand access cycles for each instruction.

Table 8-4 Stop and Wait Timing

Instruction	Operand Access	Program Access	Comment
LPSTOP1		1	Operand access only when CCR S Bit = 0
WAI	0	1	—

8.2.2.6 Move Instructions

Move instructions have normal program access cycles, but differ from regular instructions in number of operand accesses. Each move requires two operand accesses, one to read the data from the source address and one to write it to the destination address.

Table 8-5 shows program and operand access cycles for each instruction.

Table 8-5 Move Timing

Instruction	Operand Access	Program Access	Comment
MOVB/MOVW	2	2	IXP to EXT, EXT to IXP addressing modes
MOVB/MOVW	2	3	EXT to EXT addressing mode

8.2.2.7 Multiply and Accumulate Instructions

MAC instructions have normal program access cycles, but differ from regular instructions in number of operand accesses. During multiply and accumulate operation, two words pointed to by index registers X and Y are accessed and transferred to the H and I registers. MAC makes only these two operand accesses, but RMAC repeats the operation a specified number of times.

Table 8-6 shows program and operand access cycles for each instruction.

Table 8-6 MAC Timing

Instruction	Operand Access	Program Access	Comment
MAC	2	1	—
RMAC	2N	1	N = Number of iterations

8.3 Internal Operation Time

To determine the number of clock periods associated with internal operation, first determine program and operand access time using the appropriate table, then use instruction cycle time (CL_T) from the instruction glossary to evaluate the following expression:

$$CL_I = (CL_T) - (CL_P + CL_O)$$

Assume that:

1. All program and operand accesses are aligned on a 16-bit data bus.
2. Each bus cycle takes two clock periods.

This figure is constant regardless of the speed of memory used. Internal operations, prefetches, and operand fetches are wholly concurrent for many instructions — the calculated CL_I will be zero.

8.4 Calculating Execution Times for Slower Accesses

Because CL_I is constant for all bus speeds, CL_T will only change when CL_P and CL_O change. Clock periods are calculated using the following expression:

$$CL_X = \text{(Clock periods per bus cycle) (Number of bus cycles)}$$

Where:

CL_X is either CL_P or CL_O

To determine the number of clock periods required to execute an instruction when bus cycles longer than two system clock periods are necessary, determine the number of cycles needed, calculate CL_P and CL_O values, then add to CL_I.

8.5 Examples

The examples below illustrate the effect of bus width, alignment, and access speed on three instructions. Separate entries for operand and program access show the effect of accesses from differing types of memory.

The first example for each instruction assumes two system clock cycles per bus cycle and 16-bit aligned access, so that CL_I can be determined and used in the subsequent examples. Calculated values are underlined.

8.5.1 LDD (Load D) Instruction

The general form of this instruction is: LDD (operand). Examples show effects of various access parameters on a single-word instruction.

8.5.1.1 LDD IND8, X

16-bit operand data bus, 2 clocks per bus cycle, aligned
16-bit program data bus, 2 clocks per bus cycle

					CL_T
					6
Operand	**Number of Accesses**	**Bus Width**	**Number of Bus Cycles**	**Clocks per Bus Cycle**	**CL_O**
	1	16	**1**	2	**2**
Program	**Number of Accesses**	**Bus Width**	**Number of Bus Cycles**	**Clocks per Bus Cycle**	**CL_P**
	1	16	**1**	2	**2**
					CL_I
					2

8.5.1.2 LDD IND8, X

8-bit operand data bus, 3 clocks per bus cycle, aligned
16-bit program data bus, 2 clocks per bus cycle

					CL_T
					10
Operand	**Number of Accesses**	**Bus Width**	**Number of Bus Cycles**	**Clocks per Bus Cycle**	**CL_O**
	1	8	**2**	3	**6**
Program	**Number of Accesses**	**Bus Width**	**Number of Bus Cycles**	**Clocks per Bus Cycle**	**CL_P**
	1	16	**1**	2	**2**
					CL_I
					2

8.5.1.3 LDD IND8, X

16-bit operand data bus, 2 clocks per bus cycle, misaligned
8-bit program data bus, 3 clocks per bus cycle

					CL_T
					12
Operand	**Number of Accesses**	**Bus Width**	**Number of Bus Cycles**	**Clocks per Bus Cycle**	**CL_O**
	1	16	**2**	2	**4**
Program	**Number of Accesses**	**Bus Width**	**Number of Bus Cycles**	**Clocks per Bus Cycle**	**CL_P**
	1	8	**2**	3	**6**
					CL_I
					2

8.5.2 NEG (Negate) Instruction

The general form of this instruction is: NEG (operand). Examples show effects of various access parameters on a two-word instruction. Note that operand alignment affects only the 8-bit operand data bus.

8.5.2.1 NEG EXT

16-bit operand data bus, 2 clocks per bus cycle
16-bit program data bus, 2 clocks per bus cycle

					CL_T
					8
Operand	**Number of Accesses**	**Bus Width**	**Number of Bus Cycles**	**Clocks per Bus Cycle**	CL_O
	2	16	2	2	4
Program	**Number of Accesses**	**Bus Width**	**Number of Bus Cycles**	**Clocks per Bus Cycle**	CL_P
	2	16	2	2	4
					CL_I
					0

8.5.2.2 NEG EXT

8-bit operand data bus, 3 clocks per bus cycle, aligned
8-bit program data bus, 3 clocks per bus cycle

					CL_T
					18
Operand	**Number of Accesses**	**Bus Width**	**Number of Bus Cycles**	**Clocks per Bus Cycle**	CL_O
	2	8	2	3	6
Program	**Number of Accesses**	**Bus Width**	**Number of Bus Cycles**	**Clocks per Bus Cycle**	CL_P
	2	8	4	3	12
					CL_I
					0

8.5.2.3 NEG EXT

16-bit operand data bus, 3 clocks per bus cycle
16-bit program data bus, 3 clocks per bus cycle

					CL_T
					12
Operand	**Number of Accesses**	**Bus Width**	**Number of Bus Cycles**	**Clocks per Bus Cycle**	
	2	16	2	3	6
Program	**Number of Accesses**	**Bus Width**	**Number of Bus Cycles**	**Clocks per Bus Cycle**	CL_P
	2	16	2	3	6
					CL_I
					0

8.5.3 STED (Store Accumulators E and D) Instruction

The general form of this instruction is: STED (operand). Examples show effects of various access parameters on an instruction that writes to memory twice during execution.

8.5.3.1 STED EXT

16-bit operand data bus, 2 clocks per bus cycle, aligned
16-bit program data bus, 2 clocks per bus cycle

					CL_T
					8
Operand	**Number of Accesses**	**Bus Width**	**Number of Bus Cycles**	**Clocks per Bus Cycle**	CL_O
	1	16	**2**	2	**4**
Program	**Number of Accesses**	**Bus Width**	**Number of Bus Cycles**	**Clocks per Bus Cycle**	CL_P
	2	16	**2**	2	**4**
					CL_I
					0

8.5.3.2 STED EXT

8-bit operand data bus, 2 clocks per bus cycle, misaligned
16-bit program data bus, 3 clocks per bus cycle

					CL_T
					14
Operand	**Number of Accesses**	**Bus Width**	**Number of Bus Cycles**	**Clocks per Bus Cycle**	CL_O
	1	8	**4**	2	**8**
Program	**Number of Accesses**	**Bus Width**	**Number of Bus Cycles**	**Clocks per Bus Cycle**	CL_P
	2	16	**2**	3	**6**
					CL_I
					0

SECTION 9 EXCEPTION PROCESSING

This section discusses exception handling, exception processing sequence, and specific features of individual exceptions.

9.1 Definition of Exception

An exception is an event that pre-empts normal instruction process. Exception processing makes the transition from normal instruction execution to execution of a routine that deals with an exception.

Each exception has an assigned vector that points to an associated handler routine. Exception processing includes all operations required to transfer control to a handler routine, but does not include execution of the handler routine itself. Keep the distinction between exception processing and execution of an exception handler in mind while reading this section.

9.2 Exception Vectors

An exception vector is the address of a routine that handles an exception. Exception vectors are contained in a data structure called the instruction vector table, which is located in the first 512 bytes of bank 0.

All vectors except the reset vector consist of one word and reside in data space. The reset vector consists of four words that reside in program space. There are 52 predefined or reserved vectors, and 200 user-defined vectors.

Each vector is assigned an 8-bit number. Vector numbers for some exceptions are generated by external devices; others are supplied by the processor. There is a direct mapping of vector number to vector table address. The processor left shifts the vector number one place (multiplies by two) to convert it to an address.

Table 9-1 shows exception vector table organization. Vector numbers and addresses are given in hexadecimal notation.

Table 9-1 Exception Vector Table

Vector Number	Vector Address	Address Space	Type of Exception
0	0000	P	RESET — Initial ZK, SK, and PK
	0002	P	RESET — Initial PC
	0004	P	RESET — Initial SP
	0006	P	RESET — Initial IZ (Direct Page)
4	0008	D	BKPT (Breakpoint)
5	000A	D	BERR (Bus Error)
6	000C	D	SWI (Software Interrupt)
7	000E	D	Illegal Instruction
8	0010	D	Division by Zero
9 – E	0012 – 001C	D	Unassigned, Reserved
F	001E	D	Uninitialized Interrupt
10	0020	D	Unassigned, Reserved
11	0022	D	Level 1 Interrupt Autovector
12	0024	D	Level 2 Interrupt Autovector
13	0026	D	Level 3 Interrupt Autovector
14	0028	D	Level 4 Interrupt Autovector
15	002A	D	Level 5 Interrupt Autovector
16	002C	D	Level 6 Interrupt Autovector
17	002E	D	Level 7 Interrupt Autovector
18	0030	D	Spurious Interrupt
19 – 37	0032 – 006E	D	Unassigned, Reserved
38 – FF	0070 – 01FE	D	User-defined Interrupts

9.3 Types of Exceptions

Exceptions can be either internally or externally generated. External exceptions, which are defined as asynchronous, include interrupts, bus errors ($\overline{\text{BERR}}$), breakpoints ($\overline{\text{BKPT}}$), and resets ($\overline{\text{RESET}}$). Internal exceptions, which are defined as synchronous, include the software interrupt (SWI) instruction, the background ($\overline{\text{BGND}}$) instruction, illegal instruction exceptions, and the divide-by-zero exception.

9.4 Exception Stack Frame

During exception processing, a subset of the current processor state is saved on the current stack. Specifically, the contents of the program counter and condition code register at the time exception processing begins are stacked at the location pointed to by SK: SP. Unless specifically altered during exception processing, the stacked PK: PC value is the address of the next instruction in the current instruction stream, plus $0006. **Figure 9-1** shows the exception stack frame.

Low Address		⇐ SP After Exception Stacking
	Condition Code Register	
High Address	Program Counter	⇐ SP Before Exception Stacking

Figure 9-1 Exception Stack Frame Format

9.5 Exception Processing Sequence

This is a general description of exception processing. **Figure 9-2** shows detailed processing flow and relative priority of each type of exception.

Exception processing is performed in four distinct phases.

1. Priority of all pending exceptions is evaluated, and the highest priority exception is processed first.
2. Processor state is stacked, then the CCR PK extension field is cleared.
3. An exception vector number is acquired and converted to a vector address.
4. The content of the vector address is loaded into the PC, and the processor jumps to the exception handler routine.

There are variations within each phase for differing types of exceptions. However, all vectors but $\overline{\text{RESET}}$ are 16-bit addresses, and the PK field is cleared — either exception handlers must be located within bank 0, or vectors must point to a jump table. See **9.7 Processing of Specific Exceptions**.

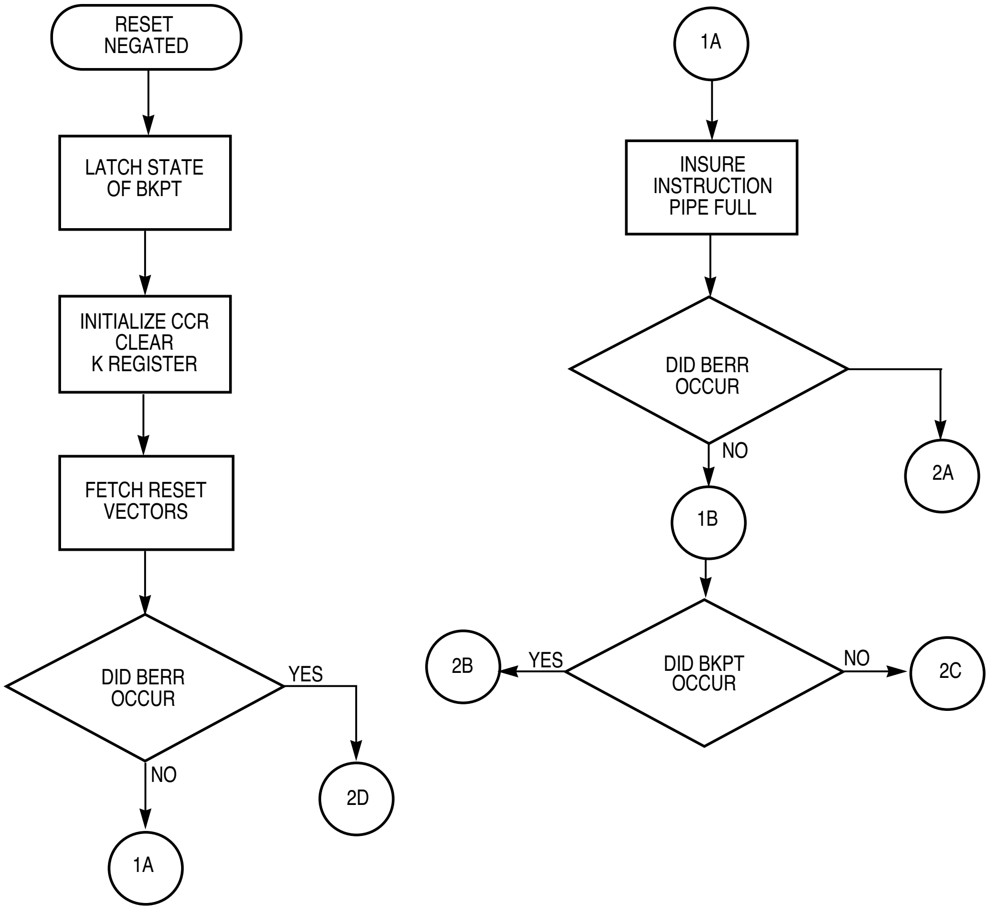


Figure 9-2 (Sheet 1 of 5) Exception Processing Flow Diagram

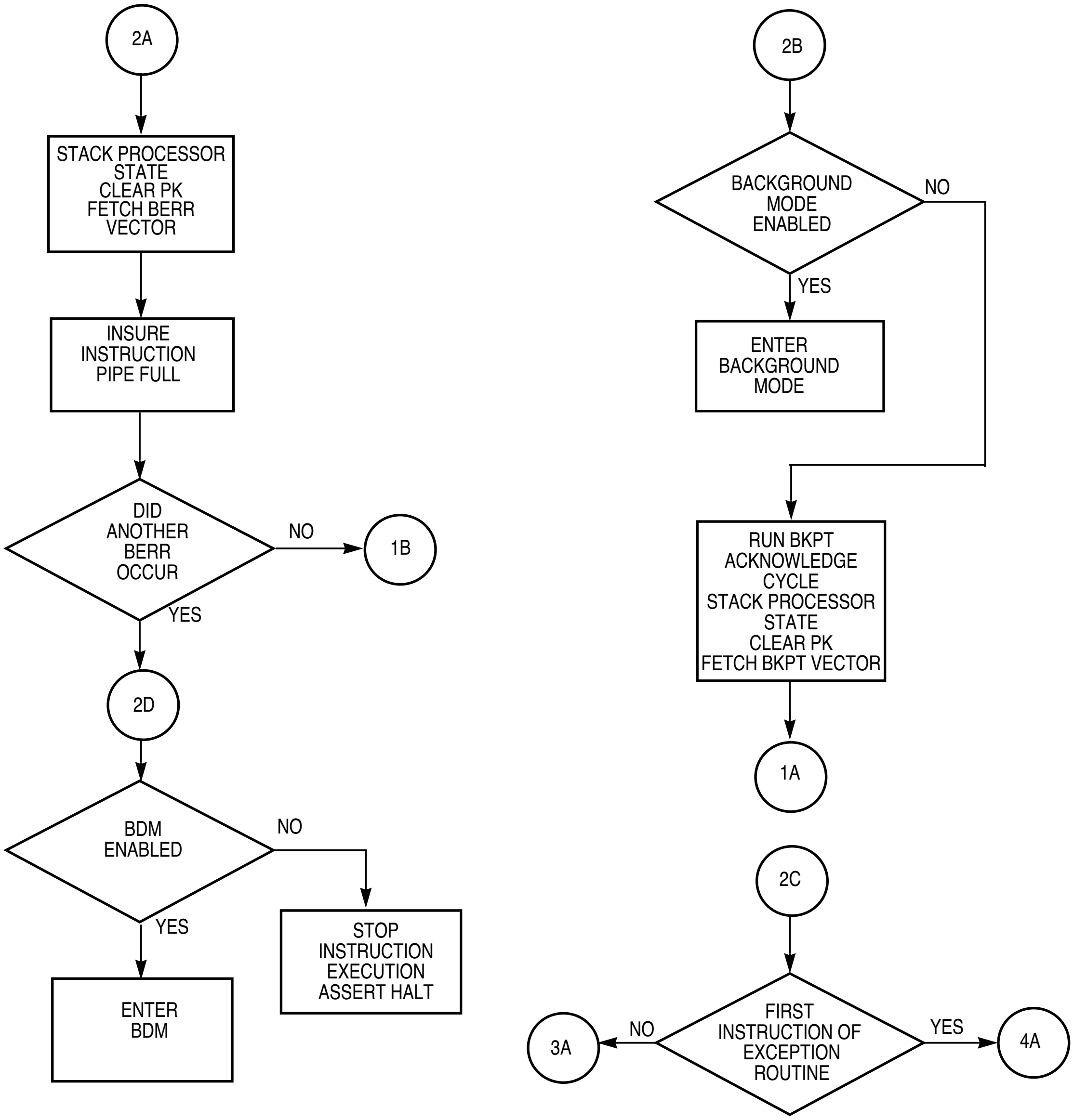

2A
STACK PROCESSOR STATE CLEAR PK FETCH BERR VECTOR
INSURE INSTRUCTION PIPE FULL
DID ANOTHER BERR OCCUR
NO
1B
YES
2D
BDM ENABLED
NO
YES
ENTER BDM
STOP INSTRUCTION EXECUTION ASSERT HALT
2B
BACKGROUND MODE ENABLED
NO
YES
ENTER BACKGROUND MODE
RUN BKPT ACKNOWLEDGE CYCLE STACK PROCESSOR STATE CLEAR PK FETCH BKPT VECTOR
1A
2C
FIRST INSTRUCTION OF EXCEPTION ROUTINE
NO
3A
YES
4A

Figure 9-2 (Sheet 2 of 5) Exception Processing Flow Diagram

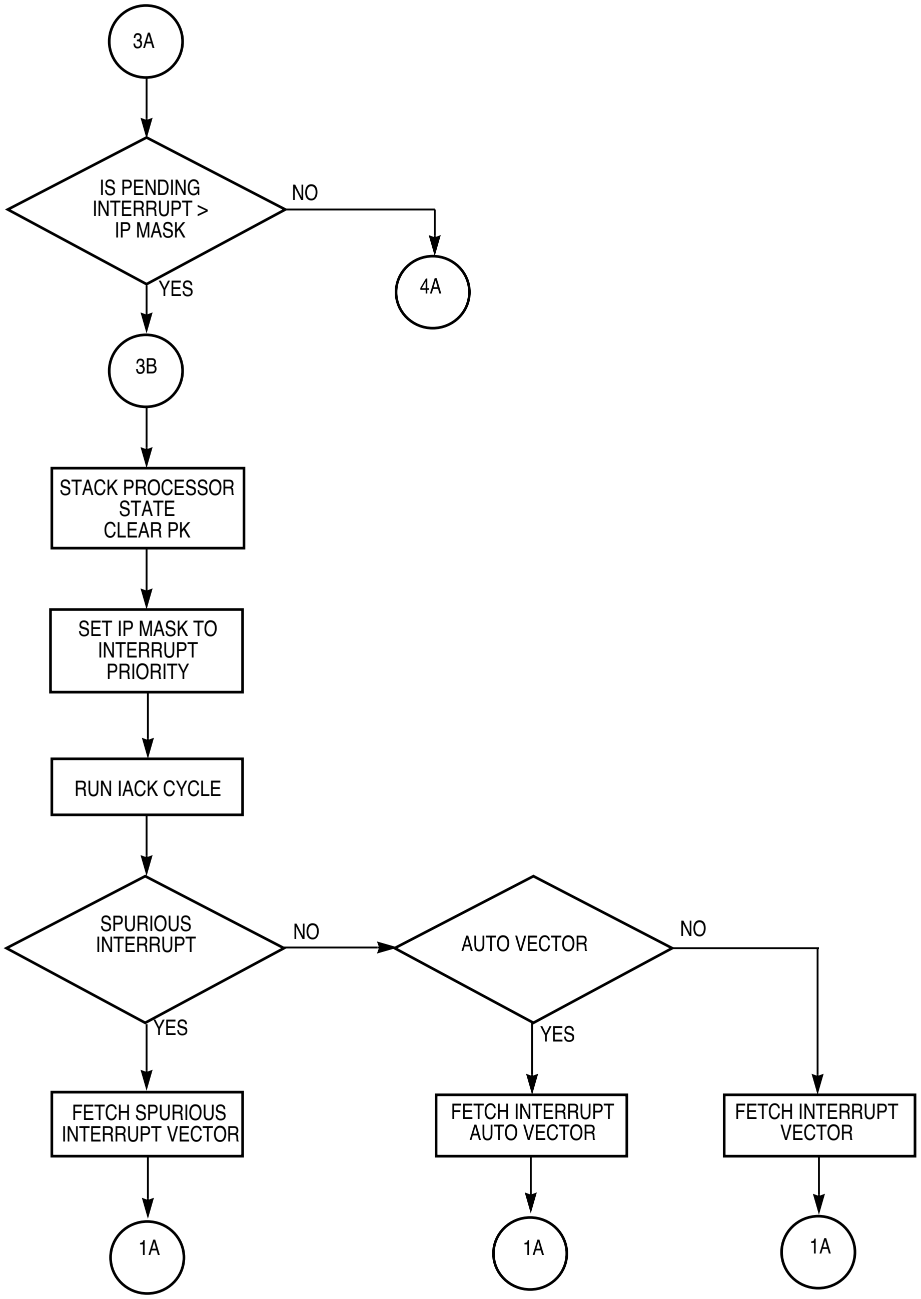

3A
IS PENDING INTERRUPT > IP MASK
NO
YES
4A
3B
STACK PROCESSOR STATE CLEAR PK
SET IP MASK TO INTERRUPT PRIORITY
RUN IACK CYCLE
SPURIOUS INTERRUPT
NO
YES
AUTO VECTOR
NO
YES
FETCH SPURIOUS INTERRUPT VECTOR
FETCH INTERRUPT AUTO VECTOR
FETCH INTERRUPT VECTOR
1A
1A
1A

Figure 9-2 (Sheet 3 of 5) Exception Processing Flow Diagram

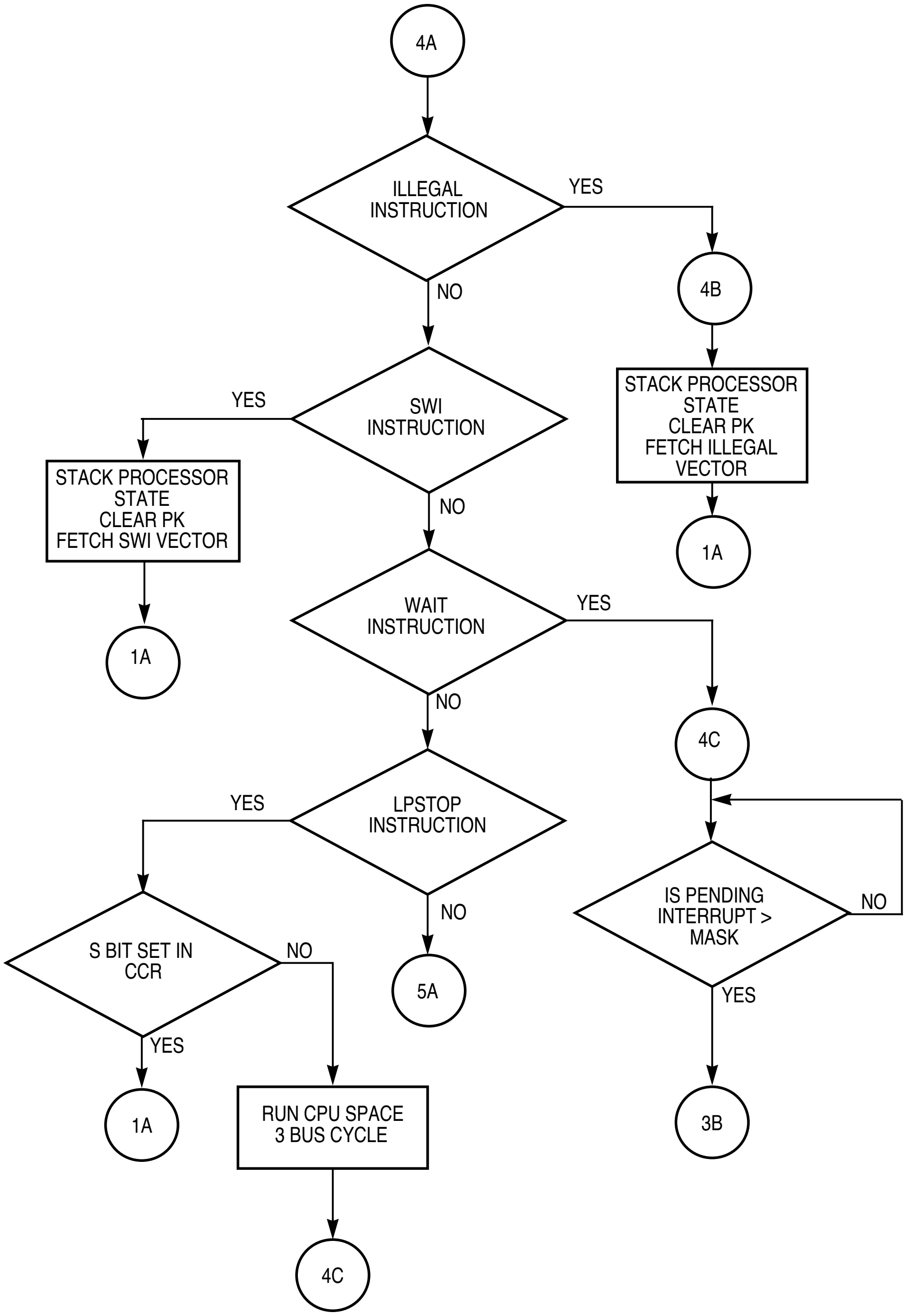


Figure 9-2 (Sheet 4 of 5) Exception Processing Flow Diagram

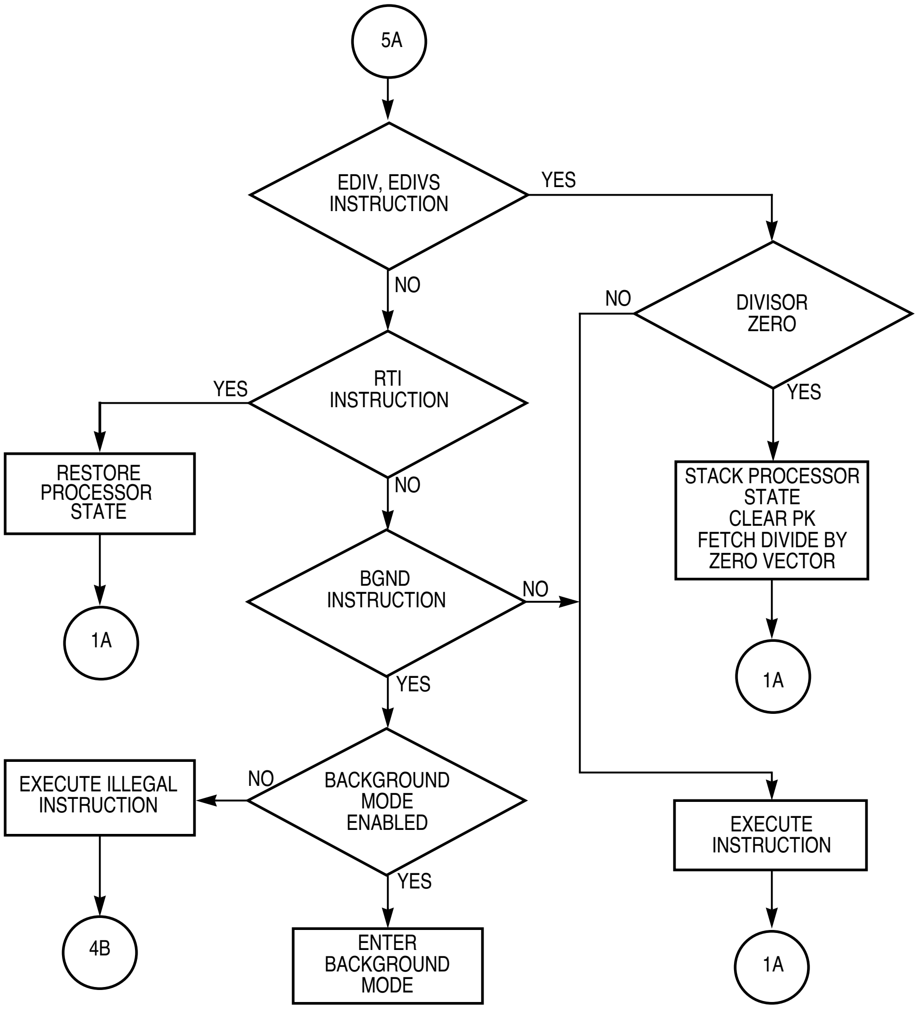


Figure 9-2 (Sheet 5 of 5) Exception Processing Flow Diagram

9.6 Multiple Exceptions

Each exception has a priority based upon its relative importance to system operation. Asynchronous exceptions have higher priorities than synchronous exceptions. Exception processing for multiple exceptions is done by priority, from highest to lowest. Priority governs the order in which exception processing occurs, not the order in which exception handlers are executed.

When simultaneous exceptions occur, handler routines for lower priority exceptions are generally executed before handler routines for higher priority exceptions.

Unless BERR, BKPT, or RESET occur during exception processing, the first instruction of all exception handler routines is guaranteed to execute before another exception is processed. Since interrupt exceptions have higher priority than synchronous exceptions, this means that the first instruction in an interrupt handler will be executed before other interrupts are sensed.

Note

> If interrupt latency is a concern, it is best to lead interrupt service routines with a NOP instruction, rather than with an instruction that requires considerable cycle time to execute, such as PSHM.

RESET, BERR, and BKPT exceptions that occur during exception processing of a previous exception will be processed before the first instruction of that exception's handler routine. The converse is not true — if an interrupt occurs during BERR exception processing, for example, the first instruction of the BERR handler will be executed before interrupts are sensed. This permits the exception handler to mask interrupts during execution.

9.7 Processing of Specific Exceptions

The following detailed discussion of exceptions is organized by type and priority. Proximate causes of each exception are discussed, as are variations from the standard processing sequence described above.

9.7.1 Asynchronous Exceptions

Asynchronous exceptions occur without reference to CPU16 or IMB clocks, but exception processing is synchronized. For all asynchronous exceptions besides RESET, exception processing begins at the first instruction boundary following detection of an exception.

Because of pipelining, the stacked return PK : PC value for all asynchronous exceptions, other than RESET, is equal to the address of the next instruction in the current instruction stream plus $0006. The RTI instruction, which must terminate all exception handler routines, subtracts $0006 from the stacked value in order to resume execution of the interrupted instruction stream.

9.7.1.1 Processor Reset (RESET)

RESET is the highest-priority exception. It provides for system initialization and for recovery from catastrophic failure. The RESET vector contains information necessary for basic CPU16 initialization. **Figure 9-3** shows the RESET vector.

Address	15	12	11	8	7	4	3	0
$0000	Reserved		Initial ZK		Initial SK		Initial PK	
$0002	Initial PC							
$0004	Initial SP							
$0006	Initial IZ (Direct Page Pointer)							

Figure 9-3 RESET Vector

RESET is caused by assertion of the IMB MSTRST signal. Conditions for assertion of MSTRST may vary among members of the modular microcontroller family. Refer to the appropriate microcontroller user's manual for details.

Unlike all other exceptions, RESET occurs at the end of a bus cycle, and not at an instruction boundary. Any processing in progress at the time RESET occurs will be aborted, and cannot be recovered.

The following events take place when MSTRST is asserted.

A. Instruction execution is aborted.
B. The condition code register is initialized.
   1. The IP field is set to $7, disabling all interrupts below priority 7.
   2. The S bit is set, disabling LPSTOP mode.
   3. The SM bit is cleared, disabling MAC saturation mode.
C. The K register is cleared.

It is important to be aware that all CCR bits that are not initialized are not affected by reset. However, out of power-on reset, these bits will be indeterminate.

The following events take place when MSTRST is negated after assertion.

A. The CPU16 samples the $\overline{\text{BKPT}}$ input.
B. The CPU16 fetches RESET vectors in the following order:
   1. Initial ZK, SK, and PK extension field values.
   2. Initial PC.
   3. Initial SP.
   4. Initial IZ value.
C. The CPU16 begins fetching instructions pointed to by the initial PK : PC.

The CPU16 samples the $\overline{\text{BKPT}}$ inputs to determine whether to enable background debugging mode.

If either $\overline{\text{BKPT}}$ input is at logic level zero when sampled, an internal BDM flag is set, and the CPU16 enters BDM whenever either $\overline{\text{BKPT}}$ input is subsequently asserted.

If both $\overline{\text{BKPT}}$ inputs are at logic level one when sampled, normal BKPT exception processing begins whenever either $\overline{\text{BKPT}}$ input is subsequently asserted.

When BDM is enabled, the CPU16 will enter debugging mode whenever the conditions for breakpoint are met. See **9.7.1.3 Breakpoint Exception (BKPT)** for more information.

ZK : IZ are initialized for use as a direct bank pointer. Using the pointer, any location in memory can be accessed out of reset by means of indexed addressing. This capability maintains compatibility with MC68HC11 routines that use direct addressing mode.

Only essential RESET tasks are performed during exception processing. Other initialization tasks must be accomplished by the exception handler routine.

9.7.1.2 Bus Error ($\overline{\text{BERR}}$)

BERR is caused by assertion of the IMB $\overline{\text{BERR}}$ signal. $\overline{\text{BERR}}$ can be asserted by any of three sources:

1. External logic, via the $\overline{\text{BERR}}$ pin.
2. Another microcontroller module.
3. Microcontroller system watchdog functions.

Refer to the appropriate microcontroller user's manual for more information.

BERR assertions do not force immediate exception processing. The signal is synchronized with normal bus cycles and is latched into the CPU16 at the end of the bus cycle in which it was asserted. Since bus cycles can overlap instruction boundaries, bus error exception processing may not occur at the end of the instruction in which the bus cycle begins. Timing of $\overline{\text{BERR}}$ detection/acknowledge is dependent upon several factors:

Which bus cycle of an instruction is terminated by assertion of $\overline{\text{BERR}}$.

The number of bus cycles in the instruction during which $\overline{\text{BERR}}$ is asserted.

The number of bus cycles in the instruction following the instruction in which $\overline{\text{BERR}}$ is asserted.

Whether $\overline{\text{BERR}}$ is asserted during a program space access or a data space access.

Because of these factors, it is impossible to predict precisely how long after occurrence of a bus error the bus error exception will be processed.

Caution

> The external bus interface in the system integration module does not latch data when an external bus cycle is terminated by a bus error. When this occurs during an instruction prefetch, the IMB precharge state (bus pulled high, or $FF) is latched into the CPU16 instruction register, with indeterminate results. Refer to **SECTION 3 SYSTEM RESOURCES** for more information concerning the IMB and bus interfacing.

Bus error exception support in the CPU16 is provided to allow for dynamic memory sizing after reset. To implement this feature, use a small routine similar to the example below. The example assumes that memory starts at address $00000, and is contiguous through the highest memory address —it must be modified for other memory maps.

Example — Dynamic Memory Sizing

```
            clrb              set xk = 0
         tbxk
         ldx      #$0000      xk:ix initialized to address $00000
loop     ldd      0,x         access memory location
         nop                  nop in case a bus error is pending
         aix      #2          increment pointer to next word address.
         bra      loop
*
*        When xk:ik is incremented past the highest available memory
*        address, a BERR exception occurs; after exception processing,
*        the CPU16 executes the exception handler at location berr_ex.
*
*        berr_ex – BERR Exception Handler for Dynamic Memory Sizing
*
*        This routine computes the address of the last word of memory,
*        then stores the bank number at a location called “bank” and the
*        word address within the bank at a location called “address”.
*        It assumes that ek is properly initialized.
*
berr_ex  aix      #–2      compute LWA of memory
         txkb
         stab     bank     store bank number
         stx      address  store address
```

Exception processing for bus error exceptions follows the standard exception processing sequence. However, two special cases of bus error, called double bus faults, can abort exception processing.

BERR assertion is not detected until an instruction is complete. The $\overline{\text{BERR}}$ latch is cleared by the first instruction of the BERR exception handler. Double bus fault occurs in two ways:

1. When bus error exception processing begins and a second $\overline{\text{BERR}}$ is detected before the first instruction of the BERR exception handler is executed.
2. When one or more bus errors occur before the first instruction after a RESET exception is executed.

Multiple bus errors within a single instruction which can generate multiple bus cycles, such as read-modify-write instructions (refer to **SECTION 8 INSTRUCTION TIMING** for more information), will cause a single bus error exception after the instruction has executed.

Immediately after assertion of a second $\overline{\text{BERR}}$, the CPU16 ceases instruction processing and asserts the IMB HALT signal. The CPU16 will remain in this state until a RESET occurs.

9.7.1.3 Breakpoint Exception (BKPT)

BKPT is caused by internal assertion of the IMB $\overline{\text{BKPT}}$ signal or by external assertion of the microcontroller $\overline{\text{BKPT}}$ pin. $\overline{\text{BKPT}}$ assertions do not force immediate exception processing. They are synchronized with normal bus cycles and latched into the CPU16 at the end of the bus cycle in which they are asserted.

When a $\overline{\text{BKPT}}$ assertion is synchronized with an instruction prefetch, processing of the BKPT exception occurs at the end of that instruction. The prefetched instruction is "tagged" with the breakpoint when it enters the instruction pipeline, and the breakpoint exception occurs after the instruction executes. When a $\overline{\text{BKPT}}$ assertion is synchronized with an operand fetch, exception processing occurs at the end of the instruction during which $\overline{\text{BKPT}}$ is latched.

When background debugging mode has been enabled, the CPU16 will enter BDM whenever either $\overline{\text{BKPT}}$ input is asserted. Refer to **SECTION 10 DEVELOPMENT SUPPORT** for complete information on background debugging mode. When background debugging mode is not enabled, a breakpoint acknowledge bus cycle is run, and subsequent exception processing follows the normal sequence.

Breakpoint acknowledge is a type of CPU space cycle. Cycles of this type are managed by the external bus interface (EBI) in the microcontroller system integration module. See **SECTION 3 SYSTEM RESOURCES** for more information.

9.7.1.4 Interrupts

There are eight levels of interrupt priority (0–7), seven automatic interrupt vectors, and 200 assignable interrupt vectors. All interrupts with priorities less than 7 can be masked by writing to the CCR interrupt priority field.

Interrupt requests do not force immediate exception processing, but are left pending until the current instruction is complete. Pending interrupts are processed at instruction boundaries or when exception processing for higher-priority exceptions is complete. All interrupt requests must be held asserted until they are acknowledged by the CPU.

Interrupt recognition and subsequent processing are based on the state of interrupt request signals $\overline{\text{IRQ7}}$ – $\overline{\text{IRQ1}}$ and the IP mask value.

$\overline{\text{IRQ6}}$ – $\overline{\text{IRQ1}}$ are active-low level-sensitive inputs. $\overline{\text{IRQ7}}$ is an active-low transition-sensitive input. A transition-sensitive input requires both an edge and a voltage level for validity. Interrupt requests are synchronized and debounced by input circuitry on consecutive rising edges of the processor clock. To be valid, an interrupt request must be asserted for at least two consecutive clock periods. Each input corresponds to an interrupt priority. $\overline{\text{IRQ1}}$ has the lowest priority, and $\overline{\text{IRQ7}}$ has the highest priority.

The IP field consists of three bits (CCR[7:5]). Binary values %000 to %111 provide eight priority masks. Masks prevent an interrupt request of a priority less than or equal to the mask value (except for $\overline{\text{IRQ7}}$) from being recognized and processed. When IP contains %000, no interrupt is masked.

$\overline{\text{IRQ6}}$ – $\overline{\text{IRQ1}}$ are maskable. $\overline{\text{IRQ7}}$ is non-maskable. The $\overline{\text{IRQ7}}$ input is transition-sensitive in order to prevent redundant servicing and stack overflow. An NMI is generated each time $\overline{\text{IRQ7}}$ is asserted, and each time the priority mask changes from %111 to a lower number while $\overline{\text{IRQ7}}$ is asserted.

The IP field is automatically set to the priority of the pending interrupt as a part of interrupt exception processing. The TDP, ANDP, and ORP instructions can be used to change the IP mask value. IP can also be changed by pushing a modified CCR onto the stack, then using the PULM instruction. IP is also modified by the action of the return from interrupt (RTI) instruction.

Interrupt exception processing sequence is as follows:

A. Priority of all pending exceptions is evaluated, and the highest priority exception is processed first.
B. Processor state is stacked, then the CCR PK extension field is cleared.
C. Mask value of the pending interrupt is written to the IP field.
D. An interrupt acknowledge cycle (IACK) is run.
   1. If the interrupting device supplies a vector number, the CPU16 acquires it.
   2. If the interrupting device asserts the autovector (AVEC) signal in response to IACK, the CPU16 generates an autovector number corresponding to the interrupt priority.
   3. If a $\overline{\text{BERR}}$ signal occurs during IACK, the CPU16 generates the spurious interrupt vector number.
E. The vector number is converted to a vector address.
F. The content of the vector address is loaded into the PC, and the processor jumps to the exception handler routine.

SECTION 3 SYSTEM RESOURCES contains more information about bus control signals and interfacing.

9.7.2 Synchronous Exceptions

Synchronous exception processing is part of an instruction definition. Exception processing for synchronous exceptions will always be completed, and the first instruction of the handler routine will always be executed, before interrupts are detected.

Because of pipelining, the value of PK : PC at the time a synchronous exception executes is equal to the address of the instruction that causes the exception plus $0006. Since RTI always subtracts $0006 upon return, the stacked PK : PC must be adjusted by the instruction that caused the exception so that execution will resume with the following instruction —$0002 is added to the PK : PC value before it is stacked.

9.7.2.1 Illegal Instructions

An illegal instruction exception can occur at two times:

1. When the execution unit identifies an opcode for which there is no instruction definition.
2. When an attempt is made to execute the BGND instruction with background debugging mode disabled.

In both cases, exception processing follows the normal sequence, except that the PK : PC value is adjusted before it is stacked.

9.7.2.2 Division By Zero

This exception is a part of the instruction definition for division instructions EDIV and EDIVS. If the divisor is zero when either is executing, the exception is taken. In both cases, exception processing follows the normal sequence, except that the PK : PC value is adjusted before it is stacked.

9.7.2.3 BGND Instruction

Execution of the BGND instruction differs depending upon whether background debugging mode has been enabled. See **9.7.1.3 Breakpoint Exception (BKPT)** for information concerning enabling BDM.

1. If BDM has been enabled, BDM is entered. See **SECTION 10 DEVELOPMENT SUPPORT** for more information concerning BDM.
2. If BDM is not enabled, an illegal instruction exception occurs. In this case, exception processing follows the normal sequence, except that the PK : PC value is adjusted before it is stacked.

9.7.2.4 SWI Instruction

The software interrupt instruction initiates synchronous exception processing. Exception processing for SWI follows the normal sequence, except that the PK : PC value is adjusted before it is stacked.

9.8 Return from Interrupt (RTI)

RTI must be the last instruction in all exception handlers except for the RESET handler. RTI pulls the exception stack frame and restores processor state. Normal program flow resumes at the address of the instruction that follows the last instruction executed before exception processing began. RTI is not used in the RESET handler because RESET initializes the stack pointer and does not create a stack frame.

SECTION 10 DEVELOPMENT SUPPORT

The CPU16 incorporates powerful tools for tracking program execution and for system debugging. These tools are deterministic opcode tracking, breakpoint exceptions, and the background debugging mode. Judicious use of CPU16 capabilities permits in-circuit emulation and system debugging using a bus state analyzer, a simple serial interface, and a terminal.

10.1 Deterministic Opcode Tracking

The CPU16 has two multiplexed outputs, IPIPE0 and IPIPE1, that enable external hardware to monitor the instruction pipeline during normal program execution. The signals IPIPE0 and IPIPE1 can be demultiplexed into six pipeline state signals that allow a state analyzer to synchronize with instruction stream activity.

10.1.1 Instruction Pipeline

There are three functional blocks involved in fetching, decoding, and executing instructions. These are the microsequencer, the instruction pipeline, and the execution unit. These elements function concurrently. **Figure 10-1** shows the functional blocks.

The microsequencer controls the order in which instructions are fetched, advanced through the pipeline, and executed. It increments the program counter and generates IPIPE0 and IPIPE1 from internal signals.

The execution unit evaluates opcodes, interfaces with the microsequencer to advance instructions through the pipeline, and performs instruction operations.

The effects of microsequencer and execution unit actions are always reflected in pipeline status — consequently, monitoring the pipeline provides an accurate picture of CPU16 operation for debugging purposes.

The pipeline is a three stage FIFO. Fetched opcodes are latched into stage A, then advanced to stage B, where opcodes are evaluated. The execution unit accesses operands from either stage A or stage B (stage B accesses are limited to 8-bit operands). After execution, opcodes are moved from stage B to stage C, where they remain until the next instruction is complete.

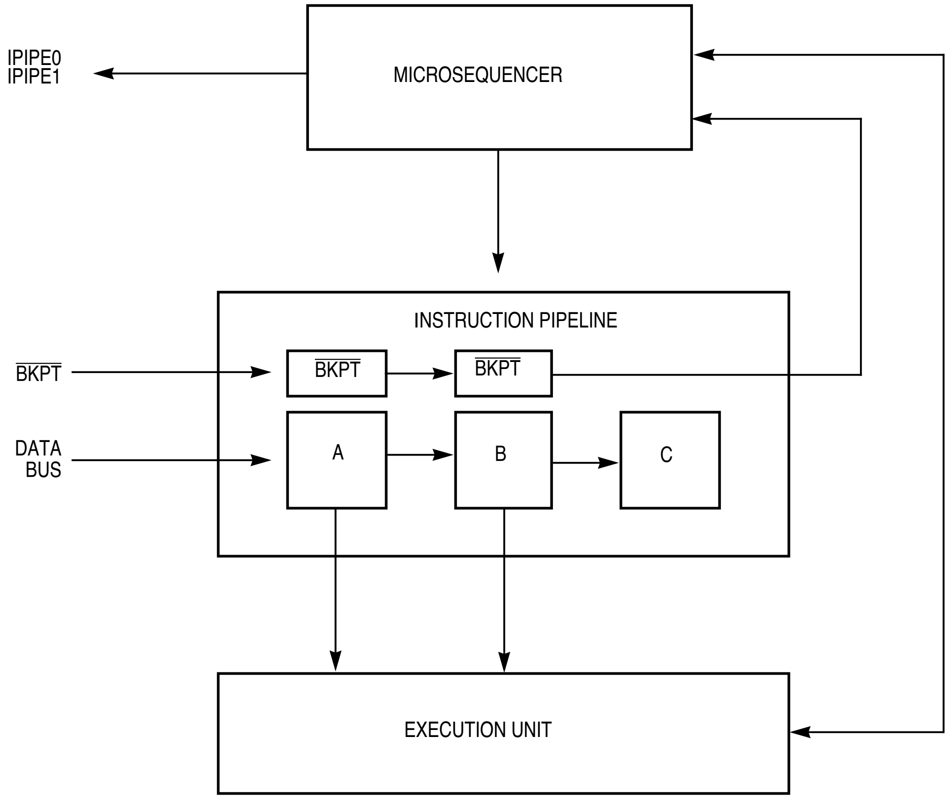


Figure 10-1 Instruction Execution Model

10.1.2 IPIPE0/IPIPE1 Multiplexing

Six types of information are required to track pipeline activity. To generate the six state signals, eight pipeline states are encoded and multiplexed into IPIPE0 and IPIPE1. The multiplexed signals have two phases. State signals are active low. **Table 10-1** shows the encoding and multiplexing scheme.

Table 10-1 IPIPE0/IPIPE1 Encoding

Phase	IPIPE1 State	IPIPE0 State	State Signal Name
1	0 0 1 1	0 1 0 1	START & FETCH FETCH START NULL
2	0 0 1 1	0 1 0 1	INVALID ADVANCE EXCEPTION NULL

IPIPE0 and IPIPE1 are timed so that a logic analyzer can capture all six pipeline state signals and address, data, or control bus state in any single bus cycle.

State signals can be latched asynchronously on the falling and rising edges of either address strobe ($\overline{\text{AS}}$) or data strobe ($\overline{\text{DS}}$). They can also be latched synchronously using the microcontroller CLKOUT signal. **SECTION 3 SYSTEM RESOURCES** contains more information about bus control signals. Refer to the appropriate microcontroller user's manual for specific timing information.

Figure 10-2 shows minimum logic required to demultiplex IPIPE0 and IPIPE1.

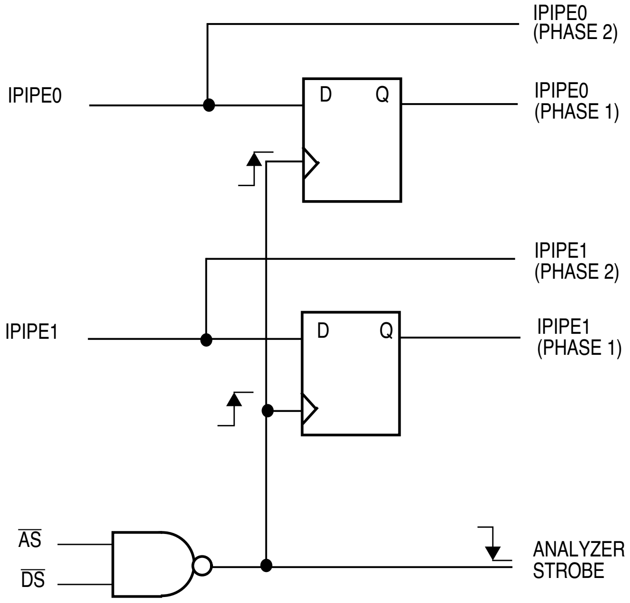


Figure 10-2 IPIPE DEMUX Logic

10.1.3 Pipeline State Signals

The six state signals show instruction execution sequence. The order in which a development system evaluates the signals is critical. In particular, the development system must first evaluate START, then ADVANCE, and then FETCH for each instruction word. When combined START & FETCH signals are asserted, START applies to the current content of pipeline stage B, while FETCH applies to current data bus content. Relationships between state signals are discussed in the following descriptions.

10.1.3.1 NULL — No Instruction Pipeline Activity

NULL assertion indicates that there is no instruction pipeline activity associated with the current bus cycle.

10.1.3.2 START — Instruction Start

START assertion indicates that an instruction in stage B has begun to execute. START affects subsequent operation of ADVANCE and FETCH. The development system must flag the instruction word in stage B as started when START is asserted.

10.1.3.3 ADVANCE — Instruction Pipeline Advance

ADVANCE assertion indicates that words in the instruction pipeline are being copied from one stage to another.

If START has been asserted for the word in stage B, the content of stage B is copied into stage C. Regardless of START assertion, content of stage A is copied into stage B.

When a word is copied from stage B to stage C, instruction execution is complete, and a new opcode must be copied into stage B.

When the content of stage A is copied into stage B, prior content of stage B is overwritten. ADVANCE assertion without an associated START assertion indicates that the pipeline is being filled, either before normal execution of instructions begins or after a change of program flow.

If the development system has flagged the instruction word in stage B as started, that flag must be cleared when ADVANCE is asserted.

10.1.3.4 FETCH — Instruction Fetch

FETCH assertion shows that the current content of the data bus is being latched into stage A. FETCH occurs only during instruction fetch bus cycles.

10.1.3.5 EXCEPTION — Exception Processing in Progress

EXCEPTION assertion indicates that all subsequent bus cycles until the next START assertion are part of an exception processing sequence.

EXCEPTION is not asserted during exceptions initiated by the SWI instruction nor during division by zero exceptions. The timing of EXCEPTION assertion for other exceptions differs according to the type of exception.

Exceptions are recognized at instruction boundaries. Time elapses between detection of the exception and the start of exception processing. A prefetch bus cycle for the next instruction is initiated during this period.

Because interrupts are recognized quickly, EXCEPTION is asserted during the prefetch bus cycle. The bus cycle is completed, and the prefetched word is overwritten when the pipeline is filled with interrupt handler instructions.

For exceptions other than interrupt, the prefetch bus cycle is completed before EXCEPTION is asserted. Assertion coincides with the first stacking operation. The prefetched word is overwritten when the pipeline is refilled with exception handler instructions.

10.1.3.6 INVALID — PHASE1/PHASE2 Signal Invalid

INVALID is always asserted during phase 2. INVALID assertion indicates that all non-null signals derived from PHASE1 must be ignored.

10.1.4 Combining Opcode Tracking with Other Capabilities

Pipeline state signals are useful during normal instruction execution and execution of exception handlers. Refer to **SECTION 9 EXCEPTION PROCESSING** for a detailed discussion of exceptions and exception handlers. The signals provide a complete model of the pipeline up to the point a breakpoint is acknowledged.

Breakpoints are acknowledged after an instruction has executed, when it is in pipeline stage C. A breakpoint can initiate either exception processing or background debugging mode. See**10.2 Breakpoints10.2 Breakpoints** and **10.3 Opcode Tracking and Breakpoints10.3 Opcode Tracking and Breakpoints**for more information. IPIPE0/IPIPE1 are not usable when the CPU16 is in background debugging mode. Complete information is contained in **10.4 Background Debug Mode (BDM)**.

10.1.5 CPU16 Instruction Pipeline State Signal Flow

Figure 10-3 is the flow diagram required to properly interpret instruction pipeline state signals.

10.2 Breakpoints

Breakpoints are set by internal assertion of the IMB $\overline{\text{BKPT}}$ signal or by external assertion of the microcontroller $\overline{\text{BKPT}}$ pin. The CPU16 supports breakpoints on any memory access. Acknowledged breakpoints can initiate either exception processing or background debugging mode. After BDM has been enabled, the CPU16 will enter BDM when either $\overline{\text{BKPT}}$ input is asserted.

If $\overline{\text{BKPT}}$ assertion is synchronized with an instruction prefetch, the instruction is "tagged" with the breakpoint when it enters the pipeline, and the breakpoint occurs after the instruction executes.

If $\overline{\text{BKPT}}$ assertion is synchronized with an operand fetch, breakpoint processing occurs at the end of the instruction during which $\overline{\text{BKPT}}$ is latched.

Breakpoints on instructions that are flushed from the pipeline before execution are not acknowledged, but operand breakpoints are always acknowledged. There is no breakpoint acknowledge bus cycle when BDM is entered. See **SECTION 9 EXCEPTION PROCESSING** for complete information about breakpoint exceptions.

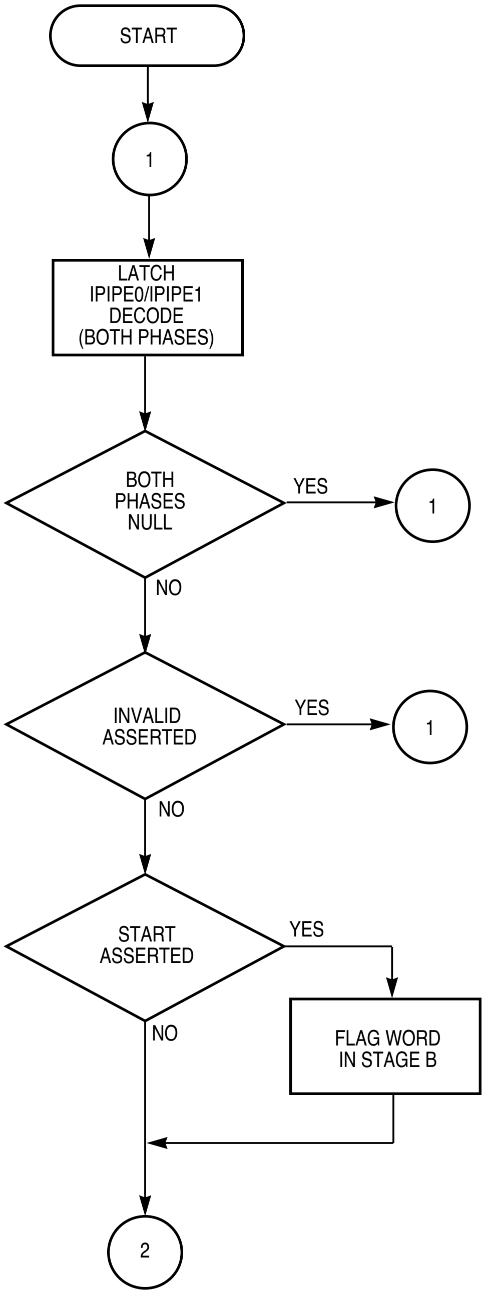


Figure 10-3 (Sheet 1 of 3) Instruction Pipeline Flow

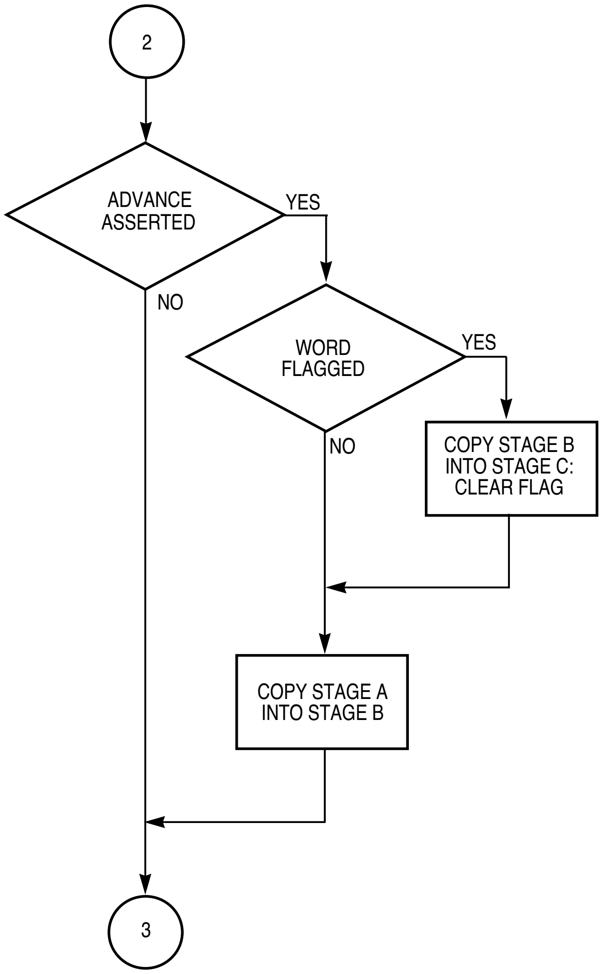

2
ADVANCE ASSERTED
YES
NO
WORD FLAGGED
YES
NO
COPY STAGE B INTO STAGE C: CLEAR FLAG
COPY STAGE A INTO STAGE B
3

Figure 10-3 (Sheet 2 of 3) Instruction Pipeline Flow

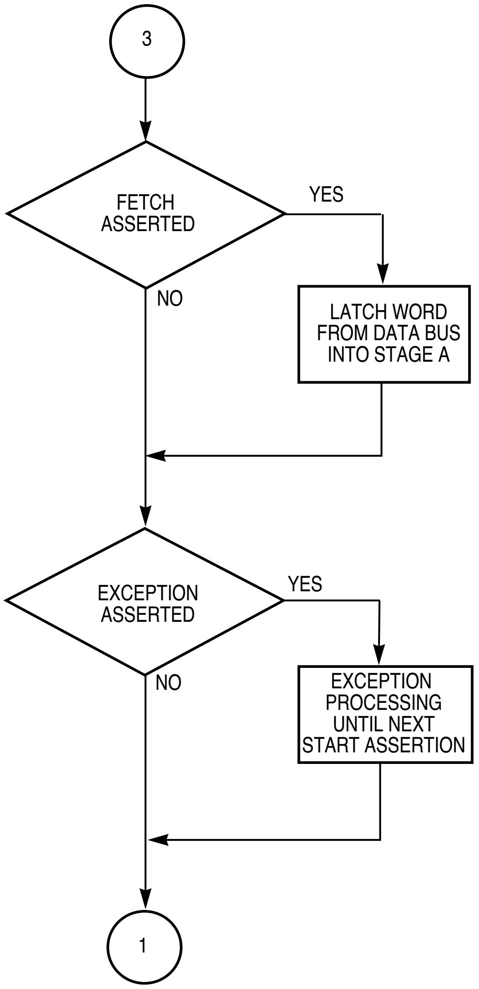

3
FETCH ASSERTED
YES
NO
LATCH WORD FROM DATA BUS INTO STAGE A
EXCEPTION ASSERTED
YES
NO
EXCEPTION PROCESSING UNTIL NEXT START ASSERTION
1

Figure 10-3 (Sheet 3 of 3) Instruction Pipeline Flow

10.3 Opcode Tracking and Breakpoints

Breakpoints are acknowledged after a tagged instruction has executed, when it is copied from pipeline stage B to stage C. At the time START is asserted for an instruction, stage C contains the opcode of the previous instruction.

When an instruction is tagged, IPIPE0/IPIPE1 show START and the appropriate number of ADVANCE and FETCH assertions for instruction execution before the breakpoint is acknowledged. If background debugging mode is enabled, these signals model the pipeline before BDM is entered.

10.4 Background Debug Mode (BDM)

Microprocessor debugging programs are generally implemented in external software. CPU16 BDM provides a debugger implemented in CPU microcode.

BDM incorporates a full set of debug options — registers can be viewed and altered, memory can be read or written, and test features can be invoked.

BDM also simplifies in-circuit emulation. In a common setup (**Figure 10-4**), emulator hardware replaces the target system processor. Communication between target system and emulator takes place via a complex interface.

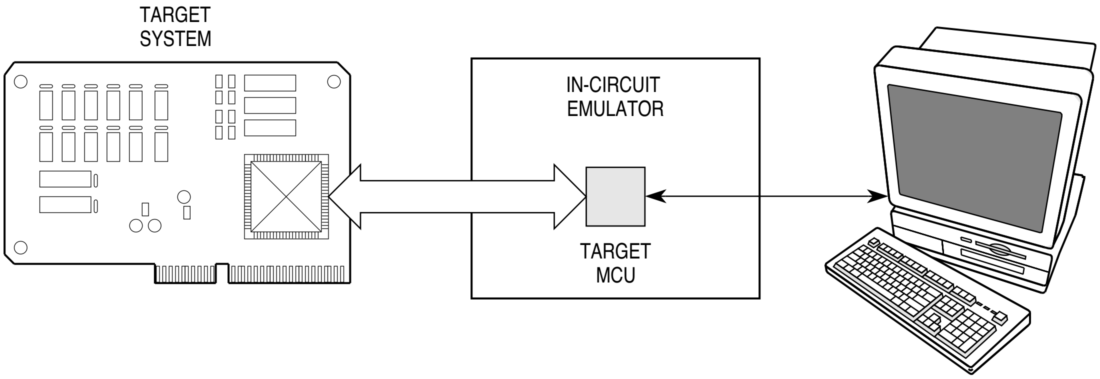


Figure 10-4 In-Circuit Emulator Configuration

CPU16 emulation requires a bus state analyzer only. The processor remains in the target system (see **Figure 10-5**) and the interface is less complex.

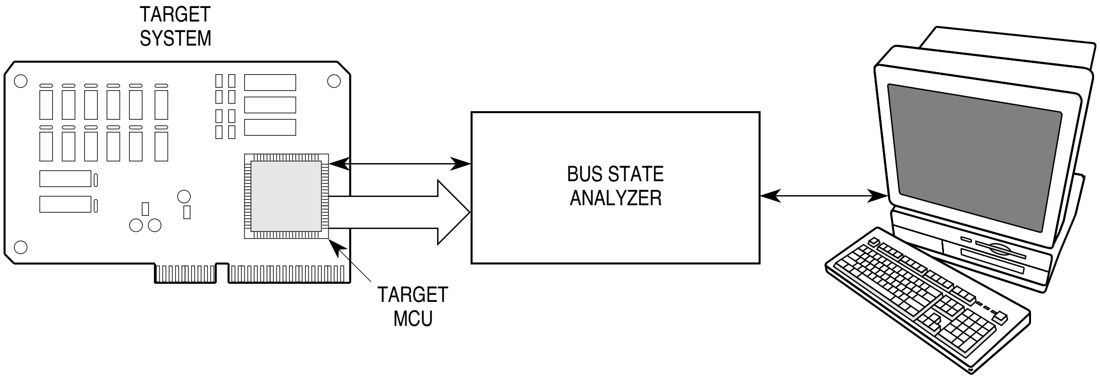


Figure 10-5 Bus State Analyzer Configuration

The analyzer monitors processor operation and the on-chip debugger controls the operating environment. Emulation is much “closer” to target hardware, and interfacing problems such as limited clock speed, AC and DC parametric mismatch, and restricted cable length are minimized.

BDM is an alternate CPU16 operating mode. During BDM, normal instruction execution is suspended, and special microcode performs debugging functions under external control.

BDM can be initiated by external assertion of the $\overline{\text{BKPT}}$ input, by internal assertion of the IMB $\overline{\text{BKPT}}$ signal, or by the BGND instruction. While in BDM, the CPU16 ceases to fetch instructions via the parallel bus and communicates with the development system via a dedicated serial interface.

10.4.1 Enabling BDM

The CPU16 samples the BKPT inputs during reset to determine whether to enable BDM. If either BKPT input is at logic level zero when sampled, an internal BDM enabled flag is set.

BDM operation is enabled when BKPT is asserted at the rising edge of the RESET signal. BDM remains enabled until the next system reset. If BKPT is at logic level one on the trailing edge of RESET, BDM is disabled. BKPT is relatched on each rising transition of RESET. BKPT is synchronized internally, and must be asserted for at least two clock cycles prior to negation of RESET.

BDM enable logic must be designed with special care. If BKPT hold time extends into the first bus cycle following reset, the bus cycle could inadvertently be tagged with a breakpoint. **Figure 10-6** shows a sample BDM enable circuit.

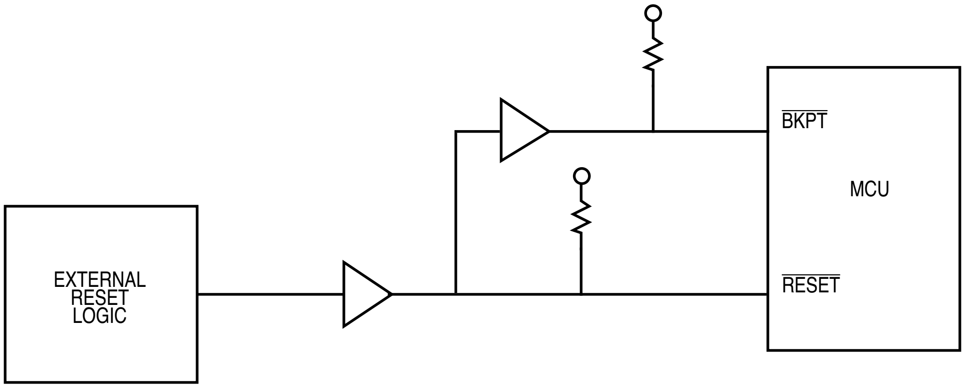


Figure 10-6 Sample BDM Enable Circuit

The microcontroller itself asserts RESET for 512 clock periods after it is released by external reset logic, and latches the state of BKPT on the rising edge of RESET at the end of this period. If enable circuitry only monitors the external reset, BKPT will not be enabled. **Figure 10-7** shows BDM enable timing. Refer to the appropriate modular microcontroller user's manual for specific timing information.

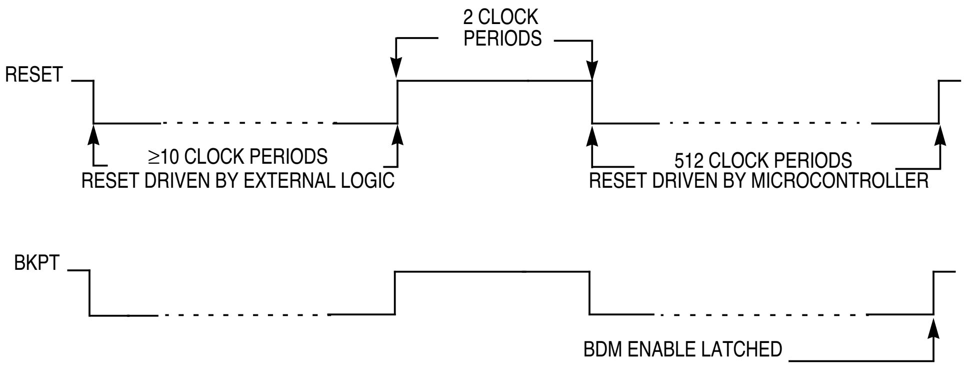


Figure 10-7 BDM Enable Waveforms

10.4.2 BDM Sources

When BDM is enabled, external breakpoint hardware, internal IMB module breakpoints, and the BGND instruction can cause the CPU16 to enter BDM. If BDM is not enabled when a breakpoint occurs, a breakpoint exception is processed. **Table 10-2** summarizes the processing of each source for both enabled and disabled cases.

Table 10-2 BDM Source Summary

Source	BDM Enabled	BDM Disabled
$\overline{\text{BKPT}}$	Background	Breakpoint Exception
BGND Instruction	Background	Illegal Instruction
Double Bus Fault	Background	Assert HALT

10.4.2.1 $\overline{\text{BKPT}}$ Signal

If enabled, BDM is initiated when assertion of $\overline{\text{BKPT}}$ is acknowledged. $\overline{\text{BKPT}}$ can be asserted on the IMB by another module in the microcontroller, or by taking the microcontroller $\overline{\text{BKPT}}$ pin low. There is no breakpoint acknowledge bus cycle when BDM is entered. See the appropriate microcontroller user's manual for more information concerning assertion of $\overline{\text{BKPT}}$.

10.4.2.2 BGND Instruction

If BDM has been enabled, executing BGND will cause the CPU16 to suspend normal operation and enter BDM. If BDM has not been correctly enabled, an illegal instruction exception is generated. Illegal instruction exceptions are discussed in **SECTION 9 EXCEPTION PROCESSING**.

10.4.2.3 Microcontroller Module Breakpoints

If BDM has been enabled, the CPU16 will enter BDM when other microcontroller modules assert the $\overline{\text{BKPT}}$ signal. Consult the appropriate microcontroller user's manual for a description of these capabilities.

10.4.2.4 Double Bus Fault

If BDM has been enabled, the CPU16 will enter BDM when a double bus fault is detected. If BDM has not been enabled, the HALT signal is asserted and processing stops.

10.4.3 BDM Signals

When BDM is entered, the $\overline{\text{BKPT}}$ and IPIPE signals change function and become BDM serial communication signals. The following table summarizes the changes.

Table 10-3 BDM Signals

State	Signal Name	Type	Description
No Background Mode	$\overline{\text{BKPT}}$ $\overline{\text{IPIPE0}}$ $\overline{\text{IPIPE1}}$	Input Output Output	Signals breakpoint to CPU16 Shows instruction pipeline state Shows instruction pipeline state
Background Mode	DSCLCK DSO DSI	Input Output Input	BDM serial clock BDM serial output BDM serial input

10.4.4 Entering BDM

When the processor detects a breakpoint or decodes a BGND instruction, it suspends instruction execution and asserts the FREEZE output. Once FREEZE has been asserted, the CPU enables the serial communication hardware and awaits a command.

Assertion of FREEZE causes opcode tracking signals IPIPE0 and IPIPE1 to change definition and become serial communication signals DSO and DSI. FREEZE is asserted at the next instruction boundary after $\overline{\text{BKPT}}$ is asserted. IPIPE0 and IPIPE1 change function before an EXCEPTION signal can be generated. The development system must use FREEZE assertion as an indication that BDM has been entered. When BDM is exited, FREEZE is negated prior to initiation of normal bus cycles — IPIPE0 and IPIPE1 will be valid when normal instruction prefetch begins.

10.4.5 Command Execution

Figure 10-8 summarizes BDM command execution. Commands consist of one 16-bit operation word and can include one or more 16-bit extension words. Each incoming word is read as it is assembled by the serial interface. The microcode routine corresponding to a command is executed as soon as the command is complete. Result operands are loaded into the output shift register to be shifted out as the next command is read. This process is repeated for each command until the CPU returns to normal operating mode.

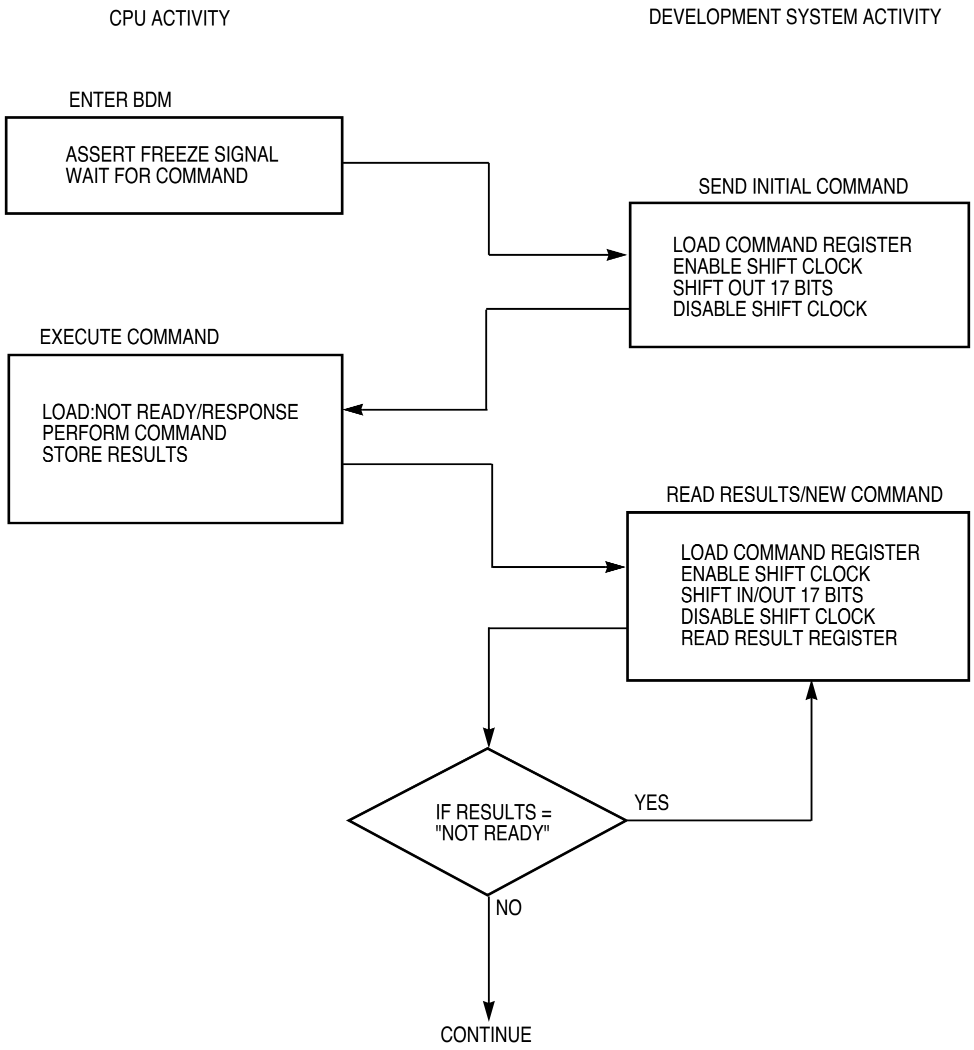


Figure 10-8 BDM Command Flow Diagram

10.4.6 Returning from BDM

BDM is terminated when a resume execution (GO) command is received. GO refills the instruction pipeline from address (PK: PC – $0006). FREEZE is negated prior to the first prefetch. Upon negation of FREEZE, the serial subsystem is disabled, and the DSO/DSI signals revert to IPIPE0/IPIPE1 functionality.

10.4.7 BDM Serial Interface

The serial interface uses a synchronous protocol similar to that of the Motorola Serial Peripheral Interface (SPI). **Figure 10-9** is a development system serial logic diagram.

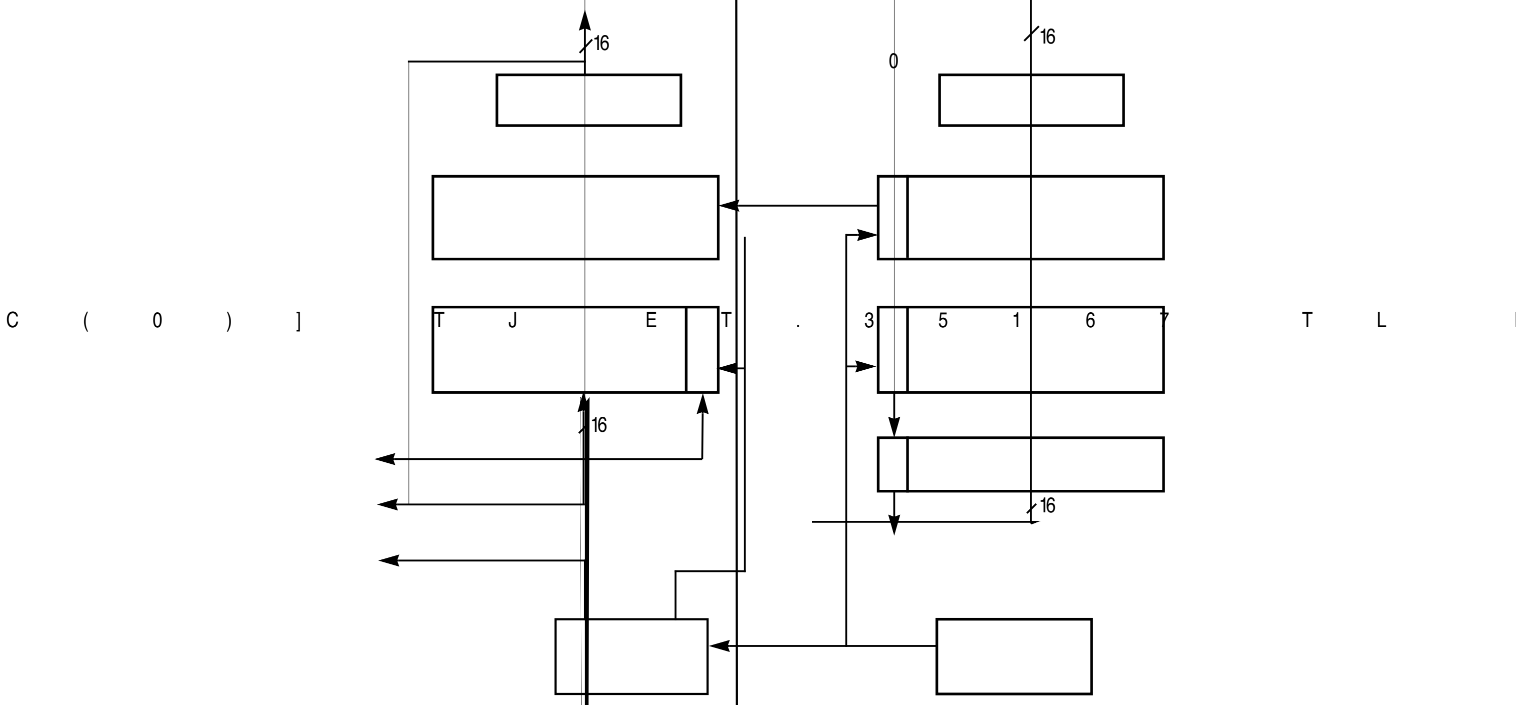


Figure 10-9 BDM Serial I/O Block Diagram

The development system serves as the master of the serial link, and is responsible for the generation of serial interface clock signal DSCLK.

Serial clock frequency range is from DC to one-half the CPU16 clock frequency. If DSCLK is derived from the CPU16 system clock, development system serial logic can be synchronized with the target processor.

The serial interface operates in full-duplex mode. Data transfers occur on the falling edge of DSCLK and are stable by the following rising edge of DSCLK. Data is transmitted MSB first, and is latched on the rising edge of DSCLK.

The serial data word is 17 bits wide — 16 data bits and a status/control bit.

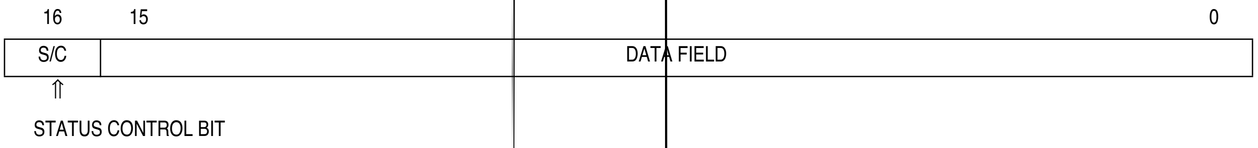


Figure 10-10 Serial Data Word Format

Bit 16 indicates status of CPU-generated messages as shown in **Table 10-4**.

Table 10-4 CPU Generated Message Encoding

Bit 16	Data	Message Type
0	xxxx	Valid Data Transfer
0	FFFF	Command Complete; Status OK
1	0000	Not Ready with Response; Come Again
1	FFFF	Illegal Command

Command and data transfers initiated by the development system must clear bit 16. All commands that return a result return 16 bits of data plus one status bit.

10.4.7.1 CPU Serial Logic

CPU16 serial logic, shown in the left-hand portion of **Figure 10-9**, consists of transmit and receive shift registers and of control logic that includes synchronization, serial clock generation circuitry, and a received bit counter.

Both DSCLK and DSI are synchronized to internal clocks. Data is sampled during the high phase of CLKOUT. At the falling edge of CLKOUT, the sampled value is made available to internal logic. If there is no synchronization between CPU16 and development system hardware, the minimum hold time on DSI with respect to DSCLK is one full period of CLKOUT.

Serial transfer is based on the DSCLK signal (see **Figure 10-11**). At the rising edge of the internal synchronized DSCLK, synchronized data is transferred to the input shift register, and the received bit counter is decremented. One-half clock period later, the output shift register is updated, bringing the next output bit to the DSO signal. DSO changes relative to the rising edge of DSCLK and does not necessarily remain stable until the falling edge of DSCLK.

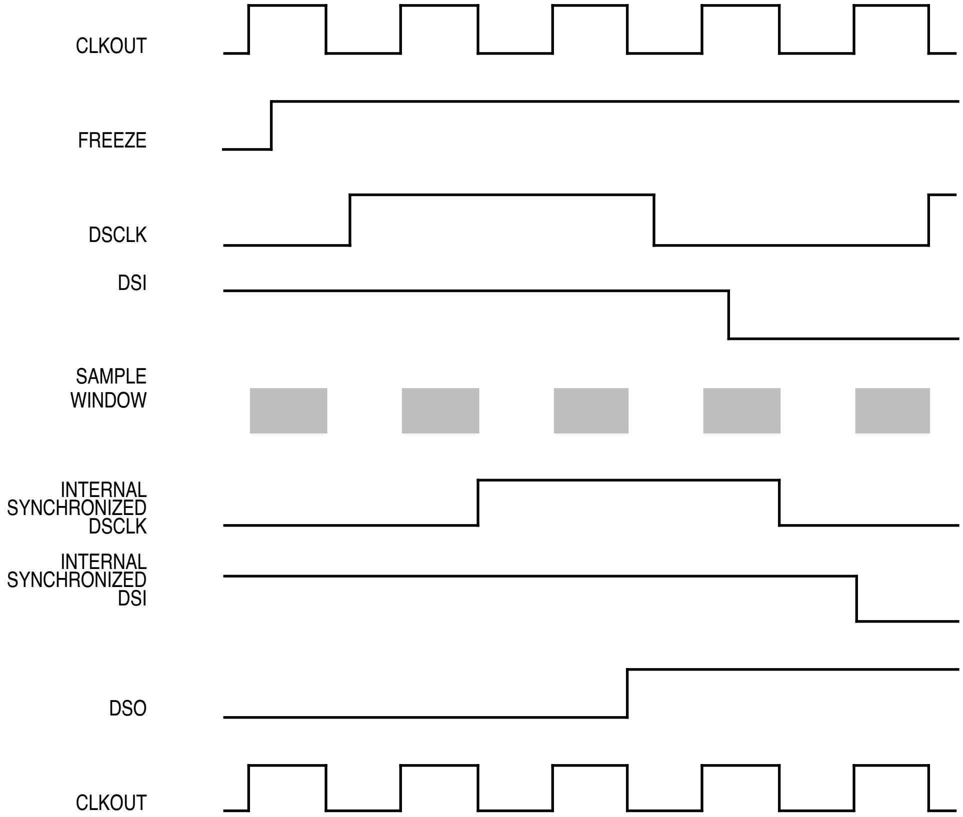


Figure 10-11 Serial Interface Timing Diagram

One full clock period after the rising edge of DSCLK, the updated counter value is checked. If the counter has reached zero, the receive data latch is updated from the input shift register. At the same time, the output shift register is reloaded with a “not ready/come again” response. When the receive data latch is loaded, the CPU is released to act on the new data. Response data overwrites “not ready” when the CPU has completed the current operation.

Data written into the output shift register appears immediately on the DSO signal. In general, this action changes the state of the signal from logic level one (“not ready”) to logic level zero (valid data). However, this level change only occurs if the transfer is completed. Error conditions cause the “not ready” status bit to be overwritten.

The DSO state change can be used to signal interface hardware that the next serial transfer may begin. A time-out of sufficient length to trap error conditions that do not change the state of DSO must be incorporated into the design. Hardware interlocks in the CPU prevent result data from corrupting serial transfers in progress.

10.4.7.2 Development System Serial Logic

The development system must initiate BDM and supply the BDM serial clock. Serial logic must be designed so that these functions do not affect one another.

Breakpoint requests are made by asserting $\overline{\text{BKPT}}$ in either of two ways. The preferred method is to assert $\overline{\text{BKPT}}$ during the bus cycle for which an exception is desired. The second method is to assert $\overline{\text{BKPT}}$ until the CPU16 responds by asserting FREEZE. This method is useful for forcing a transition into BDM when the bus is not being monitored. Both methods require logic that precludes spurious serial clocks.

Figure 10-12 shows timing for $\overline{\text{BKPT}}$ assertion during a single bus cycle. **Figure 10-13** shows $\overline{\text{BKPT}}$/FREEZE timing. In both cases, the serial clock output is left high after the final shift of each transfer. This prevents tagging the prefetch initiated when BDM terminates.

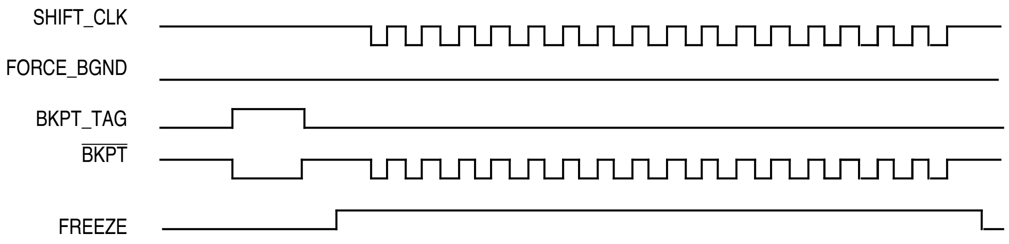


Figure 10-12 $\overline{\text{BKPT}}$ Timing for Single Bus Cycle

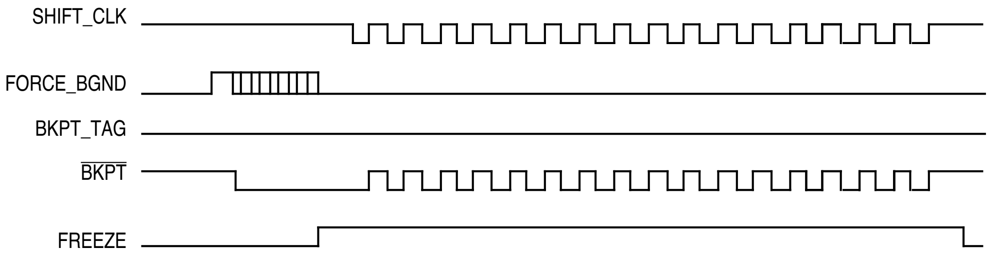


Figure 10-13 $\overline{\text{BKPT}}$ Timing for Forcing BDM

Figure 10-14 shows a sample circuit that accommodates either method of $\overline{\text{BKPT}}$ assertion. FORCE_BGND can either be pulsed or remain asserted until FREEZE is asserted. Once FORCE_BGND is asserted, the set-reset latch holds $\overline{\text{BKPT}}$ low until the first SHIFT_CLK is applied.

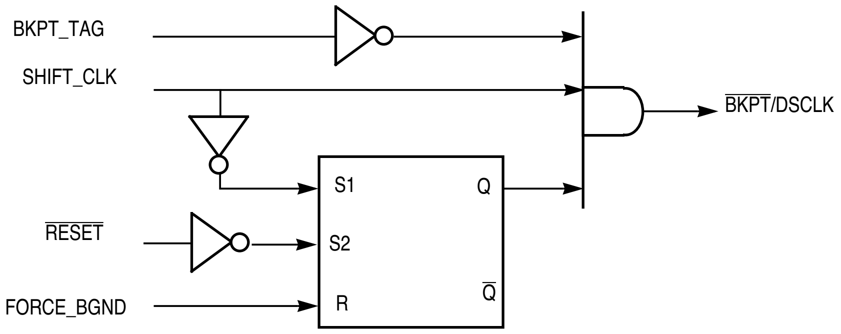


Figure 10-14 $\overline{\text{BKPT}}$/DSCLK Logic Diagram

Since it is not latched, BKPT_TAG must be synchronized with CPU16 bus cycles. If negation of BKPT_TAG extends past FREEZE assertion, the CPU16 will clock on it as though it were the first DSCLK pulse.

DSCLK is the gated serial clock. Normally high, it pulses low for each bit transferred. At the end of the seventeenth clock period, it remains high until the start of the next transmission. Clock frequency is implementation dependent and may range from dc to the maximum specified frequency.

10.4.8 BDM Command Format

The following standard bit format is utilized by all BDM commands.

15   0
OPERATION WORD
EXTENSION WORD(S)

Operation Word

All commands have a unique 16-bit operation word. No command requires an extension word to specify the operation to be performed.

Extension Words

Some commands require extension words for addresses or immediate data. Addresses require two extension words to accommodate 20 bits. Immediate data can be either one or two words in length — byte and word data each require a single extension word, long-word data requires two words. Both operands and addresses are transferred most significant word first.

10.4.9 Command Sequence Diagram

A command sequence diagram illustrates the serial bus traffic for each command. Each bubble in the diagram represents a single 17-bit transfer across the bus. The top half of each bubble shows data sent from the development system to the CPU16. The bottom half shows data returned by the CPU16 in response to commands. Transmissions overlap to minimize latency.

Figure 10-15 shows an example command sequence diagram. A description of the information in the diagram follows.

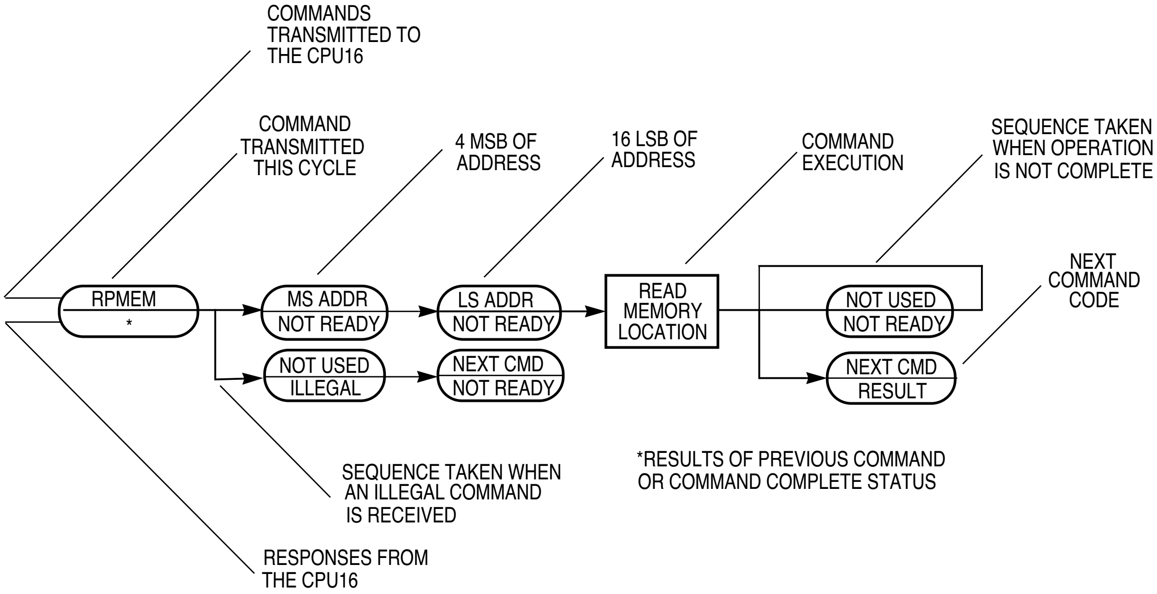

COMMANDS TRANSMITTED TO THE CPU16
COMMAND TRANSMITTED THIS CYCLE
4 MSB OF ADDRESS
16 LSB OF ADDRESS
COMMAND EXECUTION
SEQUENCE TAKEN WHEN OPERATION IS NOT COMPLETE
RPMEM
*
MS ADDR
NOT READY
LS ADDR
NOT READY
READ MEMORY LOCATION
NOT USED
NOT READY
NEXT COMMAND CODE
NOT USED
ILLEGAL
NEXT CMD
NOT READY
NEXT CMD
RESULT
SEQUENCE TAKEN WHEN AN ILLEGAL COMMAND IS RECEIVED
*RESULTS OF PREVIOUS COMMAND OR COMMAND COMPLETE STATUS
RESPONSES FROM THE CPU16

Figure 10-15 Command Sequence Diagram Example

The cycle in which the command is issued contains the command word (RPMEM). During the same cycle, the CPU16 responds with either the low order results of the previous command or with a command complete status if no results were required.

During the second cycle, the development system supplies the 4 high-order bits of a memory address. The CPU16 returns a NOT READY response unless the received command was decoded as unimplemented, in which case the response is the ILLEGAL command encoding. When an ILLEGAL response occurs, the development system must retransmit the command.

In the third cycle, the development system supplies the 16 low-order bits of the memory address. The CPU16 always returns a NOT READY response in this cycle. At the completion of the third cycle, the CPU16 initiates a memory read operation. Any serial transfers that begin while the memory access is in progress return the NOT READY response.

Results are returned in the serial transfer cycle following completion of the memory access. If the serial clock is slow, there may be additional NOT READY responses from the CPU16. The data transmitted to the CPU during the final transfer is the next command word.

10.4.10 BDM Command Set

The BDM command set is summarized in **Table 10-5**. Subsequent pages contain a BDM command glossary. Glossary entries are in the same order as the table. Each entry contains detailed information concerning commands and results, and includes a command sequence diagram.

Table 10-5 Command Summary

Command	Mnemonic	Description
Read Registers from Mask	RREGM	Read contents of registers specified by command word register mask
Write Registers from Mask	WREGM	Write to registers specified by command word register mask
Read MAC Registers	RDMAC	Read contents of entire multiply and accumulate register set
Write MAC Registers	WRMAC	Write to entire multiply and accumulate register set
Read PC and SP	RPCSP	Read contents of program counter and stack pointer
Write PC and SP	WPCSP	Write to program counter and stack pointer
Read Data Memory	RDMEM	Read data from specified 20-bit address in data space
Write Data Memory	WDMEM	Write data to specified 20-bit address in data space
Read Program Memory	RPMEM	Read data from specified 20-bit address in program space
Write Program Memory	WPMEM	Write data to specified 20-bit address in program space
Execute from current PK: PC	GO	Instruction pipeline flushed and refilled; instructions executed from current PC – $0006
Null Operation	NOP	Null command — performs no operation

10.4.10.1 BDM Memory Commands and Bus Errors

If a bus error occurs while a BDM command that accesses memory (RDMEM, WDMEM, RPMEM, or WPMEM) is executing, it is ignored by the CPU16. Data returned by a read access during which a bus error occurs is indeterminate.

RREGM Read Registers From Mask RREGM

Description: Registers specified by a register mask operand are read and returned via the serial link.

Operand: A 7-bit mask operand is right-justified in an operand word. Registers are specified as follows:

Bit 0: Condition Code Register [15:4]
Bit 1: Address Extension (K) Register
Bit 2: Index Register Z
Bit 3: Index Register Y
Bit 4: Index Register X
Bit 5: Accumulator E
Bit 6: Accumulator D

Registers are received in order from bit 0 to bit 6.

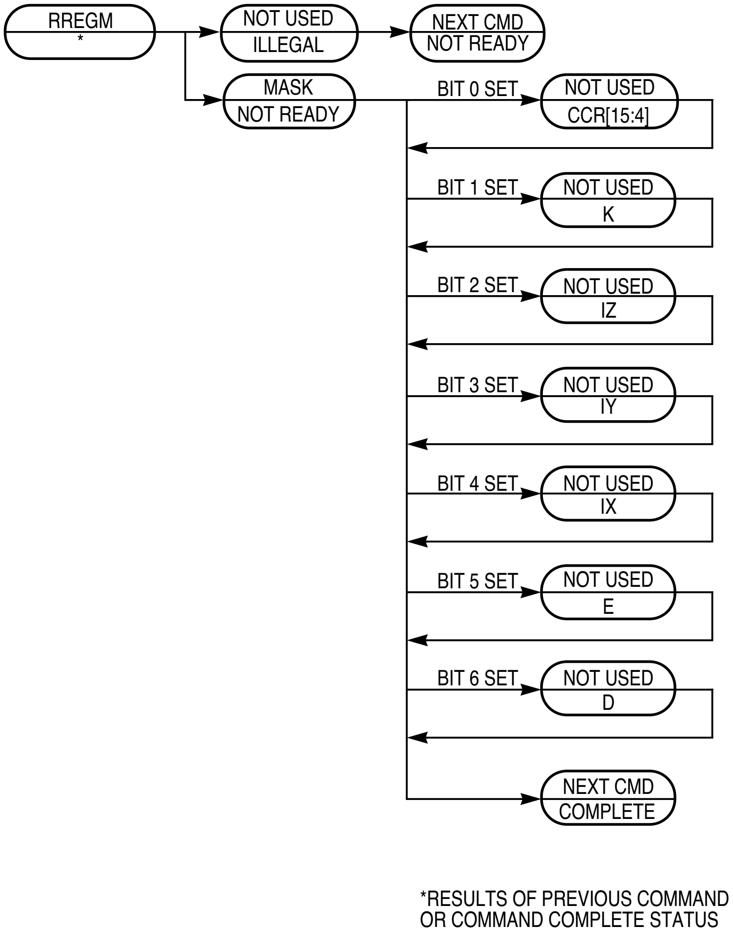
Result: A 16-bit word for each register specified. Register content is returned MSB first. Command complete status ($FFFF) is returned after the last register has been returned.

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	1	0	0	0	0	0	0	0
NOT USED									MASK						

RREGM Read Registers From Mask RREGM

Command Sequence Diagram:



WREGM Write Registers From Mask WREGM

Description: Registers specified by a register mask operand are written with data received via the serial link.

Operand: A 7-bit mask operand is right-justified in an operand word. Registers are specified as follows:

Bit 0: Condition Code Register [15:4]
Bit 1: Address Extension (K) Register
Bit 2: Index Register Z
Bit 3: Index Register Y
Bit 4: Index Register X
Bit 5: Accumulator E
Bit 6: Accumulator D

Registers are written in order from bit 0 to bit 6.

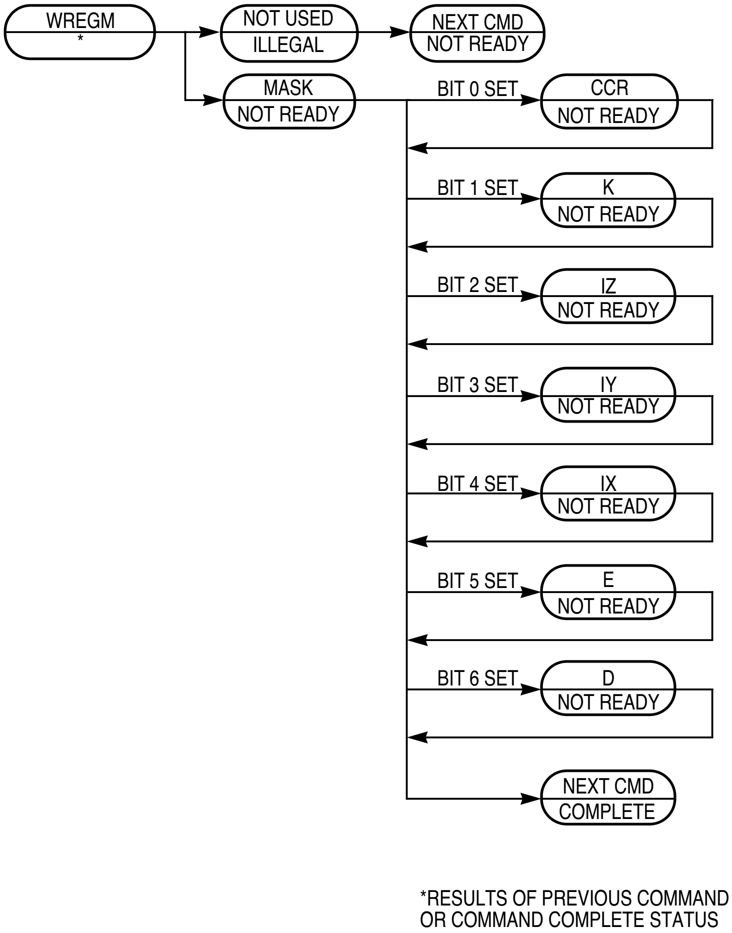
Result: A 16-bit word for each register specified. Register content is returned MSB first. Command complete status ($FFFF) is returned after the last register has been written.

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	1	0	0	0	0	0	0	1
NOT USED									MASK						

WREGM Write Registers From Mask WREGM

Command Sequence Diagram:



RDMAC Read MAC Register Set RDMAC

Description: The entire multiply and accumulate register set is read and returned via the serial link.

Operand: None

Result: A 16-bit word for each register. Register content is returned MSB first in the following order:

H Register
I Register
AM[15:0]
AM[31:16]
SL and AM[35:32]
XM: YM

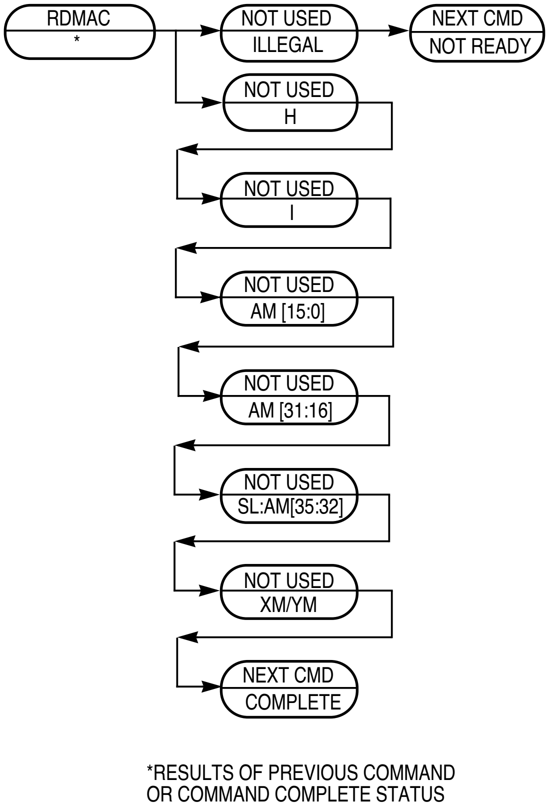
DSP sign latch bit SL is returned in bit 15 of a result word, AM[35:32] are returned in bits [3:0] of the same word, and bits [14:4] are undefined.
Command complete status ($FFFF) is returned after the last register value has been returned.

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	1	0	0	0	1	0	1	0

RDMAC Read MAC Register Set RDMAC

Command Sequence Diagram:



WRMAC Write MAC Register Set WRMAC

Description: The entire multiply and accumulate register set is written with data received via the serial link.

Operand: A 16-bit word for each register is received (MSB first) via the serial link. Words are read and written in the following order:

XM: YM
SL and AM[35:32]
AM[31:16]
AM[15:0]
I Register
H Register

DSP sign latch bit SL must be bit 15 of an operand, AM[35:32] must be bits [3:0] of the same word, and bits [14:4] can be undefined.

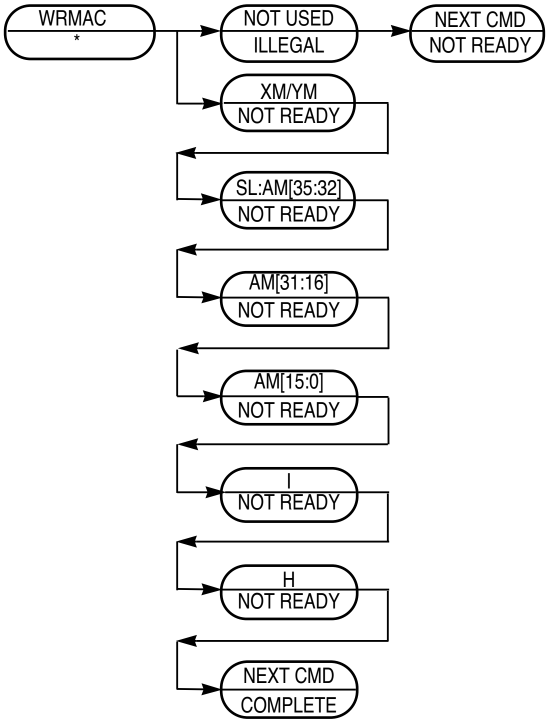
Result: Command complete status ($FFFF) is returned after the last register is written.

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	1	0	0	0	1	0	1	1

WRMAC Write MAC Register Set WRMAC

Command Sequence Diagram:



*RESULTS OF PREVIOUS COMMAND
OR COMMAND COMPLETE STATUS

RPCSP Read PC and SP RPCSP

Description: Program counter and stack pointer are read, then transmitted via the serial link.

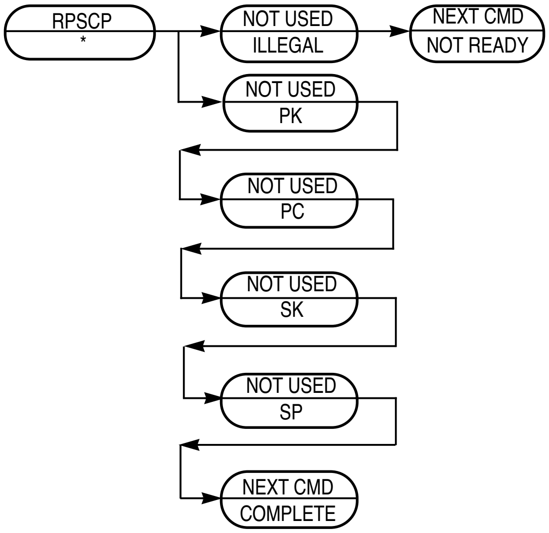
Operand: None

Result: Four words are returned MSB first in the following order:
PK extension field and PCSK extension field and SP
PK and SK are contained in bits [3:0] of the respective result words. Bits [15:4] of the words are undefined.
Command complete status ($FFFF) is returned after the last register is returned.

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	1	0	0	0	0	0	1	0

Command Sequence Diagram:



*RESULTS OF PREVIOUS COMMAND
OR COMMAND COMPLETE STATUS

WPCSP Write PC and SP WPCSP

Description: Program counter and stack pointer are written with data received via the serial link.

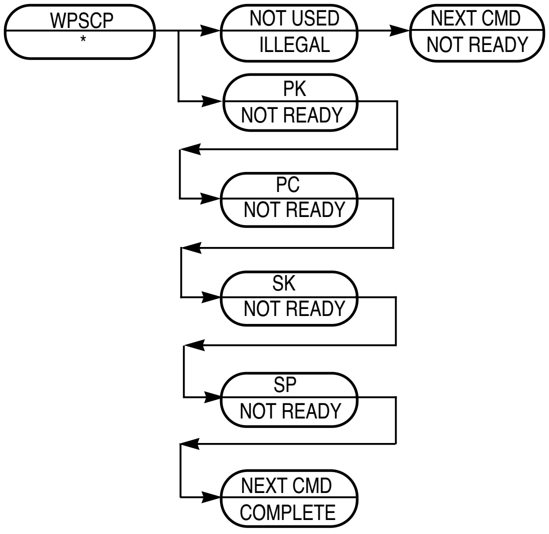
Operand: Registers are received and written in the following order:
PK extension field and PCSK extension field and SP
PK and SK are contained in bits [3:0] of the respective operand words. Bits [15:4] of the words are undefined.

Result: Command complete status ($FFFF) is returned after the last register is written.

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	1	0	0	0	0	0	1	1

Command Sequence Diagram:



*RESULTS OF PREVIOUS COMMAND OR COMMAND COMPLETE STATUS

RDMEM Read Data Space Memory RDMEM

Description: A byte, word, or long word is read from an address in data space and transmitted via the serial link.

Operand: Two extension words specify 20-bit memory address and operand size. Bits [3:0] of the first word are the bank address. Bits [15:14] are encoded to specify operand size. Bits [13:4] are reserved for future use. The second word is the operand address.

Table 10-6 Operand Size Encoding

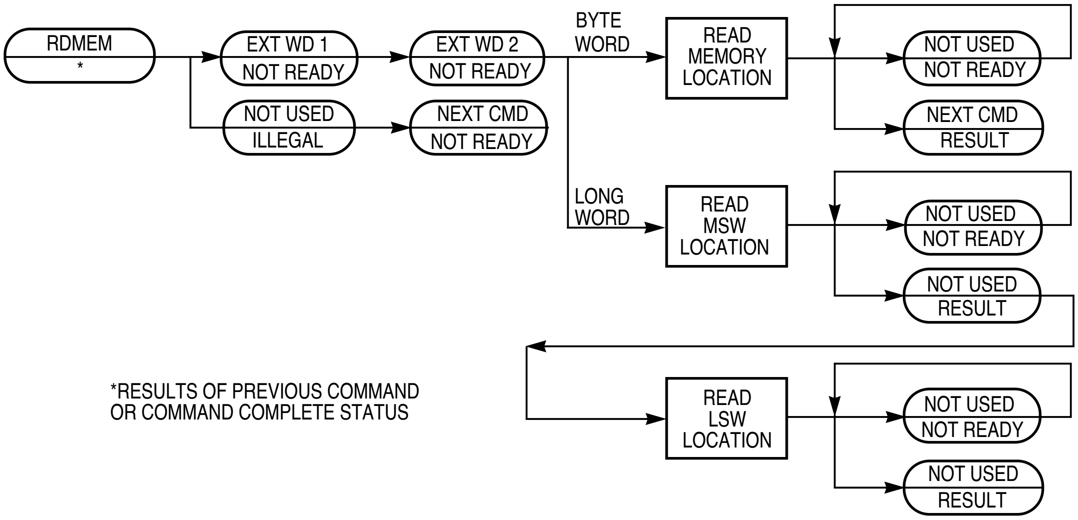
Bits [15:14]	Operand Size
00	Byte
01	Word
1X	Long Word

Result: Eight, 16, and 32-bit data. Eight and 16-bit data are transmitted as 16-bit data words, MSB first. For 8-bit data, the upper byte of each word contains $FF. 32-bit data is transmitted as two 16-bit data words in MSW, LSW order beginning with the MSB of each word.

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	1	0	0	0	0	1	0	0

Command Sequence Diagram:



WDMEM Write Data Space Memory WDMEM

Description: A byte, word, or long word is received via the serial link and written to an address in data space.

Operand: Two extension words specify 20-bit memory address and operand size. Third and fourth (long word operands only) words contain data to be written. Bits [3:0] of the first word are the bank address. Bits [15:14] are encoded to specify operand size. Bits [13:4] are reserved for future use. The second word is the operand address. When byte data is written, the upper byte of the third extension word is not used — these bits are reserved for future use.

Table 10-7 Operand Size Encoding

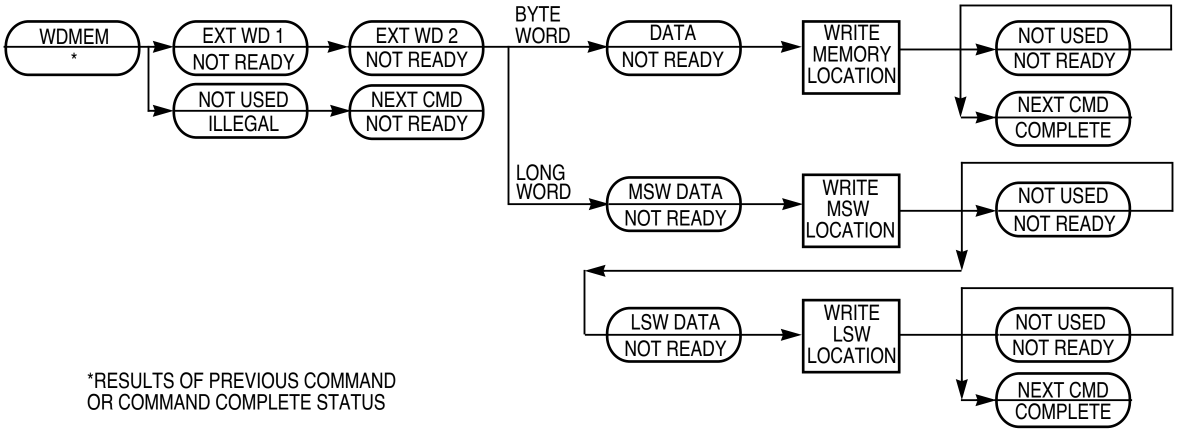
Bits [15:14]	Operand Size
00	Byte
01	Word
1X	Long Word

Result: Command complete status ($FFFF) is returned after memory is written.

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	1	0	0	0	0	1	0	1

Command Sequence Diagram:



RPMEM Read Program Space Memory RPMEM

Description: A 16-bit memory word is read from an address in program space and transmitted via the serial link.

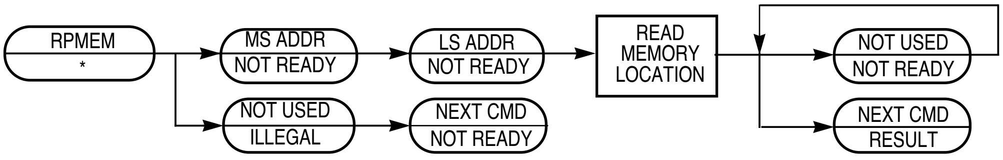
Operand: Two extension words specify the 20-bit memory address. Bits [3:0] of the first word are the bank address (bits [15:4] are undefined). The second word is the word address. A word address must be even — misaligned program space reads are not allowed — address LSB is cleared before the read.

Result: 16-bit data word, transmitted MSB first.

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	1	0	0	0	0	1	1	0

Command Sequence Diagram:



*RESULTS OF PREVIOUS COMMAND
OR COMMAND COMPLETE STATUS

WPMEM Write Program Space Memory WPMEM

Description: A 16-bit memory word is received via the serial link and written to an address in program space.

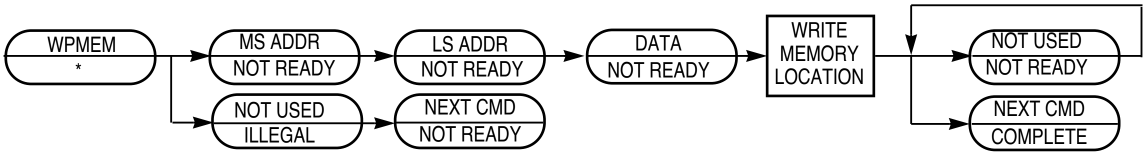
Operand: Two extension words specify the 20-bit memory address, and a third word contains the data to be written. Bits [3:0] of the first word are the bank address (bits [15:4] are undefined). The second word is the word address. A word address must be even — misaligned program space writes are not allowed — address LSB is cleared before the read.

Result: Command complete status ($FFFF) is returned after memory is written.

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	1	0	0	0	0	1	1	1

Command Sequence Diagram:



*RESULTS OF PREVIOUS COMMAND

GO Execute Instructions From Current PK: PC GO

Description: Background debugging mode is exited, the pipeline is flushed and refilled, then the CPU16 resumes normal execution of instructions at PK: PC – $0006. PK and PC retain the values they had when BDM began unless altered by a WPCSP command.

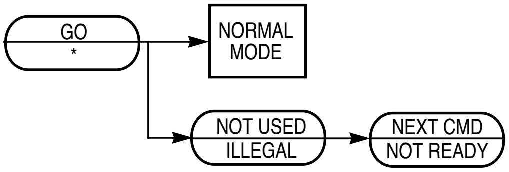
Operand: None

Result: None

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	1	0	0	0	1	0	0	0

Command Sequence Diagram:



*RESULTS OF PREVIOUS COMMAND
OR COMMAND COMPLETE STATUS

NOP Null Operation NOP

Description: A command is transmitted, but no operation is performed.

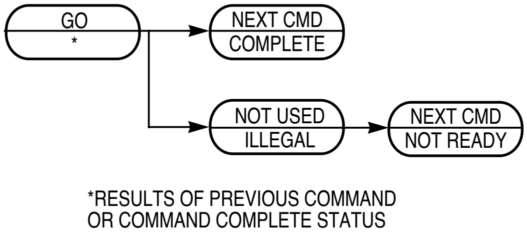
Operand: None

Result: Command complete status ($FFFF) is returned.

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	1	0	0	0	1	0	0	1

Command Sequence Diagram:



10.4.11 Future Commands

Unassigned command opcodes are reserved by Motorola for future expansion. All unused formats within any revision level will perform a NOP and return the ILLEGAL command response.

10.4.12 Recommended BDM Connection

In order to provide for use of development tools when an MCU is installed in a system, Motorola recommends that appropriate signal lines be routed to a male Berg connector or double-row header installed on the circuit board with the MCU, as shown in the following figure.

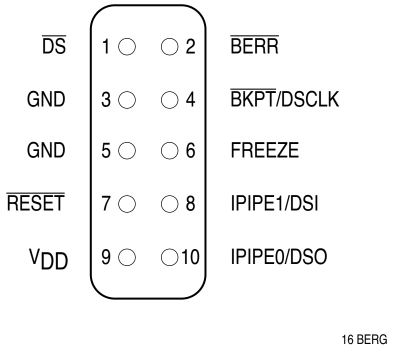


Figure 10-16 BDM Connector Pinout

SECTION 11 DIGITAL SIGNAL PROCESSING

This section contains detailed information about CPU16 digital signal processing (DSP) capabilities. A comprehensive presentation of signal processing theory is beyond the scope of this manual — discussion is limited to CPU16 hardware and instructions that support control-oriented DSP.

11.1 General

The CPU16 performs low frequency digital signal processing algorithms in real time. The most common DSP operation in embedded control applications is filtering, but the CPU16 can perform several other useful DSP functions. These include autocorrelation (detecting a periodic signal in the presence of noise), cross-correlation (determining the presence of a defined periodic signal), and closed-loop control routines (selective filtration in a feedback path).

Although derivation of DSP algorithms is often a complex mathematic task, the algorithms themselves typically consist of a series of multiply and accumulate (MAC) operations. The CPU16 contains a dedicated set of registers that are used to perform MAC operations. These are collectively called the MAC unit.

DSP operations generally require a large number of MAC iterations. The CPU16 instruction set includes instructions that perform MAC setup and repetitive MAC operations. Other instructions, such as 32-bit load and store instructions, can also be used in DSP routines.

Many DSP algorithms require extensive data address manipulation. To increase throughput, the CPU16 performs effective address calculations and data prefetches during MAC operations. In addition, the MAC unit provides modulo addressing to efficiently implement circular DSP buffers.

11.2 Digital Signal Processing Hardware

The MAC unit consists of a 16-bit multiplicand register (IR), a 16-bit multiplier register (HR), a 36-bit accumulator (AM), and two 8-bit address mask registers (XMSK and YMSK). **Figure 11-1** is a programmer's model of the MAC unit.

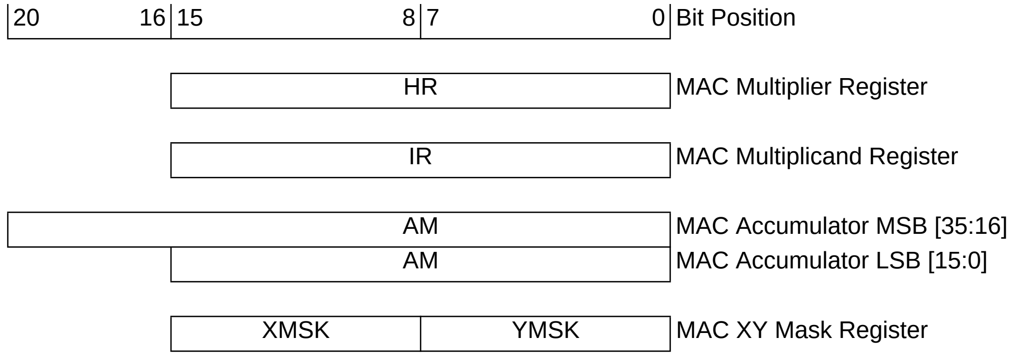


Figure 11-1 MAC Unit Register Model

11.3 Modulo Addressing

The MAC unit uses a simplified form of modulo addressing to implement finite impulse response filters and circular buffers during execution of MAC and RMAC instructions. It is accomplished by means of address masks.

During execution of MAC and RMAC, an offset is added to the content of IX and IY to compute the effective address of data accesses. XMSK and YMSK are used to determine which bits change when an offset is added.

Each address mask consists of eight bits. Each bit in the mask corresponds to a bit in the low byte of an index register. When a mask bit is set, the corresponding index register bit is changed by addition of the offset. This permits modulo addressing on any power of two boundary from 2^1 to 2^8. The possible buffer sizes are 2, 4, 8, 16, 32, 64, 128, and 256 bytes.

To enable a buffer, set the mask bits corresponding to a particular power of two. All set bits must be right-justified within the mask. For example, a mask value of $00011111 ($2^5$) enables a 32-byte buffer, while a mask value of $00001111 ($2^4$) enables a 16-byte buffer. If all set bits in the mask are not right-justified, results of the masking operation are undefined. Clear the masks to disable modulo addressing.

Modulo addressing cannot cross bank boundaries. Buffers must be within the bank specified by the current index register extension field (XK or YK).

11.4 MAC Data Types

Multiplicand and multiplier operands are 16-bit fractions. Bit 15 is the sign bit. An implied radix point lies between bits 15 and 14. There are 15 bits of magnitude. The range of values is –1 ($8000) to $1 – 2^{-15}$ ($7FFF).

The product of a MAC multiplication is a 32-bit signed fraction. Bit 31 is the sign bit. An implied radix point lies between bits 31 and 30. There are 31 bits of magnitude, but bit 0 is always cleared. The range of values is –1 ($80000000) to $1 – 2^{-30}$ ($7FFFFFFE).

The MAC accumulator uses 36-bit signed mixed numbers. The accumulator contains 36 bits. Bit 35 is the sign bit. Bits [34:31] are extension bits. Bits [30:0] are a 31-bit fixed-point fraction. There is an implied radix point between bits 31 and 30. There are 31 bits of magnitude, but use of the sign and extension bits allows representation of numbers in the range –16 ($800000000) to 15.999999999 ($7FFFFFFFF).

Figure 11-2 shows fractional data types and weighting of bits. Notice that signed fractions and signed mixed numbers can be interpreted as different arithmetic values when the same bits in the numbers are set.

15															0
	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}
±	⇐ (Radix Point) 16-BIT SIGNED FRACTION														

31															16
	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}
±	⇐ (Radix Point) MSW 32-BIT SIGNED FRACTION 1														

15															0
2^{-16}	2^{-17}	2^{-18}	2^{-19}	2^{-20}	2^{-21}	2^{-22}	2^{-23}	2^{-24}	2^{-25}	2^{-26}	2^{-27}	2^{-28}	2^{-29}	2^{-30}	2^{-31}
LSW 32-BIT SIGNED FRACTION 1															0

35			32	31															16
	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}
±	«	«	«	«	⇐ (Radix Point) MSW 32-BIT SIGNED FRACTION														

15															0
2^{-16}	2^{-17}	2^{-18}	2^{-19}	2^{-20}	2^{-21}	2^{-22}	2^{-23}	2^{-24}	2^{-25}	2^{-26}	2^{-27}	2^{-28}	2^{-29}	2^{-30}	2^{-31}
LSW 32-BIT SIGNED FRACTION															

Figure 11-2 MAC Data Types

11.5 MAC Accumulator Overflow

It is possible to accumulate to the point of overflow during successive and iterative multiply and accumulate operations. Overflow becomes important when the 36-bit number in AM is transferred to accumulator E by a TMER or TMET instruction. The 16-bit fraction in E does not have as great a range of values as the 36-bit number in AM. Two types of overflow detection are used.

11.5.1 Extension Bit Overflow

Extension bit overflow occurs when successive accumulation causes overflow into AM[34:31]. Although an overflow has occurred, sign and magnitude are still represented in 36 bits. Accumulator content cannot be directly converted into a 16-bit fraction, but it is possible to recover from extension bit overflow during subsequent multiply and accumulate operations.

A check for overflow into AM[34:31] is performed at the end of MAC, TMER, ACED, ASLM, and ACE instructions, and after each iteration of the RMAC instruction. When overflow has occurred, the EV bit in the CPU16 condition code register is set. **Table 11-1** shows the range of AM values and the effects of extension bit overflow. Bit values are binary.

Table 11-1 AM Values and Effect on EV

AM Magnitude	AM35	AM[34:31]	EV
$1 \le AM \le 15.999999999$	0	0001 — 1111	1
$0 \le AM < 1$	0	0000	0
$-1 \le AM < 0$	1	1111	0
$-16 \le AM < -1$	1	0000 — 1110	1

EV is set when extension bit overflow occurs, but will be cleared when a subsequent accumulation produces a value within the acceptable range.

Note

The RMAC instruction can be interrupted and restarted. Interrupt service routines which include branches based on EV status must be carefully designed.

11.5.2 Sign Bit Overflow

Sign bit overflow occurs when successive accumulation causes AM35 to be overwritten. The sign of the number in AM is lost. It is no longer accurately represented in 36 bits and accurate conversion to a 16-bit value is impossible.

A check for overflow into AM35 is performed at the end of MAC, TMER, ACED, ASLM, and ACE instructions, and after each iteration of the RMAC instruction. When overflow has occurred, the MV bit in the CPU16 condition code register is set. Since sign bit overflow can only occur after bits [34:31] have been overwritten, the EV bit must also be set.

The value of AM35 is latched when MV is set. The latched bit, called the sign latch (SL), shows the sign of AM immediately after overflow, and is therefore the complement of the value in AM35 at the time of overflow. SL is stacked by the PSHM instruction.

Even when a subsequent accumulation produces a value within the acceptable range, and EV is cleared, MV remains set until cleared by an ANDP, CLRM, TAP, TDP, TEM, or TEDM instruction. The SL value remains latched until the first sign bit overflow after MV has been cleared.

11.6 Data Saturation

The CPU16 can simulate the effect of saturation in analog systems. Saturation mode is enabled by setting the SM bit in the condition code register. If saturation mode is enabled, a saturation value will be written to accumulator E when either of the TMER or TMET instructions is executed while EV or MV is set. Saturation mode operation does not affect the content of AM.

$7FFF is the positive saturation value; $8000 is the negative saturation value. When extension overflow occurs, AM35 determines saturation value. When sign bit overflow occurs, SL determines saturation value. **Table 11-2** summarizes bit values and saturation values.

Table 11-2 Saturation Values

AM35	EV	MV	SL	Saturation Value
0	1	0	—	$7FFF
1	1	0	—	$8000
—	1	1	1	$7FFF
—	1	1	0	$8000

11.7 DSP Instructions

Following are detailed descriptions of each DSP instruction. Instructions are grouped by function.

11.7.1 Initialization Instructions

The following instructions are used to set up multiply and accumulate operations.

11.7.1.1 LDHI — Load Registers H and I

LDHI must be used to initialize the multiplier and multiplicand registers before execution of MAC and RMAC instructions. HR is loaded with a memory word located at address (XK : IX). IR is loaded with a memory word located at address (YK : IY). LDHI operation does not affect the CCR.

11.7.1.2 TDMSK — Transfer D to XMSK:YMSK

TDMSK must be used to initialize the X and Y address masks prior to execution of MAC and RMAC instructions. The contents of the masks are replaced by the content of accumulator D. D[15:8] are transferred to XMSK, and D[7:0] are transferred to YMSK. The masks are used in modulo addressing. TDMSK operation does not affect the CCR.

11.7.1.3 TEDM — Transfer E and D to AM

TEDM places 32 bits of data in accumulator M. The content of accumulator E is transferred to AM[31:16], and the content of accumulator D is transferred to AM[15:0]. AM[35:32] reflect the state of AM31 after transfer is complete. TEDM also clears the CCR EV and MV bits.

11.7.1.4 TEM — Transfer E to AM

TEM initializes the upper 16 bits of accumulator M and clears the lower 16 bits. The content of accumulator E is transferred to AM[31:16]. AM[15:0] are cleared. AM[35:32] reflect the state of bit 31 after transfer is complete. TEM also clears the CCR EV and MV bits.

11.7.2 Transfer Instructions

The following instructions are used to transfer MAC data to general-purpose accumulators.

11.7.2.1 TMER — Transfer AM to E Rounded

The TMER instruction rounds a signed 32-bit fraction in accumulator M to 16 bits, then places the signed 16-bit fraction in accumulator E. The value represented by bits [15:0] of the fraction are rounded into the value represented by bits [31:16].

Bits [15:0] can have any value in the range $0000 to $FFFF. A value greater than $8000 must be rounded up, and a value less than $8000 must be rounded down. However, rounding values equal to $8000 in a single direction will introduce a bias. The CPU16 uses convergent rounding to avoid bias.

In convergent rounding, bit 16 determines whether a value of $8000 in bits [15:0] will be rounded up or down. When bit 16 = 1, a value of $8000 is rounded up; when bit 16 = 0, a value of $8000 is rounded down.

The EV, MV, N and Z bits in the CCR are set according to the results of the rounding operation. When saturation mode has been enabled, and either EV or MV is set, the appropriate saturation value will be placed in accumulator E.

If TMER is executed when saturation mode has not been enabled, and either EV or MV is set, the value in accumulator E will be meaningless.

11.7.2.2 TMET — Transfer AM to E Truncated

The TMET instruction truncates a signed 32-bit fraction in accumulator M to 16 bits, then places the signed 16-bit fraction in accumulator E. AM[31:16] are transferred to accumulator E.

The N and Z bits in the CCR are set according to the results of the transfer operation. When AM31 is set, N is set. When saturation mode has been enabled, and either EV or MV is set, the appropriate saturation value will be placed in accumulator E.

If TMER is executed when saturation mode has not been enabled, and either EV or MV is set, the value in accumulator E will be meaningless.

11.7.2.3 TMXED — Transfer AM to IX : E : D

TMXED provides a way to normalize AM when saturation mode is disabled and recovery from an extension bit overflow is necessary. AM[35:32] are transferred to IX[3:0]. IX[15:4] are sign-extended according to the content of AM35. AM[31:16] are transferred to accumulator E. AM[15:0] are transferred to accumulator D.

After TMXED is executed, transfer the content of IX to a RAM location, load data into E : D, then shift and round appropriately.

11.7.2.4 LDED/STED — Long Word Load and Store Instructions

While LDED and STED are not specifically intended for DSP, they operate on the concatenated E and D accumulators, and are useful for handling DSP values. See listings in **SECTION 6 INSTRUCTION GLOSSARY**.

11.7.3 Multiplication and Accumulation Instructions

These instructions are the heart of CPU16 digital signal processing capability. The MAC and RMAC instructions provide flexible control-oriented processing with modulo addressing, while the FMULS, ACE, and ACED instructions provide the ability to prescale and add constants.

11.7.3.1 MAC — Multiply and Accumulate

MAC multiplies a 16-bit signed fractional multiplicand contained in IR by a 16-bit signed fractional multiplier contained in HR. The product is left-shifted once to align the radix point between bits 31 and 30, then placed in E : D[31:1]. D0 is cleared. The aligned product is then added to the content of AM.

As the multiply and accumulate operation takes place, 4-bit X and Y offsets (xo, yo) specified by an instruction operand are sign-extended to 16 bits and used with XMSK and YMSK values to qualify the corresponding index registers. The following expressions are used to qualify the index registers:

$$IX = ((IX) \bullet \overline{X\ MASK}) \text{✣} ((IX) + xo) \bullet X\ MASK)$$
$$IY = ((IY) \bullet \overline{Y\ MASK}) \text{✣} ((IY) + yo) \bullet Y\ MASK)$$

Writing a non-zero value into a mask register prior to MAC execution enables modulo addressing. The TDMSK instruction writes mask values. When a mask contains $0, the sign-extended offset is added to the content of the corresponding index register.

After accumulation, HR content is transferred to IZ, then a word at the address pointed09.29 389.811

As multiply and accumulate operations take place, 4-bit offsets (xo, yo) specified by an instruction operand are sign-extended to 16 bits and used with XMSK and YMSK to qualify the corresponding index registers. The following expressions are used to qualify the index registers:

$$IX = ((IX) \bullet \overline{X\ MASK}) \text{✢} ((IX) + xo) \bullet X\ MASK)$$
$$IY = ((IY) \bullet \overline{Y\ MASK}) \text{✢} ((IY) + yo) \bullet Y\ MASK)$$

Writing a non-zero value into a mask register prior to RMAC execution enables modulo addressing. The TDMSK instruction writes mask values. When a mask contains $0, the sign-extended offset is added to the content of the corresponding index register.

After accumulation, a word pointed to by XK: IX is loaded into HR, and a word pointed to by YK: IY is loaded into IR, then the value in E is decremented and tested. If these values are to be used in successive RMAC operations, the registers must be re-initialized with the LDHI instruction. RMAC always iterates at least once, even when executed with a zero or negative value in E. Since the value in E is decremented, then tested, loading E with $8000 results in 32,770 iterations.

If HR and IR both contain $8000 (–1), a value of $80000000 (1.0 in 36-bit format) is accumulated, but no condition code is set.

RMAC execution is suspended during bus error, breakpoint, and interrupt exceptions. Operation resumes when RTI is executed at the end of the exception handler. In order for execution to resume correctly, all registers used by RMAC must be stacked or left unchanged by the exception handler. The PSHMAC and PULMAC instructions stack MAC unit resources. See **SECTION 9 EXCEPTION PROCESSING** for more information.

11.7.3.3 FMULS — Signed Fractional Multiply

FMULS left-shifts the product of a 16-bit signed fractional multiplication once before placing it in concatenated accumulators E and D.

A 16-bit signed fractional multiplicand contained by accumulator E is multiplied by a 16-bit signed fractional multiplier contained by accumulator D. There are implied radix points between bits 15 and 14 of the accumulators. The product is left-shifted one place to align the radix point between bits 31 and 30, then placed in E : D[31:1]. D0 is cleared.

When both accumulators contain $8000 (–1), the product is $80000000 (–1.0) and the CCR V bit is set.

11.7.3.4 ACED — Add E: D to AM

ACED is used with either of the FMULS or MAC instructions. It allows direct addition of 32-bit signed fractions to accumulator M. The concatenated contents of accumulators E and D are added to the content of accumulator M.

The value in the concatenated accumulators is assumed to be a 32-bit signed fraction with an implied radix point aligned between bits 31 and 30.

EV and MV in the CCR are set according to the result of ACED operation.

11.7.3.5 ACE — Add E to AM

ACE is used with either of the FMULS or MAC instructions. It allows direct addition of 16-bit signed fractions to accumulator M. The content of accumulator E is added to AM[31:16]. Bits 15 to 0 of accumulator M are not affected.

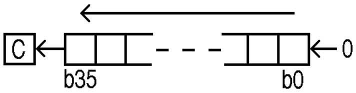
The value in E is assumed to be a 16-bit signed fraction with an implied radix point between bits 15 and 14.

EV and MV in the CCR are set according to the result of ACE operation.

11.7.4 Bit Manipulation Instructions

There are three instructions that operate directly on the bits in accumulator M. ASLM and ASRM perform 36-bit arithmetic shifts and CLRM clears the accumulator.

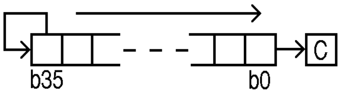
11.7.4.1 ASLM — Arithmetic Shift Left AM



Shifts all 36 bits of accumulator M one place to the left. Bit 35 is transferred to the CCR C bit. Bit 0 is loaded with a zero.

EV, MV, and N in the CCR are set according to the result of ASLM operation.

11.7.4.2 ASRM — Arithmetic Shift Right AM



Shifts all 36 bits of accumulator M one place to the right. Bit 0 is transferred to the CCR C bit. Bit 35 is held constant.

EV, MV, and N in the CCR are set according to the result of ASRM operation.

11.7.4.3 CLRM — Clear AM

CLRM provides a simple way to initialize accumulator M when a starting value of $000000000 is needed. AM[35:0] are cleared to zero. EV and MV in the CCR are also cleared.

11.7.5 Stacking Instructions

The PSHMAC and PULMAC instructions stack and restore all MAC resources.

11.7.5.1 PSHMAC — Push MAC Registers

PSHMAC stacks MAC registers in the sequence shown, beginning at the address pointed to by the stack pointer.

	15	8	7	0
(SP)	H REGISTER			
(SP) – $0002	I REGISTER			
(SP) – $0004	ACCUMULATOR M[15:0]			
(SP) – $0006	ACCUMULATOR M[31:16]			
(SP) – $0008	SL	RESERVED		AM[35:32]
(SP) – $000A	IX ADDRESS MASK		IY ADDRESS MASK	

The entire MAC unit internal state is saved on the system stack. Registers are stacked from high to low address. The stack pointer is automatically decremented after each save operation (the stack grows downward in memory). If SP overflow occurs as a result of operation, the SK field is decremented.

11.7.5.2 PULMAC — Pull MAC Registers

PULMAC restores MAC registers in the sequence shown, beginning at the address pointed to by the stack pointer.

	15	8	7	0
(SP) + $000A	IX ADDRESS MASK		IY ADDRESS MASK	
(SP) + $0008	SL	RESERVED		AM[35:32]
(SP) + $0006	ACCUMULATOR M[31:16]			
(SP) + $0004	ACCUMULATOR M[15:0]			
(SP) + $0002	I REGISTER			
(SP)	H REGISTER			

The entire MAC unit internal state is restored from the system stack. Registers are restored in order from low to high address. The SP is incremented after each restoration (stack shrinks upward in memory). If SP overflow occurs as a result of operation, the SK field is incremented.

11.7.6 Branch Instructions

LBEV and LBMV are conditional long branch instructions associated with the EV and MV bits in the CCR.

11.7.6.1 LBEV — Long Branch if EV Set

LBEV causes a long program branch if the EV bit in the condition code register has a value of one. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented.

Because the EV flag can be set and cleared more than once during the execution of RMAC instructions, exception handler routines that contain an LBEV instruction must be carefully designed.

11.7.6.2 LBMV — Long Branch if MV Set

LBMV causes a long program branch if the MV bit in the condition code register has a value of one. A 16-bit signed relative offset is added to the current value of the program counter. When the operation causes PC overflow, the PK field is incremented or decremented.

The MV bit is latched when sign bit overflow occurs, and must be cleared by an ANDP, CLRM, TAP, TDP, TEM, or TEDM instruction.

APPENDIX A COMPARISON OF CPU16/M68HC11 CPU ASSEMBLY LANGUAGE

A.1 Introduction

This appendix compares the assembly language of the M68HC11 microcontroller and the M68HC16 microcontroller. It provides information concerning functionally equivalent instructions and discusses cases that need special attention. It is intended to supplement the CPU16 Reference Manual — refer to appropriate sections of the manual for detailed information on system resources, addressing modes, instruction set, and processing flow.

The appendix is divided into eight sections. The first section shows M68HC11 CPU and CPU16 register models. The second discusses CPU16 instruction formats and pipelining. The third lists M68HC11 CPU instructions that have an equivalent CPU16 instruction. The fourth lists M68HC11 CPU instructions that operate differently on the CPU16. The fifth lists M68HC11 CPU assembler directives that operate differently on the CPU16, but for which the difference is transparent to the programmer. The sixth lists directives that have a new syntax. The seventh section discusses changes to addressing modes. The last section is an assembly language comparison in tabular format.

The CPU16 is designed for maximum compatibility with the M68HC11 CPU, and only moderate effort is required to port an application from an M68HC11 microcontroller to an M68HC16 microcontroller. Certain M68HC11instructions have been modified to support the improved addressing and exception handling capabilities of the CPU16. Other M68HC11 CPU instructions, particularly those related to manipulation of the condition code register, have been replaced.

A.2 Register Models

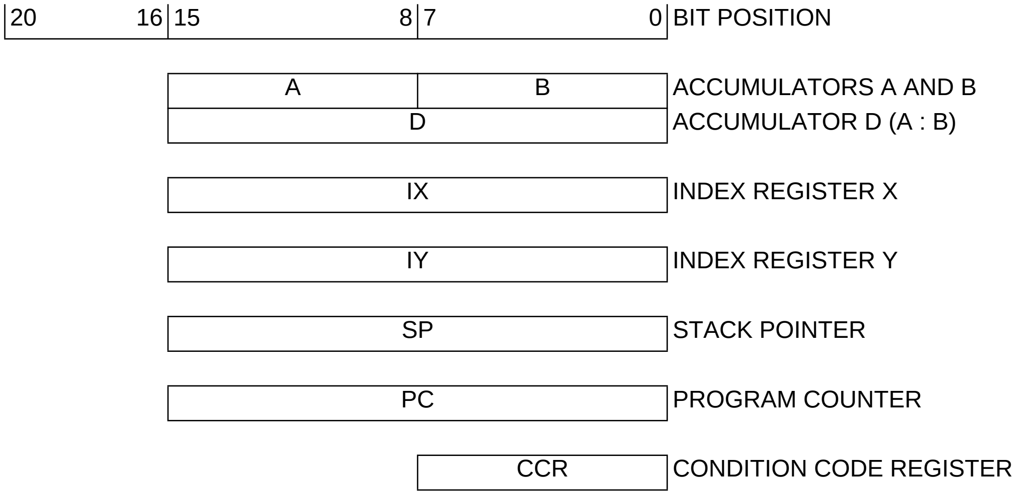


Figure A-1 M68HC11 CPU Registers

7	6	5	4	3	2	1	0
N	X	H	I	N	Z	V	C

Figure A-2 M68HC11 CPU Condition Code Register

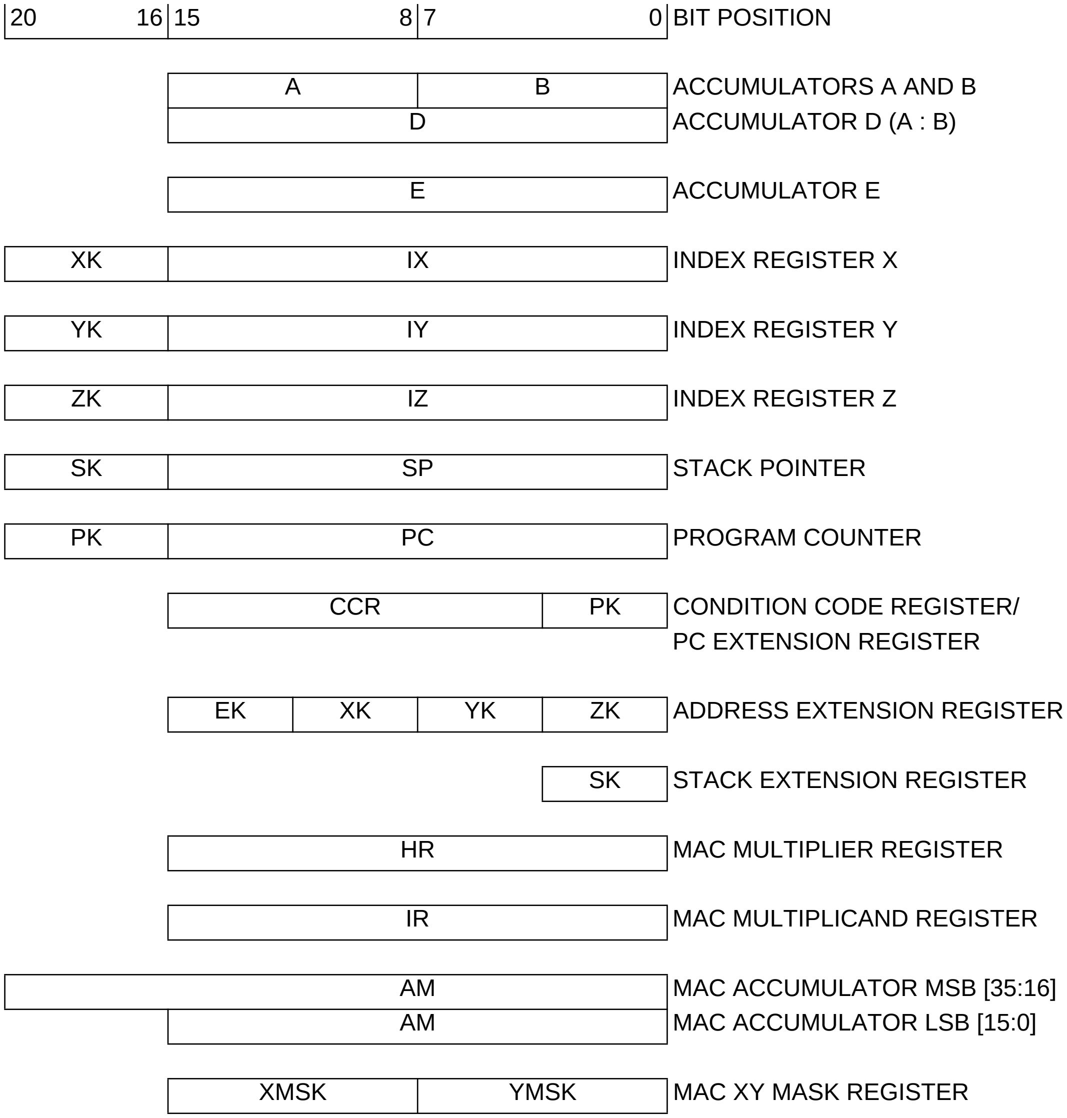


Figure A-3 CPU16 Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			

Figure A-4 CPU16 Condition Code Register

A.3 CPU16 Instruction Formats and Pipelining Mechanism

A.3.1 Instruction Format

CPU16 instructions consist of an 8-bit opcode, which may be preceded by an 8-bit prebyte and/or followed by one or more operands.

Opcodes are mapped in four 256-instruction pages. Page 0 opcodes stand alone, but page 1, 2, and 3 opcodes are pointed to by a prebyte code on page 0. The prebytes are $17 (page 1), $27 (page 2), and $37 (page 3).

Operands can be four bits, eight bits or sixteen bits in length. However, because the CPU16 fetches instructions from even byte boundaries, each instruction must contain an even number of bytes.

Operands are organized as bytes, words, or a combination of bytes and words. Four-bit operands are either zero-extended to eight bits, or packed two to a byte. The largest instructions are 6 bytes in length. Size, order, and function of operands are evaluated when an instruction is decoded.

A page 0 opcode and an 8-bit operand can be fetched simultaneously. Instructions that use 8-bit indexed, immediate, and relative addressing modes have this form — code written with these instructions is very compact.

A.3.2 Execution Model

This description is a simplified model of the mechanism the CPU16 uses to fetch and execute instructions. Functional divisions in the model do not necessarily correspond to distinct architectural subunits of the microprocessor.

There are three functional blocks involved in fetching, decoding, and executing instructions. These are the microsequencer, the instruction pipeline, and the execution unit. These elements function concurrently — at any given time, all three may be active.

A.3.2.1 Microsequencer

The microsequencer controls the order in which instructions are fetched, advanced through the pipeline, and executed. It increments the program counter and generates multiplexed external tracking signals IPIPE0 and IPIPE1 from internal signals that control execution sequence.

A.3.2.2 Instruction Pipeline

The pipeline is a three stage FIFO that holds instructions while they are decoded and executed. As many as three instructions can be in the pipeline at one time (single-word instructions, one held in stage C, one being executed in stage B, and one latched in stage A).

A.3.2.3 Execution Unit

The execution unit evaluates opcodes, interfaces with the microsequencer to advance instructions through the pipeline, and performs instruction operations.

A.3.3 Execution Process

Fetched opcodes are latched into stage A, then advanced to stage B. Opcodes are evaluated in stage B. The execution unit can access operands in either stage A or stage B (stage B accesses are limited to 8-bit operands). When execution is complete, opcodes are moved from stage B to stage C, where they remain until the next instruction is complete.

A prefetch mechanism in the microsequencer reads instruction words from memory and increments the program counter. When instruction execution begins, the program counter points to an address six bytes after the address of the first word of the instruction being executed.

The number of machine cycles necessary to complete an execution sequence varies according to the complexity of the instruction.

A.3.4 Changes in Program Flow

When program flow changes, instructions are fetched from a new address. Before execution can begin at the new address, instructions and operands from the previous instruction stream must be removed from the pipeline. If a change in flow is temporary, a return address must be stored, so that execution of the original instruction stream can resume after the change in flow.

At the time an instruction that causes a change in program flow executes, PK : PC point to the address of the first word of the instruction + $0006. During execution of the instruction, PK : PC is loaded with the address of the first word of the new instruction stream. However, stages A and B still contain words from the old instruction stream. The CPU16 prefetches to advance the new instruction to stage C, and fills the pipeline from the new instruction stream.

A.3.4.1 Jumps

The CPU16 jump instruction uses 20-bit extended and indexed addressing modes. It consists of an 8-bit opcode with a 20-bit argument. No return PK : PC is stacked for a jump.

A.3.4.2 Branches

The CPU16 supports 8-bit relative displacement (short), and 16-bit relative displacement (long) branch instructions, as well as specialized bit condition branches that use indexed addressing modes. CPU16 short branches are generally equivalent to M68HC11 CPU branches, although opcodes are not identical. M68HC11 BHI and BLO are replaced by CPU16 BCC and BCS.

Short branch instructions consist of an 8-bit opcode and an 8-bit operand contained in one word. Long branch instructions consist of an 8-bit prebyte and an 8-bit opcode in one word, followed by an operand word. Bit condition branches consist of an 8-bit opcode and an 8-bit operand in one word, followed by one or two operand words.

When a branch instruction executes, PK : PC point to an address equal to the address of the first word of the instruction plus $0006. The range of displacement for each type of branch is relative to this value. In addition, because prefetches are automatically aligned to word boundaries, only even offsets are valid — an odd offset value is rounded down.

A.3.4.3 Subroutines

Subroutines can be called by short (BSR) or long (LBSR) branches, or by a jump (JSR). The RTS instruction returns control to the calling routine. BSR consists of an 8-bit opcode with an 8-bit operand. LBSR consists of an 8-bit prebyte and an 8-bit opcode in one word, followed by an operand word. JSR consists of an 8-bit opcode with a 20-bit argument. RTS consists of an 8-bit prebyte and an 8-bit opcode in one word.

When a subroutine instruction is executed, PK: PC contain the address of the calling instruction plus $0006. All three calling instructions stack return PK : PC values prior to processing instructions from the new instruction stream. In order for RTS to work with all three calling instructions, however, the value stacked by BSR must be adjusted.

LBSR and JSR are two-word instructions. In order for program execution to resume with the instruction immediately following them, RTS must subtract $0002 from the stacked PK : PC value. BSR is a one-word instruction — it subtracts $0002 from PK : PC prior to stacking so that execution will resume correctly.

A.3.4.4 Interrupts

Interrupts are a type of exception, and are thus subject to special rules regarding execution process. This comparison is limited to the effects of SWI (software interrupt) and RTI (return from interrupt) instructions.

Both SWI and RTI consist of an 8-bit prebyte and an 8-bit opcode in one word. SWI initiates synchronous exception processing. RTI causes execution to resume with the instruction following the last instruction that completed execution prior to interrupt.

Asynchronous interrupts are serviced at instruction boundaries. PK : PC + $0006 for the following instruction is stacked, and exception processing begins. In order to resume execution with the correct instruction, RTI subtracts $0006 from the stacked value.

Interrupt exception processing is included in the SWI instruction definition. The PK : PC value at the time of execution is the first word address of SWI plus $0006. If this value were stacked, RTI would cause SWI to execute again. In order to resume execution with the instruction following SWI, $0002 is added to the PK : PC value prior to stacking.

A.3.4.5 Interrupt Priority

There are eight levels of interrupt priority. All interrupts with priorities less than seven can be masked by writing to the CCR interrupt priority (IP) field.

The IP field consists of three bits (CCR[7:5]). Binary values %000 to %111 provide eight priority masks. Masks prevent an interrupt request of a priority less than or equal to the mask value (except for NMI) from being recognized and processed. When IP contains %000, no interrupt is masked.

A.3.5 Stack Frame

When a change of flow occurs, the contents of the program counter and condition code register are stacked at the location pointed to by SK : SP. **Figure A-5** shows the stack frame. Unless it is altered during exception processing, the stacked PK : PC value is the address of the next instruction in the current instruction stream, plus $0006. RTS restores only stacked PK : PC – 2, while RTI restores PK : PC – 6 and the CCR.

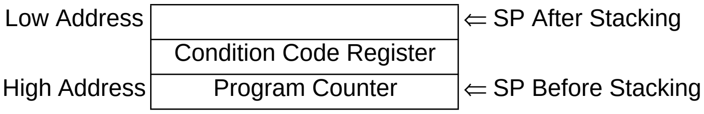


Figure A-5 CPU16 Stack Frame Format

A.4 Functionally Equivalent Instructions

A.4.1 BHS

The CPU16 uses only the BCC mnemonic. BHS is used in the M68HC11 CPU instruction set to differentiate a branch based on a comparison of unsigned numbers from a branch based on operations that clear the carry bit.

A.4.2 BLO

The CPU16 uses only the BCS mnemonic. BLO is used in the M68HC11 CPU instruction set to differentiate a branch based on a comparison of unsigned numbers from a branch based on operations that set the carry bit.

A.4.3 CLC

The CLC instruction has been replaced by ANDP. ANDP performs AND between the content of the condition code register and an unsigned immediate operand, then replaces the content of the CCR with the result. The PK extension field (CCR[0:3]) is not affected.

The following code can be used to clear the C bit in the CCR:

```
ANDP #$FEFF
```

The ANDP instruction can clear the entire CCR, except for the PK extension field, at once.

A.4.4 CLI

The CLI instruction has been replaced by ANDP. ANDP performs AND between the content of the condition code register and an unsigned immediate operand, then replaces the content of the CCR with the result. The PK extension field (CCR[0:3]) is not affected.

The following code can be used to clear the IP field in the CCR:

```
ANDP #$FF1F
```

The ANDP instruction can clear the entire CCR, except for the PK extension field, at once.

A.4.5 CLV

The CLV instruction has been replaced by ANDP. ANDP performs AND between the content of the condition code register and an unsigned immediate operand, then replaces the content of the CCR with the result. The PK extension field (CCR[0:3]) is not affected.

The following code can be used to clear the V bit in the CCR:

```
ANDP #$FDFF
```

The ANDP instruction can clear the entire CCR, except for the PK extension field, at once.

A.4.6 DES

The DES instruction has been replaced by AIS. AIS adds a 20-bit value to concatenated SK and SP. The 20-bit value is formed by sign-extending an 8-bit or 16-bit signed immediate operand.

The following code can be used to perform a DES:

```
AIS –1
```

CPU16 stacking operations normally use 16-bit words and even word addresses, while M68HC11 CPU stacking operations normally use bytes and byte addresses. If the CPU16 stack pointer is misaligned as a result of a byte operation, performance can be degraded.

A.4.7 DEX

The DEX instruction has been replaced by AIX. AIX adds a 20-bit value to concatenated XK and IX. The 20-bit value is formed by sign-extending an 8-bit or 16-bit signed immediate operand.

The following code can be used to perform a DEX:

```
AIX –1
```

A.4.8 DEY

The DEY instruction has been replaced by AIY. AIY adds a 20-bit value to concatenated YK and IY. The 20-bit value is formed by sign-extending an 8-bit or 16-bit signed immediate operand.

The following code can be used to perform a DEY:

```
AIY –1
```

A.4.9 INS

The INS instruction has been replaced by AIS. AIS adds a 20-bit value to concatenated SK and SP. The 20-bit value is formed by sign-extending an 8-bit or 16-bit signed immediate operand.

The following code can be used to perform an INS:

```
AIS –1
```

CPU16 stacking operations normally use 16-bit words and even word addresses, while M68HC11 CPU stacking operations normally use bytes and byte addresses. If the CPU16 stack pointer is misaligned as a result of a byte operation, performance can be degraded.

A.4.10 INX

The INX instruction has been replaced by AIX. AIX adds a 20-bit value to concatenated XK and IX. The 20-bit value is formed by sign-extending an 8-bit or 16-bit signed immediate operand.

The following code can be used to perform an INX:

A.4.11 INY

The INY instruction has been replaced by AIY. AIY adds a 20-bit value to concatenated YK and IY. The 20-bit value is formed by sign-extending an 8-bit or 16-bit signed immediate operand.

The following code can be used to perform an INY:

```
AIY 1
```

A.4.12 PSHX

The PSHX instruction has been replaced by PSHM. PSHM stores the contents of selected registers on the system stack. Registers are designated by setting bits in a mask byte.

The following code can be used to stack index register X:

```
PSHM X
```

The CPU16 can stack up to seven registers with a single PSHM instruction.

A.4.13 PSHY

The PSHY instruction has been replaced by PSHM. PSHM stores the contents of selected registers on the system stack. Registers are designated by setting bits in a mask byte.

The following code can be used to stack index register Y:

```
PSHM Y
```

The CPU16 can stack up to seven registers with a single PSHM instruction.

A.4.14 PULX

The PULX instruction has been replaced by PULM. PULM restores the contents of selected registers from the system stack. Registers are designated by setting bits in a mask byte.

The following code can be used to restore index register X:

```
PULM X
```

The CPU16 can restore up to seven registers with a single PULM instruction. As a part of normal execution, PULM reads an extra location in memory. The extra data is discarded. A PULM from the highest available location in memory will cause an attempt to read an unimplemented location, with unpredictable results.

A.4.15 PULY

The PULY instruction has been replaced by PULM. PULM restores the contents of selected registers from the system stack. Registers are designated by setting bits in a mask byte.

The following code can be used to restore index register Y:

```
PULM Y
```

The CPU16 can restore up to seven registers with a single PULM instruction. As a part of normal execution, PULM reads an extra location in memory. The extra data is discarded. A PULM from the highest available location in memory will cause an attempt to read an unimplemented location, with unpredictable results.

A.4.16 SEC

The SEC instruction has been replaced by ORP. ORP performs inclusive OR between the content of the condition code register and an unsigned immediate operand, then replaces the content of the CCR with the result. The PK extension field (CCR[3:0]) is not affected.

The following code can be used to set the CCR C bit:

```
ORP #$0100
```

The ORP instruction can set all CCR bits, except the PK extension field, at once.

A.4.17 SEI

The SEI instruction has been replaced by ORP. ORP performs inclusive OR between the content of the condition code register and an unsigned immediate operand, then replaces the content of the CCR with the result. The PK extension field (CCR[3:0]) is not affected.

The following code can be used to set all the bits in the CCR IP field:

```
ORP #$00E0
```

The ORP instruction can set all CCR bits, except the PK extension field, at once.

A.4.18 SEV

The SEV instruction has been replaced by ORP. ORP performs inclusive OR between the content of the condition code register and an unsigned immediate operand, then replaces the content of the CCR with the result. The PK extension field (CCR[3:0]) is not affected.

The following code can be used to set the CCR V bit:

```
ORP #$0200
```

The ORP instruction can set all CCR bits, except the PK extension field, at once.

A.4.19 STOP (LPSTOP)

LPSTOP is used to minimize microcontroller power consumption. The CPU16 has seven levels of interrupt priority. If an interrupt request of higher priority than the priority value stored when the microcontroller enters low-power stop mode is received, the microcontroller is activated, and the CPU16 processes an interrupt exception.

A.5 Instructions that Operate Differently

A.5.1 BSR

The CPU16 stack frame differs from the M68HC11 CPU stack frame. The CPU16 stacks the current PC and CCR, but restores only the return PK: PC. The programmer must designate (PSHM) which other registers are stacked during a subroutine. Because SK : SP point to the next available word address, stacked CPU16 parameters are at a different offset from the stack pointer than stacked M68HC11 CPU parameters. In order for RTS to work with all three calling instructions, the PK : PC value stacked by BSR is decremented by two before being pushed on to the stack. Stacked PC value is the return address + $0002.

A.5.2 JSR

The CPU16 stack frame differs from the M68HC11 CPU stack frame. The CPU16 stacks the current PC and CCR, but restores only the return PK : PC. The programmer must designate (PSHM) which other registers are stacked during a subroutine. Because SK : SP point to the next available word address, stacked CPU16 parameters are at a different offset from the stack pointer than stacked M68HC11 CPU parameters.

A.5.3 PSHA, PSHB

These instructions operate in the same way as the M68HC11 instructions with the same mnemonics. However, because the CPU16 normally pushes words from an even boundary, pushing byte data to the stack can misalign the stack pointer and degrade performance.

A.5.4 PULA, PULB

These instructions operate in the same way as the M68HC11 instructions with the same mnemonics. However, because the CPU16 normally pulls words from the stack, pulling byte data can misalign the stack pointer and degrade performance.

A.5.5 RTI

The CPU16 stack frame differs from the M68HC11 CPU stack frame. The CPU16 stacks only the current PC and CCR before exception processing begins. In order to resume execution after interrupt with the correct instruction, RTI subtracts $0006 from the stacked PK : PC.

A.5.6 SWI

The CPU16 stack frame differs from the M68HC11 CPU stack frame. The PK : PC value at the time of execution is the first word address of SWI plus $0006. If this value were stacked, RTI would cause SWI to execute again. In order to resume execution with the instruction following SWI, $0002 is added to the PK : PC value prior to stacking. The programmer must designate (PSHM) which other registers are stacked during an interrupt.

A.5.7 TAP

The CPU16 CCR and the M68HC11 CPU CCR are different. The CPU16 interrupt priority scheme differs from that of the M68HC11 CPU. The CPU16 interrupt priority field cannot be changed by the TAP instruction.

A.5.7.1 M68HC11 CPU Implementation:

7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0
⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩
7	6	5	4	3	2	1	0
S	X	H	I	N	Z	V	C

A.5.7.2 CPU16 Implementation:

7	6	5	4	3	2	1	0								
A7	A6	A5	A4	A3	A2	A1	A0								
⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C		IP		SM		PK		

A.5.8 TPA

The CPU16 CCR and the M68HC11 CPU CCR are different. TPA cannot be used to read CPU16 interrupt priority status. Use TPD to read the CPU16 CCR interrupt priority field.

A.5.8.1 M68HC11 CPU Implementation:

7	6	5	4	3	2	1	0
S	X	H	I	N	Z	V	C
⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩
7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0

A.5.8.2 CPU16 Implementation:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩								
7	6	5	4	3	2	1	0								
A7	A6	A5	A4	A3	A2	A1	A0								

A.5.9 WAI

The CPU16 does not stack registers during WAI. The CPU16 acknowledges interrupts faster out of WAI than LPSTOP. However, LPSTOP minimizes microcontroller power consumption.

A.6 Instructions With Transparent Changes

A.6.1 RTS

The CPU16 stack frame differs from the M68HC11 CPU stack frame. PK : PC is restored during an RTS. The PK field in the CCR is restored, then the PC value read from the stack is decremented by two before being loaded into the PC. The PC value is decremented because LBSR and JSR are two-word instructions. In order for program execution to resume with the instruction immediately following them, RTS must subtract $0002 from the stacked PK : PC value. Because BSR is a one-word instruction, it subtracts $0002 from PK : PC prior to stacking so that execution will resume correctly after RTS.

A.6.2 TSX

The CPU16 adds two to SK : SP before the transfer to XK : IX. The M68HC11 CPU adds one.

A.6.3 TSY

The CPU16 adds two to SK : SP before the transfer to YK : IY. The M68HC11 CPU adds one.

A.6.4 TXS

The CPU16 subtracts two from XK : IX before the transfer to SK : SP. The M68HC11 CPU subtracts one.

A.6.5 TYS

The CPU16 subtracts two from YK : IY before the transfer to SK : SP. The M68HC11 CPU subtracts one.

A.7 Unimplemented Instructions

A.7.1 TEST

Causes the program counter to be continuously incremented.

A.8 Addressing Mode Differences

A.8.1 Extended Addressing Mode

In M68HC11 CPU extended addressing mode, the effective address of the instruction appears explicitly in the two bytes following the opcode. In CPU16 extended addressing mode, the effective address is formed by concatenating the EK field and the 16-bit byte address. A 20-bit extended mode (EXT20) is used only by the JMP and JSR instructions. These instructions contain a 20-bit effective address that is zero-extended to 24 bits to give the instruction an even number of bytes.

A.8.2 Indexed Addressing Mode

M68HC11 CPU indexed addressing mode forms the effective address by adding the fixed, 8-bit, unsigned offset to the index register. In CPU16 indexed addressing mode, a fixed 16-bit offset can be used. Note however, that the 16-bit offset is signed and can give a negative offset from the index register. An 8-bit unsigned mode is still available on the CPU16. A 20-bit indexed mode is used for JMP and JSR instructions. In 20-bit modes, a 20-bit signed offset is added to the value contained in an index register.

A.8.3 Post-Modified Index Addressing Mode

Post-modified index mode is used with the CPU16 MOVB and MOVW instructions. A signed 8-bit offset is added to index register X after the effective address formed by XK : IX is used.

A.8.4 Use of CPU16 Indexed Mode to Replace M68HC11 CPU Direct Mode

In M68HC11 systems, direct addressing mode can be used to perform rapid accesses to RAM or I/O mapped into bank 0 ($0000 to $00FF), but the CPU16 uses the first 512 bytes of bank 0 for exception vectors. To provide an enhanced replacement for direct mode, the ZK field and index register Z have been assigned reset initialization vectors. After ZK : IZ have been initialized, indexed addressing provides rapid access to useful data structures.

Table A-1 M68HC16 Implementation of M68HC11 Instructions

M68HC11 Instruction	M68HC16 Implementation
BHS	Replaced by BCC
BLO	Replaced by BCS
BSR	Generates a different stack frame
CLC	Replaced by ANDP
CLI	Replaced by ANDP
CLV	Replaced by ANDP
DES	Replaced by AIS
DEX	Replaced by AIX
DEY	Replaced by AIY
INS	Replaced by AIS
INX	Replaced by AIX
INY	Replaced by AIY
JMP	IND8 addressing modes replaced by IND20 and EXT modes
JSR	IND8 addressing modes replaced by IND20 and EXT modes Generates a different stack frame
LSL, LSLD	Use ASL instructions*
PSHX	Replaced by PSHM
PSHY	Replaced by PSHM
PULX	Replaced by PULM
PULY	Replaced by PULM
RTI	Reloads PC and CCR only
RTS	Uses two-word stack frame
SEC	Replaced by ORP
SEI	Replaced by ORP
SEV	Replaced by ORP
STOP	Replaced by LPSTOP
TAP	CPU16 CCR bits differ from M68HC11 CPU16 interrupt priority scheme differs from M68HC11
TPA	CPU16 CCR bits differ from M68HC11 CPU16 interrupt priority scheme differs from M68HC11
TSX	Adds two to SK : SP before transfer to XK : IX
TSY	Adds two to SK : SP before transfer to YK : IY
TXS	Subtracts two from XK : IX before transfer to SK : SP
TXY	Transfers XK field to YK field
TYS	Subtracts two from YK : IY before transfer to SK : SP
TYX	Transfers YK field to XK field
WAI	Waits indefinitely for interrupt or reset Generates a different stack frame

*Motorola assemblers will automatically translate LSL mnemonics

APPENDIX B MOTOROLA ASSEMBLER SYNTAX

Name	Mode	Syntax
ABA	INH	aba
ABX	INH	abx
ABY	INH	aby
ABZ	INH	abz
ACE	INH	ace
ACED	INH	aced
ADCA	IND8, X	adca ff,x
	IND8, Y	adca ff,y
	IND8, Z	adca ff,z
	IMM8	adca #ii
	IND16, X	adca gggg,x
	IND16, Y	adca gggg,y
	IND16, Z	adca gggg,z
	EXT	adca hhll
	E, X	adca e,x
	E, Y	adca e,y
	E, Z	adca e,z
ADCB	IND8, X	adcb ff,x
	IND8, Y	adcb ff,y
	IND8, Z	adcb ff,z
	IMM8	adcb #ii
	IND16, X	adcb gggg,x
	IND16, Y	adcb gggg,y
	IND16, Z	adcb gggg,z
	EXT	adcb hhll
	E, X	adcb e,x
	E, Y	adcb e,y
	E, Z	adcb e,z
ADCD	IND8, X	adcd ff,x
	IND8, Y	adcd ff,y
	IND8, Z	adcd ff,z
	IMM16	adcd #jjkk
	IND16, X	adcd gggg,x
	IND16, Y	adcd gggg,y
	IND16, Z	adcd gggg,z
	EXT	adcd hhll
	E, X	adcd e,x
	E, Y	adcd e,y
	E, Z	adcd e,z
ADCE	IMM16	adce #jjkk
	IND16, X	adce gggg,x
	IND16, Y	adce gggg,y
	IND16, Z	adce gggg,z
	EXT	adce hhll
ADDA	IND8, X	adda ff,x
	IND8, Y	adda ff,y
	IND8, Z	adda ff,z
	IMM8	adda #ii
	IND16, X	adda gggg,x
	IND16, Y	adda gggg,y
	IND16, Z	adda gggg,z
	EXT	adda hhll
	E, X	adda e,x
	E, Y	adda e,y
	E, Z	adda e,z
ADDB	IND8, X	addb ff,x
	IND8, Y	addb ff,y
	IND8, Z	addb ff,z
	IMM8	addb #ii
	IND16, X	addb gggg,x
	IND16, Y	addb gggg,y
	IND16, Z	addb gggg,z
	EXT	addb hhll
	E, X	addb e,x
	E, Y	addb e,y
	E, Z	addb e,z
ADDD	IND8, X	addd ff,x
	IND8, Y	addd ff,y
	IND8, Z	addd ff,z
	IMM8	addd #ii
	IMM16	addd #jjkk
	IND16, X	addd gggg,x
	IND16, Y	addd gggg,y
	IND16, Z	addd gggg,z
	EXT	addd hhll
	E, X	addd e,x
	E, Y	addd e,y
	E, Z	addd e,z

Name	Mode	Syntax
ADDE	IMM8	adde #ii
	IMM16	adde #jjkk
	IND16, X	adde gggg,x
	IND16, Y	adde gggg,y
	IND16, Z	adde gggg,z
	EXT	adde hhll
ADE	INH	ade
ADX	INH	adx
ADY	INH	ady
ADZ	INH	adz
AEX	INH	aex
AEY	INH	aey
AEZ	INH	aez
AIS	IMM8	ais #ii
	IMM16	ais #jjkk
AIX	IMM8	aix #ii
	IMM16	aix #jjkk
AIY	IMM8	aiy #ii
	IMM16	aiy #jjkk
AIZ	IMM8	aiz #ii
	IMM16	aiy #jjkk
ANDA	IND8, X	anda ff,x
	IND8, Y	anda ff,y
	IND8, Z	anda ff,z
	IMM8	anda #ii
	IND16, X	anda gggg,x
	IND16, Y	anda gggg,y
	IND16, Z	anda gggg,z
	EXT	anda hhll
	E, X	anda e,x
	E, Y	anda e,y
	E, Z	anda e,z

Name	Mode	Syntax
ANDB	IND8, X	andb ff,x
	IND8, Y	andb ff,y
	IND8, Z	andb ff,z
	IMM8	andb #ii
	IND16, X	andb gggg,x
	IND16, Y	andb gggg,y
	IND16, Z	andb gggg,z
	EXT	andb hhll
	E, X	andb e,x
	E, Y	andb e,y
	E, Z	andb e,z
ANDD	IND8, X	andd ff,x
	IND8, Y	andd ff,y
	IND8, Z	andd ff,z
	IMM16	andd #jjkk
	IND16, X	andd gggg,x
	IND16, Y	andd gggg,y
	IND16, Z	andd gggg,z
	EXT	andd hhll
	E, X	andd e,x
	E, Y	andd e,y
	E, Z	andd e,z
ANDE	IMM16	ande #jjkk
	IND16, X	ande gggg,x
	IND16, Y	ande gggg,y
	IND16, Z	ande gggg,z
	EXT	ande hhll
ANDP	IMM16	andp #jjkk
ASL	IND8, X	asl ff,x
	IND8, Y	asl ff,y
	IND8, Z	asl ff,z
	IND16, X	asl gggg,x
	IND16, Y	asl gggg,y
	IND16, Z	asl gggg,z
	EXT	asl hhll
ASLA	INH	asla
ASLB	INH	aslb
ASLD	INH	asld
ASLE	INH	asle
ASLM	INH	aslm

Name	Mode	Syntax
ASLW	IND16, X	aslw gggg,x
	IND16, Y	aslw gggg,y
	IND16, Z	aslw gggg,z
	EXT	aslw hhll
ASR	IND8, X	asr ff,x
	IND8, Y	asr ff,y
	IND8, Z	asr ff,z
	IND16, X	asr gggg,x
	IND16, Y	asr gggg,y
	IND16, Z	asr gggg,z
	EXT	asr hhll
ASRA	INH	asra
ASRB	INH	asrb
ASRD	INH	asrd
ASRE	INH	asre
ASRM	INH	asrm
ASRW	IND16, X	asrw gggg,x
	IND16, Y	asrw gggg,y
	IND16, Z	asrw gggg,z
	EXT	asrw hhll
BCC	REL8	bcc rr
BCLR	IND8, X	bclr ff,x,#mm
	IND8, Y	bclr ff,y,#mm
	IND8, Z	bclr ff,z,#mm
	IND16, X	bclr gggg,x,#mm
	IND16, Y	bclr gggg,y,#mm
	IND16, Z	bclr gggg,z,#mm
	EXT	bclr hhll,#mm
BCLRW	IND16, X	bclrw gggg,x,#mmmm
	IND16, Y	bclrw gggg,y,#mmmm
	IND16, Z	bclrw gggg,z,#mmmm
	EXT	bclrw hhll,#mmmm
BCS	REL8	bcs rr
BEQ	REL8	beq rr
BGE	REL8	bge rr
BGND	INH	bgnd
BGT	REL8	bgt rr
BHI	REL8	bhi rr

Name	Mode	Syntax
BITA	IND8, X	bita ff,x
	IND8, Y	bita ff,y
	IND8, Z	bita ff,z
	IMM8	bita #ii
	IND16, X	bita gggg,x
	IND16, Y	bita gggg,y
	IND16, Z	bita gggg,z
	EXT	bita hhll
	E, X	bita e,x
	E, Y	bita e,y
	E, Z	bita e,z
BITB	IND8, X	bitb ff,x
	IND8, Y	bitb ff,y
	IND8, Z	bitb ff,z
	IMM8	bitb #ii
	IND16, X	bitb gggg,x
	IND16, Y	bitb gggg,y
	IND16, Z	bitb gggg,z
	EXT	bitb hhll
	E, X	bitb e,x
	E, Y	bitb e,y
	E, Z	bitb e,z
BLE	REL8	ble rr
BLS	REL8	bls rr
BLT	REL8	blt rr
BMI	REL8	bmi rr
BNE	REL8	bne rr
BPL	REL8	bpl rr
BRA	REL8	bra rr
BRCLR	IND8, X	brclr ff,x,#mm,rr
	IND8, Y	brclr ff,y,#mm,rr
	IND8, Z	brclr ff,z,#mm,rr
	IND16, X	brclr gggg,x,#mm,rrrr
	IND16, Y	brclr gggg,y,#mm,rrrr
	IND16, Z	brclr gggg,z,#mm,rrrr
	EXT	brclr hhll,#mm,rrrr
BRN	REL8	brn rr

Name	Mode	Syntax
BRSET	IND8, X	brset ff,x,#mm,rr
	IND8, Y	brset ff,y,#mm,rr
	IND8, Z	brset ff,z,#mm,rr
	IND16, X	brset gggg,x,#mm,rrrr
	IND16, Y	brset gggg,y,#mm,rrrr
	IND16, Z	brset gggg,z,#mm,rrrr
	EXT	brset hhll,#mm,rrrr
BSET	IND8, X	bset ff,x,#mm
	IND8, Y	bset ff,y,#mm
	IND8, Z	bset ff,z,#mm
	IND16, X	bset gggg,x,#mm
	IND16, Y	bset gggg,y,#mm
	IND16, Z	bset gggg,z,#mm
	EXT	bset hhll,#mm
BSETW	IND16, X	bsetw gggg,x,#mmmm
	IND16, Y	bsetw gggg,y,#mmmm
	IND16, Z	bsetw gggg,z,#mmmm
	EXT	bsetw hhll,#mmmm
BSR	REL8	bsr rr
BVC	REL8	bvc rr
BVS	REL8	bvs rr
CBA	INH	cba
CLR	IND8, X	clr ff,x
	IND8, Y	clr ff,y
	IND8, Z	clr ff,z
	IND16, X	clr gggg,x
	IND16, Y	clr gggg,y
	IND16, Z	clr gggg,z
	EXT	clr hhll
CLRA	INH	clra
CLRB	INH	clrb
CLRD	INH	clrd
CLRE	INH	clre
CLRM	INH	clrm
CLRW	IND16, X	clrw gggg,x
	IND16, Y	clrw gggg,y
	IND16, Z	clrw gggg,z
	EXT	clrw hhll

Name	Mode	Syntax
CMPA	IND8, X	cmpa ff,x
	IND8, Y	cmpa ff,y
	IND8, Z	cmpa ff,z
	IMM8	cmpa #ii
	IND16, X	cmpa gggg,x
	IND16, Y	cmpa gggg,y
	IND16, Z	cmpa gggg,z
	EXT	cmpa hhll
	E, X	cmpa e,x
	E, Y	cmpa e,y
	E, Z	cmpa e,z
CMPB	IND8, X	cmpb ff,x
	IND8, Y	cmpb ff,y
	IND8, Z	cmpb ff,z
	IMM8	cmpb #ii
	IND16, X	cmpb gggg,x
	IND16, Y	cmpb gggg,y
	IND16, Z	cmpb gggg,z
	EXT	cmpb hhll
	E, X	cmpb e,x
	E, Y	cmpb e,y
	E, Z	cmpb e,z
COM	IND8, X	com ff,x
	IND8, Y	com ff,y
	IND8, Z	com ff,z
	IND16, X	com gggg,x
	IND16, Y	com gggg,y
	IND16, Z	com gggg,z
	EXT	com hhll
COMA	INH	coma
COMB	INH	comb
COMD	INH	comd
COME	INH	come
COMW	IND16, X	comw gggg,x
	IND16, Y	comw gggg,y
	IND16, Z	comw gggg,z
	EXT	comw hhll

Name	Mode	Syntax
CPD	IND8, X	cpd ff,x
	IND8, Y	cpd ff,y
	IND8, Z	cpd ff,z
	IMM16	cpd #jjkk
	IND16, X	cpd gggg,x
	IND16, Y	cpd gggg,y
	IND16, Z	cpd gggg,z
	EXT	cpd hhll
	E, X	cpd e,x
	E, Y	cpd e,y
	E, Z	cpd e,z
CPE	IMM16	cpe #jjkk
	IND16, X	cpe gggg,x
	IND16, Y	cpe gggg,y
	IND16, Z	cpe gggg,z
	EXT	cpe hhll
CPS	IND8, X	cps ff,x
	IND8, Y	cps ff,y
	IND8, Z	cps ff,z
	IMM16	cps #jjkk
	IND16, X	cps gggg,x
	IND16, Y	cps gggg,y
	IND16, Z	cps gggg,z
	EXT	cps hhll
CPX	IND8, X	cpx ff,x
	IND8, Y	cpx ff,y
	IND8, Z	cpx ff,z
	IMM16	cpx #jjkk
	IND16, X	cpx gggg,x
	IND16, Y	cpx gggg,y
	IND16, Z	cpx gggg,z
	EXT	cpx hhll
CPY	IND8, X	cpy ff,x
	IND8, Y	cpy ff,y
	IND8, Z	cpy ff,z
	IMM16	cpy #jjkk
	IND16, X	cpy gggg,x
	IND16, Y	cpy gggg,y
	EXT	cpy hhll

Name	Mode	Syntax
CPZ	IND8, X	cpz ff,x
	IND8, Y	cpz ff,y
	IND8, Z	cpz ff,z
	IMM16	cpz #jjkk
	IND16, X	cpz gggg,x
	IND16, Y	cpz gggg,y
	IND16, Z	cpz gggg,z
	EXT	cpz hhll
DAA	INH	daa
DEC	IND8, X	dec ff,x
	IND8, Y	dec ff,y
	IND8, Z	dec ff,z
	IND16, X	dec gggg,x
	IND16, Y	dec gggg,y
	IND16, Z	dec gggg,z
	EXT	dec hhll
DECA	INH	deca
DECB	INH	decb
DECW	IND16, X	decw gggg,x
	IND16, Y	decw gggg,y
	IND16, Z	decw gggg,z
	EXT	decw hhll
EDIV	INH	ediv
EDIVS	INH	edivs
EMUL	INH	emul
EMULS	INH	emuls
EORA	IND8, X	eora ff,x
	IND8, Y	eora ff,y
	IND8, Z	eora ff,z
	IMM8	eora #ii
	IND16, X	eora gggg,x
	IND16, Y	eora gggg,y
	IND16, Z	eora gggg,z
	EXT	eora hhll
	E, X	eora e,x
	E, Y	eora e,y
	E, Z	eora e,z

Name	Mode	Syntax
EORB	IND8, X	eorb ff,x
	IND8, Y	eorb ff,y
	IND8, Z	eorb ff,z
	IMM8	eorb #ii
	IND16, X	eorb gggg,x
	IND16, Y	eorb gggg,y
	IND16, Z	eorb gggg,z
	EXT	eorb hhll
	E, X	eorb e,x
	E, Y	eorb e,y
	E, Z	eorb e,z
EORD	IND8, X	eord ff,x
	IND8, Y	eord ff,y
	IND8, Z	eord ff,z
	IMM16	eord #jjkk
	IND16, X	eord gggg,x
	IND16, Y	eord gggg,y
	IND16, Z	eord gggg,z
	EXT	eord hhll
	E, X	eord e,x
	E, Y	eord e,y
	E, Z	eord e,z
EORE	IMM16	eore #jjkk
	IND16, X	eore gggg,x
	IND16, Y	eore gggg,y
	IND16, Z	eore gggg,z
	EXT	eore hhll
FDIV	INH	fdiv
FMULS	INH	fmuls
IDIV	INH	idiv
INC	IND8, X	inc ff,x
	IND8, Y	inc ff,y
	IND8, Z	inc ff,z
	IND16, X	inc gggg,x
	IND16, Y	inc gggg,y
	IND16, Z	inc gggg,z
	EXT	inc hhll
INCA	INH	inca
INCB	INH	incb

Name	Mode	Syntax
INCW	IND16, X	incw gggg,x
	IND16, Y	incw gggg,y
	IND16, Z	incw gggg,z
	EXT	incw hhll
JMP	EXT20	jmp zb hhll
	IND20, X	jmp zg gggg,x
	IND20, Y	jmp zg gggg,y
	IND20, Z	jmp zg gggg,z
JSR	EXT20	jsr zb hhll
	IND20, X	jsr zg gggg,x
	IND20, Y	jsr zg gggg,y
	IND20, Z	jsr zg gggg,z
LBCC	REL8	lbcc rrrr
LBCS	REL8	lbcs rrrr
LBEQ	REL8	lbeq rrrr
LBEV	REL8	lbev rrrr
LBGE	REL8	lbge rrrr
LBGT	REL8	lbgt rrrr
LBHI	REL8	lbhi rrrr
LBLE	REL8	lble rrrr
LBLS	REL8	lbls rrrr
LBLT	REL8	lblt rrrr
LBMI	REL8	lbmi rrrr
LBMV	REL8	lbmv rrrr
LBNE	REL8	lbne rrrr
LBPL	REL8	lbpl rrrr
LBRA	REL8	lbra rrrr
LBM	REL8	lbrn rrrr
LBSR	REL8	lbsr rrrr
LBVC	REL8	lbvc rrrr
LBVS	REL8	lbvs rrrr

Name	Mode	Syntax
LDAA	IND8, X	ldaa ff,x
	IND8, Y	ldaa ff,y
	IND8, Z	ldaa ff,z
	IMM8	ldaa #ii
	IND16, X	ldaa gggg,x
	IND16, Y	ldaa gggg,y
	IND16, Z	ldaa gggg,z
	EXT	ldaa hhll
	E, X	ldaa e,x
	E, Y	ldaa e,y
	E, Z	ldaa e,z
LDAB	IND8, X	ldab ff,x
	IND8, Y	ldab ff,y
	IND8, Z	ldab ff,z
	IMM8	ldab #ii
	IND16, X	ldab gggg,x
	IND16, Y	ldab gggg,y
	IND16, Z	ldab gggg,z
	EXT	ldab hhll
	E, X	ldab e,x
	E, Y	ldab e,y
	E, Z	ldab e,z
LDD	IND8, X	ldd ff,x
	IND8, Y	ldd ff,y
	IND8, Z	ldd ff,z
	IMM16	ldd #jjkk
	IND16, X	ldd gggg,x
	IND16, Y	ldd gggg,y
	IND16, Z	ldd gggg,z
	EXT	ldd hhll
	E, X	ldd e,x
	E, Y	ldd e,y
	E, Z	ldd e,z
LDE	IMM16	lde #jjkk
	IND16, X	lde gggg,x
	IND16, Y	lde gggg,y
	IND16, Z	lde gggg,z
	EXT	lde hhll
LDED	EXT	lded hhll
LDHI	EXT	ldhi hhll

Name	Mode	Syntax
LDS	IND8, X	lds ff,x
	IND8, Y	lds ff,y
	IND8, Z	lds ff,z
	IMM16	lds #jjkk
	IND16, X	lds gggg,x
	IND16, Y	lds gggg,y
	IND16, Z	lds gggg,z
	EXT	lds hhll
LDX	IND8, X	ldx ff,x
	IND8, Y	ldx ff,y
	IND8, Z	ldx ff,z
	IMM16	ldx #jjkk
	IND16, X	ldx gggg,x
	IND16, Y	ldx gggg,y
	IND16, Z	ldx gggg,z
	EXT	ldx hhll
LDY	IND8, X	ldy ff,x
	IND8, Y	ldy ff,y
	IND8, Z	ldy ff,z
	IMM16	ldy #jjkk
	IND16, X	ldy gggg,x
	IND16, Y	ldy gggg,y
	IND16, Z	ldy gggg,z
	EXT	ldy hhll
LDZ	IND8, X	ldz ff,x
	IND8, Y	ldz ff,y
	IND8, Z	ldz ff,z
	IMM16	ldz #jjkk
	IND16, X	ldz gggg,x
	IND16, Y	ldz gggg,y
	IND16, Z	ldz gggg,z
	EXT	ldz hhll
LPSTOP	INH	lpstop
LSL	IND8, X	lsl ff,x
	IND8, Y	lsl ff,y
	IND8, Z	lsl ff,z
	IND16, X	lsl gggg,x
	IND16, Y	lsl gggg,y
	IND16, Z	lsl gggg,z
	EXT	lsl hhll
LSLA	INH	lsla
LSLB	INH	lslb

Name	Mode	Syntax
LSLD	INH	lsld
LSLE	INH	lsle
LSLM	INH	lslm
LSLW	IND16, X	lslw gggg,x
	IND16, Y	lslw gggg,y
	IND16, Z	lslw gggg,z
	EXT	lslw hhll
LSR	IND8, X	lsr ff,x
	IND8, Y	lsr ff,y
	IND8, Z	lsr ff,z
	IND16, X	lsr gggg,x
	IND16, Y	lsr gggg,y
	IND16, Z	lsr gggg,z
	EXT	lsr hhll
LSRA	INH	lsra
LSRB	INH	lsrb
LSRD	INH	lsrd
LSRE	INH	lsre
LSRW	IND16, X	lsrw gggg,y
	IND16, Y	lsrw gggg,y
	IND16, Z	lsrw gggg,z
	EXT	lsrw hhll
MAC	IMM8	mac xo,yo
MOVB	IXP to EXT	movb ff,x,hhll
	EXT to IXP	movb hhll,ff,x
	EXT to EXT	movb hhll,hhll
MOVW	IXP to EXT	movw ff,x,hhll
	EXT to IXP	movw hhll,ff,x
	EXT to EXT	movw hhll,hhll
MUL	INH	mul
NEG	IND8, X	neg ff,x
	IND8, Y	neg ff,y
	IND8, Z	neg ff,z
	IND16, X	neg gggg,x
	IND16, Y	neg gggg,y
	IND16, Z	neg gggg,z
	EXT	neg hhll
NEGA	INH	nega
NEGB	INH	negb
NEGD	INH	negd
NEGE	INH	nege
NEGW	IND16, X	negw gggg,x
	IND16, Y	negw gggg,y
	IND16, Z	negw gggg,z
	EXT	negw hhll
NOP	INH	nop
ORAA	IND8, X	oraa ff,x
	IND8, Y	oraa ff,y
	IND8, Z	oraa ff,z
	IMM8	oraa #ii
	IND16, X	oraa gggg,x
	IND16, Y	oraa gggg,y
	IND16, Z	oraa gggg,z
	EXT	oraa hhll
	E, X	oraa e,x
	E, Y	oraa e,y
	E, Z	oraa e,z
ORAB	IND8, X	orab ff,x
	IND8, Y	orab ff,y
	IND8, Z	orab ff,z
	IMM8	orab #ii
	IND16, X	orab gggg,x
	IND16, Y	orab gggg,y
	IND16, Z	orab gggg,z
	EXT	orab hhll
	E, X	orab e,x
	E, Y	orab e,y
	E, Z	orab e,z
ORD	IND8, X	ord ff,x
	IND8, Y	ord ff,y
	IND8, Z	ord ff,z
	IMM16	ord #jjkk
	IND16, X	ord gggg,x
	IND16, Y	ord gggg,y
	IND16, Z	ord gggg,z
	EXT	ord hhll
	E, X	ord e,x
	E, Y	ord e,y
	E, Z	ord e,z

Name	Mode	Syntax
ORE	IMM16	ore #jjkk
	IND16, X	ore gggg,x
	IND16, Y	ore gggg,y
	IND16, Z	ore gggg,z
	EXT	ore hhll
ORP	IMM16	orp #jjkk
PSHA	INH	psha
PSHB	INH	pshb
PSHM	IMM8	pshm d,e,x,y,z,k,ccr
PSHMAC	INH	pshmac
PULA	INH	pula
PULB	INH	pulb
PULM	IMM8	pulm d,e,x,y,z,k,ccr
PULMAC	INH	pulmac
RMAC	IMM8	rmac xo,yo
ROL	IND8, X	rol ff,x
	IND8, Y	rol ff,y
	IND8, Z	rol ff,z
	IND16, X	rol gggg,x
	IND16, Y	rol gggg,y
	IND16, Z	rol gggg,z
	EXT	rol hhll
ROLA	INH	rola
ROLB	INH	rolb
ROLD	INH	rold
ROLE	INH	role
ROLW	IND16, X	rolw gggg,x
	IND16, Y	rolw gggg,y
	IND16, Z	rolw gggg,z
	EXT	rolw hhll
ROR	IND8, X	ror ff,x
	IND8, Y	ror ff,y
	IND8, Z	ror ff,z
	IND16, X	ror gggg,x
	IND16, Y	ror gggg,y
	IND16, Z	ror gggg,z
	EXT	ror hhll
RORA	INH	rora
RORB	INH	rorb
RORD	INH	rord
RORE	INH	rore

Name	Mode	Syntax
RORW	IND16, X	rorw gggg,x
	IND16, Y	rorw gggg,y
	IND16, Z	rorw gggg.z
	EXT	rorw hhll
RTI	INH	rti
RTS	INH	rts
SBA	INH	sba
SBCA	IND8, X	sbca ff,x
	IND8, Y	sbca ff,y
	IND8, Z	sbca ff,z
	IMM8	sbca #ii
	IND16, X	sbca gggg,x
	IND16, Y	sbca gggg,y
	IND16, Z	sbca gggg,z
	EXT	sbca hhll
	E, X	sbca e,x
	E, Y	sbca e,y
	E, Z	sbca e,z
SBCB	IND8, X	sbcb ff,x
	IND8, Y	sbcb ff,y
	IND8, Z	sbcb ff,z
	IMM8	sbcb #ii
	IND16, X	sbcb gggg,x
	IND16, Y	sbcb gggg,y
	IND16, Z	sbcb gggg,z
	EXT	sbcb hhll
	E, X	sbcb e,x
	E, Y	sbcb e,y
	E, Z	sbcb e,z
SBCD	IND8, X	sbcd ff,x
	IND8, Y	sbcd ff,y
	IND8, Z	sbcd ff,z
	IMM16	sbcd #jjkk
	IND16, X	sbcd gggg,x
	IND16, Y	sbcd gggg,y
	IND16, Z	sbcd gggg,z
	EXT	sbcd hhll
	E, X	sbcd e,x
	E, Y	sbcd e,y
	E, Z	sbcd e,z

Name	Mode	Syntax
SBCE	IMM16	sbce #jjkk
	IND16, X	sbce gggg,x
	IND16, Y	sbce gggg,y
	IND16, Z	sbce gggg,z
	EXT	sbce hhll
SDE	INH	sde
STAA	IND8, X	staa ff,x
	IND8, Y	staa ff,y
	IND8, Z	staa ff,z
	IND16, X	staa gggg,x
	IND16, Y	staa gggg,y
	IND16, Z	staa gggg,z
	EXT	staa hhll
	E, X	staa e,x
	E, Y	staa e,y
	E, Z	staa e,z
STAB	IND8, X	stab ff,x
	IND8, Y	stab ff,y
	IND8, Z	stab ff,z
	IND16, X	stab gggg,x
	IND16, Y	stab gggg,y
	IND16, Z	stab gggg,z
	EXT	stab hhll
	E, X	stab e,x
	E, Y	stab e,y
	E, Z	stab e,z
STD	IND8, X	std ff,x
	IND8, Y	std ff,y
	IND8, Z	std ff,z
	IND16, X	std gggg,x
	IND16, Y	std gggg,y
	IND16, Z	std gggg,z
	EXT	std hhll
	E, X	std e,x
	E, Y	std e,y
	E, Z	std e,z
STE	IND16, X	ste gggg,x
	IND16, Y	ste gggg,y
	IND16, Z	ste gggg,z
	EXT	ste hhll
STED	EXT	sted hhll

Name	Mode	Syntax
STS	IND8, X	sts ff,x
	IND8, Y	sts ff,y
	IND8, Z	sts ff,z
	IND16, X	sts gggg,x
	IND16, Y	sts gggg,y
	IND16, Z	sts gggg,z
	EXT	sts hhll
STX	IND8, X	stx ff,x
	IND8, Y	stx ff,y
	IND8, Z	stx ff,z
	IND16, X	stx gggg,x
	IND16, Y	stx gggg,y
	IND16, Z	stx gggg,z
	EXT	stx hhll
STY	IND8, X	sty ff,x
	IND8, Y	sty ff,y
	IND8, Z	sty ff,z
	IND16, X	sty gggg,x
	IND16, Y	sty gggg,y
	IND16, Z	sty gggg,z
	EXT	sty hhll
STZ	IND8, X	stz ff,x
	IND8, Y	stz ff,y
	IND8, Z	stz ff,z
	IND16, X	stz gggg,x
	IND16, Y	stz gggg,y
	IND16, Z	stz gggg,z
	EXT	stz hhll
SUBA	IND8, X	suba ff,x
	IND8, Y	suba ff,y
	IND8, Z	suba ff,z
	IMM8	suba #ii
	IND16, X	suba gggg,x
	IND16, Y	suba gggg,y
	IND16, Z	suba gggg,z
	EXT	suba hhll
	E, X	suba e,x
	E, Y	suba e,y
	E, Z	suba e,z

Name	Mode	Syntax
SUBB	IND8, X	subb ff,x
	IND8, Y	subb ff,y
	IND8, Z	subb ff,z
	IMM8	subb #ii
	IND16, X	subb gggg,x
	IND16, Y	subb gggg,y
	IND16, Z	subb gggg,z
	EXT	subb hhll
	E, X	subb e,x
	E, Y	subb e,y
	E, Z	subb e,z
SUBD	IND8, X	subd ff,x
	IND8, Y	subd ff,y
	IND8, Z	subd ff,z
	IMM16	subd #jjkk
	IND16, X	subd gggg,x
	IND16, Y	subd gggg,y
	IND16, Z	subd gggg,z
	EXT	subd hhll
	E, X	subd e,x
	E, Y	subd e,y
	E, Z	subd e,z
SUBE	IMM16	sube #jjkk
	IND16, X	sube gggg,x
	IND16, Y	sube gggg,y
	IND16, Z	sube gggg,z
	EXT	sube hhll
SWI	INH	swi
SXT	INH	sxt
TAB	INH	tab
TAP	INH	tap
TBA	INH	tba
TBEK	INH	tbek
TBSK	INH	tbsk
TBXK	INH	tbxk
TBYK	INH	tbyk
TBZK	INH	tbzk
TDE	INH	tde
TDMSK	INH	tdmsk
TDP	INH	tdp
TED	INH	ted
TEDM	INH	tedm

Name	Mode	Syntax
TEKB	INH	tekb
TEM	INH	tem
TMER	INH	tmer
TMET	INH	tmet
TMXED	INH	tmxed
TPA	INH	tpa
TPD	INH	tpd
TSKB	INH	tskb
TST	IND8, X	tst ff,x
	IND8, Y	tst ff,y
	IND8, Z	tst ff,z
	IND16, X	tst gggg,x
	IND16, Y	tst gggg,y
	IND16, Z	tst gggg,z
	EXT	tst hhll
TSTA	INH	tsta
TSTB	INH	tstb
TSTD	INH	tstd
TSTE	INH	tste
TSTW	IND16, X	tstw ff,x
	IND16, Y	tstw ff,y
	IND16, Z	tstw ff,z
	EXT	tstw hhll
TSX	INH	tsx
TSY	INH	tsy
TSZ	INH	tsz
TXKB	INH	txkb
TXS	INH	txs
TXY	INH	txy
TXZ	INH	txz
TYKB	INH	tykb
TYS	INH	tys
TYX	INH	tyx
TYZ	INH	tyz
TZKB	INH	tzkb
TZS	INH	tzs
TZX	INH	tzx
TZY	INH	tzy
WAI	INH	wai
XGAB	INH	xgab
XGDE	INH	xgde
XGDX	INH	xgdx

Name	Mode	Syntax
XGDY	INH	xgdy
XGDZ	INH	xgdz
XGEX	INH	xgex
XGEY	INH	xgey
XGEZ	INH	xgez

INDEX