CPU S12Z

Reference Manual

Linear S12Z Microcontrollers

CPUS12ZRM
Rev. 1.01
01/2013

freescale.com

Table 0-1. Revision History

Rev	Date	Author	Description
1.00	10 Oct 2012		Initial release.
1.01	24 Jan 2013		Fixed typos and grammar errors throughout the document.

Chapter 1
Introduction

Chapter 2
Overview

Chapter 3
Addressing Modes

Chapter 4
Instruction Queue

Chapter 5
Instruction Set Overview

Chapter 6
Instruction Glossary

Chapter 7
Exceptions

Chapter 8
Instruction Execution Timing

Chapter 9
Data Bus Operation

Appendix A
Instruction Reference

Chapter 1
Introduction

1.1 Introduction to S12Z CPU

This manual describes the features and operation of the central processing unit, or S12Z CPU, used in HCS12Z microcontrollers. 68HC12, HCS12, HCS12X, and HCS12Z represent four generations of 16-bit controllers with all of them being derived from the industry standard M68HC11. The M68HC11 was, in turn, derived from the M6801 which was derived from the M6800. The M6800 was the first 8-bit MPU introduced by Motorola in 1974. Detailed information for the M68HC12 is provided in the *CPU12RM/AD Rev. 3*. Detailed information for the HCS12 and HCS12X is provided in the *S12XCPU Rev. 2*. This document covers the S12Z CPU.

There have been many changes in the years since the M6800 was introduced in 1974. Process technology has changed dramatically from early 6-micron NMOS (M6800), to 0.18 micron CMOS (S12X), and now 0.18 micron or smaller CMOS (S12Z). As chip complexity and memory size have grown, software development tools have also changed. M6800 application programs were on the order of a few kilobytes written in assembly language. S12X and S12Z application programs are hundreds of kilobytes and are written in C. This has shifted some of the burden of compatibility from absolute object code compatibility in the CPU itself to compatibility in the compiler and development tool chain.

The S12Z CPU has taken advantage of this reduced need for absolute object code compatibility to focus on improved support for C code-size efficiency and overall performance. The most obvious change has been to eliminate the paged memory model and 64-kilobyte CPU addressing limitation of the CPU12 in favor of a linear 16-megabyte address space. The X and Y index registers, as well as the stack pointer (SP) and program counter (PC), were expanded from 16 bits to 24 bits to match the width of the address bus. The next biggest change has been to replace the 8-bit A and B accumulators (sometimes used as the 16-bit D accumulator), with a set of eight general purpose data registers (D_i). D0 and D1 are 8 bits, D2 through D5 are 16 bits, and D6 and D7 are 32 bits.

As in previous generations of CPU12, the S12Z CPU has variable-length instructions ranging from a single byte to several bytes. The longest instructions in the CPU12 were moves with two extended addressing mode operand addresses (6 bytes of object code). Moves could have indexed addressing mode operands, but only indexed modes that did not require additional extension bytes. The S12Z CPU allows complete flexibility in specifying the addresses for move instructions so if both operands use an indexed postbyte plus three extension bytes, these instructions can take up to nine bytes of object code. The longest instructions in the S12Z CPU are the most complex math instructions (DIV, MAC, and MOD) which are page 2 opcodes plus a math postbyte plus two operand addresses which could each be an addressing mode postbyte plus 3 extension bytes (11 bytes total).

The CPU12 used postbytes for indexed addressing, transfer/exchange, and looping primitive instructions. The S12Z CPU instruction set has expanded the use of postbytes to improve code-size efficiency. The

indexed postbyte was re-worked into a general operand (OPR) addressing system. This new addressing mode postbyte includes indexed addressing modes like the CPU12 and extended addressing modes, a quick-immediate mode, and register-as-memory addressing mode.

In addition to this general OPR addressing postbyte, the S12Z instruction set uses postbytes for transfer/exchange, looping primitives, math (MUL, DIV, MAC, and MOD), relative addressing, shifts, bit-field instructions, and push/pull.

1.2 Features

The S12Z CPU is the next generation of CPU in the CPU12 line. This high-speed 16-bit processor has an expanded programmers model with 24-bit wide X, Y, SP, and PC registers and replaces the A, B, and D accumulators with a set of eight general purpose registers D_i. Improved addressing modes support efficient use of the 16-megabyte (24-bit) linear address space.

- 24-Bit Linear Address Space (16-megabytes)
- 24-Bit Index Registers (X and Y), Stack Pointer (SP), and Program Counter (PC)
- Eight General Purpose Data Registers (D0, D1 8-Bits; D2–D5 16-Bits; D6, D7 32-Bits)
- Separate Memory Access Controllers for Code and Data
- Variable-Length Instructions Including Single-Byte and Odd Number of Bytes
- Extensive Use of Instruction Postbytes to Optimize Code-Size Efficiency

1.3 Symbols and Notation

The symbols and notation used throughout this manual are described in this section.

1.3.1 Source form notation

Everything in the source forms columns, *except expressions in italic characters,* is literal information that must appear in the assembly source file as shown. The initial 3- to 5-letter mnemonic is a literal expression (not case-sensitive). All commas, periods, pound (#), and parentheses are literal characters.

Red italic expressions represent variable content such as register names, program labels, and expressions. Explanations are shown in this key.

bwplbwpl	— Any of the characters B, W, P, L, or 2-letter pairs BB, BW, BP, BL, WB, WW….LB, LW, LP, or LL to indicate the sizes for an instruction with two input operands. B=byte, W=16-bit word, P=24-bit pointer, L=32-bit long-word. The two-letter codes allow the size of each operand to be specified separately and the one-letter codes indicate the same size is used for both input operands.
bwl	— Any of the characters B, W, or L to indicate the size of the operation. B=byte, W=16-bit word, L=32-bit long-word
bwpl	— Any of the characters B, W, P, or L to indicate the size of the operation. B=byte, W=16-bit word, P=24-bit pointer, L=32-bit long-word
cc	— Branching condition (EQ, NE, MI, PL, GT, or LE) for loop instructions test-and branch (TBcc) or decrement and branch (DBcc). Branch if… EQ - equal; NE - not equal; MI - minus; PL - plus; GT - greater than; LE - less than or equal
cpureg	— Any of the CPU registers D0, D1, D2, D3, D4, D5, D6, D7, X, Y, SP, CCH, CCL, or CCW. Used for transfer and exchange instructions.
Di	— Any of the eight CPU data registers D2, D3, D4, D5, D0, D1, D6, or D7.
Dj	— Typically used for a second operand.
Dk	— Used for a third operand in MAC, MOD, MUL, and DIV instructions.
Ds	— Used for a source operand.
Dd	— Used for a destination operand.
Dn	— Used for a numeric control parameter such as the number of positions to shift.
Dp	— Any of the four 16-bit CPU data registers D2, D3, D4, or D5. Used to specify the width and offset parameters in bit field instructions BFEXT and BFINS.
opr1i opr5i	— Any label or expression that evaluates to a 1-bit (5-bit) immediate operand. Used to specify number of shifts for shift and rotate instructions. Immediate value is encoded in the shift postbytes (sb) or (sb+xb).
opr8i	— Any label or expression that evaluates to an 8-bit immediate operand.
opr16i	— Any label or expression that evaluates to a 16-bit immediate operand.
opr18i	— Any label or expression that evaluates to an 18-bit immediate operand. Two bits of the 18-bit operand are encoded into the opcode. The value is zero-extended and placed in X or Y.
opr24	— A 24-bit address which can be considered signed or unsigned.
opr24a	— A 24-bit address.
opr24i	— A 24-bit immediate constant.
opr24u	— A 24-bit unsigned constant offset.
opr32i	— Any label or expression that evaluates to a 32-bit immediate operand.
oprdest	— Any label or expression that evaluates to an address within +127/–128 or +/–16K from the current location. Used for 7-bit or 15-bit relative branches.
oprimmsz	— Any label or expression that evaluates to an immediate operand of the same size as the CPU register involved in the instruction (8, 16, or 32 bits).

oprmemreg — Refer to the OPR addressing summary to see how to expand this into the operand specification for 1 of 16 OPR addressing modes (allowed forms and brief description shown here below).

- *#oprsxe4i* — Short Immediate. *oprsxe4i* is any label or expression which evaluates to one of the values -1, 1, 2, 3...14, or 15. Auto sign-extended to 8, 16, 24, or 32 bits.
- Di — Register as operand. *Di* is any one of the eight CPU data registers D0, D1, D2, D3, D4, D5, D6, or D7.
- (*opru4*,*xys*) — Short offset (0-15) from X, Y, or S. *opru4* is any label or expression that evaluates to unsigned 0-15.
- (+*xy*) | (*xy*+) | (–*xy*) | (*xy*–) | (–S) | (S+) — Auto pre/post inc/dec from X, Y, or S (S=SP).
  Where *xy* is either of the two index register names X or Y.
- (*Di*,*xys*) — Register offset from X, Y, or S. *xys* is any one of the 24-bit indexing registers X, Y, or S (S=SP). 16-bit D2, D3, D4, D5 treated as signed, D0, D1, D6, D7 treated as unsigned.
- [*Di*,*xy*] — Register offset from X or Y Indirect. D2, D3, D4, D5 treated as signed, D0, D1, D6, D7 are unsigned.
- (*oprs9*,*xysp*) — 9-bit signed offset from X, Y, S, or P. *oprs9* is any label or expression that evaluates to a 9-bit signed value from –256 to +256. (0 is treated as +256) *xysp* is any one of the 24-bit registers X, Y, S or P (S=SP P=PC).
- [*oprs9*,*xysp*] — 9-bit signed offset from X, Y, S, or P Indirect.
- opru14 — Short Extended (16K). *opru14* is any label or expression that evaluates to a 14-bit unsigned address from $000000 through $003FFF. All registers and 12K of RAM.
- (*opru18*,*Di*) — 18-bit unsigned offset from Di. *opr18* is any label or expression that evaluates to an 18-bit unsigned value from $000000 through $03FFFF (256K).
- opru18 — Medium Extended (256K). Reaches any address from $000000 to $03FFFF. All on-chip RAM.
- (*opr24*,*xysp*) — 24-bit offset from X, Y, S, or P. *opr24* is any label or expression that evaluates to a 24-bit value (16M).
- [*opr24*,*xysp*] — 24-bit offset from X, Y, S, or P Indirect.
- (*opru24*,*Di*) — 24-bit offset from Di. Can also be considered as a register offset from any 16M address or label.
- opr24 — Long Extended (16M). Reaches any address in the full 16M memory space.
- [*opr24*] — 24-bit address Indirect.

oprregs1 — Any combination of the CPU registers in the list (CCH, CCL, D0, D1, D2, D3) separated by commas. Used with the PSH and PUL instructions.

oprregs2 — Any combination of the CPU registers in the list (D4, D5, D6, D7, X, Y) separated by commas. Used with the PSH and PUL instructions.

oprs9 — Any label or expression that evaluates to a 9-bit signed value from –256 to +256. (0 is treated as +256)

oprsxe4i — Any label or expression which evaluates to one of the values -1, 1, 2, 3...14, or 15. Auto sign-extended to 8, 16, 24, or 32 bits.

opru4 — Any label or expression that evaluates to the unsigned values 0 through 15.

opru14 — Any label or expression that evaluates to a 14-bit unsigned address from $000000 through $003FFF. All registers and 12K of RAM.

opru18 — Any label or expression that evaluates to an 18-bit unsigned value from $000000 through $03FFFF (256K).

trapnum — Any label or expression that evaluates to the code for one of the unused opcodes on pg2 of the opcode map. Valid values are 0x92..0x9F, 0xA8..0xAF, 0xB8..0xBF and 0xC0..0xFF.

width:offset — Any label or expression that evaluates to a 10-bit immediate operand. Used to specify field width and offset w:o for bit field instructions where w and o are each 5-bit values (w=0 treated as 32).

xy — One of the two index register names X or Y.

xys — Any one of the 24-bit indexing registers X, Y, or S (S=SP).

xysp — Any one of the 24-bit registers X, Y, S or P (S=SP P=PC).

1.3.2 Operators

- + — Add
- – — Subtract or negate (two's complement)
- ∗ — Multiply
- / — Divide
- |expression| — Absolute value of the expression shown between vertical bars

& — Boolean AND
| — Boolean OR
^ — Boolean exclusive-OR
~ — Invert (One's complement)
() — Contents of register or memory location shown inside parentheses
: — Concatenate
⇒ — Result of the operation on the left goes to...
⇔ — Exchange

1.3.3 CPU registers

The eight CPU data registers D0–D7 are referred to using various subscripts to help clarify the way these registers are used in different instructions. Some instructions use two or even three CPU data registers. In a few cases such as SWI and RTI instructions, these registers are shown as D0, D1, D2H:D2L, D3H:D3L, D4H:D4L, D5H:D5L, D6H:D6MH:D6ML:D6L, and D7H:D7MH:D7ML:D7L because it is important to show the order that bytes are used. In all cases except D_p you may substitute any of the register numbers 0–7 in place of the subscript. In the case of D_p you are limited to the four 16-bit registers because the register is used for a 10-bit parameter.

D_i — Any of the eight CPU data registers D0—D7.
D_j — Any of the eight CPU data registers D0—D7. Used for a second operand.
D_k — Any of the eight CPU data registers D0—D7. Used for a third operand.
D_n — Any of the eight CPU data registers D0—D7. Used to specify an instruction parameter such as a bit number n or a number of bit positions for a shift.
D_s — Any of the eight CPU data registers D0—D7. Used for a source operand.
D_d — Any of the eight CPU data registers D0—D7. Used for a destination operand.
D_p — Any of the four 16-bit CPU data registers D2—D5. Used to specify the width and offset parameters for BFEXT and BFINS. Low-order 10-bits used for w:o parameters.
X — 24-bit index register X, Sometimes shown as XH:XM:XL
Y — 24-bit index register Y, Sometimes shown as YH:YM:YL
SP — 24-bit stack pointer, Sometimes shown as SPH:SPM:SPL
PC — 24-bit program counter, Sometimes shown as PCH:PCM:PCL
RTNH:RTNM:RTNL — 24-bit return address which will become the program counter when program execution resumes after a return from interrupt (RTI).
CCH — High-order 8 bits of the condition code register
CCL — Low-order 8 bits of the condition code register which hold CPU status flags
CCR — Condition code register, also known as CCW and CCH:CCL
CCW — Full 16-bit condition code register made up of CCH:CCL

1.3.4 Memory and addressing

M — A memory location or immediate data. The size of M is the same as the size of the operation and generally matches the size of a CPU data register that is used for the operation result or destination. In some cases the size of the operation is indicated by a suffix (.B, .W, .P, or .L) after the instruction mnemonic.
M1, M2 — Numbered memory operands for instructions that require more than one memory operand. M1 and M2 use separate addressing modes to specify each of these operands.

MS, MD — Source and Destination memory operands. MS and MD use separate addressing modes to specify each of these operands.

M(SP) — The memory location pointed-to by the stack pointer. Similarly, the notation M(SP):M(SP+1):M(SP+2) indicates the three memory bytes at address=SP, address=SP+1, and address=SP+2.

M:M+1:M+2 — A 24-bit value in three consecutive memory locations. The higher-order (most significant) 8 bits are located at address=M, and the next two 8-bit values are located at the next higher sequential addresses.

$ — This prefix indicates a hexadecimal value

% — This prefix indicates a binary value

1.3.5 Condition code register (CCR) bits

U — User/Supervisor state status/control

IPL — Interrupt Priority Level

S — Stop mode enable

X — X Interrupt mask (pseudo non-maskable interrupt)

I — I Interrupt mask

N — Negative status flag

Z — Zero status flag

V — Two's complement overflow status flag

C — Carry/borrow status flag

1.3.6 Address mode notation

EXT24 — Long extended (16M). The 24-bit address of the operand is provided in three bytes (`a3 a2 a1`) after the LD, ST, JMP, or JSR opcode. (more efficient than the EXT3 option in the OPR3 addressing mode)

IMM — Immediate. A parameter for the instruction is supplied as immediate data in the object code for the instruction.

IMM1 — Immediate (1-byte). The 8-bit operand is located in one byte (`i1`) of immediate data in the object code for the instruction.

IMM2 — Immediate (2-byte). The 16-bit operand is located in two bytes (`i2 i1`) of immediate data in the object code for the instruction.

IMM3 — Immediate (3-byte). The 24-bit operand is located in three bytes (`i3 i2 i1`) of immediate data in the object code for the instruction.

IMM4 — Immediate (4-byte). The 32-bit operand is located in four bytes (`i4 i3 i2 i1`) of immediate data in the object code for the instruction.

INH — Inherent. All operands are implied in the instruction mnemonic (and any dot suffix such as .B or .Di).

OPR or OP — Common operand addressing (`xb`) with no extension bytes. Expands to IMMe4, REG, [REG], IDX, ++IDX, REG,IDX, or [REG,IDX] addressing modes. See Operand Addressing Summary explanation.

OPR1 or OP1 — Common operand addressing (xb) w/ one extension byte (`x1`). Expands to IDX1, [IDX1], or EXT1 addressing modes. See Operand Addressing Summary explanation.

OPR2 or OP2 — Common operand addressing (`xb`) w/ two extension bytes (`x2 x1`). Expands to IDX2,REG or EXT2 addressing modes. See Operand Addressing Summary explanation.

OPR3 or OP3 — Common operand addressing (`xb`) w/ three extension bytes (`x3 x2 x1`). Expands to IDX3, [IDX3], IDX3,REG, EXT3, or [EXT3] addressing modes. See Operand Addressing Summary explanation.

REG or RG — Register inherent. The operand(s) are in CPU data registers Di.

R7 — Short relative branch offset. The `rb` postbyte includes a mode indicator (7-bit mode) and 7 bits of offset. This allows a branch distance of –64 to +63 locations from the current PC location.

R15 — Long relative branch offset. The `rb` postbyte includes a mode indicator (15-bit mode) and the high-order 7 bits of the 15-bit offset. The low-order 8 bits of the 15-bit offset are included in one extension byte `r1`. This allows a branch distance of –16K to +16K from the current PC location.

1.3.7 Machine coding notation

Each pair of characters in the machine coding column represent one byte of object code.

12 3A CF	—	Literal hexadecimal values are expressed as a pair of characters including any combination of the numbers 0-9 and uppercase A-F.
5p	—	One hexadecimal digit followed by lowercase p indicates 2 or more opcodes corresponding to 2 or more registers of the same size or .B/.W/.P/.L variations. Refer to the opcode map to find specific opcodes.
6n	—	One hexadecimal digit followed by lowercase n indicates a range of 8 opcodes corresponding to the 8 registers. D2=`0`, D3=`1`, D4=`2`, D5=`3`, D0=`4`, D1=`5`, D6=`6`, D7=`7`.
6q	—	One hexadecimal digit followed by lowercase q indicates a range of 8 opcodes corresponding to the 8 registers. D2=`8`, D3=`9`, D4=`A`, D5=`B`, D0=`C`, D1=`D`, D6=`E`, D7=`F`.
a3 a2 a1	—	Lowercase a followed by 1, 2, or 3 in this sequence indicates a 3-byte 24-bit address. Used only with EXT3 versions of load, store, jump, and JSR.
bb	—	Postbyte `bb` for bit field extract and insert instructions BFEXT and BFINS. See tables and explanation for coding of this postbyte.
bm	—	Postbyte `bm` for bit manipulation instructions BCLR, BSET, BTGL, BRCLR, and BRSET. See tables and explanation for coding of this postbyte.
eb	—	Postbyte `eb` for exchange and sign-extend instructions EXG and SEX. See tables and explanation for coding of this postbyte.
i4 i3 i2 i1	—	Extension bytes for immediate addressing. These bytes form an 8-, 16-, 24-, or 32-bit immediate value.
lb	—	Postbyte `lb` for loop instructions DBcc and TBcc. See tables and explanation for coding of this postbyte.
mb	—	Postbyte `mb` for math instructions DIVS, DIVU, MACS, MACU, MODS, MODU, MULS, and MULU. See tables and explanation for coding of this postbyte.
op	—	Used only for LD X and LD Y where 2 bits of an 18-bit immediate value are encoded in the opcode so 4 opcodes are used for each of these two instructions.
r1	—	Low order 8 bits of a 15-bit signed relative offset.
rb	—	Postbyte `rb` for relative branch instructions. If the MSB is 0, the 7-bit signed relative offset is in rb[6:0]. If the MSB is 1, the 15-bit offset is in rb[6:0]:r1[7:0].
sb	—	Postbyte `sb` for shift and rotate instructions ASL, ASR, LSL, LSR, ROL, and ROR. See tables and explanation for coding of this postbyte.
tb	—	Postbyte `tb` for transfer and zero-extend instructions TFR and ZEX. See tables and explanation for coding of this postbyte.
x3 x2 x1	—	Extension bytes following the `xb` postbyte. There are 0, 1, 2, or 3 8-bit extension bytes after each `xb` postbyte.
xb	—	Postbyte `xb` for general operand (OPR) addressing. This code selects 1 of 16 more detailed addressing modes to identify operands. See tables and explanation for coding of this postbyte.

1.3.8 CCR activity notation

–	—	Bit not affected
0	—	Bit forced to 0
1	—	Bit forced to 1
Δ	—	Bit set or cleared according to results of the operation
⇓	—	Bit may change from 1 to 0 or remain unchanged as a result of the operation
⇑	—	Bit may change from 0 to 1 or remain unchanged as a result of the operation
c	—	Bit may be changed if the destination register in an EXG or TFR instruction is CCL, CCW, or CCR.
s	—	Bit can only be changed if CPU is in supervisor state
v	—	Bit will be set or remain unchanged depending on the source of the related interrupt

1.3.9 Definitions

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

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

Set refers specifically to establishing logic level 1 on a bit or bits.

Cleared refers specifically to establishing logic level 0 on a bit or bits.

Asserted means that a signal is in active logic state. An active low signal changes from logic level 1 to logic level 0 when asserted, and an active high signal changes from logic level 0 to logic level 1.

Negated means that an asserted signal changes logic state. An active low signal changes from logic level 0 to logic level 1 when negated, and an active high signal changes from logic level 1 to logic level 0.

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.

LSW means least significant word or words.

MSW means most significant word or words.

A range of bit locations is referred to by mnemonic and the numbers that define the range. For example, DATA[15:8] form the high byte of the data bus.

Chapter 2
Overview

2.1 Introduction

This section describes the S12Z CPU programmer's model, register set, data types used, and basic memory organization.

2.2 Programmer's Model and CPU Registers

Figure 2-1 shows the S12Z CPU registers. CPU registers are not part of the memory map.

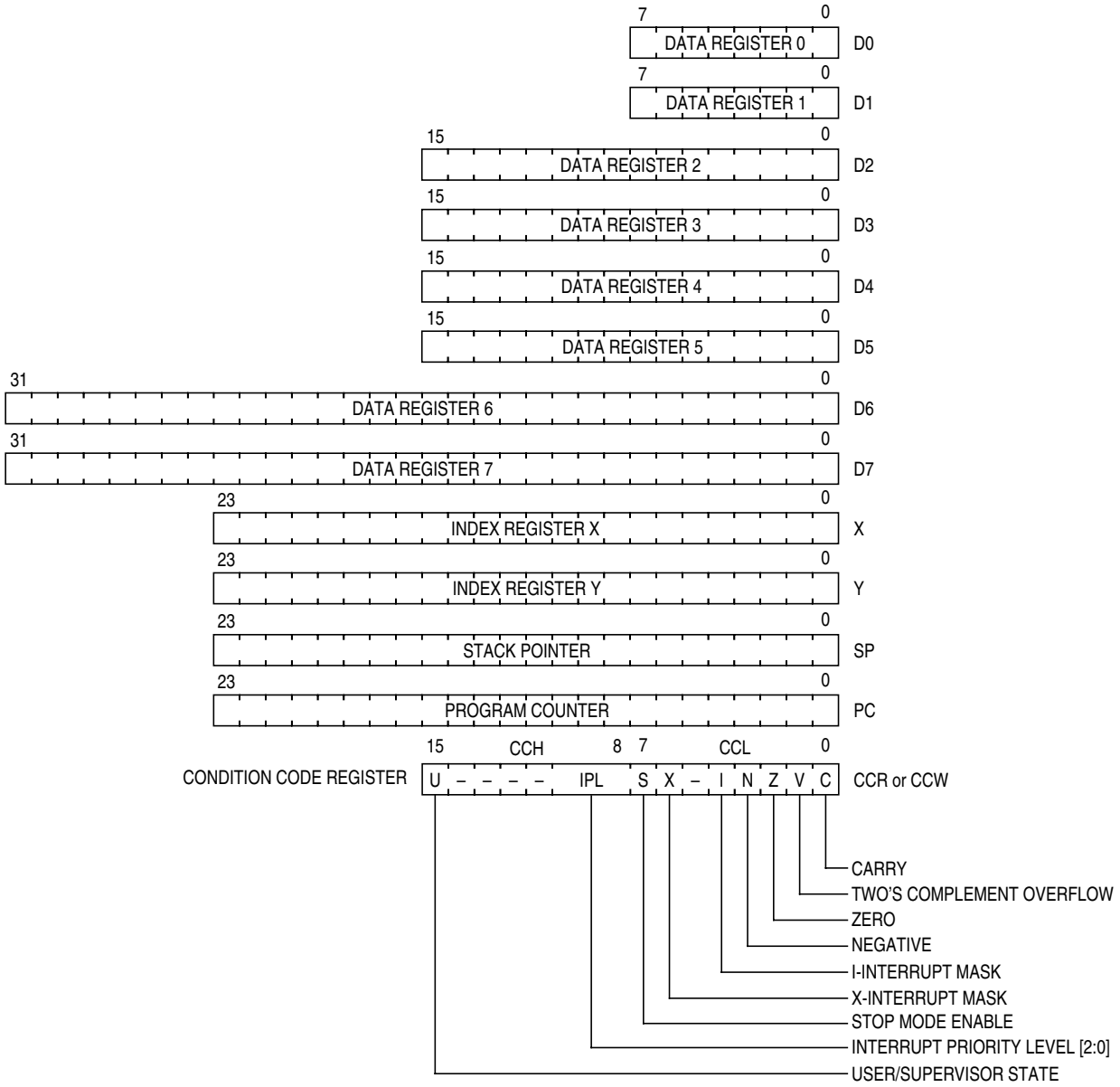


Figure 2-1. CPU Registers

2.2.1 General Purpose Data Registers (D_i)

The linear S12Z CPU includes 8 general purpose data registers. D0 and D1 are 8 bits, D2–D5 are 16 bits, and D6 and D7 are 32 bits. Normally, instructions that use these general purpose registers allow the programmer to specify any of the 8 data registers. The most common use for these general purpose data registers is to hold operands or results for instructions.

There are load effective address instructions for D6 and D7, and indexed addressing sub modes that allow an 18-bit or 24-bit constant offset from a data register D_i. This helps in programming situations where more than two index/pointer registers are needed.

Bit-field instructions use a 5-bit value to specify the width of the field to operate on and a 5-bit value to specify the offset (starting bit number) of the field to be operated on. There are variations of these instructions that allow these two 5-bit values to be supplied in one of the four 16-bit data registers D2~D5.

2.2.2 Index Registers (X, Y)

These two 24-bit registers are used as pointers into memory and as index registers for the indexed addressing modes. These registers have the same number of bits as the address bus so they can point to any memory location in the entire 16-megabyte 24-bit address space. Many of the instructions in the S12Z CPU support the 24-bit “pointer” size using a .P suffix as in CLR.P or MOV.P.

2.2.3 Stack Pointer (SP)

This 24-bit address pointer register points at the most-recently-used location on the automatic last-in-first-out (LIFO) stack. The stack may be located anywhere in the 16-megabyte address space that has RAM, and can be any size up to the amount of available RAM. The stack is used to automatically save the return address for subroutine calls, the return address and CPU registers during interrupts, and for local variables. LEA S instructions allow simple arithmetic to be performed directly on the stack pointer value to allocate or deallocate space for local variables on the stack.

The stack pointer is not affected by reset so a program must initialize SP before any interrupts or function (subroutine) calls. During program execution, SP points at the most-recently-used location on the stack. When responding to an interrupt or stacking the return address for a function call, SP is decremented so it points at the next free location on the stack before storing the first piece of information on the stack. You would typically initialize SP to point one location above the top of the RAM area for the stack to compensate for this pre-decrement behavior.

2.2.4 Program Counter (PC)

The program counter is a 24-bit register that contains the address of the next byte of object code to be processed. The actual memory read that fetched this byte into the instruction queue of the CPU occurs a few bus cycles before it is executed.

During normal program execution, the program counter automatically increments to the next sequential memory location after each instruction is executed. Jump, branch, interrupt, and return operations load the program counter with an address other than that of the next sequential location. This is called a change-of-flow or COF.

For instructions that use PC-relative indexed addressing, The value that is used for the PC is the address of the first byte of object code for the current instruction.

During reset, the program counter is loaded with the contents of the reset vector that is located at 0xFFFFFD through 0xFFFFFF. The vector stored there is the address of the first instruction that will be executed after exiting the reset state.

2.2.5 Condition Code Register (CCR)

The condition code register includes four ALU status bits (N, Z, V, C), Interrupt controls, and a user/supervisor state control bit. This register can be accessed as a 16-bit register (CCR or CCW), or you can access the high-order and low-order 8-bit bytes separately as CCH and CCL. The ALU status bits are all located in the low-order half (CCL) of the CCR.

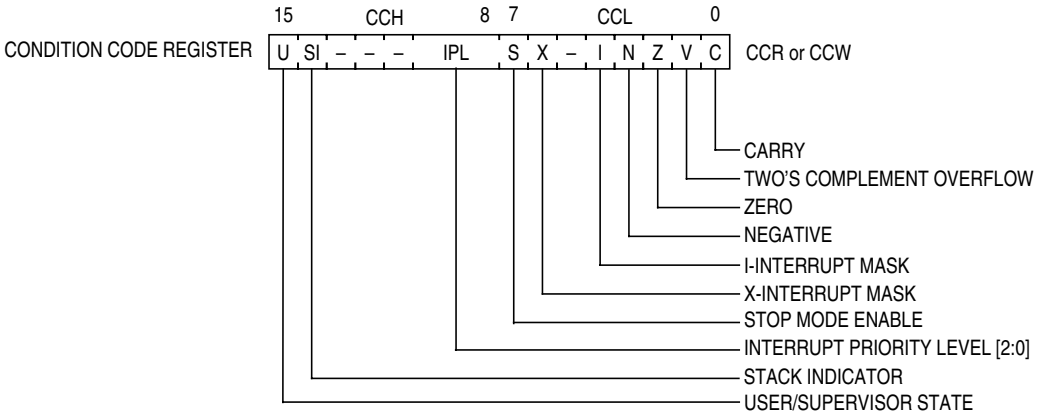


Figure 2-2. Condition Code Register

In some architectures, only a few instructions affect condition codes, so that multiple instructions must be executed in order to load and test a variable. Since most CPU S12Z instructions automatically update condition codes, it is rarely necessary to execute an extra instruction for this purpose. The challenge in using the S12Z lies in finding instructions that do not alter the condition codes. The most important of these instructions are LEA, moves, pushes, pulls, transfers, and exchanges.

It is always a good idea to refer to an instruction set summary to check which condition codes are affected by a particular instruction. For example, signed branches require a valid V condition code status flag and some instructions such as LEA do not update V. So signed branches are not useful after an LEA instruction.

The following paragraphs describe normal uses of the condition codes. There are other, more specialized uses. For instance, the C status bit is used to indicate the value of a bit prior to setting it with a BSET instruction to allow implementation of semaphores. Always refer to the detailed instruction descriptions to fully understand how CCR bits are affected.

Unused bits in the CCR are reserved for future use and should be zero for any CCR write operations.

2.2.5.1 U Control Bit

Setting this bit switches the CPU from Supervisor state (the default) to User state. In User state restrictions apply for the execution of several CPU instructions:

1. Write access to the system control bits in the Condition Code Register (U, IPL[2:0], S, X, I) is blocked. That means any attempts to change these bits are ignored. This affects the following instructions:
   — ANDCC (including the alias instruction CLI)
   — ORCC (including the alias instruction SEI)

— EXG with CCL, CCH or CCW
— TFR/ZEX/SEX with CCL, CCH or CCW as destination
— PUL CCH
— PUL CCL
— RTI

2. Instructions which would cause the CPU to suspend instruction execution are treated as No-Operation instructions (NOP). This affects the following instructions:
   — STOP
   — WAI

Exceptions cause the CPU to switch to Supervisor state. This means the U bit is automatically cleared when the CPU starts exception processing. Executing the RTI instruction when exiting the exception handler restores the state of the U bit from the exception stack frame.

2.2.5.2 IPL[2:0]

The IPL bits allow the nesting of interrupts, blocking interrupts of a lower priority. The current IPL is automatically pushed to the stack by the standard interrupt stacking procedure. The new IPL is copied to the CCR from the Priority Level of the highest priority active interrupt request channel. The copying takes place when the interrupt vector is fetched. The IPL bits are restored from the exception stack frame by executing the RTI instruction.

2.2.5.3 S Control Bit

Clearing the S bit enables the STOP instruction. Execution of a STOP instruction normally causes the on-chip oscillator to stop. This may be undesirable in some applications. If the S12Z CPU encounters a STOP instruction while the S bit is set (or while in user state) it is treated like a no-operation (NOP) instruction and continues to the next instruction. Reset sets the S bit.

2.2.5.4 X Mask Bit

The XIRQ input is an updated version of the NMI input found on earlier generations of MCUs. Non-maskable interrupts are typically used to deal with major system failures, such as loss of power. However, enabling non-maskable interrupts before a system is fully powered and initialized can lead to spurious interrupts. The X bit provides a mechanism for enabling non-maskable interrupts after a system is stable.

By default, the X bit is set to 1 during reset. As long as the X bit remains set, interrupt service requests made via the XIRQ pin are not recognized. An instruction must clear the X bit to enable non-maskable interrupt service requests made via the XIRQ pin. Once the X bit has been cleared to 0, software cannot set it to 1 by writing to the CCR. The X bit is not affected by maskable interrupts.

When an XIRQ interrupt occurs after non-maskable interrupts are enabled, both the X bit and the I bit are set automatically to prevent other interrupts from being recognized during the interrupt service routine. The mask bits are set after the registers are stacked, but before the interrupt vector is fetched.

Normally, a return-from-interrupt (RTI) instruction at the end of the interrupt service routine restores register values that were present before the interrupt occurred. Since the CCR is stacked before the X bit is set, the RTI normally clears the X bit, and thus re-enables non-maskable interrupts. While it is possible to manipulate the stacked value of X so that X is set after an RTI, there is no software method to set X (and disable XIRQ) once X has been cleared.

2.2.5.5 I Mask Bit

The I bit enables and disables maskable interrupt sources. By default, the I bit is set to 1 during reset. An instruction must clear the I bit to enable maskable interrupts. While the I bit is set, maskable interrupts can become pending and are remembered, but operation continues uninterrupted until the I bit is cleared.

When an interrupt occurs after interrupts are enabled, the I bit is automatically set to prevent other maskable interrupts during the interrupt service routine. The I bit is set after the registers are stacked, but before the first instruction in the interrupt service routine is executed.

Normally, an RTI instruction at the end of the interrupt service routine restores register values that were present before the interrupt occurred. Since the CCR is stacked before the I bit is set, the RTI normally clears the I bit, and thus re-enables interrupts. Interrupts can be re-enabled by clearing the I bit within the service routine.

2.2.5.6 N Status Bit

The N bit generally shows the state of the MSB of the result. An exception to this is the state of the N bit after the execution of an arithmetic-shift left (ASL) instruction (please refer to ASL for details). N is most commonly used in two's complement arithmetic, where the MSB of a negative number is 1 and the MSB of a positive number is 0, but it has other uses. For instance, if the MSB of a register or memory location is used as a status flag, the user can test status by simply loading a register or memory variable.

2.2.5.7 Z Status Bit

The Z bit is set when all the bits of the result are 0s. Compare instructions perform an internal implied subtraction, and the condition codes, including Z, reflect the results of that subtraction.

2.2.5.8 V Status Bit

The V bit is set when two's complement overflow occurs as a result of an operation. Two's complement overflow occurs only when the original value has its MSB set and all other bits clear (the most negative value possible for the size, i.e. 0x80, 0x8000, or 0x80000000), two's complement overflow occurs because it is not possible to express a positive two's complement value with the same magnitude.

2.2.5.9 C Status Bit

The C bit is set when a carry occurs during addition or a borrow occurs during subtraction. The C bit also acts as an error flag for multiply and divide operations. Shift and rotate instructions operate through the C bit to facilitate multiple-word shifts.

The C status bit is used to indicate the value of a bit prior to setting it with a BSET instruction to allow implementation of semaphores.

2.3 Data Types

The S12Z CPU uses these types of data:

- Bits
- 4-bit unsigned integers (only used for index offsets)
- 8-bit signed and unsigned integers
- 9-bit signed integers (only used for index offsets)
- 16-bit signed and unsigned integers
- 24-bit pointers
- 24-bit effective addresses (formed during address computations)
- 32-bit signed and unsigned integers

Negative integers are represented in two's complement form.

2.4 Memory Operand Sizes

In the linear S12Z, memory operands may be 8-bit bytes, 16-bit words, 24-bit pointers (normally associated with a 24-bit index register), or 32-bit long-words. There are bit-sized operations and bit-field operations on fields of 1-32 bits, but memory contents are always accessed 1, 2, 3, or 4 bytes at a time. Some instructions use operands that are partially or completely encoded into instructions and instruction postbytes and these operands may be other sizes (for example a 5-bit field width or shift count).

The CPU accesses memory information by the 24-bit address of the most significant byte of an operand without regard to alignment and a memory controller takes care of reading or writing the appropriate information. If necessary the CPU as well as the memory controller may access misaligned operands in multiple bus cycles.

Like earlier HC11 and HC12 CPUs, the S12Z makes no distinction between program memory and data memory. There is a single linearly addressed 16-megabyte address space and there are no separate instructions to access operands differently in program space than in RAM memory spaces. However, the linear S12Z CPU accesses program information and data information through separate memory busses and controllers. If the program is in a different memory than the data, it is possible for the CPU to access data operands at the same time as program code is loaded into the instruction queue. Otherwise these accesses are serialized by the memory-controller.

2.5 CPU Register Operands

CPU register operands include the eight data registers (D_i), the 24-bit X and Y index registers, the 24-bit stack pointer (SP), the 24-bit program counter (PC) and the condition codes register (CCR). D0 and D1 are 8 bits, D2 — D5 are 16 bits, and D6 and D7 are 32 bits. The CCR is 16 bits but the most frequently used status bits from the arithmetic logic unit (ALU) are accessible in the 8-bit CCL register which is the low order 8 bits of the 16-bit CCR. Transfer and exchange instructions can operate on the low half (CCL),

the high half (CCH), or the whole 16-bit CCR (CCW). CPU registers are hard-wired in the CPU and are not part of the 16-megabyte memory map.

2.6 Memory Organization

The S12Z CPU has a contiguous 16-megabyte address space.

Eight-bit values can be stored at any odd or even byte address in available memory.

Sixteen-bit values are stored in memory as two consecutive bytes; the high byte occupies the lowest address, but need not be aligned to an even boundary.

Twenty-four-bit values are stored in memory as three consecutive bytes; the high byte occupies the lowest address, but need not be aligned to an even boundary.

Thirty-two-bit values are stored in memory as four consecutive bytes; the high byte occupies the lowest address, but need not be aligned to an even boundary.

All input/output (I/O) and all on-chip peripherals are memory-mapped in the 16-megabyte address space. No special instruction syntax is required to access these addresses. On-chip registers and memory typically are grouped in blocks which can be relocated within the standard 16-megabyte address space. Refer to device documentation for specific information.

Although variables and I/O registers can be located anywhere in the 16-megabyte address space, there are extended addressing modes that are more efficient for the first 16 kilobyte and the first 256 kilobyte. The 14-bit extended addressing mode makes it more code-size efficient to locate control, status, and I/O registers in the first 16 kilobyte of memory space. The 18-bit extended addressing mode makes it more code-size efficient to locate program variables (RAM) in the first 256 kilobyte of memory space.

Reset and interrupt vectors are located at the highest locations in the 16-megabyte address space so MCU flash memory normally begins at the top of memory space and grows toward lower addresses.

Chapter 3
Addressing Modes

3.1 Introduction

Addressing modes determine how the central processing unit (CPU) accesses memory locations or registers to be used as operands in instructions.

3.2 Summary of Addressing Modes

The addressing modes and their variations are listed here:

- **INH** — Inherent
- **REG** — Register or Register as Operand — The operand is one of the eight CPU data registers (D_i). Register-as-Operand is a submode of general OPR addressing.
- **IMM** — Immediate — An instruction parameter or an operand is included as immediate data in the object code of the current instruction. Short Immediate (**IMMe4**) is a submode of general OPR addressing.
- **REL** — Relative addressing for branches — allows 7-bit or 15-bit signed offsets
- **EXT** — Extended — A 14-, 18-, or 24-bit address of an operand is provided in the instruction. Submodes of OPR addressing but LD, ST, JMP, and JSR have more efficient dedicated opcodes.
- Indexed submodes of general OPR addressing:
  - **IDX** — u4 Short Constant Offset Indexed submode of general OPR addressing
  - **IDX1** — s9 Constant Offset Indexed submode of general OPR addressing
  - **IDX3** — 24b Constant Offset Indexed submode of general OPR addressing
  - **REG,IDX** — Register Offset Indexed submode of general OPR addressing
  - **++IDX** — Pre/post increment/decrement Indexed submode of general OPR addressing — X, Y, or SP is used to access an operand either before or after it is incremented or decremented. The increment/decrement value is determined by the size of the operand that is being accessed.
  - **IDX2,REG** — u18 Offset from D_i Indexed submode of general OPR addressing — A CPU data register (D_i) is used as an index register in this indexed addressing mode variation.
  - **IDX3,REG** — 24b Offset from D_i Indexed submode of general OPR addressing — A CPU data registers (D_i) is used as an index register in this indexed addressing mode variation.
- Indexed Indirect submodes of general OPR addressing:
  - **[REG,IDX]** — Register Offset Indexed Indirect submode of general OPR addressing
  - **[IDX1]** — s9 Constant Offset Indexed Indirect submode of general OPR addressing
  - **[IDX3]** — 24b Constant Offset Indexed Indirect submode of general OPR addressing

- **[EXT3]** — 24-bit Address Indirect submode of general OPR addressing. This allows a 24-bit pointer to an operand to be located anywhere in the 16-megabyte memory space.

In the detailed descriptions of the addressing modes below, 16 addressing mode variations are identified with an asterisk* to indicate that these addressing modes are specified in the general operand (OPR) addressing mode postbyte (xb). All instruction opcodes that support OPR addressing have access to these same 16 addressing mode variations.

3.3 Inherent Addressing Mode (INH)

Operands (if any) are in CPU registers so no memory accesses are needed.

3.4 Register Addressing Mode (REG, REG*)

The operand is one of the eight CPU data registers (D_i) so no memory access is needed. The register number 0–7 is encoded in the opcode or an instruction postbyte. ‘Register as Operand’ is a submode of general OPR addressing.

3.5 Immediate Addressing Modes (IMM, IMM1, IMM2, IMM3, IMM4)

An instruction parameter or a one-, two-, three-, or four-byte operand is included as immediate data in the object code of the current instruction.

3.5.1 Short Immediate Addressing mode (IMMe4*)

A 4-bit immediate operand is encoded in the xb postbyte to provide a very efficient way to initialize registers or variables with the common values –1, 1, 2, 3,...13, 14, or 15 (automatically sign-extended to the required size).

For some variations of shift instructions, OPR addressing is used to specify the number of shift positions. In these cases, all OPR addressing sub-modes except short immediate and register-as-operand are available to specify a byte-sized memory operand. In these cases the short immediate sub-mode is used to supply the high-order four bits of a 5-bit immediate value n=0 to 31, and the least significant bit of the 5-bit immediate value is coded in the sb postbyte for the shift instruction.

3.6 Relative Addressing Modes (REL, REL1)

A 7-bit twos complement relative offset is included in the instruction postbyte or a 15-bit twos complement relative offset is included in the postbyte and one additional extension byte in the object code for the instruction. The relative offset is computed by adding the signed offset to the address of the first byte of object code for the current instruction.

3.7 Extended Addressing Modes (EXT1*, EXT2*, EXT3*, EXT24)

A 14-bit, 18-bit, or 24-bit address of the operand is provided in the instruction. In the case of 14-bit EXT1 and 18-bit EXT2 addressing modes, the supplied address is zero-extended to 24-bits to form the address of the operand.

EXT1 uses 6 bits in the xb postbyte plus one extension byte to specify the 14-bit extended address. EXT2 uses 2 bits in the xb postbyte plus 2 extension bytes to specify the 18-bit extended address. EXT3 and EXT24 use 3 bytes to specify a 24-bit address, but EXT3 is a sub-mode of general OPR addressing so it requires the xb postbyte in addition to the 24-bit address. EXT24 is more efficient (one less byte of object code) than EXT3 but only load, store, JMP, and JSR instructions offer EXT24 addressing mode because they are the most frequently used instructions that need to access operands anywhere in the 16-megabyte address space.

3.8 Indexed Addressing Modes

These indexed addressing modes use an index register as a base address and add a constant or register offset to form the effective address of the operand. The index register is usually X, Y, SP, or PC, but in a few modes a CPU data register D_i can be used as the index base address.

These addressing modes use a postbyte (xb) and zero, one, two, or three additional extension bytes in the object code. IDX implies zero extension bytes (everything the instruction needs is included in the postbyte or internal CPU registers). IDX1, IDX2, and IDX3 imply 1, 2, or 3 additional extension bytes are needed, respectively.

3.8.1 4-Bit Short Constant Offset from X, Y, or SP (IDX*)

A 4-bit unsigned constant (0–15) is added to X, Y, or SP to form the effective address of the operand. This addressing mode is very compact and efficient and handles the most common indexed addressing offsets. Larger offsets are supported with other indexed addressing mode variations which use additional extension bytes to specify the larger offsets.

3.8.2 9-Bit Constant Offset from X, Y, SP or PC (IDX1*)

A 9-bit signed constant (–256 to +255) is added to X, Y, SP or PC to form the effective address of the operand. This indexed addressing sub-mode uses the xb postbyte plus one extension byte. The ninth (sign) bit is encoded in the xb postbyte and the low-order 8 bits of the 9-bit offset are supplied in the extension byte.

3.8.3 24-Bit Constant Offset from X, Y, SP or PC (IDX3*)

A 24-bit constant is added to X, Y, SP or PC to form the effective address of the operand. The 24-bit offset is supplied in three extension bytes after the xb postbyte. Because the address bus is also 24 bits, you can think of the 24-bit offset as a signed or unsigned value in the range –8M to +16M.

3.8.4 Register Offset Indexed from X, Y, or SP (REG,IDX*)

A CPU data registers D_i is added to X, Y, or SP to form the effective address of the operand. This indexed addressing sub-mode allows a program-controlled offset which can change during execution of the program. For registers D0, D1, D6, and D7 the register is treated as an unsigned value. For D2~D5 the register is treated as a signed value.

3.8.5 Automatic Pre/Post Increment/Decrement from X, Y, or SP (++IDX*)

X, Y, or SP is used to access an operand either before or after it is incremented or decremented. The increment/decrement value is determined by the size of the operand that is being accessed. When SP is used as the index register, only pre-decrement (as in a PUSH) and post-increment (as in a PULL) variations are allowed. When X or Y is used as the index register, all four variations (pre-decrement, pre-increment, post-decrement, and post increment) are supported.

In cases where an instruction has more than one operand that uses indexed addressing, any auto-increment or decrement is done during processing of the current operand. For example, for the instruction...

```
MOV.W    (X+),(D2,X)
```

The CPU would first read the 16-bit memory value pointed to by index register X, then increment X (by 2 because the operand that was read was two bytes), then store the value at the address that is formed by adding D2 to index register X (the new incremented value in X, not the value X had when the instruction started).

3.8.6 18-Bit Constant Offset from D_i (IDX2,REG*)

An 18-bit unsigned constant is added to a CPU registers D_i to form the effective address of the operand. For registers D0, D1, D6, and D7 the register is treated as an unsigned value. For D2~D5 the register is treated as a signed value.

3.8.7 24-Bit Constant Offset from D_i (IDX3,REG*)

A 24-bit constant is added to a CPU registers D_i to form the effective address of the operand. For registers D0, D1, D6, and D7 the register is treated as an unsigned value. For D2~D5 the register is treated as a signed value.

3.9 Indexed Indirect Addressing Modes

These addressing modes use an indexed addressing mode to form the effective address of a pointer to the operand rather than using the indexed addressing mode to get the effective address of the operand itself. In all cases, the intermediate pointer that is fetched from the effective address is 24 bits and this 24-bit address is used to fetch the operand. The size of the operand (1, 2, 3, or 4 bytes) that this pointer points to, depends on the instruction.

3.9.1 Register Offset Indexed Indirect from X or Y ([REG,IDX]*)

A CPU data registers D_i is added to X or Y to form the effective address of the pointer to the operand. For registers D0, D1, D6, and D7 the register is treated as an unsigned value. For D2~D5 the register is treated as a signed value.

3.9.2 9-Bit Constant Offset Indexed Indirect from X, Y, SP or PC ([IDX1]*)

A 9-bit signed constant (–256 to +255) is added to X, Y, SP or PC to form the effective address of the pointer to the operand.

3.9.3 24-Bit Constant Offset Indexed Indirect from X, Y, SP or PC ([IDX3]*)

A 24-bit constant is added to X, Y, SP or PC to form the effective address of the pointer to the operand.

3.10 Address Indirect Addressing Mode ([EXT3]*)

This addressing mode uses a 24-bit constant to point to a pointer which is then used to access the operand. This allows a 24-bit pointer to an operand to be located anywhere in the 16-megabyte memory space. The 24-bit constant address that points to the pointer to the operand is supplied as three extension bytes after the xb postbyte in the object code of the instruction.

3.11 Effective Address

An effective address is the address that is (or would be) used to access memory during the execution of an instruction. Inherent and register addressing modes do not access memory so they do not generate effective addresses. Indirect addressing modes generate an effective address to access an intermediate pointer from memory and then use this pointer as the address which is used to access the instruction operand.

Load Effective Address (LEA) instructions load the effective address rather than the operand that is located at that address. Most of these instructions use the general OPR addressing modes. The short-immediate sub-mode and the register-as-operand sub-mode do not generate an effective address so it is not appropriate to use these sub-modes with an LEA instruction.

For the four indirect OPR sub-modes, the address that is loaded for an LEA instruction is the 24-bit address that would have been used to access the intermediate pointer to the operand in a normal load instruction using the same addressing mode. For the other ten OPR sub-modes, the address that is loaded for an LEA instruction is the 24-bit address that would have been used to access the operand in a normal load instruction using the same addressing mode.

In the special case of an LEA instruction with an auto pre/post increment/decrement indexed addressing mode, LEA loads the effective address that would have been used to access the operand for a load instruction using the same addressing mode. Pre increment/decrement modifies the index register before the operand would be accessed so these modification still apply for the LEA instructions. If the post increment/decrement applies to the same index register that is loaded with the LEA instruction, the post modification is ignored. If the post modification applies to a different index register than the index register that is loaded by the LEA instruction, then the post modification will be performed as expected.

3.12 Memory Operand Sizes

In the linear S12Z CPU, memory operands may be 8-bit bytes, 16-bit words, 24-bit pointers (normally associated with a 24-bit index register), or 32-bit long-words. There are bit-sized operations and bit-field operations on fields of 1-32 bits, but memory contents are always accessed 1, 2, 3, or 4 bytes at a time. Some instructions use operands that are partially or completely encoded into instructions and instruction postbytes and these operands may be other sizes (for example a 5-bit field width or shift count).

The CPU accesses memory information by the 24-bit address of the most significant byte of an operand without regard to alignment and a memory controller takes care of reading or writing the appropriate information. If necessary the memory controller may access misaligned operands in multiple bus cycles.

Like earlier HC11 and HC12 CPUs, the S12Z CPU makes no distinction between program memory and data memory. There is a single linearly addressed 16-megabyte address space and there are no separate instructions to access operands differently in program space than in RAM memory spaces. However, the linear S12Z CPU accesses program information and data information through separate memory busses and controllers. If the program is in a different memory than the data, it is possible for the CPU to access data operands at the same time as program code.

3.13 CPU Register Operands

CPU register operands include the eight data registers (D_i), the 24-bit X and Y index registers, the 24-bit stack pointer (SP), the 24-bit program counter (PC) and the condition codes register (CCR). D0 and D1 are 8 bits, D2–D5 are 16 bits, and D6 and D7 are 32 bits. The CCR is 16 bits but the most frequently used status bits from the arithmetic logic unit (ALU) are accessible in the 8-bit CCL register which is the low order 8 bits of the 16-bit CCR. Transfer and exchange instructions can operate on the low half (CCL), the high half (CCH), or the whole 16-bit CCR (CCW). CPU registers are hard-wired in the CPU and are not part of the 16-megabyte memory map.

3.14 Instructions Using Multiple Addressing Modes

Several S12Z CPU instructions have multiple operands or operands and parameters that use separate addressing modes to access each operand or parameter.

3.14.1 Shift Instructions

The shift instructions use one addressing mode to specify the register or memory location to be shifted and a separate addressing mode to specify the number of positions to shift the operand. These instructions have an opcode and one of two postbytes. If OPR addressing is specified to address the operand they also have an xb postbyte and 0 to 3 extension bytes to address the operand. The operand can use REG or OPR addressing mode and the parameter that specifies the number of positions to shift can be a 5-bit immediate value in the postbyte or a 5-bit value in a CPU data register D_i.

3.14.2 Bit Manipulation Instructions

The bit set and bit clear (BSET and BCLR) instructions use the same postbytes and addressing mode options as the shift instructions. The operand can be a register or a memory location accessed by the OPR addressing modes. The bit number to be modified is specified in a 5-bit immediate value in the postbyte, or a 5-bit value in a CPU data register. These instructions require 2 to 6 bytes of machine code.

The BRSET and BRCLR instructions have the same addressing mode options as BSET and BCLR, but they use a third addressing mode to specify an R7 or an R15 relative offset. R7 relative address mode allows a branch range of –64 to +63 from the address of the first byte of object code for the current

instruction. R15 relative address mode allows a branch range of –16,384 to +16,383 from the address of the first byte of object code for the current instruction.

3.14.3 Looping (DBcc, TBcc) Instructions

The decrement-and-branch and the test-and-branch instructions use one addressing mode to specify the operand and a second addressing mode for the relative branch. These instructions can use any of the eight CPU data registers D_i, the index registers X or Y, or a memory operand using the OPR addressing modes as the operand that is decremented or tested. The memory operand can be 8, 16, 24, or 32 bits (.B, .W, .P, or .L). They use 7-bit relative offset for –64 to +63 short branches or 15-bit relative for –16,384 to +16,383 long branches.

3.14.4 Math (MUL, MAC, DIV, and MOD) Instructions

All of these instructions perform a mathematical operation using two operands and store the result to one of the eight CPU Data registers. The result register is specified using a 3-bit field in the opcode. The first operand can be any of the eight CPU Data registers or an 8, 16, 24, or 32-bit memory operand using the OPR addressing modes. The second operand can be an 8, 16, or 32-bit immediate value or an 8, 16, 24, or 32-bit memory operand using the OPR addressing modes. The second operand can also be a CPU data register using the register-as-memory sub-mode of the OPR addressing modes.

3.14.5 Move Instructions

There are separate move instructions for 8-bit, 16-bit, 24-bit, and 32-bit operands. Each move instruction uses immediate address mode or OPR address modes for the source operand and OPR addressing modes for the destination operand.

Chapter 4
Instruction Queue

4.1 Introduction

The S12Z CPU uses an instruction queue to increase execution speed. This section describes queue operation during normal program execution and changes in execution flow. These concepts augment the descriptions of instructions and instruction execution in subsequent sections, but it is important to note that queue operation is automatic, and generally transparent to the user.

The material in this section is general. Chapter 8, “Instruction Execution Timing” contains information concerning cycle-by-cycle execution of each instruction.

4.2 Queue Description

The fetching mechanism used in the S12Z CPU is best described as a queue rather than as a pipeline. Queue logic fetches program information and positions it for execution, but instructions are executed sequentially. The S12Z CPU executes only one instruction at a time.

The queue is automatically refilled either every time the current program counter crosses a 4-byte boundary or when a change-of-flow event (for example a JMP instruction or an interrupt) occurs. Program fetches are done automatically in the background and are largely independent of instruction execution (except for change-of-flow events). Program information is fetched in memory-aligned 4-byte words.

The S12Z CPU instruction queue implementation features stage bypass logic. This is used to load the last queue stages first, so that instruction execution can continue as soon as possible after the queue was emptied.

4.2.1 S12Z CPU Instruction Queue Implementation

The instruction queue is implemented as a FIFO.

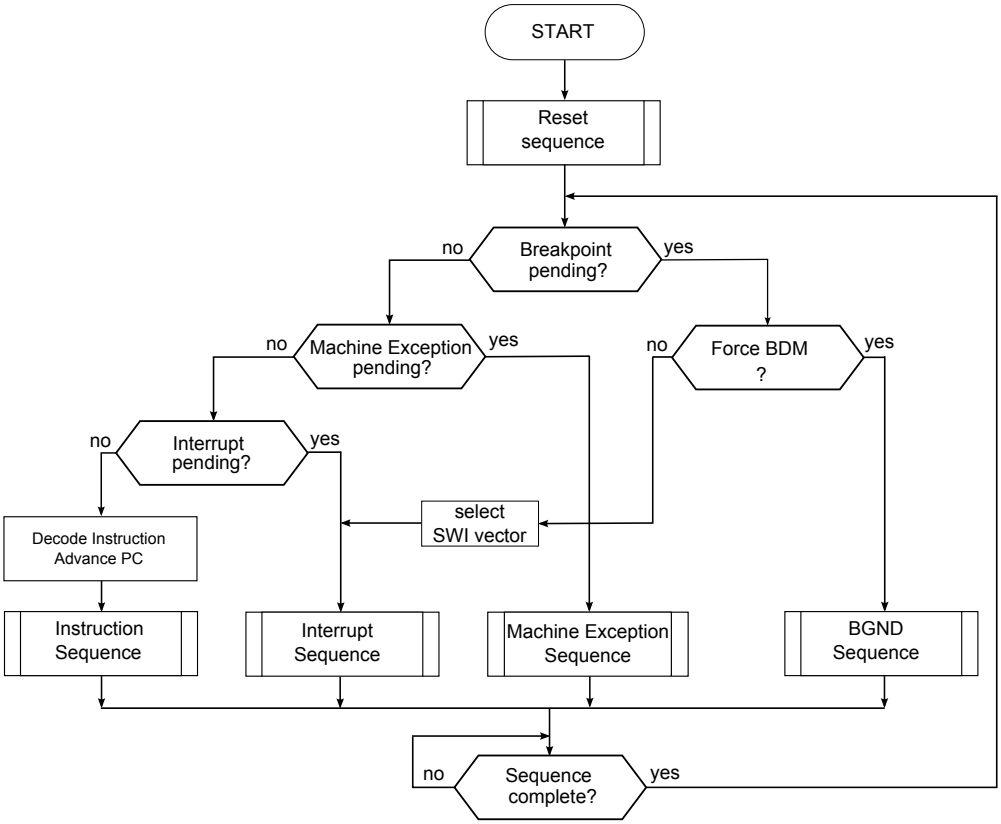
There are three 4-byte stages in the instruction queue.

Instruction execution can continue as soon as at least 4 bytes of valid program code is available in the queue.

4.2.2 S12Z CPU Operation Dispatcher

The output of the instruction queue is fed into the S12Z CPU operation dispatcher module. This module decides what operation is executed next while taking any pending breakpoints and exceptions into account. Figure 4-1 illustrates the operation dispatcher’s function.

Figure 4-1. S12Z CPU Operation Dispatcher



4.2.3 Changes in Execution Flow

During normal instruction execution, queue operations proceed as a continuous sequence of queue movement cycles. However, situations arise which call for changes in flow. These changes are categorized as resets, exceptions, subroutine calls, conditional branches, and jumps. Any such change causes the instruction queue to be reset.

Instruction execution after a change-of-flow event continues as soon as there are at least 4 bytes of new program code available in the instruction queue. This takes at least one (or two) bus-cycles, depending on the alignment of the new program counter value (for details please refer to Table 4-1).

Table 4-1. Effect of PC alignment on Execution Latency following a Change-of-Flow Event

PC[1:0]	Minimum amount of bus-cycles required after Change-of-Flow
0	1
1	2
2	2
3	2

NOTE

The numbers in Table 4-1 only represent the minimum amount of bus-cycles required to fetch 4 bytes of new program-code after a change-of-flow event.

4.2.3.1 Exceptions

Exceptions are events that require processing outside the normal flow of instruction execution. S12Z CPU Exceptions include six types of exceptions:

- Reset
- Unimplemented opcode traps
- Software interrupt instructions
- Machine exception
- X-bit interrupts
- I-bit interrupts

S12Z CPU exception handling is designed to minimize the effect of queue operation on context switching. Thus, an exception vector fetch is the first part of exception processing, and fetches to refill the queue from the address pointed to by the vector are done in parallel with the stacking operations that preserve context, so that program access time does not delay the switch. Refer to Chapter 7, “Exceptions” for detailed information.

4.2.3.2 Subroutines

The S12Z CPU can branch to (BSR) or jump to (JSR) subroutines.

BSR uses relative addressing mode to generate the effective address of the subroutine, while JSR can use various other addressing modes. Both instructions calculate a return address, stack the address, then perform a queue-flush operation to refill the instruction queue.

Subroutines are terminated with a return-from-subroutine (RTS) instruction. RTS unstacks the return address, then performs a queue-reset operation to refill the instruction queue.

4.2.4 Branches

Branch instructions cause execution flow to change when specific pre-conditions exist. The S12Z CPU instruction set includes:

- Conditional branches
- Bit-condition branches

Types and conditions of branch instructions are described in Section 5.5.1, “Branch Instructions”. All branch instructions affect the queue similarly, but there are differences in overall cycle counts between the various types. Loop primitive instructions are a special type of branch instruction used to implement counter-based loops.

Branch instructions have two execution cases:

- The branch condition is satisfied, and a change of flow takes place.
- The branch condition is not satisfied, and no change of flow occurs.

4.2.4.1 Conditional Branches

The “not-taken” case for short branches is simple. Since the instruction consists of two or three bytes containing both an opcode and a 7- or 15-bit offset, the queue advances and execution continues with the next instruction.

The “taken” case for branches requires the queue to be reset to cause a refill so that execution can continue at a new address.

4.2.4.2 Bit Condition Branches

Bit condition branch instructions read a location in memory, and branch if a specific bit in that location is in a certain state. If the branch is taken, the S12Z CPU performs a queue-reset operation to refill the instruction queue with program information from the new address.

4.2.4.3 Loop Primitives

The loop primitive instructions test a counter value in a register or accumulator and branch to an address specified by a relative offset contained in the instruction if a specified condition is met. If the branch is taken, the S12Z CPU performs a queue-reset operation to refill the instruction queue with program information from the new address.

4.2.5 Jumps

Jump (JMP) is the simplest change of flow instruction. JMP performs a queue-reset operation to refill the instruction queue with program information from the new address.

Chapter 5
Instruction Set Overview

5.1 Introduction

This section contains general information about the central processing unit (S12Z CPU) instruction set. It is organized into instruction categories grouped by function and sub-groups.

5.2 Instruction Set Description

The primary objectives of the S12Z CPU instruction set were to replace the paged memory model of the S12X with a new linear 24-bit address model and to optimize C code efficiency. CPU registers were changed to increase the width of the program counter, stack pointer, and index registers to 24 bits to match the width of the address bus. The two 8-bit accumulators A and B (which could be used together as the 16-bit D accumulator), were replaced by a larger set of eight general purpose CPU data registers. D0 and D1 are 8 bits, D2, D3, D4, and D5 are 16 bits, and D6 and D7 are 32 bits. This greatly reduces the need to save values and intermediate results on the stack or in RAM variables.

As in previous generations of CPU12, the S12Z CPU has variable-length instructions ranging from a single byte to several bytes. The longest instructions in the CPU12 were moves with two extended addressing mode operand addresses. Moves could have indexed addressing mode operands, but only indexed modes that did not require additional extension bytes. The S12Z CPU allows complete flexibility in specifying the addresses for move instructions.

The CPU12 used postbytes for indexed addressing, transfer/exchange, and looping primitive instructions. The S12Z CPU instruction set has expanded the use of postbytes to improve code-size efficiency. The indexed postbyte was re-worked into a general operand (OPR) addressing system. This new addressing mode postbyte includes indexed addressing modes like the CPU12 plus extended addressing modes, a quick-immediate mode, and register-as-memory addressing mode.

In addition to this general OPR addressing postbyte, the S12Z CPU instruction set uses postbytes for transfer/exchange, looping primitives, math (MUL, DIV, MAC, and MOD), relative addressing, shifts, bit-field instructions, and push/pull.

In the S12Z CPU architecture, all memory and input/output (I/O) are mapped in a common 16-megabyte address space (memory-mapped I/O). This allows the same set of instructions to be used to access memory, I/O, and control registers. General-purpose load, store, transfer, exchange, and move instructions facilitate movement of data to and from memory and peripherals.

The S12Z CPU supports operations on bits, 8-bit bytes, 16-bit words, and in some cases 24-bit pointers and 32-bit long-words. The instruction set supports both signed and unsigned math and branch operations. The S12Z CPU has added a 32-bit barrel shifter to improve the efficiency of shift operations. Efficient

bit-field operations were added to allow fields of up to 32 bits to be extracted-from or inserted-into operands.

Refer to Chapter 6, “Instruction Glossary” for detailed information about individual instructions. Appendix A, “Instruction Reference” contains quick-reference material, including an opcode map and postbyte encoding tables.

5.3 Instruction Set Organization

The instruction set can be divided into two major types of instructions and then each of these types can be further divided into sub groups containing closely-related instructions. The two major types are “register and memory instructions” and “program control instructions”. Register and memory instructions are related to data movement or mathematical and logical operations. Program control instructions manage the structure and flow of programs. Some instructions will appear in more than one sub-group. For example the load effective address (LEA) instructions are used to load the index registers, and they can also be used to perform arithmetic operations on index registers so they will appear in the data movement sub-group and in the arithmetic sub-group.

- Register and Memory Instructions
  - Data Movement and Initialization
    - Loading Data into CPU Registers
    - Storing CPU Register Contents into Memory
    - Memory-to-Memory Moves
    - Register-to-Register Transfer and Exchange
    - Clearing Registers or Memory Locations
    - Set or Clear Bits
  - Arithmetic Operations
    - Add
    - Increment
    - Add 8-bit Signed Immediate to X, Y, or S (LEA)
    - Subtract
    - Decrement
    - Compare
    - Negate
    - Absolute Value
    - Sign-Extend and Zero-Extend
  - Multiplication and Division
    - Multiply
    - Multiply and Accumulate
    - Divide
    - Modulo

- — Fractional Math Instructions
  - – Fractional Multiply
  - – Saturate
  - – Count Leading Bits
- — Logical (Boolean)
  - – Logical AND
  - – BIT (logical AND to set CCL but operand is left unchanged)
  - – Logical OR
  - – Logical Exclusive-OR
  - – Invert (bit-by-bit Ones Complement)
- — Shifts and Rotates
  - – Arithmetic Shift signed operand left or right through Carry by 0 to 31 bit positions
  - – Logical Shift unsigned binary operand left or right through Carry by 0 to 31 bit positions
  - – Rotate operand left or right through Carry by one bit position
- — Bit and Bit Field Manipulation
  - – Set, Clear, or Toggle Bits in Memory
  - – Set or Clear Bits in the CCR
  - – Bit Field Extract and Insert
- — Maximum and Minimum Instructions
- — Summary of Index and Stack Pointer Instructions
  - – Load
  - – Pull
  - – Restore CPU Registers after Interrupt (RTI)
  - – Store
  - – Push
  - – Stack CPU Registers on Entry to Interrupts (SWI, SYS)
  - – Load Effective Address (including signed addition)
  - – Subtract and Compare

• Program Control Instructions
- — Branch
  - – Branch on CCR conditions
  - – Branch on bit value
  - – Loop Control Branches (decrement and branch or test and branch)
- — Jump
- — Subroutine calls and returns
- — Interrupt Handling
- — Miscellaneous

- Low Power (STOP and WAI)
- No Operation (NOP)
- Go to active background debug mode (BGND)

5.4 Register and Memory Instructions

Register and memory instructions comprise the largest group of instructions in the instruction set. These instructions all involve CPU registers, memory locations, or both. The instructions in the data movement sub-group are used to load information into CPU registers, store information into memory, move information from one location to another, transfer or exchange data between two CPU registers, or clear registers, bits, or memory locations. Clear can be thought of as loading zero into a register, bit, or memory location.

The arithmetic sub-group includes addition, subtraction, compare, negate, and absolute value. Operations include signed and unsigned variations and there are sign-extend and zero-extend instructions to extend the width of signed and unsigned values. The multiplication and division sub-group includes signed and unsigned multiply, divide, multiply-and-accumulate (MAC), and modulo (MOD) operations.

Logical instructions include Boolean AND, OR, XOR, and invert (ones complement) operations. Arithmetic and logical shift by 0 to 31 bit positions are supported by a hardware barrel shifter. Shift and rotate instructions include the carry bit in the CCR to facilitate multi-precision shifts. Bit set, bit clear, and bit toggle instructions allow an individual bit in any memory location to be set, cleared, or toggled. There are also bit field instructions to extract a field from or insert a field into a CPU register or a memory operand. The field size and location can be specified with a width and offset in these instructions. Operands can be 8-, 16-, 24-, or 32-bit values.

Maximum and minimum instructions compare a value in a CPU data register to a value in memory and replace the register contents with the largest or smallest of these two values. There are both signed and unsigned versions of these instructions.

The last sub-group of instructions in the register and memory group is a summary of instructions related to the index registers and stack pointer. All of these instructions appear in the other sub-groups, but because the index registers and stack pointer are usually used for manipulating addresses and pointers rather than data, it is useful to see the summary of instructions that can be used with these index/pointer registers.

5.4.1 Data Movement and Initialization

The data movement portion of this sub-group includes loading information into registers (load, pull from stack, and load effective address), storing register contents (store, push onto stack), memory to memory move for bytes, words, pointers, and long-words, and register to register transfer and exchange. The initialization instructions include instructions to clear (load with zero) registers, bytes, words, or long-words in memory (variables), and set or clear bits in registers or memory.

Table 5-1 is a summary of the data movement instructions.

Table 5-1. Load and Store Instructions (Sheet 1 of 3)

Source Forms	Function	Operation
Load		

LD *Di*,#*oprimmsz*
LD *Di*,*opr24a*
LD *Di*,*oprmemreg*
	Load D_i from Memory	(M) ⇒ D_i

LD *xy*,#*opr18i*
LD *xy*,#*opr24i*
LD *xy*,*opr24a*
LD *xy*,*oprmemreg*
	Load index register X or Y from Memory	(M:M+1:M+2) ⇒ X or Y

LD S,#*opr24i* LD S,*oprmemreg*	Load stack pointer SP from Memory	(M:M+1:M+2) ⇒ SP
Pull (load from stack)		
PUL *oprregs1* PUL *oprregs2* PUL ALL PUL ALL16b	Pull specified CPU registers from Stack mask 1 - CCH, CCL, D0, D1, D2, D3 (D3 in LSB) mask 2 - D4, D5, D6, D7, X, Y (Y in LSB) pulls all registers in the same order as RTI	(M(SP)~M(SP+n-1)) ⇒ regs; (SP) + n ⇒ SP
Load Effective Address		
LEA D6,*oprmemreg* LEA D7,*oprmemreg*	Load Effective Address into 32-bit D6 or D7	00:Effective Address ⇒ D6, or 00:Effective Address ⇒ D7
LEA S,*oprmemreg* LEA X,*oprmemreg* LEA Y,*oprmemreg*	Load Effective Address into 24-bit X, Y, or SP	Effective Address ⇒ SP, or Effective Address ⇒ X, or Effective Address ⇒ Y
Store		
ST *Di*,*opr24a* ST *Di*,*oprmemreg*	Store Di to Memory	(D_i) ⇒ M
ST *xy*,*opr24a* ST *xy*,*oprmemreg*	Store index register X or Y to Memory	(X) ⇒ (M:M+1:M+2), or (Y) ⇒ (M:M+1:M+2)
ST S,*oprmemreg*	Store stack pointer SP to Memory	(SP) ⇒ (M:M+1:M+2)
Push (store to stack)		
PSH *oprregs1* PSH *oprregs2* PSH ALL PSH ALL16b	Push specified CPU registers onto Stack mask 1 - CCH, CCL, D0, D1, D2, D3 (D3 in LSB) mask 2 - D4, D5, D6, D7, X, Y (Y in LSB) pushes registers in the same order as SWI	(SP) – n ⇒ SP; (regs) ⇒ M(SP)~M(SP+n-1)
Move (memory-to-memory; byte, word, pointer, or long-word)		
MOV.B #*opr8i*,*oprmemreg*	Move Immediate to Memory MD, 8-bit operand	# ⇒ MD
MOV.B *oprmemreg*,*oprmemreg*	Move memory to memory, 8-bit operand	(MS) ⇒ MD
MOV.W #*opr16i*,*oprmemreg*	Move Immediate to Memory MD, 16-bit operand	# ⇒ MD

Table 5-1. Load and Store Instructions (Sheet 2 of 3)

Source Forms	Function	Operation
MOV.W *oprmemreg*,*oprmemreg*	Move memory to memory, 16-bit operand	(MS) ⇒ MD
MOV.P #*opr24i*,*oprmemreg*	Move Immediate to Memory MD, 24-bit operand	# ⇒ MD
MOV.P *oprmemreg*,*oprmemreg*	Move memory to memory, 24-bit operand	(MS) ⇒ MD
MOV.L #*opr32i*,*oprmemreg*	Move Immediate to Memory MD, 32-bit operand	# ⇒ MD
MOV.L *oprmemreg*,*oprmemreg*	Move memory to memory, 32-bit operand	(MS) ⇒ MD
Transfer and Exchange		
TFR *cpureg*,*cpureg*	Transfer CPU Register r1 to r2 D0~D7, X, Y, SP, CCH, CCL, or CCW if same size, direct transfer if 1st smaller than 2nd, zero-extend 1st to 2nd	(r1) ⇒ (r2)
EXG *cpureg*,*cpureg*	Exchange contents of CPU Registers D0~D7, X, Y, SP, CCH, CCL, or CCW if same size, direct exchange if 1st smaller than 2nd, sign extend 1st to 2nd	(r1) ⇔ (r2)

Table 5-1. Load and Store Instructions (Sheet 3 of 3)

Source Forms	Function	Operation
Clear (load with zero)		
CLR *Di* CLR.*bwpl oprmemreg* CLR X CLR Y	Clear data register D_i , Memory, or Index Pointer	$0 \Rightarrow D_i$, $0 \Rightarrow M$, $0 \Rightarrow X$, or $0 \Rightarrow Y$
Set or Clear Bits (in memory or CCR)		
BSET *Di*,#*opr5i* BSET.*bwl oprmemreg*,#*opr5i* BSET.*bwl oprmemreg*,*Dn*	Set Bit n in Memory or in D_i C equal the original value of bitn in M or D_i (semaphore)	(M) | bitn $\Rightarrow$ M or (D_i) | bitn $\Rightarrow D_i$
BCLR *Di*,#*opr5i* BCLR.*bwl oprmemreg*,#*opr5i* BCLR.*bwl oprmemreg*,*Dn*	Clear Bit n in Memory or in D_i C equal the original value of bitn in M or D_i (semaphore)	(M) & ~bitn $\Rightarrow$ M or (D_i) & ~bitn $\Rightarrow D_i$
SEC	Set Carry Bit *Translates to* ORCC #$01	$1 \Rightarrow C$
SEI	Set I Bit; (inhibit I interrupts) *Translates to* ORCC #$10	$1 \Rightarrow I$
SEV	Set Overflow Bit *Translates to* ORCC #$02	$1 \Rightarrow V$
CLC	Clear Carry Bit *Translates to* ANDCC #$FE	$0 \Rightarrow C$
CLI	Clear I Bit; (I can only be changed in supervisor state) *Translates to* ANDCC #$EF (enables I interrupts)	$0 \Rightarrow I$
CLV	Clear Overflow Bit *Translates to* ANDCC #$FD	$0 \Rightarrow V$

5.4.1.1 Loading Data into CPU Registers

Load instructions copy memory content into a CPU register. Memory content normally is not changed by the operation. Load instructions (but not LEA_ or PUL_ instructions) affect condition code bits so no separate test instructions are needed to check the loaded values for negative or 0 conditions.

Pull instructions are specialized load instructions that use the stack pointer as an index pointer. The stack pointer is automatically updated (post-incremented) to point at the new end of the stack after data is loaded (pulled) from the stack.

Load effective address instructions copy the address of a memory location into one of the index registers D6, D7, S, X, or Y. D6 and D7 are usually used as 32-bit data registers, but there are indexed addressing instructions which can use these registers as a base index register for indexed addressing.

For certain control and status register locations, reading a control register may be part of a flag clearing sequence which can change the state of a status flag or modify a FIFO pointer. These registers are clearly described in the data sheet for a specific MCU. For normal flash or RAM memory, reading a location does not change the contents of that location.

5.4.1.2 Storing CPU Register Contents into Memory

Store instructions copy the content of a CPU register to memory. Register content is not changed by the operation. Store instructions automatically update the N and Z condition code bits, which can eliminate the need for a separate test instruction in some programs. Push instructions are specialized store instructions that use the stack pointer as an index pointer. The stack pointer is automatically adjusted (pre-decremented) to point at the next available location on the stack before the data is stored (pushed).

5.4.1.3 Memory-to-Memory Moves

Move instructions move (copy) data from a source to a destination. The size of the operation can be 8-bit bytes, 16-bit words, 24-bit pointers, or 32-bit long-words. The flexible OPR addressing mode offers complete flexibility in specifying the source and destination locations using 13, 18, or 24-bit extended addressing modes, any indexed or indexed-indirect addressing modes, CPU data registers, or efficient sign-extended short immediate values. Unlike load and store instructions, move instructions do not affect CCR bits.

5.4.1.4 Register-to-Register Transfer and Exchange

Transfer and exchange instructions allow any of the CPU registers D0–D7, S, X, Y, CCH, CCL, or CCW as a source and/or destination. If the source and destination are the same size (same number of bits), a direct transfer or exchange is performed. Transfer and exchange instructions do not alter the CCR bits unless, of course, CCH, CCL, or CCW is a destination for the transfer or exchange.

Refer to Chapter 6, “Instruction Glossary” for more detailed information about cases where the source and destination are not the same width. Also check this glossary for special cases involving the CCR (CCH, CCL, or CCW).

5.4.1.5 Clearing Registers or Memory Locations

Clearing registers and memory variables is equivalent to loading them with zeros. Zero is such a common value for program variables that it improves code size efficiency to have dedicated instructions to clear registers and memory locations. For example, LD D6,#$00000000 requires five bytes of object code to clear the 32-bit D6 register while CLR D6 performs the same function but requires only one byte of object code.

There are efficient clear instructions for the eight CPU data registers D0~D7, X, Y, and bytes, words, pointers, or long-words in memory.

5.4.1.6 Set or Clear Bits

There are instructions to set or clear any one bit in a CPU data register or a variable in memory. The location to be operated on can be one of the eight CPU data registers with the bit number to be set or cleared in an immediate 5-bit value. When the variable to be operated on is in memory, the location may be a byte, word, or long-word and the bit number can be supplied in the low-order five bits of one of the eight CPU data registers or a 5-bit immediate value. These are read-modify-write instructions.

There are also specialized instructions to set or clear the C, I, or V bits in the condition codes register. These instructions are actually alternate mnemonics for ORCC to set bits or ANDCC to clear bits. Setting or clearing the carry bit (C) can be useful before shift and rotate instructions because these instructions include the carry bit. Setting the I interrupt mask blocks I-type interrupts and clearing I allows I interrupts.

5.4.2 Arithmetic Operations

This group includes arithmetic instructions for calculations involving signed and unsigned values. Basic operations include adding, subtracting, increment, decrement, twos-complement negate, absolute value, sign-extend, and zero-extend instructions. Compare instructions perform a subtraction to set condition code bits, but do not save the result or modify the operands. Load effective address (LEA) instructions add an 8-bit signed value to X, Y, or S which is useful for moving pointers through tables of data records.

Table 5-2 shows a summary of the S12Z arithmetic instructions.

Table 5-2. Arithmetic Instructions (Sheet 1 of 2)

Source Forms	Function	Operation
Addition		
ADD *Di*,#*oprimmsz* ADD *Di*,*oprmemreg*	Add without Carry to D_i	$(D_i) + (M) \Rightarrow D_i$
ADC *Di*,#*oprimmsz* ADC *Di*,*oprmemreg*	Add with Carry to D_i	$(D_i) + (M) + C \Rightarrow D_i$
Increment		
INC *Di* INC.*bwl oprmemreg*	Increment data register D_i or Memory	$(D_i) + 1 \Rightarrow D_i$, or $(M) + 1 \Rightarrow M$
Add 8-bit Signed Immediate to X, Y, or S (LEA)		
LEA S,(#*opr8i*,S) LEA X,(#*opr8i*,X) LEA Y,(#*opr8i*,Y)	Add sign-extended 8-bit Immediate to X, Y, or SP no change to CCR bits	(SP) + sign-extend (M) ⇒ SP, or (X) + sign-extend (M) ⇒ X, or (Y) + sign-extend (M) ⇒ Y
Subtraction		

Table 5-2. Arithmetic Instructions (Sheet 2 of 2)

Source Forms	Function	Operation
SUB *Di*,#*oprimmsz* SUB *Di*,*oprmemreg*	Subtract without Carry	$(D_i) - (M) \Rightarrow D_i$
SUB D6,X,Y		$(X) - (Y) \Rightarrow D6$
SUB D6,Y,X		$(Y) - (X) \Rightarrow D6$
SBC *Di*,#*oprimmsz* SBC *Di*,*oprmemreg*	Subtract with Carry from D_i	$(D_i) - (M) - C \Rightarrow D_i$
Decrement		
DEC *Di* DEC.*bwl oprmemreg*	Decrement data register D_i or Memory	$(D_i) - 1 \Rightarrow D_i$, or $(M) - 1 \Rightarrow M$
Compare		
CMP *Di*,#*oprimmsz* CMP *Di*,*oprmemreg*	Compare D_i with Memory	$(D_i) - (M)$
CMP *xy*,#*opr24i* CMP *xy*,*oprmemreg*	Compare X or Y with Memory	(xy) – (M:M+1:M+2)
CMP S,#*opr24i* CMP S,*oprmemreg*	Compare stack pointer SP with Memory	(SP) – (M:M+1:M+2)
CMP X,Y	Compare X with Y	(X) – (Y)
Negate (twos complement)		
NEG.*bwl oprmemreg*	Twos Complement Negate	$0 - (M) \Rightarrow M$ equivalent to $\sim(M) + 1 \Rightarrow M$ $0 - (D_i) \Rightarrow D_i$ equivalent to $\sim(D_i) + 1 \Rightarrow D_i$
Absolute Value		
ABS *Di*	Replace D_i with the Absolute Value of D_i	$\lvert(D_i)\rvert \Rightarrow D_i$
Sign-Extend and Zero-Extend		
SEX *cpureg*,*cpureg*	Sign-Extend (r1) ⇒ (r2) D0~D7, X, Y, SP, CCH, CCL, or CCW same as exchange EXG except r1 is smaller than r2	Sign-Extend (r1) ⇒ (r2)
ZEX *cpureg*,*cpureg*	Zero-Extend (r1) ⇒ (r2) D0~D7, X, Y, SP, CCH, CCL, or CCW same as transfer TFR except r1 is smaller than r2	Zero-Extend (r1) ⇒ (r2)

5.4.2.1 Add

8-, 16-, and 32-bit addition of signed or unsigned values can be performed between registers or between a register and memory. Instructions that add the carry bit in the condition code register (CCR) facilitate multiple precision computation.

5.4.2.2 Increment and Decrement

The increment and decrement instructions are optimized addition and subtraction operations. They are generally used to implement counters. Because they do not affect the carry bit in the CCR, they are well suited for loop counters in multiple-precision arithmetic computation routines. These instructions can be used to increment or decrement CPU data registers or 8-bit byte, 16-bit word, or 32-bit long-word variables in memory.

5.4.2.3 LEA (add immediate 8-bit signed value to X, Y, or SP)

LEA instructions can be used to increment or decrement index registers or the stack pointer, although these instructions are not limited to simple increment and decrement operations. These instructions add an 8-bit signed value between –128 and +127 to the value in X, Y, or SP so a program can efficiently move through a table by several values or records at a time. The LEA instructions are described in more detail in Section 5.4.9, “Summary of Index and Stack Pointer Instructions”. There are also indexed addressing modes that automatically increment or decrement X, Y, or S by an amount corresponding to the size of the operation in the instruction. For more detail see Section 3.8.5, “Automatic Pre/Post Increment/Decrement from X, Y, or SP (++IDX*)”.

There are looping primitive instructions that combine a decrement (DBcc) or test (TBcc) and a conditional branch in a single efficient instruction. Refer to Section 5.5.1.3, “Loop Control Branches (decrement and branch or test and branch)” for information concerning automatic counter branches.

Load effective address (LEA D6, LEA D7, LEA S, LEA X, and LEA Y) instructions could also be considered as specialized addition and subtraction instructions because several basic arithmetic operations can be performed during the formation of the effective addresses. There are also efficient 2-byte instructions to add a sign-extended 8-bit immediate value to S, X, or Y. The LEA instructions are described in more detail in Section 5.4.9, “Summary of Index and Stack Pointer Instructions”.

5.4.2.4 Subtract

8-, 16-, and 32-bit subtraction of signed or unsigned values can be performed between registers or between a register and memory. Instructions that subtract the carry bit in the CCR facilitate multiple precision computation.

24-bit index registers X and Y can be subtracted with the result going to 32-bit data register D6. In this case, the values in X and Y are treated as unsigned addresses and the result is treated as a signed long integer.

5.4.2.5 Compare

Compare instructions perform a subtraction between a pair of registers or between a register and memory. The result is not stored, but condition codes are affected by the operation. These instructions are generally used to establish conditions for branch instructions. In this architecture, most instructions update condition code bits automatically, so it is often unnecessary to include separate test or compare instructions in application programs.

5.4.2.6 Negate

Negate operations replace the value with its twos complement. This is equivalent to multiplying by –1. It inverts the sign of a twos complement signed value. There is a separate COM instruction which performs a Boolean bit-by-bit inversion which is described in Section 5.4.5.5, “Invert (bit-by-bit Ones Complement)”.

5.4.2.7 Absolute Value

The absolute value instruction returns the magnitude of a signed value in one of the CPU data registers. The value in the 8-bit, 16-bit, or 32-bit CPU data register before the ABS operation is interpreted as a twos complement signed value. If the value was negative (MSB=1), the register contents are replaced by the twos complement of the value. If the value was already positive (MSB=0), the register contents are unchanged.

5.4.2.8 Sign-Extend and Zero-Extend

If the source of an exchange instruction is smaller than the destination register, the smaller source is sign-extended (SEX) to the width of the destination and stored in the destination. The source of the sign-extend operation is not changed.

If the source of a transfer is smaller than the destination register, the source register is zero-extended (ZEX) and stored in the destination register.

Refer to Chapter 6, “Instruction Glossary” for more detailed information about cases where the source and destination are not the same width. Also check this glossary for special cases involving the CCR (CCH, CCL, or CCW).

5.4.3 Multiplication and Division

There are four basic algebraic instructions in this group — multiply (MUL), multiply-and-accumulate (MAC), divide (DIV), and modulo (MOD). Each of these four instructions have signed and unsigned variations. The 8-bit, 16-bit, or 32-bit result is always one of the eight general purpose CPU data registers (D0-D7). Operands can be 8-bit, 16-bit, 24-bit, or 32-bit values in any combination.

There is much more flexibility for specifying the two input operands for each of these instructions compared to the previous generations of the CPU12. All four instructions have the same addressing mode choices for these input operands and they include register/register, register/immediate, register/memory, and memory/memory. Operands in memory use the flexible OPR addressing modes which include indexed addressing modes like the CPU12 plus extended addressing modes, a quick-immediate mode, and register-as-memory addressing mode. Refer to Chapter 6, “Instruction Glossary” for a complete list of all allowed source form variations for all instructions.

Table 5-3 shows a summary of the multiplication and division instructions.

Table 5-3. Multiplication and Division Instructions (Sheet 1 of 2)

Source Forms	Function	Operation
Multiplication (MUL and MAC)		

MULS *Dd*,*Dj*,*Dk*
MULS *Dd*,*Dj*,#*opr8i*
MULS *Dd*,*Dj*,#*opr16i*
MULS *Dd*,*Dj*,#*opr32i*
MULS.*bwl Dd*,*Dj*,*oprmemreg*
MULS.*bwplbwpl*
Dd,*oprmemreg*,*oprmemreg*
	Signed Multiply
	result is always a register D_d
		$(D_j) * (D_k) \Rightarrow D_d$, or
		$(D_j) * (M) \Rightarrow D_d$, or
		$(M1) * (M2) \Rightarrow D_d$

MULU *Dd*,*Dj*,*Dk*
MULU *Dd*,*Dj*,#*opr8i*
MULU *Dd*,*Dj*,#*opr16i*
MULU *Dd*,*Dj*,#*opr32i*
MULU.*bwl Dd*,*Dj*,*oprmemreg*
MULU.*bwplbwpl*
Dd,*oprmemreg*,*oprmemreg*
	Unsigned Multiply
	result is always a register D_d
		$(D_j) * (D_k) \Rightarrow D_d$, or
		$(D_j) * (M) \Rightarrow D_d$, or
		$(M1) * (M2) \Rightarrow D_d$

MACS *Dd*,*Dj*,*Dk* MACS *Dd*,*Dj*,#*opr8i* MACS *Dd*,*Dj*,#*opr16i* MACS *Dd*,*Dj*,#*opr32i* MACS.*bwl Dd*,*Dj*,*oprmemreg* MACS.*bwplbwpl* *Dd*,*oprmemreg*,*oprmemreg*	Signed Multiply and Accumulate result is always a register D_d	$(D_j) * (D_k) + D_d \Rightarrow D_d$, or $(D_j) * (M) + D_d \Rightarrow D_d$, or $(M1) * (M2) + D_d \Rightarrow D_d$
MACU *Dd*,*Dj*,*Dk* MACU *Dd*,*Dj*,#*opr8i* MACU *Dd*,*Dj*,#*opr16i* MACU *Dd*,*Dj*,#*opr32i* MACU.*bwl Dd*,*Dj*,*oprmemreg* MACU.*bwplbwpl* *Dd*,*oprmemreg*,*oprmemreg*	Unsigned Multiply and Accumulate result is always a register D_d	$(D_j) * (D_k) + D_d \Rightarrow D_d$, or $(D_j) * (M) + D_d \Rightarrow D_d$, or $(M1) * (M2) + D_d \Rightarrow D_d$
Division (DIV and MOD)		
DIVS *Dd*,*Dj*,*Dk* DIVS *Dd*,*Dj*,#*opr8i* DIVS *Dd*,*Dj*,#*opr16i* DIVS *Dd*,*Dj*,#*opr32i* DIVS.*bwl Dd*,*Dj*,*oprmemreg* DIVS.*bwplbwpl* *Dd*,*oprmemreg*,*oprmemreg*	Signed Divide result is always a register D_d	$(D_j) / (D_k) \Rightarrow D_d$, or $(D_j) / (M) \Rightarrow D_d$, or $(M1) / (M2) \Rightarrow D_d$
DIVU *Dd*,*Dj*,*Dk* DIVU *Dd*,*Dj*,#*opr8i* DIVU *Dd*,*Dj*,#*opr16i* DIVU *Dd*,*Dj*,#*opr32i* DIVU.*bwl Dd*,*Dj*,*oprmemreg* DIVU.*bwplbwpl* *Dd*,*oprmemreg*,*oprmemreg*	Unsigned Divide result is always a register D_d	$(D_j) / (D_k) \Rightarrow D_d$, or $(D_j) / (M) \Rightarrow D_d$, or $(M1) / (M2) \Rightarrow D_d$

Table 5-3. Multiplication and Division Instructions (Sheet 2 of 2)

Source Forms	Function	Operation
MODS *Dd*,*Dj*,*Dk* MODS *Dd*,*Dj*,#*opr8i* MODS *Dd*,*Dj*,#*opr16i* MODS *Dd*,*Dj*,#*opr32i* MODS.*bwl Dd*,*Dj*,*oprmemreg* MODS.*bwplbwpl* *Dd*,*oprmemreg*,*oprmemreg*	Signed Modulo result is always a register D_d	(D_j) % (D_k); remainder ⇒ D_d, or (D_j) % (M); remainder ⇒ D_d, or (M1)% (M2); remainder ⇒ D_d
MODU *Dd*,*Dj*,*Dk* MODU *Dd*,*Dj*,#*opr8i* MODU *Dd*,*Dj*,#*opr16i* MODU *Dd*,*Dj*,#*opr32i* MODU.*bwl Dd*,*Dj*,*oprmemreg* MODU.*bwplbwpl* *Dd*,*oprmemreg*,*oprmemreg*	Unsigned Modulo result is always a register D_d	(D_j) % (D_k); remainder ⇒ D_d, or (D_j) % (M); remainder ⇒ D_d, or (M1)% (M2); remainder ⇒ D_d

5.4.3.1 Multiply and Multiply-and-Accumulate

MULS and MACS perform signed multiplication and MULU and MACU perform unsigned multiplication. The destination (result) is always one of the 8-bit, 16-bit, or 32-bit CPU data registers. The first input operand may be a CPU data register or a memory operand using OPR addressing. The second operand may be a CPU data register, an 8-bit, 16-bit, or 32-bit immediate value, or a memory operand using OPR addressing. Memory operands may be 8-bit, 16-bit, 24-bit, or 32-bit values.

5.4.3.2 Divide and Modulo

DIVS and MODS perform signed division and DIVU and MODU perform unsigned division. The result is always one of the 8-bit, 16-bit, or 32-bit CPU data registers. For DIVS and DIVU, the result is the quotient of the division operation. For modulo instructions MODS and MODU, the result is the remainder after the division operation is completed and the quotient is discarded. The first input operand (dividend) may be a CPU data register or a memory operand using OPR addressing. The second operand (divisor) may be a CPU data register, an 8-bit, 16-bit, or 32-bit immediate value, or a memory operand using OPR addressing. Memory operands may be 8-bit, 16-bit, 24-bit, or 32-bit values.

5.4.4 Fractional Math Instructions

There are three basic algebraic instructions in this group — saturating fractional multiply (QMULS, QMULU), saturate (SAT), and, to assist normalization of operands, there is a count-leading-bits instruction (CLB).

Table 5-4 shows a summary of the S12Z CPU fractional math instructions.

Table 5-4. Fractional Math Instructions

Source Forms	Function	Operation
Fractional Multiplication		
QMULS *Dd*,*Dj*,*Dk* QMULS *Dd*,*Dj*,#*opr8i* QMULS *Dd*,*Dj*,#*opr16i* QMULS *Dd*,*Dj*,#*opr32i* QMULS.*bwl Dd*,*Dj*,*oprmemreg* QMULS.*bwplbwpl Dd*,*oprmemreg*,*oprmemreg*	Signed Fractional Multiply result is always a register D_d operand format is either s.7, s.15, s.23 or s.31[1] result format is either s.7, s.15 or s.31[1] depending on the size of the result register	$(D_j) * (D_k) \Rightarrow D_d$, or $(D_j) * (M) \Rightarrow D_d$, or $(M1) * (M2) \Rightarrow D_d$
QMULU *Dd*,*Dj*,*Dk* QMULU *Dd*,*Dj*,#*opr8i* QMULU *Dd*,*Dj*,#*opr16i* QMULU *Dd*,*Dj*,#*opr32i* QMULU.*bwl Dd*,*Dj*,*oprmemreg* QMULU.*bwplbwpl Dd*,*oprmemreg*,*oprmemreg*	Unsigned Fractional Multiply result is always a register D_d operand format is either .8, .16, .24 or .32[2] result format is either .8, .16 or .32[2] depending on the size of the result register	$(D_j) * (D_k) \Rightarrow D_d$, or $(D_j) * (M) \Rightarrow D_d$, or $(M1) * (M2) \Rightarrow D_d$
Saturate		
SAT *Di*	Saturate(Di) ⇒ (Di) D0~D7	If (V & N) then MAX_VALUE ⇒ Di If (V & ~N) then MIN_VALUE ⇒ Di
Count Leading Bits		
CLB *cpureg*,*cpureg*	Count leading sign-bits of (r1) ⇒ (r2) D0~D7	count leading sign-bits of (r1) ⇒ (r2)

[1] The definition of signed fractional data-formats s.7, s.15, s.23 and s.31 is the same as the definition used in the ISO-C draft Technical Report document TR 18037.

[2] The definition of unsigned fractional data-formats .8, .16, .24 and .32 is the same as the definition used in the ISO-C draft Technical Report document TR 18037.

5.4.4.1 Fractional Multiply

These instruction perform fractional multiplication on operands in fractional fixed-point format as defined in the ISO-C Technical Report document TR 18037. This format is also known as “Q”-format.

QMULS performs signed multiplication and QMULU performs unsigned fractional multiplication. This means the content of the result register corresponds to the most-significant bits of the multiplication result, with any least significant bits not fitting into the result register cut-off without rounding.

The destination (result) is always one of the 8-bit, 16-bit, or 32-bit CPU data registers. The first input operand may be a CPU data register or a memory operand using OPR addressing. The second operand may be a CPU data register, an 8-bit, 16-bit, or 32-bit immediate value, or a memory operand using OPR addressing. Memory operands may be 8-bit, 16-bit, 24-bit, or 32-bit values.

Both source operands are aligned before the multiplication. This means that a smaller-sized source operand is expanded to the size of a bigger-sized source operand by right-appending zeroes.

5.4.4.2 Saturate

Saturate the content of the operand register using the information stored in the overflow (V-) and negative (N-)flags by a previous instruction. This works for most instructions which are capable of producing a signed result in two's complement format (e.g. ADD, SUB, NEG, ABS, ...).

5.4.4.3 Count Leading Sign-Bits

Counts the leading sign-bits of the content of the source register, then decrements and puts the result in the destination register. The result can directly be used as shift-width for a shift-left operation in order to normalize the fractional fixed-point value in the source operand.

5.4.5 Logical (Boolean)

This group of instructions is used to perform the basic Boolean operations, AND, OR, Exclusive-OR, and invert (COMplement) as well as the BIT instruction which is a specialized type of AND operation which affects the condition code bits but does not modify the source operands or save the result of the AND operation.

Table 5-5 shows a summary of the Boolean logic instructions.

Table 5-5. Boolean Logic Instructions

Source Forms	Function	Operation
Logical AND		
AND *Di*,#*oprimmsz* AND *Di*,*oprmemreg*	Bitwise AND D_i with Memory	(D_i) & (M) ⇒ D_i
ANDCC #*opr8i*	Bitwise AND CCL with immediate byte in Memory (S, X, and I can only be changed in supervisor state)	(CCL) & (M) ⇒ CCL
BIT *Di*,#*oprimmsz* BIT *Di*,*oprmemreg*	Bitwise AND D_i with Memory	(D_i) & (M); Sets CCR bits; Operands unchanged
Logical OR		
OR *Di*,#*oprimmsz* OR *Di*,*oprmemreg*	Bitwise OR D_i with Memory	(D_i) | (M) ⇒ D_i
ORCC #*opr8i*	Bitwise OR CCL with Immediate Mask (S, X, and I can only be changed in supervisor state)	(CCL) | (M) ⇒ CCL
Logical Exclusive-OR		
EOR *Di*,#*oprimmsz* EOR *Di*,*oprmemreg*	Exclusive OR D_i with Memory	(D_i) ^ (M) ⇒ D_i
Logical Invert (bit-by-bit ones complement)		
COM.*bwl oprmemreg*	1's Complement Memory Location or D_i	~(M) ⇒ M equivalent to $F..F – (M) ⇒ M ~$(D_i)$ ⇒ D_i equivalent to $F..F – (D_i) ⇒ D_i

5.4.5.1 Logical AND

The AND instructions perform a bit-by-bit AND operation between a CPU data register and either an immediate operand or a memory operand that uses OPR addressing. The result replaces the contents of the original CPU data register. Because OPR addressing can be used to specify another CPU data register, this instruction can perform the Boolean AND between two CPU data registers. If the OPR addressing mode is used to specify a memory operand, it is assumed to be the same width as the source/destination register.

5.4.5.2 BIT (logical AND to set CCR but operand is left unchanged)

The BIT instruction has the same source forms as the AND instruction and it performs a bit-by-bit AND between the input operands and affects the condition code bits in the same way as the AND operation. However, rather than replacing the source CPU data register with the result of the AND operation, the input operands are left unchanged.

To use the BIT instructions, use the second input operand to specify a mask with 1's in all bit positions that are to be checked. If any of these bit positions are 1 in the source CPU data register, the result of the AND will be non-zero (the Z condition code bit will be cleared). In some programming cases, this can be more

efficient than using multiple BRCLR instructions to test several bits and it has the advantage of testing several bits at exactly the same time.

5.4.5.3 Logical OR

The OR instructions perform a bit-by-bit OR operation between a CPU data register and either an immediate operand or a memory operand that uses OPR addressing. The result replaces the contents of the original CPU data register. Because OPR addressing can be used to specify another CPU data register, this instruction can perform the Boolean OR between two CPU data registers. If the OPR addressing mode is used to specify a memory operand, it is assumed to be the same width as the source/destination register.

5.4.5.4 Logical Exclusive-OR

The Exclusive-OR instructions perform a bit-by-bit Exclusive-OR operation between a CPU data register and either an immediate operand or a memory operand that uses OPR addressing. The result replaces the contents of the original CPU data register. Because OPR addressing can be used to specify another CPU data register, this instruction can perform the Boolean Exclusive-OR between two CPU data registers. If the OPR addressing mode is used to specify a memory operand, it is assumed to be the same width as the source/destination register. Exclusive-OR is often used to perform a “toggle” function.

5.4.5.5 Invert (bit-by-bit Ones Complement)

Complement (COM) performs a bit-by-bit invert operation on an 8-bit, 16-bit, or 32-bit operand that is specified by the OPR addressing mode. Because the OPR addressing mode can be used to specify a CPU data register, a program can use this form to perform a COM operation on a CPU data register although it is slightly less efficient than having dedicated instructions to complement each register.

5.4.6 Shifts and Rotates

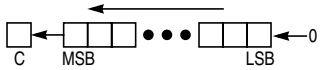
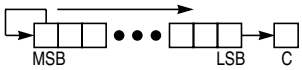
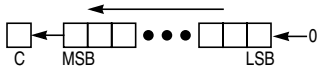
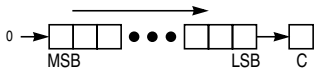
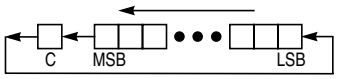
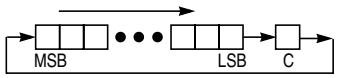
Shift operations have been significantly enhanced compared to previous generations of CPU12 and S12X. A 32-bit wide barrel shifter was added to allow very fast shifting by any number of bit positions rather than one bit position at a time. Three-operand versions of shift operations were added which allow the source and destination to be specified independently. This makes it possible to keep an unmodified version of the source operand as well as the shifted result. An arithmetic shift left instruction was added which performs the same shifting operation as the logical shift left, but the condition codes are handled differently so that multi-bit shifts of signed values can be handled differently than unsigned values.

More efficient two-operand shifts were also included to improve efficiency in some programs and to improve backward compatibility with earlier CPU12 and S12X instruction sets. The two-operand shifts only allow shifting by one or two positions at a time. The source/destination operand can be 8-bits, 16-bits, 24-bits, or 32-bits and uses OPR addressing. A CPU data register can be specified using the register-as-memory sub-mode.

Rotate instructions operate through the carry bit and they rotate by a single bit position at a time.

Table 5-6 shows a summary of the shift and rotate instructions.

Table 5-6. Shift and Rotate Instructions

Source Forms	Function	Operation
Arithmetic Shifts		
ASL *Dd*,*Ds*,*Dn* ASL *Dd*,*Ds*,#*opr1i* ASL *Dd*,*Ds*,#*opr5i* ASL.*bwpl Dd*,*oprmemreg*,#*opr1i* ASL.*bwpl Dd*,*oprmemreg*,#*opr5i* ASL.*bwpl Dd*,*oprmemreg*,*oprmemreg*	Arithmetic Shift Left D_s or memory, 0 to n positions	
ASL.*bwpl oprmemreg*,#*opr1i*	Arithmetic Shift Left memory by 1 or 2 positions	
ASR *Dd*,*Ds*,*Dn* ASR *Dd*,*Ds*,#*opr1i* ASR *Dd*,*Ds*,#*opr5i* ASR.*bwpl Dd*,*oprmemreg*,#*opr1i* ASR.*bwpl Dd*,*oprmemreg*,#*opr5i* ASR.*bwpl Dd*,*oprmemreg*,*oprmemreg*	Arithmetic Shift Right D_s or memory, 0 to n positions	
ASR.*bwpl oprmemreg*,#*opr1i*	Arithmetic Shift Right memory by 1 or 2 positions	
Logical Shifts		
LSL *Dd*,*Ds*,*Dn* LSL *Dd*,*Ds*,#*opr1i* LSL *Dd*,*Ds*,#*opr5i* LSL.*bwpl Dd*,*oprmemreg*,#*opr1i* LSL.*bwpl Dd*,*oprmemreg*,#*opr5i* LSL.*bwpl Dd*,*oprmemreg*,*oprmemreg*	Logical Shift Left D_s or memory, 0 to n positions.	
LSL.*bwpl oprmemreg*,#*opr1i*	Logical Shift Left memory by 1 or 2 position.	
LSR *Dd*,*Ds*,*Dn* LSR *Dd*,*Ds*,#*opr1i* LSR *Dd*,*Ds*,#*opr5i* LSR.*bwpl Dd*,*oprmemreg*,#*opr1i* LSR.*bwpl Dd*,*oprmemreg*,#*opr5i* LSR.*bwpl Dd*,*oprmemreg*,*oprmemreg*	Logical Shift Left D_s or memory, 0 to n positions.	
LSR.*bwpl oprmemreg*,#*opr1i*	Logical Shift Right memory by 1 or 2 position.	
Rotate (one bit position through carry C-bit)		
ROL.*bwpl oprmemreg*	Rotate Left through Carry D_i or memory, 1 bit position	
ROR.*bwpl oprmemreg*	Rotate Right through Carry D_i or memory, 1 bit position	

5.4.6.1 Arithmetic Shifts

Shift a signed operand left or right through carry (C) by 0 to 31 bit positions. Each left shift is effectively multiplying the operand by two. If the sign bit (MSB) would change value during a bit-by-bit left shift, it is considered a signed overflow. In the case of right shifts, arithmetic right shift maintains the value of the MSB as the value is shifted so the sign remains unchanged.

There are two-operand shifts and three-operand shifts. The two-operand shifts are limited to OPR addressing mode (although OPR addressing mode can specify a CPU data register), and the shift amount is limited to one or two bit positions. The three-operand shifts can shift by 0 to 31 bit positions and offer more choices for addressing modes.

5.4.6.2 Logical Shifts

Shift an unsigned binary operand left or right through carry (C) by 0 to 31 bit positions. The operands in logical shifts are not interpreted as signed values. The same addressing mode options are available for logical shifts as for arithmetic shifts and there are two-operand and three-operand versions.

5.4.6.3 Rotate Through Carry

Rotate source/destination operand by one bit position. The C bit in the condition code register is included in the rotation. These operations are not used in C but can be useful for assembly language programs to perform serial-to-parallel and parallel-to-serial conversions as well as for shifting very long operands that are more than 32-bits wide.

5.4.7 Bit and Bit Field Manipulation

BSET, BCLR, and BTGL allow any single bit in a CPU data register or an 8-bit, 16-bit, or 32-bit memory variable to be set, cleared, or toggled. These instructions are read-modify-write instructions. ANDCC and ORCC are used to clear or set multiple bits in the condition codes register CCL according to an immediate mask. There are alternate mnemonics that translate to ANDCC or ORCC with a specific mask value to set or clear the carry (C), interrupt mask (I), or overflow bit (V). The BFEXT and BFINS extract a 1 to 32 bit field from an operand or insert a 1 to 32-bit field into an operand. These instructions improve the bit-field operations in C.

Table 5-7 shows a summary of the bit manipulation and bit branch instructions.

Table 5-7. Bit and Bit Field Instructions (Sheet 1 of 3)

Source Forms	Function	Operation
Set, Clear, or Toggle Bits in Memory		
BSET *Di*,#*opr5i* BSET.*bwl oprmemreg*,#*opr5i* BSET.*bwl oprmemreg*,*Dn*	Set Bit n in Memory or in D_i C equal the original value of bitn in M or D_i (semaphore)	(M) | bitn ⇒ M or (D_i) | bitn ⇒ D_i

Table 5-7. Bit and Bit Field Instructions (Sheet 2 of 3)

Source Forms	Function	Operation
BCLR *Di*,#*opr5i* BCLR.*bwl oprmemreg*,#*opr5i* BCLR.*bwl oprmemreg*,*Dn*	Clear Bit n in Memory or in D_i C equal the original value of bitn in M or D_i (semaphore)	(M) & ~bitn ⇒ M or (D_i) & ~bitn ⇒ D_i
BTGL *Di*,#*opr5i* BTGL.*bwl oprmemreg*,#*opr5i* BTGL.*bwl oprmemreg*,*Dn*	Toggle Bit n in Memory or in D_i C equal the original value of bitn in M or D_i (semaphore)	(M) ^ bitn ⇒ M or (D_i) ^ bitn ⇒ D_i
Set or Clear Bits in the CCR		
ANDCC #*opr8i*	Bitwise AND CCL with immediate byte in Memory (Clear CCR bits that are 0 in the immediate mask) (S, X, and I can only be changed in supervisor state)	(CCL) & (M) ⇒ CCL
CLC	Clear Carry Bit *Translates to* ANDCC #$FE	0 ⇒ C
CLI	Clear I Bit; (I can only be changed in supervisor state) *Translates to* ANDCC #$EF (enables I interrupts)	0 ⇒ I
CLV	Clear Overflow Bit *Translates to* ANDCC #$FD	0 ⇒ V
ORCC #*opr8i*	Bitwise OR CCL with Immediate Mask (Set CCR bits that are 1 in the immediate mask) (S, X, and I can only be changed in supervisor state)	(CCL) | (M) ⇒ CCL
SEC	Set Carry Bit *Translates to* ORCC #$01	1 ⇒ C
SEI	Set I Bit; (inhibit I interrupts) *Translates to* ORCC #$10	1 ⇒ I
SEV	Set Overflow Bit *Translates to* ORCC #$02	1 ⇒ V

Table 5-7. Bit and Bit Field Instructions (Sheet 3 of 3)

Source Forms	Function	Operation
Bit Field Extract and Insert		
BFEXT *Dd*,*Ds*,*Dp* BFEXT *Dd*,*Ds*,#*width:offset* BFEXT.*bwpl Dd*,*oprmemreg*,*Dp* BFEXT.*bwpl oprmemreg*,*Ds*,*Dp* BFEXT.*bwpl Dd*,*oprmemreg*,#*width:offset* BFEXT.*bwpl oprmemreg*,*Ds*,#*width:offset*	Bit Field Extract	Extract bit field with width w and offset o from D_s or a memory operand, and store it into the low order bits of D_d or memory (filling unused bits with 0). The source operand or destination operand must be a register (memory to memory not allowed)
BFINS *Dd*,*Ds*,*Dp* BFINS *Dd*,*Ds*,#*width:offset* BFINS.*bwpl Dd*,*oprmemreg*,*Dp* BFINS.*bwpl oprmemreg*,*Ds*,*Dp* BFINS.*bwpl Dd*,*oprmemreg*,#*width:offset* BFINS.*bwpl oprmemreg*,*Ds*,#*width:offset*	Bit Field Insert	Insert bit field with width w from the low order bits of D_s or a memory operand into D_d or a memory operand beginning at offset bit number o. The source operand or destination operand must be a register (memory to memory not allowed)

5.4.7.1 Set, Clear, or Toggle Bits in Memory

These instructions read an operand, modify one bit in that operand, and then write the operand back to the original CPU data register or memory location. The operand can be a CPU data register or a memory operand using the OPR addressing mode. The bit number (0 to 31) is provided in a 5-bit immediate value or in the low order 5 bits of a CPU data register.

These instructions are also designed to allow a programmer to implement software semaphores. The original value of the selected bit is captured in the C bit so that software can tell if the current operation changed the bit rather than some other operation. This is sometimes called an “atomic operation” because the read and the change are done within a single uninterruptable instruction so there is no way for another program to change the bit after it was read but before it is changed. Software programs use these semaphores to control access to shared resources so that only one program can have control at a time.

5.4.7.2 Set or Clear Bits in the CCR

Clearing bits in the condition codes register (CCL) is done with an ANDCC instruction that includes a mask with zeros in the bit positions that are to be cleared and ones in the remaining positions. The instruction can clear more than one bit at a time. Three specific cases (CLC, CLI, and CLV) have alternate mnemonics so the programmer doesn’t need to remember the bit position to clear these bits. These three mnemonics are assembled into ANDCC instructions with the appropriate bit clear in the mask.

Setting bits in the condition codes register (CCL) is done with an ORCC instruction that includes a mask with ones in the bit positions that are to be set. The instruction can set more than one bit at a time. Three specific cases (SEC, SEI, and SEV) have alternate mnemonics so the programmer doesn’t need to remember the bit position to set these bits. These three mnemonics are assembled into ORCC instructions with the appropriate bit set in the mask.

5.4.7.3 Bit Field Extract and Insert

The bit field instructions operate on fields consisting of any number of adjacent bits in an operand. The fields are specified with a 5-bit width and a 5-bit offset. For example the binary value 00010:00100 selects a field 2 bits wide with the right-most bit at offset position 4 (bits 5:4 of the operand).

These instructions are designed for use by compilers to implement C bit-field functions. BFEXT extracts (copies) the field from the source operand, stores it to the low order bits of the destination operand, and fills the remaining bits of the destination operand with zeros (zero-extend). BFINS copies the field in the low order bits of the source operand and inserts it into the destination operand at the specified offset. The remaining bits in the read-write destination are not changed.

5.4.8 Maximum and Minimum Instructions

Maximum instructions compare a CPU data register to an operand that uses OPR addressing and stores the largest value in the CPU data register. MAXS treats the operands as twos complement signed values and MAXU treats the operands as unsigned values. OPR addressing allows the second operand to be another CPU data register, a short-immediate value, or a memory operand that is the same width as the source/destination register.

Minimum instructions compare a CPU data register to an operand that uses OPR addressing and stores the smallest value in the CPU data register. MINS treats the operands as twos complement signed values and MINU treats the operands as unsigned values. OPR addressing allows the second operand to be another CPU data register, a short-immediate value, or a memory operand that is the same width as the source/destination register.

Table 5-8 shows a summary of the maximum and minimum instructions.

Table 5-8. Maximum and Minimum Instructions

Source Forms	Function	Operation
MAXS *Di,oprmemreg*	MAXimum of two signed operands replaces D_i	$MAX((D_i), (M)) \Rightarrow D_i$
MAXU *Di,oprmemreg*	MAXimum of two unsigned operands replaces D_i	$MAX((D_i), (M)) \Rightarrow D_i$
MINS *Di,oprmemreg*	MINimum of two signed operands replaces D_i	$MIN((D_i), (M)) \Rightarrow D_i$
MINU *Di,oprmemreg*	MINimum of two unsigned operands replaces D_i	$MIN((D_i), (M)) \Rightarrow D_i$

5.4.9 Summary of Index and Stack Pointer Instructions

This section provides a summary of index and stack pointer instructions. All of these instructions appear in other sections of this chapter, but this section collects all of the instructions that are related to the index registers and stack pointer so that it is easier to understand what instructions are available for address and pointer calculations. Keep in mind that there are other load, store, add, subtract, and compare instructions (not included in this section) that are related to CPU data registers rather than the index and stack pointer registers.

The load, pull, and RTI instructions are used to read information from memory into CPU registers. In addition, the pull and RTI instructions automatically update the stack pointer as information is read from the stack. Store, push, SWI, and WAI are used to write information from CPU registers into memory. In addition, push, SWI, and WAI automatically update the stack pointer as information is written to the stack.

Add, subtract, and compare are a little different for the index registers and stack pointer. These address calculations use the load effective address instructions for most arithmetic calculations. 32-bit CPU registers D6 and D7 can also be used for address registers so there are LEA instructions for D6, D7, X, Y, and SP. Many or the sub-modes in the OPR addressing mode perform address calculations such as adding small constants to an index register or adding a CPU data register to an index register. LEA provides a way to save the results of these address calculations in an index or pointer register.

There are subtract instructions to subtract X–Y or Y–X and save the difference in the 32-bit D6 register. Finally there are instructions to compare X, Y, or SP to a 24-bit immediate value, a memory operand using OPR addressing, or for comparing X to Y.

Table 5-9 shows a summary of the index and pointer manipulation instructions.

Table 5-9. Index and Pointer Manipulation Instructions (Sheet 1 of 3)

Source Forms	Function	Operation
Load		

LD *xy*,#*opr18i*
LD *xy*,#*opr24i*
LD *xy*,*opr24a*
LD *xy*,*oprmemreg*
	Load index register X or Y from Memory	(M:M+1:M+2) ⇒ X or Y

LD S,#*opr24i* LD S,*oprmemreg*	Load stack pointer SP from Memory	(M:M+1:M+2) ⇒ SP
Pull (Load CPU Registers from Stack)		
PUL *oprregs1* PUL *oprregs2* PUL ALL PUL ALL16b	Pull specified CPU registers from Stack mask 1 - CCH, CCL, D0, D1, D2, D3 (D3 in LSB) mask 2 - D4, D5, D6, D7, X, Y (Y in LSB) pulls all registers in the same order as RTI	(M(SP)~M(SP+n-1)) ⇒ regs; (SP) + n ⇒ SP
Restore CPU Registers after Interrupts (RTI)		
RTI	Return from Interrupt	(M(SP)~M(SP+3)) ⇒ CCH:CCL, D0, D1; (SP)+4 ⇒ SP (M(SP)~M(SP+3)) ⇒ D2H:D2L, D3H:D3L; (SP)+4 ⇒ SP (M(SP)~M(SP+3)) ⇒ D4H:D4L, D5H:D5L; (SP)+4 ⇒ SP (M(SP)~M(SP+3)) ⇒ D6H:D6MH:D6ML:D6L; (SP)+4 ⇒ SP (M(SP)~M(SP+3)) ⇒ D7H:D7MH:D7ML:D7L; (SP)+4 ⇒ SP (M(SP)~M(SP+2)) ⇒ XH:XM:XL; (SP)+3 ⇒ SP (M(SP)~M(SP+2)) ⇒ YH:YM:YL; (SP)+3 ⇒ SP (M(SP)~M(SP+2)) ⇒ RTNH:RTNM:RTNL; (SP)+3 ⇒ SP
Store		

Table 5-9. Index and Pointer Manipulation Instructions (Sheet 2 of 3)

Source Forms	Function	Operation
ST *xy*,*opr24a* ST *xy*,*oprmemreg*	Store index register X or Y to Memory	(X) ⇒ (M:M+1:M+2), or (Y) ⇒ (M:M+1:M+2)
ST S,*oprmemreg*	Store stack pointer SP to Memory	(SP) ⇒ (M:M+1:M+2)
Push (Store CPU Registers on Stack)		
PSH *oprregs1* PSH *oprregs2* PSH ALL PSH ALL16b	Push specified CPU registers onto Stack mask 1 - CCH, CCL, D0, D1, D2, D3 (D3 in LSB) mask 2 - D4, D5, D6, D7, X, Y (Y in LSB) pushes registers in the same order as SWI	(SP) – n ⇒ SP; (regs) ⇒ M(SP)~M(SP+n-1)
Stack CPU Registers on Entry to Interrupts (SWI, WAI)		
SWI	Software Interrupt	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; YH:YM:YL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; XH:XM:XL ⇒ M(SP)~M(SP+2); (SP) – 4 ⇒ SP; D7H:D7MH:D7ML:D7L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D6H:D6MH:D6ML:D6L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D4H:D4L, D5H:D5L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D2H:D2L, D3H:D3L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; CCH, CCL, D0, D1 ⇒ M(SP)~M(SP+3); 0 ⇒ U; 1 ⇒ I; (SWI Vector) ⇒ PC
WAI	Wait for Interrupt	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; YH:YM:YL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; XH:XM:XL ⇒ M(SP)~M(SP+2); (SP) – 4 ⇒ SP; D7H:D7MH:D7ML:D7L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D6H:D6MH:D6ML:D6L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D4H:D4L, D5H:D5L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D2H:D2L, D3H:D3L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; CCH, CCL, D0, D1 ⇒ M(SP)~M(SP+3); when interrupt occurs, 1 ⇒ I; (Vector) ⇒ PC
Load Effective Address		
LEA D6,*oprmemreg* LEA D7,*oprmemreg*	Load Effective Address into 32-bit D6 or D7	00:Effective Address ⇒ D6, or 00:Effective Address ⇒ D7
LEA S,*oprmemreg* LEA X,*oprmemreg* LEA Y,*oprmemreg*	Load Effective Address into 24-bit X, Y, or SP	Effective Address ⇒ SP, or Effective Address ⇒ X, or Effective Address ⇒ Y
LEA S,(#*opr8i*,S) LEA X,(#*opr8i*,X) LEA Y,(#*opr8i*,Y)	Add sign-extended 8-bit Immediate to X, Y, or SP no change to CCR bits	(SP) + sign-extend (M) ⇒ SP, or (X) + sign-extend (M) ⇒ X, or (Y) + sign-extend (M) ⇒ Y
Subtract and Compare		
SUB D6,X,Y	Subtract without Carry	(X) – (Y) ⇒ D6
SUB D6,Y,X		(Y) – (X) ⇒ D6

Table 5-9. Index and Pointer Manipulation Instructions (Sheet 3 of 3)

Source Forms	Function	Operation
CMP *xy*,#*opr24i* CMP *xy*,*oprmemreg*	Compare X or Y with Memory	(xy) – (M:M+1:M+2)
CMP S,#*opr24i* CMP S,*oprmemreg*	Compare stack pointer SP with Memory	(SP) – (M:M+1:M+2)
CMP X,Y	Compare X with Y	(X) – (Y)

5.4.9.1 Load

Load instructions allow the 24-bit index registers or stack pointer to be loaded with a 24-bit immediate value or a 24-bit memory operand. Usually, other instructions use the 24-bit extended sub-mode of OPR addressing to handle loads from anywhere in memory, but for X, Y, and SP, there are more efficient 4-byte instructions to access global memory space.

There are even more efficient instructions to load X or Y with an 18-bit immediate value. These instructions work with other addressing modes such as 18-bit extended and 18-bit offset indexed modes to work more efficiently with control registers and RAM variables in the first 256K of the memory map.

5.4.9.2 Pull and RTI

These instructions are included in this section because they include automatic stack pointer updates during execution. When any value is pulled (read) from the stack, the stack pointer is automatically post-incremented by the number of bytes in the value that was pulled so that SP points at the next value on the stack. In addition, the values in X and Y are restored by the RTI instruction to values that were previously saved on the stack.

5.4.9.3 Store

Store instructions allow the 24-bit index registers or stack pointer to be stored in memory using OPR addressing mode. Usually, other instructions use the 24-bit extended sub-mode of OPR addressing to handle stores to anywhere in memory, but for X and Y there are more efficient 4-byte instructions to access global memory space.

5.4.9.4 Push, SWI, and WAI

These instructions are included in this section because they include automatic stack pointer updates during execution. When any value is pushed (written) onto the stack, the stack pointer is automatically pre-decremented by the number of bytes in the value that will be pushed. In addition, the values in X and Y are saved (stored) during the SWI and WAI instructions.

5.4.9.5 Load Effective Address (including signed addition)

There are two types of LEA instructions. The first type uses the OPR addressing modes and the effective address that is internally computed gets stored into D6, D7, X, Y, or SP rather than being used to access a

memory operand. The second type adds an 8-bit immediate signed value to X, Y, or SP so it provides a very efficient way to adjust an index register or SP by a value between –128 and +127.

There are LEA instructions for D6 and D7 because these 32-bit CPU registers can be used for extra index registers in some application programs.

The quick immediate sub mode and the register as memory sub mode of the OPR addressing mode are not appropriate for use with LEA because these two sub modes do not access any memory operands and therefore do not compute an effective address. For all other OPR sub modes, an effective address is internally computed by the CPU. In the case of a load data register instruction, this effective address would be used to read data from memory, and in the case of a load effective address instruction (LEA), this effective address is saved in the selected index or pointer register.

5.4.9.6 Subtract and Compare

The subtract instructions subtract X–Y or Y–X and save the difference in the 32-bit D6 register. This allows an easy way to find the difference between two pointers (one in X and the other in Y).

Compare instructions compare X, Y, or SP to a 24-bit immediate value or a memory operand using OPR addressing. There is also an instruction for comparing X to Y. After a compare instruction, a program can execute signed or unsigned conditional branch instructions to control the flow of the program.

5.5 Program Control Instructions

Program control instructions manage the structure and flow of programs while the previously described register and memory instructions were used to manipulate data and perform computations. Program control instructions include branches, jumps, subroutine calls and returns, interrupt entry and return, and a few miscellaneous instructions like stop, wait, no operation (NOP), and background debug entry.

5.5.1 Branch Instructions

All branch instructions in the S12Z have two possible offset ranges which determine how far these branches can send the program when the branch conditions are true. The choice between the shorter and the longer ranges is controlled by the most significant bit of the first byte of object code for the branch offset. A 1-byte offset includes this range select bit (0 in the MSB) and 7 bits of signed offset so this shorter branch can reach from –64 to +63 locations from the address of the first byte of object code for the branch instruction. When the range select bit is set (1 in the MSB), it indicates a 2-byte offset with the longer range. The 2-byte offset includes the range select bit (1 in the MSB) and 15 bits of signed offset so this longer branch can reach from –16,536 to +16,535 locations from the address of the first byte of object code for the branch instruction.

Most branch instructions in a typical application program can use the shorter range and the shorter branches require one less byte of object code than the longer branches. If a program ever needs to branch farther than +/–16K (which is very unusual), the programmer or compiler can choose an opposite branch around a jump instruction which can reach anywhere in memory.

There are four main kinds of branches in the S12Z. Unconditional branches include the BRA and BSR instructions where the branch is always taken. Common CCR-based branches include simple branches

based on a single CCR bit, signed branches, and unsigned branches. Bit-value branches use the state of a selected bit in a selected CPU data register or memory location to decide whether or not to branch. The third kind of branches are the loop control branches which decrement or test a counter in a CPU register or memory location and then branch on the conditions Not Equal, Equal, Plus, Minus, Greater Than, or Less Than or Equal.

Table 5-10 shows a summary of the conditional branch instructions.

Table 5-10. Conditional Branch Instructions (Sheet 1 of 2)

Source Forms	Function	Operation
Unconditional Branches		
BRA *oprdest*	Branch Always	(if 1 = 1) Branch offset is 7 bits or 15 bits
BSR *oprdest*	Branch to Subroutine	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP):M(SP+1):M(SP+2); Subroutine Address ⇒ PC Branch offset is 7 bits or 15 bits
Simple Branches		
BCC *oprdest*	Branch if Carry Clear (same as BHS)	(if C = 0) Branch offset is 7 bits or 15 bits
BCS *oprdest*	Branch if Carry Set (same as BLO)	(if C = 1) Branch offset is 7 bits or 15 bits
BEQ *oprdest*	Branch if Equal	(if Z = 1) Branch offset is 7 bits or 15 bits
BMI *oprdest*	Branch if Minus	(if N = 1) Branch offset is 7 bits or 15 bits
BNE *oprdest*	Branch if Not Equal; R ≠ 0	(if Z = 0) Branch offset is 7 bits or 15 bits
BPL *oprdest*	Branch if Plus	(if N = 0) Branch offset is 7 bits or 15 bits
BVC *oprdest*	Branch if Overflow Bit Clear	(if V = 0) Branch offset is 7 bits or 15 bits
BVS *oprdest*	Branch if Overflow Bit Set	(if V = 1) Branch offset is 7 bits or 15 bits
Signed Branches		
BGE *oprdest*	Branch if Greater Than or Equal; signed R ≥ M	(if N ^ V = 0) Branch offset is 7 bits or 15 bits
BGT *oprdest*	Branch if Greater Than; signed R > M	(if Z | (N ^ V) = 0) Branch offset is 7 bits or 15 bits

Table 5-10. Conditional Branch Instructions (Sheet 2 of 2)

Source Forms	Function	Operation
BLE *oprdest*	Branch if Less Than or Equal; signed R ≤ M	(if Z | (N ^ V) = 1) Branch offset is 7 bits or 15 bits
BLT *oprdest*	Branch if Less Than; signed R < M	(if N ^ V = 1) Branch offset is 7 bits or 15 bits
Unsigned Branches		
BHI *oprdest*	Branch if Higher; unsigned R > M	(if C | Z = 0) Branch offset is 7 bits or 15 bits
BHS *oprdest*	Branch if Higher or Same; unsigned R ≥ M alternate mnemonic for BCC	(if C = 0) Branch offset is 7 bits or 15 bits
BLO *oprdest*	Branch if Lower; unsigned R < M alternate mnemonic for BCS	(if C = 1) Branch offset is 7 bits or 15 bits
BLS *oprdest*	Branch if Lower or Same; unsigned R ≤ M	(if C | Z = 1) Branch offset is 7 bits or 15 bits
Branch on Bit Value		
BRCLR *Di*,#*opr5i*,*oprdest* BRCLR.*bwl oprmemreg*,#*opr5i*,*oprdest* BRCLR.*bwl oprmemreg*,*Dn*,*oprdest*	Test Bit n in Memory or in D_i and branch if clear	Branch if (M) & bitn = 0 or if (D_i) & bitn = 0 Branch offset is 7 bits or 15 bits
BRSET *Di*,#*opr5i*,*oprdest* BRSET.*bwl oprmemreg*,#*opr5i*,*oprdest* BRSET.*bwl oprmemreg*,*Dn*,*oprdest*	Test Bit n in Memory or in D_i and branch if set	Branch if (M) & bitn ≠ 0 or if (D_i) & bitn ≠ 0 Branch offset is 7 bits or 15 bits
Loop Control Branches		
DB*cc Di*,*oprdest* DB*cc xy*,*oprdest* DB*cc.bwpl oprmemreg*,*oprdest*	Decrement D_i, X, Y, or memory operand M, and branch if condition cc is true. cc can be Not Equal-DBNE, Equal-DBEQ, Plus-DBPL, Minus-DBMI, Greater Than-DBGT, or Less Than or Equal-DBLE	(D_i) – 1 ⇒ D_i, or (X) – 1 ⇒ X, or (Y) – 1 ⇒ Y, or (M) – 1 ⇒ M then branch on selected condition Branch offset is 7 bits or 15 bits
TB*cc Di*,*oprmemreg*,*oprdest* TB*cc xy*,*oprmemreg*,*oprdest* TB*cc.bwpl oprmemreg*,*oprdest*	Test D_i, X, Y, or memory operand M, and branch if condition cc is true. cc can be Not Equal-DBNE, Equal-DBEQ, Plus-DBPL, Minus-DBMI, Greater Than-DBGT, or Less Than or Equal-DBLE	(D_i) – 0 ⇒ D_i, or (X) – 0 ⇒ X, or (Y) – 0 ⇒ Y, or (M) – 0 ⇒ M then branch on selected condition Branch offset is 7 bits or 15 bits

5.5.1.1 Unconditional Branches and Branch on CCR conditions

Branch instructions can also 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. These classifications are:

- The unconditional branch (BRA and BSR) 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.

If the branch conditions are true, execution continues at the specified destination address (branch taken), otherwise execution continues with the next instruction after the branch instruction (branch not taken). The branch is accomplished by conditionally adding the signed offset to the PC address of the branch instruction. The programmer specifies the destination as an address or program label and the assembler or compiler translates that into the appropriate signed offset value.

5.5.1.2 Branch on Bit Value

The bit condition branches are taken when the specified bit in a CPU data register or memory operand are in a specific state. A 5-bit operand provided by the instruction determines which bit in the operand will be tested for set (1) or clear (0). The 5-bit bit-number operand may be supplied as an immediate value or as the low order 5 bits of a CPU data register. The operand to be tested may be a CPU data register or a byte, word, or long-word memory operand.

If the tested bit is in the expected state, execution continues at the specified destination address (branch taken), otherwise execution continues with the next instruction after the branch instruction (branch not taken). The branch is accomplished by conditionally adding the signed offset to the PC address of the branch instruction. The programmer specifies the destination as an address or program label and the assembler or compiler translates that into the appropriate signed offset value.

5.5.1.3 Loop Control Branches (decrement and branch or test and branch)

Loop control branches use a loop count or loop control variable which is decremented or tested before determining whether the branch should be taken or not. The loop count or control variable may be one of the eight CPU data registers, X, Y, or a byte, word, pointer, or long-word variable in memory.

When the loop count is decremented by one for each pass through the loop, the decrement is included as part of the decrement-and-branch instruction. If the loop control variable will be adjusted by some amount other than –1 for each pass through the loop, the adjustment must be done with separate instructions in the loop and the loop control branch will test the control variable and then branch or not branch based on the value (test-and-branch).

Decrement and branch and Test and branch each have the same six choices for the branch condition. The branch conditions are Not Equal-DBNE/TBNE, Equal-DBEQ/TBEQ, Plus-DBPL/TBPL, Minus-DBMI/TBMI, Greater Than-DBGT/TBGT, and Less Than or Equal-DBLE/TBLE. If the loop test condition is true, execution continues at the specified destination address (branch taken), otherwise execution continues with the next instruction after the branch instruction (branch not taken). The branch is accomplished by conditionally adding the signed offset to the PC address of the branch instruction. The programmer specifies the destination as an address or program label and the assembler or compiler translates that into the appropriate signed offset value.

5.5.2 Jump

Jump (JMP) instructions cause immediate changes in sequence. The JMP instruction loads the PC with an address in the 16 megabyte memory map, and program execution continues at that address. The address can be provided as an absolute 24-bit address or determined by the general OPR address modes. The OPR sub modes include indexed, indexed indirect, and short extended addressing mode options. Because the quick immediate and register-as-memory sub modes of OPR addressing do not generate a memory address, these two sub modes are not appropriate for a jump instruction. The 24-bit extended version of the jump instruction is just as efficient as the 18-bit extended OPR sub mode and more efficient than the 24-bit extended sub mode of OPR addressing so the absolute 24-bit extended version of the instruction is preferred compared to those OPR sub modes.

Table 5-11. Jump and Subroutine Instructions

Source Forms	Function	Operation
JMP *opr24a* JMP *oprmemreg*	Jump (unconditional)	Effective Address ⇒ PC

5.5.3 Subroutine Calls and Returns

Subroutine instructions optimize the process of transferring control to a code segment that performs a particular task, and then returning to the main program. A branch to subroutine (BSR) or a jump to subroutine (JSR) can be used to initiate subroutines. A return address is stacked, then execution begins at the subroutine address. Subroutines may be located anywhere in the 16 megabyte memory space (where there is RAM, ROM, or flash memory) and are terminated with a return-from-subroutine (RTS) instruction. RTS unstacks the return address so that execution resumes with the instruction after BSR or JSR.

Because JSR instructions are used often, there is a dedicated 24-bit extended addressing version of JSR to improve code-size efficiency. BSR is more efficient than this JSR when the subroutine is within about +/–64 or +/–16K of the calling instruction. JSR can specify the subroutine address with any of the usable OPR sub modes. Because the quick immediate and register-as-memory sub modes of OPR addressing do not generate a memory address, these two sub modes are not appropriate for a JSR instruction. The 24-bit extended version of the JSR instruction is just as efficient as the 18-bit extended OPR sub mode and more efficient than the 24-bit extended sub mode of OPR addressing so the absolute 24-bit extended version of the instruction is preferred compared to those OPR sub modes.

Table 5-12 shows a summary of the jump and subroutine instructions.

Table 5-12. Jump and Subroutine Instructions (Sheet 1 of 2)

Source Forms	Function	Operation
JSR *opr24a* JSR *oprmemreg*	Jump to Subroutine	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP):M(SP+1):M(SP+2); Subroutine Address ⇒ PC

Table 5-12. Jump and Subroutine Instructions (Sheet 2 of 2)

Source Forms	Function	Operation
BSR *oprdest*	Branch to Subroutine	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP):M(SP+1):M(SP+2); Subroutine Address ⇒ PC Branch offset is 7 bits or 15 bits
RTS	Return from Subroutine	(M(SP):M(SP+1):M(SP+2)) ⇒ PCH:PCM:PCL; (SP) + 3 ⇒ SP

5.5.4 Interrupt Handling

Interrupt instructions handle transfer of control to an interrupt service routine (ISR) that performs a time-critical task. There are five instructions related to stacking and interrupt entry, one related to returning to the main program, and two for enabling and disabling the maskable interrupts. Interrupt service routines are similar to subroutines in that they are separate blocks of program code that are executed outside the normal flow of the main program, but they differ from subroutines in two ways. First, all interrupts except SWI/SYS or TRAP/SPARE are triggered by system events outside the normal flow of (and typically asynchronous to) the main program, and second because all of the CPU registers are saved on the stack for interrupts while only the return PC is automatically saved for subroutines. Software interrupts (SWI, SYS) can be thought of as a JSR instruction that saves all of the CPU registers.

The way the program returns from an interrupt is to restore all of the CPU registers from the stack in the reverse of the order they were saved as the program entered the interrupt. The last CPU register to be restored is the PC of the instruction that would have executed next if the interrupt had not occurred. RTS was used to return from a subroutine and a program would use an RTI to return from an interrupt. Chapter 7, “Exceptions” covers interrupt exception processing in more detail.

Interrupts also have the concept of masking so that they can be prevented from interrupting the main program at times when it might be bad to be interrupted. Some critical interrupt sources such as the COP watchdog or voltage monitors are considered too important to be disabled even for a short time. The majority of interrupts such as those from I/O pins or on-chip peripheral modules, are maskable by the I control bit in the CCR. When I is set (1), so-called I-interrupts are temporarily blocked until the I bit is cleared. Interrupts that occur while I=1 are considered pending but normal processing continues undisturbed until I is cleared. When I is cleared, the highest priority pending interrupt is serviced first. As an ISR is entered, the I bit becomes set so that the CPU does not get stuck in an infinite loop trying to respond to the interrupt source. Generally, the interrupt is cleared in the ISR before returning to the main program. It is possible to clear I within an ISR, but this allows nested interrupts which require more programming skill to use properly. CLI assembles or compiles to an ANDCC instruction with a mask bit cleared corresponding to the I bit in the CCR. SEI assembles or compiles to an ORCC instruction with a mask bit set corresponding to the I bit in the CCR.

Table 5-13 shows a summary of the interrupt instructions.

Table 5-13. Interrupt Instructions (Sheet 1 of 2)

Source Forms	Function	Operation
Interrupt Stacking		
SWI	Software Interrupt	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; YH:YM:YL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; XH:XM:XL ⇒ M(SP)~M(SP+2); (SP) – 4 ⇒ SP; D7H:D7MH:D7ML:D7L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D6H:D6MH:D6ML:D6L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D4H:D4L, D5H:D5L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D2H:D2L, D3H:D3L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; CCH, CCL, D0, D1 ⇒ M(SP)~M(SP+3); 0 ⇒ U; 1 ⇒ I; (SWI Vector) ⇒ PC
SYS	System Call Software Interrupt	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; YH:YM:YL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; XH:XM:XL ⇒ M(SP)~M(SP+2); (SP) – 4 ⇒ SP; D7H:D7MH:D7ML:D7L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D6H:D6MH:D6ML:D6L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D4H:D4L, D5H:D5L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D2H:D2L, D3H:D3L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; CCH, CCL, D0, D1 ⇒ M(SP)~M(SP+3); 0 ⇒ U; 1 ⇒ I; (SYS Vector) ⇒ PC
TRAP *#trapnum*	Unimplemented (pg2) Opcode Trap Interrupt	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; YH:YM:YL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; XH:XM:XL ⇒ M(SP)~M(SP+2); (SP) – 4 ⇒ SP; D7H:D7MH:D7ML:D7L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D6H:D6MH:D6ML:D6L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D4H:D4L, D5H:D5L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D2H:D2L, D3H:D3L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; CCH, CCL, D0, D1 ⇒ M(SP)~M(SP+3); 0 ⇒ U; 1 ⇒ I; (TRAP Vector) ⇒ PC
SPARE	Unimplemented pg1 Opcode Trap Interrupt	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; YH:YM:YL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; XH:XM:XL ⇒ M(SP)~M(SP+2); (SP) – 4 ⇒ SP; D7H:D7MH:D7ML:D7L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D6H:D6MH:D6ML:D6L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D4H:D4L, D5H:D5L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D2H:D2L, D3H:D3L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; CCH, CCL, D0, D1 ⇒ M(SP)~M(SP+3); 0 ⇒ U; 1 ⇒ I; (pg1 TRAP Vector) ⇒ PC

Table 5-13. Interrupt Instructions (Sheet 2 of 2)

Source Forms	Function	Operation
WAI	Wait for Interrupt	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; YH:YM:YL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; XH:XM:XL ⇒ M(SP)~M(SP+2); (SP) – 4 ⇒ SP; D7H:D7MH:D7ML:D7L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D6H:D6MH:D6ML:D6L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D4H:D4L, D5H:D5L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D2H:D2L, D3H:D3L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; CCH, CCL, D0, D1 ⇒ M(SP)~M(SP+3); when interrupt occurs, 1 ⇒ I; (Vector) ⇒ PC
Interrupt Return		
RTI	Return from Interrupt	(M(SP)~M(SP+3)) ⇒ CCH:CCL, D0, D1; (SP)+4 ⇒ SP (M(SP)~M(SP+3)) ⇒ D2H:D2L, D3H:D3L; (SP)+4 ⇒ SP (M(SP)~M(SP+3)) ⇒ D4H:D4L, D5H:D5L; (SP)+4 ⇒ SP (M(SP)~M(SP+3)) ⇒ D6H:D6MH:D6ML:D6L; (SP)+4 ⇒ SP (M(SP)~M(SP+3)) ⇒ D7H:D7MH:D7ML:D7L; (SP)+4 ⇒ SP (M(SP)~M(SP+2)) ⇒ XH:XM:XL; (SP)+3 ⇒ SP (M(SP)~M(SP+2)) ⇒ YH:YM:YL; (SP)+3 ⇒ SP (M(SP)~M(SP+2)) ⇒ RTNH:RTNM:RTNL; (SP)+3 ⇒ SP
Interrupt Enable and Disable		
CLI	Clear I Bit; (I can only be changed in supervisor state) *Translates to* ANDCC #$EF (enables I interrupts)	0 ⇒ I
SEI	Set I Bit; (inhibit I interrupts) *Translates to* ORCC #$10	1 ⇒ I

The SYS instruction is similar to SWI except that is located on page 2 of the opcode map (requires 2 bytes of object code instead of 1) and it uses a separate vector from SWI. SYS provides for a way to change from user state to supervisor state (change U from 1 to 0). This is not usually possible using other instructions in a user state program because all instructions except SYS/SWI and TRAP/SPARE are prevented from changing U from 1 to 0. SWI could be used to change from user to supervisor state, but SWI is sometimes used by debug programs so a separate SYS instruction was included.

A TRAP exception is caused by any unimplemented opcode on page 2 of the opcode map. TRAP causes an exception using the separate TRAP vector. The TRAP ISR can determine which unimplemented opcode caused the TRAP exception by checking the return address on the stack and then reading the instruction opcode from memory at the two bytes before the return address.

SPARE is similar to TRAP except it is caused by execution of an unimplemented (spare) opcode on page 1 of the opcode map and it uses a separate vector. It is important to distinguish between page 1 and page 2 unimplemented opcodes so that the ISR knows where to look in memory for the unimplemented opcode that was responsible for the exception.

WAI causes the CPU to save the CPU register context on the stack as if an exception had occurred, and then suspend processing until an exception does occur. This can be useful to synchronize program execution to an event with less uncertainty. The event could be an external signal or a system event that is effectively independent of the running program such as a timer event or a received character. WAI reduces response time to the interrupt by stacking the registers on entry to wait so that this doesn't need to be done when the interrupt arrives. WAI also eliminates the uncertainty of waiting for the current instruction to complete before responding to the interrupt.

Wait instructions can only be executed when the CPU is in supervisor state. In user state WAI acts similar to a NOP instruction and execution continues to the next instruction. This helps prevent accidental entry into standby modes.

5.5.5 Miscellaneous Instructions

There are a few more instructions that do not fit neatly into any of the categories discussed so far. These instructions are used to place the MCU system in low power operating modes, a no-operation instruction which doesn't do anything except take up a byte of program space and a bus cycle of execution time, and an instruction that can place the system in background debug mode for development purposes.

Table 5-14 shows these remaining miscellaneous instructions.

Table 5-14. Stop, Wait, NOP, and BGND Instructions (Sheet 1 of 2)

Source Forms	Function	Operation
Low-Power Stop and Wait		
STOP	STOP All Clocks and enter a low power state If S control bit = 1, the STOP instruction is disabled and acts like a NOP.	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; YH:YM:YL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; XH:XM:XL ⇒ M(SP)~M(SP+2); (SP) – 4 ⇒ SP; D7H:D7MH:D7ML:D7L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D6H:D6MH:D6ML:D6L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D4H:D4L, D5H:D5L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D2H:D2L, D3H:D3L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; CCH, CCL, D0, D1 ⇒ M(SP)~M(SP+3); STOP All Clocks
WAI	Wait for Interrupt	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; YH:YM:YL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; XH:XM:XL ⇒ M(SP)~M(SP+2); (SP) – 4 ⇒ SP; D7H:D7MH:D7ML:D7L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D6H:D6MH:D6ML:D6L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D4H:D4L, D5H:D5L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D2H:D2L, D3H:D3L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; CCH, CCL, D0, D1 ⇒ M(SP)~M(SP+3); when interrupt occurs, 1 ⇒ I; (Vector) ⇒ PC
No Operation (NOP)		
NOP	No operation	–

Table 5-14. Stop, Wait, NOP, and BGND Instructions (Sheet 2 of 2)

Source Forms	Function	Operation
Enter Active Background Debug Mode (BGND)		
BGND	Enter active Background mode	enter Background if BDM enabled; else continue

5.5.5.1 Low Power (Stop and Wait)

Two instructions put the S12Z CPU in inactive states that reduce power consumption. The stop instruction (STOP) stacks a return address and the contents of CPU registers and accumulators, then halts all system clocks. Refer to the data sheet for a specific MCU to learn more about variations of stop modes. Stop modes are commonly used to reduce system power to an absolute minimum when there is no processing to be done. A common power-saving strategy is to keep the system in stop mode most of the time and wake up briefly at regular intervals to check to see if any new activity needs processing attention.

The wait instruction (WAI) stacks a return address and the contents of CPU registers, then waits for an interrupt service request; however, system clock signals continue to run. This reduces power by placing the CPU in a standby state, but for more significant power savings, stop modes are recommended.

The time needed to wake up from stop or wait modes depends upon how much circuitry was shut down during the stop or wait mode. Wait mode leaves regulators turned on and clocks running so the wake-up time is very fast. The lowest power stop modes turn off clocks, oscillators, and, in some technologies, even regulator power to large sections of the MCU. In those cases, extra time is needed to get regulators running and stable as well as oscillator startup time. Refer to the data sheet for each specific MCU for more details about stop modes and wake-up times.

Stop and wait instructions can only be executed when the CPU is in supervisor state. In user state these instructions act similar to NOP instructions and execution continues to the next instruction. This helps prevent accidental entry into standby modes.

5.5.5.2 No Operation (NOP)

Null operations are often used to replace other instructions during software debugging. Null operations can also be used in software delay programs to consume execution time without disturbing the contents of other CPU registers or memory; however, using instruction delays to control program timing is discouraged because maintaining such programs is difficult as processor technology advances and speeds increase.

5.5.5.3 Go to active background debug mode (BGND)

Background debug mode (BDM) is a special S12Z operating mode that is used for system development and debugging. Executing enter background debug mode (BGND) when BDM is enabled puts the S12Z in this mode. In normal application programs, there is no debug tool connected to the system and the background debug mode is disabled by ENBDM=0 so the BGND instruction acts similar to a NOP instruction. This feature is intended to prevent accidental entry into background debug mode when there is no debug system connected.

Chapter 6
Instruction Glossary

6.1 Introduction

This section is a comprehensive reference to the Linear S12Z CPU instruction set.

6.2 Glossary

This subsection contains an entry for each assembly mnemonic, in alphabetic order.

ABS — Absolute Value — ABS

Operation

$|(D_i)| \Rightarrow D_i$

Syntax Variations	Addressing Modes
ABS *Di*	INH

Description

Replace the content of D*i* with its absolute value. Operation size depends on (matches) the size of D*i*.

If the content of D*i* is negative, it is replaced with its two's complement value. If the content of D*i* is either positive, zero or a two's complement overflow occurred, D*i* remains unchanged. Two's complement overflow occurs only when the original value has its MSB set and all other bits clear (the most negative value possible for the size, that is either 0x80, 0x8000, or 0x80000000), two's complement overflow occurs because it is not possible to express a positive two's complement value with the same magnitude.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	–

N: Set if a two's complement overflow resulted from the operation. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Set if a two's complement overflow resulted from the operation. Cleared otherwise.

Detailed Instruction Formats

INH

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	1	0	0	0	SD REGISTER D*i*			4n

1B 4n ABS *Di*

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* used for the source and destination for the operation (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

ADC — Add with Carry — ADC

Operation

(D*i*) + (M) + C ⇒ D*i*

Syntax Variations	Addressing Modes
ADC *Di,#oprimmsz*	**IMM1/2/4**
ADC *Di,oprmemreg*	**OPR/1/2/3**

Description

Add with carry to register D*i* and store the result to D*i*. When the operand is an immediate value, it has the same size as register D*i*. In the case of the general OPR addressing operand, *oprmemreg* can be a sign-extended immediate value (–1, 1, 2, 3..14, 15), a data register, a memory operand the same size as D*i* at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Cleared if the result is non-zero, unchanged otherwise to allow Z to reflect the cumulative result of an extended series if ADD and ADC instructions.

V: Set if a two's complement overflow resulted from the operation. Cleared otherwise.

C: Set if there is a carry from the MSB of the result. Cleared otherwise.

Detailed Instruction Formats

IMM1/2/4

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	1	0	1	0	SD REGISTER D*i*			5p
IMMEDIATE DATA								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								

```
1B 5p i1                    ADC     Di,#opr8i ;for Di = 8-bit D0 or D1
1B 5p i2 i1                 ADC     Di,#opr16i ;for Di = 16-bit D2, D3, D4, or D5
1B 5p i4 i3 i2 i1           ADC     Di,#opr32i ;for Di = 32-bit D6 or D7
```

OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	1	1	0	0	SD REGISTER D*i*			6n
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 6n xb                    ADC     Di,#oprsxe4i ;-1, +1, 2, 3...14, 15
1B 6n xb                    ADC     Di,Dj
1B 6n xb                    ADC     Di,(opru4,xys)
1B 6n xb                    ADC     Di,{(+-xy)|(xy+-)|(-s)|(s+)}
1B 6n xb                    ADC     Di,(Dj,xys)
1B 6n xb                    ADC     Di,[Dj,xy]
1B 6n xb x1                 ADC     Di,(oprs9,xysp)
1B 6n xb x1                 ADC     Di,[oprs9,xysp]
1B 6n xb x1                 ADC     Di,opru14
1B 6n xb x2 x1              ADC     Di,(opru18,Dj)
1B 6n xb x2 x1              ADC     Di,opru18
1B 6n xb x3 x2 x1           ADC     Di,(opr24,xysp)
1B 6n xb x3 x2 x1           ADC     Di,[opr24,xysp]
1B 6n xb x3 x2 x1           ADC     Di,(opru24,Dj)
1B 6n xb x3 x2 x1           ADC     Di,opr24
1B 6n xb x3 x2 x1           ADC     Di,[opr24]
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as a source operand and for the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size of the register D*i*.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

ADD — Add without Carry — ADD

Operation

(D*i*) + (M) ⇒ D*i*

Syntax Variations	Addressing Modes
ADD *Di,#oprimmsz*	**IMM1/2/4**
ADD *Di,oprmemreg*	**OPR/1/2/3**

Description

Add without carry to register D*i* and store the result to D*i*. When the operand is an immediate value, it has the same size as register D*i*. In the case of the general OPR addressing operand, *oprmemreg* can be a sign-extended immediate value (–1, 1, 2, 3..14, 15), a data register, a memory operand the same size as D*i* at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Set if a two's complement overflow resulted from the operation. Cleared otherwise.

C: Set if there is a carry from the MSB of the result. Cleared otherwise.

Detailed Instruction Formats

IMM1/2/4

7	6	5	4	3	2	1	0	
0	1	0	1	0	SD REGISTER D*i*			5p
IMMEDIATE DATA								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								

```
5p i1                    ADD    Di,#opr8i ;for Di = 8-bit D0 or D1
5p i2 i1                 ADD    Di,#opr16i ;for Di = 16-bit D2, D3, D4, or D5
5p i4 i3 i2 i1           ADD    Di,#opr32i ;for Di = 32-bit D6 or D7
```

OPR/1/2/3

7	6	5	4	3	2	1	0	
0	1	1	0	0	SD REGISTER D*i*			6n
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
6n xb                    ADD     Di,#oprsxe4i ;-1, +1, 2, 3...14, 15
6n xb                    ADD     Di,Dj
6n xb                    ADD     Di,(opru4,xys)
6n xb                    ADD     Di,{(+-xy)|(xy+-)|(-s)|(s+)}
6n xb                    ADD     Di,(Dj,xys)
6n xb                    ADD     Di,[Dj,xy]
6n xb x1                 ADD     Di,(oprs9,xysp)
6n xb x1                 ADD     Di,[oprs9,xysp]
6n xb x1                 ADD     Di,opru14
6n xb x2 x1              ADD     Di,(opru18,Dj)
6n xb x2 x1              ADD     Di,opru18
6n xb x3 x2 x1           ADD     Di,(opr24,xysp)
6n xb x3 x2 x1           ADD     Di,[opr24,xysp]
6n xb x3 x2 x1           ADD     Di,(opru24,Dj)
6n xb x3 x2 x1           ADD     Di,opr24
6n xb x3 x2 x1           ADD     Di,[opr24]
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as a source operand and for the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size of the register D*i*.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

AND — Bitwise AND — AND

Operation

(D*i*) & (M) ⇒ D*i*

Syntax Variations	Addressing Modes
AND *Di,#oprimmsz*	**IMM1/2/4**
AND *Di,oprmemreg*	**OPR/1/2/3**

Description

Bitwise AND register D*i* with a memory operand and store the result to D*i*. When the operand is an immediate value, it has the same size as register D*i*. In the case of the general OPR addressing operand, *oprmemreg* can be a sign-extended immediate value (–1, 1, 2, 3..14, 15), a data register, a memory operand the same size as D*i* at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	–

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Cleared.

Detailed Instruction Formats

IMM1/2/4

7	6	5	4	3	2	1	0	
0	1	0	1	1	SD REGISTER D*i*			5p
IMMEDIATE DATA								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								

```
5p i1                    AND     Di,#opr8i ;for Di = 8-bit D0 or D1
5p i2 i1                 AND     Di,#opr16i ;for Di = 16-bit D2, D3, D4, or D5
5p i4 i3 i2 i1           AND     Di,#opr32i ;for Di = 32-bit D6 or D7
```

OPR/1/2/3

7	6	5	4	3	2	1	0	
0	1	1	0	1	SD REGISTER D*i*			6q
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
6q xb                    AND     Di,#oprsxe4i ;-1, +1, 2, 3...14, 15
6q xb                    AND     Di,Dj
6q xb                    AND     Di,(opru4,xys)
6q xb                    AND     Di,{(+-xy)|(xy+-)|(-s)|(s+)}
6q xb                    AND     Di,(Dj,xys)
6q xb                    AND     Di,[Dj,xy]
6q xb x1                 AND     Di,(oprs9,xysp)
6q xb x1                 AND     Di,[oprs9,xysp]
6q xb x1                 AND     Di,opru14
6q xb x2 x1              AND     Di,(opru18,Dj)
6q xb x2 x1              AND     Di,opru18
6q xb x3 x2 x1           AND     Di,(opr24,xysp)
6q xb x3 x2 x1           AND     Di,[opr24,xysp]
6q xb x3 x2 x1           AND     Di,(opru24,Dj)
6q xb x3 x2 x1           AND     Di,opr24
6q xb x3 x2 x1           AND     Di,[opr24]
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as a source operand and for the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size of the register D*i*.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

ANDCC Bitwise AND CCL with Immediate ANDCC

Operation

(CCL) & (M) ⇒ CCL

Syntax Variations	Addressing Modes
ANDCC #*opr8i*	**IMM1**

Description

Performs a bitwise AND operation between the 8-bit immediate memory operand and the content of CCL (the low order 8 bits of the CCR). The result is stored in CCL.

When the CPU is in user state, this instruction is restricted to changing the condition codes (the flags N, Z, V, C) and cannot change the settings in the S, X, or I bits.

CCR Details

U	-	-	-	-	IPL			S	X	-	I	N	Z	V	C	
–	–	–	–	–	–	–	–	⇓	⇓	–	⇓	⇓	⇓	⇓	⇓	supervisor state
–	–	–	–	–	–	–	–	–	–	–	–	⇓	⇓	⇓	⇓	user state

Condition code bits are cleared if the corresponding bit in the immediate mask is 0. Condition code bits remain 0 if they were 0 before the operation.

Detailed Instruction Formats

IMM1

7	6	5	4	3	2	1	0	
1	1	0	0	1	1	1	0	CE
IMMEDIATE DATA								i1

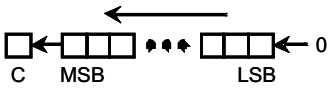
CE i1 ANDCC #*opr8i*

ASL

Arithmetic Shift Left

ASL

Operation



Syntax Variations	Addressing Modes
`ASL Dd,Ds,Dn`	**REG-REG**
`ASL Dd,Ds,#opr5i`	**REG-IMM (1-bit, or 5-bit)**
`ASL Dd,Ds,oprmemreg`	**REG-OPR/1/2/3**
`ASL.bwpl Dd,oprmemreg,#opr5i`	**OPR/1/2/3-IMM (1-bit, or 5-bit)**
`ASL.bwpl Dd,oprmemreg,oprmemreg`	**OPR/1/2/3-OPR/1/2/3**
`ASL Di,#opr1i ;2-operand, n=1 or 2`	**REG-IMM (2-operand)**
`ASL.bwpl oprmemreg,#opr1i ;2-operand, n=1 or 2`	**OPR/1/2/3-IMM (2-operand)**

Description

Arithmetically shifts an operand n bit-positions to the left. The result is saved in a CPU register, or in the case of a 2-operand shift the result is saved in the same memory location or register used for the source. The operand to be shifted may be one of the eight data registers or an 8-, 16-, 24-, or 32-bit memory operand. In the case of the general OPR addressing operand, *oprmemreg* can be a data register, a memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. The number of bit positions to shift the operand is supplied in a 1-bit or 5-bit immediate operand or in the low order 5 bits of a register or byte-sized memory operand. When the number of bit positions to shift is provided in a 5-bit immediate value, the least significant bit is encoded in the sb postbyte and the higher four bits are encoded as a short-immediate value in the xb postbyte. If the destination register is wider than the source operand, the source operand is sign-extended to the width of the destination register before shifting. If the destination register is narrower than the source operand, the operand is shifted and then truncated to the width of the destination register. Zero is shifted into the LSB and the MSB is shifted out through the carry bit (C).

The N-flag is set according to the inverted MSB of the operand. This can be used by the SAT instruction (together with the V-flag) to saturate the result.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the operand is zero. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Set if there is a signed overflow (if the MSB would change state during a bit-by-bit shift). Set if truncation changes the sign or magnitude of the result. Cleared otherwise.

C: Set if the last bit shifted out of the MSB of the operand was set before the shift, cleared otherwise. If the shift count is 0, C is not changed.

Detailed Instruction Formats

REG-REG

7	6	5	4	3	2 1 0	
0	0	0	1	0	DESTINATION REGISTER D*d*	1n
A/L=1	L/R=1	0	1	x	SOURCE REGISTER D*s*	sb
1	0	1	1	1	PARAMETER REGISTER D*n*	xb

```
1n sb xb                    ASL        Dd,Ds,Dn
```

REG-IMM (efficient shift by 1 (N[0]=0) or by 2 (N[0]=1) positions)

7	6	5	4	3	2 1 0	
0	0	0	1	0	DESTINATION REGISTER D*d*	1n
A/L=1	L/R=1	0	0	N[0]	SOURCE REGISTER D*s*	sb

```
1n sb                       ASL        Dd,Ds,#opr1i
```

REG-IMM (normal shift by 0 to 31 positions)

7	6	5	4	3	2 1 0	
0	0	0	1	0	DESTINATION REGISTER D*d*	1n
A/L=1	L/R=1	0	0	N[0]	SOURCE REGISTER D*s*	sb
0	1	1	1	N[4:1]		xb

```
1n sb xb                    ASL        Dd,Ds,#opr5i ;N[0] in sb, N[4:1] in xb
```

REG-OPR/1/2/3

7	6	5	4	3	2 1 0	
0	0	0	1	0	DESTINATION REGISTER D*d*	1n
A/L=1	L/R=1	0	0	x or N[0]	SOURCE REGISTER D*s*	sb
OPR POSTBYTE (specifes number of shifts in byte-sized memory operand)						xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)						
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)						
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)						

```
1n sb xb                    ASL        Dd,Ds,Dn ;see REG-REG
1n sb xb                    ASL        Dd,Ds,#opr5i ;see REG-IMM n=xb[3:0]:sb[3]
1n sb xb                    ASL        Dd,Ds,(opru4,xys)
1n sb xb                    ASL        Dd,Ds,{(+-xy)|(xy+-)|(-s)|(s+)}
1n sb xb                    ASL        Dd,Ds,(Di,xys)
1n sb xb                    ASL        Dd,Ds,[Di,xy]
1n sb xb x1                 ASL        Dd,Ds,(oprs9,xysp)
1n sb xb x1                 ASL        Dd,Ds,[oprs9,xysp]
1n sb xb x1                 ASL        Dd,Ds,opru14
1n sb xb x2 x1              ASL        Dd,Ds,(opru18,Di)
1n sb xb x2 x1              ASL        Dd,Ds,opru18
1n sb xb x3 x2 x1           ASL        Dd,Ds,(opr24,xysp)
1n sb xb x3 x2 x1           ASL        Dd,Ds,[opr24,xysp]
1n sb xb x3 x2 x1           ASL        Dd,Ds,(opru24,Di)
1n sb xb x3 x2 x1           ASL        Dd,Ds,opr24
1n sb xb x3 x2 x1           ASL        Dd,Ds,[opr24]
```

OPR/1/2/3-IMM (efficient shift by 1 (N[0]=0) or by 2 (N[0]=1) positions)

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=1	L/R=1	1	0	N[0]	0	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (specifes source operand to be shifted)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1n sb xb                    ASL.bwpl    #oprsxe4i,#opr1i
1n sb xb                    ASL.bwpl    Ds,#opr1i ;see more efficient REG-IMM version
1n sb xb                    ASL.bwpl    (opru4,xys),#opr1i
1n sb xb                    ASL.bwpl    {(+-xy)|(xy+-)|(-s)|(s+)},#opr1i
1n sb xb                    ASL.bwpl    (Di,xys),#opr1i
1n sb xb                    ASL.bwpl    [Di,xy],#opr1i
1n sb xb x1                 ASL.bwpl    (oprs9,xysp),#opr1i
1n sb xb x1                 ASL.bwpl    [oprs9,xysp],#opr1i
1n sb xb x1                 ASL.bwpl    opru14,#opr1i
1n sb xb x2 x1              ASL.bwpl    (opru18,Di),#opr1i
1n sb xb x2 x1              ASL.bwpl    opru18,#opr1i
1n sb xb x3 x2 x1           ASL.bwpl    (opr24,xysp),#opr1i
1n sb xb x3 x2 x1           ASL.bwpl    [opr24,xysp],#opr1i
1n sb xb x3 x2 x1           ASL.bwpl    (opru24,Di),#opr1i
1n sb xb x3 x2 x1           ASL.bwpl    opr24,#opr1i
1n sb xb x3 x2 x1           ASL.bwpl    [opr24],#opr1i
```

OPR/1/2/3-IMM (normal shift by 0 to 31 positions)

The upper four bits of the 5-bit number of shifts are in a second xb postbyte.

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=1	L/R=1	1	1	N[0]	0	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (specifes source operand to be shifted)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
0	1	1	1	N[4:1]				xb

```
1n sb xb xb                 ASL.bwpl    #oprsxe4i,#opr5i
1n sb xb xb                 ASL.bwpl    Di,#opr5i ;see more efficient REG-IMM version
1n sb xb xb                 ASL.bwpl    (opru4,xys),#opr5i
1n sb xb xb                 ASL.bwpl    {(+-xy)|(xy+-)|(-s)|(s+)},#opr5i
1n sb xb xb                 ASL.bwpl    (Di,xys),#opr5i
1n sb xb xb                 ASL.bwpl    [Di,xy],#opr5i
1n sb xb x1 xb              ASL.bwpl    (oprs9,xysp),#opr5i
1n sb xb x1 xb              ASL.bwpl    [oprs9,xysp],#opr5i
1n sb xb x1 xb              ASL.bwpl    opru14,#opr5i
1n sb xb x2 x1 xb           ASL.bwpl    (opru18,Di),#opr5i
1n sb xb x2 x1 xb           ASL.bwpl    opru18,#opr5i
1n sb xb x3 x2 x1 xb        ASL.bwpl    (opr24,xysp),#opr5i
1n sb xb x3 x2 x1 xb        ASL.bwpl    [opr24,xysp],#opr5i
1n sb xb x3 x2 x1 xb        ASL.bwpl    (opru24,Di),#opr5i
1n sb xb x3 x2 x1 xb        ASL.bwpl    opr24,#opr5i
1n sb xb x3 x2 x1 xb        ASL.bwpl    [opr24],#opr5i
```

OPR/1/2/3-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=1	L/R=1	1	1	x or N[0]	0	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (for source operand)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
OPR POSTBYTE (for number of shifts - byte sized memory operands)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

Opcode postbyte	Source operand object code	Parameter # of shifts object code	Instruction Mnemonic		Source Format for Source Operand (select 1 option in this col)	Source Format for Parameter (# of shifts) (select 1 option in this col)
1n sb	xb	xb	ASL.*bwpl*	Dd,	#*oprsxe4i*,	#*opr5i ;N[4:1] in xb*
	xb	xb			*Ds*,	*Dn*
	xb	xb			(*opru4*,*xys*),	(*opru4*,*xys*)
	xb	xb			(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+),	(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+)
	xb	xb			(*Dj*,*xys*),	(*Dk*,*xys*)
	xb	xb			[*Dj*,*xy*],	[*Dk*,*xy*]
	xb x1	xb x1			(*oprs9*,*xysp*),	(*oprs9*,*xysp*)
	xb x1	xb x1			[*oprs9*,*xysp*],	[*oprs9*,*xysp*]
	xb x1	xb x1			*opru14*,	*opru14*
	xb x2 x1	xb x2 x1			(*opru18*,*Dj*),	(*opru18*,*Dk*)
	xb x2 x1	xb x2 x1			*opru18*,	*opru18*
	xb x3 x2 x1	xb x3 x2 x1			(*opr24*,*xysp*),	(*opr24*,*xysp*)
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*,*xysp*],	[*opr24*,*xysp*]
	xb x3 x2 x1	xb x3 x2 x1			(*opru24*,*Dj*),	(*opru24*,*Dk*)
	xb x3 x2 x1	xb x3 x2 x1			*opr24*,	*opr24*
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*],	[*opr24*]

The .bwpl suffix on the instruction mnemonic refers to the size (byte, word, pointer, or long) of the source operand. The parameter operand is N[4:1]:N[0], the low five bits in a register Dn, or the low five bits in a byte sized memory operand.

OPR/1/2/3-IMM (2-operand register or memory shift by 1 (N[0]=0) or by 2 (N[0]=1) positions)

7	6	5	4	3	2	1	0	
0	0	0	1	0	x	x	x	1n
A/L=1	L/R=1	1	1	N[0]	1	x:x or SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (specifes source operand to be shifted)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1n sb xb                    ASL.bwpl    #oprsxe4i,#opr1i ;shifting IMM const not allowed
1n sb xb                    ASL         Di,#opr1i ;2-operand register shift by 1 or 2
1n sb xb                    ASL.bwpl    (opru4,xys),#opr1i
1n sb xb                    ASL.bwpl    {(+-xy)|(xy+-)|(-s)|(s+)},#opr1i
1n sb xb                    ASL.bwpl    (Di,xys),#opr1i
1n sb xb                    ASL.bwpl    [Di,xy],#opr1i
1n sb xb x1                 ASL.bwpl    (oprs9,xysp),#opr1i
1n sb xb x1                 ASL.bwpl    [oprs9,xysp],#opr1i
1n sb xb x1                 ASL.bwpl    opru14,#opr1i
1n sb xb x2 x1              ASL.bwpl    (opru18,Di),#opr1i
1n sb xb x2 x1              ASL.bwpl    opru18,#opr1i
1n sb xb x3 x2 x1           ASL.bwpl    (opr24,xysp),#opr1i
1n sb xb x3 x2 x1           ASL.bwpl    [opr24,xysp],#opr1i
1n sb xb x3 x2 x1           ASL.bwpl    (opru24,Di),#opr1i
1n sb xb x3 x2 x1           ASL.bwpl    opr24,#opr1i
1n sb xb x3 x2 x1           ASL.bwpl    [opr24],#opr1i
```

Instruction Fields

A/L - This bit selects arithmetic (1) or logical (0) shifts.

L/R - This bit selects the shift direction, left (1) or right (0).

DESTINATION REGISTER D*d* - This field specifies data register D*d* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) where the result of the shift is stored.

SOURCE REGISTER D*s* - This field specifies data register D*s* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) which is the source operand to be shifted.

PARAMETER REGISTER D*n* - This field specifies the number of the data register D*n* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) which is used to specify the number of positions (0–31) to shift the operand. Only the low-order 5 bits of the parameter register are used.

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as the source operand and as the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) for a 2-operand shift operation.

N[0] - This field contains the least significant bit of the 5-bit immediate operand n=0–31, or in the case of the efficient shifts, this bit selects shifting by 1 (N[0]=0) or shifting by 2 (N[0]=1).

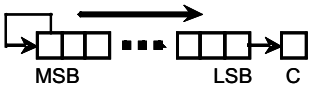
N[4:1] - This field contains the upper four bits of the 5-bit immediate operand n=0–31.

SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01), 24-bit pointer (0b10) or 32-bit long-word (0b11) as the size of the source operand.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. In the case of the parameter operand, short immediate mode is used to specify the upper four bits of the 5-bit immediate value that specifies the number of bit positions to shift the source operand.

ASR — Arithmetic Shift Right — ASR

Operation



Syntax Variations	Addressing Modes
`ASR Dd,Ds,Dn`	**REG-REG**
`ASR Dd,Ds,#opr5i`	**REG-IMM**
`ASR.bwpl Dd,oprmemreg,#opr5i`	**OPR/1/2/3-IMM**
`ASR.bwpl Dd,oprmemreg,oprmemreg`	**OPR/1/2/3-OPR/1/2/3**
`ASR.bwpl oprmemreg,#opr1i`	**OPR/1/2/3-IMM**

Description

Arithmetically shifts an operand n bit-positions to the right. The result is saved in a CPU register, or in the case of a 2-operand memory shift the result is saved in the same memory location used for the source. The operand to be shifted may be one of the eight data registers or an 8-, 16-, 24-, or 32-bit memory operand. In the case of the general OPR addressing operand, *oprmemreg* can be a data register, a memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. The number of bit positions to shift the operand is supplied in a 1-bit or 5-bit immediate operand or in the low order 5 bits of a register or byte-sized memory operand. When the number of bit positions to shift is provided in a 5-bit immediate value, the least significant bit is encoded in the sb postbyte and the higher four bits are encoded as a short-immediate value in the xb postbyte. If the destination register is wider than the source operand, the source operand is sign-extended to the width of the destination register before shifting. If the destination register is narrower than the source operand, the operand is shifted and then truncated to the width of the destination register. A copy of the original MSB sign value is shifted into the MSB and the LSB is shifted out through the carry bit (C).

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Normally cleared. Set if truncation changes the sign or magnitude of the result.

C: Set if the last bit shifted out of the LSB of the operand was set before the shift, cleared otherwise. If the shift count is 0, C is not changed.

Detailed Instruction Formats

REG-REG

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=1	L/R=0	1	0	N[0]	SOURCE REGISTER D*s*			sb
1	0	1	1	1	PARAMETER REGISTER D*n*			xb

```
1n sb xb                    ASR        Dd,Ds,Dn
```

REG-IMM (efficient shift by 1 (N[0]=0) or by 2 (N[0]=1) positions)

7	6	5	4	3	2	1	0	
0	0	0	0	0	DESTINATION REGISTER D*d*			1n
A/L=1	L/R=0	0	1	N[0]	SOURCE REGISTER D*s*			sb

```
1n sb                       ASR        Dd,Ds,#opr1i
```

REG-IMM (normal shift by 0 to 31 positions)

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=1	L/R=0	0	1	N[0]	SOURCE REGISTER D*s*			sb
0	1	1	1	N[4:1]				xb

```
1n sb xb                    ASR        Dd,Ds,#opr5i ;N[0] in sb, N[4:1] in xb
```

OPR/1/2/3-IMM (efficient shift by 1 (N[0]=0) or by 2 (N[0]=1) positions)

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=1	L/R=0	1	0	N[0]	0	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (specifes source operand to be shifted)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1n sb xb                    ASR.bwpl   #oprsxe4i,#opr1i
1n sb xb                    ASR.bwpl   Ds,#opr1i ;see more efficient REG-IMM version
1n sb xb                    ASR.bwpl   (opru4,xys),#opr1i
1n sb xb                    ASR.bwpl   {(+-xy)|(xy+-)|(-s)|(s+)},#opr1i
1n sb xb                    ASR.bwpl   (Di,xys),#opr1i
1n sb xb                    ASR.bwpl   [Di,xy],#opr1i
1n sb xb x1                 ASR.bwpl   (oprs9,xysp),#opr1i
1n sb xb x1                 ASR.bwpl   [oprs9,xysp],#opr1i
1n sb xb x1                 ASR.bwpl   opru14,#opr1i
1n sb xb x2 x1              ASR.bwpl   (opru18,Di),#opr1i
1n sb xb x2 x1              ASR.bwpl   opru18,#opr1i
1n sb xb x3 x2 x1           ASR.bwpl   (opr24,xysp),#opr1i
1n sb xb x3 x2 x1           ASR.bwpl   [opr24,xysp],#opr1i
1n sb xb x3 x2 x1           ASR.bwpl   (opru24,Di),#opr1i
1n sb xb x3 x2 x1           ASR.bwpl   opr24,#opr1i
1n sb xb x3 x2 x1           ASR.bwpl   [opr24],#opr1i
```

OPR/1/2/3-IMM (normal shift by 0 to 31 positions)

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=1	L/R=1	1	1	N[0]	0	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (specifes source operand to be shifted)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
0	1	1	1	N[4:1]				xb

```
1n sb xb xb                    ASR.bwpl    #oprsxe4i,#opr5i
1n sb xb xb                    ASR.bwpl    Di,#opr5i ;see more efficient REG-IMM version
1n sb xb xb                    ASR.bwpl    (opru4,xys),#opr5i
1n sb xb xb                    ASR.bwpl    {(+-xy)|(xy+-)|(-s)|(s+)},#opr5i
1n sb xb xb                    ASR.bwpl    (Di,xys),#opr5i
1n sb xb xb                    ASR.bwpl    [Di,xy],#opr5i
1n sb xb x1 xb                 ASR.bwpl    (oprs9,xysp),#opr5i
1n sb xb x1 xb                 ASR.bwpl    [oprs9,xysp],#opr5i
1n sb xb x1 xb                 ASR.bwpl    opru14,#opr5i
1n sb xb x2 x1 xb              ASR.bwpl    (opru18,Di),#opr5i
1n sb xb x2 x1 xb              ASR.bwpl    opru18,#opr5i
1n sb xb x3 x2 x1 xb           ASR.bwpl    (opr24,xysp),#opr5i
1n sb xb x3 x2 x1 xb           ASR.bwpl    [opr24,xysp],#opr5i
1n sb xb x3 x2 x1 xb           ASR.bwpl    (opru24,Di),#opr5i
1n sb xb x3 x2 x1 xb           ASR.bwpl    opr24,#opr5i
1n sb xb x3 x2 x1 xb           ASR.bwpl    [opr24],#opr5i
```

OPR/1/2/3-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=1	L/R=0	1	1	N[0]	0	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (for source operand)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
OPR POSTBYTE (for number of shifts - byte sized memory operands)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

Opcode postbyte	Source operand object code	Parameter # of shifts object code	Instruction Mnemonic		Source Format for Source Operand (select 1 option in this col)	Source Format for Parameter (# of shifts) (select 1 option in this col)
1n sb	xb	xb	ASR.*bwpl*	Dd,	#*oprsxe4i*,	#*oprsxe4i*
	xb	xb			*Ds*,	*Dn*
	xb	xb			(*opru4*,*xys*),	(*opru4*,*xys*)
	xb	xb			(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+),	(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+)
	xb	xb			(*Dj*,*xys*),	(*Dk*,*xys*)
	xb	xb			[*Dj*,*xy*],	[*Dk*,*xy*]
	xb x1	xb x1			(*oprs9*,*xysp*),	(*oprs9*,*xysp*)
	xb x1	xb x1			[*oprs9*,*xysp*],	[*oprs9*,*xysp*]
	xb x1	xb x1			*opru14*,	*opru14*
	xb x2 x1	xb x2 x1			(*opru18*,*Dj*),	(*opru18*,*Dk*)
	xb x2 x1	xb x2 x1			*opru18*,	*opru18*
	xb x3 x2 x1	xb x3 x2 x1			(*opr24*,*xysp*),	(*opr24*,*xysp*)
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*,*xysp*],	[*opr24*,*xysp*]
	xb x3 x2 x1	xb x3 x2 x1			(*opru24*,*Dj*),	(*opru24*,*Dk*)
	xb x3 x2 x1	xb x3 x2 x1			*opr24*,	*opr24*
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*],	[*opr24*]

The .bwpl suffix on the instruction mnemonic refers to the size (byte, word, pointer, or long) of the source operand. The parameter operand is always the low five bits in a byte sized memory operand.

OPR/1/2/3-IMM (2-operand memory shift by 1 (N[0]=0) or by 2 (N[0]=1) positions)

7	6	5	4	3	2	1	0	
0	0	0	1	0	x	x	x	1n
A/L=1	L/R=0	1	1	N[0]	1	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (specifes source operand to be shifted)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1n sb xb                    ASR.bwpl    #oprsxe4i,#opr1i
1n sb xb                    ASR.bwpl    Ds,#opr1i ;see more efficient REG-IMM version
1n sb xb                    ASR.bwpl    (opru4,xys),#opr1i
1n sb xb                    ASR.bwpl    {(+-xy)|(xy+-)|(-s)|(s+)},#opr1i
1n sb xb                    ASR.bwpl    (Di,xys),#opr1i
1n sb xb                    ASR.bwpl    [Di,xy],#opr1i
1n sb xb x1                 ASR.bwpl    (oprs9,xysp),#opr1i
1n sb xb x1                 ASR.bwpl    [oprs9,xysp],#opr1i
1n sb xb x1                 ASR.bwpl    opru14,#opr1i
1n sb xb x2 x1              ASR.bwpl    (opru18,Di),#opr1i
1n sb xb x2 x1              ASR.bwpl    opru18,#opr1i
1n sb xb x3 x2 x1           ASR.bwpl    (opr24,xysp),#opr1i
1n sb xb x3 x2 x1           ASR.bwpl    [opr24,xysp],#opr1i
1n sb xb x3 x2 x1           ASR.bwpl    (opru24,Di),#opr1i
1n sb xb x3 x2 x1           ASR.bwpl    opr24,#opr1i
1n sb xb x3 x2 x1           ASR.bwpl    [opr24],#opr1i
```

Instruction Fields

A/L - This bit selects arithmetic (1) or logical (0) shifts.

L/R - This bit selects the shift direction, left (1) or right (0).

DESTINATION REGISTER D*d* - This field specifies data register D*d* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) where the result of the shift is stored.

SOURCE REGISTER D*s* - This field specifies data register D*s* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) which is the source operand to be shifted.

PARAMETER REGISTER D*n* - This field specifies the number of the data register D*n* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) which is used to specify the number of positions (0–31) to shift the operand. Only the low-order 5 bits of the parameter register are used.

N[0] - This field contains the least significant bit of the 5-bit immediate operand n=0–31, or in the case of the efficient shifts, this bit selects shifting by 1 (N[0]=0) or shifting by 2 (N[0]=1).

N[4:1] - This field contains the upper four bits of the 5-bit immediate operand n=0–31.

SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01), 24-bit pointer (0b10) or 32-bit long-word (0b11) as the size of the source operand.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. In the case of the parameter operand, short immediate mode is used to specify the upper four bits of the 5-bit immediate value that specifies the number of bit positions to shift the source operand.

BCC — Branch if Carry Clear — BCC

Operation

If C = 0, then (PC) + REL ⇒ PC

Simple branch

Syntax Variations	Addressing Modes
BCC *oprdest*	REL

Description

Tests the C status bit. If C = 0 then program execution continues at location (PC) + REL

See Section 3.6, “Relative Addressing Modes (REL, REL1)” for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	0	1	0	0	24
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
24 rb                    BCC     oprdest ;Dest is within +63/-64 (7-bit offset)
24 rb r1                 BCC     oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

BCLR — Test and Clear Bit — BCLR

Operation

bitn of D*i* ⇒ C; then (D*i*) & ~bitn ⇒ D*i*

bitn of M ⇒ C; then (M) & ~bitn ⇒ M

Syntax Variations	Addressing Modes
`BCLR Di,#opr5i`	**REG-IMM**
`BCLR Di,Dn`	**REG-REG**
`BCLR.bwl oprmemreg,#opr5i`	**OPR/1/2/3-IMM**
`BCLR.bwl oprmemreg,Dn`	**OPR/1/2/3-REG**

Description

Tests and copies the original state of the specified bit into the C condition code bit to be used for semaphores. Then clears the specified bit in D*i* or a memory operand by performing a bitwise AND with a mask that has all bits set except the specified bit. The bit to be cleared is specified in a 5-bit immediate value or in the low order five bits of a data register D*n*. In the case of the general OPR addressing operand, *oprmemreg* can be a data register, an 8-, 16-, or 32-bit memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. *It is not appropriate to specify a short-immediate operand with the OPR addressing mode because it is not possible to modify (clear a bit in) the immediate operand.*

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	Δ

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Cleared.

C: Set if the bit being cleared was set before the operation. Cleared otherwise.

Detailed Instruction Formats

REG-IMM

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	0	EC
n[4:0]					SD REGISTER D*i*			bm

EC bm BCLR *Di,#opr5i*

REG-REG

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	0	EC
1	PARAMETER REGISTER D*n*			0	0	0	1	bm
1	0	1	1	1	SD REGISTER D*i*			xb

```
EC bm xb                    BCLR        Di,Dn
```

OPR/1/2/3-IMM

Byte-sized operand (.B)

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	0	EC
1	n[2:0]			0	0	0	0	bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

Word-sized operand (.W)

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	0	EC
1	n[2:0]			0	0	1	n[3]	bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

Long-word sized operand (.L)

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	0	EC
1	n[2:0]			1	0	n[4:3]		bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
EC sb xb                    BCLR.bwl    #oprsxe4i,#opr5i ;not appropriate for destination
EC sb xb                    BCLR.bwl    Di,#opr5i ;see more efficient REG-IMM1 version
EC sb xb                    BCLR.bwl    (opru4,xys),#opr5i
EC sb xb                    BCLR.bwl    {(+-xy)|(xy+-)|(-s)|(s+)},#opr5i
EC sb xb                    BCLR.bwl    (Di,xys),#opr5i
EC sb xb                    BCLR.bwl    [Di,xy],#opr5i
EC sb xb x1                 BCLR.bwl    (oprs9,xysp),#opr5i
EC sb xb x1                 BCLR.bwl    [oprs9,xysp],#opr5i
EC sb xb x1                 BCLR.bwl    opru14,#opr5i
EC sb xb x2 x1              BCLR.bwl    (opru18,Di),#opr5i
EC sb xb x2 x1              BCLR.bwl    opru18,#opr5i
EC sb xb x3 x2 x1           BCLR.bwl    (opr24,xysp),#opr5i
EC sb xb x3 x2 x1           BCLR.bwl    [opr24,xysp],#opr5i
EC sb xb x3 x2 x1           BCLR.bwl    (opru24,Di),#opr5i
EC sb xb x3 x2 x1           BCLR.bwl    opr24,#opr5i
EC sb xb x3 x2 x1           BCLR.bwl    [opr24],#opr5i
```

OPR/1/2/3-REG

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	0	EC
1	PARAMETER REGISTER D*n*			SIZE (.B-0:0, .W-0:1, .L-1:1)		0	1	bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
EC bm xb                    BCLR.bwl   #oprsxe4i,Dn ;not appropriate for destination
EC bm xb                    BCLR.bwl   Di,Dn
EC bm xb                    BCLR.bwl   (opru4,xys),Dn
EC bm xb                    BCLR.bwl   {(+-xy)|(xy+-)|(-s)|(s+)},Dn
EC bm xb                    BCLR.bwl   (Di,xys),Dn
EC bm xb                    BCLR.bwl   [Di,xy],Dn
EC bm xb x1                 BCLR.bwl   (oprs9,xysp),Dn
EC bm xb x1                 BCLR.bwl   [oprs9,xysp],Dn
EC bm xb x1                 BCLR.bwl   opru14,Dn
EC bm xb x2 x1              BCLR.bwl   (opru18,Di),Dn
EC bm xb x2 x1              BCLR.bwl   opru18,Dn
EC bm xb x3 x2 x1           BCLR.bwl   (opr24,xysp),Dn
EC bm xb x3 x2 x1           BCLR.bwl   [opr24,xysp],Dn
EC bm xb x3 x2 x1           BCLR.bwl   (opru24,Di),Dn
EC bm xb x3 x2 x1           BCLR.bwl   opr24,Dn
EC bm xb x3 x2 x1           BCLR.bwl   [opr24],Dn
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as a source operand and the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

PARAMETER REGISTER D*n* - This field specifies the number of the data register D*n* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) which is used to specify the bit number of the bit in the operand that is to be cleared. Only the low-order 5 bits of the parameter register are used.

n[4:0] - This field contains the 5-bit immediate parameter that specifies the bit number of the bit in the operand that is to be cleared.

SIZE - This field specifies 8-bit byte (0:0), 16-bit word (0:1), or 32-bit long-word (1:1) as the size of the operation.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Short immediate mode is not appropriate for instructions that store a result to the specified operand.

BCS — Branch if Carry Set — BCS

Operation

If C = 1, then (PC) + REL ⇒ PC

Simple branch

Syntax Variations	Addressing Modes
`BCS` *oprdest*	**REL**

Description

Tests the C status bit. If C = 1 then program execution continues at location (PC) + REL

See Section 3.6, “Relative Addressing Modes (REL, REL1)” for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	0	1	0	1	25
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
25 rb                    BCS        oprdest ;Dest is within +63/-64 (7-bit offset)
25 rb r1                 BCS        oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

BEQ — Branch if Equal — BEQ

Operation

If Z = 1, then (PC) + REL ⇒ PC

Simple branch

Syntax Variations	Addressing Modes
`BEQ` *oprdest*	**REL**

Description

Tests the Z status bit. If Z = 1 then program execution continues at location (PC) + REL

See Section 3.6, “Relative Addressing Modes (REL, REL1)” for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	0	1	1	1	27
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
27 rb                    BEQ        oprdest ;Dest is within +63/-64 (7-bit offset)
27 rb r1                 BEQ        oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

BFEXT — Bit Field Extract — BFEXT

Syntax Variations	Destination-Source-Parameter
`BFEXT Dd,Ds,Dp`	**REG-REG-REG**
`BFEXT Dd,Ds,#width:offset`	**REG-REG-IMM**
`BFEXT.bwplDd,oprmemreg,Dp`	**REG-OPR/1/2/3-REG**
`BFEXT.bwploprmemreg,Ds,Dp`	**OPR/1/2/3-REG-REG**
`BFEXT.bwplDd,oprmemreg,#width:offset`	**REG-OPR/1/2/3-IMM**
`BFEXT.bwploprmemreg,Ds,#width:offset`	**OPR/1/2/3-REG-IMM**

Description

Extracts a bit field from the specified source (register D*s* or memory location), if necessary zero extends to the width of the destination, and stores the result to the destination (register D*d* or memory location). The bit field width and offset are specified in the parameter (register D*p* or immediate operand). The field width determines the number of bits in the field (0b00000 is treated as 32). The field offset specifies the right-most starting bit of the field in D*s*.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	–

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: 0; Cleared.

Detailed Instruction Formats

REG-REG-REG

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	1	DESTINATION REGISTER D*d*			0q
0	0	0	SOURCE REGISTER D*s*			PARAMETER REGISTER		bb

`1B 0q bb` `BFEXT` *`Dd,Ds,Dp`*

REG-REG-IMM

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	1	DESTINATION REGISTER D*d*			0q
0	0	1	SOURCE REGISTER D*s*			WIDTH[4:3]		bb
WIDTH[2:0]			OFFSET[4:0]					i1

`1B 0q bb i1` `BFEXT` *`Dd,Ds,#width:offset`*

REG-OPR/1/2/3-REG

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	1	DESTINATION REGISTER D*d*			0q
0	1	0	0	SIZE (.B, .W, .P, .L)		PARAMETER REG D*p*		bb
OPR POSTBYTE (specifies source)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 0q bb xb                    BFEXT.bwpl  Dd,#oprsxe4i,Dp ;-1, +1, 2, 3...14, 15
1B 0q bb xb                    BFEXT.bwpl  Dd,Ds,Dp ;see more efficient REG-REG-REG version
1B 0q bb xb                    BFEXT.bwpl  Dd,(opru4,xys),Dp
1B 0q bb xb                    BFEXT.bwpl  Dd,{(+-xy)|(xy+-)|(-s)|(s+)},Dp
1B 0q bb xb                    BFEXT.bwpl  Dd,(Di,xys),Dp
1B 0q bb xb                    BFEXT.bwpl  Dd,[Di,xy],Dp
1B 0q bb xb x1                 BFEXT.bwpl  Dd,(oprs9,xysp),Dp
1B 0q bb xb x1                 BFEXT.bwpl  Dd,[oprs9,xysp],Dp
1B 0q bb xb x1                 BFEXT.bwpl  Dd,opru14,Dp
1B 0q bb xb x2 x1              BFEXT.bwpl  Dd,(opru18,Di),Dp
1B 0q bb xb x2 x1              BFEXT.bwpl  Dd,opru18,Dp
1B 0q bb xb x3 x2 x1           BFEXT.bwpl  Dd,(opr24,xysp),Dp
1B 0q bb xb x3 x2 x1           BFEXT.bwpl  Dd,[opr24,xysp],Dp
1B 0q bb xb x3 x2 x1           BFEXT.bwpl  Dd,(opru24,Di),Dp
1B 0q bb xb x3 x2 x1           BFEXT.bwpl  Dd,opr24,Dp
1B 0q bb xb x3 x2 x1           BFEXT.bwpl  Dd,[opr24],Dp
```

OPR/1/2/3-REG-REG

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	1	SOURCE REGISTER D*s*			0q
0	1	0	1	SIZE (.B, .W, .P, .L)		PARAMETER REG D*p*		bb
OPR POSTBYTE (specifes destination)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 0q bb xb                    BFEXT.bwpl  #oprsxe4i,Ds,Dp ;not appropriate for destination
1B 0q bb xb                    BFEXT.bwpl  Dd,Ds,Dp ;see more efficient REG-REG-REG version
1B 0q bb xb                    BFEXT.bwpl  (opru4,xys),Ds,Dp
1B 0q bb xb                    BFEXT.bwpl  {(+-xy)|(xy+-)|(-s)|(s+)},Ds,Dp
1B 0q bb xb                    BFEXT.bwpl  (Di,xys),Ds,Dp
1B 0q bb xb                    BFEXT.bwpl  [Di,xy],Ds,Dp
1B 0q bb xb x1                 BFEXT.bwpl  (oprs9,xysp),Ds,Dp
1B 0q bb xb x1                 BFEXT.bwpl  [oprs9,xysp],Ds,Dp
1B 0q bb xb x1                 BFEXT.bwpl  opru14,Ds,Dp
1B 0q bb xb x2 x1              BFEXT.bwpl  (opru18,Di),Ds,Dp
1B 0q bb xb x2 x1              BFEXT.bwpl  opru18,Ds,Dp
1B 0q bb xb x3 x2 x1           BFEXT.bwpl  (opr24,xysp),Ds,Dp
1B 0q bb xb x3 x2 x1           BFEXT.bwpl  [opr24,xysp],Ds,Dp
1B 0q bb xb x3 x2 x1           BFEXT.bwpl  (opru24,Di),Ds,Dp
1B 0q bb xb x3 x2 x1           BFEXT.bwpl  opr24,Ds,Dp
1B 0q bb xb x3 x2 x1           BFEXT.bwpl  [opr24],Ds,Dp
```

REG-OPR/1/2/3-IMM

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	1	DESTINATION REGISTER D*d*			0q
0	1	1	0	SIZE (.B, .W, .P, .L)		WIDTH[4:3]		bb
WIDTH[2:0]			OFFSET[4:0]					i1
OPR POSTBYTE (specifies source)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 0q bb i1 xb                BFEXT.bwpl  Dd,#oprsxe4i,#width:offset ;-1, +1, 2, 3...14, 15
1B 0q bb i1 xb                BFEXT.bwpl  Dd,Ds,Dp ;see more efficient REG-REG-REG version
1B 0q bb i1 xb                BFEXT.bwpl  Dd,(opru4,xys),#width:offset
1B 0q bb i1 xb                BFEXT.bwpl  Dd,{(+-xy)|(xy+-)|(-s)|(s+)},#width:offset
1B 0q bb i1 xb                BFEXT.bwpl  Dd,(Di,xys),#width:offset
1B 0q bb i1 xb                BFEXT.bwpl  Dd,[Di,xy],#width:offset
1B 0q bb i1 xb x1             BFEXT.bwpl  Dd,(oprs9,xysp),#width:offset
1B 0q bb i1 xb x1             BFEXT.bwpl  Dd,[oprs9,xysp],#width:offset
1B 0q bb i1 xb x1             BFEXT.bwpl  Dd,opru14,#width:offset
1B 0q bb i1 xb x2 x1          BFEXT.bwpl  Dd,(opru18,Di),#width:offset
1B 0q bb i1 xb x2 x1          BFEXT.bwpl  Dd,opru18,#width:offset
1B 0q bb i1 xb x3 x2 x1       BFEXT.bwpl  Dd,(opr24,xysp),#width:offset
1B 0q bb i1 xb x3 x2 x1       BFEXT.bwpl  Dd,[opr24,xysp],#width:offset
1B 0q bb i1 xb x3 x2 x1       BFEXT.bwpl  Dd,(opru24,Di),#width:offset
1B 0q bb i1 xb x3 x2 x1       BFEXT.bwpl  Dd,opr24,#width:offset
1B 0q bb i1 xb x3 x2 x1       BFEXT.bwpl  Dd,[opr24],#width:offset
```

OPR/1/2/3-REG-IMM

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	1	SOURCE REGISTER			0q
0	1	1	1	SIZE (.B, .W, .P, .L)		WIDTH[4:3]		bb
WIDTH[2:0]			OFFSET[4:0]					I1
OPR POSTBYTE (specifies destination)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 0q bb i1 xb                BFEXT.bwpl  #oprsxe4i,Ds,#width:offset ;don't use for dest
1B 0q bb i1 xb                BFEXT.bwpl  Dd,Ds,#width:offset ;REG-REG-IMM more efficient
1B 0q bb i1 xb                BFEXT.bwpl  (opru4,xys),Ds,#width:offset
1B 0q bb i1 xb                BFEXT.bwpl  {(+-xy)|(xy+-)|(-s)|(s+)},Ds,#width:offset
1B 0q bb i1 xb                BFEXT.bwpl  (Di,xys),Ds,#width:offset
1B 0q bb i1 xb                BFEXT.bwpl  [Di,xy],Ds,#width:offset
1B 0q bb i1 xb x1             BFEXT.bwpl  (oprs9,xysp),Ds,#width:offset
1B 0q bb i1 xb x1             BFEXT.bwpl  [oprs9,xysp],Ds,#width:offset
1B 0q bb i1 xb x1             BFEXT.bwpl  opru14,Ds,#width:offset
1B 0q bb i1 xb x2 x1          BFEXT.bwpl  (opru18,Di),Ds,#width:offset
1B 0q bb i1 xb x2 x1          BFEXT.bwpl  opru18,Ds,#width:offset
1B 0q bb i1 xb x3 x2 x1       BFEXT.bwpl  (opr24,xysp),Ds,#width:offset
1B 0q bb i1 xb x3 x2 x1       BFEXT.bwpl  [opr24,xysp],Ds,#width:offset
1B 0q bb i1 xb x3 x2 x1       BFEXT.bwpl  (opru24,Di),Ds,#width:offset
```

```
1B 0q bb i1 xb x3 x2 x1          BFEXT.bwpl  opr24,Ds,#width:offset
1B 0q bb i1 xb x3 x2 x1          BFEXT.bwpl  [opr24],Ds,#width:offset
```

Instruction Fields

DESTINATION REGISTER D*d* - This field specifies the number of the data register D*d* used for the destination (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SOURCE REGISTER D*s* - This field specifies the number of the data register D*s* used for the source operand (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

PARAMETER REG D*p* - This field specifies the number of the 16-bit data register which contains both width and offset parameters for the operation (0b00 = D2, 0b01 = D3, 0b10 = D4, and 0b11 = D5). The width parameter is 5 bits wide and is taken from bits [9:5] of the parameter register; the values 1..31 represent width-values 1..31. The value zero represents a width of 32. The offset parameter is 5 bits wide and is taken from bits [4:0] of the parameter register; it represents a value range of 0..31.

WIDTH - This field specifies the width of the bit-field to be extracted from the source operand. This field is 5 bits wide. The values 1..31 represent width-values 1..31. The value zero represents a width of 32.

OFFSET - This field specifies the offset of the low-order bit of the bit-field to be extracted from the source operand. This field is 5 bits wide. The values 0..31 directly represent the offset values 0..31.

SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01), 24-bit pointer (0b10) or 32-bit long-word (0b11) as the size of the operation.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

BFINS — Bit Field Insert — BFINS

Syntax Variations	Destination-Source-Parameter
`BFINS Dd,Ds,Dp`	**REG-REG-REG**
`BFINS Dd,Ds,#width:offset`	**REG-REG-IMM**
`BFINS.bwplDd,oprmemreg,Dp`	**REG-OPR/1/2/3-REG**
`BFINS.bwploprmemreg,Ds,Dp`	**OPR/1/2/3-REG-REG**
`BFINS.bwplDd,oprmemreg,#width:offset`	**REG-OPR/1/2/3-IMM**
`BFINS.bwploprmemreg,Ds,#width:offset`	**OPR/1/2/3-REG-IMM**

Description

Inserts a bit field of specified width from the low-order bits of a specified source (register D*s* or memory location), into the destination (register D*d* or memory location), beginning at the specified offset. The bit field width and offset are specified in the parameter (register D*p* or immediate operand). The field width determines the number of bits in the field (0b00000 is treated as 32). The field offset specifies the right-most starting bit where the field will be inserted.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	–

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: 0; Cleared.

Detailed Instruction Formats

REG-REG-REG

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	1	DESTINATION REGISTER D*d*			0q
1	0	0	SOURCE REGISTER D*s*			PARAMETER REG D*p*		bb

`1B 0q bb` `BFINS` *`Dd,Ds,Dp`*

REG-REG-IMM

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	1	DESTINATION REGISTER D*d*			0q
1	0	1	SOURCE REGISTER D*s*			WIDTH[4:3]		bb
WIDTH[2:0]			OFFSET[4:0]					i1

`1B 0q bb i1` `BFINS` *`Dd,Ds,#width:offset`*

REG-OPR/1/2/3-REG

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	1	DESTINATION REGISTER D*d*			0q
1	1	0	0	SIZE (.B, .W, .P, .L)		PARAMETER REG D*p*		bb
OPR POSTBYTE (specifies source)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 0q bb xb                    BFINS.bwpl  Dd,#oprsxe4i,Dp ;-1, +1, 2, 3...14, 15
1B 0q bb xb                    BFINS.bwpl  Dd,Ds,Dp ;see more efficient REG-REG-REG version
1B 0q bb xb                    BFINS.bwpl  Dd,(opru4,xys),Dp
1B 0q bb xb                    BFINS.bwpl  Dd,{(+-xy)|(xy+-)|(-s)|(s+)},Dp
1B 0q bb xb                    BFINS.bwpl  Dd,(Di,xys),Dp
1B 0q bb xb                    BFINS.bwpl  Dd,[Di,xy],Dp
1B 0q bb xb x1                 BFINS.bwpl  Dd,(oprs9,xysp),Dp
1B 0q bb xb x1                 BFINS.bwpl  Dd,[oprs9,xysp],Dp
1B 0q bb xb x1                 BFINS.bwpl  Dd,opru14,Dp
1B 0q bb xb x2 x1              BFINS.bwpl  Dd,(opru18,Di),Dp
1B 0q bb xb x2 x1              BFINS.bwpl  Dd,opru18,Dp
1B 0q bb xb x3 x2 x1           BFINS.bwpl  Dd,(opr24,xysp),Dp
1B 0q bb xb x3 x2 x1           BFINS.bwpl  Dd,[opr24,xysp],Dp
1B 0q bb xb x3 x2 x1           BFINS.bwpl  Dd,(opru24,Di),Dp
1B 0q bb xb x3 x2 x1           BFINS.bwpl  Dd,opr24,Dp
1B 0q bb xb x3 x2 x1           BFINS.bwpl  Dd,[opr24],Dp
```

OPR/1/2/3-REG-REG

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	1	SOURCE REGISTER D*s*			0q
1	1	0	1	SIZE (.B, .W, .P, .L)		PARAMETER REG D*p*		bb
OPR POSTBYTE (specifes destination)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 0q bb xb                    BFINS.bwpl  #oprsxe4i,Ds,Dp ;not appropriate for destination
1B 0q bb xb                    BFINS.bwpl  Dd,Ds,Dp ;see more efficient REG-REG-REG version
1B 0q bb xb                    BFINS.bwpl  (opru4,xys),Ds,Dp
1B 0q bb xb                    BFINS.bwpl  {(+-xy)|(xy+-)|(-s)|(s+)},Ds,Dp
1B 0q bb xb                    BFINS.bwpl  (Di,xys),Ds,Dp
1B 0q bb xb                    BFINS.bwpl  [Di,xy],Ds,Dp
1B 0q bb xb x1                 BFINS.bwpl  (oprs9,xysp),Ds,Dp
1B 0q bb xb x1                 BFINS.bwpl  [oprs9,xysp],Ds,Dp
1B 0q bb xb x1                 BFINS.bwpl  opru14,Ds,Dp
1B 0q bb xb x2 x1              BFINS.bwpl  (opru18,Di),Ds,Dp
1B 0q bb xb x2 x1              BFINS.bwpl  opru18,Ds,Dp
1B 0q bb xb x3 x2 x1           BFINS.bwpl  (opr24,xysp),Ds,Dp
1B 0q bb xb x3 x2 x1           BFINS.bwpl  [opr24,xysp],Ds,Dp
1B 0q bb xb x3 x2 x1           BFINS.bwpl  (opru24,Di),Ds,Dp
1B 0q bb xb x3 x2 x1           BFINS.bwpl  opr24,Ds,Dp
1B 0q bb xb x3 x2 x1           BFINS.bwpl  [opr24],Ds,Dp
```

REG-OPR/1/2/3-IMM

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	1	DESTINATION REGISTER D*d*			0q
1	1	1	0	SIZE (.B, .W, .P, .L)		WIDTH[4:3]		bb
WIDTH[2:0]			OFFSET[4:0]					i1
OPR POSTBYTE (specifies source)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 0q bb i1 xb                BFINS.bwpl  Dd,#oprsxe4i,#width:offset ;-1, +1, 2, 3...14, 15
1B 0q bb i1 xb                BFINS.bwpl  Dd,Ds,Dp ;see more efficient REG-REG-REG version
1B 0q bb i1 xb                BFINS.bwpl  Dd,(opru4,xys),#width:offset
1B 0q bb i1 xb                BFINS.bwpl  Dd,{(+-xy)|(xy+-)|(-s)|(s+)},#width:offset
1B 0q bb i1 xb                BFINS.bwpl  Dd,(Di,xys),#width:offset
1B 0q bb i1 xb                BFINS.bwpl  Dd,[Di,xy],#width:offset
1B 0q bb i1 xb x1             BFINS.bwpl  Dd,(oprs9,xysp),#width:offset
1B 0q bb i1 xb x1             BFINS.bwpl  Dd,[oprs9,xysp],#width:offset
1B 0q bb i1 xb x1             BFINS.bwpl  Dd,opru14,#width:offset
1B 0q bb i1 xb x2 x1          BFINS.bwpl  Dd,(opru18,Di),#width:offset
1B 0q bb i1 xb x2 x1          BFINS.bwpl  Dd,opru18,#width:offset
1B 0q bb i1 xb x3 x2 x1       BFINS.bwpl  Dd,(opr24,xysp),#width:offset
1B 0q bb i1 xb x3 x2 x1       BFINS.bwpl  Dd,[opr24,xysp],#width:offset
1B 0q bb i1 xb x3 x2 x1       BFINS.bwpl  Dd,(opru24,Di),#width:offset
1B 0q bb i1 xb x3 x2 x1       BFINS.bwpl  Dd,opr24,#width:offset
1B 0q bb i1 xb x3 x2 x1       BFINS.bwpl  Dd,[opr24],#width:offset
```

OPR/1/2/3-REG-IMM

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	1	SOURCE REGISTER D*s*			0q
1	1	1	1	SIZE (.B, .W, .P, .L)		WIDTH[4:3]		bb
WIDTH[2:0]			OFFSET[4:0]					I1
OPR POSTBYTE (specifies destination)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 0q bb i1 xb                BFINS.bwpl  #oprsxe4i,Ds,#width:offset ;don't use for dest
1B 0q bb i1 xb                BFINS.bwpl  Dd,Ds,#width:offset ;REG-REG-IMM more efficient
1B 0q bb i1 xb                BFINS.bwpl  (opru4,xys),Ds,#width:offset
1B 0q bb i1 xb                BFINS.bwpl  {(+-xy)|(xy+-)|(-s)|(s+)},Ds,#width:offset
1B 0q bb i1 xb                BFINS.bwpl  (Di,xys),Ds,#width:offset
1B 0q bb i1 xb                BFINS.bwpl  [Di,xy],Ds,#width:offset
1B 0q bb i1 xb x1             BFINS.bwpl  (oprs9,xysp),Ds,#width:offset
1B 0q bb i1 xb x1             BFINS.bwpl  [oprs9,xysp],Ds,#width:offset
1B 0q bb i1 xb x1             BFINS.bwpl  opru14,Ds,#width:offset
1B 0q bb i1 xb x2 x1          BFINS.bwpl  (opru18,Di),Ds,#width:offset
1B 0q bb i1 xb x2 x1          BFINS.bwpl  opru18,Ds,#width:offset
1B 0q bb i1 xb x3 x2 x1       BFINS.bwpl  (opr24,xysp),Ds,#width:offset
1B 0q bb i1 xb x3 x2 x1       BFINS.bwpl  [opr24,xysp],Ds,#width:offset
1B 0q bb i1 xb x3 x2 x1       BFINS.bwpl  (opru24,Di),Ds,#width:offset
```

```
1B 0q bb i1 xb x3 x2 x1          BFINS.bwpl  opr24,Ds,#width:offset
1B 0q bb i1 xb x3 x2 x1          BFINS.bwpl  [opr24],Ds,#width:offset
```

Instruction Fields

DESTINATION REGISTER D*d*- This field specifies the number of the data register D*d* used for the destination (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SOURCE REGISTER D*s*- This field specifies the number of the data register D*s* used for the source operand (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

PARAMETER REG D*p* - This field specifies the number of the 16 bit data register which contains both width and offset parameters for the operation (0b00 = D2, 0b01 = D3, 0b10 = D4, and 0b11 = D5). The width parameter is 5 bits wide and is taken from bits [9:5] of the parameter register; the values 1..31 represent width-values 1..31. The value zero represents a width of 32. The offset parameter is 5 bits wide and is taken from bits [4:0] of the parameter register; it represents a value range of 0..31.

WIDTH - This field specifies the width of the bit-field to be extracted from the source operand. This field is 5 bits wide. The values 1..31 represent width-values 1..31. The value zero represents a width of 32.

OFFSET - This field specifies the offset of the low-order bit of the bit-field to be extracted from the source operand. This field is 5 bits wide. The values 0..31 directly represent the offset values 0..31.

SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01), 24-bit pointer (0b10) or 32-bit long-word (0b11) as the size of the operation.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

BGE

Branch if Greater Than or Equal

(Signed Branch)

BGE

Operation

If N ^ V = 0, then (PC) + REL ⇒ PC

For signed two's complement values
if (Accumulator) ≥ (Memory), then branch

Syntax Variations	Addressing Modes
`BGE` *oprdest*	**REL**

Description

BGE can be used to branch after subtracting or comparing signed two's complement values. After CMP, SBC, or SUB, the branch occurs if the CPU register value is greater than or equal to the value in memory (or a second register if the OPR addressing mode is used to specify a data register).

See Section 3.6, "Relative Addressing Modes (REL, REL1)" for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	1	1	0	0	2C
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
2C rb                    BGE        oprdest ;Dest is within +63/-64 (7-bit offset)
2C rb r1                 BGE        oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

BGND Enter Background Debug Mode BGND

Operation

Enter Active Background Mode

Syntax Variations	Addressing Modes
BGND	INH

Description

If the background debug mode is enabled by the ENBDM control bit=1 in the Background Debug Controller (BDC), stop processing application instructions and enter the active background debug mode to await serial BDM commands. If the background debug mode is not enabled, this instruction behaves like a NOP and the application program continues to execute.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

INH

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	00

00 BGND

BGT

Branch if Greater Than

(Signed Branch)

BGT

Operation

If Z | (N ^ V) = 0, then (PC) + REL ⇒ PC

For signed two's complement values
if (Accumulator) > (Memory), then branch

Syntax Variations	Addressing Modes
`BGT` *oprdest*	**REL**

Description

BGT can be used to branch after subtracting or comparing signed two's complement values. After CMP, SBC, or SUB, the branch occurs if the CPU register value is greater than the value in memory (or a second register if the OPR addressing mode is used to specify a data register).

See Section 3.6, "Relative Addressing Modes (REL, REL1)" for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	1	1	1	0	2E
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
2E rb                    BGT      oprdest ;Dest is within +63/-64 (7-bit offset)
2E rb r1                 BGT      oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

BHI

Branch if Higher

(Unsigned Branch)

BHI

Operation

If C | Z = 0, then (PC) + REL ⇒ PC

For unsigned values
if (Accumulator) > (Memory), then branch

Syntax Variations	Addressing Modes
`BHI` *oprdest*	**REL**

Description

BHI can be used to branch after subtracting or comparing unsigned values. After CMP, SBC, or SUB, the branch occurs if the CPU register value is greater than the value in memory (or a second register if the OPR addressing mode is used to specify a data register). BHI should not be used for branching after instructions that do not affect the C bit in the CCR, such as INC, DEC, LD, or ST.

See Section 3.6, "Relative Addressing Modes (REL, REL1)" for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	0	0	1	0	22
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
22 rb                    BHI        oprdest ;Dest is within +63/-64 (7-bit offset)
22 rb r1                 BHI        oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

BHS

Branch if Higher or Same

(Unsigned Branch; Same as BCC)

BHS

Operation

If C = 0, then (PC) + REL ⇒ PC

For unsigned values
if (Accumulator) ≥ (Memory), then branch

Syntax Variations	Addressing Modes
`BHS` *oprdest*	**REL**

Description

BHS can be used to branch after subtracting or comparing unsigned values. After CMP, SBC, or SUB, the branch occurs if the CPU register value is greater than or equal to the value in memory (or a second register if the OPR addressing mode is used to specify a data register). BHS should not be used for branching after instructions that do not affect the C bit in the CCR, such as INC, DEC, LD, or ST.

See Section 3.6, "Relative Addressing Modes (REL, REL1)" for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	0	1	0	0	24
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
24 rb                    BHS        oprdest ;Dest is within +63/-64 (7-bit offset)
24 rb r1                 BHS        oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

BIT — Bit Test — BIT

Operation

(D*i*) & (M)

Syntax Variations	Addressing Modes
`BIT Di,#oprimmsz`	**IMM1/2/4**
`BIT Di,oprmemreg`	**OPR/1/2/3**

Description

Bitwise AND register D*i* with a memory operand to set condition code bits but do not change the contents of the register or memory operand. When the operand is an immediate value, it has the same size as register D*i*. In the case of the general OPR addressing operand, *oprmemreg* can be a sign-extended immediate value (–1, 1, 2, 3..14, 15), a data register, a memory operand the same size as D*i* at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	–

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Cleared.

Detailed Instruction Formats

IMM1/2/4

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	1	0	1	1	SD REGISTER D*i*			5p
IMMEDIATE DATA								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								

```
1B 5p i1                    BIT     Di,#opr8i ;for Di = 8-bit D0 or D1
1B 5p i2 i1                 BIT     Di,#opr16i ;for Di = 16-bit D2, D3, D4, or D5
1B 5p i4 i3 i2 i1           BIT     Di,#opr32i ;for Di = 32-bit D6 or D7
```

OPR/1/2/3

7	6	5	4	3	2	1	0	
0	1	1	0	1	SD REGISTER D*i*			6q
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 6q xb                    BIT     Di,#oprsxe4i ;-1, +1, 2, 3...14, 15
1B 6q xb                    BIT     Di,Dj
1B 6q xb                    BIT     Di,(opru4,xys)
1B 6q xb                    BIT     Di,{(+-xy)|(xy+-)|(-s)|(s+)}
1B 6q xb                    BIT     Di,(Dj,xys)
1B 6q xb                    BIT     Di,[Dj,xy]
1B 6q xb x1                 BIT     Di,(oprs9,xysp)
1B 6q xb x1                 BIT     Di,[oprs9,xysp]
1B 6q xb x1                 BIT     Di,opru14
1B 6q xb x2 x1              BIT     Di,(opru18,Dj)
1B 6q xb x2 x1              BIT     Di,opru18
1B 6q xb x3 x2 x1           BIT     Di,(opr24,xysp)
1B 6q xb x3 x2 x1           BIT     Di,[opr24,xysp]
1B 6q xb x3 x2 x1           BIT     Di,(opru24,Dj)
1B 6q xb x3 x2 x1           BIT     Di,opr24
1B 6q xb x3 x2 x1           BIT     Di,[opr24]
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as a source operand and for the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size of the register D*i*.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

BLE — Branch if Less Than or Equal (Signed Branch) — BLE

Operation

If Z | (N ^ V) = 1, then (PC) + REL ⇒ PC

For signed two's complement values
if (Accumulator) ≤ (Memory), then branch

Syntax Variations	Addressing Modes
BLE *oprdest*	REL

Description

BLE can be used to branch after subtracting or comparing signed two's complement values. After CMP, SBC, or SUB, the branch occurs if the CPU register value is less than or equal to the value in memory (or a second register if the OPR addressing mode is used to specify a data register).

See Section 3.6, "Relative Addressing Modes (REL, REL1)" for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	1	1	1	1	2F
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
2F rb                    BLE      oprdest ;Dest is within +63/-64 (7-bit offset)
2F rb r1                 BLE      oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

BLO

Branch if Lower

(Unsigned Branch; same as BCS)

BLO

Operation

If C = 1, then (PC) + REL ⇒ PC

For unsigned values
if (Accumulator) < (Memory), then branch

Syntax Variations	Addressing Modes
`BLO` *oprdest*	**REL**

Description

If BLO is executed immediately after execution of a CMP, SBC, or SUB instruction, a branch occurs if and only if the unsigned binary number in the CPU register is less than the unsigned number in memory (or a second register if the OPR addressing mode is used to specify a data register). BLO should not be used for branching after instructions that do not affect the C bit in the CCR, such as INC, DEC, LD, or ST.

See Section 3.6, "Relative Addressing Modes (REL, REL1)" for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	0	1	0	1	25
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
25 rb                    BLO        oprdest ;Dest is within +63/-64 (7-bit offset)
25 rb r1                 BLO        oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

BLS

Branch if Lower or Same

(Unsigned Branch)

BLS

Operation

If Z | C = 1, then (PC) + REL ⇒ PC

For unsigned values
if (Accumulator) ≤ (Memory), then branch

Syntax Variations	Addressing Modes
`BLS` *oprdest*	**REL**

Description

If BLS is executed immediately after execution of a CMP, SBC, or SUB instruction, a branch occurs if and only if the unsigned binary number in the CPU register is less than or equal to the unsigned number in memory (or a second register if the OPR addressing mode is used to specify a data register). BLS should not be used for branching after instructions that do not affect the C bit in the CCR, such as INC, DEC, LD, or ST.

See Section 3.6, "Relative Addressing Modes (REL, REL1)" for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	0	0	1	1	23
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
23 rb                    BLS        oprdest ;Dest is within +63/-64 (7-bit offset)
23 rb r1                 BLS        oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

BLT — Branch if Less Than (Signed Branch) — BLT

Operation

If N ^ V = 1, then (PC) + REL ⇒ PC

For signed two's complement values
if (Accumulator) < (Memory), then branch

Syntax Variations	Addressing Modes
BLT *oprdest*	REL

Description

BLTE can be used to branch after subtracting or comparing signed two's complement values. After CMP, SBC, or SUB, the branch occurs if the CPU register value is less than the value in memory (or a second register if the OPR addressing mode is used to specify a data register).

See Section 3.6, "Relative Addressing Modes (REL, REL1)" for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	1	1	0	1	2D
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
2D rb                    BLT      oprdest ;Dest is within +63/-64 (7-bit offset)
2D rb r1                 BLT      oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

BMI — Branch if Minus — BMI

Operation

If N = 1, then (PC) + REL ⇒ PC

Simple branch

Syntax Variations	Addressing Modes
`BMI` *oprdest*	**REL**

Description

Tests the N status bit. If N = 1 then program execution continues at location (PC) + REL

See Section 3.6, “Relative Addressing Modes (REL, REL1)” for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	1	0	1	1	2B
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
2B rb                    BMI        oprdest ;Dest is within +63/-64 (7-bit offset)
2B rb r1                 BMI        oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

BNE — Branch if Not Equal — BNE

Operation

If Z = 0, then (PC) + REL ⇒ PC

Simple branch

Syntax Variations	Addressing Modes
`BNE` *oprdest*	**REL**

Description

Tests the Z status bit. If Z = 0 then program execution continues at location (PC) + REL

See Section 3.6, “Relative Addressing Modes (REL, REL1)” for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	0	1	1	0	26
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
26 rb                    BNE        oprdest ;Dest is within +63/-64 (7-bit offset)
26 rb r1                 BNE        oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

BPL — Branch if Plus — BPL

Operation

If N = 0, then (PC) + REL ⇒ PC

Simple branch

Syntax Variations	Addressing Modes
`BPL` *oprdest*	**REL**

Description

Tests the N status bit. If N = 0 then program execution continues at location (PC) + REL

See Section 3.6, “Relative Addressing Modes (REL, REL1)” for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	1	0	1	0	2A
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
2A rb                    BPL        oprdest ;Dest is within +63/-64 (7-bit offset)
2A rb r1                 BPL        oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

BRA — Branch Always — BRA

Operation

(PC) + REL ⇒ PC

Simple unconditional branch

Syntax Variations	Addressing Modes
`BRA` *oprdest*	**REL**

Description

Unconditional branch to an address formed by adding the address of the current PC (the address of the opcode for the current branch instruction) plus the 7-bit or 15-bit two's complement displacement that is included in the second or second and third bytes of the branch instruction. A displacement of zero will result in an infinite loop back to the beginning of the current branch instruction.

Since the BRA condition is always satisfied, the branch is always taken, and the instruction queue must always be refilled.

See Section 3.6, "Relative Addressing Modes (REL, REL1)" for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	20
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
20 rb                    BRA      oprdest ;Dest is within +63/-64 (7-bit offset)
20 rb r1                 BRA      oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

BRCLR Test Bit and Branch if Clear BRCLR

Operation

Copy bitn to C; Then if (D*i*) & bitn = 0, (PC) + REL ⇒ PC

Copy bitn to C; Then if (M) & bitn = 0, (PC) + REL ⇒ PC

Syntax Variations	Addressing Modes
`BRCLR Di,#opr5i,oprdest`	**REG-IMM-REL**
`BRCLR Di,Dn,oprdest`	**REG-REG-REL**
`BRCLR.bwloprmemreg,#opr5i,oprdest`	**OPR/1/2/3-IMM-REL**
`BRCLR.bwloprmemreg,Dn,oprdest`	**OPR/1/2/3-REG-REL**

Description

Tests the specified bit in D*i* or a memory operand, and branches if the bit was clear. The bit to be tested is specified in a 5-bit immediate value or in the low order five bits of a data register D*n*. In the case of the general OPR addressing operand, *oprmemreg* can be a short immediate value (–1, 1, 2, 3...14, 15), a data register, an 8-, 16-, or 32-bit memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	Δ

C: Set if the bit being tested was set before the operation. Cleared otherwise.

Detailed Instruction Formats

REG-IMM-REL

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	0	02
n[4:0]					SD REGISTER D*i*			bm
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
02 bm rb          BRCLR    Di,#opr5i,oprdest ;Dest is within +63/-64 (7-bit)
02 bm rb r1       BRCLR    Di,#opr5i,oprdest ;Dest within ~ +/-16K (15-bit)
```

REG-REG-REL

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	0	02
1	PARAMETER REGISTER D*n*			0	0	0	1	bm
1	0	1	1	1	SD REGISTER D*i*			xb
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
02 bm xb rb       BRCLR    Di,Dn,oprdest ;Dest is within +63/-64 (7-bit)
```

```
02 bm xb rb r1                    BRCLR      Di,Dn,oprdest ;Dest within ~ +/-16K (15-bit)
```

OPR/1/2/3-IMM-REL

Byte-sized operand (.B)

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	0	02
1	n[2:0]			0	0	0	0	bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

Word-sized operand (.W)

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	0	02
1	n[2:0]			0	0	1	n[3]	bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

Long-word sized operand (.L)

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	0	02
1	n[2:0]			1	0	n[4:3]		bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
02 bm xb rb                    BRCLR.bwl   #oprsxe4i,#opr5i,oprdest ;(7-bit)
02 bm xb rb r1                 BRCLR.bwl   #oprsxe4i,#opr5i,oprdest ;(15-bit)
02 bm xb rb                    BRCLR.bwl   Di,#opr5i,oprdest ;see efficient REG-IMM version
02 bm xb rb r1                 BRCLR.bwl   Di,#opr5i,oprdest ;see efficient REG-IMM version
02 bm xb rb                    BRCLR.bwl   (opru4,xys),#opr5i,oprdest ;(7-bit)
02 bm xb rb r1                 BRCLR.bwl   (opru4,xys),#opr5i,oprdest ;(15-bit)
02 bm xb rb                    BRCLR.bwl   {(+-xy)|(xy+-)|(-s)|(s+)},#opr5i,oprdest ;(7-bit)
02 bm xb rb r1                 BRCLR.bwl   {(+-xy)|(xy+-)|(-s)|(s+)},#opr5i,oprdest ;(15-bit)
02 bm xb rb                    BRCLR.bwl   (Di,xys),#opr5i,oprdest ;(7-bit)
02 bm xb rb r1                 BRCLR.bwl   (Di,xys),#opr5i,oprdest ;(15-bit)
02 bm xb rb                    BRCLR.bwl   [Di,xy],#opr5i,oprdest ;(7-bit)
02 bm xb rb r1                 BRCLR.bwl   [Di,xy],#opr5i,oprdest ;(15-bit)
02 bm xb x1 rb                 BRCLR.bwl   (oprs9,xysp),#opr5i,oprdest ;(7-bit)
02 bm xb x1 rb r1              BRCLR.bwl   (oprs9,xysp),#opr5i,oprdest ;(15-bit)
02 bm xb x1 rb                 BRCLR.bwl   [oprs9,xysp],#opr5i,oprdest ;(7-bit)
02 bm xb x1 rb r1              BRCLR.bwl   [oprs9,xysp],#opr5i,oprdest ;(15-bit)
02 bm xb x1 rb                 BRCLR.bwl   opru14,#opr5i,oprdest ;(7-bit)
02 bm xb x1 rb r1              BRCLR.bwl   opru14,#opr5i,oprdest ;(15-bit)
02 bm xb x2 x1 rb              BRCLR.bwl   (opru18,Di),#opr5i,oprdest ;(7-bit)
02 bm xb x2 x1 rb r1           BRCLR.bwl   (opru18,Di),#opr5i,oprdest ;(15-bit)
02 bm xb x2 x1 rb              BRCLR.bwl   opru18,#opr5i,oprdest ;(7-bit)
02 bm xb x2 x1 rb r1           BRCLR.bwl   opru18,#opr5i,oprdest ;(15-bit)
02 bm xb x3 x2 x1 rb           BRCLR.bwl   (opr24,xysp),#opr5i,oprdest ;(7-bit)
02 bm xb x3 x2 x1 rb r1        BRCLR.bwl   (opr24,xysp),#opr5i,oprdest ;(15-bit)
02 bm xb x3 x2 x1 rb           BRCLR.bwl   [opr24,xysp],#opr5i,oprdest ;(7-bit)
02 bm xb x3 x2 x1 rb r1        BRCLR.bwl   [opr24,xysp],#opr5i,oprdest ;(15-bit)
02 bm xb x3 x2 x1 rb           BRCLR.bwl   (opru24,Di),#opr5i,oprdest ;(7-bit)
02 bm xb x3 x2 x1 rb r1        BRCLR.bwl   (opru24,Di),#opr5i,oprdest ;(15-bit)
02 bm xb x3 x2 x1 rb           BRCLR.bwl   opr24,#opr5i,oprdest ;(7-bit)
02 bm xb x3 x2 x1 rb r1        BRCLR.bwl   opr24,#opr5i,oprdest ;(15-bit)
02 bm xb x3 x2 x1 rb           BRCLR.bwl   [opr24],#opr5i,oprdest ;(7-bit)
02 bm xb x3 x2 x1 rb r1        BRCLR.bwl   [opr24],#opr5i,oprdest ;(15-bit)
```

OPR/1/2/3-REG-REL

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	0	02
1	PARAMETER REGISTER D*n*			SIZE (.B-0:0, .W-0:1, .L-1:1)		0	1	bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
02 bm xb rb                    BRCLR.bwl  #oprsxe4i,Dn,oprdest ;(7-bit)
02 bm xb rb r1                 BRCLR.bwl  #oprsxe4i,Dn,oprdest ;(15-bit)
02 bm xb rb                    BRCLR.bwl  Di,Dn,oprdest ;see more efficient REG-REG version
02 bm xb rb r1                 BRCLR.bwl  Di,Dn,oprdest ;see more efficient REG-REG version
02 bm xb rb                    BRCLR.bwl  (opru4,xys),Dn,oprdest ;(7-bit)
02 bm xb rb r1                 BRCLR.bwl  (opru4,xys),Dn,oprdest ;(15-bit)
02 bm xb rb                    BRCLR.bwl  {(+-xy)|(xy+-)|(-s)|(s+)},Dn,oprdest ;(7-bit)
02 bm xb rb r1                 BRCLR.bwl  {(+-xy)|(xy+-)|(-s)|(s+)},Dn,oprdest ;(15-bit)
02 bm xb rb                    BRCLR.bwl  (Di,xys),Dn,oprdest ;(7-bit)
02 bm xb rb r1                 BRCLR.bwl  (Di,xys),Dn,oprdest ;(15-bit)
02 bm xb rb                    BRCLR.bwl  [Di,xy],Dn,oprdest ;(7-bit)
02 bm xb rb r1                 BRCLR.bwl  [Di,xy],Dn,oprdest ;(15-bit)
02 bm xb x1 rb                 BRCLR.bwl  (oprs9,xysp),Dn,oprdest ;(7-bit)
02 bm xb x1 rb r1              BRCLR.bwl  (oprs9,xysp),Dn,oprdest ;(15-bit)
02 bm xb x1 rb                 BRCLR.bwl  [oprs9,xysp],Dn,oprdest ;(7-bit)
02 bm xb x1 rb r1              BRCLR.bwl  [oprs9,xysp],Dn,oprdest ;(15-bit)
02 bm xb x1 rb                 BRCLR.bwl  opru14,Dn,oprdest ;(7-bit)
02 bm xb x1 rb r1              BRCLR.bwl  opru14,Dn,oprdest ;(15-bit)
02 bm xb x2 x1 rb              BRCLR.bwl  (opru18,Di),Dn,oprdest ;(7-bit)
02 bm xb x2 x1 rb r1           BRCLR.bwl  (opru18,Di),Dn,oprdest ;(15-bit)
02 bm xb x2 x1 rb              BRCLR.bwl  opru18,Dn,oprdest ;(7-bit)
02 bm xb x2 x1 rb r1           BRCLR.bwl  opru18,Dn,oprdest ;(15-bit)
02 bm xb x3 x2 x1 rb           BRCLR.bwl  (opr24,xysp),Dn,oprdest ;(7-bit)
02 bm xb x3 x2 x1 rb r1        BRCLR.bwl  (opr24,xysp),Dn,oprdest ;(15-bit)
02 bm xb x3 x2 x1 rb           BRCLR.bwl  [opr24,xysp],Dn,oprdest ;(7-bit)
02 bm xb x3 x2 x1 rb r1        BRCLR.bwl  [opr24,xysp],Dn,oprdest ;(15-bit)
02 bm xb x3 x2 x1 rb           BRCLR.bwl  (opru24,Di),Dn,oprdest ;(7-bit)
02 bm xb x3 x2 x1 rb r1        BRCLR.bwl  (opru24,Di),Dn,oprdest ;(15-bit)
02 bm xb x3 x2 x1 rb           BRCLR.bwl  opr24,Dn,oprdest ;(7-bit)
02 bm xb x3 x2 x1 rb r1        BRCLR.bwl  opr24,Dn,oprdest ;(15-bit)
02 bm xb x3 x2 x1 rb           BRCLR.bwl  [opr24],Dn,oprdest ;(7-bit)
02 bm xb x3 x2 x1 rb r1        BRCLR.bwl  [opr24],Dn,oprdest ;(15-bit)
```

Instruction Fields

REGISTER - This field specifies the number of the data register D*i* which is used as the source operand (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

PARAMETER REGISTER D*n* - This field specifies the number of the data register D*n* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) which is used to specify the bit number of the bit in the operand that is to be tested. Only the low-order 5 bits of the parameter register are used.

n[4:0] - This field contains the 5-bit immediate parameter that specifies the bit number of the bit in the operand that is to be tested.

SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01), 24-bit pointer (0b10) or 32-bit long-word (0b11) as the size of the operation.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Using OPR addressing mode to specify a register operand, performs the same function as the REG-IMM or REG-REG versions but is less efficient.

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.
DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

BRSET — Test Bit and Branch if Set — BRSET

Operation

Copy bitn to C; Then if (Di) & (bitn) ≠ 0, (PC) + REL ⇒ PC

Copy bitn to C; Then if (M) & (bitn) ≠ 0, (PC) + REL ⇒ PC

Syntax Variations	Addressing Modes
`BRSET Di,#opr5i,oprdest`	**REG-IMM-REL**
`BRSET Di,Dn,oprdest`	**REG-REG-REL**
`BRSET.bwloprmemreg,#opr5i,oprdest`	**OPR/1/2/3-IMM-REL**
`BRSET.bwloprmemreg,Dn,oprdest`	**OPR/1/2/3-REG-REL**

Description

Tests the specified bit in D*i* or a memory operand, and branches if the bit was set. The bit to be tested is specified in a 5-bit immediate value or in the low order five bits of a data register D*n*. In the case of the general OPR addressing operand, *oprmemreg* can be a short immediate value (–1, 1, 2, 3...14, 15), a data register, an 8-, 16-, or 32-bit memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	Δ

C: Set if the bit being tested was set before the operation. Cleared otherwise.

Detailed Instruction Formats

REG-IMM-REL

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	1	03
n[4:0]					SD REGISTER D*i*			bm
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
03 bm rb          BRSET    Di,#opr5i,oprdest ;Dest is within +63/-64 (7-bit)
03 bm rb r1       BRSET    Di,#opr5i,oprdest ;Dest within ~ +/-16K (15-bit)
```

REG-REG-REL

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	1	03
1	PARAMETER REGISTER D*n*			0	0	0	1	bm
1	0	1	1	1	SD REGISTER D*i*			xb
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
03 bm xb rb       BRSET    Di,Dn,oprdest ;Dest is within +63/-64 (7-bit)
```

```
03 bm xb rb r1                    BRSET      Di,Dn,oprdest ;Dest within ~ +/-16K (15-bit)
```

OPR/1/2/3-IMM-REL

Byte-sized operand (.B)

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	1	03
1	n[2:0]			0	0	0	0	bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

Word-sized operand (.W)

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	1	03
1	n[2:0]			0	0	1	n[3]	bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

Long-word sized operand (.L)

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	1	03
1	n[2:0]			1	0	n[4:3]		bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
03 bm xb rb                    BRSET.bwl   #oprsxe4i,#opr5i,oprdest ;(7-bit)
03 bm xb rb r1                 BRSET.bwl   #oprsxe4i,#opr5i,oprdest ;(15-bit)
03 bm xb rb                    BRSET.bwl   Di,#opr5i,oprdest ;see efficient REG-IMM version
03 bm xb rb r1                 BRSET.bwl   Di,#opr5i,oprdest ;see efficient REG-IMM version
03 bm xb rb                    BRSET.bwl   (opru4,xys),#opr5i,oprdest ;(7-bit)
03 bm xb rb r1                 BRSET.bwl   (opru4,xys),#opr5i,oprdest ;(15-bit)
03 bm xb rb                    BRSET.bwl   {(+-xy)|(xy+-)|(-s)|(s+)},#opr5i,oprdest ;(7-bit)
03 bm xb rb r1                 BRSET.bwl   {(+-xy)|(xy+-)|(-s)|(s+)},#opr5i,oprdest ;(15-bit)
03 bm xb rb                    BRSET.bwl   (Di,xys),#opr5i,oprdest ;(7-bit)
03 bm xb rb r1                 BRSET.bwl   (Di,xys),#opr5i,oprdest ;(15-bit)
03 bm xb rb                    BRSET.bwl   [Di,xy],#opr5i,oprdest ;(7-bit)
03 bm xb rb r1                 BRSET.bwl   [Di,xy],#opr5i,oprdest ;(15-bit)
03 bm xb x1 rb                 BRSET.bwl   (oprs9,xysp),#opr5i,oprdest ;(7-bit)
03 bm xb x1 rb r1              BRSET.bwl   (oprs9,xysp),#opr5i,oprdest ;(15-bit)
03 bm xb x1 rb                 BRSET.bwl   [oprs9,xysp],#opr5i,oprdest ;(7-bit)
03 bm xb x1 rb r1              BRSET.bwl   [oprs9,xysp],#opr5i,oprdest ;(15-bit)
03 bm xb x1 rb                 BRSET.bwl   opru14,#opr5i,oprdest ;(7-bit)
03 bm xb x1 rb r1              BRSET.bwl   opru14,#opr5i,oprdest ;(15-bit)
03 bm xb x2 x1 rb              BRSET.bwl   (opru18,Di),#opr5i,oprdest ;(7-bit)
03 bm xb x2 x1 rb r1           BRSET.bwl   (opru18,Di),#opr5i,oprdest ;(15-bit)
03 bm xb x2 x1 rb              BRSET.bwl   opru18,#opr5i,oprdest ;(7-bit)
03 bm xb x2 x1 rb r1           BRSET.bwl   opru18,#opr5i,oprdest ;(15-bit)
03 bm xb x3 x2 x1 rb           BRSET.bwl   (opr24,xysp),#opr5i,oprdest ;(7-bit)
03 bm xb x3 x2 x1 rb r1        BRSET.bwl   (opr24,xysp),#opr5i,oprdest ;(15-bit)
03 bm xb x3 x2 x1 rb           BRSET.bwl   [opr24,xysp],#opr5i,oprdest ;(7-bit)
03 bm xb x3 x2 x1 rb r1        BRSET.bwl   [opr24,xysp],#opr5i,oprdest ;(15-bit)
03 bm xb x3 x2 x1 rb           BRSET.bwl   (opru24,Di),#opr5i,oprdest ;(7-bit)
03 bm xb x3 x2 x1 rb r1        BRSET.bwl   (opru24,Di),#opr5i,oprdest ;(15-bit)
03 bm xb x3 x2 x1 rb           BRSET.bwl   opr24,#opr5i,oprdest ;(7-bit)
03 bm xb x3 x2 x1 rb r1        BRSET.bwl   opr24,#opr5i,oprdest ;(15-bit)
03 bm xb x3 x2 x1 rb           BRSET.bwl   [opr24],#opr5i,oprdest ;(7-bit)
03 bm xb x3 x2 x1 rb r1        BRSET.bwl   [opr24],#opr5i,oprdest ;(15-bit)
```

OPR/1/2/3-REG-REL

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	1	03
1	PARAMETER REGISTER D*n*			SIZE (.B-0:0, .W-0:1, .L-1:1)		0	1	bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
03 bm xb rb                    BRSET.bwl  #oprsxe4i,Dn,oprdest ;(7-bit)
03 bm xb rb r1                 BRSET.bwl  #oprsxe4i,Dn,oprdest ;(15-bit)
03 bm xb rb                    BRSET.bwl  Di,Dn,oprdest ;see more efficient REG-REG version
03 bm xb rb r1                 BRSET.bwl  Di,Dn,oprdest ;see more efficient REG-REG version
03 bm xb rb                    BRSET.bwl  (opru4,xys),Dn,oprdest ;(7-bit)
03 bm xb rb r1                 BRSET.bwl  (opru4,xys),Dn,oprdest ;(15-bit)
03 bm xb rb                    BRSET.bwl  {(+-xy)|(xy+-)|(-s)|(s+)},Dn,oprdest ;(7-bit)
03 bm xb rb r1                 BRSET.bwl  {(+-xy)|(xy+-)|(-s)|(s+)},Dn,oprdest ;(15-bit)
03 bm xb rb                    BRSET.bwl  (Di,xys),Dn,oprdest ;(7-bit)
03 bm xb rb r1                 BRSET.bwl  (Di,xys),Dn,oprdest ;(15-bit)
03 bm xb rb                    BRSET.bwl  [Di,xy],Dn,oprdest ;(7-bit)
03 bm xb rb r1                 BRSET.bwl  [Di,xy],Dn,oprdest ;(15-bit)
03 bm xb x1 rb                 BRSET.bwl  (oprs9,xysp),Dn,oprdest ;(7-bit)
03 bm xb x1 rb r1              BRSET.bwl  (oprs9,xysp),Dn,oprdest ;(15-bit)
03 bm xb x1 rb                 BRSET.bwl  [oprs9,xysp],Dn,oprdest ;(7-bit)
03 bm xb x1 rb r1              BRSET.bwl  [oprs9,xysp],Dn,oprdest ;(15-bit)
03 bm xb x1 rb                 BRSET.bwl  opru14,Dn,oprdest ;(7-bit)
03 bm xb x1 rb r1              BRSET.bwl  opru14,Dn,oprdest ;(15-bit)
03 bm xb x2 x1 rb              BRSET.bwl  (opru18,Di),Dn,oprdest ;(7-bit)
03 bm xb x2 x1 rb r1           BRSET.bwl  (opru18,Di),Dn,oprdest ;(15-bit)
03 bm xb x2 x1 rb              BRSET.bwl  opru18,Dn,oprdest ;(7-bit)
03 bm xb x2 x1 rb r1           BRSET.bwl  opru18,Dn,oprdest ;(15-bit)
03 bm xb x3 x2 x1 rb           BRSET.bwl  (opr24,xysp),Dn,oprdest ;(7-bit)
03 bm xb x3 x2 x1 rb r1        BRSET.bwl  (opr24,xysp),Dn,oprdest ;(15-bit)
03 bm xb x3 x2 x1 rb           BRSET.bwl  [opr24,xysp],Dn,oprdest ;(7-bit)
03 bm xb x3 x2 x1 rb r1        BRSET.bwl  [opr24,xysp],Dn,oprdest ;(15-bit)
03 bm xb x3 x2 x1 rb           BRSET.bwl  (opru24,Di),Dn,oprdest ;(7-bit)
03 bm xb x3 x2 x1 rb r1        BRSET.bwl  (opru24,Di),Dn,oprdest ;(15-bit)
03 bm xb x3 x2 x1 rb           BRSET.bwl  opr24,Dn,oprdest ;(7-bit)
03 bm xb x3 x2 x1 rb r1        BRSET.bwl  opr24,Dn,oprdest ;(15-bit)
03 bm xb x3 x2 x1 rb           BRSET.bwl  [opr24],Dn,oprdest ;(7-bit)
03 bm xb x3 x2 x1 rb r1        BRSET.bwl  [opr24],Dn,oprdest ;(15-bit)
```

Instruction Fields

REGISTER - This field specifies the number of the data register D*i* which is used as the source operand (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

PARAMETER REGISTER D*n* - This field specifies the number of the data register D*n* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) which is used to specify the bit number of the bit in the operand that is to be tested. Only the low-order 5 bits of the parameter register are used.

n[4:0] - This field contains the 5-bit immediate parameter that specifies the bit number of the bit in the operand that is to be tested.

SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01), or 32-bit long-word (0b11) as the size of the operation.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Using OPR addressing mode to specify a register operand, performs the same function as the REG-IMM or REG-REG versions but is less efficient.

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.
DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

BSET — Test and Set Bit — BSET

Operation

bitn of D*i* ⇒ C; then (D*i*) | (bitn) ⇒ D*i*

bitn of M ⇒ C; then (M) | (bitn) ⇒ M

Syntax Variations	Addressing Modes
`BSET Di,#opr5i`	**REG-IMM**
`BSET Di,Dn`	**REG-REG**
`BSET.bwl oprmemreg,#opr5i`	**OPR/1/2/3-IMM**
`BSET.bwl oprmemreg,Dn`	**OPR/1/2/3-REG**

Description

Tests and copies the original state of the specified bit into the C condition code bit to be used for semaphores. Then sets the specified bit in D*i* or a memory operand by performing a bitwise OR with a mask that has all bits clear except the specified bit. The bit to be set is specified in a 5-bit immediate value or in the low order five bits of a data register D*n*. In the case of the general OPR addressing operand, *oprmemreg* can be a data register, an 8-, 16-, or 32-bit memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. *It is not appropriate to specify a short-immediate operand with the OPR addressing mode because it is not possible to modify (set a bit in) the immediate operand.*

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	Δ

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Cleared.

C: Set if the bit being set was set before the operation. Cleared otherwise.

Detailed Instruction Formats

REG-IMM

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	1	ED
n[4:0]					SD REGISTER D*i*			bm

`ED bm` `BSET` `Di,#opr5i`

REG-REG

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	1	ED
1	PARAMETER REGISTER D*n*			0	0	0	1	bm
1	0	1	1	1	SD REGISTER D*i*			xb

```
ED bm xb                    BSET        Di,Dn
```

OPR/1/2/3-IMM

Byte-sized operand (.B)

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	1	ED
1	n[2:0]			0	0	0	0	bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

Word-sized operand (.W)

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	1	ED
1	n[2:0]			0	0	1	n[3]	bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

Long-word sized operand (.L)

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	1	ED
1	n[2:0]			1	0	n[4:3]		bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
ED sb xb                    BSET.bwl    #oprsxe4i,#opr5i ;not appropriate for destination
ED sb xb                    BSET.bwl    Di,#opr5i ;see more efficient REG-IMM1 version
ED sb xb                    BSET.bwl    (opru4,xys),#opr5i
ED sb xb                    BSET.bwl    {(+-xy)|(xy+-)|(-s)|(s+)},#opr5i
ED sb xb                    BSET.bwl    (Di,xys),#opr5i
ED sb xb                    BSET.bwl    [Di,xy],#opr5i
ED sb xb x1                 BSET.bwl    (oprs9,xysp),#opr5i
ED sb xb x1                 BSET.bwl    [oprs9,xysp],#opr5i
ED sb xb x1                 BSET.bwl    opru14,#opr5i
ED sb xb x2 x1              BSET.bwl    (opru18,Di),#opr5i
ED sb xb x2 x1              BSET.bwl    opru18,#opr5i
ED sb xb x3 x2 x1           BSET.bwl    (opr24,xysp),#opr5i
ED sb xb x3 x2 x1           BSET.bwl    [opr24,xysp],#opr5i
ED sb xb x3 x2 x1           BSET.bwl    (opru24,Di),#opr5i
ED sb xb x3 x2 x1           BSET.bwl    opr24,#opr5i
ED sb xb x3 x2 x1           BSET.bwl    [opr24],#opr5i
```

OPR/1/2/3-REG

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	0	1	ED
1	PARAMETER REGISTER D*n*			SIZE (.B-0:0, .W-0:1, .L-1:1)		0	1	bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
ED bm xb                    BSET.bwl   #oprsxe4i,Dn ;not appropriate for destination
ED bm xb                    BSET.bwl   Di,Dn
ED bm xb                    BSET.bwl   (opru4,xys),Dn
ED bm xb                    BSET.bwl   {(+-xy)|(xy+-)|(-s)|(s+)},Dn
ED bm xb                    BSET.bwl   (Di,xys),Dn
ED bm xb                    BSET.bwl   [Di,xy],Dn
ED bm xb x1                 BSET.bwl   (oprs9,xysp),Dn
ED bm xb x1                 BSET.bwl   [oprs9,xysp],Dn
ED bm xb x1                 BSET.bwl   opru14,Dn
ED bm xb x2 x1              BSET.bwl   (opru18,Di),Dn
ED bm xb x2 x1              BSET.bwl   opru18,Dn
ED bm xb x3 x2 x1           BSET.bwl   (opr24,xysp),Dn
ED bm xb x3 x2 x1           BSET.bwl   [opr24,xysp],Dn
ED bm xb x3 x2 x1           BSET.bwl   (opru24,Di),Dn
ED bm xb x3 x2 x1           BSET.bwl   opr24,Dn
ED bm xb x3 x2 x1           BSET.bwl   [opr24],Dn
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as a source operand and the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

PARAMETER REGISTER D*n* - This field specifies the number of the data register D*n* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) which is used to specify the bit number of the bit in the operand that is to be set. Only the low-order 5 bits of the parameter register are used.

n[4:0] - This field contains the 5-bit immediate parameter that specifies the bit number of the bit in the operand that is to be set.

SIZE - This field specifies 8-bit byte (0:0), 16-bit word (0:1), or 32-bit long-word (1:1) as the size of the operation.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Short immediate mode is not appropriate for instructions that store a result to the specified operand.

BSR — Branch to Subroutine — BSR

Operation

(SP) – 3 ⇒ SP
RTN[23:0] ⇒ $M_{(SP)}$: $M_{(SP + 1)}$: $M_{(SP + 2)}$
(PC) + REL ⇒ PC

Syntax Variations	Addressing Modes
`BSR` *`oprdest`*	**REL**

Description

Sets up conditions to return to normal program flow, then transfers control to a subroutine. Uses the address of the instruction after the BSR as a return address.

Decrements the SP by three, to allow the three bytes of the return address to be stacked.

Stacks the return address (the SP points to the most-significant byte of the return address).

Branches to the location (PC) + REL.

Subroutines are normally terminated with an RTS instruction, which restores the return address from the stack.

See Section 3.6, “Relative Addressing Modes (REL, REL1)” for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	1	21
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
21 rb                    BSR        oprdest ;Dest is within +63/-64 (7-bit offset)
21 rb r1                 BSR        oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

BTGL

Test and Toggle Bit

(invert bit)

BTGL

Operation

bitn of D*i* ⇒ C; then (D*i*) ^ bitn ⇒ D*i*

bitn of M ⇒ C; then (M) ^ bitn ⇒ M

Syntax Variations	Addressing Modes
`BTGL Di,#opr5i`	**REG-IMM**
`BTGL Di,Dn`	**REG-REG**
`BTGL.bwploprmemreg,#opr5i`	**OPR/1/2/3-IMM**
`BTGL.bwploprmemreg,Dn`	**OPR/1/2/3-REG**

Description

Tests and copies the original state of the specified bit into the C condition code bit to be used for semaphores. Then toggles (inverts) the specified bit in D*i* or a memory operand by performing a bitwise Exclusive-OR with a mask that has all bits cleared except the specified bit. The bit to be toggled is specified in a 5-bit immediate value or in the low order five bits of a data register D*n*. In the case of the general OPR addressing operand, *oprmemreg* can be a data register, an 8-, 16-, 24-, or 32-bit memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. *It is not appropriate to specify a short-immediate operand with the OPR addressing mode because it is not possible to modify (toggle a bit in) the immediate operand.*

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	Δ

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Cleared.

C: Set if the bit being cleared was set before the operation. Cleared otherwise.

Detailed Instruction Formats

REG-IMM

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	1	0	EE
n[4:0]					SD REGISTER D*i*			bm

EE bm BTGL *Di,#opr5i*

REG-REG

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	1	0	EE
1	PARAMETER REGISTER D*n*			0	0	0	1	bm
1	0	1	1	1	SD REGISTER D*i*			xb

```
EE bm xb                    BTGL        Di,Dn
```

OPR/1/2/3-IMM

Byte-sized operand (.B)

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	1	0	EE
1	n[2:0]			0	0	0	0	bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

Word-sized operand (.W)

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	1	0	EE
1	n[2:0]			0	0	1	n[3]	bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

Long-word sized operand (.L)

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	1	0	EE
1	n[2:0]			1	0	n[4:3]		bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
EE sb xb                    BTGL.bwl    #oprsxe4i,#opr5i ;not appropriate for destination
EE sb xb                    BTGL.bwl    Di,#opr5i ;see more efficient REG-IMM1 version
EE sb xb                    BTGL.bwl    (opru4,xys),#opr5i
EE sb xb                    BTGL.bwl    {(+-xy)|(xy+-)|(-s)|(s+)},#opr5i
EE sb xb                    BTGL.bwl    (Di,xys),#opr5i
EE sb xb                    BTGL.bwl    [Di,xy],#opr5i
EE sb xb x1                 BTGL.bwl    (oprs9,xysp),#opr5i
EE sb xb x1                 BTGL.bwl    [oprs9,xysp],#opr5i
EE sb xb x1                 BTGL.bwl    opru14,#opr5i
EE sb xb x2 x1              BTGL.bwl    (opru18,Di),#opr5i
EE sb xb x2 x1              BTGL.bwl    opru18,#opr5i
EE sb xb x3 x2 x1           BTGL.bwl    (opr24,xysp),#opr5i
EE sb xb x3 x2 x1           BTGL.bwl    [opr24,xysp],#opr5i
EE sb xb x3 x2 x1           BTGL.bwl    (opru24,Di),#opr5i
EE sb xb x3 x2 x1           BTGL.bwl    opr24,#opr5i
EE sb xb x3 x2 x1           BTGL.bwl    [opr24],#opr5i
```

OPR/1/2/3-REG

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	1	0	EE
1	PARAMETER REGISTER D*n*			SIZE (.B-0:0, .W-0:1, .L-1:1)		0	1	bm
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
EE bm xb                    BTGL.bwl   #oprsxe4i,Dn ;not appropriate for destination
EE bm xb                    BTGL.bwl   Di,Dn
EE bm xb                    BTGL.bwl   (opru4,xys),Dn
EE bm xb                    BTGL.bwl   {(+-xy)|(xy+-)|(-s)|(s+)},Dn
EE bm xb                    BTGL.bwl   (Di,xys),Dn
EE bm xb                    BTGL.bwl   [Di,xy],Dn
EE bm xb x1                 BTGL.bwl   (oprs9,xysp),Dn
EE bm xb x1                 BTGL.bwl   [oprs9,xysp],Dn
EE bm xb x1                 BTGL.bwl   opru14,Dn
EE bm xb x2 x1              BTGL.bwl   (opru18,Di),Dn
EE bm xb x2 x1              BTGL.bwl   opru18,Dn
EE bm xb x3 x2 x1           BTGL.bwl   (opr24,xysp),Dn
EE bm xb x3 x2 x1           BTGL.bwl   [opr24,xysp],Dn
EE bm xb x3 x2 x1           BTGL.bwl   (opru24,Di),Dn
EE bm xb x3 x2 x1           BTGL.bwl   opr24,Dn
EE bm xb x3 x2 x1           BTGL.bwl   [opr24],Dn
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as a source operand and the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

PARAMETER REGISTER D*n* - This field specifies the number of the data register D*n* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) which is used to specify the bit number of the bit in the operand that is to be toggled. Only the low-order 5 bits of the parameter register are used.

n[4:0] - This field contains the 5-bit immediate parameter that specifies the bit number of the bit in the operand that is to be toggled.

SIZE - This field specifies 8-bit byte (0:0), 16-bit word (0:1), or 32-bit long-word (1:1) as the size of the operation.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Short immediate mode is not appropriate for instructions that store a result to the specified operand.

BVC — Branch if Overflow Clear — BVC

Operation

If V = 0, then (PC) + REL ⇒ PC

Simple branch

Syntax Variations	Addressing Modes
`BVC` *oprdest*	**REL**

Description

Tests the V status bit. If V = 0 then program execution continues at location (PC) + REL

See Section 3.6, “Relative Addressing Modes (REL, REL1)” for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	1	0	0	0	28
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
28 rb                    BVC        oprdest ;Dest is within +63/-64 (7-bit offset)
28 rb r1                 BVC        oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

BVS — Branch if Overflow Set — BVS

Operation

If V = 1, then (PC) + REL ⇒ PC

Simple branch

Syntax Variations	Addressing Modes
`BVS` *oprdest*	**REL**

Description

Tests the V status bit. If V = 1 then program execution continues at location (PC) + REL

See Section 3.6, “Relative Addressing Modes (REL, REL1)” for details of branch execution.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REL

7	6	5	4	3	2	1	0	
0	0	1	0	1	0	0	1	29
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
29 rb                    BVS      oprdest ;Dest is within +63/-64 (7-bit offset)
29 rb r1                 BVS      oprdest ;Dest is within ~ +/-16K (15-bit offset)
```

Instruction Fields

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit.

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z | (N ^ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ^ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z | (N ^ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ^ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C | Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C | Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	—	—	—	—

CLB — Count Leading Sign-Bits — CLB

Syntax Variations	Addressing Modes
`CLB` *cpureg,cpureg*	**REG-REG**

Description

Counts the number of leading sign-bits in the source register, decrements this number and then copies the result into the destination register.

The result can be directly used as shift-width operand to normalize a fractional number in the source register by shifting its content to the left.

Only the data-registers D0..D7 can be used as arguments for this instruction.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	0	Δ	0	–

N: 0, cleared.

Z: Set if the result is zero. Cleared otherwise.

V: 0, cleared.

Detailed Instruction Formats

INH

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
1	0	0	1	0	0	0	1	91
0	SOURCE REGISTER Di			0	DESTINATION REGISTER Di			cb

`1B 91 cb` `CLB` *cpureg,cpureg*

Instruction Fields

SOURCE REGISTER D*i* - This field specifies the number of the data register D*i* which is used as the source operand (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

DESTINATION REGISTER D*i* - This field specifies the number of the data register D*i* which is used as the result register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

CLC

Clear Carry

(Translates to ANDCC #$FE)

CLC

Operation

$0 \Rightarrow$ C bit

Syntax Variations	Addressing Modes
CLC	**IMM1**

Description

Clears the C status bit. This instruction is assembled as ANDCC #$FE. The ANDCC instruction can be used to clear any combination of bits in the CCL in one operation.

CLC can be used to set up the C bit prior to a shift or rotate instruction involving the C bit.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	0

C: Cleared.

Detailed Instruction Formats

IMM1

7	6	5	4	3	2	1	0	
1	1	0	0	1	1	1	0	CE
1	1	1	1	1	1	1	0	FE

CE FE CLC

CLI

Clear Interrupt Mask

(Translates to ANDCC #$EF)

CLI

Operation

$0 \Rightarrow$ I bit

Syntax Variations	Addressing Modes
CLI	**IMM1**

Description

Clears the I mask bit. This instruction is assembled as ANDCC #$EF. The ANDCC instruction can be used to clear any combination of bits in the CCL in one operation.

When the I bit is cleared, interrupts are enabled.

There is a 1-cycle (bus clock) delay in the clearing mechanism for the I bit so that, if interrupts were previously disabled, the next instruction after a CLI will always be executed, even if there was an interrupt pending prior to execution of the CLI instruction.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C	
–	–	–	–	–	–	–	–	–	0	–	–	–	–	supervisor state
–	–	–	–	–	–	–	–	–	–	–	–	–	–	user state

Detailed Instruction Formats

IMM1

7	6	5	4	3	2	1	0	
1	1	0	0	1	1	1	0	CE
1	1	1	0	1	1	1	1	EF

```
CE EF                    CLI
```

CLR — Clear Memory, Register, or Index Register — CLR

Operation

0 ⇒ M; or 0 ⇒ D*i*; or 0 ⇒ X; or 0 ⇒ Y

Syntax Variations	Addressing Modes
`CLR.`*`bwpl oprmemreg`	**OPR/1/2/3**
`CLR  D`*`i`	**INH**
`CLR  X`	**INH**
`CLR  Y`	**INH**

Description

Clears a memory operand M, a CPU register D*i*, or index registers X or Y. In the case of the general OPR addressing operand, *oprmemreg* can be a data register, a memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. The size of the memory operand M is determined by the suffix (b=8 bit byte, w=16 bit word, p=24 bit pointer, or l=32 bit long-word). If the OPR memory addressing mode is used to specify a data register D*i*, the register determines the size for the operation and the *.bwpl* suffix is ignored. *It is inappropriate to specify a short immediate operand using the OPR addressing mode for this instruction because it is not possible to clear the immediate operand.*

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	0	1	0	0

N: 0, cleared.

Z: 1, set.

V: 0, cleared.

C: 0, cleared.

Detailed Instruction Formats

INH

7	6	5	4	3	2	1	0	
0	0	1	1	1	SD REGISTER D*i*			3q

`3q  CLR  D`*`i`*

INH

7	6	5	4	3	2	1	0	
1	0	0	1	1	0	1	Y/X	9p

`9A  CLR  X`
`9B  CLR  Y`

OPR/1/2/3

7	6	5	4	3	2	1	0	
1	0	1	1	1	1	SIZE (.B, .W, .P, .L)		Bp
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
Bp xb                    CLR.bwpl   #oprsxe4i ;not appropriate for destination
Bp xb                    CLR.bwpl   Di ;INH version is more efficient
Bp xb                    CLR.bwpl   (opru4,xys)
Bp xb                    CLR.bwpl   {(+-xy)|(xy+-)|(-s)|(s+)}
Bp xb                    CLR.bwpl   (Di,xys)
Bp xb                    CLR.bwpl   [Di,xy]
Bp xb x1                 CLR.bwpl   (oprs9,xysp)
Bp xb x1                 CLR.bwpl   [oprs9,xysp]
Bp xb x1                 CLR.bwpl   opru14
Bp xb x2 x1              CLR.bwpl   (opru18,Di)
Bp xb x2 x1              CLR.bwpl   opru18
Bp xb x3 x2 x1           CLR.bwpl   (opr24,xysp)
Bp xb x3 x2 x1           CLR.bwpl   [opr24,xysp]
Bp xb x3 x2 x1           CLR.bwpl   (opru24,Di)
Bp xb x3 x2 x1           CLR.bwpl   opr24
Bp xb x3 x2 x1           CLR.bwpl   [opr24]
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as the first source operand (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

Y/X - This field selects either Y (1) or X (0) to be cleared.

SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01), 24-bit pointer (0b10) or 32-bit long-word (0b11) as the size of the operation.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

CLV

Clear Overflow

(Translates to ANDCC #$FD)

CLV

Operation

0 ⇒ V bit

Syntax Variations	Addressing Modes
CLV	IMM1

Description

Clears the V status bit. This instruction is assembled as ANDCC #$FD. The ANDCC instruction can be used to clear any combination of bits in the CCL in one operation.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	0	–

V: Cleared.

Detailed Instruction Formats

IMM1

7	6	5	4	3	2	1	0	
1	1	0	0	1	1	1	0	CE
1	1	1	1	1	1	0	1	FD

CE FD CLV

CMP Compare CMP

Operation

(D*i*) – (M); (X) – (M); (Y) – (M); (S) – (M); or (X) – (Y)

Syntax Variations	Addressing Modes
`CMP Di,#oprimmsz`	**IMM1/2/4**
`CMP Di,oprmemreg`	**OPR/1/2/3**
`CMP xy,#opr24i`	**IMM3**
`CMP xy,oprmemreg`	**OPR/1/2/3**
`CMP S,#opr24i`	**IMM3**
`CMP S,oprmemreg`	**OPR/1/2/3**
`CMP X,Y`	**INH**

Description

Compare register D*i*, X, Y, or S to an immediate value or to a memory operand, or compare X to Y and set the condition codes, which may then be used for arithmetic and logical conditional branching. The operation is equivalent to a subtract but the result is not stored and the contents of the CPU register and the memory operand are not changed. When the operand is an immediate value, it has the same size as the CPU register. In the case of the general OPR addressing operand, *oprmemreg* can be a sign-extended immediate value (–1, 1, 2, 3..14, 15), a data register, a memory operand the same size as the CPU register at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	-	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Set if a two's complement overflow resulted from the operation. Cleared otherwise.

C: Set if there is a borrow from the MSB of the result. Cleared otherwise.

Detailed Instruction Formats

IMM1/2/4 (for CMP D*i*)

7	6	5	4	3	2	1	0	
1	1	1	0	0	SD REGISTER D*i*			Ep
IMMEDIATE DATA								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								

```
Ep i1                    CMP     Di,#opr8i ;for Di = 8-bit D0 or D1
Ep i2 i1                 CMP     Di,#opr16i ;for Di = 16-bit D2, D3, D4, or D5
Ep i4 i3 i2 i1           CMP     Di,#opr32i ;for Di = 32-bit D6 or D7
```

OPR/1/2/3 (for CMP D*i*)

7	6	5	4	3	2	1	0	
1	1	1	1	0	SD REGISTER D*i*			Fn
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
Fn xb                    CMP     Di,#oprsxe4i ;-1, +1, 2, 3...14, 15
Fn xb                    CMP     Di,Dj
Fn xb                    CMP     Di,(opru4,xys)
Fn xb                    CMP     Di,{(+-xy)|(xy+-)|(-s)|(s+)}
Fn xb                    CMP     Di,(Dj,xys)
Fn xb                    CMP     Di,[Dj,xy]
Fn xb x1                 CMP     Di,(oprs9,xysp)
Fn xb x1                 CMP     Di,[oprs9,xysp]
Fn xb x1                 CMP     Di,opru14
Fn xb x2 x1              CMP     Di,(opru18,Dj)
Fn xb x2 x1              CMP     Di,opru18
Fn xb x3 x2 x1           CMP     Di,(opr24,xysp)
Fn xb x3 x2 x1           CMP     Di,[opr24,xysp]
Fn xb x3 x2 x1           CMP     Di,(opru24,Dj)
Fn xb x3 x2 x1           CMP     Di,opr24
Fn xb x3 x2 x1           CMP     Di,[opr24]
```

IMM3 (for CMP X and Y)

7	6	5	4	3	2	1	0	
1	1	1	0	1	0	0	Y/X	Ep
IMMEDIATE DATA[23:16]								i3
IMMEDIATE DATA[15:8]								i2
IMMEDIATE DATA[7:0]								i1

```
Ep i3 i2 i1              CMP     xy,#opr24i
```

OPR/1/2/3 (for CMP X and Y)

7	6	5	4	3	2	1	0	
1	1	1	1	1	0	0	Y/X	Fp
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
Fp xb                    CMP     xy,#oprsxe4i ;-1, +1, 2, 3...14, 15
Fp xb                    CMP     xy,Dj
Fp xb                    CMP     xy,(opru4,xys)
Fp xb                    CMP     xy,{(+-xy)|(xy+-)|(-s)|(s+)}
Fp xb                    CMP     xy,(Dj,xys)
Fp xb                    CMP     xy,[Dj,xy]
Fp xb x1                 CMP     xy,(oprs9,xysp)
Fp xb x1                 CMP     xy,[oprs9,xysp]
Fp xb x1                 CMP     xy,opru14
Fp xb x2 x1              CMP     xy,(opru18,Dj)
Fp xb x2 x1              CMP     xy,opru18
Fp xb x3 x2 x1           CMP     xy,(opr24,xysp)
Fp xb x3 x2 x1           CMP     xy,[opr24,xysp]
Fp xb x3 x2 x1           CMP     xy,(opru24,Dj)
Fp xb x3 x2 x1           CMP     xy,opr24
Fp xb x3 x2 x1           CMP     xy,[opr24]
```

IMM3 (for CMP S)

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	0	1	0	0	04
IMMEDIATE DATA[23:16]								i3
IMMEDIATE DATA[15:8]								i2
IMMEDIATE DATA[7:0]								11

```
1B 04 i3 i2 i1           CMP     S,#opr24i
```

OPR/1/2/3 (for CMP S)

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	0	0	1	0	02
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 02 xb                 CMP     S,#oprsxe4i ;-1, +1, 2, 3...14, 15
1B 02 xb                 CMP     S,Dj
1B 02 xb                 CMP     S,(opru4,xys)
1B 02 xb                 CMP     S,{(+-xy)|(xy+-)|(-s)|(s+)}
1B 02 xb                 CMP     S,(Dj,xys)
1B 02 xb                 CMP     S,[Dj,xy]
1B 02 xb x1              CMP     S,(oprs9,xysp)
1B 02 xb x1              CMP     S,[oprs9,xysp]
1B 02 xb x1              CMP     S,opru14
```

```
1B 02 xb x2 x1                CMP        S,(opru18,Dj)
1B 02 xb x2 x1                CMP        S,opru18
1B 02 xb x3 x2 x1             CMP        S,(opr24,xysp)
1B 02 xb x3 x2 x1             CMP        S,[opr24,xysp]
1B 02 xb x3 x2 x1             CMP        S,(opru24,Dj)
1B 02 xb x3 x2 x1             CMP        S,opr24
1B 02 xb x3 x2 x1             CMP        S,[opr24]
```

INH (for CMP X,Y)

7	6	5	4	3	2	1	0	
1	1	1	1	1	1	0	0	FC

```
FC                            CMP        X,Y
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as the first source operand (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes 3 bytes or 4 bytes wide, depending on the size of the source register.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

Y/X - This field specified either the X (0) or Y (1) index register as the first source operand.

COM — Complement Memory — COM

Operation

~(M) ⇒ M

Syntax Variations	Addressing Modes
`COM.bwl oprmemreg`	**OPR/1/2/3**

Description

Complements (inverts) a memory operand M. The memory operand *oprmemreg* can be a data register, a memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. The size of the memory operand M is determined by the suffix (b=8 bit byte, w=16 bit word, or l=32 bit long-word). If the OPR memory addressing mode is used to specify a data register D*i*, the register determines the size for the operation and the *.bwl* suffix is ignored. *It is inappropriate to specify a short immediate operand using the OPR addressing mode for this instruction because it is not possible to modify the immediate operand.*

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	–

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: 0, cleared.

Detailed Instruction Formats

OPR/1/2/3

7	6	5	4	3	2	1	0	
1	1	0	0	1	1	SIZE (.B, .W, –, .L)		Cp
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
Cp xb          COM.bwl   #oprsxe4i ;not appropriate for destination
Cp xb          COM.bwl   Di
Cp xb          COM.bwl   (opru4,xys)
Cp xb          COM.bwl   {(+-xy)|(xy+-)|(-s)|(s+)}
Cp xb          COM.bwl   (Di,xys)
Cp xb          COM.bwl   [Di,xy]
Cp xb x1       COM.bwl   (oprs9,xysp)
Cp xb x1       COM.bwl   [oprs9,xysp]
Cp xb x1       COM.bwl   opru14
Cp xb x2 x1    COM.bwl   (opru18,Di)
```

```
Cp xb x2 x1                    COM.bwl    opru18
Cp xb x3 x2 x1                 COM.bwl    (opr24,xysp)
Cp xb x3 x2 x1                 COM.bwl    [opr24,xysp]
Cp xb x3 x2 x1                 COM.bwl    (opru24,Di)
Cp xb x3 x2 x1                 COM.bwl    opr24
Cp xb x3 x2 x1                 COM.bwl    [opr24]
```

Instruction Fields

SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01), or 32-bit long-word (0b11) as the size of the operation.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

DBcc — Decrement and Branch — DBcc

Operation

(D*i*) – 1 ⇒ D*i*; then Branch if (condition) true

(X) – 1 ⇒ X; then Branch if (condition) true

(Y) – 1 ⇒ Y; then Branch if (condition) true

(M) – 1 ⇒ M; then Branch if (condition) true

Condition may be...

NE (Z=0), EQ (Z=1), PL (N=0), MI (N=1), GT (Z | N=0), or LE (Z | N=1)

Syntax Variations	Addressing Modes
`DBcc Di,oprdest`	**REG-REL**
`DBcc X,oprdest`	**REG-REL**
`DBcc Y,oprdest`	**REG-REL**
`DBcc.bwploprmemreg,oprdest`	**OPR/1/2/3-REL**

Description

Decrement the operand (internally determining the N and Z conditions but not modifying the CCR) then branch if the specified condition is true. The condition (cc) can be NE (not equal), EQ (equal), PL (plus), MI (minus), GT (greater than), or LE (less than or equal). The operand may be one of the eight data registers, index register X, index register Y, or an 8-, 16-, 24-, or 32-bit memory operand. In the case of the general OPR addressing operand, *oprmemreg* can be a data register, a memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. *It is not appropriate to specify a short-immediate operand with the OPR addressing mode because it is not possible to modify (decrement) the immediate operand.* The relative offset for the branch can be either 7 bits (–64 to +63) or 15 bits (~+/–16K) displacement from the DBcc opcode location.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REG-REL (D*i*)

7	6	5	4	3	2	1	0	
0	0	0	0	1	1	0	1	0B
1	CC (NE,EQ,PL,MI,GT,LE,–,–)			0	REGISTER D*i*			lb
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
0B lb rb          DBcc    Di,oprdest ;destination within -64..+63 (7-bit)
0B lb rb r1       DBcc    Di,oprdest ;destination within ~+/-16k (15-bit)
```

REG-REL (X, Y)

7	6	5	4	3	2	1	0	
0	0	0	0	1	1	0	1	0B
1	CC (NE,EQ,PL,MI,GT,LE,–,–)			1	0	don't care	Y/X	lb
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
0B lb rb          DBcc    X,oprdest ;destination within -64..+63 (7-bit)
0B lb rb r1       DBcc    X,oprdest ;destination within ~+/-16k (15-bit)
0B lb rb          DBcc    Y,oprdest ;destination within -64..+63 (7-bit)
0B lb rb r1       DBcc    Y,oprdest ;destination within ~+/-16k (15-bit)
```

OPR/1/2/3-REL

7	6	5	4	3	2	1	0	
0	0	0	0	1	1	0	1	0B
1	CC (NE,EQ,PL,MI,GT,LE,–,–)			1	1	SIZE (.B, .W, .P, .L)		lb
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
0B lb xb rb          DBcc.bwpl   #oprsxe4i,oprdest ;not appropriate for destination
0B lb xb rb r1       DBcc.bwpl   #oprsxe4i,oprdest ;not appropriate for destination
0B lb xb rb          DBcc.bwpl   Di,oprdest ;see efficient REG-REL version
0B lb xb rb r1       DBcc.bwpl   Di,oprdest ;see efficient REG-REL version
0B lb xb rb          DBcc.bwpl   (opru4,xys),oprdest ;(7-bit)
0B lb xb rb r1       DBcc.bwpl   (opru4,xys),oprdest ;(15-bit)
0B lb xb rb          DBcc.bwpl   {(+-xy)|(xy+-)|(-s)|(s+)},oprdest ;(7-bit)
0B lb xb rb r1       DBcc.bwpl   {(+-xy)|(xy+-)|(-s)|(s+)},oprdest ;(15-bit)
0B lb xb rb          DBcc.bwpl   (Di,xys),oprdest ;(7-bit)
0B lb xb rb          DBcc.bwpl   (Di,xys),oprdest ;(7-bit)
0B lb xb rb r1       DBcc.bwpl   [Di,xy],oprdest ;(15-bit)
0B lb xb rb r1       DBcc.bwpl   [Di,xy],oprdest ;(15-bit)
0B lb xb x1 rb       DBcc.bwpl   (oprs9,xysp),oprdest ;(7-bit)
0B lb xb x1 rb r1    DBcc.bwpl   (oprs9,xysp),oprdest ;(15-bit)
0B lb xb x1 rb       DBcc.bwpl   [oprs9,xysp],oprdest ;(7-bit)
0B lb xb x1 rb r1    DBcc.bwpl   [oprs9,xysp],oprdest ;(15-bit)
0B lb xb x1 rb       DBcc.bwpl   opru14,oprdest ;(7-bit)
0B lb xb x1 rb r1    DBcc.bwpl   opru14,oprdest ;(15-bit)
0B lb xb x2 x1 rb    DBcc.bwpl   (opru18,Di),oprdest ;(7-bit)
0B lb xb x2 x1 rb r1 DBcc.bwpl   (opru18,Di),oprdest ;(15-bit)
0B lb xb x2 x1 rb    DBcc.bwpl   opru18,oprdest ;(7-bit)
0B lb xb x2 x1 rb r1 DBcc.bwpl   opru18,oprdest ;(15-bit)
0B lb xb x3 x2 x1 rb DBcc.bwpl   (opr24,xysp),oprdest ;(7-bit)
0B lb xb x3 x2 x1 rb r1 DBcc.bwpl (opr24,xysp),oprdest ;(15-bit)
0B lb xb x3 x2 x1 rb DBcc.bwpl   [opr24,xysp],oprdest ;(7-bit)
0B lb xb x3 x2 x1 rb r1 DBcc.bwpl [opr24,xysp],oprdest ;(15-bit)
0B lb xb x3 x2 x1 rb DBcc.bwpl   (opru24,Di),oprdest ;(7-bit)
0B lb xb x3 x2 x1 rb r1 DBcc.bwpl (opru24,Di),oprdest ;(15-bit)
0B lb xb x3 x2 x1 rb DBcc.bwpl   opr24,oprdest ;(7-bit)
0B lb xb x3 x2 x1 rb r1 DBcc.bwpl opr24,oprdest ;(15-bit)
```

```
0B lb xb x3 x2 x1 rb          DBcc.bwpl   [opr24],oprdest ;(7-bit)
0B lb xb x3 x2 x1 rb r1       DBcc.bwpl   [opr24],oprdest ;(15-bit)
```

Instruction Fields

CC - This field specifies the condition for the branch according to the table below:

Test	Mnemonic	Condition	Boolean
NE; r≠0	DBNE	000	Z = 0
EQ; r=0	DBEQ	001	Z = 1
PL; r≥0	DBPL	010	N = 0
MI; r<0	DBMI	011	N = 1
GT; r>0	DBGT	100	Z | N = 0
LE; r≤0	DBLE	101	Z | N = 1
reserved (Decrement and Branch Never)		110	–
		111	

REGISTER - This field specifies the number of the data register D*i* which is used as the source operand (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01), 24-bit pointer (0b10) or 32-bit long-word (0b11) as the size of the operation.

Y/X - This field specifies either index register X (0) or index register Y (1) as the source operand.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. *Using OPR addressing mode to specify a short-immediate operand, is not appropriate because you cannot alter (decrement) the immediate operand value. Using OPR addressing mode to specify a register operand, performs the same function as the REG-REL versions but is less efficient.*

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit .
DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

DEC — Decrement — DEC

Operation

$(D_i) - 1 \Rightarrow D_i$

$(M) - 1 \Rightarrow M$

Syntax Variations	Addressing Modes
`DEC Di`	**INH**
`DEC.bwl oprmemreg`	**OPR/1/2/3**

Description

Decrement a register D*i* or memory operand M. The memory operand *oprmemreg* can be a data register, a memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. The size of the memory operand M is determined by the suffix (b=8 bit byte, w=16 bit word, or l=32 bit long-word). If the OPR memory addressing mode is used to specify a data register D*j*, the register determines the size for the operation and the *.bwl* suffix is ignored. *It is inappropriate to specify a short immediate operand using the OPR addressing mode for this instruction because it is not possible to modify the immediate operand.*

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	–

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Set if there was a two's complement overflow as a result of the operation. Cleared otherwise.

Detailed Instruction Formats

INH

7	6	5	4	3	2	1	0	
0	1	0	0	0	REGISTER D*i*			4n

4n DEC Di

OPR/1/2/3

7	6	5	4	3	2	1	0	
1	0	1	0	1	1	SIZE (.B, .W, –, .L)		Ap
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
Ap xb                    DEC.bwl    #oprsxe4i ;not appropriate for destination
Ap xb                    DEC.bwl    Dj ;INH version is more efficient
Ap xb                    DEC.bwl    (opru4,xys)
Ap xb                    DEC.bwl    {(+-xy)|(xy+-)|(-s)|(s+)}
Ap xb                    DEC.bwl    (Dj,xys)
Ap xb                    DEC.bwl    [Dj,xy]
Ap xb x1                 DEC.bwl    (oprs9,xysp)
Ap xb x1                 DEC.bwl    [oprs9,xysp]
Ap xb x1                 DEC.bwl    opru14
Ap xb x2 x1              DEC.bwl    (opru18,Dj)
Ap xb x2 x1              DEC.bwl    opru18
Ap xb x3 x2 x1           DEC.bwl    (opr24,xysp)
Ap xb x3 x2 x1           DEC.bwl    [opr24,xysp]
Ap xb x3 x2 x1           DEC.bwl    (opru24,Dj)
Ap xb x3 x2 x1           DEC.bwl    opr24
Ap xb x3 x2 x1           DEC.bwl    [opr24]
```

Instruction Fields

SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01), or 32-bit long-word (0b11) as the size of the operation.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

DIVS — Signed Divide — DIVS

Operation

(D*j*) ÷ (D*k*) ⇒ D*d*

(D*j*) ÷ IMM ⇒ D*d*

(D*j*) ÷ (M) ⇒ D*d*

(M1) ÷ (M2) ⇒ D*d*

Syntax Variations	Addressing Modes
`DIVS Dd,Dj,Dk`	**REG-REG**
`DIVS.B Dd,Dj,#opr8i`	**REG-IMM1**
`DIVS.W Dd,Dj,#opr16i`	**REG-IMM2**
`DIVS.L Dd,Dj,#opr32i`	**REG-IMM4**
`DIVS.bwl Dd,Dj,oprmemreg`	**REG-OPR/1/2/3**
`DIVS.bwplbwplDd,oprmemreg,oprmemreg`	**OPR/1/2/3-OPR/1/2/3**

Description

Divides a signed two's complement dividend by a signed two's complement divisor to produce a signed two's complement quotient in a register D*d*. The dividend may be a register D*j* or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M1. The divisor may be a register D*k*, an 8-bit, 16-bit , or 32-bit immediate value, or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M or M2. To ensure compatibility with the C standard, the sign of the quotient is the exclusive-OR of the sign of the dividend and the divisor.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result is set. Undefined after overflow or division by zero. Cleared otherwise.

Z: Set if the result is zero. Undefined after overflow or division by zero. Cleared otherwise.

V: Set if the signed result does not fit in the result register D*d*. Undefined after division by zero. Cleared otherwise.

C: Set if divisor was zero. Cleared otherwise. (Indicates division by zero).

Detailed Instruction Formats

REG-REG

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	1	0	QUOTIENT REGISTER D*d*			3n
1	0	DIVIDEND REGISTER D*j*			DIVISOR REGISTER D*k*			mb

```
1B 3n mb                    DIVS       Dd,Dj,Dk
```

REG-IMM1/2/4

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	1	0	QUOTIENT REGISTER D*d*			3n
1	1	DIVIDEND REGISTER D*j*			1	IMM_SIZE (.B, .W, –, .L)		mb
IMMEDIATE DATA (Divisor)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								

```
1B 3n mb i1                 DIVS.B     Dd,Dj,#opr8i
1B 3n mb i2 i1              DIVS.W     Dd,Dj,#opr16i ;short-imm better for some values
1B 3n mb i4 i3 i2 i1        DIVS.L     Dd,Dj,#opr32i ;short-imm better for some values
```

REG-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	1	0	QUOTIENT REGISTER D*d*			3n
1	1	DIVIDEND REGISTER D*j*				M2_SIZE (.B, .W, –, .L)		mb
OPR POSTBYTE (for M2 divisor)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 3n mb xb                 DIVS.bwl   Dd,Dj,#oprsxe4i
1B 3n mb xb                 DIVS.bwl   Dd,Dj,Dk ;see more efficient REG-REG version
1B 3n mb xb                 DIVS.bwl   Dd,Dj,(opru4,xys)
1B 3n mb xb                 DIVS.bwl   Dd,Dj,{(+-xy)|(xy+-)|(-s)|(s+)}
1B 3n mb xb                 DIVS.bwl   Dd,Dj,(Di,xys)
1B 3n mb xb                 DIVS.bwl   Dd,Dj,[Di,xy]
1B 3n mb xb x1              DIVS.bwl   Dd,Dj,(oprs9,xysp)
1B 3n mb xb x1              DIVS.bwl   Dd,Dj,[oprs9,xysp]
1B 3n mb xb x1              DIVS.bwl   Dd,Dj,opru14
1B 3n mb xb x2 x1           DIVS.bwl   Dd,Dj,(opru18,Dj)
1B 3n mb xb x2 x1           DIVS.bwl   Dd,Dj,opru18
1B 3n mb xb x3 x2 x1        DIVS.bwl   Dd,Dj,(opr24,xysp)
1B 3n mb xb x3 x2 x1        DIVS.bwl   Dd,Dj,[opr24,xysp]
1B 3n mb xb x3 x2 x1        DIVS.bwl   Dd,Dj,(opru24,Dj)
1B 3n mb xb x3 x2 x1        DIVS.bwl   Dd,Dj,opr24
1B 3n mb xb x3 x2 x1        DIVS.bwl   Dd,Dj,[opr24]
```

OPR/1/2/3-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	1	0	QUOTIENT REGISTER D*d*			3n
1	1	M1_SIZE (.B, .W, .P, .L)		M2_SIZE (.B, .W, .P, .L)		1	0	mb
OPR POSTBYTE (for M1 dividend)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
OPR POSTBYTE (for M2 divisor)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

page 2 Opcode postbyte	M1 object code	M2 object code	Instruction Mnemonic		Source Format for M1 (Dividend) (select 1 option in this col)	Source Format for M2 (Divisor) (select 1 option in this col)
1B 3n mb	xb	xb	DIVS.*bwplbwpl*	Dd,	#*oprsxe4i*,	#*oprsxe4i*
	xb	xb			*Dj*,	*Dk*
	xb	xb			(*opru4*,*xys*),	(*opru4*,*xys*)
	xb	xb			(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+),	(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+)
	xb	xb			(*Dj*,*xys*),	(*Dk*,*xys*)
	xb	xb			[*Dj*,*xy*],	[*Dk*,*xy*]
	xb x1	xb x1			(*oprs9*,*xysp*),	(*oprs9*,*xysp*)
	xb x1	xb x1			[*oprs9*,*xysp*],	[*oprs9*,*xysp*]
	xb x1	xb x1			*opru14*,	*opru14*
	xb x2 x1	xb x2 x1			(*opru18*,*Dj*),	(*opru18*,*Dk*)
	xb x2 x1	xb x2 x1			*opru18*,	*opru18*
	xb x3 x2 x1	xb x3 x2 x1			(*opr24*,*xysp*),	(*opr24*,*xysp*)
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*,*xysp*],	[*opr24*,*xysp*]
	xb x3 x2 x1	xb x3 x2 x1			(*opru24*,*Dj*),	(*opru24*,*Dk*)
	xb x3 x2 x1	xb x3 x2 x1			*opr24*,	*opr24*
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*],	[*opr24*]

All combinations are valid although some, such as specifying a data register for both M1 and M2 can be done more efficiently using the REG-REG version of the instruction.

Instruction Fields

QUOTIENT REGISTER- This field specifies the number of the data register D*d* used for the result (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

DIVIDEND REGISTER - This field specifies the number of the data register D*j* used as dividend (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

DIVISOR REGISTER - This field specifies the number of the data register D*k* used as divisor (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMM_SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01) or 32-bit long-word (0b11) as the size of the divisor. The 0b10 combination is not available for the REG-IMM version of the instruction because those codes are used for the OPR-OPR version.

M1_SIZE and M2_SIZE - These fields specify the size of M1 (dividend) and M2 (divisor) which use the general OPR addressing mode to specify short-immediate, register, or memory operands (0b00 = 8-bit byte, 0b01 = 16-bit word, 0b10 = 24-bit pointer, and 0b11 = 32-bit long-word). When a short-immediate operand is specified, it is internally sign-extended to the size specified by the M1_SIZE and/or M2_SIZE specifications. When a register is specified, it determines the size and the M1_SIZE and/or M2_SIZE specifications are ignored.

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size specified by IMM_SIZE.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Using OPR addressing mode to specify a register operand for both the dividend and the divisor, is less efficient than using the REG-REG version of the instruction.

DIVU — Unsigned Divide — DIVU

Operation

(D*j*) ÷ (D*k*) ⇒ D*d*

(D*j*) ÷ IMM ⇒ D*d*

(D*j*) ÷ (M) ⇒ D*d*

(M1) ÷ (M2) ⇒ D*d*

Syntax Variations	Addressing Modes
`DIVU Dd,Dj,Dk`	**REG-REG**
`DIVU.B Dd,Dj,#opr8i`	**REG-IMM1**
`DIVU.W Dd,Dj,#opr16i`	**REG-IMM2**
`DIVU.L Dd,Dj,#opr32i`	**REG-IMM4**
`DIVU.bwl Dd,Dj,oprmemreg`	**REG-OPR/1/2/3**
`DIVU.bwplbwplDd,oprmemreg,oprmemreg`	**OPR/1/2/3-OPR/1/2/3**

Description

Divides an unsigned dividend by an unsigned divisor to produce an unsigned quotient in a register D*d*. The dividend may be a register D*j* or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M1. The divisor may be a register D*k*, an 8-bit, 16-bit , or 32-bit immediate value, or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M or M2.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result is set. Undefined after overflow or division by zero. Cleared otherwise.

Z: Set if the result is zero. Undefined after overflow or division by zero. Cleared otherwise.

V: Set if the unsigned result does not fit in the result register D*d*. Undefined after division by zero. Cleared otherwise.

C: Set if divisor was zero. Cleared otherwise. (Indicates division by zero).

Detailed Instruction Formats

REG-REG

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	1	0	QUOTIENT REGISTER D*d*			3n
0	0	DIVIDEND REGISTER D*j*			DIVISOR REGISTER D*k*			mb

```
1B 3n mb                    DIVU      Dd,Dj,Dk
```

REG-IMM1/2/4

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	1	0	QUOTIENT REGISTER D*d*			3n
0	1	DIVIDEND REGISTER D*j*			1	IMM_SIZE (.B, .W, –, .L)		mb
IMMEDIATE DATA (divisor)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								

```
1B 3n mb i1                 DIVU.B    Dd,Dj,#opr8i
1B 3n mb i2 i1              DIVU.W    Dd,Dj,#opr16i ;short-imm better for some values
1B 3n mb i4 i3 i2 i1        DIVU.L    Dd,Dj,#opr32i ;short-imm better for some values
```

REG-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	1	0	QUOTIENT REGISTER D*d*			3n
0	1	DIVIDEND REGISTER D*j*				M2_SIZE (.B, .W, –, .L)		mb
OPR POSTBYTE (for M2 divisor)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 3n mb xb                 DIVU.bwl  Dd,Dj,#oprsxe4i
1B 3n mb xb                 DIVU.bwl  Dd,Dj,Dk ;see more efficient REG-REG version
1B 3n mb xb                 DIVU.bwl  Dd,Dj,(opru4,xys)
1B 3n mb xb                 DIVU.bwl  Dd,Dj,{(+-xy)|(xy+-)|(-s)|(s+)}
1B 3n mb xb                 DIVU.bwl  Dd,Dj,(Di,xys)
1B 3n mb xb                 DIVU.bwl  Dd,Dj,[Di,xy]
1B 3n mb xb x1              DIVU.bwl  Dd,Dj,(oprs9,xysp)
1B 3n mb xb x1              DIVU.bwl  Dd,Dj,[oprs9,xysp]
1B 3n mb xb x1              DIVU.bwl  Dd,Dj,opru14
1B 3n mb xb x2 x1           DIVU.bwl  Dd,Dj,(opru18,Di)
1B 3n mb xb x2 x1           DIVU.bwl  Dd,Dj,opru18
1B 3n mb xb x3 x2 x1        DIVU.bwl  Dd,Dj,(opr24,xysp)
1B 3n mb xb x3 x2 x1        DIVU.bwl  Dd,Dj,[opr24,xysp]
1B 3n mb xb x3 x2 x1        DIVU.bwl  Dd,Dj,(opru24,Di)
1B 3n mb xb x3 x2 x1        DIVU.bwl  Dd,Dj,opr24
1B 3n mb xb x3 x2 x1        DIVU.bwl  Dd,Dj,[opr24]
```

OPR/1/2/3-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	1	0	QUOTIENT REGISTER D*d*			3n
0	1	M1_SIZE (.B, .W, .P, .L)		M2_SIZE (.B, .W, .P, .L)		1	0	mb
OPR POSTBYTE (for M1 dividend)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
OPR POSTBYTE (for M2 divisor)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

page 2 Opcode postbyte	M1 object code	M2 object code	Instruction Mnemonic		Source Format for M1 (Dividend) (select 1 option in this col)	Source Format for M2 (Divisor) (select 1 option in this col)
1B 3n mb	xb	xb	DIVU.*bwplbwpl*	Dd,	#*oprsxe4i*,	#*oprsxe4i*
	xb	xb			*Dj*,	*Dk*
	xb	xb			(*opru4*,*xys*),	(*opru4*,*xys*)
	xb	xb			(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+),	(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+)
	xb	xb			(*Dj*,*xys*),	(*Dk*,*xys*)
	xb	xb			[*Dj*,*xy*],	[*Dk*,*xy*]
	xb x1	xb x1			(*oprs9*,*xysp*),	(*oprs9*,*xysp*)
	xb x1	xb x1			[*oprs9*,*xysp*],	[*oprs9*,*xysp*]
	xb x1	xb x1			*opru14*,	*opru14*
	xb x2 x1	xb x2 x1			(*opru18*,*Dj*),	(*opru18*,*Dk*)
	xb x2 x1	xb x2 x1			*opru18*,	*opru18*
	xb x3 x2 x1	xb x3 x2 x1			(*opr24*,*xysp*),	(*opr24*,*xysp*)
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*,*xysp*],	[*opr24*,*xysp*]
	xb x3 x2 x1	xb x3 x2 x1			(*opru24*,*Dj*),	(*opru24*,*Dk*)
	xb x3 x2 x1	xb x3 x2 x1			*opr24*,	*opr24*
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*],	[*opr24*]

All combinations are valid although some, such as specifying a data register for both M1 and M2 can be done more efficiently using the REG-REG version of the instruction.

Instruction Fields

QUOTIENT REGISTER- This field specifies the number of the data register D*d* used for the result (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

DIVIDEND REGISTER - This field specifies the number of the data register D*j* used as dividend (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

DIVISOR REGISTER - This field specifies the number of the data register D*k* used as divisor (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMM_SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01) or 32-bit long-word (0b11) as the size of the divisor. The 0b10 combination is not available for the REG-IMM version of the instruction because those codes are used for the OPR-OPR version.

M1_SIZE and M2_SIZE - These fields specify the size of M1 (dividend) and M2 (divisor) which use the general OPR addressing mode to specify short-immediate, register, or memory operands (0b00 = 8-bit byte, 0b01 = 16-bit word, 0b10 = 24-bit pointer, and 0b11 = 32-bit long-word). When a short-immediate operand is specified, it is internally sign-extended to the size specified by the M1_SIZE and/or M2_SIZE specifications. When a register is specified, it determines the size and the M1_SIZE and/or M2_SIZE specifications are ignored.

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size specified by IMM_SIZE.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Using OPR addressing mode to specify a register operand for both the dividend and the divisor, is less efficient than using the REG-REG version of the instruction.

EOR — Exclusive OR — EOR

Operation

(D*i*) ^ (M) ⇒ D*i*

Syntax Variations	Addressing Modes
`EOR Di,#oprimmsz`	**IMM1/2/4**
`EOR Di,oprmemreg`	**OPR/1/2/3**

Description

Bitwise Exclusive-OR register D*i* with a memory operand and store the result to D*i*. When the operand is an immediate value, it has the same size as register D*i*. In the case of the general OPR addressing operand, *oprmemreg* can be a sign-extended immediate value (–1, 1, 2, 3..14, 15), a data register, a memory operand the same size as D*i* at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	–

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Cleared.

Detailed Instruction Formats

IMM1/2/4

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	1	1	1	1	SD REGISTER D*i*			7p
IMMEDIATE DATA								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								

```
1B 7p i1                    EOR     Di,#opr8i ;for Di = 8-bit D0 or D1
1B 7p i2 i1                 EOR     Di,#opr16i ;for Di = 16-bit D2, D3, D4, or D5
1B 7p i4 i3 i2 i1           EOR     Di,#opr32i ;for Di = 32-bit D6 or D7
```

OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
1	0	0	0	1	SD REGISTER D*i*			8q
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								x1
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								x2
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								x3

```
1B 8q xb                    EOR     Di,#oprsxe4i ;-1, +1, 2, 3...14, 15
1B 8q xb                    EOR     Di,Dj
1B 8q xb                    EOR     Di,(opru4,xys)
1B 8q xb                    EOR     Di,{(+-xy)|(xy+-)|(-s)|(s+)}
1B 8q xb                    EOR     Di,(Dj,xys)
1B 8q xb                    EOR     Di,[Dj,xy]
1B 8q xb x1                 EOR     Di,(oprs9,xysp)
1B 8q xb x1                 EOR     Di,[oprs9,xysp]
1B 8q xb x1                 EOR     Di,opru14
1B 8q xb x2 x1              EOR     Di,(opru18,Dj)
1B 8q xb x2 x1              EOR     Di,opru18
1B 8q xb x3 x2 x1           EOR     Di,(opr24,xysp)
1B 8q xb x3 x2 x1           EOR     Di,[opr24,xysp]
1B 8q xb x3 x2 x1           EOR     Di,(opru24,Dj)
1B 8q xb x3 x2 x1           EOR     Di,opr24
1B 8q xb x3 x2 x1           EOR     Di,[opr24]
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as a source operand and for the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size of the register D*i*.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

EXG — Exchange Register Contents — EXG

Syntax Variations	Addressing Modes
EXG *cpureg*,*cpureg*	INH

Description

Exchange contents of CPU registers.

If both registers have the same size, a direct exchange is performed.

If the first register is smaller than the second register, it is sign-extended and written to the second register. In this case the first register is not changed. When the first register is smaller than the second register, the SEX instruction mnemonic may be used instead of EXG.

If the first register is larger than the second register, the smaller register is sign-extended as it is transferred into the larger register and the larger register is truncated during the transfer into the smaller register. These are not considered useful operations, this description simply documents what would happen if these unexpected combinations occur.

The two special cases EXG CCW,CCL and EXG CCW,CCH are ambiguous so CCW is not changed (this is equivalent to a NOP instruction).

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

In some cases (such as exchanging CCL with D0) the exchange instruction can cause the contents of another register to be written into the CCR so the CCR effects shown above do not apply. Unused bits in the CCR cannot be changed by any exchange instruction. The X interrupt mask can be cleared by an instruction in supervisor state but cannot be set (changed from 0 to 1) by any exchange instruction. In user state, the X and I interrupt masks cannot be changed by any exchange instruction.

Detailed Instruction Formats

INH

7	6	5	4	3	2	1	0	
1	0	1	0	1	1	1	0	AE
FIRST (SOURCE) REGISTER				SECOND (DESTINATION) REGISTER				eb

AE eb EXG *cpureg*,*cpureg*

Table 6-1. Exchange and Sign-Extend Postbyte (eb) Coding Map

	source	D2	D3	D4	D5	D0	D1	D6	D7	X	Y	S	–	CCH	CCL	CCW	
destination		0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
D2	-0	–	D3 ⇔ D2	D4 ⇔ D2	D5 ⇔ D2	sex:D0 ⇒ D2	sex:D1 ⇒ D2	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		sex:CCH ⇒ D2	sex:CCL ⇒ D2	CCW ⇔ D2	
D3	-1	D2 ⇔ D3	–	D4 ⇔ D3	D5 ⇔ D3	sex:D0 ⇒ D3	sex:D1 ⇒ D3	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		sex:CCH ⇒ D3	sex:CCL ⇒ D3	CCW ⇔ D3	
D4	-2	D2 ⇔ D4	D3 ⇔ D4	–	D5 ⇔ D4	sex:D0 ⇒ D4	sex:D1 ⇒ D4	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		sex:CCH ⇒ D4	sex:CCL ⇒ D4	CCW ⇔ D4	
D5	-3	D2 ⇔ D5	D3 ⇔ D5	D4 ⇔ D5	–	sex:D0 ⇒ D5	sex:D1 ⇒ D5	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		sex:CCH ⇒ D5	sex:CCL ⇒ D5	CCW ⇔ D5	
D0	-4	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	–	D1 ⇔ D0	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		CCH ⇔ D0	CCL ⇔ D0	Big ⇔Small	
D1	-5	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	D0 ⇔ D1	–	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		CCH ⇔ D1	CCL ⇔ D1	Big ⇔Small	
D6	-6	sex:D2 ⇒ D6	sex:D3 ⇒ D6	sex:D4 ⇒ D6	sex:D5 ⇒ D6	sex:D0 ⇒ D6	sex:D1 ⇒ D6	–	D7 ⇔ D6	sex:X ⇒ D6	sex:Y ⇒ D6	sex:S ⇒ D6		sex:CCH ⇒ D6	sex:CCL ⇒ D6	sex:CCW ⇒ D6	
D7	-7	sex:D2 ⇒ D7	sex:D3 ⇒ D7	sex:D4 ⇒ D7	sex:D5 ⇒ D7	sex:D0 ⇒ D7	sex:D1 ⇒ D7	D6 ⇔ D7	–	sex:X ⇒ D7	sex:Y ⇒ D7	sex:S ⇒ D7		sex:CCH ⇒ D7	sex:CCL ⇒ D7	sex:CCW ⇒ D7	
X	-8	sex:D2 ⇒ X	sex:D3 ⇒ X	sex:D4 ⇒ X	sex:D5 ⇒ X	sex:D0 ⇒ X	sex:D1 ⇒ X	Big ⇔Small	Big ⇔Small	–	Y ⇔ X	S ⇔ X		sex:CCH ⇒ X	sex:CCL ⇒ X	sex:CCW ⇒ X	
Y	-9	sex:D2 ⇒ Y	sex:D3 ⇒ Y	sex:D4 ⇒ Y	sex:D5 ⇒ Y	sex:D0 ⇒ Y	sex:D1 ⇒ Y	Big ⇔Small	Big ⇔Small	X ⇔ Y	–	S ⇔ Y		sex:CCH ⇒ Y	sex:CCL ⇒ Y	sex:CCW ⇒ Y	
S	-A	sex:D2 ⇒ S	sex:D3 ⇒ S	sex:D4 ⇒ S	sex:D5 ⇒ S	sex:D0 ⇒ S	sex:D1 ⇒ S	Big ⇔Small	Big ⇔Small	X ⇔ S	Y ⇔ S	–		sex:CCH ⇒ S	sex:CCL ⇒ S	sex:CCW ⇒ S	
reserved	-B																
CCH	-C	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	D0 ⇔ CCH	D1 ⇔ CCH	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		–	CCL ⇔ CCH	NOP	
CCL	-D	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	D0 ⇔ CCL	D1 ⇔ CCL	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		CCH ⇔ CCL	–	NOP	
CCW	-E	D2 ⇔ CCW	D3 ⇔ CCW	D4 ⇔ CCW	D5 ⇔ CCW	sex:D0 ⇒ CCW	sex:D1 ⇒ CCW	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		sex:CCH ⇒ CCW	sex:CCL ⇒ CCW	–	
	-F																–

EXG Big,Small: Small register gets low part of Big register, Big register gets sign-extended Small register. These cases are not expected to be useful in application programs.
EXG CCW,CCH and EXG CCW,CCL are ambiguous cases so CCW is not changed (equivalent to NOP)

INC — Increment — INC

Operation

$(D_i) + 1 \Rightarrow D_i$

$(M) + 1 \Rightarrow M$

Syntax Variations	Addressing Modes
`INC Di`	**INH**
`INC.bwl oprmemreg`	**OPR/1/2/3**

Description

Increment a register D*i* or memory operand M. The memory operand *oprmemreg* can be a data register, a memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. The size of the memory operand M is determined by the suffix (b=8 bit byte, w=16 bit word, or l=32 bit long-word). If the OPR memory addressing mode is used to specify a data register D*j*, the register determines the size for the operation and the *.bwl* suffix is ignored. *It is inappropriate to specify a short immediate operand using the OPR addressing mode for this instruction because it is not possible to modify the immediate operand.*

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	–

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Set if there was a two's complement overflow as a result of the operation. Cleared otherwise.

Detailed Instruction Formats

INH

7	6	5	4	3	2	1	0	
0	0	1	1	0	SD REGISTER D*i*			3n

`3n` `INC` `Di`

OPR/1/2/3

7	6	5	4	3	2	1	0	
1	0	0	1	1	1	SIZE (.B, .W, –, .L)		9p
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
9p xb                    INC.bwl    #oprsxe4i ;not appropriate for destination
9p xb                    INC.bwl    Dj ;INH version is more efficient
9p xb                    INC.bwl    (opru4,xys)
9p xb                    INC.bwl    {(+-xy)|(xy+-)|(-s)|(s+)}
9p xb                    INC.bwl    (Dj,xys)
9p xb                    INC.bwl    [Dj,xy]
9p xb x1                 INC.bwl    (oprs9,xysp)
9p xb x1                 INC.bwl    [oprs9,xysp]
9p xb x1                 INC.bwl    opru14
9p xb x2 x1              INC.bwl    (opru18,Dj)
9p xb x2 x1              INC.bwl    opru18
9p xb x3 x2 x1           INC.bwl    (opr24,xysp)
9p xb x3 x2 x1           INC.bwl    [opr24,xysp]
9p xb x3 x2 x1           INC.bwl    (opru24,Dj)
9p xb x3 x2 x1           INC.bwl    opr24
9p xb x3 x2 x1           INC.bwl    [opr24]
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as a source operand and for the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01), or 32-bit long-word (0b11) as the size of the operation.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

JMP — Jump — JMP

Operation

Effective Address ⇒ PC

Syntax Variations	Addressing Modes
`JMP` *opr24a*	**EXT3**
`JMP` *oprmemreg*	**OPR/1/2/3**

Description

Unconditional jump to extended address.

A JMP instruction causes the instruction queue to be refilled before execution resumes at the new address.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

EXT3

7	6	5	4	3	2	1	0	
1	0	1	1	1	0	1	0	BA
ADDRESS[23:16]								a3
ADDRESS[15:8]								a2
ADDRESS[7:0]								a1

```
BA a3 a2 a1          JMP     opr24a
```

OPR/1/2/3

7	6	5	4	3	2	1	0	
1	0	1	0	1	0	1	0	AA
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
AA xb                JMP     #oprsxe4i ;not appropriate for destination
AA xb                JMP     Di ;not appropriate for a jump destination
AA xb                JMP     (opru4,xys)
AA xb                JMP     {(+-xy)|(xy+-)|(-s)|(s+)}
AA xb                JMP     (Di,xys)
AA xb                JMP     [Di,xy]
AA xb x1             JMP     (oprs9,xysp)
AA xb x1             JMP     [oprs9,xysp]
AA xb x1             JMP     opru14
```

```
AA xb x2 x1                JMP        (opru18,Di)
AA xb x2 x1                JMP        opru18 ;EXT version is just as efficient
AA xb x3 x2 x1             JMP        (opr24,xysp)
AA xb x3 x2 x1             JMP        [opr24,xysp]
AA xb x3 x2 x1             JMP        (opru24,Di)
AA xb x3 x2 x1             JMP        opr24 ;EXT version is more efficient
AA xb x3 x2 x1             JMP        [opr24]
```

Instruction Fields

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. *Short immediate is not appropriate as a jump destination. Di cannot be used as the destination of a jump instruction. There is no advantage to using the 18-bit or 24-bit variations of OPR addressing compared to using the 24-bit EXT version of the jump instruction.*

JSR — Jump to Subroutine — JSR

Operation

(SP) – 3 ⇒ SP
RTN[23:0] ⇒ $M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
Effective Address ⇒ PC

Syntax Variations	Addressing Modes
`JSR` *opr24a*	**EXT3**
`JSR` *oprmemreg*	**OPR/1/2/3**

Description

Sets up conditions to return to normal program flow, then transfers control to a subroutine. Uses the address of the instruction after the JSR as a return address.

Decrements the SP by three, to allow the three bytes of the return address to be stacked.

Stacks the return address (the SP points to the most-significant byte of the return address).

Jumps to the effective address.

Subroutines are normally terminated with an RTS instruction, which restores the return address from the stack.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

EXT3

7	6	5	4	3	2	1	0	
1	0	1	1	1	0	1	1	BB
ADDRESS[23:16]								a3
ADDRESS[15:8]								a2
ADDRESS[7:0]								a1

```
BB a3 a2 a1                    JSR        opr24a
```

OPR/1/2/3

7	6	5	4	3	2	1	0	
1	0	1	0	1	0	1	1	AB
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
AB xb                          JSR        #oprsxe4i ;not appropriate for destination
AB xb                          JSR        Di ;not appropriate for a jump destination
```

```
AB xb                    JSR     (opru4,xys)
AB xb                    JSR     {(+-xy)|(xy+-)|(-s)|(s+)}
AB xb                    JSR     (Di,xys)
AB xb                    JSR     [Di,xy]
AB xb x1                 JSR     (oprs9,xysp)
AB xb x1                 JSR     [oprs9,xysp]
AB xb x1                 JSR     opru14
AB xb x2 x1              JSR     (opru18,Di)
AB xb x2 x1              JSR     opru18 ;EXT version is just as efficient
AB xb x3 x2 x1           JSR     (opr24,xysp)
AB xb x3 x2 x1           JSR     [opr24,xysp]
AB xb x3 x2 x1           JSR     (opru24,Di)
AB xb x3 x2 x1           JSR     opr24 ;EXT version is more efficient
AB xb x3 x2 x1           JSR     [opr24]
```

Instruction Fields

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. *Short immediate is not appropriate as a JSR destination. Di cannot be used as the destination of a JSR instruction. There is no advantage to using the 18-bit or 24-bit variations of OPR addressing compared to using the 24-bit EXT version of the JSR instruction.*

LD — Load (D*i*, X, Y, or SP) — LD

Operation

(M) ⇒ D*i*

(M) ⇒ X

(M) ⇒ Y

(M) ⇒ SP

Syntax Variations	Addressing Modes
`LD Di,#oprimmsz`	**IMM1/2/4** (same size as D*i*)
`LD Di,opr24a`	**EXT3** (24-bit address)
`LD Di,oprmemreg`	**OPR/1/2/3**
`LD xy,#opr18i`	**IMM2** (efficient 18-bit)
`LD xy,#opr24i`	**IMM3** (same size as X or Y)
`LD xy,opr24a`	**EXT3** (24-bit address)
`LD xy,oprmemreg`	**OPR/1/2/3**
`LD S,#opr24i`	**IMM3** (same size as SP)
`LD S,oprmemreg`	**OPR/1/2/3**

Description

Load a register D*i*, X, Y, or SP with the contents of a memory location. In the case of the general OPR addressing operand, *oprmemreg* can be a sign-extended immediate value (–1, 1, 2, 3..14, 15), a data register, a memory operand the same size as D*i* , X, Y, or SP at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. There is also an efficient 24-bit extended addressing mode version of the instructions for D*i*, X and Y. For immediate addressing mode, the memory operand is usually the same size as the register that is being loaded, however, in addition to the 24-bit immediate versions of LD X and LD Y, there are also more efficient 18-bit immediate versions for X and Y which compliment the 18-bit OPR extended addressing mode to work efficiently with variables in the first 256 kilobyte of memory.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	–

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Cleared.

Detailed Instruction Formats

IMM1/2/4 (D*i*)

7	6	5	4	3	2	1	0	
1	0	0	1	0	REGISTER			9p
IMMEDIATE DATA								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								

```
9p i1                    LD      Di,#opr8i ;for Di = 8-bit D0 or D1
9p i2 i1                 LD      Di,#opr16i ;for Di = 16-bit D2, D3, D4, or D5
9p i4 i3 i2 i1           LD      Di,#opr32i ;for Di = 32-bit D6 or D7
```

EXT3 (D*i*)

7	6	5	4	3	2	1	0	
1	0	1	1	0	REGISTER			Bn
ADDRESS[23:16]								a3
ADDRESS[15:8]								a2
ADDRESS[7:0]								a1

```
Bn a3 a2 a1              LD      Di,opr24a
```

OPR/1/2/3 (D*i*)

7	6	5	4	3	2	1	0	
1	0	1	0	0	REGISTER			An
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
An xb                    LD      Di,#oprsxe4i ;-1, +1, 2, 3...14, 15
An xb                    LD      Di,Dj
An xb                    LD      Di,(opru4,xys)
An xb                    LD      Di,{(+-xy)|(xy+-)|(-s)|(s+)}
An xb                    LD      Di,(Dj,xys)
An xb                    LD      Di,[Dj,xy]
An xb x1                 LD      Di,(oprs9,xysp)
An xb x1                 LD      Di,[oprs9,xysp]
An xb x1                 LD      Di,opru14
An xb x2 x1              LD      Di,(opru18,Dj)
An xb x2 x1              LD      Di,opru18
An xb x3 x2 x1           LD      Di,(opr24,xysp)
An xb x3 x2 x1           LD      Di,[opr24,xysp]
An xb x3 x2 x1           LD      Di,(opru24,Dj)
An xb x3 x2 x1           LD      Di,opr24
An xb x3 x2 x1           LD      Di,[opr24]
```

IMM2, IMM3 (X or Y)

7	6	5	4	3	2	1	0	
1	1	IMMEDIATE DATA[17:16]		1	0	1	Y/X	op
IMMEDIATE DATA[15:8]								i2
IMMEDIATE DATA[7:0]								i1

```
op i2 i1                 LD      xy,#opr18i ;uses 4 opcodes ea. for X & Y
```

7	6	5	4	3	2	1	0	
1	0	0	1	1	0	0	Y/X	9p
IMMEDIATE DATA[23:16]								i3
IMMEDIATE DATA[16:8]								i2
IMMEDIATE DATA[7:0]								i1

```
9p i3 i2 i1                    LD          xy,#opr24i
```

EXT3 (X or Y)

7	6	5	4	3	2	1	0	
1	0	1	1	1	0	0	Y/X	Bp
ADDRESS[23:16]								a3
ADDRESS[15:8]								a2
ADDRESS[7:0]								a1

```
Bp a3 a2 a1                    LD          xy,opr24a
```

OPR/1/2/3 (X or Y)

7	6	5	4	3	2	1	0	
1	0	1	0	1	0	0	Y/X	Ap
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
Ap xb                          LD          xy,#oprsxe4i ;-1, +1, 2, 3...14, 15
Ap xb                          LD          xy,Dj
Ap xb                          LD          xy,(opru4,xys)
Ap xb                          LD          xy,{(+-xy)|(xy+-)|(-s)|(s+)}
Ap xb                          LD          xy,(Dj,xys)
Ap xb                          LD          xy,[Dj,xy]
Ap xb x1                       LD          xy,(oprs9,xysp)
Ap xb x1                       LD          xy,[oprs9,xysp]
Ap xb x1                       LD          xy,opru14
Ap xb x2 x1                    LD          xy,(opru18,Dj)
Ap xb x2 x1                    LD          xy,opru18
Ap xb x3 x2 x1                 LD          xy,(opr24,xysp)
Ap xb x3 x2 x1                 LD          xy,[opr24,xysp]
Ap xb x3 x2 x1                 LD          xy,(opru24,Dj)
Ap xb x3 x2 x1                 LD          xy,opr24
Ap xb x3 x2 x1                 LD          xy,[opr24]
```

IMM3 (S)

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	0	0	1	1	03
IMMEDIATE DATA[23:16]								i3
IMMEDIATE DATA[16:8]								i2
IMMEDIATE DATA[7:0]								i1

```
1B 03 i3 i2 i1                 LD          S,#opr24i
```

OPR/1/2/3 (SP)

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	0	0	0	0	00
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 00 xb                    LD        S,#oprsxe4i ;-1, +1, 2, 3...14, 15
1B 00 xb                    LD        S,Dj
1B 00 xb                    LD        S,(opru4,xys)
1B 00 xb                    LD        S,{(+-xy)|(xy+-)|(-s)|(s+)}
1B 00 xb                    LD        S,(Dj,xys)
1B 00 xb                    LD        S,[Dj,xy]
1B 00 xb x1                 LD        S,(oprs9,xysp)
1B 00 xb x1                 LD        S,[oprs9,xysp]
1B 00 xb x1                 LD        S,opru14
1B 00 xb x2 x1              LD        S,(opru18,Dj)
1B 00 xb x2 x1              LD        S,opru18
1B 00 xb x3 x2 x1           LD        S,(opr24,xysp)
1B 00 xb x3 x2 x1           LD        S,[opr24,xysp]
1B 00 xb x3 x2 x1           LD        S,(opru24,Dj)
1B 00 xb x3 x2 x1           LD        S,opr24
1B 00 xb x3 x2 x1           LD        S,[opr24]
```

Instruction Fields

REGISTER - This field specifies the number of the data register D*i* which is used as the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

Y/X - This field selects either the X index register or the Y index register.

ADDRESS - This field is used for address bits used for extended addressing mode.

IMMEDIATE DATA[17:16] - This field holds address bits 17 and 16 of an 18-bit address.

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes, 3 bytes, or 4 bytes wide, depending on the size of the register D*i*, X, Y, or SP.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

LEA — Load Effective Address — LEA

Operation

00:Effective Address ⇒ D6 or D7 ;zero-extended 24-bit effective address into a 32-bit register

Effective Address ⇒ SP

Effective Address ⇒ X

Effective Address ⇒ Y

(SP) + (IMM8) ⇒ SP; signed 8-bit immediate offset

(X) + (IMM8) ⇒ X; signed 8-bit immediate offset

(Y) + (IMM8) ⇒ Y; signed 8-bit immediate offset

Syntax Variations	Addressing Modes
`LEA D67,oprmemreg`	**OPR/1/2/3**
`LEA S,oprmemreg`	**OPR/1/2/3**
`LEA xy,oprmemreg`	**OPR/1/2/3**
`LEA S,(opr8i,S)`	**IMM1** (8-bit signed offset)
`LEA xy,(opr8i,xy)`	**IMM1** (8-bit signed offset)

Description

Load Di, X, Y, or SP with an effective address or add a signed 8-bit immediate value to X, Y, or SP. This description needs quite a bit of work to explain the odd cases such as short-imm, Di, pre/post inc/dec, and indirect variations of OPR addressing.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

OPR/1/2/3 (D6 or D7)

7	6	5	4	3	2	1	0	
0	0	0	0	0	1	1	D7/D6	0p
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
0p xb                    LEA     D67,#oprsxe4i ;not appropriate for LEA
0p xb                    LEA     D67,Dj ;not appropriate for LEA
0p xb                    LEA     D67,(opru4,xys)
0p xb                    LEA     D67,{(+-xy)|(xy+-)|(-s)|(s+)}
0p xb                    LEA     D67,(Dj,xys)
0p xb                    LEA     D67,[Dj,xy]
0p xb x1                 LEA     D67,(oprs9,xysp)
0p xb x1                 LEA     D67,[oprs9,xysp]
0p xb x1                 LEA     D67,opru14
0p xb x2 x1              LEA     D67,(opru18,Dj)
0p xb x2 x1              LEA     D67,opru18
0p xb x3 x2 x1           LEA     D67,(opr24,xysp)
0p xb x3 x2 x1           LEA     D67,[opr24,xysp]
0p xb x3 x2 x1           LEA     D67,(opru24,Dj)
0p xb x3 x2 x1           LEA     D67,opr24
0p xb x3 x2 x1           LEA     D67,[opr24]
```

OPR/1/2/3 (SP)

7	6	5	4	3	2	1	0	
0	0	0	0	1	0	1	0	0A
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
0A xb                    LEA     S,#oprsxe4i ;not appropriate for LEA
0A xb                    LEA     S,Dj ;not appropriate for LEA
0A xb                    LEA     S,(opru4,xys)
0A xb                    LEA     S,{(+-xy)|(xy+-)|(-s)|(s+)}
0A xb                    LEA     S,(Dj,xys)
0A xb                    LEA     S,[Dj,xy]
0A xb x1                 LEA     S,(oprs9,xysp)
0A xb x1                 LEA     S,[oprs9,xysp]
0A xb x1                 LEA     S,opru14
0A xb x2 x1              LEA     S,(opru18,Dj)
0A xb x2 x1              LEA     S,opru18
0A xb x3 x2 x1           LEA     S,(opr24,xysp)
0A xb x3 x2 x1           LEA     S,[opr24,xysp]
0A xb x3 x2 x1           LEA     S,(opru24,Dj)
0A xb x3 x2 x1           LEA     S,opr24
0A xb x3 x2 x1           LEA     S,[opr24]
```

OPR/1/2/3 (X or Y)

7	6	5	4	3	2	1	0	
0	0	0	0	1	0	0	Y/X	0p
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
0p xb                    LEA     xy,#oprsxe4i ;not appropriate for LEA
0p xb                    LEA     xy,Dj ;not appropriate for LEA
0p xb                    LEA     xy,(opru4,xys)
0p xb                    LEA     xy,{(+-xy)|(xy+-)|(-s)|(s+)}
0p xb                    LEA     xy,(Dj,xys)
0p xb                    LEA     xy,[Dj,xy]
0p xb x1                 LEA     xy,(oprs9,xysp)
0p xb x1                 LEA     xy,[oprs9,xysp]
0p xb x1                 LEA     xy,opru14
0p xb x2 x1              LEA     xy,(opru18,Dj)
0p xb x2 x1              LEA     xy,opru18
0p xb x3 x2 x1           LEA     xy,(opr24,xysp)
0p xb x3 x2 x1           LEA     xy,[opr24,xysp]
0p xb x3 x2 x1           LEA     xy,(opru24,Dj)
0p xb x3 x2 x1           LEA     xy,opr24
0p xb x3 x2 x1           LEA     xy,[opr24]
```

IMM1 (SP)

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	0	1A
IMMEDIATE DATA								i1

```
1A i1                    LEA     S,(opr8i,S) ;adjust by 8-bit signed offset
```

IMM1 (X or Y)

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	0	Y/X	1p
IMMEDIATE DATA								i1

```
1p i1                    LEA     xy,(opr8i,xy) ;adjust by 8-bit signed offset
```

Instruction Fields

D7/D6 - This field selects either D6 (0) or D7 (1) as the destination register.

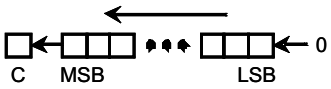
Y/X - This field selects either the X index register or the Y index register.

IMMEDIATE DATA - This field contains the signed 8-bit immediate operand.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Unlike other indexed addressing, LEA uses the address produced by the addressing mode rather than the operand that is located at this address. Short immediate and register D*i* variations of OPR addressing are not appropriate for LEA because these values do not have an associated effective address.

LSL — Logical Shift Left — LSL

Operation



Syntax Variations	Addressing Modes
`LSL Dd,Ds,Dn`	**REG-REG**
`LSL Dd,Ds,#opr5i`	**REG-IMM**
`LSL.bwpl Dd,oprmemreg,#opr5i`	**OPR/1/2/3-IMM**
`LSL.bwpl Dd,oprmemreg,oprmemreg`	**OPR/1/2/3-OPR/1/2/3**
`LSL.bwpl oprmemreg,#opr1i`	**OPR/1/2/3-IMM**

Description

Logically shift an operand n bit-positions to the left. The result is saved in a CPU register, or in the case of a 2-operand memory shift the result is saved in the same memory location used for the source. The operand to be shifted may be one of the eight data registers or an 8-, 16-, 24-, or 32-bit memory operand. In the case of the general OPR addressing operand, *oprmemreg* can be a data register, a memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. The number of bit positions to shift the operand is supplied in a 1-bit or 5-bit immediate operand or in the low order 5 bits of a register or byte-sized memory operand. When the number of bit positions to shift is provided in a 5-bit immediate value, the least significant bit is encoded in the sb postbyte and the higher four bits are encoded as a short-immediate value in the xb postbyte. If the destination register is wider than the source operand, the source operand is zero-extended to the width of the destination register before shifting. If the destination register is narrower than the source operand, the operand is shifted and then truncated to the width of the destination register. Zero is shifted into the LSB and the MSB is shifted out through the carry bit (C).

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Set if a one would be shifted out of the MSB during a bit-by-bit shift. Set if truncation changes the sign or magnitude of the result. Cleared otherwise.

C: Set if the last bit shifted out of the MSB of the operand was set before the shift, cleared otherwise. If the shift count is 0, C is not changed.

Detailed Instruction Formats

REG-REG

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=0	L/R=1	1	0	N[0]	SOURCE REGISTER D*s*			sb
1	0	1	1	1	PARAMETER REGISTER D*n*			xb

```
1n sb xb                    LSL        Dd,Ds,Dn
```

REG-IMM (efficient shift by 1 (N[0]=0) or by 2 (N[0]=1) positions)

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=0	L/R=1	0	0	N[0]	SOURCE REGISTER D*s*			sb

```
1n sb                       LSL        Dd,Ds,#opr1i
```

REG-IMM (normal shift by 0 to 31 positions)

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=0	L/R=1	0	1	N[0]	SOURCE REGISTER D*s*			sb
0	1	1	1	N[4:1]				xb

```
1n sb xb                    LSL        Dd,Ds,#opr5i ;N[0] in sb, N[4:1] in xb
```

OPR/1/2/3-IMM (efficient shift by 1 (N[0]=0) or by 2 (N[0]=1) positions)

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=0	L/R=1	1	0	N[0]	0	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (specifes source operand to be shifted)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1n sb xb                    LSL.bwpl   #oprsxe4i,#opr1i
1n sb xb                    LSL.bwpl   Ds,#opr1i ;see more efficient REG-IMM version
1n sb xb                    LSL.bwpl   (opru4,xys),#opr1i
1n sb xb                    LSL.bwpl   {(+-xy)|(xy+-)|(-s)|(s+)},#opr1i
1n sb xb                    LSL.bwpl   (Di,xys),#opr1i
1n sb xb                    LSL.bwpl   [Di,xy],#opr1i
1n sb xb x1                 LSL.bwpl   (oprs9,xysp),#opr1i
1n sb xb x1                 LSL.bwpl   [oprs9,xysp],#opr1i
1n sb xb x1                 LSL.bwpl   opru14,#opr1i
1n sb xb x2 x1              LSL.bwpl   (opru18,Di),#opr1i
1n sb xb x2 x1              LSL.bwpl   opru18,#opr1i
1n sb xb x3 x2 x1           LSL.bwpl   (opr24,xysp),#opr1i
1n sb xb x3 x2 x1           LSL.bwpl   [opr24,xysp],#opr1i
1n sb xb x3 x2 x1           LSL.bwpl   (opru24,Di),#opr1i
1n sb xb x3 x2 x1           LSL.bwpl   opr24,#opr1i
1n sb xb x3 x2 x1           LSL.bwpl   [opr24],#opr1i
```

OPR/1/2/3-IMM (normal shift by 0 to 31 positions)

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=0	L/R=1	1	1	N[0]	0	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (specifes source operand to be shifted)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
0	1	1	1	N[4:1]				xb

```
1n sb xb xb                    LSL.bwpl    #oprsxe4i,#opr5i
1n sb xb xb                    LSL.bwpl    Di,#opr5i ;see more efficient REG-IMM version
1n sb xb xb                    LSL.bwpl    (opru4,xys),#opr5i
1n sb xb xb                    LSL.bwpl    {(+-xy)|(xy+-)|(-s)|(s+)},#opr5i
1n sb xb xb                    LSL.bwpl    (Di,xys),#opr5i
1n sb xb xb                    LSL.bwpl    [Di,xy],#opr5i
1n sb xb x1 xb                 LSL.bwpl    (oprs9,xysp),#opr5i
1n sb xb x1 xb                 LSL.bwpl    [oprs9,xysp],#opr5i
1n sb xb x1 xb                 LSL.bwpl    opru14,#opr5i
1n sb xb x2 x1 xb              LSL.bwpl    (opru18,Di),#opr5i
1n sb xb x2 x1 xb              LSL.bwpl    opru18,#opr5i
1n sb xb x3 x2 x1 xb           LSL.bwpl    (opr24,xysp),#opr5i
1n sb xb x3 x2 x1 xb           LSL.bwpl    [opr24,xysp],#opr5i
1n sb xb x3 x2 x1 xb           LSL.bwpl    (opru24,Di),#opr5i
1n sb xb x3 x2 x1 xb           LSL.bwpl    opr24,#opr5i
1n sb xb x3 x2 x1 xb           LSL.bwpl    [opr24],#opr5i
```

OPR/1/2/3-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=0	L/R=1	1	1	N[0]	0	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (for source operand)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
OPR POSTBYTE (for number of shifts - byte sized memory operands)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

Opcode postbyte	Source operand object code	Parameter # of shifts object code	Instruction Mnemonic		Source Format for Source Operand (select 1 option in this col)	Source Format for Parameter (# of shifts) (select 1 option in this col)
1n sb	xb	xb	LSL.*bwpl*	Dd,	#*oprsxe4i*,	#*oprsxe4i*
	xb	xb			*Ds*,	*Dn*
	xb	xb			(*opru4*,*xys*),	(*opru4*,*xys*)
	xb	xb			(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+),	(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+)
	xb	xb			(*Dj*,*xys*),	(*Dk*,*xys*)
	xb	xb			[*Dj*,*xy*],	[*Dk*,*xy*]
	xb x1	xb x1			(*oprs9*,*xysp*),	(*oprs9*,*xysp*)
	xb x1	xb x1			[*oprs9*,*xysp*],	[*oprs9*,*xysp*]
	xb x1	xb x1			*opru14*,	*opru14*
	xb x2 x1	xb x2 x1			(*opru18*,*Dj*),	(*opru18*,*Dk*)
	xb x2 x1	xb x2 x1			*opru18*,	*opru18*
	xb x3 x2 x1	xb x3 x2 x1			(*opr24*,*xysp*),	(*opr24*,*xysp*)
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*,*xysp*],	[*opr24*,*xysp*]
	xb x3 x2 x1	xb x3 x2 x1			(*opru24*,*Dj*),	(*opru24*,*Dk*)
	xb x3 x2 x1	xb x3 x2 x1			*opr24*,	*opr24*
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*],	[*opr24*]

The .bwpl suffix on the instruction mnemonic refers to the size (byte, word, pointer, or long) of the source operand. The parameter operand is always the low five bits in a byte sized memory operand.

OPR/1/2/3-IMM (2-operand memory shift by 1 (N[0]=0) or by 2 (N[0]=1) positions)

7	6	5	4	3	2	1	0	
0	0	0	1	0	x	x	x	1n
A/L=0	L/R=1	1	1	N[0]	1	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (specifes source operand to be shifted)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1n sb xb                    LSL.bwpl    #oprsxe4i,#opr1i
1n sb xb                    LSL.bwpl    Ds,#opr1i ;see more efficient REG-IMM version
1n sb xb                    LSL.bwpl    (opru4,xys),#opr1i
1n sb xb                    LSL.bwpl    {(+-xy)|(xy+-)|(-s)|(s+)},#opr1i
1n sb xb                    LSL.bwpl    (Di,xys),#opr1i
1n sb xb                    LSL.bwpl    [Di,xy],#opr1i
1n sb xb x1                 LSL.bwpl    (oprs9,xysp),#opr1i
1n sb xb x1                 LSL.bwpl    [oprs9,xysp],#opr1i
1n sb xb x1                 LSL.bwpl    opru14,#opr1i
1n sb xb x2 x1              LSL.bwpl    (opru18,Di),#opr1i
1n sb xb x2 x1              LSL.bwpl    opru18,#opr1i
1n sb xb x3 x2 x1           LSL.bwpl    (opr24,xysp),#opr1i
1n sb xb x3 x2 x1           LSL.bwpl    [opr24,xysp],#opr1i
1n sb xb x3 x2 x1           LSL.bwpl    (opru24,Di),#opr1i
1n sb xb x3 x2 x1           LSL.bwpl    opr24,#opr1i
1n sb xb x3 x2 x1           LSL.bwpl    [opr24],#opr1i
```

Instruction Fields

A/L - This bit selects arithmetic (1) or logical (0) shifts.

L/R - This bit selects the shift direction, left (1) or right (0).

DESTINATION REGISTER D*d* - This field specifies data register D*d* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) where the result of the shift is stored.

SOURCE REGISTER D*s* - This field specifies data register D*s* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) which is the source operand to be shifted.

PARAMETER REGISTER D*n* - This field specifies the number of the data register D*n* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) which is used to specify the number of positions (0–31) to shift the operand. Only the low-order 5 bits of the parameter register are used.

N[0] - This field contains the least significant bit of the 5-bit immediate operand n=0–31, or in the case of the efficient shifts, this bit selects shifting by 1 (N[0]=0) or shifting by 2 (N[0]=1).

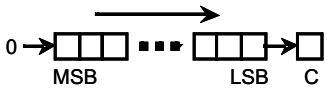
N[4:1] - This field contains the upper four bits of the 5-bit immediate operand n=0–31.

SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01), 24-bit pointer (0b10) or 32-bit long-word (0b11) as the size of the source operand.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. In the case of the parameter operand, short immediate mode is used to specify the upper four bits of the 5-bit immediate value that specifies the number of bit positions to shift the source operand.

LSR — Logical Shift Right — LSR

Operation



Syntax Variations	Addressing Modes
`LSR Dd,Ds,Dn`	**REG-REG**
`LSR Dd,Ds,#opr5i`	**REG-IMM**
`LSR.bwpl Dd,oprmemreg,#opr5i`	**OPR/1/2/3-IMM**
`LSR.bwpl Dd,oprmemreg,oprmemreg`	**OPR/1/2/3-OPR/1/2/3**
`LSR.bwpl oprmemreg,#opr1i`	**OPR/1/2/3-IMM**

Description

Logically shift an operand n bit-positions to the right. The result is saved in a CPU register, or in the case of a 2-operand memory shift the result is saved in the same memory location used for the source. The operand to be shifted may be one of the eight data registers or an 8-, 16-, 24-, or 32-bit memory operand. In the case of the general OPR addressing operand, *oprmemreg* can be a data register, a memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. The number of bit positions to shift the operand is supplied in a 1-bit or 5-bit immediate operand or in the low order 5 bits of a register or byte-sized memory operand. When the number of bit positions to shift is provided in a 5-bit immediate value, the least significant bit is encoded in the sb postbyte and the higher four bits are encoded as a short-immediate value in the xb postbyte. If the destination register is wider than the source operand, the source operand is zero-extended to the width of the destination register before shifting. If the destination register is narrower than the source operand, the operand is shifted and then truncated to the width of the destination register. Zero is shifted into the LSB and the MSB is shifted out through the carry bit (C).

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	Δ

N: Normally cleared. Set if MSB was set and shift count was 0 (no shift).

Z: Set if the result is zero. Cleared otherwise.

V: Normally cleared. Set if truncation changes the sign or magnitude of the result.

C: Set if the last bit shifted out of the LSB of the operand was set before the shift, cleared otherwise. If the shift count is 0, C is not changed.

Detailed Instruction Formats

REG-REG

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=0	L/R=0	1	0	N[0]	SOURCE REGISTER D*s*			sb
1	0	1	1	1	PARAMETER REGISTER D*n*			xb

```
1n sb xb                    LSR        Dd,Ds,Dn
```

REG-IMM (efficient shift by 1 (N[0]=0) or by 2 (N[0]=1) positions)

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=0	L/R=0	0	0	N[0]	SOURCE REGISTER D*s*			sb

```
1n sb                       LSR        Dd,Ds,#opr1i
```

REG-IMM (normal shift by 0 to 31 positions)

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=0	L/R=0	0	1	N[0]	SOURCE REGISTER D*s*			sb
0	1	1	1	N[4:1]				xb

```
1n sb xb                    LSR        Dd,Ds,#opr5i ;N[0] in sb, N[4:1] in xb
```

OPR/1/2/3-IMM (efficient shift by 1 (N[0]=0) or by 2 (N[0]=1) positions)

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=0	L/R=0	1	0	N[0]	0	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (specifes source operand to be shifted)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1n sb xb                    LSR.bwpl   #oprsxe4i,#opr1i
1n sb xb                    LSR.bwpl   Ds,#opr1i ;see more efficient REG-IMM version
1n sb xb                    LSR.bwpl   (opru4,xys),#opr1i
1n sb xb                    LSR.bwpl   {(+-xy)|(xy+-)|(-s)|(s+)},#opr1i
1n sb xb                    LSR.bwpl   (Di,xys),#opr1i
1n sb xb                    LSR.bwpl   [Di,xy],#opr1i
1n sb xb x1                 LSR.bwpl   (oprs9,xysp),#opr1i
1n sb xb x1                 LSR.bwpl   [oprs9,xysp],#opr1i
1n sb xb x1                 LSR.bwpl   opru14,#opr1i
1n sb xb x2 x1              LSR.bwpl   (opru18,Di),#opr1i
1n sb xb x2 x1              LSR.bwpl   opru18,#opr1i
1n sb xb x3 x2 x1           LSR.bwpl   (opr24,xysp),#opr1i
1n sb xb x3 x2 x1           LSR.bwpl   [opr24,xysp],#opr1i
1n sb xb x3 x2 x1           LSR.bwpl   (opru24,Di),#opr1i
1n sb xb x3 x2 x1           LSR.bwpl   opr24,#opr1i
1n sb xb x3 x2 x1           LSR.bwpl   [opr24],#opr1i
```

OPR/1/2/3-IMM (normal shift by 0 to 31 positions)

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=0	L/R=0	1	1	N[0]	0	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (specifes source operand to be shifted)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
0	1	1	1	N[4:1]				xb

```
1n sb xb xb                    LSR.bwpl    #oprsxe4i,#opr5i
1n sb xb xb                    LSR.bwpl    Di,#opr5i ;see more efficient REG-IMM version
1n sb xb xb                    LSR.bwpl    (opru4,xys),#opr5i
1n sb xb xb                    LSR.bwpl    {(+-xy)|(xy+-)|(-s)|(s+)},#opr5i
1n sb xb xb                    LSR.bwpl    (Di,xys),#opr5i
1n sb xb xb                    LSR.bwpl    [Di,xy],#opr5i
1n sb xb x1 xb                 LSR.bwpl    (oprs9,xysp),#opr5i
1n sb xb x1 xb                 LSR.bwpl    [oprs9,xysp],#opr5i
1n sb xb x1 xb                 LSR.bwpl    opru14,#opr5i
1n sb xb x2 x1 xb              LSR.bwpl    (opru18,Di),#opr5i
1n sb xb x2 x1 xb              LSR.bwpl    opru18,#opr5i
1n sb xb x3 x2 x1 xb           LSR.bwpl    (opr24,xysp),#opr5i
1n sb xb x3 x2 x1 xb           LSR.bwpl    [opr24,xysp],#opr5i
1n sb xb x3 x2 x1 xb           LSR.bwpl    (opru24,Di),#opr5i
1n sb xb x3 x2 x1 xb           LSR.bwpl    opr24,#opr5i
1n sb xb x3 x2 x1 xb           LSR.bwpl    [opr24],#opr5i
```

OPR/1/2/3-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	0	DESTINATION REGISTER D*d*			1n
A/L=0	L/R=0	1	1	N[0]	0	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (for source operand)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
OPR POSTBYTE (for number of shifts - byte sized memory operands)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

Opcode postbyte	Source operand object code	Parameter # of shifts object code	Instruction Mnemonic		Source Format for Source Operand (select 1 option in this col)	Source Format for Parameter (# of shifts) (select 1 option in this col)
1n sb	xb	xb	LSR.*bwpl*	D*d*,	#*oprsxe4i*,	#*oprsxe4i*
	xb	xb			*Ds*,	*Dn*
	xb	xb			(*opru4*,*xys*),	(*opru4*,*xys*)
	xb	xb			(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+),	(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+)
	xb	xb			(*Dj*,*xys*),	(*Dk*,*xys*)
	xb	xb			[*Dj*,*xy*],	[*Dk*,*xy*]
	xb x1	xb x1			(*oprs9*,*xysp*),	(*oprs9*,*xysp*)
	xb x1	xb x1			[*oprs9*,*xysp*],	[*oprs9*,*xysp*]
	xb x1	xb x1			*opru14*,	*opru14*
	xb x2 x1	xb x2 x1			(*opru18*,*Dj*),	(*opru18*,*Dk*)
	xb x2 x1	xb x2 x1			*opru18*,	*opru18*
	xb x3 x2 x1	xb x3 x2 x1			(*opr24*,*xysp*),	(*opr24*,*xysp*)
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*,*xysp*],	[*opr24*,*xysp*]
	xb x3 x2 x1	xb x3 x2 x1			(*opru24*,*Dj*),	(*opru24*,*Dk*)
	xb x3 x2 x1	xb x3 x2 x1			*opr24*,	*opr24*
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*],	[*opr24*]

The .bwpl suffix on the instruction mnemonic refers to the size (byte, word, pointer, or long) of the source operand. The parameter operand is always the low five bits in a byte sized memory operand.

OPR/1/2/3-IMM (2-operand memory shift by 1 (N[0]=0) or by 2 (N[0]=1) positions)

7	6	5	4	3	2	1	0	
0	0	0	1	0	x	x	x	1n
A/L=0	L/R=0	1	1	N[0]	1	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (specifes source operand to be shifted)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1n sb xb                    LSR.bwpl    #oprsxe4i,#opr1i
1n sb xb                    LSR.bwpl    Ds,#opr1i ;see more efficient REG-IMM version
1n sb xb                    LSR.bwpl    (opru4,xys),#opr1i
1n sb xb                    LSR.bwpl    {(+-xy)|(xy+-)|(-s)|(s+)},#opr1i
1n sb xb                    LSR.bwpl    (Di,xys),#opr1i
1n sb xb                    LSR.bwpl    [Di,xy],#opr1i
1n sb xb x1                 LSR.bwpl    (oprs9,xysp),#opr1i
1n sb xb x1                 LSR.bwpl    [oprs9,xysp],#opr1i
1n sb xb x1                 LSR.bwpl    opru14,#opr1i
1n sb xb x2 x1              LSR.bwpl    (opru18,Di),#opr1i
1n sb xb x2 x1              LSR.bwpl    opru18,#opr1i
1n sb xb x3 x2 x1           LSR.bwpl    (opr24,xysp),#opr1i
1n sb xb x3 x2 x1           LSR.bwpl    [opr24,xysp],#opr1i
1n sb xb x3 x2 x1           LSR.bwpl    (opru24,Di),#opr1i
1n sb xb x3 x2 x1           LSR.bwpl    opr24,#opr1i
1n sb xb x3 x2 x1           LSR.bwpl    [opr24],#opr1i
```

Instruction Fields

A/L - This bit selects arithmetic (1) or logical (0) shifts.

L/R - This bit selects the shift direction, left (1) or right (0).

DESTINATION REGISTER D*d* - This field specifies data register D*d* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) where the result of the shift is stored.

SOURCE REGISTER D*s* - This field specifies data register D*s* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) which is the source operand to be shifted.

PARAMETER REGISTER D*n* - This field specifies the number of the data register D*n* (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7) which is used to specify the number of positions (0–31) to shift the operand. Only the low-order 5 bits of the parameter register are used.

N[0] - This field contains the least significant bit of the 5-bit immediate operand n=0–31, or in the case of the efficient shifts, this bit selects shifting by 1 (N[0]=0) or shifting by 2 (N[0]=1).

N[4:1] - This field contains the upper four bits of the 5-bit immediate operand n=0–31.

SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01), 24-bit pointer (0b10) or 32-bit long-word (0b11) as the size of the source operand.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. In the case of the parameter operand, short immediate mode is used to specify the upper four bits of the 5-bit immediate value that specifies the number of bit positions to shift the source operand.

MACS — Signed Multiply and Accumulate — MACS

Operation

(D*j*) × (D*k*) + (D*d*) ⇒ D*d*

(D*j*) × IMM + (D*d*) ⇒ D*d*

(D*j*) × (M) + (D*d*) ⇒ D*d*

(M1) × (M2) + (D*d*) ⇒ D*d*

Syntax Variations	Addressing Modes
`MACS Dd,Dj,Dk`	**REG-REG**
`MACS.B Dd,Dj,#opr8i`	**REG-IMM1**
`MACS.W Dd,Dj,#opr16i`	**REG-IMM2**
`MACS.L Dd,Dj,#opr32i`	**REG-IMM4**
`MACS.bwl Dd,Dj,oprmemreg`	**REG-OPR/1/2/3**
`MACS.bwplbwplDd,oprmemreg,oprmemreg`	**OPR/1/2/3-OPR/1/2/3**

Description

Multiplies two signed two's complement operands, adds this product to a register D*d*, and stores the accumulated result to register D*d*. The first source operand may be a register D*j* or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M1. The second source operand may be a register D*k*, an 8-bit, 16-bit , or 32-bit immediate value, or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M or M2. Both source operands and the result are interpreted as signed two's complement values.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Set if the signed result of the multiply operation does not fit in the result register D*d* or if there is an overflow from the addition. Cleared otherwise.

C: Set if there is a carry from the addition. Cleared otherwise.

Detailed Instruction Formats

REG-REG

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	1	0	0	1	RESULT REGISTER D*d*			4q
1	0	SOURCE 1 REGISTER D*j*			SOURCE 2 REGISTER D*k*			mb

```
1B 4q mb                    MACS      Dd,Dj,Dk
```

REG-IMM1/2/4

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	1	0	0	1	RESULT REGISTER D*d*			4q
1	1	SOURCE REGISTER D*j*			1	IMM_SIZE (.B, .W, –, .L)		mb
IMMEDIATE DATA								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								

```
1B 4q mb i1                 MACS.B    Dd,Dj,#opr8i
1B 4q mb i2 i1              MACS.W    Dd,Dj,#opr16i ;short-imm better for some values
1B 4q mb i4 i3 i2 i1        MACS.L    Dd,Dj,#opr32i ;short-imm better for some values
```

REG-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	1	0	0	1	RESULT REGISTER D*d*			4q
1	1	SOURCE REGISTER D*j*				M2_SIZE (.B, .W, –, .L)		mb
OPR POSTBYTE (for M2)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 4q mb xb                 MACS.bwl  Dd,Dj,#oprsxe4i
1B 4q mb xb                 MACS.bwl  Dd,Dj,Dk ;see more efficient REG-REG version
1B 4q mb xb                 MACS.bwl  Dd,Dj,(opru4,xys)
1B 4q mb xb                 MACS.bwl  Dd,Dj,{(+-xy)|(xy+-)|(-s)|(s+)}
1B 4q mb xb                 MACS.bwl  Dd,Dj,(Di,xys)
1B 4q mb xb                 MACS.bwl  Dd,Dj,[Di,xy]
1B 4q mb xb x1              MACS.bwl  Dd,Dj,(oprs9,xysp)
1B 4q mb xb x1              MACS.bwl  Dd,Dj,[oprs9,xysp]
1B 4q mb xb x1              MACS.bwl  Dd,Dj,opru14
1B 4q mb xb x2 x1           MACS.bwl  Dd,Dj,(opru18,Di)
1B 4q mb xb x2 x1           MACS.bwl  Dd,Dj,opru18
1B 4q mb xb x3 x2 x1        MACS.bwl  Dd,Dj,(opr24,xysp)
1B 4q mb xb x3 x2 x1        MACS.bwl  Dd,Dj,[opr24,xysp]
1B 4q mb xb x3 x2 x1        MACS.bwl  Dd,Dj,(opru24,Di)
1B 4q mb xb x3 x2 x1        MACS.bwl  Dd,Dj,opr24
1B 4q mb xb x3 x2 x1        MACS.bwl  Dd,Dj,[opr24]
```

OPR/1/2/3-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	1	0	0	1	RESULT REGISTER D*d*			4q
1	1	M1_SIZE (.B, .W, .P, .L)		M2_SIZE (.B, .W, .P, .L)		1	0	mb
OPR POSTBYTE (for M1)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
OPR POSTBYTE (for M2)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

page 2 Opcode postbyte	M1 object code	M2 object code	Instruction Mnemonic		Source Format for M1 (select 1 option in this col)	Source Format for M2 (select 1 option in this col)
1B 4q mb	xb	xb	MACS.*bwplbwpl*	Dd,	#*oprsxe4i*,	#*oprsxe4i*
	xb	xb			*Dj*,	*Dk*
	xb	xb			(*opru4*,*xys*),	(*opru4*,*xys*)
	xb	xb			(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+),	(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+)
	xb	xb			(*Dj*,*xys*),	(*Dk*,*xys*)
	xb	xb			[*Dj*,*xy*],	[*Dk*,*xy*]
	xb x1	xb x1			(*oprs9*,*xysp*),	(*oprs9*,*xysp*)
	xb x1	xb x1			[*oprs9*,*xysp*],	[*oprs9*,*xysp*]
	xb x1	xb x1			*opru14*,	*opru14*
	xb x2 x1	xb x2 x1			(*opru18*,*Dj*),	(*opru18*,*Dk*)
	xb x2 x1	xb x2 x1			*opru18*,	*opru18*
	xb x3 x2 x1	xb x3 x2 x1			(*opr24*,*xysp*),	(*opr24*,*xysp*)
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*,*xysp*],	[*opr24*,*xysp*]
	xb x3 x2 x1	xb x3 x2 x1			(*opru24*,*Dj*),	(*opru24*,*Dk*)
	xb x3 x2 x1	xb x3 x2 x1			*opr24*,	*opr24*
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*],	[*opr24*]

All combinations are valid although some, such as specifying a data register for both M1 and M2 can be done more efficiently using the REG-REG version of the instruction.

Instruction Fields

RESULT REGISTER- This field specifies the number of the data register D*d* used for the result (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SOURCE REGISTER or SOURCE 1 REGISTER - This field specifies the number of the data register D*j* used as an operand for the multiplication (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SOURCE 2 REGISTER - This field specifies the number of the data register D*k* used as the second operand for the multiplication (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMM_SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01) or 32-bit long-word (0b11) as the size of the divisor. The 0b10 combination is not available for the REG-IMM1/2/4 version of the instruction because those codes are used for the OPR-OPR version.

M1_SIZE and M2_SIZE - These fields specify the size of M1 and M2 which use the general OPR addressing mode to specify short-immediate, register, or memory operands (0b00 = 8-bit byte, 0b01 = 16-bit word, 0b10 = 24-bit pointer, and 0b11 = 32-bit long-word). When a short-immediate operand is specified, it is internally sign-extended to the size specified by the M1_SIZE and/or M2_SIZE specifications. When a register is specified, it determines the size and the M1_SIZE and/or M2_SIZE specifications are ignored.

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size specified by IMM_SIZE.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Using OPR addressing mode to specify a register operand for both source operands, is less efficient than using the REG-REG version of the instruction.

MACU Unsigned Multiply and Accumulate MACU

Operation

(D*j*) × (D*k*) + (D*d*) ⇒ D*d*

(D*j*) × IMM + (D*d*) ⇒ D*d*

(D*j*) × (M) + (D*d*) ⇒ D*d*

(M1) × (M2) + (D*d*) ⇒ D*d*

Syntax Variations	Addressing Modes
`MACU Dd,Dj,Dk`	**REG-REG**
`MACU.B Dd,Dj,#opr8i`	**REG-IMM1**
`MACU.W Dd,Dj,#opr16i`	**REG-IMM2**
`MACU.L Dd,Dj,#opr32i`	**REG-IMM4**
`MACU.bwl Dd,Dj,oprmemreg`	**REG-OPR/1/2/3**
`MACU.bwplbwplDd,oprmemreg,oprmemreg`	**OPR/1/2/3-OPR/1/2/3**

Description

Multiplies two unsigned operands, adds this product to a register D*d*, and stores the accumulated result to register D*d*. The first source operand may be a register D*j* or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M1. The second source operand may be a register D*k*, an 8-bit, 16-bit , or 32-bit immediate value, or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M or M2. Both source operands and the result are interpreted as unsigned values.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Set if the unsigned result of the multiply operation does not fit in the result register D*d* or if there is an overflow from the addition. Cleared otherwise.

C: Set if there is a carry from the addition. Cleared otherwise.

Detailed Instruction Formats

REG-REG

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	1	0	0	1	RESULT REGISTER D*d*			4q
0	0	SOURCE 1 REGISTER D*j*			SOURCE 2 REGISTER D*k*			mb

```
1B 4q mb                    MACU      Dd,Dj,Dk
```

REG-IMM1/2/4

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	1	0	0	1	RESULT REGISTER D*d*			4q
0	1	SOURCE REGISTER D*j*			1	IMM_SIZE (.B, .W, –, .L)		mb
IMMEDIATE DATA								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								

```
1B 4q mb i1                 MACU.B    Dd,Dj,#opr8i
1B 4q mb i2 i1              MACU.W    Dd,Dj,#opr16i ;short-imm better for some values
1B 4q mb i4 i3 i2 i1        MACU.L    Dd,Dj,#opr32i ;short-imm better for some values
```

REG-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	1	0	0	1	RESULT REGISTER D*d*			4q
0	1	SOURCE REGISTER D*j*				M2_SIZE (.B, .W, –, .L)		mb
OPR POSTBYTE (for M2)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 4q mb xb                 MACU.bwl  Dd,Dj,#oprsxe4i
1B 4q mb xb                 MACU.bwl  Dd,Dj,Dk ;see more efficient REG-REG version
1B 4q mb xb                 MACU.bwl  Dd,Dj,(opru4,xys)
1B 4q mb xb                 MACU.bwl  Dd,Dj,{(+-xy)|(xy+-)|(-s)|(s+)}
1B 4q mb xb                 MACU.bwl  Dd,Dj,(Di,xys)
1B 4q mb xb                 MACU.bwl  Dd,Dj,[Di,xy]
1B 4q mb xb x1              MACU.bwl  Dd,Dj,(oprs9,xysp)
1B 4q mb xb x1              MACU.bwl  Dd,Dj,[oprs9,xysp]
1B 4q mb xb x1              MACU.bwl  Dd,Dj,opru14
1B 4q mb xb x2 x1           MACU.bwl  Dd,Dj,(opru18,Di)
1B 4q mb xb x2 x1           MACU.bwl  Dd,Dj,opru18
1B 4q mb xb x3 x2 x1        MACU.bwl  Dd,Dj,(opr24,xysp)
1B 4q mb xb x3 x2 x1        MACU.bwl  Dd,Dj,[opr24,xysp]
1B 4q mb xb x3 x2 x1        MACU.bwl  Dd,Dj,(opru24,Di)
1B 4q mb xb x3 x2 x1        MACU.bwl  Dd,Dj,opr24
1B 4q mb xb x3 x2 x1        MACU.bwl  Dd,Dj,[opr24]
```

OPR/1/2/3-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	1	0	0	1	RESULT REGISTER D*d*			4q
0	1	M1_SIZE (.B, .W, .P, .L)		M2_SIZE (.B, .W, .P, .L)		1	0	mb
OPR POSTBYTE (for M1)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
OPR POSTBYTE (for M2)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

page 2 Opcode postbyte	M1 object code	M2 object code	Instruction Mnemonic		Source Format for M1 (select 1 option in this col)	Source Format for M2 (select 1 option in this col)
1B 4q mb	xb	xb	MACU.*bwplbwpl*	Dd,	#*oprsxe4i*,	#*oprsxe4i*
	xb	xb			*Dj*,	*Dk*
	xb	xb			(*opru4*,*xys*),	(*opru4*,*xys*)
	xb	xb			(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+),	(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+)
	xb	xb			(*Dj*,*xys*),	(*Dk*,*xys*)
	xb	xb			[*Dj*,*xy*],	[*Dk*,*xy*]
	xb x1	xb x1			(*oprs9*,*xysp*),	(*oprs9*,*xysp*)
	xb x1	xb x1			[*oprs9*,*xysp*],	[*oprs9*,*xysp*]
	xb x1	xb x1			*opru14*,	*opru14*
	xb x2 x1	xb x2 x1			(*opru18*,*Dj*),	(*opru18*,*Dk*)
	xb x2 x1	xb x2 x1			*opru18*,	*opru18*
	xb x3 x2 x1	xb x3 x2 x1			(*opr24*,*xysp*),	(*opr24*,*xysp*)
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*,*xysp*],	[*opr24*,*xysp*]
	xb x3 x2 x1	xb x3 x2 x1			(*opru24*,*Dj*),	(*opru24*,*Dk*)
	xb x3 x2 x1	xb x3 x2 x1			*opr24*,	*opr24*
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*],	[*opr24*]

All combinations are valid although some, such as specifying a data register for both M1 and M2 can be done more efficiently using the REG-REG version of the instruction.

Instruction Fields

RESULT REGISTER- This field specifies the number of the data register D*d* used for the result (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SOURCE REGISTER or SOURCE 1 REGISTER - This field specifies the number of the data register D*j* used as an operand for the multiplication (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SOURCE 2 REGISTER - This field specifies the number of the data register D*k* used as the second operand for the multiplication (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMM_SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01) or 32-bit long-word (0b11) as the size of the divisor. The 0b10 combination is not available for the REG-IMM1/2/4 version of the instruction because those codes are used for the OPR-OPR version.

M1_SIZE and M2_SIZE - These fields specify the size of M1 and M2 which use the general OPR addressing mode to specify short-immediate, register, or memory operands (0b00 = 8-bit byte, 0b01 = 16-bit word, 0b10 = 24-bit pointer, and 0b11 = 32-bit long-word). When a short-immediate operand is specified, it is internally sign-extended to the size specified by the M1_SIZE and/or M2_SIZE specifications. When a register is specified, it determines the size and the M1_SIZE and/or M2_SIZE specifications are ignored.

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size specified by IMM_SIZE.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Using OPR addressing mode to specify a register operand for both source operands, is less efficient than using the REG-REG version of the instruction.

MAXS — Maximum of Two Signed Values to D*i* — MAXS

Operation

MAX((D*i*), (M)) ⇒ D*i*

Syntax Variations	Addressing Modes
`MAXS Di,oprmemreg`	**OPR/1/2/3**

Description

Subtracts the signed value of memory operand M from the signed value in register D*i* to determine which is larger. The larger of the two values is stored in register D*i*. The size of memory operand M is determined by the size of register D*i*.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result of the subtract operation is set. Cleared otherwise.

Z: Set if the result of the subtract operation is zero. Cleared otherwise.

V: Set if there is a two's complement overflow as a result of the subtract operation. Cleared otherwise.

C: Set if the subtract operation requires a borrow. Cleared otherwise.

Detailed Instruction Formats

OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	0	1	SD REGISTER D*i*			2q
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 2q xb                MAXS    Di,#oprsxe4i ;-1, +1, 2, 3...14, 15
1B 2q xb                MAXS    Di,Dj
1B 2q xb                MAXS    Di,(opru4,xys)
1B 2q xb                MAXS    Di,{(+-xy)|(xy+-)|(-s)|(s+)}
1B 2q xb                MAXS    Di,(Dj,xys)
1B 2q xb                MAXS    Di,[Dj,xy]
1B 2q xb x1             MAXS    Di,(oprs9,xysp)
1B 2q xb x1             MAXS    Di,[oprs9,xysp]
1B 2q xb x1             MAXS    Di,opru14
1B 2q xb x2 x1          MAXS    Di,(opru18,Dj)
```

```
1B 2q xb x2 x1                    MAXS       Di,opru18
1B 2q xb x3 x2 x1                 MAXS       Di,(opr24,xysp)
1B 2q xb x3 x2 x1                 MAXS       Di,[opr24,xysp]
1B 2q xb x3 x2 x1                 MAXS       Di,(opru24,Dj)
1B 2q xb x3 x2 x1                 MAXS       Di,opr24
1B 2q xb x3 x2 x1                 MAXS       Di,[opr24]
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as a source operand and for the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

MAXU Maximum of Two Unsigned Values to D*i* MAXU

Operation

MAX((D*i*), (M)) ⇒ D*i*

Syntax Variations	Addressing Modes
`MAXU Di,oprmemreg`	**OPR/1/2/3**

Description

Subtracts the unsigned value of memory operand M from the unsigned value in register D*i* to determine which is larger. The larger of the two values is stored in register D*i*. The size of memory operand M is determined by the size of register D*i*.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result of the subtract operation is set. Cleared otherwise.

Z: Set if the result of the subtract operation is zero. Cleared otherwise.

V: Set if there is a two's complement overflow as a result of the subtract operation. Cleared otherwise.

C: Set if the subtract operation requires a borrow. Cleared otherwise.

Detailed Instruction Formats

OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	1	1	SD REGISTER D*i*			1q
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 1q xb                MAXU    Di,#oprsxe4i ;-1, +1, 2, 3...14, 15
1B 1q xb                MAXU    Di,Dj
1B 1q xb                MAXU    Di,(opru4,xys)
1B 1q xb                MAXU    Di,{(+-xy)|(xy+-)|(-s)|(s+)}
1B 1q xb                MAXU    Di,(Dj,xys)
1B 1q xb                MAXU    Di,[Dj,xy]
1B 1q xb x1             MAXU    Di,(oprs9,xysp)
1B 1q xb x1             MAXU    Di,[oprs9,xysp]
1B 1q xb x1             MAXU    Di,opru14
1B 1q xb x2 x1          MAXU    Di,(opru18,Dj)
```

```
1B 1q xb x2 x1                    MAXU       Di,opru18
1B 1q xb x3 x2 x1                 MAXU       Di,(opr24,xysp)
1B 1q xb x3 x2 x1                 MAXU       Di,[opr24,xysp]
1B 1q xb x3 x2 x1                 MAXU       Di,(opru24,Dj)
1B 1q xb x3 x2 x1                 MAXU       Di,opr24
1B 1q xb x3 x2 x1                 MAXU       Di,[opr24]
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as a source operand and for the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

MINS — Minimum of Two Signed Values to D*i* — MINS

Operation

MIN((D*i*), (M)) ⇒ D*i*

Syntax Variations	Addressing Modes
`MINS Di,oprmemreg`	**OPR/1/2/3**

Description

Subtracts the signed value of memory operand M from the signed value in register D*i* to determine which is smaller. The smaller of the two values is stored in register D*i*. The size of memory operand M is determined by the size of register D*i*.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result of the subtract operation is set. Cleared otherwise.

Z: Set if the result of the subtract operation is zero. Cleared otherwise.

V: Set if there is a two's complement overflow as a result of the subtract operation. Cleared otherwise.

C: Set if the subtract operation requires a borrow. Cleared otherwise.

Detailed Instruction Formats

OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	0	0	SD REGISTER D*i*			2n
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 2n xb                MINS    Di,#oprsxe4i ;-1, +1, 2, 3...14, 15
1B 2n xb                MINS    Di,Dj
1B 2n xb                MINS    Di,(opru4,xys)
1B 2n xb                MINS    Di,{(+-xy)|(xy+-)|(-s)|(s+)}
1B 2n xb                MINS    Di,(Dj,xys)
1B 2n xb                MINS    Di,[Dj,xy]
1B 2n xb x1             MINS    Di,(oprs9,xysp)
1B 2n xb x1             MINS    Di,[oprs9,xysp]
1B 2n xb x1             MINS    Di,opru14
1B 2n xb x2 x1          MINS    Di,(opru18,Dj)
```

```
1B 2n xb x2 x1                    MINS       Di,opru18
1B 2n xb x3 x2 x1                 MINS       Di,(opr24,xysp)
1B 2n xb x3 x2 x1                 MINS       Di,[opr24,xysp]
1B 2n xb x3 x2 x1                 MINS       Di,(opru24,Dj)
1B 2n xb x3 x2 x1                 MINS       Di,opr24
1B 2n xb x3 x2 x1                 MINS       Di,[opr24]
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as a source operand and for the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

MINU — Minimum of Two Unsigned Values to D*i* — MINU

Operation

MIN((D*i*), (M)) ⇒ D*i*

Syntax Variations	Addressing Modes
`MINU Di,oprmemreg`	**OPR/1/2/3**

Description

Subtracts the unsigned value of memory operand M from the unsigned value in register D*i* to determine which is smaller. The smaller of the two values is stored in register D*i*. The size of memory operand M is determined by the size of register D*i*.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result of the subtract operation is set. Cleared otherwise.

Z: Set if the result of the subtract operation is zero. Cleared otherwise.

V: Set if there is a two's complement overflow as a result of the subtract operation. Cleared otherwise.

C: Set if the subtract operation requires a borrow. Cleared otherwise.

Detailed Instruction Formats

OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	1	0	SD REGISTER D*i*			1n
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 1n xb                MINU    Di,#oprsxe4i ;-1, +1, 2, 3...14, 15
1B 1n xb                MINU    Di,Dj
1B 1n xb                MINU    Di,(opru4,xys)
1B 1n xb                MINU    Di,{(+-xy)|(xy+-)|(-s)|(s+)}
1B 1n xb                MINU    Di,(Dj,xys)
1B 1n xb                MINU    Di,[Dj,xy]
1B 1n xb x1             MINU    Di,(oprs9,xysp)
1B 1n xb x1             MINU    Di,[oprs9,xysp]
1B 1n xb x1             MINU    Di,opru14
1B 1n xb x2 x1          MINU    Di,(opru18,Dj)
```

```
1B 1n xb x2 x1                    MINU       Di,opru18
1B 1n xb x3 x2 x1                 MINU       Di,(opr24,xysp)
1B 1n xb x3 x2 x1                 MINU       Di,[opr24,xysp]
1B 1n xb x3 x2 x1                 MINU       Di,(opru24,Dj)
1B 1n xb x3 x2 x1                 MINU       Di,opr24
1B 1n xb x3 x2 x1                 MINU       Di,[opr24]
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as a source operand and for the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

MODS — Signed Modulo — MODS

Operation

(D*j*) % (D*k*) ⇒ D*d*

(D*j*) % IMM ⇒ D*d*

(D*j*) % (M) ⇒ D*d*

(M1) % (M2) ⇒ D*d*

Syntax Variations	Addressing Modes
`MODS Dd,Dj,Dk`	**REG-REG**
`MODS.B Dd,Dj,#opr8i`	**REG-IMM1**
`MODS.W Dd,Dj,#opr16i`	**REG-IMM2**
`MODS.L Dd,Dj,#opr32i`	**REG-IMM4**
`MODS.bwl Dd,Dj,oprmemreg`	**REG-OPR/1/2/3**
`MODS.bwplbwplDd,oprmemreg,oprmemreg`	**OPR/1/2/3-OPR/1/2/3**

Description

Divides a signed two's complement dividend by a signed two's complement divisor to produce a signed two's complement remainder in a register D*d*. The dividend may be a register D*j* or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M1. The divisor may be a register D*k*, an 8-bit, 16-bit, or 32-bit immediate value, or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M or M2. To ensure compatibility with the C standard requirement that a = (a/b)*b + (a % b), the sign of the result (remainder) is the same as the sign of the dividend.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result is set. Undefined after overflow or division by zero. Cleared otherwise.

Z: Set if the result is zero. Undefined after overflow or division by zero. Cleared otherwise.

V: Set if the signed remainder does not fit in the result register D*d*. Undefined after division by zero. Cleared otherwise.

C: Set if divisor was zero. Cleared otherwise. (Indicates division by zero).

Detailed Instruction Formats

REG-REG

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	1	1	RESULT REGISTER D*d*			3q
1	0	DIVIDEND REGISTER D*j*			DIVISOR REGISTER D*k*			mb

```
1B 3q mb                    MODS      Dd,Dj,Dk
```

REG-IMM1/2/4

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	1	1	RESULT REGISTER D*d*			3q
1	1	DIVIDEND REGISTER D*j*			1	IMM_SIZE (.B, .W, –, .L)		mb
IMMEDIATE DATA (Divisor)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								

```
1B 3q mb i1                 MODS.B    Dd,Dj,#opr8i
1B 3q mb i2 i1              MODS.W    Dd,Dj,#opr16i ;short-imm better for some values
1B 3q mb i4 i3 i2 i1        MODS.L    Dd,Dj,#opr32i ;short-imm better for some values
```

REG-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	1	1	RESULT REGISTER D*d*			3q
1	1	DIVIDEND REGISTER D*j*				M2_SIZE (.B, .W, –, .L)		mb
OPR POSTBYTE (for M2 divisor)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 3q mb xb                 MODS.bwl  Dd,Dj,#oprsxe4i
1B 3q mb xb                 MODS.bwl  Dd,Dj,Dk ;see more efficient REG-REG version
1B 3q mb xb                 MODS.bwl  Dd,Dj,(opru4,xys)
1B 3q mb xb                 MODS.bwl  Dd,Dj,{(+-xy)|(xy+-)|(-s)|(s+)}
1B 3q mb xb                 MODS.bwl  Dd,Dj,(Di,xys)
1B 3q mb xb                 MODS.bwl  Dd,Dj,[Di,xy]
1B 3q mb xb x1              MODS.bwl  Dd,Dj,(oprs9,xysp)
1B 3q mb xb x1              MODS.bwl  Dd,Dj,[oprs9,xysp]
1B 3q mb xb x1              MODS.bwl  Dd,Dj,opru14
1B 3q mb xb x2 x1           MODS.bwl  Dd,Dj,(opru18,Di)
1B 3q mb xb x2 x1           MODS.bwl  Dd,Dj,opru18
1B 3q mb xb x3 x2 x1        MODS.bwl  Dd,Dj,(opr24,xysp)
1B 3q mb xb x3 x2 x1        MODS.bwl  Dd,Dj,[opr24,xysp]
1B 3q mb xb x3 x2 x1        MODS.bwl  Dd,Dj,(opru24,Di)
1B 3q mb xb x3 x2 x1        MODS.bwl  Dd,Dj,opr24
1B 3q mb xb x3 x2 x1        MODS.bwl  Dd,Dj,[opr24]
```

OPR/1/2/3-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	1	1	RESULT REGISTER D*d*			3q
1	1	M1_SIZE (.B, .W, .P, .L)		M2_SIZE (.B, .W, .P, .L)		1	0	mb
OPR POSTBYTE (for M1 dividend)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
OPR POSTBYTE (for M2 divisor)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

page 2 Opcode postbyte	M1 object code	M2 object code	Instruction Mnemonic		Source Format for M1 (Dividend) (select 1 option in this col)	Source Format for M2 (Divisor) (select 1 option in this col)
1B 3q mb	xb	xb	MODS.*bwplbwpl*	Dd,	#*oprsxe4i*,	#*oprsxe4i*
	xb	xb			*Dj*,	*Dk*
	xb	xb			(*opru4*,*xys*),	(*opru4*,*xys*)
	xb	xb			(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+),	(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+)
	xb	xb			(*Dj*,*xys*),	(*Dk*,*xys*)
	xb	xb			[*Dj*,*xy*],	[*Dk*,*xy*]
	xb x1	xb x1			(*oprs9*,*xysp*),	(*oprs9*,*xysp*)
	xb x1	xb x1			[*oprs9*,*xysp*],	[*oprs9*,*xysp*]
	xb x1	xb x1			*opru14*,	*opru14*
	xb x2 x1	xb x2 x1			(*opru18*,*Dj*),	(*opru18*,*Dk*)
	xb x2 x1	xb x2 x1			*opru18*,	*opru18*
	xb x3 x2 x1	xb x3 x2 x1			(*opr24*,*xysp*),	(*opr24*,*xysp*)
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*,*xysp*],	[*opr24*,*xysp*]
	xb x3 x2 x1	xb x3 x2 x1			(*opru24*,*Dj*),	(*opru24*,*Dk*)
	xb x3 x2 x1	xb x3 x2 x1			*opr24*,	*opr24*
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*],	[*opr24*]

All combinations are valid although some, such as specifying a data register for both M1 and M2 can be done more efficiently using the REG-REG version of the instruction.

Instruction Fields

RESULT REGISTER- This field specifies the number of the data register D*d* used for the result (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

DIVIDEND REGISTER - This field specifies the number of the data register D*j* used as dividend (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

DIVISOR REGISTER - This field specifies the number of the data register D*k* used as divisor (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMM_SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01) or 32-bit long-word (0b11) as the size of the divisor. The 0b10 combination is not available for the REG-IMM1/2/4 version of the instruction because those codes are used for the OPR-OPR version.

M1_SIZE and M2_SIZE - These fields specify the size of M1 (dividend) and M2 (divisor) which use the general OPR addressing mode to specify short-immediate, register, or memory operands (0b00 = 8-bit byte, 0b01 = 16-bit word, 0b10 = 24-bit pointer, and 0b11 = 32-bit long-word). When a short-immediate operand is specified, it is internally sign-extended to the size specified by the M1_SIZE and/or M2_SIZE specifications. When a register is specified, it determines the size and the M1_SIZE and/or M2_SIZE specifications are ignored.

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size specified by IMM_SIZE.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Using OPR addressing mode to specify a register operand for both the dividend and the divisor, is less efficient than using the REG-REG version of the instruction.

MODU Unsigned Modulo MODU

Operation

(D*j*) % (D*k*) ⇒ D*d*

(D*j*) % IMM ⇒ D*d*

(D*j*) % (M) ⇒ D*d*

(M1) % (M2) ⇒ D*d*

Syntax Variations	Addressing Modes
`MODU Dd,Dj,Dk`	**REG-REG**
`MODU.B Dd,Dj,#opr8i`	**REG-IMM1**
`MODU.W Dd,Dj,#opr16i`	**REG-IMM2**
`MODU.L Dd,Dj,#opr32i`	**REG-IMM4**
`MODU.bwl Dd,Dj,oprmemreg`	**REG-OPR/1/2/3**
`MODU.bwplbwplDd,oprmemreg,oprmemreg`	**OPR/1/2/3-OPR/1/2/3**

Description

Divides an unsigned dividend by an unsigned divisor to produce an unsigned remainder in a register D*d*. The dividend may be a register D*j* or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M1. The divisor may be a register D*k*, an 8-bit, 16-bit, or 32-bit immediate value, or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M or M2.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result is set. Undefined after overflow or division by zero. Cleared otherwise.

Z: Set if the result is zero. Undefined after overflow or division by zero. Cleared otherwise.

V: Set if the unsigned remainder does not fit in the result register D*d*. Undefined after division by zero. Cleared otherwise.

C: Set if divisor was zero. Cleared otherwise. (Indicates division by zero).

Detailed Instruction Formats

REG-REG

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	1	1	RESULT REGISTER D*d*			3q
0	0	DIVIDEND REGISTER D*j*			DIVISOR REGISTER D*k*			mb

```
1B 3q mb                    MODU        Dd,Dj,Dk
```

REG-IMM1/2/4

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	1	1	RESULT REGISTER D*d*			3q
0	1	DIVIDEND REGISTER D*j*			1	IMM_SIZE (.B, .W, –, .L)		mb
IMMEDIATE DATA (divisor)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								

```
1B 3q mb i1                 MODU.B      Dd,Dj,#opr8i
1B 3q mb i2 i1              MODU.W      Dd,Dj,#opr16i ;short-imm better for some values
1B 3q mb i4 i3 i2 i1        MODU.L      Dd,Dj,#opr32i ;short-imm better for some values
```

REG-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	1	1	RESULT REGISTER D*d*			3q
0	1	DIVIDEND REGISTER D*j*				M2_SIZE (.B, .W, –, .L)		mb
OPR POSTBYTE (for M2 divisor)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 3q mb xb                 MODU.bwl    Dd,Dj,#oprsxe4i
1B 3q mb xb                 MODU.bwl    Dd,Dj,Dk ;see more efficient REG-REG version
1B 3q mb xb                 MODU.bwl    Dd,Dj,(opru4,xys)
1B 3q mb xb                 MODU.bwl    Dd,Dj,{(+-xy)|(xy+-)|(-s)|(s+)}
1B 3q mb xb                 MODU.bwl    Dd,Dj,(Di,xys)
1B 3q mb xb                 MODU.bwl    Dd,Dj,[Di,xy]
1B 3q mb xb x1              MODU.bwl    Dd,Dj,(oprs9,xysp)
1B 3q mb xb x1              MODU.bwl    Dd,Dj,[oprs9,xysp]
1B 3q mb xb x1              MODU.bwl    Dd,Dj,opru14
1B 3q mb xb x2 x1           MODU.bwl    Dd,Dj,(opru18,Di)
1B 3q mb xb x2 x1           MODU.bwl    Dd,Dj,opru18
1B 3q mb xb x3 x2 x1        MODU.bwl    Dd,Dj,(opr24,xysp)
1B 3q mb xb x3 x2 x1        MODU.bwl    Dd,Dj,[opr24,xysp]
1B 3q mb xb x3 x2 x1        MODU.bwl    Dd,Dj,(opru24,Di)
1B 3q mb xb x3 x2 x1        MODU.bwl    Dd,Dj,opr24
1B 3q mb xb x3 x2 x1        MODU.bwl    Dd,Dj,[opr24]
```

OPR/1/2/3-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	1	1	1	RESULT REGISTER D*d*			3q
0	1	M1_SIZE (.B, .W, .P, .L)		M2_SIZE (.B, .W, .P, .L)		1	0	mb
OPR POSTBYTE (for M1 dividend)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
OPR POSTBYTE (for M2 divisor)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

page 2 Opcode postbyte	M1 object code	M2 object code	Instruction Mnemonic		Source Format for M1 (Dividend) (select 1 option in this col)	Source Format for M2 (Divisor) (select 1 option in this col)
1B 3q mb	xb	xb	MODU.*bwplbwpl*	Dd,	#*oprsxe4i*,	#*oprsxe4i*
	xb	xb			*Dj*,	*Dk*
	xb	xb			(*opru4*,*xys*),	(*opru4*,*xys*)
	xb	xb			(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+),	(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+)
	xb	xb			(*Dj*,*xys*),	(*Dk*,*xys*)
	xb	xb			[*Dj*,*xy*],	[*Dk*,*xy*]
	xb x1	xb x1			(*oprs9*,*xysp*),	(*oprs9*,*xysp*)
	xb x1	xb x1			[*oprs9*,*xysp*],	[*oprs9*,*xysp*]
	xb x1	xb x1			*opru14*,	*opru14*
	xb x2 x1	xb x2 x1			(*opru18*,*Dj*),	(*opru18*,*Dk*)
	xb x2 x1	xb x2 x1			*opru18*,	*opru18*
	xb x3 x2 x1	xb x3 x2 x1			(*opr24*,*xysp*),	(*opr24*,*xysp*)
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*,*xysp*],	[*opr24*,*xysp*]
	xb x3 x2 x1	xb x3 x2 x1			(*opru24*,*Dj*),	(*opru24*,*Dk*)
	xb x3 x2 x1	xb x3 x2 x1			*opr24*,	*opr24*
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*],	[*opr24*]

All combinations are valid although some, such as specifying a data register for both M1 and M2 can be done more efficiently using the REG-REG version of the instruction.

Instruction Fields

RESULT REGISTER- This field specifies the number of the data register D*d* used for the result (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

DIVIDEND REGISTER - This field specifies the number of the data register D*j* used as dividend (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

DIVISOR REGISTER - This field specifies the number of the data register D*k* used as divisor (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMM_SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01) or 32-bit long-word (0b11) as the size of the divisor. The 0b10 combination is not available for the REG-IMM1/2/4 version of the instruction because those codes are used for the OPR-OPR version.

M1_SIZE and M2_SIZE - These fields specify the size of M1 (dividend) and M2 (divisor) which use the general OPR addressing mode to specify short-immediate, register, or memory operands (0b00 = 8-bit byte, 0b01 = 16-bit word, 0b10 = 24-bit pointer, and 0b11 = 32-bit long-word). When a short-immediate operand is specified, it is internally sign-extended to the size specified by the M1_SIZE and/or M2_SIZE specifications. When a register is specified, it determines the size and the M1_SIZE and/or M2_SIZE specifications are ignored.

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size specified by IMM_SIZE.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Using OPR addressing mode to specify a register operand for both the dividend and the divisor, is less efficient than using the REG-REG version of the instruction.

MOV

Move Data

(8, 16, 24, or 32-bits; IMM-OPR or OPR-OPR)

MOV

Operation

(M1) ⇒ M2

Syntax Variations	Addressing Modes
`MOV.B #opr8i,oprmemreg`	**IMM1-OPR/1/2/3**
`MOV.W #opr16i,oprmemreg`	**IMM2-OPR/1/2/3**
`MOV.P #opr24i,oprmemreg`	**IMM3-OPR/1/2/3**
`MOV.L #opr32i,oprmemreg`	**IMM4-OPR/1/2/3**
`MOV.bwpl oprmemreg,oprmemreg`	**OPR/1/2/3-OPR/1/2/3**

Description

Move (copy) an 8-bit, 16-bit, 24-bit, or 32-bit immediate value to a memory location of the same size (or a register D*i*), or move (copy) 8-bits, 16-bits, 24-bits, or 32-bits from one memory location (or register D*i*) to another memory location of the same size (or register D*j*). The size of the operation is normally specified by the dot suffix B, W, P, or L on the MOV instruction.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

IMM1-OPR/1/2/3 (.B 8-bit byte)

7	6	5	4	3	2	1	0	
0	0	0	0	1	1	0	0	0C
IMMEDIATE DATA (source)								i1
OPR POSTBYTE (destination)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
0C i1 xb                    MOV.B    #opr8i,#oprsxe4i ;not appropriate as destination
0C i1 xb                    MOV.B    #opr8i,Di ;consider using LD Di,#
0C i1 xb                    MOV.B    #opr8i,(opru4,xys)
0C i1 xb                    MOV.B    #opr8i,{(+-xy)|(xy+-)|(-s)|(s+)}
0C i1 xb                    MOV.B    #opr8i,[Di,xy])
0C i1 xb                    MOV.B    #opr8i,(Di,xys)
0C i1 xb x1                 MOV.B    #opr8i,(oprs9,xysp)
0C i1 xb x1                 MOV.B    #opr8i,[oprs9,xysp]
0C i1 xb x1                 MOV.B    #opr8i,opru14
0C i1 xb x2 x1              MOV.B    #opr8i,(opru18,Di)
0C i1 xb x2 x1              MOV.B    #opr8i,opru18
0C i1 xb x3 x2 x1           MOV.B    #opr8i,(opr24,xysp)
0C i1 xb x3 x2 x1           MOV.B    #opr8i,[opr24,xysp]
0C i1 xb x3 x2 x1           MOV.B    #opr8i,(opru24,Di)
0C i1 xb x3 x2 x1           MOV.B    #opr8i,opr24
0C i1 xb x3 x2 x1           MOV.B    #opr8i,[opr24]
```

IMM2-OPR/1/2/3 (.W 16-bit word)

7	6	5	4	3	2	1	0	
0	0	0	0	1	1	0	1	0D
IMMEDIATE DATA (source)								i2
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE)								i1
OPR POSTBYTE (destination)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
0D i2 i1 xb                 MOV.W    #opr16i,#oprsxe4i ;not appropriate as destination
0D i2 i1 xb                 MOV.W    #opr16i,Di ;consider using LD Di,#
0D i2 i1 xb                 MOV.W    #opr16i,(opru4,xys)
0D i2 i1 xb                 MOV.W    #opr16i,{(+-xy)|(xy+-)|(-s)|(s+)}
0D i2 i1 xb                 MOV.W    #opr16i,(Di,xys)
0D i2 i1 xb                 MOV.W    #opr16i,[Di,xy]
0D i2 i1 xb x1              MOV.W    #opr16i,(oprs9,xysp)
0D i2 i1 xb x1              MOV.W    #opr16i,[oprs9,xysp]
0D i2 i1 xb x1              MOV.W    #opr16i,opru14
0D i2 i1 xb x2 x1           MOV.W    #opr16i,(opru18,Di)
0D i2 i1 xb x2 x1           MOV.W    #opr16i,opru18
0D i2 i1 xb x3 x2 x1        MOV.W    #opr16i,(opr24,xysp)
0D i2 i1 xb x3 x2 x1        MOV.W    #opr16i,[opr24,xysp]
0D i2 i1 xb x3 x2 x1        MOV.W    #opr16i,(opru24,Di)
0D i2 i1 xb x3 x2 x1        MOV.W    #opr16i,opr24
0D i2 i1 xb x3 x2 x1        MOV.W    #opr16i,[opr24]
```

IMM3-OPR/1/2/3 (.P 24-bit pointer)

7	6	5	4	3	2	1	0	
0	0	0	0	1	1	1	0	0E
IMMEDIATE DATA (source)								i3
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE)								i2
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE)								i1
OPR POSTBYTE (destination)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
0E i3 i2 i1 xb                MOV.P    #opr24i,#oprsxe4i ;not appropriate as destination
0E i3 i2 i1 xb                MOV.P    #opr24i,Di ;consider using LD Di,#
0E i3 i2 i1 xb                MOV.P    #opr24i,(opru4,xys)
0E i3 i2 i1 xb                MOV.P    #opr24i,{(+-xy)|(xy+-)|(-s)|(s+)}
0E i3 i2 i1 xb                MOV.P    #opr24i,(Di,xys)
0E i3 i2 i1 xb                MOV.P    #opr24i,[Di,xy]
0E i3 i2 i1 xb x1             MOV.P    #opr24i,(oprs9,xysp)
0E i3 i2 i1 xb x1             MOV.P    #opr24i,[oprs9,xysp]
0E i3 i2 i1 xb x1             MOV.P    #opr24i,opru14
0E i3 i2 i1 xb x2 x1          MOV.P    #opr24i,(opru18,Di)
0E i3 i2 i1 xb x2 x1          MOV.P    #opr24i,opru18
0E i3 i2 i1 xb x3 x2 x1       MOV.P    #opr24i,(opr24,xysp)
0E i3 i2 i1 xb x3 x2 x1       MOV.P    #opr24i,[opr24,xysp]
0E i3 i2 i1 xb x3 x2 x1       MOV.P    #opr24i,(opru24,Di)
0E i3 i2 i1 xb x3 x2 x1       MOV.P    #opr24i,opr24
0E i3 i2 i1 xb x3 x2 x1       MOV.P    #opr24i,[opr24]
```

IMM4-OPR/1/2/3 (.L 32-bit long-word)

7	6	5	4	3	2	1	0	
0	0	0	0	1	1	1	1	0F
IMMEDIATE DATA (source)								i4
IMMEDIATE DATA[23:16]								i3
IMMEDIATE DATA[15:8]								i2
IMMEDIATE DATA[7:0]								i1
OPR POSTBYTE (destination)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
0F i4 i3 i2 i1 xb             MOV.L    #opr32i,#oprsxe4i ;not appropriate as destination
0F i4 i3 i2 i1 xb             MOV.L    #opr32i,Di ;consider using LD Di,#
0F i4 i3 i2 i1 xb             MOV.L    #opr32i,(opru4,xys)
0F i4 i3 i2 i1 xb             MOV.L    #opr32i,{(+-xy)|(xy+-)|(-s)|(s+)}
0F i4 i3 i2 i1 xb             MOV.L    #opr32i,(Di,xys)
0F i4 i3 i2 i1 xb             MOV.L    #opr32i,[Di,xy]
0F i4 i3 i2 i1 xb x1          MOV.L    #opr32i,(oprs9,xysp)
0F i4 i3 i2 i1 xb x1          MOV.L    #opr32i,[oprs9,xysp]
0F i4 i3 i2 i1 xb x1          MOV.L    #opr32i,opru14
0F i4 i3 i2 i1 xb x2 x1       MOV.L    #opr32i,(opru18,Di)
0F i4 i3 i2 i1 xb x2 x1       MOV.L    #opr32i,opru18
0F i4 i3 i2 i1 xb x3 x2 x1    MOV.L    #opr32i,(opr24,xysp)
0F i4 i3 i2 i1 xb x3 x2 x1    MOV.L    #opr32i,[opr24,xysp]
0F i4 i3 i2 i1 xb x3 x2 x1    MOV.L    #opr32i,(opru24,Di)
```

```
0F i4 i3 i2 i1 xb x3 x2 x1        MOV.L       #opr32i,opr24
0F i4 i3 i2 i1 xb x3 x2 x1        MOV.L       #opr32i,[opr24]
```

OPR/1/2/3-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	1	SIZE (.B, .W, .P, .L)		1p
OPR POSTBYTE (for source)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
OPR POSTBYTE (for destination)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

Opcode	M1 object code	M2 object code	Instruction Mnemonic	Source Format for M1 (Source) (select 1 option in this col)	Source Format for M2 (Destination) (select 1 option in this col)
1p	xb	xb	MOV.*bwpl*	#*oprsxe4i*,	#*oprsxe4i*
	xb	xb		*Dj*,	*Dk*
	xb	xb		(*opru4*,*xys*),	(*opru4*,*xys*)
	xb	xb		(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+),	(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+)
	xb	xb		(*Dj*,*xys*),	(*Dk*,*xys*)
	xb	xb		[*Dj*,*xy*],	[*Dk*,*xy*]
	xb x1	xb x1		(*oprs9*,*xysp*),	(*oprs9*,*xysp*)
	xb x1	xb x1		[*oprs9*,*xysp*],	[*oprs9*,*xysp*]
	xb x1	xb x1		*opru14*,	*opru14*
	xb x2 x1	xb x2 x1		(*opru18*,*Dj*),	(*opru18*,*Dk*)
	xb x2 x1	xb x2 x1		*opru18*,	*opru18*
	xb x3 x2 x1	xb x3 x2 x1		(*opr24*,*xysp*),	(*opr24*,*xysp*)
	xb x3 x2 x1	xb x3 x2 x1		[*opr24*,*xysp*],	[*opr24*,*xysp*]
	xb x3 x2 x1	xb x3 x2 x1		(*opru24*,*Dj*),	(*opru24*,*Dk*)
	xb x3 x2 x1	xb x3 x2 x1		*opr24*,	*opr24*
	xb x3 x2 x1	xb x3 x2 x1		[*opr24*],	[*opr24*]

Short-immediate is not appropriate for the destination of a move instruction.

Instruction Fields

SIZE - This field specifies the size of the memory value to move (0b00 = 8-bit byte, 0b01 = 16-bit word, 0b10 = 24-bit pointer, and 0b11 = 32-bit long-word).

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes, 3 bytes, or 4 bytes wide, depending on the size specified by SIZE or by the instrruction opcode.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Using OPR addressing mode to specify a short-immediate operand for the destination, is not appropriate because the move instruction cannot modify the immediate operand.

MULS — Signed Multiply — MULS

Operation

$(Dj) \times (Dk) \Rightarrow Dd$

$(Dj) \times IMM \Rightarrow Dd$

$(Dj) \times (M) \Rightarrow Dd$

$(M1) \times (M2) \Rightarrow Dd$

Syntax Variations	Addressing Modes
`MULS Dd,Dj,Dk`	**REG-REG**
`MULS.B Dd,Dj,#opr8i`	**REG-IMM1**
`MULS.W Dd,Dj,#opr16i`	**REG-IMM2**
`MULS.L Dd,Dj,#opr32i`	**REG-IMM4**
`MULS.bwl Dd,Dj,oprmemreg`	**REG-OPR/1/2/3**
`MULS.bwplbwplDd,oprmemreg,oprmemreg`	**OPR/1/2/3-OPR/1/2/3**

Description

Multiplies two signed two's complement operands and stores the signed two's complement result to register D*d*. The first source operand may be a register D*j* or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M1. The second source operand may be a register D*k*, an 8-bit, 16-bit , or 32-bit immediate value, or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M or M2. Both source operands and the result are interpreted as signed two's complement values.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	0

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Set if the signed result does not fit in the result register D*d*. Cleared otherwise.

C: Cleared.

Detailed Instruction Formats

REG-REG

7	6	5	4	3	2	1	0	
0	1	0	0	1	RESULT REGISTER D*d*			4q
1	0	SOURCE 1 REGISTER D*j*			SOURCE 2 REGISTER D*k*			mb

```
4q mb                    MULS      Dd,Dj,Dk
```

REG-IMM1/2/4

7	6	5	4	3	2	1	0	
0	1	0	0	1	RESULT REGISTER D*d*			4q
1	1	SOURCE REGISTER D*j*			1	IMM_SIZE (.B, .W, –, .L)		mb
IMMEDIATE DATA								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								

```
4q mb i1                 MULS.B    Dd,Dj,#opr8i
4q mb i2 i1              MULS.W    Dd,Dj,#opr16i ;short-imm better for some values
4q mb i4 i3 i2 i1        MULS.L    Dd,Dj,#opr32i ;short-imm better for some values
```

REG-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	1	0	0	1	RESULT REGISTER D*d*			4q
1	1	SOURCE REGISTER D*j*				M2_SIZE (.B, .W, –, .L)		mb
OPR POSTBYTE (for M2)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
4q mb xb                 MULS.bwl  Dd,Dj,#oprsxe4i
4q mb xb                 MULS.bwl  Dd,Dj,Dk ;see more efficient REG-REG version
4q mb xb                 MULS.bwl  Dd,Dj,(opru4,xys)
4q mb xb                 MULS.bwl  Dd,Dj,{(+-xy)|(xy+-)|(-s)|(s+)}
4q mb xb                 MULS.bwl  Dd,Dj,(Di,xys)
4q mb xb                 MULS.bwl  Dd,Dj,[Di,xy]
4q mb xb x1              MULS.bwl  Dd,Dj,(oprs9,xysp)
4q mb xb x1              MULS.bwl  Dd,Dj,[oprs9,xysp]
4q mb xb x1              MULS.bwl  Dd,Dj,opru14
4q mb xb x2 x1           MULS.bwl  Dd,Dj,(opru18,Di)
4q mb xb x2 x1           MULS.bwl  Dd,Dj,opru18
4q mb xb x3 x2 x1        MULS.bwl  Dd,Dj,(opr24,xysp)
4q mb xb x3 x2 x1        MULS.bwl  Dd,Dj,[opr24,xysp]
4q mb xb x3 x2 x1        MULS.bwl  Dd,Dj,(opru24,Di)
4q mb xb x3 x2 x1        MULS.bwl  Dd,Dj,opr24
4q mb xb x3 x2 x1        MULS.bwl  Dd,Dj,[opr24]
```

OPR/1/2/3-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	1	0	0	1	RESULT REGISTER D*d*			4q
1	1	M1_SIZE (.B, .W, .P, .L)		M2_SIZE (.B, .W, .P, .L)		1	0	mb
OPR POSTBYTE (for M1)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
OPR POSTBYTE (for M2)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

page 2 Opcode postbyte	M1 object code	M2 object code	Instruction Mnemonic		Source Format for M1 (select 1 option in this col)	Source Format for M2 (select 1 option in this col)
4q mb	xb	xb	MULS.*bwplbwpl*	Dd,	#*oprsxe4i*,	#*oprsxe4i*
	xb	xb			*Dj*,	*Dk*
	xb	xb			(*opru4*,*xys*),	(*opru4*,*xys*)
	xb	xb			(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+),	(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+)
	xb	xb			(*Dj*,*xys*),	(*Dk*,*xys*)
	xb	xb			[*Dj*,*xy*],	[*Dk*,*xy*]
	xb x1	xb x1			(*oprs9*,*xysp*),	(*oprs9*,*xysp*)
	xb x1	xb x1			[*oprs9*,*xysp*],	[*oprs9*,*xysp*]
	xb x1	xb x1			*opru14*,	*opru14*
	xb x2 x1	xb x2 x1			(*opru18*,*Dj*),	(*opru18*,*Dk*)
	xb x2 x1	xb x2 x1			*opru18*,	*opru18*
	xb x3 x2 x1	xb x3 x2 x1			(*opr24*,*xysp*),	(*opr24*,*xysp*)
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*,*xysp*],	[*opr24*,*xysp*]
	xb x3 x2 x1	xb x3 x2 x1			(*opru24*,*Dj*),	(*opru24*,*Dk*)
	xb x3 x2 x1	xb x3 x2 x1			*opr24*,	*opr24*
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*],	[*opr24*]

All combinations are valid although some, such as specifying a data register for both M1 and M2 can be done more efficiently using the REG-REG version of the instruction.

Instruction Fields

RESULT REGISTER- This field specifies the number of the data register D*d* used for the result (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SOURCE REGISTER or SOURCE 1 REGISTER - This field specifies the number of the data register D*j* used as an operand for the multiplication (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SOURCE 2 REGISTER - This field specifies the number of the data register D*k* used as the second operand for the multiplication (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMM_SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01) or 32-bit long-word (0b11) as the size of the immediate operand. The 0b10 combination is not available for the REG-IMM1/2/4 version of the instruction because those codes are used for the OPR-OPR version.

M1_SIZE and M2_SIZE - These fields specify the size of M1 and M2 which use the general OPR addressing mode to specify short-immediate, register, or memory operands (0b00 = 8-bit byte, 0b01 = 16-bit word, 0b10 = 24-bit pointer, and 0b11 = 32-bit long-word). When a short-immediate operand is specified, it is internally sign-extended to the size specified by the M1_SIZE and/or M2_SIZE specifications. When a register is specified, it determines the size and the M1_SIZE and/or M2_SIZE specifications are ignored.

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size specified by IMM_SIZE.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Using OPR addressing mode to specify a register operand for both source operands, is less efficient than using the REG-REG version of the instruction.

MULU — Unsigned Multiply — MULU

Operation

(D*j*) × (D*k*) ⇒ D*d*

(D*j*) × IMM ⇒ D*d*

(D*j*) × (M) ⇒ D*d*

(M1) × (M2) ⇒ D*d*

Syntax Variations	Addressing Modes
`MULU Dd,Dj,Dk`	**REG-REG**
`MULU.B Dd,Dj,#opr8i`	**REG-IMM1**
`MULU.W Dd,Dj,#opr16i`	**REG-IMM2**
`MULU.L Dd,Dj,#opr32i`	**REG-IMM4**
`MULU.bwl Dd,Dj,oprmemreg`	**REG-OPR/1/2/3**
`MULU.bwplbwplDd,oprmemreg,oprmemreg`	**OPR/1/2/3-OPR/1/2/3**

Description

Multiplies two unsigned operands and stores the unsigned result to register D*d*. The first source operand may be a register D*j* or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M1. The second source operand may be a register D*k*, an 8-bit, 16-bit , or 32-bit immediate value, or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M or M2. Both source operands and the result are interpreted as unsigned values.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	0

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Set if the unsigned result does not fit in the result register D*d*. Cleared otherwise.

C: Cleared.

Detailed Instruction Formats

REG-REG

7	6	5	4	3	2	1	0	
0	1	0	0	1	RESULT REGISTER D*d*			4q
0	0	SOURCE 1 REGISTER D*j*			SOURCE 2 REGISTER D*k*			mb

```
4q mb                    MULU      Dd,Dj,Dk
```

REG-IMM1/2/4

7	6	5	4	3	2	1	0	
0	1	0	0	1	RESULT REGISTER D*d*			4q
0	1	SOURCE REGISTER D*j*			1	IMM_SIZE (.B, .W, –, .L)		mb
IMMEDIATE DATA								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								

```
4q mb i1                 MULU.B    Dd,Dj,#opr8i
4q mb i2 i1              MULU.W    Dd,Dj,#opr16i ;short-imm better for some values
4q mb i4 i3 i2 i1        MULU.L    Dd,Dj,#opr32i ;short-imm better for some values
```

REG-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	1	0	0	1	RESULT REGISTER D*d*			4q
0	1	SOURCE REGISTER D*j*				M2_SIZE (.B, .W, –, .L)		mb
OPR POSTBYTE (for M2)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
4q mb xb                 MULU.bwl  Dd,Dj,#oprsxe4i
4q mb xb                 MULU.bwl  Dd,Dj,Dk ;see more efficient REG-REG version
4q mb xb                 MULU.bwl  Dd,Dj,(opru4,xys)
4q mb xb                 MULU.bwl  Dd,Dj,{(+-xy)|(xy+-)|(-s)|(s+)}
4q mb xb                 MULU.bwl  Dd,Dj,(Di,xys)
4q mb xb                 MULU.bwl  Dd,Dj,[Di,xy]
4q mb xb x1              MULU.bwl  Dd,Dj,(oprs9,xysp)
4q mb xb x1              MULU.bwl  Dd,Dj,[oprs9,xysp]
4q mb xb x1              MULU.bwl  Dd,Dj,opru14
4q mb xb x2 x1           MULU.bwl  Dd,Dj,(opru18,Di)
4q mb xb x2 x1           MULU.bwl  Dd,Dj,opru18
4q mb xb x3 x2 x1        MULU.bwl  Dd,Dj,(opr24,xysp)
4q mb xb x3 x2 x1        MULU.bwl  Dd,Dj,[opr24,xysp]
4q mb xb x3 x2 x1        MULU.bwl  Dd,Dj,(opru24,Di)
4q mb xb x3 x2 x1        MULU.bwl  Dd,Dj,opr24
4q mb xb x3 x2 x1        MULU.bwl  Dd,Dj,[opr24]
```

OPR/1/2/3-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	1	0	0	1	RESULT REGISTER D*d*			4q
0	1	M1_SIZE (.B, .W, .P, .L)		M2_SIZE (.B, .W, .P, .L)		1	0	mb
OPR POSTBYTE (for M1)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
OPR POSTBYTE (for M2)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

page 2 Opcode postbyte	M1 object code	M2 object code	Instruction Mnemonic		Source Format for M1 (select 1 option in this col)	Source Format for M2 (select 1 option in this col)
4q mb	xb	xb	MULU.*bwplbwpl*	Dd,	#*oprsxe4i*,	#*oprsxe4i*
	xb	xb			*Dj*,	*Dk*
	xb	xb			(*opru4*,*xys*),	(*opru4*,*xys*)
	xb	xb			(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+),	(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+)
	xb	xb			(*Dj*,*xys*),	(*Dk*,*xys*)
	xb	xb			[*Dj*,*xy*],	[*Dk*,*xy*]
	xb x1	xb x1			(*oprs9*,*xysp*),	(*oprs9*,*xysp*)
	xb x1	xb x1			[*oprs9*,*xysp*],	[*oprs9*,*xysp*]
	xb x1	xb x1			*opru14*,	*opru14*
	xb x2 x1	xb x2 x1			(*opru18*,*Dj*),	(*opru18*,*Dk*)
	xb x2 x1	xb x2 x1			*opru18*,	*opru18*
	xb x3 x2 x1	xb x3 x2 x1			(*opr24*,*xysp*),	(*opr24*,*xysp*)
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*,*xysp*],	[*opr24*,*xysp*]
	xb x3 x2 x1	xb x3 x2 x1			(*opru24*,*Dj*),	(*opru24*,*Dk*)
	xb x3 x2 x1	xb x3 x2 x1			*opr24*,	*opr24*
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*],	[*opr24*]

All combinations are valid although some, such as specifying a data register for both M1 and M2 can be done more efficiently using the REG-REG version of the instruction.

Instruction Fields

RESULT REGISTER- This field specifies the number of the data register D*d* used for the result (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SOURCE REGISTER or SOURCE 1 REGISTER - This field specifies the number of the data register D*j* used as an operand for the multiplication (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SOURCE 2 REGISTER - This field specifies the number of the data register D*k* used as the second operand for the multiplication (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMM_SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01) or 32-bit long-word (0b11) as the size of the immediate operand. The 0b10 combination is not available for the REG-IMM1/2/4 version of the instruction because those codes are used for the OPR-OPR version.

M1_SIZE and M2_SIZE - These fields specify the size of M1 and M2 which use the general OPR addressing mode to specify short-immediate, register, or memory operands (0b00 = 8-bit byte, 0b01 = 16-bit word, 0b10 = 24-bit pointer, and 0b11 = 32-bit long-word). When a short-immediate operand is specified, it is internally sign-extended to the size specified by the M1_SIZE and/or M2_SIZE specifications. When a register is specified, it determines the size and the M1_SIZE and/or M2_SIZE specifications are ignored.

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size specified by IMM_SIZE.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Using OPR addressing mode to specify a register operand for both source operands, is less efficient than using the REG-REG version of the instruction.

NEG — Two's Complement Negate — NEG

Operation

$0 - (M) = \sim(M) + 1 \Rightarrow M$

Syntax Variations	Addressing Modes
`NEG.bwl oprmemreg`	**OPR/1/2/3**

Description

Replaces the content of memory location M with its two's complement. The memory operand *oprmemreg* can be a data register, a memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. The size of the memory operand M is determined by the suffix (.B=8 bit byte, .W=16 bit word, or .L=32 bit long-word). If the OPR memory addressing mode is used to specify a data register D*j*, the register determines the size for the operation and the *.bwl* suffix is ignored. *It is inappropriate to specify a short immediate operand using the OPR addressing mode for this instruction because it is not possible to modify the immediate operand.*

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Set if a two's complement overflow was the result of the implied subtraction from zero. Cleared otherwise.

C: Set if there is a borrow in the implied subtraction from zero. Cleared otherwise. Set in all cases, except when (M) = 0.

Detailed Instruction Formats

OPR/1/2/3

7	6	5	4	3	2	1	0	
1	1	0	1	1	1	SIZE (.B, .W, –, .L)		Dp
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
Dp xb    NEG.bwl   #oprsxe4i ;not appropriate for destination
Dp xb    NEG.bwl   Di
Dp xb    NEG.bwl   (opru4,xys)
Dp xb    NEG.bwl   {(+-xy)|(xy+-)|(-s)|(s+)}
Dp xb    NEG.bwl   (Di,xys)
```

```
Dp xb                          NEG.bwl    [Di,xy]
Dp xb x1                       NEG.bwl    (oprs9,xysp)
Dp xb x1                       NEG.bwl    [oprs9,xysp]
Dp xb x1                       NEG.bwl    opru14
Dp xb x2 x1                    NEG.bwl    (opru18,Di)
Dp xb x2 x1                    NEG.bwl    opru18
Dp xb x3 x2 x1                 NEG.bwl    (opr24,xysp)
Dp xb x3 x2 x1                 NEG.bwl    [opr24,xysp]
Dp xb x3 x2 x1                 NEG.bwl    (opru24,Di)
Dp xb x3 x2 x1                 NEG.bwl    opr24
Dp xb x3 x2 x1                 NEG.bwl    [opr24]
```

Instruction Fields

SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01), or 32-bit long-word (0b11) as the size of the operation.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. *It is inappropriate to specify a short immediate operand using the OPR addressing mode for this instruction because it is not possible to modify the immediate operand.*

NOP

Null Operation

NOP

Operation

No operation.

Syntax Variations	Addressing Modes
NOP	INH

Description

This single-byte instruction increments the PC and does nothing else. No CPU registers are affected. NOP is typically used to produce a time delay, although some software disciplines discourage CPU frequency-based time delays. During debug, NOP instructions are sometimes used to temporarily replace other machine code instructions, thus disabling the replaced instruction(s).

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

INH

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	1	01

01 NOP

OR — Bitwise OR — OR

Operation

(D*i*) | (M) ⇒ D*i*

Syntax Variations	Addressing Modes
OR *Di,#oprimmsz*	**IMM1/2/4**
OR *Di,oprmemreg*	**OPR/1/2/3**

Description

Bitwise OR register D*i* with a memory operand and store the result to D*i*. When the operand is an immediate value, it has the same size as register D*i*. In the case of the general OPR addressing operand, *oprmemreg* can be a sign-extended immediate value (–1, 1, 2, 3..14, 15), a data register, a memory operand the same size as D*i* at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	–

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Cleared.

Detailed Instruction Formats

IMM1/2/4

7	6	5	4	3	2	1	0	
0	1	1	1	1	SD REGISTER D*i*			7p
IMMEDIATE DATA								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								

```
7p i1                    OR        Di,#opr8i ;for Di = 8-bit D0 or D1
7p i2 i1                 OR        Di,#opr16i ;for Di = 16-bit D2, D3, D4, or D5
7p i4 i3 i2 i1           OR        Di,#opr32i ;for Di = 32-bit D6 or D7
```

OPR/1/2/3

7	6	5	4	3	2	1	0	
1	0	0	0	1	SD REGISTER D*i*			8q
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
8q xb                    OR    Di,#oprsxe4i ;-1, +1, 2, 3...14, 15
8q xb                    OR    Di,Dj
8q xb                    OR    Di,(opru4,xys)
8q xb                    OR    Di,{(+-xy)|(xy+-)|(-s)|(s+)}
8q xb                    OR    Di,(Dj,xys)
8q xb                    OR    Di,[Dj,xy]
8q xb x1                 OR    Di,(oprs9,xysp)
8q xb x1                 OR    Di,[oprs9,xysp]
8q xb x1                 OR    Di,opru14
8q xb x2 x1              OR    Di,(opru18,Dj)
8q xb x2 x1              OR    Di,opru18
8q xb x3 x2 x1           OR    Di,(opr24,xysp)
8q xb x3 x2 x1           OR    Di,[opr24,xysp]
8q xb x3 x2 x1           OR    Di,(opru24,Dj)
8q xb x3 x2 x1           OR    Di,opr24
8q xb x3 x2 x1           OR    Di,[opr24]
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as a source operand and for the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size of the register D*i*.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

ORCC Bitwise OR CCL with Immediate ORCC

Operation

(CCL) | (M) ⇒ CCL

Syntax Variations	Addressing Modes
ORCC #*opr8i*	**IMM1**

Description

Performs a bitwise OR operation between the 8-bit immediate memory operand and the content of CCL (the low order 8 bits of the CCR). The result is stored in CCL.

When the CPU is in user state, this instruction is restricted to changing the condition codes (the flags N, Z, V, C) and cannot change the settings in the S, X, or I bits.

No software instruction can change the X bit from 0 to 1 in user or supervisor state.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C	
–	–	–	–	–	–	⇑	–	–	⇑	⇑	⇑	⇑	⇑	supervisor state
–	–	–	–	–	–	–	–	–	–	⇑	⇑	⇑	⇑	user state

Condition code bits are set if the corresponding bit was 1 before the operation or if the corresponding bit in the immediate mask is 1.

Detailed Instruction Formats

IMM1

7	6	5	4	3	2	1	0	
1	1	0	1	1	1	1	0	DE
IMMEDIATE DATA								i1

DE i1 ORCC #*opr8i*

PSH — Push Registers onto Stack — PSH

Operation

Push specified registers onto the stack.

for push mask oprregs2...

If Y specified: $(SP) - 3 \Rightarrow SP$; $Y \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
If X specified: $(SP) - 3 \Rightarrow SP$; $X \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
If D7 specified: $(SP) - 4 \Rightarrow SP$; $D7 \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)} : M_{(SP+3)}$
If D6 specified: $(SP) - 4 \Rightarrow SP$; $D6 \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)} : M_{(SP+3)}$
If D5 specified: $(SP) - 2 \Rightarrow SP$; $D5 \Rightarrow M_{(SP)} : M_{(SP+1)}$
If D4 specified: $(SP) - 2 \Rightarrow SP$; $D4 \Rightarrow M_{(SP)} : M_{(SP+1)}$

or for push mask oprregs1...

If D3 specified: $(SP) - 2 \Rightarrow SP$; $D3 \Rightarrow M_{(SP)} : M_{(SP+1)}$
If D2 specified: $(SP) - 2 \Rightarrow SP$; $D2 \Rightarrow M_{(SP)} : M_{(SP+1)}$
If D1 specified: $(SP) - 1 \Rightarrow SP$; $D1 \Rightarrow M_{(SP)}$
If D0 specified: $(SP) - 1 \Rightarrow SP$; $D0 \Rightarrow M_{(SP)}$
If CCL specified: $(SP) - 1 \Rightarrow SP$; $CCL \Rightarrow M_{(SP)}$
If CCH specified: $(SP) - 1 \Rightarrow SP$; $CCH \Rightarrow M_{(SP)}$

Syntax Variations		Addressing Modes
PSH	*oprregs1*	INH
PSH	*oprregs2*	INH
PSH	ALL	INH
PSH	ALL16b	INH

Description

Push specified CPU registers onto stack.

There are two possible register lists (*oprregs1*, *oprregs2*) and two special cases:

- *oprregs1* includes any combination of the registers CCH, CCL, D0, D1, D2, D3
- *oprregs2* includes any combination of the registers D4, D5, D6, D7, X, Y
- If pb postbyte = 0x00, push all registers in the order Y,X,D7,D6,D5,D4,D3,D2,D1,D0,CCL,CCH
- If pb postbyte = 0x40, push all 4 16-bit registers in the order D5,D4,D3,D2

The registers to be pushed are encoded in an instruction postbyte (pb) which includes one mask bit for each of the registers in the list as well as a control bit that specifies which list the registers are from and whether they should be pushed or pulled. If a combination of registers includes random registers from both lists, two PSH instructions are required. Registers are pushed starting with the lowest order byte of the register that is furthest to the right in the list. The stack pointer is decremented by one for each byte that is pushed onto the stack. After the PSH instruction, SP points at the last byte that was pushed.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

INH

7	6	5	4	3	2	1	0	
0	0	0	0	0	1	0	0	04
0	MASK2/1	R5	R4	R3	R2	R1	R0	pb

```
04 pb          PSH     oprregs1
04 pb          PSH     oprregs2
04 00          PSH     ALL
04 40          PSH     ALL16b
```

Instruction Fields

The MASK2/1 and R0..R5 fields specify the registers to be pushed onto the stack as listed in the table below.

MASK2/1	R5	R4	R3	R2	R1	R0
0	CCH	CCL	D0	D1	D2	D3
1	D4	D5	D6	D7	X	Y

The R0..R5 fields are treated as a mask to determine if the associated register is to be pushed on the stack (“1”) or not (“0”).

The register are pushed on the stack in right-to-left sequence (the register associated with R0 is pushed first, the register associated with R5 is pushed last).

PUL — Pull Registers from Stack — PUL

Operation

Pull specified registers from the stack.

for pull mask oprregs1...

If CCH specified: $M_{(SP)} \Rightarrow CCH$; $(SP) + 1 \Rightarrow SP$
If CCL specified: $M_{(SP)} \Rightarrow CCL$; $(SP) + 1 \Rightarrow SP$
If D0 specified: $M_{(SP)} \Rightarrow D0$; $(SP) + 1 \Rightarrow SP$
If D1 specified: $M_{(SP)} \Rightarrow D1$; $(SP) + 1 \Rightarrow SP$
If D2 specified: $M_{(SP)} : M_{(SP + 1)} \Rightarrow D2$; $(SP) + 2 \Rightarrow SP$
If D3 specified: $M_{(SP)} : M_{(SP + 1)} \Rightarrow D3$; $(SP) + 2 \Rightarrow SP$

or for pull mask oprregs2...

If D4 specified: $M_{(SP)} : M_{(SP + 1)} \Rightarrow D4$; $(SP) + 2 \Rightarrow SP$
If D5 specified: $M_{(SP)} : M_{(SP + 1)} \Rightarrow D5$; $(SP) + 2 \Rightarrow SP$
If D6 specified: $M_{(SP)}: M_{(SP + 1)}: M_{(SP + 2)} : M_{(SP + 3)} \Rightarrow D6$; $(SP) + 4 \Rightarrow SP$
If D7 specified: $M_{(SP)}: M_{(SP + 1)}: M_{(SP + 2)} : M_{(SP + 3)} \Rightarrow D7$; $(SP) + 4 \Rightarrow SP$
If X specified: $M_{(SP)}: M_{(SP + 1)}: M_{(SP + 2)} \Rightarrow X$; $(SP) + 3 \Rightarrow SP$
If Y specified: $M_{(SP)}: M_{(SP + 1)}: M_{(SP + 2)} \Rightarrow Y$; $(SP) + 3 \Rightarrow SP$

Syntax Variations		Addressing Modes
PUL	*oprregs1*	**INH**
PUL	*oprregs2*	**INH**
PUL	ALL	**INH**
PUL	ALL16b	**INH**

Description

Pull specified CPU registers from stack.

There are two possible register lists (*oprregs1*, *oprregs2*) and two special cases:

- *oprregs1* includes any combination of the registers CCH, CCL, D0, D1, D2, D3
- *oprregs2* includes any combination of the registers D4, D5, D6, D7, X, Y
- If pb postbyte = 0x80, pull all registers in the order CCH,CCL,D0,D1,D2,D3,D4,D5,D6,D7,X,Y
- If pb postbyte = 0xC0, pull all 4 16-bit registers in the order D2,D3,D4,D5

The registers to be pulled are encoded in an instruction postbyte which includes one mask bit for each of the registers in the list as well as a control bit that specifies which list the registers are from and whether they should be pushed or pulled. If a combination of registers includes random registers from both lists, two PUL instructions are required. Registers are pulled starting with the highest order byte of the register that is furthest to the left in the list. The stack pointer is incremented by one for each byte that is pulled from the stack. After the PUL instruction, SP points at the next higher address above the last byte that was pulled.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

If CCH or CCL are pulled, the values pulled are written directly into the CCR and the CCR details shown in the figure above do not apply. Unimplemented bits in the CCR can not be changed. In user state, only the four flag bits N, Z, V, and C can be modified. In supervisor state, any of the implemented CCR bits can be modified however the X bit can never be changed from 0 to 1 by any instruction in any mode.

Detailed Instruction Formats

INH

7	6	5	4	3	2	1	0	
0	0	0	0	0	1	0	0	04
1	MASK2/1	R5	R4	R3	R2	R1	R0	pb

```
04 pb        PUL    oprregs1
04 pb        PUL    oprregs2
04 80        PUL    ALL
04 C0        PUL    ALL16b
```

Instruction Fields

The MASK2/1 and R0..R5 fields specify the registers to be pulled from the stack as listed in the table below.

MASK2/1	R5	R4	R3	R2	R1	R0
0	CCH	CCL	D0	D1	D2	D3
1	D4	D5	D6	D7	X	Y

The R0..R5 fields are treated as a mask to determine if the associated register is to be pulled from the stack (“1”) or not (“0”).

The register are pulled on the stack in left-to-right sequence (the register associated with R5 is pulled first, the register associated with R0 is pulled last).

QMULS Signed Fractional Multiply QMULS

Operation

$(Dj) \times (Dk) \Rightarrow Dd$

$(Dj) \times IMM \Rightarrow Dd$

$(Dj) \times (M) \Rightarrow Dd$

$(M1) \times (M2) \Rightarrow Dd$

Syntax Variations		Addressing Modes
`QMULS`	`Dd,Dj,Dk`	**REG-REG**
`QMULS.B`	`Dd,Dj,#opr8i`	**REG-IMM1**
`QMULS.W`	`Dd,Dj,#opr16i`	**REG-IMM2**
`QMULS.L`	`Dd,Dj,#opr32i`	**REG-IMM4**
`QMULS.bwl`	`Dd,Dj,oprmemreg`	**REG-OPR/1/2/3**
`QMULS.bwplbwpl`	`Dd,oprmemreg,oprmemreg`	**OPR/1/2/3-OPR/1/2/3**

Description

Multiplies two signed fractional two's complement operands and stores the signed fractional two's complement result to register D*d*. The first source operand may be a register D*j* or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M1. The second source operand may be a register D*k*, an 8-bit, 16-bit , or 32-bit immediate value, or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M or M2.

Both source operands and the result are interpreted as signed fractional two's complement numbers in s.7, s.15, s.23 or s.31 formats as defined in ISO-C Technical Report TR 18037. That means the MSB is interpreted as sign, the remaining 7, 15, 23 or 31 bits are interpreted as fractional portion of a fixed-point number (also known as "Q"-format).

In order to allow operands of different sizes to be multiplied, the source operands are aligned. This means that smaller operands are right-appended with zeroes to make the sizes of both operands match. This ensures the alignment of the position of the binary point of the source operands before the actual multiplication operation commences.

The content of the result register represents the most-significant portion of the actual multiplication result. Any least significant multiplication result-bits not fitting into the result register are cut-off without rounding.

If both source operands contain the representation of the minimum negative number of the fixed-point range, this operation saturates. In this case the result is the representation of the maximum positive number of the fixed-point range.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	0

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Set if saturation has occurred. Cleared otherwise.

C: Cleared.

Detailed Instruction Formats

REG-REG

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
1	0	1	1	0	RESULT REGISTER D*d*			Bn
1	0	SOURCE 1 REGISTER D*j*			SOURCE 2 REGISTER D*k*			mb

```
1B Bn mb                    QMULS      Dd,Dj,Dk
```

REG-IMM1/2/4

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
1	0	1	1	0	RESULT REGISTER D*d*			Bn
1	1	SOURCE REGISTER D*j*			1	IMM_SIZE (.B, .W, –, .L)		mb
IMMEDIATE DATA								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								

```
1B Bn mb i1                 QMULS.B    Dd,Dj,#opr8i
1B Bn mb i2 i1              QMULS.W    Dd,Dj,#opr16i ;short-imm better for some values
1B Bn mb i4 i3 i2 i1        QMULS.L    Dd,Dj,#opr32i ;short-imm better for some values
```

REG-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
1	0	1	1	0	RESULT REGISTER D*d*			Bn
1	1	SOURCE REGISTER D*j*				M2_SIZE (.B, .W, –, .L)		mb
OPR POSTBYTE (for M2)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B Bn mb xb                    QMULS.bwl  Dd,Dj,#oprsxe4i
1B Bn mb xb                    QMULS.bwl  Dd,Dj,Dk ;see more efficient REG-REG version
1B Bn mb xb                    QMULS.bwl  Dd,Dj,(opru4,xys)
1B Bn mb xb                    QMULS.bwl  Dd,Dj,{(+-xy)|(xy+-)|(-s)|(s+)}
1B Bn mb xb                    QMULS.bwl  Dd,Dj,(Di,xys)
1B Bn mb xb                    QMULS.bwl  Dd,Dj,[Di,xy]
1B Bn mb xb x1                 QMULS.bwl  Dd,Dj,(oprs9,xysp)
1B Bn mb xb x1                 QMULS.bwl  Dd,Dj,[oprs9,xysp]
1B Bn mb xb x1                 QMULS.bwl  Dd,Dj,opru14
1B Bn mb xb x2 x1              QMULS.bwl  Dd,Dj,(opru18,Di)
1B Bn mb xb x2 x1              QMULS.bwl  Dd,Dj,opru18
1B Bn mb xb x3 x2 x1           QMULS.bwl  Dd,Dj,(opr24,xysp)
1B Bn mb xb x3 x2 x1           QMULS.bwl  Dd,Dj,[opr24,xysp]
1B Bn mb xb x3 x2 x1           QMULS.bwl  Dd,Dj,(opru24,Di)
1B Bn mb xb x3 x2 x1           QMULS.bwl  Dd,Dj,opr24
1B Bn mb xb x3 x2 x1           QMULS.bwl  Dd,Dj,[opr24]
```

OPR/1/2/3-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
1	0	1	1	0	RESULT REGISTER D*d*			Bn
1	1	M1_SIZE (.B, .W, .P, .L)		M2_SIZE (.B, .W, .P, .L)		1	0	mb
OPR POSTBYTE (for M1)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
OPR POSTBYTE (for M2)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

page 2 Opcode postbyte	M1 object code	M2 object code	Instruction Mnemonic		Source Format for M1 (select 1 option in this col)	Source Format for M2 (select 1 option in this col)
1B Bn mb	xb	xb	QMULS.*bwplbwpl*	Dd,	#*oprsxe4i*,	#*oprsxe4i*
	xb	xb			*Dj*,	*Dk*
	xb	xb			(*opru4*,*xys*),	(*opru4*,*xys*)
	xb	xb			(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+),	(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+)
	xb	xb			(*Dj*,*xys*),	(*Dk*,*xys*)
	xb	xb			[*Dj*,*xy*],	[*Dk*,*xy*]
	xb x1	xb x1			(*oprs9*,*xysp*),	(*oprs9*,*xysp*)
	xb x1	xb x1			[*oprs9*,*xysp*],	[*oprs9*,*xysp*]
	xb x1	xb x1			*opru14*,	*opru14*
	xb x2 x1	xb x2 x1			(*opru18*,*Dj*),	(*opru18*,*Dk*)
	xb x2 x1	xb x2 x1			*opru18*,	*opru18*
	xb x3 x2 x1	xb x3 x2 x1			(*opr24*,*xysp*),	(*opr24*,*xysp*)
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*,*xysp*],	[*opr24*,*xysp*]
	xb x3 x2 x1	xb x3 x2 x1			(*opru24*,*Dj*),	(*opru24*,*Dk*)
	xb x3 x2 x1	xb x3 x2 x1			*opr24*,	*opr24*
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*],	[*opr24*]

All combinations are valid although some, such as specifying a data register for both M1 and M2 can be done more efficiently using the REG-REG version of the instruction.

Instruction Fields

RESULT REGISTER- This field specifies the number of the data register D*d* used for the result (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SOURCE REGISTER or SOURCE 1 REGISTER - This field specifies the number of the data register D*j* used as an operand for the multiplication (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SOURCE 2 REGISTER - This field specifies the number of the data register D*k* used as the second operand for the multiplication (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMM_SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01) or 32-bit long-word (0b11) as the size of the immediate operand. The 0b10 combination is not available for the REG-IMM1/2/4 version of the instruction because those codes are used for the OPR-OPR version.

M1_SIZE and M2_SIZE - These fields specify the size of M1 and M2 which use the general OPR addressing mode to specify short-immediate, register, or memory operands (0b00 = 8-bit byte, 0b01 = 16-bit word, 0b10 = 24-bit pointer, and 0b11 = 32-bit long-word). When a short-immediate operand is specified, it is internally sign-extended to the size specified by the M1_SIZE and/or M2_SIZE specifications. When a register is specified, it determines the size and the M1_SIZE and/or M2_SIZE specifications are ignored.

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size specified by IMM_SIZE.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Using OPR addressing mode to specify a register operand for both source operands, is less efficient than using the REG-REG version of the instruction.

QMULU Unsigned Fractional Multiply QMULU

Operation

(D*j*) × (D*k*) ⇒ D*d*

(D*j*) × IMM ⇒ D*d*

(D*j*) × (M) ⇒ D*d*

(M1) × (M2) ⇒ D*d*

Syntax Variations		Addressing Modes
`QMULU`	`Dd,Dj,Dk`	**REG-REG**
`QMULU.B`	`Dd,Dj,#opr8i`	**REG-IMM1**
`QMULU.W`	`Dd,Dj,#opr16i`	**REG-IMM2**
`QMULU.L`	`Dd,Dj,#opr32i`	**REG-IMM4**
`QMULU.bwl`	`Dd,Dj,oprmemreg`	**REG-OPR/1/2/3**
`QMULU.bwplbwpl`	`Dd,oprmemreg,oprmemreg`	**OPR/1/2/3-OPR/1/2/3**

Description

Multiplies two unsigned operands and stores the unsigned result to register D*d*. The first source operand may be a register D*j* or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M1. The second source operand may be a register D*k*, an 8-bit, 16-bit , or 32-bit immediate value, or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M or M2. Both source operands and the result are interpreted as unsigned values.

Both source operands and the result are interpreted as unsigned numbers in .8, .16, .24 or .32 formats as defined in ISO-C Technical Report TR 18037. That means all 8, 16, 24 or 32 bits are interpreted as fractional portion of a fixed-point number (also known as “Q”-format).

In order to allow operands of different sizes to be multiplied, the source operands are aligned. This means that smaller operands are right-appended with zeroes to make the sizes of both operands match. This ensures the alignment of the position of the binary point of the source operands before the actual multiplication operation commences.

The content of the result register represents the most-significant portion of the actual multiplication result. Any least significant multiplication result-bits not fitting into the result register are cut-off without rounding.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	0

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Cleared.

C: Cleared.

Detailed Instruction Formats

REG-REG

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
1	0	1	1	0	RESULT REGISTER D*d*			Bn
0	0	SOURCE 1 REGISTER D*j*			SOURCE 2 REGISTER D*k*			mb

```
1B Bn mb                    QMULU      Dd,Dj,Dk
```

REG-IMM1/2/4

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
1	0	1	1	0	RESULT REGISTER D*d*			Bn
0	1	SOURCE REGISTER D*j*			1	IMM_SIZE (.B, .W, –, .L)		mb
IMMEDIATE DATA								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)								

```
1B Bn mb i1                 QMULU.B    Dd,Dj,#opr8i
1B Bn mb i2 i1              QMULU.W    Dd,Dj,#opr16i ;short-imm better for some values
1B Bn mb i4 i3 i2 i1        QMULU.L    Dd,Dj,#opr32i ;short-imm better for some values
```

REG-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
1	0	1	1	0	RESULT REGISTER D*d*			Bn
0	1	SOURCE REGISTER D*j*				M2_SIZE (.B, .W, –, .L)		mb
OPR POSTBYTE (for M2)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B Bn mb xb                 QMULU.bwl  Dd,Dj,#oprsxe4i
1B Bn mb xb                 QMULU.bwl  Dd,Dj,Dk ;see more efficient REG-REG version
1B Bn mb xb                 QMULU.bwl  Dd,Dj,(opru4,xys)
1B Bn mb xb                 QMULU.bwl  Dd,Dj,{(+-xy)|(xy+-)|(-s)|(s+)}
1B Bn mb xb                 QMULU.bwl  Dd,Dj,(Di,xys)
1B Bn mb xb                 QMULU.bwl  Dd,Dj,[Di,xy]
1B Bn mb xb x1              QMULU.bwl  Dd,Dj,(oprs9,xysp)
1B Bn mb xb x1              QMULU.bwl  Dd,Dj,[oprs9,xysp]
```

```
1B Bn mb xb x1                QMULU.bwl  Dd,Dj,opru14
1B Bn mb xb x2 x1             QMULU.bwl  Dd,Dj,(opru18,Di)
1B Bn mb xb x2 x1             QMULU.bwl  Dd,Dj,opru18
1B Bn mb xb x3 x2 x1          QMULU.bwl  Dd,Dj,(opr24,xysp)
1B Bn mb xb x3 x2 x1          QMULU.bwl  Dd,Dj,[opr24,xysp]
1B Bn mb xb x3 x2 x1          QMULU.bwl  Dd,Dj,(opru24,Di)
1B Bn mb xb x3 x2 x1          QMULU.bwl  Dd,Dj,opr24
1B Bn mb xb x3 x2 x1          QMULU.bwl  Dd,Dj,[opr24]
```

OPR/1/2/3-OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
1	0	1	1	0	RESULT REGISTER D*d*			Bn
0	1	M1_SIZE (.B, .W, .P, .L)		M2_SIZE (.B, .W, .P, .L)		1	0	mb
OPR POSTBYTE (for M1)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
OPR POSTBYTE (for M2)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

page 2 Opcode postbyte	M1 object code	M2 object code	Instruction Mnemonic		Source Format for M1 (select 1 option in this col)	Source Format for M2 (select 1 option in this col)
1B Bn mb	xb	xb	QMULU.*bwplbwpl*	D*d*,	#*oprsxe4i*,	#*oprsxe4i*
	xb	xb			*Dj*,	*Dk*
	xb	xb			(*opru4*,*xys*),	(*opru4*,*xys*)
	xb	xb			(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+),	(+-*xy*)|(*xy*+-)|(-*s*)|(*s*+)
	xb	xb			(*Dj*,*xys*),	(*Dk*,*xys*)
	xb	xb			[*Dj*,*xy*],	[*Dk*,*xy*]
	xb x1	xb x1			(*oprs9*,*xysp*),	(*oprs9*,*xysp*)
	xb x1	xb x1			[*oprs9*,*xysp*],	[*oprs9*,*xysp*]
	xb x1	xb x1			*opru14*,	*opru14*
	xb x2 x1	xb x2 x1			(*opru18*,*Dj*),	(*opru18*,*Dk*)
	xb x2 x1	xb x2 x1			*opru18*,	*opru18*
	xb x3 x2 x1	xb x3 x2 x1			(*opr24*,*xysp*),	(*opr24*,*xysp*)
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*,*xysp*],	[*opr24*,*xysp*]
	xb x3 x2 x1	xb x3 x2 x1			(*opru24*,*Dj*),	(*opru24*,*Dk*)
	xb x3 x2 x1	xb x3 x2 x1			*opr24*,	*opr24*
	xb x3 x2 x1	xb x3 x2 x1			[*opr24*],	[*opr24*]

All combinations are valid although some, such as specifying a data register for both M1 and M2 can be done more efficiently using the REG-REG version of the instruction.

Instruction Fields

RESULT REGISTER- This field specifies the number of the data register D*d* used for the result (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SOURCE REGISTER or SOURCE 1 REGISTER - This field specifies the number of the data register D*j* used as an operand for the multiplication (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SOURCE 2 REGISTER - This field specifies the number of the data register D*k* used as the second operand for the multiplication (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMM_SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01) or 32-bit long-word (0b11) as the size of the immediate operand. The 0b10 combination is not available for the REG-IMM1/2/4 version of the instruction because those codes are used for the OPR-OPR version.

M1_SIZE and M2_SIZE - These fields specify the size of M1 and M2 which use the general OPR addressing mode to specify short-immediate, register, or memory operands (0b00 = 8-bit byte, 0b01 = 16-bit word, 0b10 = 24-bit pointer, and 0b11 = 32-bit long-word). When a short-immediate operand is specified, it is internally sign-extended to the size specified by the M1_SIZE and/or M2_SIZE specifications. When a register is specified, it determines the size and the M1_SIZE and/or M2_SIZE specifications are ignored.

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size specified by IMM_SIZE.

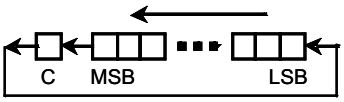
OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. Using OPR addressing mode to specify a register operand for both source operands, is less efficient than using the REG-REG version of the instruction.

ROL

Rotate Left Through Carry

ROL

Operation



Syntax Variations	Addressing Modes
`ROL.`*`bwpl oprmemreg`*	**OPR/1/2/3**

Description

Rotate an operand left (through the carry bit) 1 bit-position. The 8-bit byte (.B), 16-bit word (.W), 24-bit pointer (.P), or 32-bit long-word (.L) memory operand to be rotated is specified using general OPR addressing. The operand, *oprmemreg,* can be a data register, a memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. The original carry bit is shifted into the LSB and the MSB is shifted out to the carry bit (C). *It is not appropriate to specify a short-immediate operand with the OPR addressing mode because it is not possible to modify the immediate operand.*

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	Δ

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Cleared

C: Set if the bit shifted out of the MSB of the operand was set before the shift, cleared otherwise.

Detailed Instruction Formats

OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	0	x	x	x	1n
x	L/R=1	1	x	x	1	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (specifes source operand to be rotated)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1n sb xb                    ROL.bwpl    #oprsxe4i ;not appropriate for destination
1n sb xb                    ROL.bwpl    Di
1n sb xb                    ROL.bwpl    (opru4,xys)
1n sb xb                    ROL.bwpl    {(+-xy)|(xy+-)|(-s)|(s+)}
1n sb xb                    ROL.bwpl    (Di,xys)
1n sb xb                    ROL.bwpl    [Di,xy]
1n sb xb x1                 ROL.bwpl    (oprs9,xysp)
1n sb xb x1                 ROL.bwpl    [oprs9,xysp]
1n sb xb x1                 ROL.bwpl    opru14
1n sb xb x2 x1              ROL.bwpl    (opru18,Di)
1n sb xb x2 x1              ROL.bwpl    opru18
1n sb xb x3 x2 x1           ROL.bwpl    (opr24,xysp)
1n sb xb x3 x2 x1           ROL.bwpl    [opr24,xysp]
1n sb xb x3 x2 x1           ROL.bwpl    (opru24,Di)
1n sb xb x3 x2 x1           ROL.bwpl    opr24
1n sb xb x3 x2 x1           ROL.bwpl    [opr24]
```

Instruction Fields

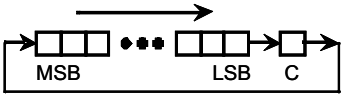
L/R - This bit selects the rotate direction, left (1) or right (0).

SIZE (.B, .W, .P, .L) - This field specifies 8-bit byte (0b00), 16-bit word (0b01), 24-bit pointer (0b10) or 32-bit long-word (0b11) as the size of the source operand.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

ROR — Rotate Right Through Carry — ROR

Operation



Syntax Variations	Addressing Modes
`ROR.`*`bwpl oprmemreg`*	**OPR/1/2/3**

Description

Rotate an operand right (through the carry bit) 1 bit-position. The 8-bit byte (.B), 16-bit word (.W), 24-bit pointer (.P), or 32-bit long-word (.L) memory operand to be rotated is specified using general OPR addressing. The operand, *oprmemreg,* can be a data register, a memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. The original carry bit is shifted into the MSB and the LSB is shifted out to the carry bit (C). *It is not appropriate to specify a short-immediate operand with the OPR addressing mode because it is not possible to modify the immediate operand.*

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	Δ

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Cleared

C: Set if the bit shifted out of the LSB of the operand was set before the shift, cleared otherwise.

Detailed Instruction Formats

OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	0	x	x	x	1n
x	L/R=0	1	x	x	1	SIZE (.B, .W, .P, .L)		sb
OPR POSTBYTE (specifes source operand to be rotated)								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1n sb xb                    ROR.bwpl    #oprsxe4i ;not appropriate for destination
1n sb xb                    ROR.bwpl    Di
1n sb xb                    ROR.bwpl    (opru4,xys)
1n sb xb                    ROR.bwpl    {(+-xy)|(xy+-)|(-s)|(s+)}
1n sb xb                    ROR.bwpl    (Di,xys)
1n sb xb                    ROR.bwpl    [Di,xy]
1n sb xb x1                 ROR.bwpl    (oprs9,xysp)
1n sb xb x1                 ROR.bwpl    [oprs9,xysp]
1n sb xb x1                 ROR.bwpl    opru14
1n sb xb x2 x1              ROR.bwpl    (opru18,Di)
1n sb xb x2 x1              ROR.bwpl    opru18
1n sb xb x3 x2 x1           ROR.bwpl    (opr24,xysp)
1n sb xb x3 x2 x1           ROR.bwpl    [opr24,xysp]
1n sb xb x3 x2 x1           ROR.bwpl    (opru24,Di)
1n sb xb x3 x2 x1           ROR.bwpl    opr24
1n sb xb x3 x2 x1           ROR.bwpl    [opr24]
```

Instruction Fields

L/R - This bit selects the rotate direction, left (1) or right (0).

SIZE (.B, .W, .P, .L) - This field specifies 8-bit byte (0b00), 16-bit word (0b01), 24-bit pointer (0b10) or 32-bit long-word (0b11) as the size of the source operand.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

RTI — Return from Interrupt — RTI

Operation

$M_{(SP)} : M_{(SP+1)} \Rightarrow CCH:CCL; (SP) + 2 \Rightarrow SP$
$M_{(SP)} \Rightarrow D0; (SP) + 1 \Rightarrow SP$
$M_{(SP)} \Rightarrow D1; (SP) + 1 \Rightarrow SP$
$M_{(SP)} : M_{(SP+1)} \Rightarrow D2; (SP) + 2 \Rightarrow SP$
$M_{(SP)} : M_{(SP+1)} \Rightarrow D3; (SP) + 2 \Rightarrow SP$
$M_{(SP)} : M_{(SP+1)} \Rightarrow D4; (SP) + 2 \Rightarrow SP$
$M_{(SP)} : M_{(SP+1)} \Rightarrow D5; (SP) + 2 \Rightarrow SP$
$M_{(SP)}: M_{(SP+1)}: M_{(SP+2)} : M_{(SP+3)} \Rightarrow D6; (SP) + 4 \Rightarrow SP$
$M_{(SP)}: M_{(SP+1)}: M_{(SP+2)} : M_{(SP+3)} \Rightarrow D7; (SP) + 4 \Rightarrow SP$
$M_{(SP)}: M_{(SP+1)}: M_{(SP+2)} \Rightarrow X; (SP) + 3 \Rightarrow SP$
$M_{(SP)}: M_{(SP+1)}: M_{(SP+2)} \Rightarrow Y; (SP) + 3 \Rightarrow SP$
$M_{(SP)}: M_{(SP+1)}: M_{(SP+2)} \Rightarrow PC; (SP) + 3 \Rightarrow SP$

Syntax Variations	Addressing Modes
`RTI`	**INH**

Description

Restores system context after exception processing is completed. The condition codes, data registers D0..D7, the pointer registers X and Y, and the PC (return address) are restored to a state pulled from the stack.

If another interrupt is pending when RTI has finished restoring registers from the stack, the SP is adjusted to preserve stack content, and the new vector is fetched.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
⇑	–	–	–	–	Δ	Δ	⇓	–	Δ	Δ	Δ	Δ	Δ

CCR contents are restored from the stack. Unimplemented bits in the CCR can not be changed. Normally RTI is executed from within an interrupt service routine and the MCU is in supervisor state, however it is possible that RTI could be executed from user state due to runaway or a software error. In user state, only the four flag bits N, Z, V, and C can be modified. In supervisor state, any of the implemented CCR bits can be modified however the X bit can never be changed from 0 to 1 by any instruction in any mode.

Detailed Instruction Format

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
1	0	0	1	0	0	0	0	90

`1B 90` `RTI`

RTS — Return from Subroutine — RTS

Operation

$M_{(SP)} : M_{(SP+1)} : M_{(SP+2)} \Rightarrow PC; (SP) + 3 \Rightarrow SP$

Syntax Variations	Addressing Modes
RTS	INH

Description

Restores context at the end of a subroutine. Loads the PC with a 24-bit value pulled from the stack and updates the SP (incremented by 3). Program execution continues at the address restored from the stack.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Format

7	6	5	4	3	2	1	0	
0	0	0	0	0	1	0	1	05

05 RTS

SAT — Saturate — SAT

Operation

saturated(D*i*) ⇒ D*i*

Syntax Variations	Addressing Modes
SAT *Di*	INH

Description

Replace the content of D*i* with its saturated value. The operand is treated as a signed value. Operation size depends on (matches) the size of D*i*.

This instruction uses the information left by a previous operation in the overflow (V-)-flag and the negative (N-)flag to decide whether the content of D*i* is replaced by a value representing the positive or the negative boundary of the signed value range defined by the size of D*i*.

If the overflow (V-)flag is set, the content of D*i* is replaced with the value as defined by the state of negative (N-)flag.

If the negative (N-)flag is set, the value written to D*i* is the maximum positive number of the signed value range. Otherwise (N==0) the minimum negative number of the signed value range is used.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	–

N: Set according to the MSB of the result.

Z: Set if the result is zero. Cleared otherwise.

V: Cleared.

Detailed Instruction Formats

INH

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
1	0	1	0	0	SD REGISTER D*i*			An

1B An SAT *Di*

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* used for the source and destination for the operation (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SBC — Subtract with Borrow — SBC

Operation

(D*i*) – (M) – C ⇒ D*i*

Syntax Variations	Addressing Modes
`SBC Di,#oprimmsz`	**IMM1/2/4**
`SBC Di,oprmemreg`	**OPR/1/2/3**

Description

Subtract with borrow from register D*i* and store the result to D*i*. When the operand is an immediate value, it has the same size as register D*i*. In the case of the general OPR addressing operand, *oprmemreg* can be a sign-extended immediate value (–1, 1, 2, 3..14, 15), a data register, a memory operand the same size as D*i* at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Cleared if the result is non-zero, unchanged otherwise to allow Z to reflect the cumulative result of an extended series if SUB and SBC instructions.

V: Set if a two's complement overflow resulted from the operation. Cleared otherwise.

C: Set if there is a borrow from the MSB of the result. Cleared otherwise.

Detailed Instruction Formats

IMM1/2/4

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	1	1	1	0	SD REGISTER D*i*			7p
IMMEDIATE DATA								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								

```
1B 7p i1                    SBC     Di,#opr8i ;for Di = 8-bit D0 or D1
1B 7p i2 i1                 SBC     Di,#opr16i ;for Di = 16-bit D2, D3, D4, or D5
1B 7p i4 i3 i2 i1           SBC     Di,#opr32i ;for Di = 32-bit D6 or D7
```

OPR/1/2/3

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
1	0	0	0	0	SD REGISTER D*i*			8n
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
8n xb                    SBC     Di,#oprsxe4i ;-1, +1, 2, 3...14, 15
8n xb                    SBC     Di,Dj
8n xb                    SBC     Di,(opru4,xys)
8n xb                    SBC     Di,{(+-xy)|(xy+-)|(-s)|(s+)}
8n xb                    SBC     Di,(Dj,xys)
8n xb                    SBC     Di,[Dj,xy]
8n xb x1                 SBC     Di,(oprs9,xysp)
8n xb x1                 SBC     Di,[oprs9,xysp]
8n xb x1                 SBC     Di,opru14
8n xb x2 x1              SBC     Di,(opru18,Dj)
8n xb x2 x1              SBC     Di,opru18
8n xb x3 x2 x1           SBC     Di,(opr24,xysp)
8n xb x3 x2 x1           SBC     Di,[opr24,xysp]
8n xb x3 x2 x1           SBC     Di,(opru24,Dj)
8n xb x3 x2 x1           SBC     Di,opr24
8n xb x3 x2 x1           SBC     Di,[opr24]
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as a source operand and for the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size of the register D*i*.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

SEC

Set Carry Flag

(Translates to ORCC #$01)

SEC

Operation

$1 \Rightarrow$ C bit

Syntax Variations	Addressing Modes
SEV	**IMM1**

Description

Sets the C status bit. This instruction is assembled as ORCC #$01. The ORCC instruction can be used to set any combination of bits in the CCL in one operation.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	1

C: Set.

Detailed Instruction Formats

IMM1

7	6	5	4	3	2	1	0	
1	1	0	1	1	1	1	0	DE
0	0	0	0	0	0	0	1	01

DE 01 SEV

SEI

Set Interrupt Mask

(Translates to ORCC #$10)

SEI

Operation

1 ⇒ I bit

Syntax Variations	Addressing Modes
SEI	**IMM1**

Description

Sets the I mask bit. This instruction is assembled as ORCC #$10. The ORCC instruction can be used to set any combination of bits in the CCL in one operation.

When the I bit is set, interrupts are disabled.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C	
–	–	–	–	–	–	–	–	–	1	–	–	–	–	supervisor state
–	–	–	–	–	–	–	–	–	–	–	–	–	–	user state

Detailed Instruction Formats

IMM1

7	6	5	4	3	2	1	0	
1	1	0	1	1	1	1	0	DE
0	0	0	1	0	0	0	0	10

DE 10 SEI

SEV

Set Overflow Flag

(Translates to ORCC #$02)

SEV

Operation

1 ⇒ V bit

Syntax Variations	Addressing Modes
SEV	IMM1

Description

Sets the V status bit. This instruction is assembled as ORCC #$02. The ORCC instruction can be used to set any combination of bits in the CCL in one operation.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	1	–

V: Set.

Detailed Instruction Formats

IMM1

7	6	5	4	3	2	1	0	
1	1	0	1	1	1	1	0	DE
0	0	0	0	0	0	1	0	02

DE 02 SEV

SEX — Sign-Extend (smaller CPU register to a larger CPU register) — SEX

Syntax Variations	Addressing Modes
SEX *cpureg*,*cpureg*	INH

Description

Provided the first register is smaller than the second register, it is sign-extended and written to the second register.

If the first register is the same size or larger than the second register, an exchange operation is done. see the EXG instruction.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

In some cases (such as sign-extending D0 to CCW) the sign-extend instruction can cause the contents of another register to be written into the CCR so the CCR effects shown above do not apply. Unused bits in the CCR cannot be changed by any sign-extend or exchange instruction. The X interrupt mask can be cleared by an instruction in supervisor state but cannot be set (changed from 0 to 1) by any sign-extend or exchange instruction. In user state, the X and I interrupt masks cannot be changed by any sign-extend or exchange instruction.

Detailed Instruction Formats

INH

7	6	5	4	3	2	1	0	
1	0	1	0	1	1	1	0	AE
FIRST (SOURCE) REGISTER				SECOND (DESTINATION) REGISTER				eb

AE eb SEX *cpureg*,*cpureg*

Table 6-2. Postbyte (eb) Coding for Sign-Extend Operations

source		D2	D3	D4	D5	D0	D1	D6	D7	X	Y	S	–	CCH	CCL	CCW	
destination		0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
D2	-0	–	D3 ⇔ D2	D4 ⇔ D2	D5 ⇔ D2	sex:D0 ⇒ D2	sex:D1 ⇒ D2	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		sex:CCH ⇒ D2	sex:CCL ⇒ D2	CCW ⇔ D2	
D3	-1	D2 ⇔ D3	–	D4 ⇔ D3	D5 ⇔ D3	sex:D0 ⇒ D3	sex:D1 ⇒ D3	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		sex:CCH ⇒ D3	sex:CCL ⇒ D3	CCW ⇔ D3	
D4	-2	D2 ⇔ D4	D3 ⇔ D4	–	D5 ⇔ D4	sex:D0 ⇒ D4	sex:D1 ⇒ D4	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		sex:CCH ⇒ D4	sex:CCL ⇒ D4	CCW ⇔ D4	
D5	-3	D2 ⇔ D5	D3 ⇔ D5	D4 ⇔ D5	–	sex:D0 ⇒ D5	sex:D1 ⇒ D5	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		sex:CCH ⇒ D5	sex:CCL ⇒ D5	CCW ⇔ D5	
D0	-4	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	–	D1 ⇔ D0	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		CCH ⇔ D0	CCL ⇔ D0	Big ⇔Small	
D1	-5	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	D0 ⇔ D1	–	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		CCH ⇔ D1	CCL ⇔ D1	Big ⇔Small	
D6	-6	sex:D2 ⇒ D6	sex:D3 ⇒ D6	sex:D4 ⇒ D6	sex:D5 ⇒ D6	sex:D0 ⇒ D6	sex:D1 ⇒ D6	–	D7 ⇔ D6	sex:X ⇒ D6	sex:Y ⇒ D6	sex:S ⇒ D6		sex:CCH ⇒ D6	sex:CCL ⇒ D6	sex:CCW ⇒ D6	
D7	-7	sex:D2 ⇒ D7	sex:D3 ⇒ D7	sex:D4 ⇒ D7	sex:D5 ⇒ D7	sex:D0 ⇒ D7	sex:D1 ⇒ D7	D6 ⇔ D7	–	sex:X ⇒ D7	sex:Y ⇒ D7	sex:S ⇒ D7		sex:CCH ⇒ D7	sex:CCL ⇒ D7	sex:CCW ⇒ D7	
X	-8	sex:D2 ⇒ X	sex:D3 ⇒ X	sex:D4 ⇒ X	sex:D5 ⇒ X	sex:D0 ⇒ X	sex:D1 ⇒ X	Big ⇔Small	Big ⇔Small	–	Y ⇔ X	S ⇔ X		sex:CCH ⇒ X	sex:CCL ⇒ X	sex:CCW ⇒ X	
Y	-9	sex:D2 ⇒ Y	sex:D3 ⇒ Y	sex:D4 ⇒ Y	sex:D5 ⇒ Y	sex:D0 ⇒ Y	sex:D1 ⇒ Y	Big ⇔Small	Big ⇔Small	X ⇔ Y	–	S ⇔ Y		sex:CCH ⇒ Y	sex:CCL ⇒ Y	sex:CCW ⇒ Y	
S	-A	sex:D2 ⇒ S	sex:D3 ⇒ S	sex:D4 ⇒ S	sex:D5 ⇒ S	sex:D0 ⇒ S	sex:D1 ⇒ S	Big ⇔Small	Big ⇔Small	X ⇔ S	Y ⇔ S	–		sex:CCH ⇒ S	sex:CCL ⇒ S	sex:CCW ⇒ S	
reserved	-B																
CCH	-C	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	D0 ⇔ CCH	D1 ⇔ CCH	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		–	CCL ⇔ CCH	NOP	
CCL	-D	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	D0 ⇔ CCL	D1 ⇔ CCL	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		CCH ⇔ CCL	–	NOP	
CCW	-E	D2 ⇔ CCW	D3 ⇔ CCW	D4 ⇔ CCW	D5 ⇔ CCW	sex:D0 ⇒ CCW	sex:D1 ⇒ CCW	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		sex:CCH ⇒ CCW	sex:CCL ⇒ CCW	–	
	-F																–

EXG Big,Small: Small register gets low part of Big register, Big register gets sign-extended Small register. These cases are not expected to be useful in application programs.
EXG CCW,CCH and EXG CCW,CCL are ambiguous cases so CCW is not changed (equivalent to NOP)

SPARE *Unimplemented Page1 Opcode Trap* SPARE

Operation

(SP) – 3 ⇒ SP; RTN[23:0] ⇒ $M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
(SP) – 3 ⇒ SP; Y ⇒ $M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
(SP) – 3 ⇒ SP; X ⇒ $M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
(SP) – 4 ⇒ SP; D7 ⇒ $M_{(SP)} : M_{(SP+1)} : M_{(SP+2)} : M_{(SP+3)}$
(SP) – 4 ⇒ SP; D6 ⇒ $M_{(SP)} : M_{(SP+1)} : M_{(SP+2)} : M_{(SP+3)}$
(SP) – 2 ⇒ SP; D5 ⇒ $M_{(SP)} : M_{(SP+1)}$
(SP) – 2 ⇒ SP; D4 ⇒ $M_{(SP)} : M_{(SP+1)}$
(SP) – 2 ⇒ SP; D3 ⇒ $M_{(SP)} : M_{(SP+1)}$
(SP) – 2 ⇒ SP; D2 ⇒ $M_{(SP)} : M_{(SP+1)}$
(SP) – 1 ⇒ SP; D1 ⇒ $M_{(SP)}$
(SP) – 1 ⇒ SP; D0 ⇒ $M_{(SP)}$
(SP) – 2 ⇒ SP; CCH:CCL ⇒ $M_{(SP)} : M_{(SP+1)}$
0 ⇒ U; 1 ⇒ I; (Page 1 TRAP Vector) ⇒ PC

Syntax Variations	Addressing Modes
`not a user instruction`	-

Description

This instruction mnemonic is used as a placeholder for the unimplemented opcodes on page 1 of the opcode map. If any of these unimplemented opcodes is encountered in an application program, the CPU context is saved on the stack as in an SWI instruction and program execution continues at the address specified in the Page1 TRAP Vector.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
0	–	–	–	–	–	–	–	–	1	–	–	–	–

U: Cleared.

I: Set.

Detailed Instruction Format

7	6	5	4	3	2	1	0	
1	1	1	0	1	1	1	1	EF

At this time, the one unimplemented opcode on Page 1 of the opcode map are 0xEF. It is expected that some of these codes will be used for additional instructions in the final version of this instruction set.

ST — Store (D*i*, X, Y, or SP) — ST

Operation

(D*i*) ⇒ M

(X) ⇒ M

(Y) ⇒ M

(SP) ⇒ M

Syntax Variations	Addressing Modes
`ST Di,opr24a`	**EXT** (24-bit address)
`ST Di,oprmemreg`	**OPR/1/2/3**
`ST xy,opr24a`	**EXT** (24-bit address)
`ST xy,oprmemreg`	**OPR/1/2/3**
`ST S,oprmemreg`	**OPR/1/2/3**

Description

Store a register D*i*, X, Y, or SP to a memory location. In the case of the general OPR addressing operand, *oprmemreg* can be a data register, a memory operand the same size as D*i* , X, Y, or SP at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. There are also efficient 24-bit extended addressing mode versions of the instructions to store D*i*, X or Y. *It is inappropriate to specify a short immediate operand using the OPR addressing mode for this instruction because it is not possible to modify the immediate operand.*

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	0	–

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Cleared.

Detailed Instruction Formats

EXT (D*i*)

7	6	5	4	3	2	1	0	
1	1	0	1	0	REGISTER			Dn
ADDRESS[23:16]								
ADDRESS[15:8]								
ADDRESS[7:0]								

`Dn a3 a2 a1` `ST` `Di,opr24a`

OPR/1/2/3 (D*i*)

7	6	5	4	3	2	1	0	
1	1	0	0	0	REGISTER			Cn
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
Cn xb                    ST      Di,#oprsxe4i ;not appropriate for a destination
Cn xb                    ST      Di,Dj
Cn xb                    ST      Di,(opru4,xys)
Cn xb                    ST      Di,{(+-xy)|(xy+-)|(-s)|(s+)}
Cn xb                    ST      Di,(Dj,xys)
Cn xb                    ST      Di,[Dj,xy]
Cn xb x1                 ST      Di,(oprs9,xysp)
Cn xb x1                 ST      Di,[oprs9,xysp]
Cn xb x1                 ST      Di,opru14
Cn xb x2 x1              ST      Di,(opru18,Dj)
Cn xb x2 x1              ST      Di,opru18
Cn xb x3 x2 x1           ST      Di,(opr24,xysp)
Cn xb x3 x2 x1           ST      Di,[opr24,xysp]
Cn xb x3 x2 x1           ST      Di,(opru24,Dj)
Cn xb x3 x2 x1           ST      Di,opr24
Cn xb x3 x2 x1           ST      Di,[opr24]
```

EXT (X or Y)

7	6	5	4	3	2	1	0	
1	1	0	1	1	0	0	Y/X	Dp
ADDRESS[23:16]								
ADDRESS[15:8]								
ADDRESS[7:0]								

```
Dp a3 a2 a1              ST      xy,opr24a
```

OPR/1/2/3 (X or Y)

7	6	5	4	3	2	1	0	
1	1	0	0	1	0	0	Y/X	Cp
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
Cp xb                    ST      xy,#oprsxe4i ;not appropriate for a destination
Cp xb                    ST      xy,Dj
Cp xb                    ST      xy,(opru4,xys)
Cp xb                    ST      xy,{(+-xy)|(xy+-)|(-s)|(s+)}
Cp xb                    ST      xy,(Dj,xys)
Cp xb                    ST      xy,[Dj,xy]
Cp xb x1                 ST      xy,(oprs9,xysp)
Cp xb x1                 ST      xy,[oprs9,xysp]
Cp xb x1                 ST      xy,opru14
Cp xb x2 x1              ST      xy,(opru18,Dj)
Cp xb x2 x1              ST      xy,opru18
Cp xb x3 x2 x1           ST      xy,(opr24,xysp)
```

```
Cp xb x3 x2 x1                    ST          xy,[opr24,xysp]
Cp xb x3 x2 x1                    ST          xy,(opru24,Dj)
Cp xb x3 x2 x1                    ST          xy,opr24
Cp xb x3 x2 x1                    ST          xy,[opr24]
```

OPR/1/2/3 (SP)

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	0	0	0	1	01
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
1B 01 xb                          ST          S,#oprsxe4i ;not appropriate for a destination
1B 01 xb                          ST          S,Dj ;suggest using more efficient TFR
1B 01 xb                          ST          S,(opru4,xys)
1B 01 xb                          ST          S,{(+-xy)|(xy+-)|(-s)|(s+)}
1B 01 xb                          ST          S,(Dj,xys)
1B 01 xb                          ST          S,[Dj,xy]
1B 01 xb x1                       ST          S,(oprs9,xysp)
1B 01 xb x1                       ST          S,[oprs9,xysp]
1B 01 xb x1                       ST          S,opru14
1B 01 xb x2 x1                    ST          S,(opru18,Dj)
1B 01 xb x2 x1                    ST          S,opru18
1B 01 xb x3 x2 x1                 ST          S,(opr24,xysp)
1B 01 xb x3 x2 x1                 ST          S,[opr24,xysp]
1B 01 xb x3 x2 x1                 ST          S,(opru24,Dj)
1B 01 xb x3 x2 x1                 ST          S,opr24
1B 01 xb x3 x2 x1                 ST          S,[opr24]
```

Instruction Fields

REGISTER - This field specifies the number of the data register D*i* which is used as the source register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

Y/X - This field selects either the X index register or the Y index register.

ADDRESS - This field is used for address bits used for extended addressing mode.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. *The short-immediate variation is not appropriate for a store instruction.*

STOP

Stop Processing

(if enabled by S bit in CCR = 0)

STOP

Operation

If S bit =1, treat STOP as a NOP; else if S=0;

$(SP) - 3 \Rightarrow SP;\ RTN[23:0] \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
$(SP) - 3 \Rightarrow SP;\ Y \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
$(SP) - 3 \Rightarrow SP;\ X \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
$(SP) - 4 \Rightarrow SP;\ D7 \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)} : M_{(SP+3)}$
$(SP) - 4 \Rightarrow SP;\ D6 \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)} : M_{(SP+3)}$
$(SP) - 2 \Rightarrow SP;\ D5 \Rightarrow M_{(SP)} : M_{(SP+1)}$
$(SP) - 2 \Rightarrow SP;\ D4 \Rightarrow M_{(SP)} : M_{(SP+1)}$
$(SP) - 2 \Rightarrow SP;\ D3 \Rightarrow M_{(SP)} : M_{(SP+1)}$
$(SP) - 2 \Rightarrow SP;\ D2 \Rightarrow M_{(SP)} : M_{(SP+1)}$
$(SP) - 1 \Rightarrow SP;\ D1 \Rightarrow M_{(SP)}$
$(SP) - 1 \Rightarrow SP;\ D0 \Rightarrow M_{(SP)}$
$(SP) - 2 \Rightarrow SP;\ CCW \Rightarrow M_{(SP)} : M_{(SP+1)}$
Stop system clocks

Complete instruction and resume processing at next reset or enabled interrupt.

Syntax Variations	Addressing Modes
`STOP`	**INH**

Description

If the CPU is in user state or if the S control bit in the CCR is set, STOP acts like a NOP instruction. If the CPU is in supervisor state and the S bit is cleared, STOP stacks the CPU context, stops system clocks, and puts the device in a standby mode. Standby operation minimizes system power consumption. The contents of registers and the states of I/O pins remain unchanged.

Asserting $\overline{RESET}$, $\overline{XIRQ}$, or $\overline{IRQ}$ signals (if enabled) ends the standby mode. Stacking on entry to STOP allows the CPU to recover quickly when an interrupt is used, provided a stable clock is present.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Format

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	0	1	0	1	05

`1B 05` `STOP`

SUB — Subtract without Borrow — SUB

Operation

(D*i*) – (M) ⇒ D*i*

(X) – (Y) ⇒ D6

(Y) – (X) ⇒ D6

Syntax Variations	Addressing Modes
SUB *Di*,#*oprimmsz*	**IMM1/2/4**
SUB *Di*,*oprmemreg*	**OPR/1/2/3**
SUB *D6*,*X*,*Y*	**INH**
SUB *D6*,*Y*,*X*	**INH**

Description

Subtract without borrow from register D*i* and store the result to D*i*, or Subtract X–Y or Y–X and store the result to D6. When the operand is an immediate value, it has the same size as register D*i*. In the case of the general OPR addressing operand, *oprmemreg* can be a sign-extended immediate value (–1, 1, 2, 3..14, 15), a data register, a memory operand the same size as D*i* at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode.

In the case of SUB D6,X,Y or SUB D6,Y,X source operands X and Y are treated as unsigned and the result is a signed long int.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	Δ	Δ	Δ	Δ

N: Set if the MSB of the result is set. Cleared otherwise.

Z: Set if the result is zero. Cleared otherwise.

V: Set if a two's complement overflow resulted from the operation. Cleared otherwise.

C: Set if there is a borrow from the MSB of the result. Cleared otherwise.

Detailed Instruction Formats

INH

7	6	5	4	3	2	1	0	
1	1	1	1	1	1	0	1	FD

FD SUB *D6*,X,Y

7	6	5	4	3	2	1	0	
1	1	1	1	1	1	1	0	FE

FE SUB *D6*,Y,X

IMM1/2/4

7	6	5	4	3	2	1	0	
0	1	1	1	0	SD REGISTER D*i*			7p
IMMEDIATE DATA								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								
(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D*i*)								

```
7p i1                    SUB        Di,#opr8i ;for Di = 8-bit D0 or D1
7p i2 i1                 SUB        Di,#opr16i ;for Di = 16-bit D2, D3, D4, or D5
7p i4 i3 i2 i1           SUB        Di,#opr32i ;for Di = 32-bit D6 or D7
```

OPR/1/2/3

7	6	5	4	3	2	1	0	
1	0	0	0	0	SD REGISTER D*i*			8n
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								

```
8n xb                    SUB        Di,#oprsxe4i ;-1, +1, 2, 3...14, 15
8n xb                    SUB        Di,Dj
8n xb                    SUB        Di,(opru4,xys)
8n xb                    SUB        Di,{(+-xy)|(xy+-)|(-s)|(s+)}
8n xb                    SUB        Di,(Dj,xys)
8n xb                    SUB        Di,[Dj,xy]
8n xb x1                 SUB        Di,(oprs9,xysp)
8n xb x1                 SUB        Di,[oprs9,xysp]
8n xb x1                 SUB        Di,opru14
8n xb x2 x1              SUB        Di,(opru18,Dj)
8n xb x2 x1              SUB        Di,opru18
8n xb x3 x2 x1           SUB        Di,(opr24,xysp)
8n xb x3 x2 x1           SUB        Di,[opr24,xysp]
8n xb x3 x2 x1           SUB        Di,(opru24,Dj)
8n xb x3 x2 x1           SUB        Di,opr24
8n xb x3 x2 x1           SUB        Di,[opr24]
```

Instruction Fields

SD REGISTER D*i* - This field specifies the number of the data register D*i* which is used as a source operand and for the destination register (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

IMMEDIATE and OPTIONAL IMMEDIATE DATA - These fields contain the immediate operand. This operand is either 1 byte, 2 bytes or 4 bytes wide, depending on the size of the register D*i*.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location.

SWI — Software Interrupt — SWI

Operation

$(SP) - 3 \Rightarrow SP;\ RTN[23:0] \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
$(SP) - 3 \Rightarrow SP;\ Y \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
$(SP) - 3 \Rightarrow SP;\ X \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
$(SP) - 4 \Rightarrow SP;\ D7 \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)} : M_{(SP+3)}$
$(SP) - 4 \Rightarrow SP;\ D6 \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)} : M_{(SP+3)}$
$(SP) - 2 \Rightarrow SP;\ D5 \Rightarrow M_{(SP)} : M_{(SP+1)}$
$(SP) - 2 \Rightarrow SP;\ D4 \Rightarrow M_{(SP)} : M_{(SP+1)}$
$(SP) - 2 \Rightarrow SP;\ D3 \Rightarrow M_{(SP)} : M_{(SP+1)}$
$(SP) - 2 \Rightarrow SP;\ D2 \Rightarrow M_{(SP)} : M_{(SP+1)}$
$(SP) - 1 \Rightarrow SP;\ D1 \Rightarrow M_{(SP)}$
$(SP) - 1 \Rightarrow SP;\ D0 \Rightarrow M_{(SP)}$
$(SP) - 2 \Rightarrow SP;\ CCH{:}CCL \Rightarrow M_{(SP)} : M_{(SP+1)}$
$0 \Rightarrow U;\ 1 \Rightarrow I;\ (\text{SWI vector}) \Rightarrow PC$

Syntax Variations	Addressing Modes
`SWI`	**INH**

Description

Causes an interrupt without an external interrupt service request. Uses the address of the next instruction after the SWI as a return address. Stacks the CPU context, then sets the I mask and clears the U bit to change to supervisor state. SWI is not affected by the state of the I interrupt mask (SWI interrupts cannot be blocked by the interrupt mask).

Because the opcode for SWI is 0xFF, if the CPU encounters an uninitialized area of memory that reads 0xFF, an SWI instruction will be performed.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
0	–	–	–	–	–	–	–	–	1	–	–	–	–

U: Cleared.

I: Set.

Detailed Instruction Format

7	6	5	4	3	2	1	0	
1	1	1	1	1	1	1	1	FF
FF			SWI					

SYS — System Call Software Interrupt — SYS

Operation

$(SP) - 3 \Rightarrow SP; RTN[23:0] \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
$(SP) - 3 \Rightarrow SP; Y \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
$(SP) - 3 \Rightarrow SP; X \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
$(SP) - 4 \Rightarrow SP; D7 \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)} : M_{(SP+3)}$
$(SP) - 4 \Rightarrow SP; D6 \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)} : M_{(SP+3)}$
$(SP) - 2 \Rightarrow SP; D5 \Rightarrow M_{(SP)} : M_{(SP+1)}$
$(SP) - 2 \Rightarrow SP; D4 \Rightarrow M_{(SP)} : M_{(SP+1)}$
$(SP) - 2 \Rightarrow SP; D3 \Rightarrow M_{(SP)} : M_{(SP+1)}$
$(SP) - 2 \Rightarrow SP; D2 \Rightarrow M_{(SP)} : M_{(SP+1)}$
$(SP) - 1 \Rightarrow SP; D1 \Rightarrow M_{(SP)}$
$(SP) - 1 \Rightarrow SP; D0 \Rightarrow M_{(SP)}$
$(SP) - 2 \Rightarrow SP; CCH:CCL \Rightarrow M_{(SP)} : M_{(SP+1)}$
$0 \Rightarrow U; 1 \Rightarrow I; (SYS\ vector) \Rightarrow PC$

Syntax Variations	Addressing Modes
`SYS`	**INH**

Description

Enter System operating state. Similar to SWI except the SYS Vector is used instead of the SWI vector. Uses the address of the next instruction after the SYS as a return address. Stacks the CPU context, then sets the I mask and clears the U bit to change to supervisor state. SYS is not affected by the state of the I interrupt mask (SYS interrupts cannot be blocked by the interrupt mask).

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
0	–	–	–	–	–	–	–	–	1	–	–	–	–

U: Cleared.

I: Set.

Detailed Instruction Format

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	0	1	1	1	07

`1B 07` `SYS`

TBcc — Test and Branch — TBcc

Operation

(D*i*) – 0 ⇒ D*i*; then Branch if (condition) true

(X) – 0 ⇒ X; then Branch if (condition) true

(Y) – 0 ⇒ Y; then Branch if (condition) true

(M) – 0 ⇒ Y; then Branch if (condition) true

Condition may be...

NE (Z=0), EQ (Z=1), PL (N=0), MI (N=1), GT (Z | N=0), or LE (Z | N=1)

Syntax Variations	Addressing Modes
`TBcc Di,oprdest`	**REG-REL**
`TBcc X,oprdest`	**REG-REL**
`TBcc Y,oprdest`	**REG-REL**
`TBcc.bwploprmemreg,oprdest`	**OPR/1/2/3-REL**

Description

Test the operand (internally determining the N and Z conditions but not modifying the CCR) then branch if the specified condition is true. The condition (cc) can be NE (not equal), EQ (equal), PL (plus), MI (minus), GT (greater than), or LE (less than or equal). The operand may be one of the eight data registers, index register X, index register Y, or an 8-, 16-, 24-, or 32-bit memory operand. In the case of the general OPR addressing operand, *oprmemreg* can be a data register, a memory operand at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode. The relative offset for the branch can be either 7 bits (–64 to +63) or 15 bits (~+/–16K) displacement from the TBcc opcode location.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Formats

REG-REL (D*i*)

7	6	5	4	3	2	1	0	
0	0	0	0	1	1	0	1	0B
0	CC (NE,EQ,PL,MI,GT,LE,–,–)			0	REGISTER D*i*			lb
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
0B lb rb          TBcc    Di,oprdest ;destination within -64..+63 (7-bit)
0B lb rb r1       TBcc    Di,oprdest ;destination within ~+/-16k (15-bit)
```

REG-REL (X, Y)

7	6	5	4	3	2	1	0	
0	0	0	0	1	1	0	1	0B
0	CC (NE,EQ,PL,MI,GT,LE,–,–)			1	0	don't care	Y/X	lb
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
0B lb rb          TBcc    X,oprdest ;destination within -64..+63 (7-bit)
0B lb rb r1       TBcc    X,oprdest ;destination within ~+/-16k (15-bit)
0B lb rb          TBcc    Y,oprdest ;destination within -64..+63 (7-bit)
0B lb rb r1       TBcc    Y,oprdest ;destination within ~+/-16k (15-bit)
```

OPR/1/2/3-REL

7	6	5	4	3	2	1	0	
0	0	0	0	1	1	0	1	0B
0	CC (NE,EQ,PL,MI,GT,LE,–,–)			1	1	SIZE (.B, .W, .P, .L)		lb
OPR POSTBYTE								xb
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)								
REL_SIZE	7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)							rb
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)								r1

```
0B lb xb rb                TBcc.bwpl   #oprsxe4i,oprdest
0B lb xb rb r1             TBcc.bwpl   #oprsxe4i,oprdest
0B lb xb rb                TBcc.bwpl   Di,oprdest ;see efficient REG-REL version
0B lb xb rb r1             TBcc.bwpl   Di,oprdest ;see efficient REG-REL version
0B lb xb rb                TBcc.bwpl   (opru4,xys),oprdest ;(7-bit)
0B lb xb rb r1             TBcc.bwpl   (opru4,xys),oprdest ;(15-bit)
0B lb xb rb                TBcc.bwpl   {(+-xy)|(xy+-)|(-s)|(s+)},oprdest ;(7-bit)
0B lb xb rb r1             TBcc.bwpl   {(+-xy)|(xy+-)|(-s)|(s+)},oprdest ;(15-bit)
0B lb xb rb                TBcc.bwpl   (Di,xys),oprdest ;(7-bit)
0B lb xb rb r1             TBcc.bwpl   (Di,xys),oprdest ;(15-bit)
0B lb xb rb                TBcc.bwpl   [Di,xy],oprdest ;(7-bit)
0B lb xb rb r1             TBcc.bwpl   [Di,xy],oprdest ;(15-bit)
0B lb xb x1 rb             TBcc.bwpl   (oprs9,xysp),oprdest ;(7-bit)
0B lb xb x1 rb r1          TBcc.bwpl   (oprs9,xysp),oprdest ;(15-bit)
0B lb xb x1 rb             TBcc.bwpl   [oprs9,xysp],oprdest ;(7-bit)
0B lb xb x1 rb r1          TBcc.bwpl   [oprs9,xysp],oprdest ;(15-bit)
0B lb xb x1 rb             TBcc.bwpl   opru14,oprdest ;(7-bit)
0B lb xb x1 rb r1          TBcc.bwpl   opru14,oprdest ;(15-bit)
0B lb xb x2 x1 rb          TBcc.bwpl   (opru18,Di),oprdest ;(7-bit)
0B lb xb x2 x1 rb r1       TBcc.bwpl   (opru18,Di),oprdest ;(15-bit)
0B lb xb x2 x1 rb          TBcc.bwpl   opru18,oprdest ;(7-bit)
0B lb xb x2 x1 rb r1       TBcc.bwpl   opru18,oprdest ;(15-bit)
0B lb xb x3 x2 x1 rb       TBcc.bwpl   (opr24,xysp),oprdest ;(7-bit)
0B lb xb x3 x2 x1 rb r1    TBcc.bwpl   (opr24,xysp),oprdest ;(15-bit)
0B lb xb x3 x2 x1 rb       TBcc.bwpl   [opr24,xysp],oprdest ;(7-bit)
0B lb xb x3 x2 x1 rb r1    TBcc.bwpl   [opr24,xysp],oprdest ;(15-bit)
0B lb xb x3 x2 x1 rb       TBcc.bwpl   (opru24,Di),oprdest ;(7-bit)
0B lb xb x3 x2 x1 rb r1    TBcc.bwpl   (opru24,Di),oprdest ;(15-bit)
0B lb xb x3 x2 x1 rb       TBcc.bwpl   opr24,oprdest ;(7-bit)
0B lb xb x3 x2 x1 rb r1    TBcc.bwpl   opr24,oprdest ;(15-bit)
```

```
0B lb xb x3 x2 x1 rb          TBcc.bwpl   [opr24],oprdest ;(7-bit)
0B lb xb x3 x2 x1 rb r1       TBcc.bwpl   [opr24],oprdest ;(15-bit)
```

Instruction Fields

CC - This field specifies the condition for the branch according to the table below:

Test	Mnemonic	Condition	Boolean
NE; r≠0	TBNE	000	Z = 0
EQ; r=0	TBEQ	001	Z = 1
PL; r≥0	TBPL	010	N = 0
MI; r<0	TBMI	011	N = 1
GT; r>0	TBGT	100	Z | N = 0
LE; r≤0	TBLE	101	Z | N = 1
reserved (Test and Branch Never)		110	–
		111	

REGISTER - This field specifies the number of the data register D*i* which is used as the source operand (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7).

SIZE - This field specifies 8-bit byte (0b00), 16-bit word (0b01), 24-bit pointer (0b10) or 32-bit long-word (0b11) as the size of the operation.

Y/X - This field specifies either index register X (0) or index register Y (1) as the source operand.

OPR POSTBYTE and the associated 0, 1, 2, or 3 optional extension byte(s) specify an operand according to the rules for the xb postbyte. This operand may be a short-immediate value, a data register, a 14- 18- or 24-bit extended memory address, or an indexed or indexed-indirect memory location. *Using OPR addressing mode to specify a register operand, performs the same function as the REG-REL versions but is less efficient.*

REL_SIZE - This field specifies the size of the DISPLACEMENT 0=7-bit; 1=15=bit .
DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

TFR — Transfer Register Contents — TFR

Syntax Variations	Addressing Modes
`TFR` *cpureg,cpureg*	INH

Description

Transfer (copy) the contents of one CPU register to another CPU register.

If both registers are the same size, a direct transfer is performed.

If the first register is larger than the second register, only the low portion is transferred (truncate).

If the first register is smaller than the second register, it is zero-extended and written to the second register.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

In some cases (such as transferring D0 to CCL) the transfer instruction can cause the contents of another register to be written into the CCR so the CCR effects shown above do not apply. Unused bits in the CCR cannot be changed by any transfer instruction. The X interrupt mask can be cleared by an instruction in supervisor state but cannot be set (changed from 0 to 1) by any transfer instruction. In user state, the X and I interrupt masks cannot be changed by any transfer instruction.

Detailed Instruction Formats

INH

7	6	5	4	3	2	1	0	
1	0	0	1	1	1	1	0	9E
FIRST (SOURCE) REGISTER				SECOND (DESTINATION) REGISTER				tb

`9E tb` `TFR` *cpureg,cpureg*

Table 6-3. Transfer Postbyte (tb) Coding Map

	source	D2	D3	D4	D5	D0	D1	D6	D7	X	Y	S	–	CCH	CCL	CCW	
destination		0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
D2	-0	–	D3 ⇒ D2	D4 ⇒ D2	D5 ⇒ D2	00:D0 ⇒ D2	00:D1 ⇒ D2	D6L ⇒ D2	D7L ⇒ D2	XL ⇒ D2	YL ⇒ D2	SL ⇒ D2		00:CCH ⇒ D2	00:CCL ⇒ D2	CCW ⇒ D2	
D3	-1	D2 ⇒ D3	–	D4 ⇒ D3	D5 ⇒ D3	00:D0 ⇒ D3	00:D1 ⇒ D3	D6L ⇒ D3	D7L ⇒ D3	XL ⇒ D3	YL ⇒ D3	SL ⇒ D3		00:CCH ⇒ D3	00:CCL ⇒ D3	CCW ⇒ D3	
D4	-2	D2 ⇒ D4	D3 ⇒ D4	–	D5 ⇒ D4	00:D0 ⇒ D4	00:D1 ⇒ D4	D6L ⇒ D4	D7L ⇒ D4	XL ⇒ D4	YL ⇒ D4	SL ⇒ D4		00:CCH ⇒ D4	00:CCL ⇒ D4	CCW ⇒ D4	
D5	-3	D2 ⇒ D5	D3 ⇒ D5	D4 ⇒ D5	–	00:D0 ⇒ D5	00:D1 ⇒ D5	D6L ⇒ D5	D7L ⇒ D5	XL ⇒ D5	YL ⇒ D5	SL ⇒ D5		00:CCH ⇒ D5	00:CCL ⇒ D5	CCW ⇒ D5	
D0	-4	D2L ⇒ D0	D3L ⇒ D0	D4L ⇒ D0	D5L ⇒ D0	–	D1 ⇒ D0	D6L ⇒ D0	D7L ⇒ D0	XL ⇒ D0	YL ⇒ D0	SL ⇒ D0		CCH ⇒ D0	CCL ⇒ D0	CCL ⇒ D0	
D1	-5	D2L ⇒ D1	D3L ⇒ D1	D4L ⇒ D1	D5L ⇒ D1	D0 ⇒ D1	–	D6L ⇒ D1	D7L ⇒ D1	XL ⇒ D1	YL ⇒ D1	SL ⇒ D1		CCH ⇒ D1	CCL ⇒ D1	CCL ⇒ D1	
D6	-6	0000:D2 ⇒ D6	0000:D3 ⇒ D6	0000:D4 ⇒ D6	0000:D5 ⇒ D6	000000:D0 ⇒ D6	000000:D1 ⇒ D6	–	D7 ⇒ D6	00:X ⇒ D6	00:Y ⇒ D6	00:SP ⇒ D6		000000:CCH ⇒ D6	000000:CCL ⇒ D6	0000:CCW ⇒ D6	
D7	-7	0000:D2 ⇒ D7	0000:D3 ⇒ D7	0000:D4 ⇒ D7	0000:D5 ⇒ D7	000000:D0 ⇒ D7	000000:D1 ⇒ D7	D6 ⇒ D7	–	00:X ⇒ D7	00:Y ⇒ D7	00:SP ⇒ D7		000000:CCH ⇒ D7	000000:CCL ⇒ D7	0000:CCW ⇒ D7	
X	-8	00:D2 ⇒ X	00:D3 ⇒ X	00:D4 ⇒ X	00:D5 ⇒ X	0000:D0 ⇒ X	0000:D1 ⇒ X	D6L ⇒ X	D7L ⇒ X	–	Y ⇒ X	S ⇒ X		0000:CCH ⇒ X	0000:CCL ⇒ X	00:CCW ⇒ X	
Y	-9	00:D2 ⇒ Y	00:D3 ⇒ Y	00:D4 ⇒ Y	00:D5 ⇒ Y	0000:D0 ⇒ Y	0000:D1 ⇒ Y	D6L ⇒ Y	D7L ⇒ Y	X ⇒ Y	–	S ⇒ Y		0000:CCH ⇒ Y	0000:CCL ⇒ Y	00:CCW ⇒ Y	
S	-A	00:D2 ⇒ S	00:D3 ⇒ S	00:D4 ⇒ S	00:D5 ⇒ S	0000:D0 ⇒ S	0000:D1 ⇒ S	D6L ⇒ S	D7L ⇒ S	X ⇒ S	Y ⇒ S	–		0000:CCH ⇒ S	0000:CCL ⇒ S	00:CCW ⇒ S	
reserved	-B																
CCH	-C	D2L ⇒ CCH	D3L ⇒ CCH	D4L ⇒ CCH	D5L ⇒ CCH	D0 ⇒ CCH	D1 ⇒ CCH	D6L ⇒ CCH	D7L ⇒ CCH	XL ⇒ CCH	YL ⇒ CCH	SL ⇒ CCH		–	CCL ⇒ CCH	CCL ⇒ CCH	
CCL	-D	D2L ⇒ CCL	D3L ⇒ CCL	D4L ⇒ CCL	D5L ⇒ CCL	D0 ⇒ CCL	D1 ⇒ CLL	D6L ⇒ CCL	D7L ⇒ CCL	XL ⇒ CCL	YL ⇒ CCL	SL ⇒ CCL		CCH ⇒ CCL	–	CCL ⇒ CCL	
CCW	-E	D2 ⇒ CCW	D3 ⇒ CCW	D4 ⇒ CCW	D5 ⇒ CCW	00:D0 ⇒ CCW	00:D1 ⇒ CCW	D6L ⇒ CCW	D7L ⇒ CCW	XL ⇒ CCW	YL ⇒ CCW	SL ⇒ CCW		00:CCH ⇒ CCW	00:CCL ⇒ CCW	–	
	-F																–

TRAP — Unimplemented Page2 Opcode Trap — TRAP

Operation

(SP) – 3 ⇒ SP; RTN[23:0] ⇒ $M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
(SP) – 3 ⇒ SP; Y ⇒ $M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
(SP) – 3 ⇒ SP; X ⇒ $M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
(SP) – 4 ⇒ SP; D7 ⇒ $M_{(SP)} : M_{(SP+1)} : M_{(SP+2)} : M_{(SP+3)}$
(SP) – 4 ⇒ SP; D6 ⇒ $M_{(SP)} : M_{(SP+1)} : M_{(SP+2)} : M_{(SP+3)}$
(SP) – 2 ⇒ SP; D5 ⇒ $M_{(SP)} : M_{(SP+1)}$
(SP) – 2 ⇒ SP; D4 ⇒ $M_{(SP)} : M_{(SP+1)}$
(SP) – 2 ⇒ SP; D3 ⇒ $M_{(SP)} : M_{(SP+1)}$
(SP) – 2 ⇒ SP; D2 ⇒ $M_{(SP)} : M_{(SP+1)}$
(SP) – 1 ⇒ SP; D1 ⇒ $M_{(SP)}$
(SP) – 1 ⇒ SP; D0 ⇒ $M_{(SP)}$
(SP) – 2 ⇒ SP; CCH:CCL ⇒ $M_{(SP)} : M_{(SP+1)}$
0 ⇒ U; 1 ⇒ I; (Page 2 TRAP Vector) ⇒ PC

Syntax Variations	Addressing Modes
`TRAP #trapnum`	**INH**

Description

This instruction mnemonic is used for the unimplemented opcodes on page 2 of the opcode map. If any of these unimplemented opcodes is encountered in an application program, the CPU context is saved on the stack as in an SWI instruction and program execution continues at the address specified in the Page 2 TRAP Vector.

These opcodes and the TRAP ISR can be used to extend the instruction set with software routines.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
0	–	–	–	–	–	–	–	–	1	–	–	–	–

U: Cleared.

I: Set.

Detailed Instruction Format

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
TRAP NUMBER								tn

`1B tn` `TRAP` *tn*

Refer to the opcode map to identify unimplemented page 2 opcodes.

WAI — Wait for Interrupt — WAI

Operation

$(SP) - 3 \Rightarrow SP;\ RTN[23:0] \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
$(SP) - 3 \Rightarrow SP;\ Y \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
$(SP) - 3 \Rightarrow SP;\ X \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)}$
$(SP) - 4 \Rightarrow SP;\ D7 \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)} : M_{(SP+3)}$
$(SP) - 4 \Rightarrow SP;\ D6 \Rightarrow M_{(SP)} : M_{(SP+1)} : M_{(SP+2)} : M_{(SP+3)}$
$(SP) - 2 \Rightarrow SP;\ D5 \Rightarrow M_{(SP)} : M_{(SP+1)}$
$(SP) - 2 \Rightarrow SP;\ D4 \Rightarrow M_{(SP)} : M_{(SP+1)}$
$(SP) - 2 \Rightarrow SP;\ D3 \Rightarrow M_{(SP)} : M_{(SP+1)}$
$(SP) - 2 \Rightarrow SP;\ D2 \Rightarrow M_{(SP)} : M_{(SP+1)}$
$(SP) - 1 \Rightarrow SP;\ D1 \Rightarrow M_{(SP)}$
$(SP) - 1 \Rightarrow SP;\ D0 \Rightarrow M_{(SP)}$
$(SP) - 2 \Rightarrow SP;\ CCH:CCL \Rightarrow M_{(SP)} : M_{(SP+1)}$
Stop CPU clock and wait for an interrupt

Syntax Variations	Addressing Modes
`WAI`	**INH**

Description

If the CPU is in user state, WAI acts like a NOP instruction. If the CPU is in supervisor state, WAI stacks the CPU context and stops the CPU clock. Other system clocks can continue to operate so peripheral modules can continue to run. The contents of registers and the states of I/O pins remain unchanged.

Asserting $\overline{RESET}$, $\overline{XIRQ}$, or $\overline{IRQ}$ signals (if enabled) ends the standby mode. Stacking on entry to WAI allows the CPU to recover quickly when an interrupt is used.

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

Detailed Instruction Format

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	1	1B
0	0	0	0	0	1	1	0	06

`1B 06` `WAI`

ZEX — Zero-Extend (smaller CPU register to a larger CPU register) — ZEX

Syntax Variations	Addressing Modes
`ZEX` *cpureg,cpureg*	INH

Description

Zero-extend the contents of a smaller CPU register to a larger CPU register. This is an alternate mnemonic for the TFR instruction in the special case when the source register is smaller than the destination register.

If both registers are the same size, a direct transfer is performed. (see TFR instruction)

If the first register is larger than the second register, only the low portion is transferred (truncate). (see TFR instruction)

CCR Details

U	-	-	-	-	IPL	S	X	-	I	N	Z	V	C
–	–	–	–	–	–	–	–	–	–	–	–	–	–

In some cases (such as transferring D0 to CCL) the transfer instruction can cause the contents of another register to be written into the CCR so the CCR effects shown above do not apply. Unused bits in the CCR cannot be changed by any transfer instruction. The X interrupt mask can be cleared by an instruction in supervisor state but cannot be set (changed from 0 to 1) by any transfer instruction. In user state, the X and I interrupt masks cannot be changed by any transfer instruction.

Detailed Instruction Formats

INH

7	6	5	4	3	2	1	0	
1	0	0	1	1	1	1	0	9E
FIRST (SOURCE) REGISTER				SECOND (DESTINATION) REGISTER				tb

`9E tb` `ZEX` *cpureg,cpureg*

Table 6-4. Transfer Postbyte (tb) Coding Map

	source	D2	D3	D4	D5	D0	D1	D6	D7	X	Y	S	–	CCH	CCL	CCW	
destination		0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
D2	-0	–	D3 ⇒ D2	D4 ⇒ D2	D5 ⇒ D2	00:D0 ⇒ D2	00:D1 ⇒ D2	D6L ⇒ D2	D7L ⇒ D2	XL ⇒ D2	YL ⇒ D2	SL ⇒ D2		00:CCH ⇒ D2	00:CCL ⇒ D2	CCW ⇒ D2	
D3	-1	D2 ⇒ D3	–	D4 ⇒ D3	D5 ⇒ D3	00:D0 ⇒ D3	00:D1 ⇒ D3	D6L ⇒ D3	D7L ⇒ D3	XL ⇒ D3	YL ⇒ D3	SL ⇒ D3		00:CCH ⇒ D3	00:CCL ⇒ D3	CCW ⇒ D3	
D4	-2	D2 ⇒ D4	D3 ⇒ D4	–	D5 ⇒ D4	00:D0 ⇒ D4	00:D1 ⇒ D4	D6L ⇒ D4	D7L ⇒ D4	XL ⇒ D4	YL ⇒ D4	SL ⇒ D4		00:CCH ⇒ D4	00:CCL ⇒ D4	CCW ⇒ D4	
D5	-3	D2 ⇒ D5	D3 ⇒ D5	D4 ⇒ D5	–	00:D0 ⇒ D5	00:D1 ⇒ D5	D6L ⇒ D5	D7L ⇒ D5	XL ⇒ D5	YL ⇒ D5	SL ⇒ D5		00:CCH ⇒ D5	00:CCL ⇒ D5	CCW ⇒ D5	
D0	-4	D2L ⇒ D0	D3L ⇒ D0	D4L ⇒ D0	D5L ⇒ D0	–	D1 ⇒ D0	D6L ⇒ D0	D7L ⇒ D0	XL ⇒ D0	YL ⇒ D0	SL ⇒ D0		CCH ⇒ D0	CCL ⇒ D0	CCL ⇒ D0	
D1	-5	D2L ⇒ D1	D3L ⇒ D1	D4L ⇒ D1	D5L ⇒ D1	D0 ⇒ D1	–	D6L ⇒ D1	D7L ⇒ D1	XL ⇒ D1	YL ⇒ D1	SL ⇒ D1		CCH ⇒ D1	CCL ⇒ D1	CCL ⇒ D1	
D6	-6	0000:D2 ⇒ D6	0000:D3 ⇒ D6	0000:D4 ⇒ D6	0000:D5 ⇒ D6	000000:D0 ⇒ D6	000000:D1 ⇒ D6	–	D7 ⇒ D6	00:X ⇒ D6	00:Y ⇒ D6	00:S ⇒ D6		000000:CCH ⇒ D6	000000:CCL ⇒ D6	0000:CCW ⇒ D6	
D7	-7	0000:D2 ⇒ D7	0000:D3 ⇒ D7	0000:D4 ⇒ D7	0000:D5 ⇒ D7	000000:D0 ⇒ D7	000000:D1 ⇒ D7	D6 ⇒ D7	–	00:X ⇒ D7	00:Y ⇒ D7	00:S ⇒ D7		000000:CCH ⇒ D7	000000:CCL ⇒ D7	0000:CCW ⇒ D7	
X	-8	00:D2 ⇒ X	00:D3 ⇒ X	00:D4 ⇒ X	00:D5 ⇒ X	0000:D0 ⇒ X	0000:D1 ⇒ X	D6L ⇒ X	D7L ⇒ X	–	Y ⇒ X	S ⇒ X		0000:CCH ⇒ X	0000:CCL ⇒ X	00:CCW ⇒ X	
Y	-9	00:D2 ⇒ Y	00:D3 ⇒ Y	00:D4 ⇒ Y	00:D5 ⇒ Y	0000:D0 ⇒ Y	0000:D1 ⇒ Y	D6L ⇒ Y	D7L ⇒ Y	X ⇒ Y	–	S ⇒ Y		0000:CCH ⇒ Y	0000:CCL ⇒ Y	00:CCW ⇒ Y	
S	-A	00:D2 ⇒ S	00:D3 ⇒ S	00:D4 ⇒ S	00:D5 ⇒ S	0000:D0 ⇒ S	0000:D1 ⇒ S	D6L ⇒ S	D7L ⇒ S	X ⇒ S	Y ⇒ S	–		0000:CCH ⇒ S	0000:CCL ⇒ S	00:CCW ⇒ S	
reserved	-B																
CCH	-C	D2L ⇒ CCH	D3L ⇒ CCH	D4L ⇒ CCH	D5L ⇒ CCH	D0 ⇒ CCH	D1 ⇒ CCH	D6L ⇒ CCH	D7L ⇒ CCH	XL ⇒ CCH	YL ⇒ CCH	SL ⇒ CCH		–	CCL ⇒ CCH	CCL ⇒ CCH	
CCL	-D	D2L ⇒ CCL	D3L ⇒ CCL	D4L ⇒ CCL	D5L ⇒ CCL	D0 ⇒ CCL	D1 ⇒ CLL	D6L ⇒ CCL	D7L ⇒ CCL	XL ⇒ CCL	YL ⇒ CCL	SL ⇒ CCL		CCH ⇒ CCL	–	CCL ⇒ CCL	
CCW	-E	D2 ⇒ CCW	D3 ⇒ CCW	D4 ⇒ CCW	D5 ⇒ CCW	00:D0 ⇒ CCW	00:D1 ⇒ CCW	D6L ⇒ CCW	D7L ⇒ CCW	XL ⇒ CCW	YL ⇒ CCW	SL ⇒ CCW		00:CCH ⇒ CCW	00:CCL ⇒ CCW	–	
	-F																–

Chapter 7
Exceptions

7.1 Introduction

Exceptions are events that require processing outside the normal flow of instruction execution. This chapter describes exceptions and the way each is handled.

7.2 Types of Exceptions

Central Processor Unit (CPU) exceptions on the S12Z CPU include:

1. Reset
2. Software exceptions:
   - Unimplemented page 1 opcode trap (SPARE)
   - Unimplemented page 2 opcode trap (TRAP)
   - Software interrupt instruction (SWI)
   - System call interrupt instruction (SYS)
3. Machine exception
4. A non-maskable (X-bit) interrupt
5. Maskable (I-bit) interrupts

Each exception has an associated 24-bit vector, which points to the memory location where the routine that handles the exception is located. The 24-bit exception vectors are taken from a vector table. For more details about the content and the location of the exception vector table please refer to the relevant chapters in the MCU reference manual of the device, specifically the chapters describing the Reset- and Interrupt-vectors in the device top-level and the interrupt module.

The S12Z CPU can handle up to 128 exception vectors, but the number actually used varies from device to device, and some vectors are reserved for Freescale use.

Exceptions can be classified into different categories, depending on the effect of the different ways to mask interrupts.

1. Reset.
   This exception is not maskable.
2. Software exceptions.
   These include the unimplemented op-code traps, the SWI instruction and the SYS instruction. Software exceptions are not maskable.

3. Machine exception.
   A machine exception cannot be masked. Sources for a machine exception are defined in the Memory Map Control module (MMC). Please refer to the MMC chapter in the MCU Reference Manual for details.
4. X-bit interrupt.
   The interrupt service requests from the $\overline{\text{XIRQ}}$ pin is handled as a X-bit interrupt.
   This exception can be masked with the X-bit (X=1). The I-bit and the IPL-bits have no effect.
5. All remaining interrupt service requests can be masked with the I-bit (I=1) and are subject to priority filtering using the IPL-bits.

7.3 Exception Priority

A hardware priority hierarchy determines which reset or interrupt is serviced first when simultaneous requests are made. Refer to the Interrupt Module (INT) chapter in the MCU reference manual for more details concerning interrupt priority and servicing.

The priority for the different classes of exception is listed below, in descending order:

1. Reset
   This has the highest exception-processing priority.
2. Software exceptions
   This includes the SPARE and TRAP unimplemented op-codes as well as the SYS and SWI instructions.
3. Machine exception
   Machine exceptions are generated by the Memory Map Control module (MMC). Please refer to the MMC chapter in the MCU reference manual for details.
4. The X-bit interrupt
   This is used by the $\overline{\text{XIRQ}}$ pin interrupt. It is pseudo-non-maskable:
   — After reset, the X-bit in the CCR is set, which inhibits all interrupt service requests from the $\overline{\text{XIRQ}}$ pin until the X-bit is cleared.
   — The X-bit can be cleared by a program instruction, but program instructions cannot change X from 0 to 1.
   — Once the X-bit is cleared, interrupt service requests made via the $\overline{\text{XIRQ}}$ pin become non-maskable.
5. All remaining interrupts are subject to masking via the I-bit in the CCR. Relative priority between different I bit maskable interrupt requests is defined by the programmable interrupt priority level and by the position of the associated interrupt vector in the interrupt vector table. Please refer to the Interrupt Module (INT) chapter in the MCU reference manual for more details.

7.3.1 Reset

Unlike other exceptions which are normally detected and processed at instruction boundaries only, a Reset is always performed immediately. Integration module circuitry determines the type of reset that has occurred, performs basic system configuration, then passes control to the CPU. The CPU fetches the Reset

vector, jumps to the address pointed to by the vector, and begins to execute code at that address. For more information on possible causes of a reset please refer to the MCU reference manual of the device.

7.3.2 Software Exceptions

7.3.2.1 Unimplemented Op-code Traps (SPARE, TRAP)

The S12Z CPU has opcodes in only 255 of the 256 positions in the page 1 opcode map and only 162 of the 256 positions on page 2 of the opcode map are used. If the S12Z CPU attempts to execute one of the 95 unused opcodes, an unimplemented opcode trap occurs. While the unimplemented opcode on page 1 has its own separate interrupt vector, the unimplemented opcodes on page 2 share a common interrupt vector.

The S12Z CPU uses the next address after an unimplemented opcode as a return address. The stacked return address can be used to calculate the address of the unimplemented opcode for software-controlled traps.

7.3.2.2 Software Interrupt and System Call Instructions (SWI, SYS)

Execution of the SWI or SYS instruction causes an exception without a hardware interrupt service request. SWI and SYS both cannot be masked by the global mask bits in the CCR, and execution of SWI or SYS sets the I-bit. Once processing of an SWI or SYS instruction begins, I-bit maskable interrupts are inhibited until the I-bit in the CCR is cleared again. This typically occurs when an RTI instruction at the end of the service routine restores context.

7.3.3 Machine Exception

Machine exceptions are caused by the Memory Map Control module (MMC).

A Machine Exception causes the S12Z CPU to jump to the address in the Machine Exception vector as soon as the current instruction finishes execution.

When execution of a Machine Exception begins, both the X- and I-bits are set and the U-bit is cleared.

A Machine Exception is considered a severe system error, so nothing is written on the stack. The MMC module saves information about the S12Z CPU state which otherwise would be lost due to exception processing (e.g. the Program Counter register and X-, I- and U-bits from the Condition Code Register). This information can then be used to identify the source of the Machine Exception.

Please refer to the MCU reference manual for more information about possible sources of machine exceptions present on a specific MCU.

NOTE

Machine exceptions are meant to signal severe system problems. Software is expected to re-initialize the system when a machine exception occurs. Unlike interrupts or software exceptions, a machine exception causes the CPU to not perform any stack operations, so it is not possible to return to application code by simply using an RTI (or an RTS) instruction.

7.3.4 X-bit-Maskable Interrupt Request ($\overline{\text{XIRQ}}$)

The $\overline{\text{XIRQ}}$ function is disabled after system reset and upon entering the interrupt service routine for an $\overline{\text{XIRQ}}$ interrupt.

Software can clear the X-bit using an instruction such as ANDCC #$BF.

Software cannot set the X-bit from 0 to 1 once it has been cleared, and interrupt requests made via the $\overline{\text{XIRQ}}$ pin become non-maskable.

When an X-bit-maskable interrupt is recognized, both the X- and I-bits are set and the U-bit is cleared after context is saved. The X-bit is not affected by I-bit maskable interrupts. Execution of an return-from-interrupt (RTI) instruction at the end of the interrupt service routine restores the X-, I- and U-bits from the stack.

7.3.5 I-bit-Maskable Interrupt Requests

Maskable interrupt sources include on-chip peripheral systems and external interrupt service requests. Interrupts from these sources are recognized when the global interrupt mask bit (I) in the CCR is cleared. The default state of the I-bit out of reset is 1, but it can be written at any time if the CPU is not in user state.

The interrupt module manages maskable interrupt priorities. Typically, an on-chip interrupt source is subject to masking by associated bits in control registers in addition to global masking by the I-bit in the CCR. Sources generally must be enabled by writing one or more bits in associated control registers. There may be other interrupt-related control bits and flags, and there may be specific register read-write sequences associated with interrupt service. Refer to individual on-chip peripheral descriptions for details.

7.3.6 Return-from-Interrupt Instruction (RTI)

RTI is used to terminate interrupt service routines. RTI returns to the main program if no other interrupt is pending. If another interrupt is pending, RTI causes a jump to the next Interrupt service routine without returning to the main program first. In either case, RTI restores the CPU context from the stack. If no other interrupt is pending at this point, the instruction queue is refilled from the area of the return address and processing proceeds from there.

If another interrupt is pending after registers are restored, a new vector is fetched, and the stack pointer is adjusted to point at the CCR value that was just recovered (SP = SP – 29). This makes it appear that the registers have been stacked again. After the SP is adjusted, the instruction queue is refilled starting at the address the vector points to. Processing then continues with execution of the first instruction of the new interrupt service routine.

7.4 Interrupt Recognition

Once enabled, an interrupt request can be recognized at any time. When an interrupt service request is recognized, the CPU responds at the completion of the instruction being executed. Interrupt latency varies according to the number of cycles required to complete the current instruction. Instruction execution resumes when interrupt execution is complete.

When the CPU begins to service an interrupt the return address is calculated.

Then the address stored in the interrupt vector is fetched and copied to the program counter.

Next, the return address and the content of the registers are stacked as shown in Table 7-1.

In parallel to the stacking sequence new program code is fetched to start to re-fill the instruction queue.

Table 7-1. S12Z CPU Stacking Order on Entry to Interrupts

Memory Location[1]	CPU12 Registers
SP + 26	Return Address
SP + 23	Y
SP + 20	X
SP + 16	D7
SP + 12	D6
SP + 10	D5
SP + 8	D4
SP + 6	D3
SP + 4	D2
SP + 3	D1
SP + 2	D0
SP + 1	CCL
SP	CCH

[1] SP denotes the value of the stack-pointer at the end of the exception stacking sequence

After the CCR is stacked, the I-bit (and the X-bit, if an $\overline{\text{XIRQ}}$ interrupt service request caused the interrupt) is set.

The U-bit is cleared to make sure the interrupt service routine is executed in supervisor state.

Execution continues at the address pointed to by the vector for the highest-priority interrupt that was pending at the beginning of the interrupt sequence.

At the end of the interrupt service routine, an RTI instruction restores context from the stacked registers, and normal program execution resumes.

7.5 Exception Processing Flow

The first cycle in the exception processing flow for all S12Z CPU exceptions is the same, regardless of the source of the exception. Between the first and second cycles of execution, the CPU chooses one of four alternative paths. The first path is for reset, the second path is for machine exceptions, the third path is for pending hardware interrupts, and the fourth path is used for software exceptions (SWI, SYS) and trapping unimplemented opcodes (SPARE, TRAP). The last two paths are virtually identical, differing only in the details of calculating the return address. Refer to Figure 7-2 for the following description of events.

7.5.1 Vector Fetch

The first cycle of all exception processing, regardless of the cause, is a vector fetch. The vector points to the address where exception processing will continue. Exception vectors are stored in a table located at the top of the memory map ($FFxxxx) if not placed elsewhere using the Interrupt Vector Base Register (please

refer to the s12z_int module chapter in the device reference manual for more information on the Interrupt Vector Base Register).

Supervisor state is forced regardless of the current state of the U-bit. This ensures the vector fetch cycle and the entire exception stacking sequence taking place in supervisor state. This is independent from the actual clearing of the U-bit which during an interrupt sequence does not happen until the CCH register was stacked. Please refer to Figure 7-1 for details.

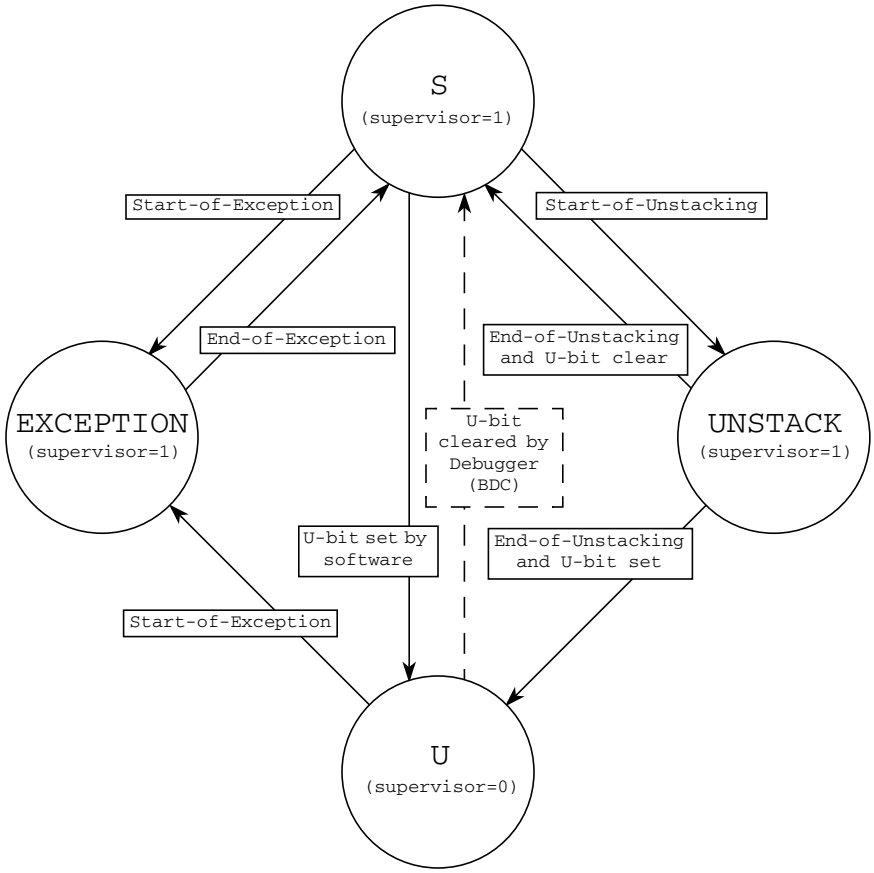


Figure 7-1. S12Z CPU Supervisor-State/User-State Transition Diagram

Right before the vector fetch cycle, the S12Z CPU issues a signal to ask the interrupt module for the vector address of the highest priority, pending exception. This address is then used to fetch the address of the interrupt service routine (ISR).

After the vector fetch, the CPU selects one of the four alternate execution paths, depending upon the cause of the exception (please refer to Figure 7-2 for details).

7.5.2 Reset Exception Processing

A system reset sets the S-, X-, and I-bits and clears the U- and IPL[2:0]-bits in the CCL and CCH registers. Opcode fetches start at the address pointed to by the reset vector. When the instruction queue contains enough program data, the CPU starts executing the instruction at the head of the instruction queue.

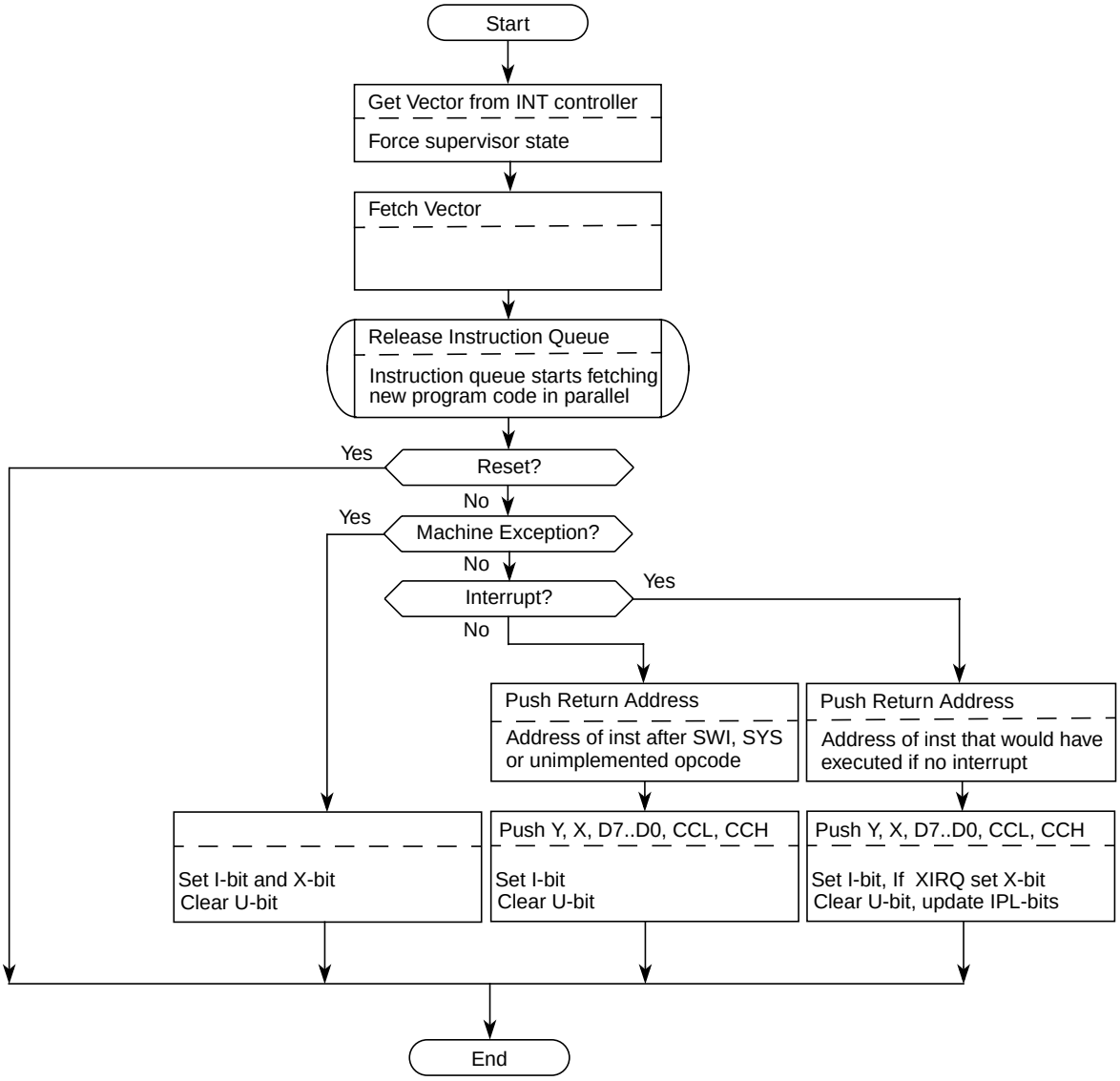


Figure 7-2. Exception Processing Flow Diagram

7.5.3 Interrupt and Unimplemented Opcode Trap Exception Processing

If an exception was not caused by a reset or a machine exception, a return address is calculated.

- The CPU performs different return address calculations for each type of exception.
  - When an X-bit or I-bit maskable interrupt causes the exception, the return address points to the next instruction that would have been executed had processing not been interrupted.

  — When an exception is caused by an SWI opcode, a SYS opcode or by an unimplemented opcode (see Section 7.3.2, “Software Exceptions), the return address points to the next address after the opcode.

- Then the return address and the CPU registers Y, X, D7..D0, CCL and CCH are pushed onto the stack. The entire stacking sequence takes eight bus-cycles, independent of stack-alignment.
- At the end of the stacking sequence, the I-bit is set and the U-bit is cleared. If the exception is caused by an interrupt, the IPL-bits are updated and if the interrupt is caused by an XIRQ the X-bit is set as well.

Chapter 8
Instruction Execution Timing

8.1 Introduction

This section contains listings of the S12Z CPU instruction execution times in terms of bus-clock cycles. In this data, it is assumed that data-aligned memory read cycles consist of one clock period while data-aligned memory write cycles consist of one half clock period.

Misaligned data or a longer memory cycle can cause the generation of wait states that must be added to the total instruction times.

The number of bus read and write cycles for each instruction is also included with the timing data. This data is shown as:

Table 8-1. Instruction Cycle Timing Format

n(r/w)

n –	This is the total number of required bus-clock cycles to execute the instruction. Internal CPU cycles are included as well as cycles required for operand fetches, if applicable. This number represents the minimum number of required clock-cycles (best case) to execute an instruction; any (optional) instruction queue fetches and additional wait-cycles for memory accesses are not included.
r/w –	This represents the number of operand reads (r) and operand writes (w). For example: an instruction which does a read-modify-write from/to memory shows (1/1) here.

8.2 Instruction Execution Timing

8.2.1 No Operation Instruction Execution Times (NOP)

Table 8-2 shows the number of clock cycles required for execution of the No-Operation instruction (NOP).

Table 8-2. No-Operation Execution Timing

Operation	Cycles
NOP	1(0/0)

8.2.2 Move Instruction Execution Times (MOV)

Table 8-3 shows the number of clock cycles required for execution of the Move instruction (MOV).

Table 8-3. Move Data Execution Timing

	Destination				
Source	REG	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
IMM1	1(0/0)	2.5(0/1)	3(0/1)	4(1/1)	4.5(1/1)
IMM2	1.5(0/0)	2.5(0/1)	3(0/1)	4(1/1)	4.5(1/1)
IMM3	2(0/0)	3(0/1)	3(0/1)	4.5(1/1)	4.5(1/1)
IMM4	2(0/0)	3(0/1)	3.5(0/1)	4.5(1/1)	5(1/1)
REG IMMe4	2(0/0)	3(0/1)	3(0/1)	4.5(1/1)	4.5(1/1)
(IDX) (++IDX) (REG,IDX)	3.5(1/0)	4.5(1/1)	4.5(1/1)	6(2/1)	6(2/1)
(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	4(1/0)	5(1/1)	5(1/1)	6.5(2/1)	6.5(2/1)
[REG,IDX]	5(2/0)	6(2/1)	6(2/1)	7.5(3/1)	7.5(3/1)
[IDX1] [IDX3] [EXT3]	5.5(2/0)	6.5(2/1)	6.5(2/1)	8(3/1)	8(3/1)

8.2.3 Load Instruction Execution Times (LD)

Table 8-4 shows the number of clock cycles required for execution of the Load instruction (LD).

Table 8-4. Load Register Execution Timing

Operation	Cycles	Operation	Cycles
LD Dn,#IMM1 LD Dn,#IMM2	1(0/0)	LD XY,#IMMu18 LD XY,#IMM3	1(0/0)
LD Dn,#IMM4	1.5(0/0)	LD S,#IMM3	1.5(0/0)
LD Dn,EXT24	2.5(1/0)	LD XY,EXT24	2.5(1/0)
LD Dn,REG LD Dn,#IMMe4	1(0/0)	LD XYS,REG LD XYS,#IMMe4	1(0/0)

LD Dn,(IDX)
LD Dn,(++IDX)
LD Dn,(REG,IDX)
	2.5(1/0)	LD XYS,(IDX)
		LD XYS,(++IDX)
		LD XYS,(REG,IDX)
			2.5(1/0)

LD Dn,(IDX1)
LD Dn,(IDX3)
LD Dn,(IDX2,REG)
LD Dn,(IDX3,REG)
LD Dn,EXT1
LD Dn,EXT2
LD Dn,EXT3
	3(1/0)	LD XYS,(IDX1)
		LD XYS,(IDX3)
		LD XYS,(IDX2,REG)
		LD XYS,(IDX3,REG)
		LD XYS,EXT1
		LD XYS,EXT2
		LD XYS,EXT3
			3(1/0)

Table 8-4. Load Register Execution Timing

Operation	Cycles	Operation	Cycles
LD Dn,[REG,IDX]	4(2/0)	LD XYS,[REG,IDX]	4(2/0)

LD Dn,[IDX1]
LD Dn,[IDX3]
LD Dn,[EXT3]
	4.5(2/0)	LD XYS,[IDX1]
		LD XYS,[IDX3]
		LD XYS,[EXT3]
			4.5(2/0)

8.2.4 Store Instruction Execution Times (ST)

Table 8-5 shows the number of clock cycles required for execution of the Store instruction (ST).

Table 8-5. Store Register Execution Timing

Operation	Cycles	Operation	Cycles
ST Dn,EXT24	2(0/1)	ST XY,EXT24	2(0/1)
ST Dn,REG	1(0/0)	ST XYS,REG	1(0/0)
ST Dn,(IDX) ST Dn,(++IDX) ST Dn,(REG,IDX)	2(0/1)	ST XYS,(IDX) ST XYS,(++IDX) ST XYS,(REG,IDX)	2(0/1)
ST Dn,(IDX1) ST Dn,(IDX3) ST Dn,(IDX2,REG) ST Dn,(IDX3,REG) ST Dn,EXT1 ST Dn,EXT2 ST Dn,EXT3	2.5(0/1)	ST XYS,(IDX1) ST XYS,(IDX3) ST XYS,(IDX2,REG) ST XYS,(IDX3,REG) ST XYS,EXT1 ST XYS,EXT2 ST XYS,EXT3	2.5(0/1)
ST Dn,[REG,IDX]	3.5(1/1)	ST XYS,[REG,IDX]	3.5(1/1)
ST Dn,[IDX1] ST Dn,[IDX3] ST Dn,[EXT3]	4(1/1)	ST XYS,[IDX1] ST XYS,[IDX3] ST XYS,[EXT3]	4(1/1)

8.2.5 Push Register(s) onto Stack Instruction Execution Times (PSH)

Table 8-6 shows the number of clock cycles required for execution of the Push Register(s) onto Stack instruction (PSH).

Table 8-6. Push Register(s) onto Stack Execution Timing

Operation	Cycles
PSH oprregs	1.5 + 0.5*n

8.2.6 Pull Register(s) from Stack Instruction Execution Times (PUL)

Table 8-7 shows the number of clock cycles required for execution of the Pull Register(s) from Stack instruction (PUL).

Table 8-7. Pull Register(s) from Stack Execution Timing

Operation	Cycles
PUL oprregs	2.5 + 0.5*n

8.2.7 Load Effective Address Instruction Execution Times (LEA)

Table 8-8 shows the number of clock cycles required for execution of the Load Effective Address instruction (LEA).

Table 8-8. Load Effective Address Execution Timing

Operation	Cycles
LEA XYS,(IMMs8,XYS)	1(0/0)
LEA D67XYS,(IDX) LEA D67XYS,(++IDX) LEA D67XYS,(REG,IDX)	1(0/0)
LEA D67XYS,(IDX1) LEA D67XYS,(IDX3) LEA D67XYS,(IDX2,REG) LEA D67XYS,(IDX3,REG) LEA D67XYS,EXT1 LEA D67XYS,EXT2 LEA D67XYS,EXT3	1.5(0/0)
LEA D67XYS,[REG,IDX]	2.5(1/0)
LEA D67XYS,[IDX1] LEA D67XYS,[IDX3] LEA D67XYS,[EXT3]	3(1/0)

8.2.8 Clear Instruction Execution Times (CLR)

Table 8-9 shows the number of clock cycles required for execution of the Clear instruction (CLR).

Table 8-9. Clear Execution Timing

Operation	Cycles
CLR Dn CLR XY	1(0/0)
CLR REG	1(0/0)
CLR.bwpl (IDX) CLR.bwpl (++IDX) CLR.bwpl (REG,IDX)	2(0/1)
CLR.bwpl (IDX1) CLR.bwpl (IDX3) CLR.bwpl (IDX2,REG) CLR.bwpl (IDX3,REG) CLR.bwpl EXT1 CLR.bwpl EXT2 CLR.bwpl EXT3	2.5(0/1)
CLR.bwpl [REG,IDX]	3.5(1/1)
CLR.bwpl [IDX1] CLR.bwpl [IDX3] CLR.bwpl [EXT3]	4(1/1)

8.2.9 Register-To-Register Transfer and Exchange Execution Times (TFR, EXG, SEX, ZEX)

Table 8-10 and Table 8-11 show the number of clock cycles required for execution of Register-To-Register Transfer and Exchange instructions (TFR, EXG, SEX, ZEX).

Table 8-10. Register-To-Register Transfer (TFR, SEX, ZEX) Execution Timing

	Destination				
Source	Dn	XYS	CCL	CCH	CCW
Dn XYS	1(0/0)	1(0/0)	1(0/0)	1.5(0/0)	1.5(0/0)
CCL	1(0/0)	1(0/0)	–	1.5(0/0)	–
CCH	1(0/0)	1(0/0)	1(0/0)	–	–
CCW	1(0/0)	1(0/0)	–	–	–

Table 8-11. Register-To-Register Exchange (EXG) Execution Timing

	Destination				
Source	Dn	XYS	CCL	CCH	CCW
Dn XYS	1(0/0)	1(0/0)	1(0/0)	1.5(0/0)	1.5(0/0)
CCL	1(0/0)	1(0/0)	–	1.5(0/0)	–
CCH	1.5(0/0)	1.5(0/0)	1.5(0/0)	–	–
CCW	1.5(0/0)	1.5(0/0)	–	–	–

8.2.10 Logical AND/OR Instruction Execution Times (AND, OR, BIT, EOR)

Table 8-12 shows the number of clock cycles required for execution of a logical AND/OR instruction (AND, OR, BIT, EOR).

Table 8-12. Logical Operation Execution Timing

Operation	Cycles
<OP> Dn,#IMM1 <OP> Dn,#IMM2	1(0/0)
<OP> Dn,#IMM4	1.5(0/0)
<OP> Dn,EXT24	2.5(1/0)
<OP> Dn,REG <OP> Dn,#IMMe4	1(0/0)
<OP> Dn,(IDX) <OP> Dn,(++IDX) <OP> Dn,(REG,IDX)	2.5(1/0)
<OP> Dn,(IDX1) <OP> Dn,(IDX3) <OP> Dn,(IDX2,REG) <OP> Dn,(IDX3,REG) <OP> Dn,EXT1 <OP> Dn,EXT2 <OP> Dn,EXT3	3(1/0)
<OP> Dn,[REG,IDX]	4(2/0)
<OP> Dn,[IDX1] <OP> Dn,[IDX3] <OP> Dn,[EXT3]	4.5(2/0)

8.2.11 One's Complement (Invert) Instruction Execution Times (COM)

Table 8-13 shows the number of clock cycles required for execution of a One's Complement (logical invert) instruction (COM).

Table 8-13. One's Complement (Invert) Execution Timing

Operation	Cycles
COM REG	1(0/0)
COM.bwl (IDX) COM.bwl (++IDX) COM.bwl (REG,IDX)	3.5(1/1)
COM.bwl (IDX1) COM.bwl (IDX3) COM.bwl (IDX2,REG) COM.bwl (IDX3,REG) COM.bwl EXT1 COM.bwl EXT2 COM.bwl EXT3	4(1/1)
COM.bwl [REG,IDX]	5(2/1)
COM.bwl [IDX1] COM.bwl [IDX3] COM.bwl [EXT3]	5.5(2/1)

8.2.12 Increment and Decrement Instruction Execution Times (INC, DEC)

Table 8-14 shows the number of clock cycles required for execution of an Increment or Decrement instruction (INC, DEC).

Table 8-14. Increment or Decrement Execution Timing

Operation	Cycles
<OP> Dn	1(0/0)
<OP> REG	1(0/0)
<OP>.bwl (IDX) <OP>.bwl (++IDX) <OP>.bwl (REG,IDX)	3.5(1/1)
<OP>.bwl (IDX1) <OP>.bwl (IDX3) <OP>.bwl (IDX2,REG) <OP>.bwl (IDX3,REG) <OP>.bwl EXT1 <OP>.bwl EXT2 <OP>.bwl EXT3	4(1/1)
<OP>.bwl [REG,IDX]	5(2/1)
<OP>.bwl [IDX1] <OP>.bwl [IDX3] <OP>.bwl [EXT3]	5.5(2/1)

8.2.13 Add and Subtract Instruction Execution Times (ADD, ADC, SUB, SBC, CMP)

Table 8-15 and Table 8-16 show the number of clock cycles required for execution of an Add, Subtract or Compare instruction (ADD, ADC, SUB, SBC, CMP).

Table 8-15. Arithmetic Operation Execution Timing

Operation	Cycles
<OP> Dn,#IMM1 <OP> Dn,#IMM2	1(0/0)
<OP> Dn,#IMM4	1.5(0/0)
<OP> Dn,EXT24	2.5(1/0)
<OP> Dn,REG <OP> Dn,#IMMe4	1(0/0)
<OP> Dn,(IDX) <OP> Dn,(++IDX) <OP> Dn,(REG,IDX)	2.5(1/0)
<OP> Dn,(IDX1) <OP> Dn,(IDX3) <OP> Dn,(IDX2,REG) <OP> Dn,(IDX3,REG) <OP> Dn,EXT1 <OP> Dn,EXT2 <OP> Dn,EXT3	3(1/0)
<OP> Dn,[REG,IDX]	4(2/0)
<OP> Dn,[IDX1] <OP> Dn,[IDX3] <OP> Dn,[EXT3]	4.5(2/0)

Table 8-16. Pointer Arithmetic Operation Execution Timing

Operation	Cycles

SUB D6,X,Y
CMP X,Y
CMP Y,X
	1(0/0)

8.2.14 Two's Complement (Negate) Instruction Execution Times (NEG)

Table 8-17 shows the number of clock cycles required for execution of a Two's Complement (negate) instruction (NEG).

Table 8-17. Two's Complement (Negate) Execution Timing

Operation	Cycles
NEG REG	1(0/0)
NEG.bwl (IDX) NEG.bwl (++IDX) NEG.bwl (REG,IDX)	3.5(1/1)

Table 8-17. Two's Complement (Negate) Execution Timing

Operation	Cycles
NEG.bwl (IDX1) NEG.bwl (IDX3) NEG.bwl (IDX2,REG) NEG.bwl (IDX3,REG) NEG.bwl EXT1 NEG.bwl EXT2 NEG.bwl EXT3	4(1/1)
NEG.bwl [REG,IDX]	5(2/1)
NEG.bwl [IDX1] NEG.bwl [IDX3] NEG.bwl [EXT3]	5.5(2/1)

8.2.15 Absolute Value Instruction Execution Time (ABS)

Table 8-18 shows the number of clock cycles required for execution of the Absolute Value instruction (ABS).

Table 8-18. Absolute Value Execution Timing

Operation	Cycles
ABS Dn	1(0/0)

8.2.16 Saturate Instruction Execution Time (SAT)

Table 8-19 shows the number of clock cycles required for execution of the Saturate instruction (SAT).

Table 8-19. Saturate Execution Timing

Operation	Cycles
SAT Dn	1(0/0)

8.2.17 Count Leading Sign-Bits Execution Time (CLB)

Table 8-20 shows the number of clock cycles required for execution of the Count Leading Sign-Bits instruction (CLB).

Table 8-20. Count Leading Sign-Bits Execution Timing

Operation	Cycles
CLB Ds,Dd	1(0/0)

8.2.18 Multiply Instruction Execution Times (MULS, MULU)

Table 8-21 and Table 8-22 show the number of clock cycles required for execution of Signed Multiply (MULS) and Unsigned Multiply (MULU) operations.

Table 8-21. Signed Multiply (MULS) Execution Timing[1]

	Source1					
Source2	Dn	REG IMMe4	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
Dn	2(0/0)	–	–	–	–	–
	3.5(0/0)	–	–	–	–	–
IMM1 IMM2	2(0/0)	–	–	–	–	–
	3.5(0/0)	–	–	–	–	–
IMM4	4(0/0)	–	–	–	–	–
REG IMMe4	2(0/0)	3(0/0)	3.5(1/0)	5(1/0)	6(2/0)	6.5(2/0)
	3.5(0/0)	4.5(0/0)	6(1/0)	6.5(1/0)	7.5(2/0)	8(2/0)
(IDX) (++IDX) (REG,IDX)	3.5(1/0)	4.5(1/0)	6(2/0)	6.5(2/0)	7.5(3/0)	8(3/0)
	5(1/0)	6(1/0)	7.5(2/0)	8(2/0)	9(3/0)	9.5(3/0)
(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	4(1/0)	5(1/0)	6(2/0)	6.5(2/0)	7.5(3/0)	8(3/0)
	5.5(1/0)	6.5(1/0)	7.5(2/0)	8(2/0)	9(3/0)	9.5(3/0)
[REG,IDX]	5(2/0)	6(2/0)	7.5(3/0)	8(3/0)	9(4/0)	9.5(4/0)
	6.5(2/0)	7.5(2/0)	9(3/0)	9.5(3/0)	10.5(4/0)	11(4/0)
[IDX1] [IDX2] [EXT3]	5.5(2/0)	6(2/0)	7.5(3/0)	8(3/0)	9(4/0)	9.5(4/0)
	7(2/0)	7.5(2/0)	9(3/0)	9.5(3/0)	10.5(4/0)	11(4/0)

[1] The rows with shaded background describe the instruction execution timing if at least one of the source operands is bigger than 16 bits. Otherwise the instruction timing shown in the rows with white background is valid.

Table 8-22. Unsigned Multiply (MULU) Execution Timing[1]

	Source1					
Source2	Dn	REG IMMe4	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
Dn	1(0/0)	–	–	–	–	–
	2.5(0/0)	–	–	–	–	–
IMM1 IMM2	1(0/0)	–	–	–	–	–
	2.5(0/0)	–	–	–	–	–

Table 8-22. Unsigned Multiply (MULU) Execution Timing[1]

	Source1					
Source2	Dn	REG IMMe4	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
IMM4	3(0/0)	–	–	–	–	–
REG IMMe4	1(0/0)	2(0/0)	3.5(1/0)	4(1/0)	5(2/0)	5.5(2/0)
	2.5(0/0)	3.5(0/0)	5(1/0)	5.5(1/0)	6.5(2/0)	7(2/0)
(IDX) (++IDX) (REG,IDX)	2.5(1/0)	3.5(1/0)	5(2/0)	5.5(2/0)	6.5(3/0)	7(3/0)
	4(1/0)	5(1/0)	6.5(2/0)	7(2/0)	8(3/0)	8.5(3/0)
(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	3(1/0)	3.5(1/0)	5(2/0)	5.5(2/0)	6.5(3/0)	7(3/0)
	4.5(1/0)	5(1/0)	6.5(2/0)	7(2/0)	8(3/0)	8.5(3/0)
[REG,IDX]	4(2/0)	5(2/0)	6.5(3/0)	7(3/0)	8(4/0)	8.5(4/0)
	5.5(2/0)	6.5(2/0)	8(3/0)	8.5(3/0)	9.5(4/0)	10(4/0)
[IDX1] [IDX2] [EXT3]	4.5(2/0)	5(2/0)	6.5(3/0)	7(3/0)	8(4/0)	8.5(4/0)
	6(2/0)	6.5(2/0)	8(3/0)	8.5(3/0)	9.5(4/0)	10(4/0)

1 The rows with shaded background describe the instruction execution timing if at least one of the source operands is bigger than 16 bits. Otherwise the instruction timing shown in the rows with white background is valid.

8.2.19 Fractional Multiply Instruction Execution Times (QMULS, QMULU)

Table 8-23 and Table 8-24 show the number of clock cycles required for execution of Signed Fractional Multiply (QMULS) and Unsigned Fractional Multiply (QMULU) operations.

Table 8-23. Signed Fractional Multiply (QMULS) Execution Timing[1]

	Source1					
Source2	Dn	REG IMMe4	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
Dn	3.5(0/0)	–	–	–	–	–
	6.5(0/0)	–	–	–	–	–
IMM1	3.5(0/0)	–	–	–	–	–
	6.5(0/0)	–	–	–	–	–

Table 8-23. Signed Fractional Multiply (QMULS) Execution Timing[1]

	Source1					
Source2	Dn	REG IMMe4	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
IMM2	4(0/0)	–	–	–	–	–
	7(0/0)	–	–	–	–	–
IMM4	7(0/0)	–	–	–	–	–
REG IMMe4	3.5(0/0)	4.5(0/0)	6(1/0)	6.5(1/0)	7.5(2/0)	8(2/0)
	6.5(0/0)	7.5(0/0)	9(1/0)	9.5(1/0)	10.5(2/0)	11(2/0)
(IDX) (++IDX) (REG,IDX)	5(1/0)	6(1/0)	7.5(2/0)	8(2/0)	9(3/0)	9.5(3/0)
	8(1/0)	9(1/0)	10.5(2/0)	11(2/0)	12(3/0)	12.5(3/0)
(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	5.5(1/0)	6(1/0)	7.5(2/0)	8(2/0)	9(3/0)	9.5(3/0)
	8.5(1/0)	9(1/0)	10.5(2/0)	11(2/0)	12(3/0)	12.5(3/0)
[REG,IDX]	6.5(2/0)	7.5(2/0)	9(3/0)	9.5(3/0)	10.5(4/0)	11(4/0)
	9.5(2/0)	10.5(2/0)	12(3/0)	12.5(3/0)	13.5(4/0)	14(4/0)
[IDX1] [IDX2] [EXT3]	7(2/0)	7.5(2/0)	9(3/0)	9.5(3/0)	10.5(4/0)	11(4/0)
	10(2/0)	10.5(2/0)	12(3/0)	12.5(3/0)	13.5(4/0)	14(4/0)

[1] The rows with shaded background describe the instruction execution timing if at least one of the source operands for the implied multiply operation is bigger than 16 bits. Otherwise the instruction timing shown in the rows with white background is valid.

Table 8-24. Unsigned Fractional Multiply (QMULU) Execution Timing[1]

	Source1					
Source2	Dn	REG IMMe4	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
Dn	2.5(0/0)	–	–	–	–	–
	5.5(0/0)	–	–	–	–	–
IMM1	3(0/0)	–	–	–	–	–
	6(0/0)	–	–	–	–	–
IMM2	3.5(0/0)	–	–	–	–	–
	6.5(0/0)	–	–	–	–	–

Table 8-24. Unsigned Fractional Multiply (QMULU) Execution Timing[1]

	Source1					
Source2	Dn	REG IMMe4	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
IMM4	6.5(0/0)	–	–	–	–	–
REG IMMe4	3(0/0)	4(0/0)	5.5(1/0)	6(1/0)	7(2/0)	7.5(2/0)
	6(0/0)	7(0/0)	8.5(1/0)	9(1/0)	10(2/0)	10.5(2/0)
(IDX) (++IDX) (REG,IDX)	4.5(1/0)	5.5(1/0)	7(2/0)	7.5(2/0)	8.5(3/0)	9(3/0)
	7.5(1/0)	8.5(1/0)	10(2/0)	10.5(2/0)	11.5(3/0)	12(3/0)
(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	5(1/0)	5.5(1/0)	7(2/0)	7.5(2/0)	8.5(3/0)	9(3/0)
	8(1/0)	8.5(1/0)	10(2/0)	10.5(2/0)	11.5(3/0)	12(3/0)
[REG,IDX]	6(2/0)	7(2/0)	8.5(3/0)	9(3/0)	10(4/0)	10.5(4/0)
	9(2/0)	10(2/0)	11.5(3/0)	12(3/0)	13(4/0)	13.5(4/0)
[IDX1] [IDX2] [EXT3]	6.5(2/0)	8.5(2/0)	8.5(3/0)	9(3/0)	10(4/0)	10.5(4/0)
	9.5(2/0)	10(2/0)	11.5(3/0)	12(3/0)	13(4/0)	13.5(4/0)

1 The rows with shaded background describe the instruction execution timing if at least one of the source operands for the implied multiply operation is bigger than 16 bits. Otherwise the instruction timing shown in the rows with white background is valid.

8.2.20 Multiply and Accumulate Instruction Execution Times (MACS, MACU)

Table 8-25 and Table 8-26 show the number of clock cycles required for execution of Signed Multiply-and-Accumulate (MACS) and Unsigned Multiply-and-Accumulate (MACU) operations.

Table 8-25. Signed Multiply-and-Accumulate (MACS) Execution Timing[1]

	Source1					
Source2	Dn	REG IMMe4	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
Dn	2.5(0/0)	–	–	–	–	–
	4(0/0)	–	–	–	–	–
IMM1	2.5(0/0)	–	–	–	–	–
	4(0/0)	–	–	–	–	–

Table 8-25. Signed Multiply-and-Accumulate (MACS) Execution Timing[1]

	Source1					
Source2	Dn	REG IMMe4	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
IMM2	3(0/0)	–	–	–	–	–
	4.5(0/0)	–	–	–	–	–
IMM4	4.5(0/0)	–	–	–	–	–
REG IMMe4	2.5(0/0)	3.5(0/0)	5(1/0)	5.5(1/0)	6.5(2/0)	7(2/0)
	4(0/0)	5(0/0)	6.5(1/0)	7(1/0)	8(2/0)	8.5(2/0)
(IDX) (++IDX) (REG,IDX)	4(1/0)	5(1/0)	6.5(2/0)	7(2/0)	8(3/0)	8.5(3/0)
	5.5(1/0)	6.5(1/0)	8(2/0)	8.5(2/0)	9.5(3/0)	10(3/0)
(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	4.5(1/0)	5.5(1/0)	6.5(2/0)	7(2/0)	8(3/0)	8.5(3/0)
	6(1/0)	7(1/0)	8(2/0)	8.5(2/0)	9.5(3/0)	10(3/0)
[REG,IDX]	5.5(2/0)	6.5(2/0)	8(3/0)	8.5(3/0)	9.5(4/0)	10(4/0)
	7(2/0)	8(2/0)	9.5(3/0)	10(3/0)	11(4/0)	11.5(4/0)
[IDX1] [IDX2] [EXT3]	6(2/0)	7(2/0)	8(3/0)	8.5(3/0)	9.5(4/0)	10(4/0)
	7.5(2/0)	8.5(2/0)	9.5(3/0)	10(3/0)	11(4/0)	11.5(4/0)

[1] The rows with shaded background describe the instruction execution timing if at least one of the source operands for the implied multiply operation is bigger than 16 bits. Otherwise the instruction timing shown in the rows with white background is valid.

Table 8-26. Unsigned Multiply-and-Accumulate (MACU) Execution Timing[1]

	Source1					
Source2	Dn	REG IMMe4	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
Dn	1.5(0/0)	–	–	–	–	–
	3(0/0)	–	–	–	–	–
IMM1	1.5(0/0)	–	–	–	–	–
	3(0/0)	–	–	–	–	–
IMM2	2(0/0)	–	–	–	–	–
	3.5(0/0)	–	–	–	–	–

Table 8-26. Unsigned Multiply-and-Accumulate (MACU) Execution Timing[1]

	Source1					
Source2	Dn	REG IMMe4	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
IMM4	3.5(0/0)	–	–	–	–	–
REG IMMe4	1.5(0/0)	2.5(0/0)	4(1/0)	4.5(1/0)	5.5(2/0)	6(2/0)
	3(0/0)	4(0/0)	5.5(1/0)	6(1/0)	7(2/0)	7.5(2/0)
(IDX) (++IDX) (REG,IDX)	3(1/0)	4(1/0)	5.5(2/0)	6(2/0)	7(3/0)	7.5(3/0)
	4.5(1/0)	5.5(1/0)	7(2/0)	7.5(2/0)	8.5(3/0)	9(3/0)
(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	3.5(1/0)	4(1/0)	5.5(2/0)	6(2/0)	7(3/0)	7.5(3/0)
	5(1/0)	5.5(1/0)	7(2/0)	7.5(2/0)	8.5(3/0)	9(3/0)
[REG,IDX]	4.5(2/0)	5.5(2/0)	7(3/0)	7.5(3/0)	8.5(4/0)	9(4/0)
	6(2/0)	7(2/0)	8.5(3/0)	9(3/0)	10(4/0)	10.5(4/0)
[IDX1] [IDX2] [EXT3]	5(2/0)	5.5(2/0)	7(3/0)	7.5(3/0)	8.5(4/0)	9(4/0)
	6.5(2/0)	7(2/0)	8.5(3/0)	9(3/0)	10(4/0)	10.5(4/0)

1 The rows with shaded background describe the instruction execution timing if at least one of the source operands for the implied multiply operation is bigger than 16 bits. Otherwise the instruction timing shown in the rows with white background is valid.

8.2.21 Divide and Modulo Instruction Execution Times (DIVS, DIVU, MODS, MODU)

Table 8-27 and Table 8-28 show the number of clock cycles required for execution of Signed Divide or Modulo (DIVS, MODS) and Unsigned Divide or Modulo (DIVU, MODU) operations.

Table 8-27. Signed Divide/Modulo (DIVS/MODS) Execution Timing[1]

	Source1					
Source2	Dn	REG IMMe4	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
Dn	3+n(0/0)	–	–	–	–	–
IMM1	3+n(0/0)	–	–	–	–	–
IMM2 IMM4	3.5+n(0/0)	–	–	–	–	–

Table 8-27. Signed Divide/Modulo (DIVS/MODS) Execution Timing[1]

	Source1					
Source2	Dn	REG IMMe4	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
REG IMMe4	3+n(0/0)	4+n(0/0)	5.5+n(1/0)	6+n(1/0)	7+n(2/0)	7.5+n(2/0)
(IDX) (++IDX) (REG,IDX)	4.5+n(1/0)	5.5+n(1/0)	7+n(2/0)	7.5+n(2/0)	8.5+n(3/0)	9+n(3/0)
(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	5+n(1/0)	5.5+n(1/0)	7+n(2/0)	7.5+n(2/0)	8.5+n(3/0)	9+n(3/0)
[REG,IDX]	6+n(2/0)	7+n(2/0)	8.5+n(3/0)	9+n(3/0)	10+n(4/0)	10.5+n(4/0)
[IDX1] [IDX2] [EXT3]	6.5+n(2/0)	7+n(2/0)	8.5+n(3/0)	9+n(3/0)	10+n(4/0)	10.5+n(4/0)

1 The letter ‘n’ denotes the number of cycles to be added depending on the size (number of bits divided by 2) of the dividend (or nominator) operand; ‘n’ is either 4, 8, 12 or 16.

Table 8-28. Unsigned Divide/Modulo (DIVU/MODU) Execution Timing[1]

	Source1					
Source2	Dn	REG IMMe4	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
Dn	2.5+n(0/0)	–	–	–	–	–
IMM1	2.5+n(0/0)	–	–	–	–	–
IMM2 IMM4	3+n(0/0)	–	–	–	–	–
REG IMMe4	2.5+n(0/0)	3.5+n(0/0)	5+n(1/0)	5.5+n(1/0)	6.5+n(2/0)	7+n(2/0)
(IDX) (++IDX) (REG,IDX)	4+n(1/0)	5+n(1/0)	6.5+n(2/0)	7+n(2/0)	8+n(3/0)	8.5+n(3/0)
(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	4.5+n(1/0)	5+n(1/0)	6.5+n(2/0)	7+n(2/0)	8+n(3/0)	8.5+n(3/0)

Table 8-28. Unsigned Divide/Modulo (DIVU/MODU) Execution Timing[1]

	Source1					
Source2	Dn	REG IMMe4	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
[REG,IDX]	5.5+n(2/0)	6.5+n(2/0)	8+n(3/0)	8.5+n(3/0)	9.5+n(4/0)	10+n(4/0)
[IDX1] [IDX2] [EXT3]	6+n(2/0)	6.5+n(2/0)	8+n(3/0)	8.5+n(3/0)	9.5+n(4/0)	10+n(4/0)

1 The letter ‘n’ denotes the number of cycles to be added depending on the size (in number of bits divided by 2) of the dividend (or nominator) operand; ‘n’ is either 4, 8, 12 or 16.

8.2.22 Maximum and Minimum Instruction Execution Times (MAXS, MAXU, MINS, MINU)

Table 8-29 shows the number of clock cycles required for execution of the Minimum and Maximum operations (MAXS, MAXU, MINS, MINU).

Table 8-29. Minimum and Maximum Execution Timing

Operation	Cycles
<OP> Dn,REG <OP> Dn,#IMMe4	2(0/0)
<OP> Dn,(IDX) <OP> Dn,(++IDX) <OP> Dn,(REG,IDX)	3.5(1/0)
<OP> Dn,(IDX1) <OP> Dn,(IDX3) <OP> Dn,(IDX2,REG) <OP> Dn,(IDX3,REG) <OP> Dn,EXT1 <OP> Dn,EXT2 <OP> Dn,EXT3	4(1/0)
<OP> Dn,[REG,IDX]	5(2/0)
<OP> Dn,[IDX1] <OP> Dn,[IDX3] <OP> Dn,[EXT3]	5.5(2/0)

8.2.23 Shift Instruction Execution Times (ASL, ASR, LSL, LSR)

Table 8-30 shows the number of clock cycles required for execution of Shift operations (ASL, ASR, LSL, LSR) with a data-register as destination. Likewise Table 8-31 shows the number of clock cycles required for shifting a memory operand by 1 or 2.

Table 8-30. Shift (ASL, ASR, LSL, LSR) to Register Execution Timing

	Source1 (shift operand)					
Source2 (shift width)	Ds	REG IMMe4	(IDX) (++IDX) (REG,IDX)	(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	[REG,IDX]	[IDX1] [IDX3] [EXT3]
IMM (1 or 2)	1(0/0)	1.5(0/0)	2.5(1/0)	3(1/0)	4(2/0)	4.5(2/0)
IMM (3..31) REG	1(0/0)	2(0/0)	3.5(1/0)	4(1/0)	5(2/0)	5.5(2/0)
(IDX) (++IDX) (REG,IDX)	2.5(1/0)	3.5(1/0)	5(2/0)	5.5(2/0)	6.5(3/0)	7(3/0)
(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3	3(1/0)	3.5(1/0)	5(2/0)	5.5(2/0)	6.5(3/0)	7(3/0)
[REG,IDX]	4(2/0)	5(2/0)	6.5(3/0)	7(3/0)	8.5(4/0)	9(4/0)
[IDX1] [IDX2] [EXT3]	4.5(2/0)	5(2/0)	6.5(3/0)	7(3/0)	8.5(4/0)	9(4/0)

Table 8-31. Execution Timing for Shifting a Memory Operand by 1 or 2

Operation	Cycles
<OP> REG	1(0/0)
<OP>.bwpl (IDX) <OP>.bwpl (++IDX) <OP>.bwpl (REG,IDX)	3.5(1/1)
<OP>.bwpl (IDX1) <OP>.bwpl (IDX3) <OP>.bwpl (IDX2,REG) <OP>.bwpl (IDX3,REG) <OP>.bwpl EXT1 <OP>.bwpl EXT2 <OP>.bwpl EXT3	4(1/1)
<OP>.bwpl [REG,IDX]	5(2/1)
<OP>.bwpl [IDX1] <OP>.bwpl [IDX3] <OP>.bwpl [EXT3]	5.5(2/1)

8.2.24 Rotate Instruction Execution Times (ROL, ROR)

Table 8-32 shows the number of clock cycles required for execution of Rotate operations (ROL, ROR).

Table 8-32. Rotate (ROL, ROR) Execution Timing

Operation	Cycles
<OP> REG	1(0/0)
<OP>.bwpl (IDX) <OP>.bwpl (++IDX) <OP>.bwpl (REG,IDX)	3.5(1/1)
<OP>.bwpl (IDX1) <OP>.bwpl (IDX3) <OP>.bwpl (IDX2,REG) <OP>.bwpl (IDX3,REG) <OP>.bwpl EXT1 <OP>.bwpl EXT2 <OP>.bwpl EXT3	4(1/1)
<OP>.bwpl [REG,IDX]	5(2/1)
<OP>.bwpl [IDX1] <OP>.bwpl [IDX3] <OP>.bwpl [EXT3]	5.5(2/1)

8.2.25 Bit Manipulation Instruction Execution Times (BCLR, BSET, BTGL)

Table 8-33 shows the number of clock cycles required for execution of a Bit-manipulation operation (BCLR, BSET, BTGL).

Table 8-33. Bit-Manipulation (BCLR, BSET, BTGL) Execution Timing

Operation	Cycles
<OP> Di,#opr5i	1.5(0/0)
<OP> REG,#opr5i <OP> REG,Dn	1.5(0/0)
<OP>.bwl (IDX),#opr5i <OP>.bwl (++IDX),#opr5i <OP>.bwl (REG,IDX),#opr5i <OP>.bwl (IDX),Dn <OP>.bwl (++IDX),Dn <OP>.bwl (REG,IDX),Dn	4(1/1)
<OP>.bwl (IDX1),#opr5i <OP>.bwl (IDX3),#opr5i <OP>.bwl (IDX2,REG),#opr5i <OP>.bwl (IDX3,REG),#opr5i <OP>.bwl EXT1,#opr5i <OP>.bwl EXT2,#opr5i <OP>.bwl EXT3,#opr5i <OP>.bwl (IDX1),Dn <OP>.bwl (IDX3),Dn <OP>.bwl (IDX2,REG),Dn <OP>.bwl (IDX3,REG),Dn <OP>.bwl EXT1,Dn <OP>.bwl EXT2,Dn <OP>.bwl EXT3,Dn	4.5(1/1)
<OP>.bwl [REG,IDX],#opr5i <OP>.bwl [REG,IDX],Dn	5.5(2/1)

Table 8-33. Bit-Manipulation (BCLR, BSET, BTGL) Execution Timing

Operation	Cycles
<OP>.bwl [IDX1],#opr5i <OP>.bwl [IDX3],#opr5i <OP>.bwl [EXT3],#opr5i <OP>.bwl [IDX1],Dn <OP>.bwl [IDX3],Dn <OP>.bwl [EXT3],Dn	6(2/1)

8.2.26 Bit Field Instruction Execution Times (BFEXT, BFINS)

Table 8-34 and Table 8-35 show the number of clock cycles required for execution of Bit Field operations (BFEXT, BFINS).

Table 8-34. Bit Field Extract Execution Timing

Operation	Cycles	Operation	Cycles
BFEXT Dd,Ds,#width:offset BFEXT Dd,Ds,Dp	2(0/0)		
BFEXT Dd,REG,#width:offset BFEXT Dd,#IMMe4,#width:offset BFEXT Dd,REG,Dp BFEXT Dd,#IMMe4,Dp	2.5(0/0)	BFEXT REG,Ds,#width:offset BFEXT REG,Ds,REG,Dp	2(0/0)
BFEXT.bwpl Dd,(IDX),#width:offset BFEXT.bwpl Dd,(++IDX),#width:offset BFEXT.bwpl Dd,(REG,IDX),#width:offset BFEXT.bwpl Dd,(IDX),Dp BFEXT.bwpl Dd,(++IDX),Dp BFEXT.bwpl Dd,(REG,IDX),Dp	4(1/0)	BFEXT.bwpl (IDX),Ds,#width:offset BFEXT.bwpl (++IDX),Ds,#width:offset BFEXT.bwpl (REG,IDX),Ds,#width:offset BFEXT.bwpl (IDX),Ds,Dp BFEXT.bwpl (++IDX),Ds,Dp BFEXT.bwpl (REG,IDX),Ds,Dp	3.5(0/1)
BFEXT.bwpl Dd,(IDX1),#width:offset BFEXT.bwpl Dd,(IDX3),#width:offset BFEXT.bwpl Dd,(IDX2,REG),#width:offset BFEXT.bwpl Dd,(IDX3,REG),#width:offset BFEXT.bwpl Dd,EXT1,#width:offset BFEXT.bwpl Dd,EXT2,#width:offset BFEXT.bwpl Dd,EXT3,#width:offset BFEXT.bwpl Dd,(IDX1),Dp BFEXT.bwpl Dd,(IDX3),Dp BFEXT.bwpl Dd,(IDX2,REG),Dp BFEXT.bwpl Dd,(IDX3,REG),Dp BFEXT.bwpl Dd,EXT1,Dp BFEXT.bwpl Dd,EXT2,Dp BFEXT.bwpl Dd,EXT3,Dp	4(1/0)	BFEXT.bwpl (IDX1),Ds,#width:offset BFEXT.bwpl (IDX3),Ds,#width:offset BFEXT.bwpl (IDX2,REG),Ds,#width:offset BFEXT.bwpl (IDX3,REG),Ds,#width:offset BFEXT.bwpl EXT1,Ds,#width:offset BFEXT.bwpl EXT2,Ds,#width:offset BFEXT.bwpl EXT3,Ds,#width:offset BFEXT.bwpl (IDX1),Ds,Dp BFEXT.bwpl (IDX3),Ds,Dp BFEXT.bwpl (IDX2,REG),Ds,Dp BFEXT.bwpl (IDX3,REG),Ds,Dp BFEXT.bwpl EXT1,Ds,Dp BFEXT.bwpl EXT2,Ds,Dp BFEXT.bwpl EXT3,Ds,Dp	3.5(0/1)
BFEXT.bwpl Dd,[REG,IDX],#width:offset BFEXT.bwpl Dd,[REG,IDX],Dp	5.5(2/0)	BFEXT.bwpl [REG,IDX],Ds,#width:offset BFEXT.bwpl [REG,IDX],Ds,Dp	5(1/1)
BFEXT.bwpl Dd,[IDX1],#width:offset BFEXT.bwpl Dd,[IDX3],#width:offset BFEXT.bwpl Dd,[EXT3],#width:offset BFEXT.bwpl Dd,[IDX1],Dp BFEXT.bwpl Dd,[IDX3],Dp BFEXT.bwpl Dd,[EXT3],Dp	5.5(2/0)	BFEXT.bwpl [IDX1],Ds,#width:offset BFEXT.bwpl [IDX3],Ds,#width:offset BFEXT.bwpl [EXT3],Ds,#width:offset BFEXT.bwpl [IDX1],Ds,Dp BFEXT.bwpl [IDX3],Ds,Dp BFEXT.bwpl [EXT3],Ds,Dp	5(1/1)

Table 8-35. Bit Field Insert Execution Timing

Operation	Cycles	Operation	Cycles
BFINS Dd,Ds,#width:offset BFINS Dd,Ds,Dp	2.5(0/0)		
BFINS Dd,REG,#width:offset BFINS Dd,#IMMe4,#width:offset BFINS Dd,REG,Dp BFINS Dd,#IMMe4,Dp	2.5(0/0)	BFINS REG,Ds,#width:offset BFINS REG,Ds,REG,Dp	2.5(0/0)
BFINS.bwpl Dd,(IDX),#width:offset BFINS.bwpl Dd,(++IDX),#width:offset BFINS.bwpl Dd,(REG,IDX),#width:offset BFINS.bwpl Dd,(IDX),Dp BFINS.bwpl Dd,(++IDX),Dp BFINS.bwpl Dd,(REG,IDX),Dp	4(1/0)	BFINS.bwpl (IDX),Ds,#width:offset BFINS.bwpl (++IDX),Ds,#width:offset BFINS.bwpl (REG,IDX),Ds,#width:offset BFINS.bwpl (IDX),Ds,Dp BFINS.bwpl (++IDX),Ds,Dp BFINS.bwpl (REG,IDX),Ds,Dp	5(1/1)
BFINS.bwpl Dd,(IDX1),#width:offset BFINS.bwpl Dd,(IDX3),#width:offset BFINS.bwpl Dd,(IDX2,REG),#width:offset BFINS.bwpl Dd,(IDX3,REG),#width:offset BFINS.bwpl Dd,EXT1,#width:offset BFINS.bwpl Dd,EXT2,#width:offset BFINS.bwpl Dd,EXT3,#width:offset BFINS.bwpl Dd,(IDX1),Dp BFINS.bwpl Dd,(IDX3),Dp BFINS.bwpl Dd,(IDX2,REG),Dp BFINS.bwpl Dd,(IDX3,REG),Dp BFINS.bwpl Dd,EXT1,Dp BFINS.bwpl Dd,EXT2,Dp BFINS.bwpl Dd,EXT3,Dp	4(1/0)	BFINS.bwpl (IDX1),Ds,#width:offset BFINS.bwpl (IDX3),Ds,#width:offset BFINS.bwpl (IDX2,REG),Ds,#width:offset BFINS.bwpl (IDX3,REG),Ds,#width:offset BFINS.bwpl EXT1,Ds,#width:offset BFINS.bwpl EXT2,Ds,#width:offset BFINS.bwpl EXT3,Ds,#width:offset BFINS.bwpl (IDX1),Ds,Dp BFINS.bwpl (IDX3),Ds,Dp BFINS.bwpl (IDX2,REG),Ds,Dp BFINS.bwpl (IDX3,REG),Ds,Dp BFINS.bwpl EXT1,Ds,Dp BFINS.bwpl EXT2,Ds,Dp BFINS.bwpl EXT3,Ds,Dp	5(1/1)
BFINS.bwpl Dd,[REG,IDX],#width:offset BFINS.bwpl Dd,[REG,IDX],Dp	5.5(2/0)	BFINS.bwpl [REG,IDX],Ds,#width:offset BFINS.bwpl [REG,IDX],Ds,Dp	6.5(2/1)
BFINS.bwpl Dd,[IDX1],#width:offset BFINS.bwpl Dd,[IDX3],#width:offset BFINS.bwpl Dd,[EXT3],#width:offset BFINS.bwpl Dd,[IDX1],Dp BFINS.bwpl Dd,[IDX3],Dp BFINS.bwpl Dd,[EXT3],Dp	5.5(2/0)	BFINS.bwpl [IDX1],Ds,#width:offset BFINS.bwpl [IDX3],Ds,#width:offset BFINS.bwpl [EXT3],Ds,#width:offset BFINS.bwpl [IDX1],Ds,Dp BFINS.bwpl [IDX3],Ds,Dp BFINS.bwpl [EXT3],Ds,Dp	6.5(2/1)

8.2.27 Branch Always Instruction Execution Times (BRA)

Table 8-36 shows the number of clock cycles required for execution of the Unconditional Branch instruction (BRA).

The BRA instruction causes a reset of the instruction queue. That means additional cycles to fetch new program-code are required after execution of this instruction (for details please refer to Chapter 4, “Instruction Queue”).

Table 8-36. Unconditional Branch Execution Timing

Operation	Cycles
BRA oprdest	1.5(0/0)

8.2.28 Jump Instruction Execution Times (JMP)

Table 8-37 shows the number of clock cycles required for execution of the Jump instruction (JMP).

The JMP instruction causes a reset of the instruction queue. That means additional cycles to fetch new program-code are required after execution of this instruction (for details please refer to Chapter 4, “Instruction Queue”).

Table 8-37. Jump Execution Timing

Operation	Cycles
JMP EXT24	1.5(0/0)
JMP (IDX) JMP (++IDX) JMP (REG,IDX)	2.0(0/0)
JMP (IDX1) JMP (IDX3) JMP (IDX2,REG) JMP (IDX3,REG) JMP EXT1 JMP EXT2 JMP EXT3	2.5(0/0)
JMP [REG,IDX]	3(1/0)
JMP [IDX1] JMP [IDX3] JMP [EXT3]	3.5(1/0)

8.2.29 Branch on CCR Condition Instruction Execution Times (Bcc)

Table 8-38 shows the number of clock cycles required for execution of a Conditional Branch instruction (BHI/BLS, BCC/BCS, BNE/BEQ, BVC/BVS, BPL/BMI, BGE/BLT or BGT/BLE).

The Bcc instructions cause a reset of the instruction queue if the branch is taken. That means additional cycles to fetch new program-code may be required after execution of one of these instructions (for details please refer to Chapter 4, “Instruction Queue”).

Table 8-38. Conditional Branch Execution Timing

Operation	Cycles (taken)	Cycles (not taken)
Bcc oprdest	1.5(0/0)	1(0/0)

8.2.30 Branch on Bit-Value Instruction Execution Times (BRCLR, BRSET)

Table 8-39 shows the number of clock cycles required for execution of a Branch on Bit-Value instruction (BRCLR, BRSET).

The BRCLR/BRSET instructions cause a reset of the instruction queue if the branch is taken. That means additional cycles to fetch new program-code may be required after execution of one of these instructions (for details please refer to Chapter 4, “Instruction Queue”).

Table 8-39. Branch on Bit-Value Execution Timing

Operation	Cycles (taken)	Cycles (not taken)
<OP> Di,#opr5i,oprdest	3(0/0)	2.5(0/0)
<OP> REG,#opr5i,oprdest <OP> REG,Dn,oprdest	3(0/0)	2.5(0/0)
<OP>.bwl (IDX),#opr5i,oprdest <OP>.bwl (++IDX),#opr5i,oprdest <OP>.bwl (REG,IDX),#opr5i,oprdest <OP>.bwl (IDX),Dn,oprdest <OP>.bwl (++IDX),Dn,oprdest <OP>.bwl (REG,IDX),Dn,oprdest	4.5(1/0)	4(1/0)
<OP>.bwl (IDX1),#opr5i,oprdest <OP>.bwl (IDX3),#opr5i,oprdest <OP>.bwl (IDX2,REG),#opr5i,oprdest <OP>.bwl (IDX3,REG),#opr5i,oprdest <OP>.bwl EXT1,#opr5i,oprdest <OP>.bwl EXT2,#opr5i,oprdest <OP>.bwl EXT3,#opr5i,oprdest <OP>.bwl (IDX1),Dn,oprdest <OP>.bwl (IDX3),Dn,oprdest <OP>.bwl (IDX2,REG),Dn,oprdest <OP>.bwl (IDX3,REG),Dn,oprdest <OP>.bwl EXT1,Dn,oprdest <OP>.bwl EXT2,Dn,oprdest <OP>.bwl EXT3,Dn,oprdest	5(1/0)	4.5(1/0)
<OP>.bwl [REG,IDX],#opr5i,oprdest <OP>.bwl [REG,IDX],Dn,oprdest	6(2/0)	5.5(2/0)
<OP>.bwl [IDX1],#opr5i,oprdest <OP>.bwl [IDX3],#opr5i,oprdest <OP>.bwl [EXT3],#opr5i,oprdest <OP>.bwl [IDX1],Dn,oprdest <OP>.bwl [IDX3],Dn,oprdest <OP>.bwl [EXT3],Dn,oprdest	6.5(2/0)	6(2/0)

8.2.31 Decrement and Branch Instruction Execution Times (DBcc)

Table 8-40 shows the number of clock cycles required for execution of a Decrement and Branch instruction (DBcc).

The DBcc instruction causes a reset of the instruction queue if the branch is taken. That means additional cycles to fetch new program-code may be required after execution of one of these instructions (for details please refer to Chapter 4, “Instruction Queue”).

Table 8-40. Decrement and Branch Execution Timing

Operation	Cycles (taken)	Cycles (not taken)
DBcc Di,oprdest DBcc xy,oprdest DBcc REG,oprdest	2.5(0/0)	2(0/0)
DBcc.bwpl (IDX),oprdest DBcc,bwpl (++IDX),oprdest DBcc.bwpl (REG,IDX),oprdest	4.5(1/1)	4(1/1)

Table 8-40. Decrement and Branch Execution Timing

Operation	Cycles (taken)	Cycles (not taken)
DBcc.bwpl (IDX1),oprdest DBcc.bwpl (IDX3),oprdest DBcc.bwpl (IDX2,REG),oprdest DBcc.bwpl (IDX3,REG),oprdest DBcc.bwpl EXT1,oprdest DBcc.bwpl EXT2,oprdest DBcc.bwpl EXT3,oprdest	5(1/1)	4.5(1/1)
DBcc.bwpl [REG,IDX],oprdest	6(2/1)	5.5(2/1)
DBcc.bwpl [IDX1],oprdest DBcc.bwpl [IDX3],oprdest DBcc.bwpl [EXT3],oprdest	6.5(2/1)	6(2/1)

8.2.32 Test and Branch Instruction Execution Times (TBcc)

Table 8-41 shows the number of clock cycles required for execution of a Test and Branch instruction (TBcc).

The TBcc instruction causes a reset of the instruction queue if the branch is taken. That means additional cycles to fetch new program-code may be required after execution of one of these instructions (for details please refer to Chapter 4, “Instruction Queue”).

Table 8-41. Test and Branch Execution Timing

Operation	Cycles (taken)	Cycles (not taken)
TBcc Di,oprdest TBcc xy,oprdest TBcc REG,oprdest	2.5(0/0)	2(0/0)
TBcc.bwpl (IDX),oprdest TBcc,bwpl (++IDX),oprdest TBcc.bwpl (REG,IDX),oprdest	4(1/0)	3.5(1/0)
TBcc.bwpl (IDX1),oprdest TBcc.bwpl (IDX3),oprdest TBcc.bwpl (IDX2,REG),oprdest TBcc.bwpl (IDX3,REG),oprdest TBcc.bwpl EXT1,oprdest TBcc.bwpl EXT2,oprdest TBcc.bwpl EXT3,oprdest	4.5(1/0)	4(1/0)
TBcc.bwpl [REG,IDX],oprdest	5.5(2/0)	5(2/0)
TBcc.bwpl [IDX1],oprdest TBcc.bwpl [IDX3],oprdest TBcc.bwpl [EXT3],oprdest	6(2/0)	5.5(2/0)

8.2.33 Jump Subroutine Instruction Execution Times (JSR)

Table 8-42 shows the number of clock cycles required for execution of the Jump-to-Subroutine instruction (JSR).

The JSR instruction causes a reset of the instruction queue. That means additional cycles to fetch new program-code are required after execution of this instruction (for details please refer to Chapter 4, “Instruction Queue”).

Table 8-42. Jump-to-Subroutine Execution Timing

Operation	Cycles
JSR EXT24	2.5(0/1)
JSR (IDX) JSR (++IDX) JSR (REG,IDX)	2.5(0/1)
JSR (IDX1) JSR (IDX3) JSR (IDX2,REG) JSR (IDX3,REG) JSR EXT1 JSR EXT2 JSR EXT3	3(0/1)
JSR [REG,IDX]	4(1/1)
JSR [IDX1] JSR [IDX3] JSR [EXT3]	4.5(1/1)

8.2.34 Branch Subroutine Instruction Execution Times (BSR)

Table 8-43 shows the number of clock cycles required for execution of the Branch-to-Subroutine instruction (BSR).

The BSR instruction causes a reset of the instruction queue. That means additional cycles to fetch new program-code are required after execution of this instruction (for details please refer to Chapter 4, “Instruction Queue”).

Table 8-43. Branch-to-Subroutine Execution Timing

Operation	Cycles
BSR oprdest	2.5(0/1)

8.2.35 Return from Subroutine Instruction Execution Times (RTS)

Table 8-44 shows the number of clock cycles required for execution of the Return-from-Subroutine instruction (RTS).

The RTS instruction causes a reset of the instruction queue. That means additional cycles to fetch new program-code are required after execution of this instruction (for details please refer to Chapter 4, “Instruction Queue”).

Table 8-44. Return-fromSubroutine Execution Timing

Operation	Cycles
RTS	3(1/0)

8.2.36 Machine Exception Sequence Execution Times

Table 8-45 shows the number of clock cycles required for execution of the Machine Exception Sequence.

The Machine Exception Sequence causes a reset of the instruction queue. That means additional cycles to fetch new program-code are required after execution of this sequence (for details please refer to Chapter 4, “Instruction Queue”).

Table 8-45. Machine Exception Execution Timing

Operation	Cycles
<Machine Exception>	4(1/0)

8.2.37 Hardware Interrupt Sequence Execution Times

Table 8-46 shows the number of clock cycles required for execution of the Hardware Interrupt Sequence.

The Hardware Interrupt Sequence causes a reset of the instruction queue. That means additional cycles to fetch new program-code may be required after execution of this sequence (for details please refer to Chapter 4, “Instruction Queue”). Due to separated busses for program-code and data the program-code fetches can be done in parallel to the exception stacking sequence. Ideally, the source of the program-code for the Interrupt Service Routine (usually NVM) differs from the destination of the stack cycles (usually SRAM) so the cycles required for fetching the new program-code are not visible as an additional delay before instruction execution continues.

Table 8-46. No-Operation Execution Timing

Operation	Cycles
<Hardware Interrupt>	8(1/8)

8.2.38 Unimplemented Op-code Trap Execution Times (SPARE, TRAP)

Table 8-47 shows the number of clock cycles required for execution of the No-Operation instruction (NOP).

The SPARE and TRAP instructions cause a reset of the instruction queue. That means additional cycles to fetch new program-code may be required after execution of one of these op-codes (for details please refer to Chapter 4, “Instruction Queue”). Due to separated busses for program-code and data the program-code fetches can be done in parallel to the exception stacking sequence. Ideally, the source of the program-code for the Interrupt Service Routine (usually NVM) differs from the destination of the stack cycles (usually SRAM) so the cycles required for fetching the new program-code are not visible as an additional delay before instruction execution continues.

Table 8-47. Unimplemented Op-code Trap Execution Timing

Operation	Cycles
SPARE TRAP num	8(1/8)

8.2.39 Software Interrupt and System Call Instruction Execution Times (SWI, SYS)

Table 8-48 shows the number of clock cycles required for execution of the No-Operation instruction (NOP).

The SWI and SYS instructions cause a reset of the instruction queue. That means additional cycles to fetch new program-code may be required after execution of one of these instructions (for details please refer to Chapter 4, “Instruction Queue”). Due to separated busses for program-code and data the program-code fetches can be done in parallel to the exception stacking sequence. Ideally, the source of the program-code for the Interrupt Service Routine (usually NVM) differs from the destination of the stack cycles (usually SRAM) so the cycles required for fetching the new program-code are not visible as an additional delay before instruction execution continues.

Table 8-48. Software Interrupt Execution Timing

Operation	Cycles
SWI SYS	8(1/8)

8.2.40 Return from Interrupt Instruction Execution Times (RTI)

Table 8-49 shows the number of clock cycles required for execution of the Return-from-Interrupt instruction (RTI).

The RTI instruction causes a reset of the instruction queue. That means additional cycles to fetch new program-code are required after execution of this instruction (for details please refer to Chapter 4, “Instruction Queue”).

Table 8-49. Return-from-Interrupt Execution Timing

Operation	Cycles
RTI (no pending interrupt)	6.5(8/0)
RTI (pending interrupt)	8.5(9/0)

8.2.41 Low Power Instruction Execution Times (WAI, STOP)

Table 8-50 shows the number of clock cycles required for execution of Low-power Stop or Wait instructions (STOP, WAI).

The STOP or WAI instructions cause a reset of the instruction queue. That means additional cycles to fetch new program-code are required after execution of one of these instructions if the low-power state is left with an interrupt (for details please refer to Chapter 4, “Instruction Queue”).

Table 8-50. Low-Power Stop/Wait Execution Timing

Operation	Cycles	
WAI (CPU in supervisor state)	Enter Wait	5(0/8)
	Exit Wait (interrupt)	3(1/0)
	Exit Wait (continue)	1(0/0)
WAI (CPU in user state)	1(0/0)	
STOP (STOP enabled and CPU in supervisor state)	Enter Stop	5(0/8)
	Exit Stop (interrupt)	3(1/0)
	Exit Stop (continue)	1(0/0)
STOP (STOP disabled or CPU in user state)	1(0/0)	

8.2.42 Go to Active Background Debug Mode Instruction Execution Times (BGND)

Table 8-51 shows the number of clock cycles required for execution of the Go to Active Background Debug Mode instruction (BGND).

The BGND instruction causes a reset of the instruction queue. That means additional cycles to fetch new program-code are required after execution of this instruction (for details please refer to Chapter 4, “Instruction Queue”).

Table 8-51. Go to Active Background Debug Mode Execution Timing

Operation	Cycles
BGND (BDC disabled)	1(0/0)
BGND (BDC enabled)	S12Z CPU halted until BDC ‘Go’ command is executed

Chapter 9
Data Bus Operation

9.1 Introduction

The S12Z CPU features two independent bus interfaces. One is used for fetching program code, the other is used to transfer data from/to the CPU to/from memory or peripheral modules.

The S12Z CPU program bus interface is restricted to aligned 32-bit read-transfers.

The S12Z CPU data bus interface, while also operating on 32-bit address boundaries, supports transfers of all native data types. That means 8-bit, 16-bit, 24-bit and 32-bit transfers are supported.

9.2 Access Timing

The S12Z CPU data bus supports both read and write accesses. Each kind of access has a minimum number of bus-clock cycles defined which are required to complete the access:

- Write accesses take at least 0.5 bus-clock cycles.
- Read accesses take at least 1 bus-clock cycle.

However, it must be mentioned that the S12Z CPU data bus interface features mechanisms to add wait-cycles to adjust the access timing to different timing requirements of peripheral modules and memories.

Please refer to the Device Reference Manual for details on implemented bus access timing for different bus targets.

9.3 Data Transfer Alignment

All data types supported by the S12Z CPU data bus interface must be Byte-aligned. The alignment of the operand data on the data bus is done automatically, depending on address alignment.

However, due to the fact that all data bus transfers are done on 32-bit address boundaries, there are combinations of address alignment and transfer sizes which require the data transfer to be split into two consecutive bus accesses. The second bus access is then done on the next 32-bit address boundary. The two accesses required to complete a split bus transfer are initiated back-to-back.

Please refer to Table 9-1 for details on transfer sizes, address alignment and number of bus accesses required to complete the transfer. The alignment of the operand bytes on the data bus is done automatically and is only listed here for information.

Table 9-1. Data Transfer Alignment for Read and Write Cycles

Transfer Size	Address Alignment	Number of Accesses Required	Data Alignment[1]			
	[A1:A0]		[D31:D24]	[D23:D16]	[D15:D8]	[D7:D0]
Byte (8-bit)	00	1	[OP7:OP0]	–	–	–
	01	1	–	[OP7:OP0]	–	–
	10	1	–	–	[OP7:OP0]	–
	11	1	–	–	–	[OP7:OP0]
Word (16-bit)	00	1	[OP15:OP8]	[OP7:OP0]	–	–
	01	1	–	[OP15:OP8]	[OP7:OP0]	–
	10	1	–	–	[OP15:OP8]	[OP7:OP0]
	11	2	[OP7:OP0]	–	–	[OP15:OP8]
Pointer (24-bit)	00	1	[OP23:OP16]	[OP15:OP8]	[OP7:OP0]	–
	01	1	-	[OP23:OP16]	[OP15:OP8]	[OP7:OP0]
	10	2	[OP7:OP0]	–	[OP23:OP16]	[OP15:OP8]
	11	2	[OP15:OP8]	[OP7:OP0]	–	[OP23:OP16]
Long Word (32-bit)	00	1	[OP31:OP24]	[OP23:OP16]	[OP15:OP8]	[OP7:OP0]
	01	2	[OP7:OP0]	[OP31:OP24]	[OP23:OP16]	[OP15:OP8]
	10	2	[OP15:OP8]	[OP7:OP0]	[OP31:OP24]	[OP23:OP16]
	11	2	[OP23:OP16]	[OP15:OP8]	[OP7:OP0]	[OP31:OP24]

1 The operand bytes in shaded fields are transferred in the second access of a split bus transfer

Appendix A
Instruction Reference

A.1 Introduction

This appendix provides quick reference tables for the instruction set, opcode map, and postbyte encoding. The nomenclature used in the instruction descriptions in Table A-1 are explained in Section 1.3, “Symbols and Notation”.

A.2 S12Z Instruction Set Summary Table

The table below provides a summary of all CPU S12Z instructions, their operation, addressing modes, machine coding and condition code effects.

Table A-1. S12Z Instruction Set Summary (Sheet 1 of 17)

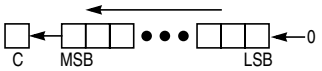
Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	 N Z V C
ABS *Di*	$\lvert(D_i)\rvert \Rightarrow D_i$ Replace D_i with the Absolute Value of D_i	INH	1B 4n	– – – –	Δ Δ Δ –
ADC *Di*,#*oprimmsz*	$(D_i) + (M) + C \Rightarrow D_i$ Add with Carry to D_i Memory operand M is the same size as D_i	IMM1 IMM2 IMM4	1B 5p i1 1B 5p i2 i1 1B 5p i4 i3 i2 i1	– – – –	Δ Δ Δ Δ
ADC *Di*,*oprmemreg*	M can be a memory operand or another register D_j	OPR OPR1 OPR2 OPR3	1B 6n xb 1B 6n xb x1 1B 6n xb x2 x1 1B 6n xb x3 x2 x1		
ADD *Di*,#*oprimmsz*	$(D_i) + (M) \Rightarrow D_i$ Add without Carry to D_i Memory operand M is the same size as D_i	IMM1 IMM2 IMM4	5p i1 5p i2 i1 5p i4 i3 i2 i1	– – – –	Δ Δ Δ Δ
ADD *Di*,*oprmemreg*	M can be a memory operand or another register D_j	OPR OPR1 OPR2 OPR3	6n xb 6n xb x1 6n xb x2 x1 6n xb x3 x2 x1		
AND *Di*,#*oprimmsz*	(D_i) & $(M) \Rightarrow D_i$ Bitwise AND D_i with Memory Memory operand M is the same size as D_i	IMM1 IMM2 IMM4	5p i1 5p i2 i1 5p i4 i3 i2 i1	– – – –	Δ Δ 0 –
AND *Di*,*oprmemreg*	M can be a memory operand or another register D_j if D_j wider, AND D_i with low portion of D_j if D_j narrower, AND D_i with zero-extended D_j	OPR OPR1 OPR2 OPR3	6q xb 6q xb x1 6q xb x2 x1 6q xb x3 x2 x1		
ANDCC #*opr8i*	(CCL) & (M) ⇒ CCL Bitwise AND CCL with immediate byte in Memory (S, X, and I can only be changed in supervisor state)	IMM1	CE i1	⇓ ⇓ – ⇓ s s s	⇓ ⇓ ⇓ ⇓
ASL *Dd*,*Ds*,*Dn*	 Arithmetic Shift Left D_s or memory, 0 to n positions where n+1 is the number of bits in the operand The result is saved in register D_d (encoded in the opcode). n is specified in postbyte sb, sb+xb, a byte-sized memory operand, or register D_n If the destination is wider than the source, sign-extend to the width of the destination before shifting. If the destination is narrower than the source, shift and then truncate to the width of the destination. In the case of two OPR operands, the parameter n operand is the last operand in the source form and the object code.	REG-REG	1n sb xb	– – – –	Δ Δ Δ Δ
ASL *Dd*,*Ds*,#*opr1i* ASL *Dd*,*Ds*,#*opr5i*		REG-IMM REG-IMM	1n sb 1n sb xb		
ASL.*bwpl Dd*,*oprmemreg*,#*opr1i*		OPR-IMM OPR1-IMM OPR2-IMM OPR3-IMM	1n sb xb 1n sb xb x1 1n sb xb x2 x1 1n sb xb x3 x2 x1		
ASL.*bwpl Dd*,*oprmemreg*,#*opr5i*		OPR-IMM OPR1-IMM OPR2-IMM OPR3-IMM	1n sb xb xb 1n sb xb x1 xb 1n sb xb x2 x1 xb 1n sb xb x3 x2 x1 x1 xb		
ASL.*bwpl Dd*,*oprmemreg*,*oprmemreg*		OPR-OPR OPR-OPR1 OPR-OPR2 OPR-OPR3	1n sb xb xb 1n sb xb xb x1 1n sb xb xb x2 x1 1n sb xb xb x3 x2 x1		
		OPR1-OPR OPR1-OPR1 OPR1-OPR2 OPR1-OPR3	1n sb xb x1 xb 1n sb xb x1 xb x1 1n sb xb x1 xb x2 x1 1n sb xb x1 xb x3 x2 x1		
		OPR2-OPR OPR2-OPR1 OPR2-OPR2 OPR2-OPR3	1n sb xb x2 x1 xb 1n sb xb x2 x1 xb x1 1n sb xb x2 x1 xb x2 x1 1n sb xb x2 x1 xb x3 x2 x1		
		OPR3-OPR OPR3-OPR1 OPR3-OPR2 OPR3-OPR3	1n sb xb x3 x2 x1 xb 1n sb xb x3 x2 x1 xb x1 1n sb xb x3 x2 x1 xb x2 x1 1n sb xb x3 x2 x1 xb x3 x2 x1		
ASL.*bwpl oprmemreg*,#*opr1i*	Shift memory location by 1 or 2 position. Source and destination are the same memory operand	OPR-IMM OPR1-IMM OPR2-IMM OPR3-IMM	1n sb xb 1n sb xb x1 1n sb xb x2 x1 1n sb xb x3 x2 x1		

Table A-1. S12Z Instruction Set Summary (Sheet 2 of 17)

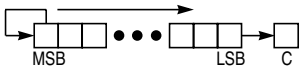
Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	Condition Codes N Z V C
ASR *Dd*,*Ds*,*Dn*		REG-REG	1n sb xb	– – – –	Δ Δ Δ Δ
ASR *Dd*,*Ds*,#*opr1i*		REG-IMM	1n sb		
ASR *Dd*,*Ds*,#*opr5i*		REG-IMM	1n sb xb		
ASR.*bwpl Dd*,*oprmemreg*,#*opr1i*	Arithmetic Shift Right D_s or memory, 0 to n positions where n+1 is the number of bits in the operand The result is saved in register D_d (encoded in the opcode). n is specified in postbyte sb, sb+xb, a byte-sized memory operand, or register D_n If the destination is wider than the source, sign-extend to the width of the destination. If the destination is narrower than the source, shift and then truncate to the width of the destination. In the case of two OPR operands, the parameter n operand is the last operand in the source form and the object code.	OPR-IMM	1n sb xb		
		OPR1-IMM	1n sb xb x1		
		OPR2-IMM	1n sb xb x2 x1		
		OPR3-IMM	1n sb xb x3 x2 x1		
ASR.*bwpl Dd*,*oprmemreg*,#*opr5i*		OPR-IMM	1n sb xb xb		
		OPR1-IMM	1n sb xb x1 xb		
		OPR2-IMM	1n sb xb x2 x1 xb		
		OPR3-IMM	1n sb xb x3 x2 x1 x1 xb		
ASR.*bwpl Dd*,*oprmemreg*,*oprmemreg*		OPR-OPR	1n sb xb xb		
		OPR-OPR1	1n sb xb xb x1		
		OPR-OPR2	1n sb xb xb x2 x1		
		OPR-OPR3	1n sb xb xb x3 x2 x1		
		OPR1-OPR	1n sb xb x1 xb		
		OPR1-OPR1	1n sb xb x1 xb x1		
		OPR1-OPR2	1n sb xb x1 xb x2 x1		
		OPR1-OPR3	1n sb xb x1 xb x3 x2 x1		
		OPR2-OPR	1n sb xb x2 x1 xb		
		OPR2-OPR1	1n sb xb x2 x1 xb x1		
		OPR2-OPR2	1n sb xb x2 x1 xb x2 x1		
		OPR2-OPR3	1n sb xb x2 x1 xb x3 x2 x1		
		OPR3-OPR	1n sb xb x3 x2 x1 xb		
		OPR3-OPR1	1n sb xb x3 x2 x1 xb x1		
		OPR3-OPR2	1n sb xb x3 x2 x1 xb x2 x1		
		OPR3-OPR3	1n sb xb x3 x2 x1 xb x3 x2 x1		
ASR.*bwpl oprmemreg*,#*opr1i*	Shift memory location by 1 or 2 position. Source and destination are the same memory operand	OPR-IMM	1n sb xb		
		OPR1-IMM	1n sb xb x1		
		OPR2-IMM	1n sb xb x2 x1		
		OPR3-IMM	1n sb xb x3 x2 x1		
BCC *oprdest*	Branch if Carry Clear (if C = 0)	R7	24 rb	– – – –	– – – –
		R15	24 rb r1		
BCLR *Di*,#*opr5i*	(M) & ~bitn ⇒ M or (D_i) & ~bitn ⇒ D_i	REG-IMM	EC bm	– – – –	Δ Δ 0 Δ
BCLR.*bwl oprmemreg*,#*opr5i*	Clear Bit n in Memory or in D_i where n is the number of the bit to be cleared n is specified in an immediate value or D_n n is encoded in the postbyte (sb) ~bitn is a mask with all bits except bit n set N and Z set/cleared based on the result, V cleared C equal the original value of bitn in M or D_i (semaphore)	OPR-IMM	EC bm xb		
		OPR1-IMM	EC bm xb x1		
		OPR2-IMM	EC bm xb x2 x1		
		OPR3-IMM	EC bm xb x3 x2 x1		
BCLR.*bwl oprmemreg*,*Dn*		OPR-REG	EC bm xb		
		OPR1-REG	EC bm xb x1		
		OPR2-REG	EC bm xb x2 x1		
		OPR3-REG	EC bm xb x3 x2 x1		
BCS *oprdest*	Branch if Carry Set (if C = 1)	R7	25 rb	– – – –	– – – –
		R15	25 rb r1		
BEQ *oprdest*	Branch if Equal (if Z = 1)	R7	27 rb	– – – –	– – – –
		R15	27 rb r1		

Table A-1. S12Z Instruction Set Summary (Sheet 3 of 17)

Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	Condition Codes N Z V C
BFEXT *Dd,Ds,Dp*	Extract bit field with width w and offset o from D_s or a memory operand, and store it into the low order bits of D_d or memory (filling unused bits with 0). Operands in the source code are in the order destination, source, parameter Parameter is encode in the low 10 bits of D_p or an immediate operand as two 5-bit values w:o w=0 is treated as 32 (0b00010 01000) means 2 bits beginning at bit-8 The source operand or destination operand must be a register (memory to memory not allowed)	RG-RG-RG	1B 0q bb	– – – –	Δ Δ 0 –
BFEXT *Dd,Ds,#width:offset*		RG-RG-IMM	1B 0q bb i1		
BFEXT.*bwpl Dd,oprmemreg,Dp*		RG-OP-RG RG-OP1-RG RG-OP2-RG RG-OP3-RG	1B 0q bb xb 1B 0q bb xb x1 1B 0q bb xb x2 x1 1B 0q bb xb x3 x2 x1		
BFEXT.*bwpl oprmemreg,Ds,Dp*		OP-RG-RG OP1-RG-RG OP2-RG-RG OP3-RG-RG	1B 0q bb xb 1B 0q bb xb x1 1B 0q bb xb x2 x1 1B 0q bb xb x3 x2 x1		
BFEXT.*bwpl Dd,oprmemreg,#width:offset*		RG-OP-IMM RG-OP1-IMM RG-OP2-IMM RG-OP3-IMM	1B 0q bb i1 xb 1B 0q bb i1 xb x1 1B 0q bb i1 xb x2 x1 1B 0q bb i1 xb x3 x2 x1		
BFEXT.*bwpl oprmemreg,Ds,#width:offset*		OP-RG-IMM OP1-RG-IMM OP2-RG-IMM OP3-RG-IMM	1B 0q bb i1 xb 1B 0q bb i1 xb x1 1B 0q bb i1 xb x2 x1 1B 0q bb i1 xb x3 x2 x1		
BFINS *Dd,Ds,Dp*	Insert bit field with width w from the low order bits of D_s or a memory operand into D_d or a memory operand beginning at offset bit number o. Operands in the source code are in the order destination, source, parameter Parameter is encode in the low 10 bits of D_p or an immediate operand as two 5-bit values w:o w=0 is treated as 32 (0b00010 01000) means 2 bits beginning at bit-8 The source operand or destination operand must be a register (memory to memory not allowed)	RG-RG-RG	1B 0q bb	– – – –	Δ Δ 0 –
BFINS *Dd,Ds,#width:offset*		RG-RG-IMM	1B 0q bb i1		
BFINS.*bwpl Dd,oprmemreg,Dp*		RG-OP-RG RG-OP1-RG RG-OP2-RG RG-OP3-RG	1B 0q bb xb 1B 0q bb xb x1 1B 0q bb xb x2 x1 1B 0q bb xb x3 x2 x1		
BFINS.*bwpl oprmemreg,Ds,Dp*		OP-RG-RG OP1-RG-RG OP2-RG-RG OP3-RG-RG	1B 0q bb xb 1B 0q bb xb x1 1B 0q bb xb x2 x1 1B 0q bb xb x3 x2 x1		
BFINS.*bwpl Dd,oprmemreg,#width:offset*		RG-OP-IMM RG-OP1-IMM RG-OP2-IMM RG-OP3-IMM	1B 0q bb i1 xb 1B 0q bb i1 xb x1 1B 0q bb i1 xb x2 x1 1B 0q bb i1 xb x3 x2 x1		
BFINS.*bwpl oprmemreg,Ds,#width:offset*		OP-RG-IMM OP1-RG-IMM OP2-RG-IMM OP3-RG-IMM	1B 0q bb i1 xb 1B 0q bb i1 xb x1 1B 0q bb i1 xb x2 x1 1B 0q bb i1 xb x3 x2 x1		
BGE *oprdest*	Branch if Greater Than or Equal (if N ^ V = 0) (signed)	R7 R15	2C rb 2C rb r1	– – – –	– – – –
BGND	Place CPU in Background Mode	INH	00	– – – –	– – – –
BGT *oprdest*	Branch if Greater Than (if Z | (N ^ V) = 0) (signed)	R7 R15	2E rb 2E rb r1	– – – –	– – – –
BHI *oprdest*	Branch if Higher (if C | Z = 0) (unsigned)	R7 R15	22 rb 22 rb r1	– – – –	– – – –
BHS *oprdest*	Branch if Higher or Same (same function as BCC) (if C = 0) (unsigned)	R7 R15	24 rb 24 rb r1	– – – –	– – – –
BIT *Di,#oprimmsz*	(D_i) & (M) Bitwise AND D_i with Memory Memory operand M is the same size as D_i M can be a memory operand or another register D_j if D_j wider, AND D_i with low portion of D_j if D_j narrower, AND D_i with zero-extended D_j Does not change D_i, D_j, or Memory	IMM1 IMM2 IMM4	1B 5p i1 1B 5p i2 i1 1B 5p i4 i3 i2 i1	– – – –	Δ Δ 0 –
BIT *Di,oprmemreg*		OPR OPR1 OPR2 OPR3	1B 6q xb 1B 6q xb x1 1B 6q xb x2 x1 1B 6q xb x3 x2 x1		
BLE *oprdest*	Branch if Less Than or Equal (if Z | (N ^ V) = 1) (signed)	R7 R15	2F rb 2F rb r1	– – – –	– – – –
BLO *oprdest*	Branch if Lower (same function as BCS) (if C = 1) (unsigned)	R7 R15	25 rb 25 rb r1	– – – –	– – – –
BLS *oprdest*	Branch if Lower or Same (if C | Z = 1) (unsigned)	R7 R15	23 rb 23 rb r1	– – – –	– – – –
BLT *oprdest*	Branch if Less Than (if N ^ V = 1) (signed)	R7 R15	2D rb 2D rb r1	– – – –	– – – –
BMI *oprdest*	Branch if Minus (if N = 1)	R7 R15	2B rb 2B rb r1	– – – –	– – – –
BNE *oprdest*	Branch if Not Equal (if Z = 0)	R7 R15	26 rb 26 rb r1	– – – –	– – – –

Table A-1. S12Z Instruction Set Summary (Sheet 4 of 17)

Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	Condition Codes N Z V C
BPL *oprdest*	Branch if Plus (if N = 0)	R7 R15	2A rb 2A rb r1	– – – –	– – – –
BRA *oprdest*	Branch Always (if 1 = 1)	R7 R15	20 rb 20 rb r1	– – – –	– – – –
BRCLR *Di*,#*opr5i*,*oprdest*	Branch if (M) & bitn = 0 or if (D_i) & bitn = 0 Test Bit n in Memory or in D_i and branch if clear	REG-IMM-R7 REG-IMM-R15	02 bm rb 02 bm rb r1	– – – –	– – – Δ
BRCLR.*bwl oprmemreg*,#*opr5i*,*oprdest*	n is specified in an immediate value or D_n n is encoded in the postbyte (sb) bitn is a mask with only bit n set Branch offset is 7 bits or 15 bits	OP-IMM-R7 OP-IMM-R15 OP1-IMM-R7 OP1-IMM-R15 OP2-IMM-R7 OP2-IMM-R15 OP3-IMM-R7 OP3-IMM-R15	02 bm xb rb 02 bm xb rb r1 02 bm xb x1 rb 02 bm xb x1 rb r1 02 bm xb x2 x1 rb 02 bm xb x2 x1 rb r1 02 bm xb x3 x2 x1 rb 02 bm xb x3 x2 x1 rb r1		
BRCLR.*bwl oprmemreg*,*Dn*,*oprdest*		OP-REG-R7 OP-REG-R15 OP1-REG-R7 OP1-REG-R15 OP2-REG-R7 OP2-REG-R15 OP3-REG-R7 OP3-REG-R15	02 bm xb rb 02 bm xb rb r1 02 bm xb x1 rb 02 bm xb x1 rb r1 02 bm xb x2 x1 rb 02 bm xb x2 x1 rb r1 02 bm xb x3 x2 x1 rb 02 bm xb x3 x2 x1 rb r1		
BRSET *Di*,#*opr5i*,*oprdest*	Branch if (M) & bitn = 0 or if (D_i) & bitn = 0 Test Bit n in Memory or in D_i and branch if set	REG-IMM-R7 REG-IMM-R15	03 bm rb 03 bm rb r1	– – – –	– – – Δ
BRSET.*bwl oprmemreg*,#*opr5i*,*oprdest*	n is specified in an immediate value or D_n n is encoded in the postbyte (sb) bitn is a mask with only bit n set Branch offset is 7 bits or 15 bits	OP-IMM-R7 OP-IMM-R15 OP1-IMM-R7 OP1-IMM-R15 OP2-IMM-R7 OP2-IMM-R15 OP3-IMM-R7 OP3-IMM-R15	03 bm xb rb 03 bm xb rb r1 03 bm xb x1 rb 03 bm xb x1 rb r1 03 bm xb x2 x1 rb 03 bm xb x2 x1 rb r1 03 bm xb x3 x2 x1 rb 03 bm xb x3 x2 x1 rb r1		
BRSET.*bwl oprmemreg*,*Dn*,*oprdest*		OP-REG-R7 OP-REG-R15 OP1-REG-R7 OP1-REG-R15 OP2-REG-R7 OP2-REG-R15 OP3-REG-R7 OP3-REG-R15	03 bm xb rb 03 bm xb rb r1 03 bm xb x1 rb 03 bm xb x1 rb r1 03 bm xb x2 x1 rb 03 bm xb x2 x1 rb r1 03 bm xb x3 x2 x1 rb 03 bm xb x3 x2 x1 rb r1		
BSET *Di*,#*opr5i*	(M) | bitn ⇒ M or (D_i) | bitn ⇒ D_i	REG-IMM	ED bm	– – – –	Δ Δ 0 Δ
BSET.*bwl oprmemreg*,#*opr5i*	Set Bit n in Memory or in D_i where n is the number of the bit to be set n is specified in an immediate value or D_n n is encoded in the postbyte (sb) bitn is a mask with only bit n set N and Z set/cleared based on the result, V cleared C equal the original value of bitn in M or D_i (semaphore)	OPR-IMM OPR1-IMM OPR2-IMM OPR3-IMM	ED bm xb ED bm xb x1 ED bm xb x2 x1 ED bm xb x3 x2 x1		
BSET.*bwl oprmemreg*,*Dn*		OPR-REG OPR1-REG OPR2-REG OPR3-REG	ED bm xb ED bm xb x1 ED bm xb x2 x1 ED bm xb x3 x2 x1		
BSR *oprdest*	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP):M(SP+1):M(SP+2); Subroutine Address ⇒ PC Branch to Subroutine	R7 R15	21 rb 21 rb r1	– – – –	– – – –
BTGL *Di*,#*opr5i*	(M) ^ bitn ⇒ M or (D_i) ^ bitn ⇒ D_i	REG-IMM	EE bm	– – – –	Δ Δ 0 Δ
BTGL.*bwl oprmemreg*,#*opr5i*	Toggle Bit n in Memory or in D_i where n is the number of the bit to be changed n is specified in an immediate value or D_n n is encoded in the postbyte (sb) bitn is a mask with only bit n set N and Z set/cleared based on the result, V cleared C equal the original value of bitn in M or D_i (semaphore)	OPR-IMM OPR1-IMM OPR2-IMM OPR3-IMM	EE bm xb EE bm xb x1 EE bm xb x2 x1 EE bm xb x3 x2 x1		
BTGL.*bwl oprmemreg*,*Dn*		OPR-REG OPR1-REG OPR2-REG OPR3-REG	EE bm xb EE bm xb x1 EE bm xb x2 x1 EE bm xb x3 x2 x1		
BVC *oprdest*	Branch if Overflow Bit Clear (if V = 0)	R7 R15	28 rb 28 rb r1	– – – –	– – – –

Table A-1. S12Z Instruction Set Summary (Sheet 5 of 17)

Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	Condition Codes N Z V C
BVS *oprdest*	Branch if Overflow Bit Set (if V = 1)	R7 R15	29 rb 29 rb r1	– – – –	– – – –
CLB *cpureg*,*cpureg*	count leading sign bits count leading sign bits of (r1) and put the result into r2 result is either a positive number or zero only data-registers D0~D7 are allowed for r1 and r2	REG-REG	1B 91 cb	– – – –	0 Δ 0 –
CLC	0 ⇒ C *Translates to* ANDCC #$FE	IMM1	CE FE	– – – –	– – – 0
CLI	0 ⇒ I; (I bit can only be changed in supervisor state) *Translates to* ANDCC #$EF (enables I interrupts)	IMM1	CE EF	– – – 0 s	– – – –
CLR *Di*	0 ⇒ D_i Clear data register D_i	INH	3q	– – – –	0 1 0 0
CLR.*bwpl oprmemreg*	0 ⇒ M Clear Memory operand M Memory operand M can be 1, 2, 3, or 4 bytes use .B for D0,D1; .W for D2~D5; and .L for D6,D7	OPR OPR1 OPR2 OPR3	Bp xb Bp xb x1 Bp xb x1 x2 Bp xb x1 x2 x3		
CLR X CLR Y	0 ⇒ X Clear index register X 0 ⇒ Y Clear index register Y	INH INH	9A 9B		
CLV	0 ⇒ V *Translates to* ANDCC #$FD	IMM1	CE FD	– – – –	– – 0 –
CMP *Di*,#*oprimmsz*	(D_i) – (M) Compare D_i with Memory Memory operand M is the same size as D_i	IMM1 IMM2 IMM4	Ep i1 Ep i2 i1 Ep i4 i3 i2 i1	– – – –	Δ Δ Δ Δ
CMP *Di*,*oprmemreg*	M can be a memory operand or another register D_j D_i determines the size of the operation If D_j smaller than D_i, zero-extend D_j If D_j larger than D_i, truncate D_j	OPR OPR1 OPR2 OPR3	Fn xb Fn xb x1 Fn xb x2 x1 Fn xb x3 x2 x1		
CMP *xy*,#*opr24i*	(xy) – (M:M+1:M+2)	IMM3	Ep i3 i2 i1		
CMP *xy*,*oprmemreg*	Compare X or Y with Memory	OPR OPR1 OPR2 OPR3	Fp xb Fp xb x1 Fp xb x2 x1 Fp xb x3 x2 x1		
CMP S,#*opr24i*	(SP) – (M:M+1:M+2)	IMM3	1B 04 i3 i2 i1		
CMP S,*oprmemreg*	Compare stack pointer SP with Memory	OPR OPR1 OPR2 OPR3	1B 02 xb 1B 02 xb x1 1B 02 xb x2 x1 1B 02 xb x3 x2 x1		
CMP X,Y	(X) – (Y) Compare X with Y	INH	FC		
COM.*bwl oprmemreg*	~(M) ⇒ M equivalent to $F..F – (M) ⇒ M ~($D_i$) ⇒ D_i equivalent to $F..F – ($D_i$) ⇒ D_i 1's Complement Memory Location or D_i Memory operand M can be 1, 2, or 4 bytes use .B for D0,D1; .W for D2~D5; and .L for D6,D7	OPR OPR1 OPR2 OPR3	Cp xb Cp xb x1 Cp xb x2 x1 Cp xb x3 x2 x1	– – – –	Δ Δ 0 –
DB*cc Di*,*oprdest*	(D_i) – 1 ⇒ D_i Decrement and branch	REG-R7 REG-R15	0B lb rb 0B lb rb r1	– – – –	– – – –
DB*cc xy*,*oprdest*	(X) – 1 ⇒ X or (Y) – 1 ⇒ Y Decrement and branch	REG-R7 REG-R15	0B lb rb 0B lb rb r1		
DB*cc*.*bwpl oprmemreg*,*oprdest*	(M) – 1 ⇒ M Decrement D_i, X, Y, or memory operand M, and branch if condition cc is true. Memory operand M may be 8, 16, 24, or 32 bits long cc can be Not Equal-DBNE, Equal-DBEQ, Plus-DBPL, Minus-DBMI, Greater Than-DBGT, or Less Than or Equal-DBLE (encoded in postbyte lb) Branch offset is 7 or 15 bits	OPR-R7 OPR-R15 OPR1-R7 OPR1-R15 OPR2-R7 OPR2-R15 OPR3-R7 OPR3-R15	0B lb xb rb 0B lb xb rb r1 0B lb xb x1 rb 0B lb xb x1 rb r1 0B lb xb x2 x1 rb 0B lb xb x2 x1 rb r1 0B lb xb x3 x2 x1 rb 0B lb xb x3 x2 x1 rb r1		
DEC *Di*	(D_i) – 1 ⇒ D_i Decrement data register D_i	INH	4n	– – – –	Δ Δ Δ –
DEC.*bwl oprmemreg*	(M) – 1 ⇒ M Decrement Memory Memory operand M can be 1, 2, or 4 bytes	OPR OPR1 OPR2 OPR3	Ap xb Ap xb x1 Ap xb x2 x1 Ap xb x3 x2 x1		

Table A-1. S12Z Instruction Set Summary (Sheet 6 of 17)

Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	Condition Codes N Z V C
DIVS *Dd,Dj,Dk*	(D_j) $(D_k) \Rightarrow D_d$ signed Divide result is always a register D_d	REG-REG	1B 3n mb	– – – –	Δ Δ Δ Δ
DIVS *Dd,Dj,#opr8i* DIVS *Dd,Dj,#opr16i* DIVS *Dd,Dj,#opr32i*	(D_j) $(M) \Rightarrow D_d$ Memory operand M can be 8, 16, or 32 bits	REG-IMM1 REG-IMM2 REG-IMM4	1B 3n mb i1 1B 3n mb i2 i1 1B 3n mb i4 i3 i2 i1		
DIVS.*bwl Dd,Dj,oprmemreg*	(D_j) $(M) \Rightarrow D_d$ Memory operand M can be 8, 16, or 32 bits	REG-OPR REG-OPR1 REG-OPR2 REG-OPR3	1B 3n mb xb 1B 3n mb xb x1 1B 3n mb xb x2 x1 1B 3n mb xb x3 x2 x1		
DIVS.*bwlbwpl Dd,oprmemreg,oprmemreg*	(M1) (M2) $\Rightarrow D_d$ Memory operands M1 and M2 can be 8, 16, 24, or 32 bits M1 and M2 can be different sizes Memory operand M1 appears first in the object code size of memory operands is encoded in the postbyte mb although memory operands could be registers, the register and register/memory versions are more efficient memory/register is possible by using the second memory postbyte to specify a register as memory	OPR-OPR OPR-OPR1 OPR-OPR2 OPR-OPR3 OPR1-OPR OPR1-OPR1 OPR1-OPR2 OPR1-OPR3 OPR2-OPR OPR2-OPR1 OPR2-OPR2 OPR2-OPR3 OPR3-OPR OPR3-OPR1 OPR3-OPR2 OPR3-OPR3	1B 3n mb xb xb 1B 3n mb xb xb x1 1B 3n mb xb xb x2 x1 1B 3n mb xb xb x3 x2 x1 1B 3n mb xb x1 xb 1B 3n mb xb x1 xb x1 1B 3n mb xb x1 xb x2 x1 1B 3n mb xb x1 xb x3 x2 x1 1B 3n mb xb x2 x1 xb 1B 3n mb xb x2 x1 xb x1 1B 3n mb xb x2 x1 xb x2 x1 1B 3n mb xb x2 x1 xb x3 x2 x1 1B 3n mb xb x3 x2 x1 xb 1B 3n mb xb x3 x2 x1 xb x1 1B 3n mb xb x3 x2 x1 xb x2 x1 1B 3n mb xb x3 x2 x1 xb x3 x2 x1		
DIVU *Dd,Dj,Dk*	(D_j) $(D_k) \Rightarrow D_d$ unsigned Divide result is always a register D_d	REG-REG	1B 3n mb	– – – –	Δ Δ Δ Δ
DIVU *Dd,Dj,#opr8i* DIVU *Dd,Dj,#opr16i* DIVU *Dd,Dj,#opr32i*	(D_j) $(M) \Rightarrow D_d$ Memory operand M can be 8, 16, or 32 bits	REG-IMM1 REG-IMM2 REG-IMM4	1B 3n mb i1 1B 3n mb i2 i1 1B 3n mb i4 i3 i2 i1		
DIVU.*bwl Dd,Dj,oprmemreg*	(D_j) $(M) \Rightarrow D_d$ Memory operand M can be 8, 16, or 32 bits	REG-OPR REG-OPR1 REG-OPR2 REG-OPR3	1B 3n mb xb 1B 3n mb xb x1 1B 3n mb xb x2 x1 1B 3n mb xb x3 x2 x1		
DIVU.*bwlbwpl Dd,oprmemreg,oprmemreg*	(M1) (M2) $\Rightarrow D_d$ Memory operands M1 and M2 can be 8, 16, 24, or 32 bits M1 and M2 can be different sizes Memory operand M1 appears first in the object code size of memory operands is encoded in the postbyte mb although memory operands could be registers, the register and register/memory versions are more efficient memory/register is possible by using the second memory postbyte to specify a register as memory	OPR-OPR OPR-OPR1 OPR-OPR2 OPR-OPR3 OPR1-OPR OPR1-OPR1 OPR1-OPR2 OPR1-OPR3 OPR2-OPR OPR2-OPR1 OPR2-OPR2 OPR2-OPR3 OPR3-OPR OPR3-OPR1 OPR3-OPR2 OPR3-OPR3	1B 3n mb xb xb 1B 3n mb xb xb x1 1B 3n mb xb xb x2 x1 1B 3n mb xb xb x3 x2 x1 1B 3n mb xb x1 xb 1B 3n mb xb x1 xb x1 1B 3n mb xb x1 xb x2 x1 1B 3n mb xb x1 xb x3 x2 x1 1B 3n mb xb x2 x1 xb 1B 3n mb xb x2 x1 xb x1 1B 3n mb xb x2 x1 xb x2 x1 1B 3n mb xb x2 x1 xb x3 x2 x1 1B 3n mb xb x3 x2 x1 xb 1B 3n mb xb x3 x2 x1 xb x1 1B 3n mb xb x3 x2 x1 xb x2 x1 1B 3n mb xb x3 x2 x1 xb x3 x2 x1		
EOR *Di,#oprimmsz*	(D_i) ^ (M) $\Rightarrow D_i$ Exclusive OR D_i with Memory Memory operand M is the same size as D_i	IMM1 IMM2 IMM4	1B 7p i1 1B 7p i2 i1 1B 7p i4 i3 i2 i1	– – – –	Δ Δ 0 –
EOR *Di,oprmemreg*	M can be a memory operand or another register D_j if D_j wider, OR D_i with low portion of D_j if D_j narrower, OR D_i with zero-extended D_j	OPR OPR1 OPR2 OPR3	1B 8q xb 1B 8q xb x1 1B 8q xb x2 x1 1B 8q xb x3 x2 x1		
EXG *cpureg,cpureg*	(r1) ⇔ (r2) Exchange contents of CPU Registers D0~D7, X, Y, S, CCH, CCL, or CCW if same size, direct exchange if 1st smaller than 2nd, sign extend 1st to 2nd if 1st larger than 2nd, sign-extend small to big and truncate big to small. CCW,CCH and CCW,CCL act as NOP.	REG-REG	AE eb	– – – –	– – – –

Table A-1. S12Z Instruction Set Summary (Sheet 7 of 17)

Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	Condition Codes N Z V C
INC *Di*	$(D_i) + 1 \Rightarrow D_i$ Increment data register D_i	INH	3n	– – – –	Δ Δ Δ –
INC.*bwl oprmemreg*	(M) + 1 ⇒ M Increment Memory Memory operand M can be 1, 2, or 4 bytes	OPR OPR1 OPR2 OPR3	9p xb 9p xb x1 9p xb x2 x1 9p xb x3 x2 x1		
JMP *opr24a*	Effective Address ⇒ PC Jump (unconditional)	EXT24	BA a3 a2 a1	– – – –	– – – –
JMP *oprmemreg*		OPR OPR1 OPR2 OPR3	AA xb AA xb x1 AA xb x2 x1 AA xb x3 x2 x1		
JSR *opr24a*	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP):M(SP+1):M(SP+2); Subroutine Address ⇒ PC Jump to Subroutine	EXT24	BB a3 a2 a1	– – – –	– – – –
JSR *oprmemreg*		OPR OPR1 OPR2 OPR3	AB xb AB xb x1 AB xb x2 x1 AB xb x3 x2 x1		
LD *Di*,#*oprimmsz*	(M) ⇒ D_i Load D_i Memory operand M is the same size as D_i M can be a memory operand or another register D_j	IMM1 IMM2 IMM4	9p i1 9p i2 i1 9p i4 i3 i2 i1	– – – –	Δ Δ 0 –
LD *Di*,*opr24a*		EXT24	Bn a3 a2 a1		
LD *Di*,*oprmemreg*		OPR OPR1 OPR2 OPR3	An xb An xb x1 An xb x2 x1 An xb x3 x2 x1		
LD *xy*,#*opr18i* LD *xy*,#*opr24i*	(M:M+1:M+2) ⇒ X or Y Load index register X or Y M can be a memory operand or a register D_j	IMM2 IMM3	op i2 i1 9p i3 i2 i1		
LD *xy*,*opr24a*		EXT3	Bp a3 a2 a1		
LD *xy*,*oprmemreg*		OPR OPR1 OPR2 OPR3	Ap xb Ap xb x1 Ap xb x2 x1 Ap xb x3 x2 x1		
LD S,#*opr24i*	(M:M+1:M+2) ⇒ SP Load Stack Pointer SP M can be a memory operand or a register D_j	EXT24	1B 03 i3 i2 i1		
LD S,*oprmemreg*		OPR OPR1 OPR2 OPR3	1B 00 xb 1B 00 xb x1 1B 00 xb x2 x1 1B 00 xb x3 x2 x1		
LEA D6,*oprmemreg*	Effective Address ⇒ D6	OPR OPR1 OPR2 OPR3	06 xb 06 xb x1 06 xb x2 x1 06 xb x3 x2 x1	– – – –	– – – –
LEA D7,*oprmemreg*	Effective Address ⇒ D7	OPR OPR1 OPR2 OPR3	07 xb 07 xb x1 07 xb x2 x1 07 xb x3 x2 x1		
LEA S,*oprmemreg*	Effective Address ⇒ SP	OPR OPR1 OPR2 OPR3	0A xb 0A xb x1 0A xb x2 x1 0A xb x3 x2 x1		
LEA X,*oprmemreg*	Effective Address ⇒ X	OPR OPR1 OPR2 OPR3	08 xb 08 xb x1 08 xb x2 x1 08 xb x3 x2 x1		
LEA Y,*oprmemreg*	Effective Address ⇒ Y	OPR OPR1 OPR2 OPR3	09 xb 09 xb x1 09 xb x2 x1 09 xb x3 x2 x1		
LEA S,(*opr8i*,S)	(SP) + sign-extend (M) ⇒ SP 8-bit immediate signed value	IMM1	1A i1		
LEA X,(*opr8i*,X)	(X) + sign-extend (M) ⇒ X 8-bit immediate signed value	IMM1	18 i1		
LEA Y,(*opr8i*,Y)	(Y) + sign-extend (M) ⇒ Y 8-bit immediate signed value	IMM1	19 i1		

Table A-1. S12Z Instruction Set Summary (Sheet 8 of 17)

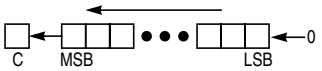
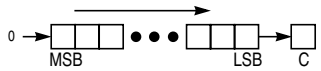
Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	Condition Codes N Z V C
LSL *Dd*,*Ds*,*Dn*		REG-REG	1n sb xb	– – – –	Δ Δ Δ Δ
LSL *Dd*,*Ds*,#*opr1i*		REG-IMM	1n sb		
LSL *Dd*,*Ds*,#*opr5i*		REG-IMM	1n sb xb		
LSL.*bwpl Dd*,*oprmemreg*,#*opr1i*	Logical Shift Left D_s or memory, 0 to n positions where n+1 is the number of bits in the operand The result is saved in register D_d (encoded in the opcode). n is specified in postbyte sb, sb+xb, a byte-sized memory operand, or register D_n If the destination is wider than the source, zero-extend to the width of the destination before shifting. If the destination is narrower than the source, shift and then truncate to the width of the destination. In the case of two OPR operands, the parameter n operand is the last operand in the source form and the object code.	OPR-IMM	1n sb xb		
		OPR1-IMM	1n sb xb x1		
		OPR2-IMM	1n sb xb x2 x1		
		OPR3-IMM	1n sb xb x3 x2 x1		
LSL.*bwpl Dd*,*oprmemreg*,#*opr5i*		OPR-IMM	1n sb xb xb		
		OPR1-IMM	1n sb xb x1 xb		
		OPR2-IMM	1n sb xb x2 x1 xb		
		OPR3-IMM	1n sb xb x3 x2 x1 x1 xb		
LSL.*bwpl Dd*,*oprmemreg*,*oprmemreg*		OPR-OPR	1n sb xb xb		
		OPR-OPR1	1n sb xb xb x1		
		OPR-OPR2	1n sb xb xb x2 x1		
		OPR-OPR3	1n sb xb xb x3 x2 x1		
		OPR1-OPR	1n sb xb x1 xb		
		OPR1-OPR1	1n sb xb x1 xb x1		
		OPR1-OPR2	1n sb xb x1 xb x2 x1		
		OPR1-OPR3	1n sb xb x1 xb x3 x2 x1		
		OPR2-OPR	1n sb xb x2 x1 xb		
		OPR2-OPR1	1n sb xb x2 x1 xb x1		
		OPR2-OPR2	1n sb xb x2 x1 xb x2 x1		
		OPR2-OPR3	1n sb xb x2 x1 xb x3 x2 x1		
		OPR3-OPR	1n sb xb x3 x2 x1 xb		
		OPR3-OPR1	1n sb xb x3 x2 x1 xb x1		
		OPR3-OPR2	1n sb xb x3 x2 x1 xb x2 x1		
		OPR3-OPR3	1n sb xb x3 x2 x1 xb x3 x2 x1		
LSL.*bwpl oprmemreg*,#*opr1i*	Shift memory location by 1 or 2 position. Source and destination are the same memory operand	OPR-IMM	1n sb xb		
		OPR1-IMM	1n sb xb x1		
		OPR2-IMM	1n sb xb x2 x1		
		OPR3-IMM	1n sb xb x3 x2 x1		
LSR *Dd*,*Ds*,*Dn*		REG-REG	1n sb xb	– – – –	0 Δ Δ Δ
LSR *Dd*,*Ds*,#*opr1i*		REG-IMM	1n sb		
LSR *Dd*,*Ds*,#*opr5i*		REG-IMM	1n sb xb		
LSR.*bwpl Dd*,*oprmemreg*,#*opr1i*	Logical Shift Right D_s or memory, 0 to n positions where n+1 is the number of bits in the operand The result is saved in register D_d (encoded in the opcode). n is specified in postbyte sb, sb+xb, a byte-sized memory operand, or register D_n If the destination is wider than the source, zero-extend to the width of the destination before shifting. If the destination is narrower than the source, shift and then truncate to the width of the destination. In the case of two OPR operands, the parameter n operand is the last operand in the source form and the object code.	OPR-IMM	1n sb xb		
		OPR1-IMM	1n sb xb x1		
		OPR2-IMM	1n sb xb x2 x1		
		OPR3-IMM	1n sb xb x3 x2 x1		
LSR.*bwpl Dd*,*oprmemreg*,#*opr5i*		OPR-IMM	1n sb xb xb		
		OPR1-IMM	1n sb xb x1 xb		
		OPR2-IMM	1n sb xb x2 x1 xb		
		OPR3-IMM	1n sb xb x3 x2 x1 x1 xb		
LSR.*bwpl Dd*,*oprmemreg*,*oprmemreg*		OPR-OPR	1n sb xb xb		
		OPR-OPR1	1n sb xb xb x1		
		OPR-OPR2	1n sb xb xb x2 x1		
		OPR-OPR3	1n sb xb xb x3 x2 x1		
		OPR1-OPR	1n sb xb x1 xb		
		OPR1-OPR1	1n sb xb x1 xb x1		
		OPR1-OPR2	1n sb xb x1 xb x2 x1		
		OPR1-OPR3	1n sb xb x1 xb x3 x2 x1		
		OPR2-OPR	1n sb xb x2 x1 xb		
		OPR2-OPR1	1n sb xb x2 x1 xb x1		
		OPR2-OPR2	1n sb xb x2 x1 xb x2 x1		
		OPR2-OPR3	1n sb xb x2 x1 xb x3 x2 x1		
		OPR3-OPR	1n sb xb x3 x2 x1 xb		
		OPR3-OPR1	1n sb xb x3 x2 x1 xb x1		
		OPR3-OPR2	1n sb xb x3 x2 x1 xb x2 x1		
		OPR3-OPR3	1n sb xb x3 x2 x1 xb x3 x2 x1		
LSR.*bwpl oprmemreg*,#*opr1i*	Shift memory location by 1 or 2 position. Source and destination are the same memory operand	OPR-IMM	1n sb xb		
		OPR1-IMM	1n sb xb x1		
		OPR2-IMM	1n sb xb x2 x1		
		OPR3-IMM	1n sb xb x3 x2 x1		

Table A-1. S12Z Instruction Set Summary (Sheet 9 of 17)

Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	Condition Codes N Z V C
MACS *Dd*,*Dj*,*Dk*	(D_j) $(D_k) + D_d \Rightarrow D_d$ signed multiply and accumulate result is always a register D_d	REG-REG	1B 4q mb	– – – –	Δ Δ Δ Δ
MACS *Dd*,*Dj*,#*opr8i*	(D_j) $(M) + D_d \Rightarrow D_d$	REG-IMM1	1B 4q mb i1		
MACS *Dd*,*Dj*,#*opr16i*	Memory operand M can be 8, 16, or 32 bits	REG-IMM2	1B 4q mb i2 i1		
MACS *Dd*,*Dj*,#*opr32i*		REG-IMM4	1B 4q mb i4 i3 i2 i1		
MACS.*bwl Dd*,*Dj*,*oprmemreg*	(D_j) $(M) + D_d \Rightarrow D_d$	REG-OPR	1B 4q mb xb		
	Memory operand M can be 8, 16, or 32 bits	REG-OPR1	1B 4q mb xb x1		
		REG-OPR2	1B 4q mb xb x2 x1		
		REG-OPR3	1B 4q mb xb x3 x2 x1		
MACS.*bwplbwpl Dd*,*oprmemreg*,*oprmemreg*	(M1) (M2) + $D_d \Rightarrow D_d$	OPR-OPR	1B 4q mb xb xb		
	Memory operands M1 and M2 can be 8, 16, 24, or 32 bits	OPR-OPR1	1B 4q mb xb xb x1		
	M1 and M2 can be different sizes	OPR-OPR2	1B 4q mb xb xb x2 x1		
	Memory operand M1 appears first in the object code	OPR-OPR3	1B 4q mb xb xb x3 x2 x1		
	size of memory operands is encoded in the postbyte mb	OPR1-OPR	1B 4q mb xb x1 xb		
	although memory operands could be registers, the	OPR1-OPR1	1B 4q mb xb x1 xb x1		
	register and register-memory versions are more efficient	OPR1-OPR2	1B 4q mb xb x1 xb x2 x1		
		OPR1-OPR3	1B 4q mb xb x1 xb x3 x2 x1		
		OPR2-OPR	1B 4q mb xb x2 x1 xb		
		OPR2-OPR1	1B 4q mb xb x2 x1 xb x1		
		OPR2-OPR2	1B 4q mb xb x2 x1 xb x2 x1		
		OPR2-OPR3	1B 4q mb xb x2 x1 xb x3 x2 x1		
		OPR3-OPR	1B 4q mb xb x3 x2 x1 xb		
		OPR3-OPR1	1B 4q mb xb x3 x2 x1 xb x1		
		OPR3-OPR2	1B 4q mb xb x3 x2 x1 xb x2 x1		
		OPR3-OPR3	1B 4q mb xb x3 x2 x1 xb x3 x2 x1		
MACU *Dd*,*Dj*,*Dk*	(D_j) $(D_k) + D_d \Rightarrow D_d$ unsigned multiply and accumulate result is always a register D_d	REG-REG	1B 4q mb	– – – –	Δ Δ Δ Δ
MACU *Dd*,*Dj*,#*opr8i*	(D_j) $(M) + D_d \Rightarrow D_d$	REG-IMM1	1B 4q mb i1		
MACU *Dd*,*Dj*,#*opr16i*	Memory operand M can be 8, 16, or 32 bits	REG-IMM2	1B 4q mb i2 i1		
MACU *Dd*,*Dj*,#*opr32i*		REG-IMM4	1B 4q mb i4 i3 i2 i1		
MACU.*bwl Dd*,*Dj*,*oprmemreg*	(D_j) $(M) + D_d \Rightarrow D_d$	REG-OPR	1B 4q mb xb		
	Memory operand M can be 8, 16, or 32 bits	REG-OPR1	1B 4q mb xb x1		
		REG-OPR2	1B 4q mb xb x2 x1		
		REG-OPR3	1B 4q mb xb x3 x2 x1		
MACU.*bwplbwpl Dd*,*oprmemreg*,*oprmemreg*	(M1) (M2) + $D_d \Rightarrow D_d$	OPR-OPR	1B 4q mb xb xb		
	Memory operands M1 and M2 can be 8, 16, 24, or 32 bits	OPR-OPR1	1B 4q mb xb xb x1		
	M1 and M2 can be different sizes	OPR-OPR2	1B 4q mb xb xb x2 x1		
	Memory operand M1 appears first in the object code	OPR-OPR3	1B 4q mb xb xb x3 x2 x1		
	size of memory operands is encoded in the postbyte mb	OPR1-OPR	1B 4q mb xb x1 xb		
	although memory operands could be registers, the	OPR1-OPR1	1B 4q mb xb x1 xb x1		
	register and register-memory versions are more efficient	OPR1-OPR2	1B 4q mb xb x1 xb x2 x1		
		OPR1-OPR3	1B 4q mb xb x1 xb x3 x2 x1		
		OPR2-OPR	1B 4q mb xb x2 x1 xb		
		OPR2-OPR1	1B 4q mb xb x2 x1 xb x1		
		OPR2-OPR2	1B 4q mb xb x2 x1 xb x2 x1		
		OPR2-OPR3	1B 4q mb xb x2 x1 xb x3 x2 x1		
		OPR3-OPR	1B 4q mb xb x3 x2 x1 xb		
		OPR3-OPR1	1B 4q mb xb x3 x2 x1 xb x1		
		OPR3-OPR2	1B 4q mb xb x3 x2 x1 xb x2 x1		
		OPR3-OPR3	1B 4q mb xb x3 x2 x1 xb x3 x2 x1		
MAXS *Di*,*oprmemreg*	$MAX((D_i), (M)) \Rightarrow D_i$	OPR	1B 2q xb	– – – –	Δ Δ Δ Δ
	MAXimum of two signed operands replaces D_i	OPR1	1B 2q xb x1		
	Memory operand M is the same size as D_i	OPR2	1B 2q xb x2 x1		
		OPR3	1B 2q xb x3 x2 x1		
MAXU *Di*,*oprmemreg*	$MAX((D_i), (M)) \Rightarrow D_i$	OPR	1B 1q xb	– – – –	Δ Δ Δ Δ
	MAXimum of two unsigned operands replaces D_i	OPR1	1B 1q xb x1		
	Memory operand M is the same size as D_i	OPR2	1B 1q xb x2 x1		
		OPR3	1B 1q xb x3 x2 x1		
MINS *Di*,*oprmemreg*	$MIN((D_i), (M)) \Rightarrow D_i$	OPR	1B 2n xb	– – – –	Δ Δ Δ Δ
	MINimum of two signed operands replaces D_i	OPR1	1B 2n xb x1		
	Memory operand M is the same size as D_i	OPR2	1B 2n xb x2 x1		
		OPR3	1B 2n xb x3 x2 x1		

Table A-1. S12Z Instruction Set Summary (Sheet 10 of 17)

Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	Condition Codes N Z V C
MINU *Di*,*oprmemreg*	MIN((D_i), (M)) ⇒ D_i MINimum of two unsigned operands replaces D_i Memory operand M is the same size as D_i	OPR OPR1 OPR2 OPR3	1B 1n xb 1B 1n xb x1 1B 1n xb x2 x1 1B 1n xb x3 x2 x1	– – – –	Δ Δ Δ Δ
MODS *Dd*,*Dj*,*Dk*	(D_j) (D_k); remainder ⇒ D_d signed modulo operation result is always a register D_d	REG-REG	1B 3q mb	– – – –	Δ Δ Δ Δ
MODS *Dd*,*Dj*,#*opr8i* MODS *Dd*,*Dj*,#*opr16i* MODS *Dd*,*Dj*,#*opr32i*	(D_j) (M); remainder⇒ D_d Memory operand M can be 8, 16, or 32 bits	REG-IMM1 REG-IMM2 REG-IMM4	1B 3q mb i1 1B 3q mb i2 i1 1B 3q mb i4 i3 i2 i1		
MODS.*bwl Dd*,*Dj*,*oprmemreg*	(D_j) (M); remainder⇒ D_d Memory operand M can be 8, 16, or 32 bits	REG-OPR REG-OPR1 REG-OPR2 REG-OPR3	1B 3q mb xb 1B 3q mb xb x1 1B 3q mb xb x2 x1 1B 3q mb xb x3 x2 x1		
MODS.*bwplbwpl Dd*,*oprmemreg*,*oprmemreg*	(M1) (M2); remainder⇒ D_d Memory operands M1 and M2 can be 8, 16, 24, or 32 bits M1 and M2 can be different sizes Memory operand M1 appears first in the object code size of memory operands is encoded in the postbyte mb although memory operands could be registers, the register and register/memory versions are more efficient memory/register is possible by using the second memory postbyte to specify a register as memory	OPR-OPR OPR-OPR1 OPR-OPR2 OPR-OPR3 OPR1-OPR OPR1-OPR1 OPR1-OPR2 OPR1-OPR3 OPR2-OPR OPR2-OPR1 OPR2-OPR2 OPR2-OPR3 OPR3-OPR OPR3-OPR1 OPR3-OPR2 OPR3-OPR3	1B 3q mb xb xb 1B 3q mb xb xb x1 1B 3q mb xb xb x2 x1 1B 3q mb xb xb x3 x2 x1 1B 3q mb xb x1 xb 1B 3q mb xb x1 xb x1 1B 3q mb xb x1 xb x2 x1 1B 3q mb xb x1 xb x3 x2 x1 1B 3q mb xb x2 x1 xb 1B 3q mb xb x2 x1 xb x1 1B 3q mb xb x2 x1 xb x2 x1 1B 3q mb xb x2 x1 xb x3 x2 x1 1B 3q mb xb x3 x2 x1 xb 1B 3q mb xb x3 x2 x1 xb x1 1B 3q mb xb x3 x2 x1 xb x2 x1 1B 3q mb xb x3 x2 x1 xb x3 x2 x1		
MODU *Dd*,*Dj*,*Dk*	(D_j) (D_k); remainder ⇒ D_d unsigned modulo operation result is always a register D_d	REG-REG	1B 3q mb	– – – –	Δ Δ Δ Δ
MODU *Dd*,*Dj*,#*opr8i* MODU *Dd*,*Dj*,#*opr16i* MODU *Dd*,*Dj*,#*opr32i*	(D_j) (M); remainder⇒ D_d Memory operand M can be 8, 16, or 32 bits	REG-IMM1 REG-IMM2 REG-IMM4	1B 3q mb i1 1B 3q mb i2 i1 1B 3q mb i4 i3 i2 i1		
MODU.*bwl Dd*,*Dj*,*oprmemreg*	(D_j) (M); remainder⇒ D_d Memory operand M can be 8, 16, or 32 bits	REG-OPR REG-OPR1 REG-OPR2 REG-OPR3	1B 3q mb xb 1B 3q mb xb x1 1B 3q mb xb x2 x1 1B 3q mb xb x3 x2 x1		
MODU.*bwplbwpl Dd*,*oprmemreg*,*oprmemreg*	(M1) (M2); remainder⇒ D_d Memory operands M1 and M2 can be 8, 16, 24, or 32 bits M1 and M2 can be different sizes Memory operand M1 appears first in the object code size of memory operands is encoded in the postbyte mb although memory operands could be registers, the register and register/memory versions are more efficient memory/register is possible by using the second memory postbyte to specify a register as memory	OPR-OPR OPR-OPR1 OPR-OPR2 OPR-OPR3 OPR1-OPR OPR1-OPR1 OPR1-OPR2 OPR1-OPR3 OPR2-OPR OPR2-OPR1 OPR2-OPR2 OPR2-OPR3 OPR3-OPR OPR3-OPR1 OPR3-OPR2 OPR3-OPR3	1B 3q mb xb xb 1B 3q mb xb xb x1 1B 3q mb xb xb x2 x1 1B 3q mb xb xb x3 x2 x1 1B 3q mb xb x1 xb 1B 3q mb xb x1 xb x1 1B 3q mb xb x1 xb x2 x1 1B 3q mb xb x1 xb x3 x2 x1 1B 3q mb xb x2 x1 xb 1B 3q mb xb x2 x1 xb x1 1B 3q mb xb x2 x1 xb x2 x1 1B 3q mb xb x2 x1 xb x3 x2 x1 1B 3q mb xb x3 x2 x1 xb 1B 3q mb xb x3 x2 x1 xb x1 1B 3q mb xb x3 x2 x1 xb x2 x1 1B 3q mb xb x3 x2 x1 xb x3 x2 x1		

Table A-1. S12Z Instruction Set Summary (Sheet 11 of 17)

Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	Condition Codes N Z V C
MOV.B #*opr8i*,*oprmemreg*	# ⇒ MD Move Immediate to Memory MD, 8-bit operands suggest load or transfer for register operands	IMM1-OPR IMM1-OPR1 IMM1-OPR2 IMM1-OPR3	0C i1 xb 0C i1 xb x1 0C i1 xb x2 x1 0C i1 xb x3 x2 x1	– – – –	– – – –
MOV.B *oprmemreg*,*oprmemreg*	(MS) ⇒ MD; Memory to memory, 8-bit operand Source (MS) reference is first in the object code suggest load or transfer for register to register moves If MS is a larger register, truncate before store to MD Unsigned widening is possible for register-register, register-memory, or memory-register If MD is a larger register, zero-extend MS and store to reg If MS uses short-immediate to specify -1, 1, 2, 3…14, 15; the short IMM value is sign-extended and stored to MD even if MD is a larger register than the move size	OPR-OPR OPR-OPR1 OPR-OPR2 OPR-OPR3 OPR1-OPR OPR1-OPR1 OPR1-OPR2 OPR1-OPR3 OPR2-OPR OPR2-OPR1 OPR2-OPR2 OPR2-OPR3 OPR3-OPR OPR3-OPR1 OPR3-OPR2 OPR3-OPR3	1C xb xb 1C xb xb x1 1C xb xb x2 x1 1C xb xb x3 x2 x1 1C xb x1 xb 1C xb x1 xb x1 1C xb x1 xb x2 x1 1C xb x1 xb x3 x2 x1 1C xb x2 x1 xb 1C xb x2 x1 xb x1 1C xb x2 x1 xb x2 x1 1C xb x2 x1 xb x3 x2 x1 1C xb x3 x2 x1 xb 1C xb x3 x2 x1 xb x1 1C xb x3 x2 x1 xb x2 x1 1C xb x3 x2 x1 xb x3 x2 x1		
MOV.L #*opr32i*,*oprmemreg*	# ⇒ MD Move Immediate to Memory MD, 32-bit operands suggest load or transfer for register operands	IMM4-OPR IMM4-OPR1 IMM4-OPR2 IMM4-OPR3	0F i4 i3 i2 i1 xb 0F i4 i3 i2 i1 xb x1 0F i4 i3 i2 i1 xb x2 x1 0F i4 i3 i2 i1 xb x3 x2 x1	– – – –	– – – –
MOV.L *oprmemreg*,*oprmemreg*	(MS) ⇒ MD; Memory to memory, 32-bit operand Source (MS) reference is first in the object code suggest load or transfer for register to register moves If MD is a smaller register, truncate MS and store to reg Unsigned widening is possible for register-register, register-memory, or memory-register If MS is a smaller register, zero-extend before store to MD If MS uses short-immediate to specify -1, 1, 2, 3…14, 15; the short IMM value is sign-extended and stored to MD	OPR-OPR OPR-OPR1 OPR-OPR2 OPR-OPR3 OPR1-OPR OPR1-OPR1 OPR1-OPR2 OPR1-OPR3 OPR2-OPR OPR2-OPR1 OPR2-OPR2 OPR2-OPR3 OPR3-OPR OPR3-OPR1 OPR3-OPR2 OPR3-OPR3	1F xb xb 1F xb xb x1 1F xb xb x2 x1 1F xb xb x3 x2 x1 1F xb x1 xb 1F xb x1 xb x1 1F xb x1 xb x2 x1 1F xb x1 xb x3 x2 x1 1F xb x2 x1 xb 1F xb x2 x1 xb x1 1F xb x2 x1 xb x2 x1 1F xb x2 x1 xb x3 x2 x1 1F xb x3 x2 x1 xb 1F xb x3 x2 x1 xb x1 1F xb x3 x2 x1 xb x2 x1 1F xb x3 x2 x1 xb x3 x2 x1		
MOV.P #*opr24i*,*oprmemreg*	# ⇒ MD Move Immediate to Memory MD, 24-bit operands suggest load or transfer for register operands	IMM3-OPR IMM3-OPR1 IMM3-OPR2 IMM3-OPR3	0E i3 i2 i1 xb 0E i3 i2 i1 xb x1 0E i3 i2 i1 xb x2 x1 0E i3 i2 i1 xb x3 x2 x1	– – – –	– – – –
MOV.P *oprmemreg*,*oprmemreg*	(MS) ⇒ MD; Memory to memory, 24-bit operand Source (MS) reference is first in the object code suggest load or transfer for register to register moves If MS is a larger register, truncate before store to MD If MD is a smaller register, truncate MS and store to reg Unsigned widening is possible for register-register, register-memory, or memory-register If MS is a smaller register, zero-extend before store to MD If MD is a larger register, zero-extend MS and store to reg If MS uses short-immediate to specify -1, 1, 2, 3…14, 15; the short IMM value is sign-extended and stored to MD even if MD is a larger register than the move size	OPR-OPR OPR-OPR1 OPR-OPR2 OPR-OPR3 OPR1-OPR OPR1-OPR1 OPR1-OPR2 OPR1-OPR3 OPR2-OPR OPR2-OPR1 OPR2-OPR2 OPR2-OPR3 OPR3-OPR OPR3-OPR1 OPR3-OPR2 OPR3-OPR3	1E xb xb 1E xb xb x1 1E xb xb x2 x1 1E xb xb x3 x2 x1 1E xb x1 xb 1E xb x1 xb x1 1E xb x1 xb x2 x1 1E xb x1 xb x3 x2 x1 1E xb x2 x1 xb 1E xb x2 x1 xb x1 1E xb x2 x1 xb x2 x1 1E xb x2 x1 xb x3 x2 x1 1E xb x3 x2 x1 xb 1E xb x3 x2 x1 xb x1 1E xb x3 x2 x1 xb x2 x1 1E xb x3 x2 x1 xb x3 x2 x1		

Table A-1. S12Z Instruction Set Summary (Sheet 12 of 17)

Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	Condition Codes N Z V C
MOV.W #*opr16i*,*oprmemreg*	# ⇒ MD Move Immediate to Memory MD, 16-bit operands suggest load or transfer for register operands	IMM2-OPR	0D i2 i1 xb	– – – –	– – – –
		IMM2-OPR1	0D i2 i1 xb x1		
		IMM2-OPR2	0D i2 i1 xb x2 x1		
		IMM2-OPR3	0D i2 i1 xb x3 x2 x1		
MOV.W *oprmemreg*,*oprmemreg*	(MS) ⇒ MD; Memory to memory, 16-bit operand Source (MS) reference is first in the object code suggest load or transfer for register to register moves If MS is a larger register, truncate before store to MD If MD is a smaller register, truncate MS and store to reg Unsigned widening is possible for register-register, register-memory, or memory-register If MS is a smaller register, zero-extend before store to MD If MD is a larger register, zero-extend MS and store to reg If MS uses short-immediate to specify -1, 1, 2, 3...14, 15; the short IMM value is sign-extended and stored to MD even if MD is a larger register than the move size	OPR-OPR	1D xb xb		
		OPR-OPR1	1D xb xb x1		
		OPR-OPR2	1D xb xb x2 x1		
		OPR-OPR3	1D xb xb x3 x2 x1		
		OPR1-OPR	1D xb x1 xb		
		OPR1-OPR1	1D xb x1 xb x1		
		OPR1-OPR2	1D xb x1 xb x2 x1		
		OPR1-OPR3	1D xb x1 xb x3 x2 x1		
		OPR2-OPR	1D xb x2 x1 xb		
		OPR2-OPR1	1D xb x2 x1 xb x1		
		OPR2-OPR2	1D xb x2 x1 xb x2 x1		
		OPR2-OPR3	1D xb x2 x1 xb x3 x2 x1		
		OPR3-OPR	1D xb x3 x2 x1 xb		
		OPR3-OPR1	1D xb x3 x2 x1 xb x1		
		OPR3-OPR2	1D xb x3 x2 x1 xb x2 x1		
		OPR3-OPR3	1D xb x3 x2 x1 xb x3 x2 x1		
MULS *Dd*,*Dj*,*Dk*	(D_j) (D_k) ⇒ D_d signed multiply result is always a register D_d	REG-REG	4q mb	– – – –	Δ Δ Δ 0

MULS *Dd*,*Dj*,#*opr8i*
MULS *Dd*,*Dj*,#*opr16i*
MULS *Dd*,*Dj*,#*opr32i*
	(D_j) (M) D_d
	Memory operand M can be 8, 16, or 32 bits
		REG-IMM1	4q mb i1		

		REG-IMM2	4q mb i2 i1		
		REG-IMM4	4q mb i4 i3 i2 i1		
MULS.*bwl Dd*,*Dj*,*oprmemreg*	(D_j) (M)⇒ D_d Memory operand M can be 8, 16, or 32 bits	REG-OPR	4q mb xb		
		REG-OPR1	4q mb xb x1		
		REG-OPR2	4q mb xb x2 x1		
		REG-OPR3	4q mb xb x3 x2 x1		
MULS.*bwplbwpl Dd*,*oprmemreg*,*oprmemreg*	(M1) (M2)⇒ D_d Memory operands M1 and M2 can be 8, 16, 24, or 32 bits M1 and M2 can be different sizes Memory operand M1 appears first in the object code size of memory operands is encoded in the postbyte mb although memory operands could be registers, the register and register-memory versions are more efficient	OPR-OPR	4q mb xb xb		
		OPR-OPR1	4q mb xb xb x1		
		OPR-OPR2	4q mb xb xb x2 x1		
		OPR-OPR3	4q mb xb xb x3 x2 x1		
		OPR1-OPR	4q mb xb x1 xb		
		OPR1-OPR1	4q mb xb x1 xb x1		
		OPR1-OPR2	4q mb xb x1 xb x2 x1		
		OPR1-OPR3	4q mb xb x1 xb x3 x2 x1		
		OPR2-OPR	4q mb xb x2 x1 xb		
		OPR2-OPR1	4q mb xb x2 x1 xb x1		
		OPR2-OPR2	4q mb xb x2 x1 xb x2 x1		
		OPR2-OPR3	4q mb xb x2 x1 xb x3 x2 x1		
		OPR3-OPR	4q mb xb x3 x2 x1 xb		
		OPR3-OPR1	4q mb xb x3 x2 x1 xb x1		
		OPR3-OPR2	4q mb xb x3 x2 x1 xb x2 x1		
		OPR3-OPR3	4q mb xb x3 x2 x1 xb x3 x2 x1		

Table A-1. S12Z Instruction Set Summary (Sheet 13 of 17)

Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	Condition Codes N Z V C
MULU *Dd,Dj,Dk*	(D_j) $(D_k) \Rightarrow D_d$ unsigned multiply result is always a register D_d	REG-REG	4q mb	– – – –	Δ Δ Δ 0

MULU *Dd,Dj,#opr8i*
MULU *Dd,Dj,#opr16i*
MULU *Dd,Dj,#opr32i*
	(D_j) $(M) \Rightarrow D_d$
	Memory operand M can be 8, 16, or 32 bits
		REG-IMM1
		REG-IMM2
		REG-IMM4
			4q mb i1
			4q mb i2 i1
			4q mb i4 i3 i2 i1		

MULU.*bwl Dd,Dj,oprmemreg*	(D_j) $(M) \Rightarrow D_d$ Memory operand M can be 8, 16, or 32 bits	REG-OPR REG-OPR1 REG-OPR2 REG-OPR3	4q mb xb 4q mb xb x1 4q mb xb x2 x1 4q mb xb x3 x2 x1		
MULU.*bwplbwpl Dd,oprmemreg,oprmemreg*	(M1) (M2) $\Rightarrow D_d$ Memory operands M1 and M2 can be 8, 16, 24, or 32 bits M1 and M2 can be different sizes Memory operand M1 appears first in the object code size of memory operands is encoded in the postbyte mb although memory operands could be registers, the register and register-memory versions are more efficient	OPR-OPR OPR-OPR1 OPR-OPR2 OPR-OPR3 OPR1-OPR OPR1-OPR1 OPR1-OPR2 OPR1-OPR3 OPR2-OPR OPR2-OPR1 OPR2-OPR2 OPR2-OPR3 OPR3-OPR OPR3-OPR1 OPR3-OPR2 OPR3-OPR3	4q mb xb xb 4q mb xb xb x1 4q mb xb xb x2 x1 4q mb xb xb x3 x2 x1 4q mb xb x1 xb 4q mb xb x1 xb x1 4q mb xb x1 xb x2 x1 4q mb xb x1 xb x3 x2 x1 4q mb xb x2 x1 xb 4q mb xb x2 x1 xb x1 4q mb xb x2 x1 xb x2 x1 4q mb xb x2 x1 xb x3 x2 x1 4q mb xb x3 x2 x1 xb 4q mb xb x3 x2 x1 xb x1 4q mb xb x3 x2 x1 xb x2 x1 4q mb xb x3 x2 x1 xb x3 x2 x1		
NEG.*bwl oprmemreg*	0 – (M) $\Rightarrow$ M equivalent to ~(M) + 1 $\Rightarrow$ M 0 – $(D_i) \Rightarrow D_i$ equivalent to ~(D_i) + 1 $\Rightarrow D_i$ Two's Complement Negate Memory operand M can be 1, 2, or 4 bytes use .B for D0,D1; .W for D2~D5; and .L for D6,D7	OPR OPR1 OPR2 OPR3	Dp xb Dp xb x1 Dp xb x2 x1 Dp xb x3 x2 x1	– – – –	Δ Δ Δ Δ
NOP	No operation	INH	01	– – – –	– – – –
OR *Di,#oprimmsz*	(D_i) | (M) $\Rightarrow D_i$ Bitwise OR D_i with Memory Memory operand M is the same size as D_i	IMM1 IMM2 IMM4	7p i1 7p i2 i1 7p i4 i3 i2 i1	– – – –	Δ Δ 0 –
OR *Di,oprmemreg*	M can be a memory operand or another register D_j if D_j wider, OR D_i with low portion of D_j if D_j narrower, OR D_i with zero-extended D_j	OPR OPR1 OPR2 OPR3	8q xb 8q xb x1 8q xb x2 x1 8q xb x3 x2 x1		
ORCC #*opr8i*	(CCL) | (M) $\Rightarrow$ CCL Bitwise OR CCL with Immediate Mask (S and I can only be changed in supervisor state)	IMM1	DE i1	⇑ – – ⇑ s s	⇑ ⇑ ⇑ ⇑
PSH *oprregs1* PSH *oprregs2* PSH ALL PSH ALL16b	(SP) – n $\Rightarrow$ SP; (regs) $\Rightarrow$ M(SP)~M(SP+n-1) Push specified CPU registers onto Stack register mask 1 - CCH, CCL, D0, D1, D2, D3 (D3 in LSB) register mask 2 - D4, D5, D6, D7, X, Y (Y in LSB) PSH ALL or PSH D4,D5,D6,D7,X,Y + PSH CCH,CCL,D0,D1,D2,D3 pushes all registers in the same order as SWI	INH	04 pb 04 00 04 40	– – – –	– – – –
PUL *oprregs1* PUL *oprregs2* PUL ALL PUL ALL16b	(M(SP)~M(SP+n-1)) $\Rightarrow$ regs; (SP) + n $\Rightarrow$ SP Pull specified CPU registers from Stack register mask 1 - CCH, CCL, D0, D1, D2, D3 (D3 in LSB) register mask 2 - D4, D5, D6, D7, X, Y (Y in LSB) PUL ALL or PUL CCH,CCL,D0,D1,D2,D3 + PUL D4,D5,D6,D7,X,Y pulls all registers in the same order as RTI	INH	04 pb 04 80 04 C0	– – – – pull CCL Δ ⇓ – Δ	– – – – Δ Δ Δ Δ

Table A-1. S12Z Instruction Set Summary (Sheet 14 of 17)

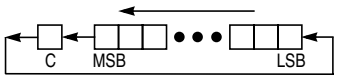
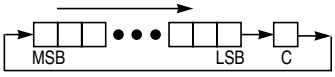
Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	Condition Codes N Z V C
QMULS *Dd,Dj,Dk*	(D_j) $(D_k) \Rightarrow D_d$ signed fractional multiply result is always a register D_d	REG-REG	1B Bn mb	– – – –	Δ Δ Δ 0
QMULS *Dd,Dj,#opr8i* QMULS *Dd,Dj,#opr16i* QMULS *Dd,Dj,#opr32i*	(D_j) (M) D_d Memory operand M can be 8, 16, or 32 bits	REG-IMM1 REG-IMM2 REG-IMM4	1B Bn mb i1 1B Bn mb i2 i1 1B Bn mb i4 i3 i2 i1		
QMULS.*bwl Dd,Dj,oprmemreg*	(D_j) (M)$\Rightarrow D_d$ Memory operand M can be 8, 16, or 32 bits	REG-OPR REG-OPR1 REG-OPR2 REG-OPR3	1B Bn mb xb 1B Bn mb xb x1 1B Bn mb xb x2 x1 1B Bn mb xb x3 x2 x1		
QMULS.*bwlbwpl* *Dd,oprmemreg,oprmemreg*	(M1) (M2)$\Rightarrow D_d$ Memory operands M1 and M2 can be 8, 16, 24, or 32 bits M1 and M2 can be different sizes Memory operand M1 appears first in the object code size of memory operands is encoded in the postbyte mb although memory operands could be registers, the register and register-memory versions are more efficient	OPR-OPR OPR-OPR1 OPR-OPR2 OPR-OPR3 OPR1-OPR OPR1-OPR1 OPR1-OPR2 OPR1-OPR3 OPR2-OPR OPR2-OPR1 OPR2-OPR2 OPR2-OPR3 OPR3-OPR OPR3-OPR1 OPR3-OPR2 OPR3-OPR3	1B Bn mb xb xb 1B Bn mb xb xb x1 1B Bn mb xb xb x2 x1 1B Bn mb xb xb x3 x2 x1 1B Bn mb xb x1 xb 1B Bn mb xb x1 xb x1 1B Bn mb xb x1 xb x2 x1 1B Bn mb xb x1 xb x3 x2 x1 1B Bn mb xb x2 x1 xb 1B Bn mb xb x2 x1 xb x1 1B Bn mb xb x2 x1 xb x2 x1 1B Bn mb xb x2 x1 xb x3 x2 x1 1B Bn mb xb x3 x2 x1 xb 1B Bn mb xb x3 x2 x1 xb x1 1B Bn mb xb x3 x2 x1 xb x2 x1 1B Bn mb xb x3 x2 x1 xb x3 x2 x1		
QMULU *Dd,Dj,Dk*	(D_j) $(D_k) \Rightarrow D_d$ unsigned fractional multiply result is always a register D_d	REG-REG	1B Bn mb	– – – –	Δ Δ 0 0
QMULU *Dd,Dj,#opr8i* QMULU *Dd,Dj,#opr16i* QMULU *Dd,Dj,#opr32i*	(D_j) (M)$\Rightarrow D_d$ Memory operand M can be 8, 16, or 32 bits	REG-IMM1 REG-IMM2 REG-IMM4	1B Bn mb i1 1B Bn mb i2 i1 1B Bn mb i4 i3 i2 i1		
QMULU.*bwl Dd,Dj,oprmemreg*	(D_j) (M)$\Rightarrow D_d$ Memory operand M can be 8, 16, or 32 bits	REG-OPR REG-OPR1 REG-OPR2 REG-OPR3	1B Bn mb xb 1B Bn mb xb x1 1B Bn mb xb x2 x1 1B Bn mb xb x3 x2 x1		
QMULU.*bwlbwpl* *Dd,oprmemreg,oprmemreg*	(M1) (M2)$\Rightarrow D_d$ Memory operands M1 and M2 can be 8, 16, 24, or 32 bits M1 and M2 can be different sizes Memory operand M1 appears first in the object code size of memory operands is encoded in the postbyte mb although memory operands could be registers, the register and register-memory versions are more efficient	OPR-OPR OPR-OPR1 OPR-OPR2 OPR-OPR3 OPR1-OPR OPR1-OPR1 OPR1-OPR2 OPR1-OPR3 OPR2-OPR OPR2-OPR1 OPR2-OPR2 OPR2-OPR3 OPR3-OPR OPR3-OPR1 OPR3-OPR2 OPR3-OPR3	1B Bn mb xb xb 1B Bn mb xb xb x1 1B Bn mb xb xb x2 x1 1B Bn mb xb xb x3 x2 x1 1B Bn mb xb x1 xb 1B Bn mb xb x1 xb x1 1B Bn mb xb x1 xb x2 x1 1B Bn mb xb x1 xb x3 x2 x1 1B Bn mb xb x2 x1 xb 1B Bn mb xb x2 x1 xb x1 1B Bn mb xb x2 x1 xb x2 x1 1B Bn mb xb x2 x1 xb x3 x2 x1 1B Bn mb xb x3 x2 x1 xb 1B Bn mb xb x3 x2 x1 xb x1 1B Bn mb xb x3 x2 x1 xb x2 x1 1B Bn mb xb x3 x2 x1 xb x3 x2 x1		
ROL.*bwpl oprmemreg*	 Rotate Left through Carry D_i or memory, 1 bit position	OPR OPR1 OPR2 OPR3	1n sb xb 1n sb xb x1 1n sb xb x2 x1 1n sb xb x3 x2 x1	– – – –	Δ Δ 0 Δ
ROR.*bwpl oprmemreg*	 Rotate Right through Carry D_i or memory, 1 bit position	OPR OPR1 OPR2 OPR3	1n sb xb 1n sb xb x1 1n sb xb x2 x1 1n sb xb x3 x2	– – – –	Δ Δ 0 Δ

Table A-1. S12Z Instruction Set Summary (Sheet 15 of 17)

Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	Condition Codes N Z V C
RTI	(M(SP)~M(SP+3)) ⇒ CCH:CCL, D0, D1; (SP)+4 ⇒ SP (M(SP)~M(SP+3)) ⇒ D2H:D2L, D3H:D3L; (SP)+4 ⇒ SP (M(SP)~M(SP+3)) ⇒ D4H:D4L, D5H:D5L; (SP)+4 ⇒ SP (M(SP)~M(SP+3)) ⇒ D6H:D6MH:D6ML:D6L; (SP)+4 ⇒ SP (M(SP)~M(SP+3)) ⇒ D7H:D7MH:D7ML:D7L; (SP)+4 ⇒ SP (M(SP)~M(SP+2)) ⇒ XH:XM:XL; (SP)+3 ⇒ SP (M(SP)~M(SP+2)) ⇒ YH:YM:YL; (SP)+3 ⇒ SP (M(SP)~M(SP+2)) ⇒ RTNH:RTNM:RTNL; (SP)+3 ⇒ SP Return from Interrupt (S, X, and I can only be changed in supervisor state)	INH	1B 90	Δ ⇓ – Δ s s s	Δ Δ Δ Δ
RTS	(M(SP):M(SP+1):M(SP+2)) ⇒ PCH:PCM:PCL; (SP) + 3 ⇒ SP Return from Subroutine	INH	05	– – – –	– – – –
SAT *Di*	saturate(D_i) ⇒ D_i Replace D_i with the Saturated Value of D_i	INH	1B An	– – – –	Δ Δ Δ –
SBC *Di*,#*oprimmsz*	(D_i) – (M) – C ⇒ D_i Subtract with Carry from D_i Memory operand M is the same size as D_i	IMM1 IMM2 IMM4	1B 7p i1 1B 7p i2 i1 1B 7p i4 i3 i2 i1	– – – –	Δ Δ Δ Δ
SBC *Di*,*oprmemreg*	M can be a memory operand or another register D_j	OPR OPR1 OPR2 OPR3	1B 8n xb 1B 8n xb x1 1B 8n xb x2 x1 1B 8n xb x3 x2 x1		
SEC	1 ⇒ C Translates to ORCC #$01	IMM	DE 01	– – – –	– – – 1
SEI	1 ⇒ I; (inhibit I interrupts) Translates to ORCC #$10 (I can only be changed in supervisor state)	IMM	DE 10	– – – 1 s	– – – –
SEV	1 ⇒ V Translates to ORCC #$02	IMM	DE 02	– – – –	– – 1 –
SEX *cpureg*,*cpureg*	Sign-Extend (r1) ⇒ (r2) D0~D7, X, Y, S, CCH, CCL, or CCW same as exchange EXG except r1 is smaller than r2 (If r2 = CCL or CCW, CCR bits may be written)	REG-REG	AE eb	– – – – c c c	– – – – c c c c
SPARE	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; YH:YM:YL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; XH:XM:XL ⇒ M(SP)~M(SP+2); (SP) – 4 ⇒ SP; D7H:D7MH:D7ML:D7L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D6H:D6MH:D6ML:D6L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D4H:D4L, D5H:D5L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D2H:D2L, D3H:D3L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; CCH, CCL, D0, D1 ⇒ M(SP)~M(SP+3); 1 ⇒ I; (pg1 TRAP Vector) ⇒ PC Unimplemented pg1 Opcode Trap Interrupt (I bit can only be changed in supervisor state)	INH	EF	– – – 1 s	– – – –
ST *Di*,*opr24a*	(D_i) ⇒ M Store Di to Memory Memory operand M is the same size as D_i M can be a memory operand or another register D_j	EXT24	Dn i3 i2 i1	– – – –	Δ Δ 0 –
ST *Di*,*oprmemreg*		OPR OPR1 OPR2 OPR3	Cn xb Cn xb x1 Cn xb x2 x1 Cn xb x3 x2 x1		
ST *xy*,*opr24a*	(X) ⇒ (M:M+1:M+2) or (Y) ⇒ (M:M+1:M+2) Store index register X or Y to Memroy M can be a memory operand or a register D_j	EXT24	Dp i3 i2 i1		
ST *xy*,*oprmemreg*		OPR OPR1 OPR2 OPR3	Cp xb Cp xb x1 Cp xb x2 x1 Cp xb x3 x2 x1		
ST S,*oprmemreg*	(SP) ⇒ (M:M+1:M+2) Store Stack Pointer SP to Memory M can be a memory operand or a register D_j	OPR OPR1 OPR2 OPR3	1B 01 xb 1B 01 xb x1 1B 01 xb x2 x1 1B 01 xb x3 x2 x1		

Table A-1. S12Z Instruction Set Summary (Sheet 16 of 17)

Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	Condition Codes N Z V C
STOP	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; YH:YM:YL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; XH:XM:XL ⇒ M(SP)~M(SP+2); (SP) – 4 ⇒ SP; D7H:D7MH:D7ML:D7L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D6H:D6MH:D6ML:D6L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D4H:D4L, D5H:D5L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D2H:D2L, D3H:D3L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; CCH, CCL, D0, D1 ⇒ M(SP)~M(SP+3); STOP All Clocks Registers stacked to allow quicker recovery by interrupt. If S control bit = 1, the STOP instruction is disabled and acts like a NOP.	INH	1B 05	– – – –	– – – –
SUB *Di*,#*oprimmsz*	(D_i) – (M) ⇒ D_i Subtract without Carry to D_i Memory operand M is the same size as D_i M can be a memory operand or another register D_j	IMM1 IMM2 IMM4	7p i1 7p i2 i1 7p i4 i3 i2 i1	– – – –	Δ Δ Δ Δ
SUB *Di*,*oprmemreg*		OPR OPR1 OPR2 OPR3	8n xb 8n xb x1 8n xb x2 x1 8n xb x3 x2 x1		
SUB D6,X,Y	(X) – (Y) ⇒ D6 Subtract without carry	INH	FD		
SUB D6,Y,X	(Y) – (X) ⇒ D6 Subtract without carry	INH	FE		
SWI	(SP) – 4 ⇒ SP; YL, RTNH:RTNM:RTNL ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; XM:XL, YH:YM ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D2L, D1, D0, XH ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D4L, D3H:D3L, D2H ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D6L, D5H:D5L, D4H ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D7L, D6H:D6MH:D6ML ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; CCL, D7H:D7MH:D7ML ⇒ M(SP)~M(SP+3); (SP) – 1 ⇒ SP; (CCH) ⇒ M(SP); 1 ⇒ I; (SWI Vector) ⇒ PC Software Interrupt (I bit can only be changed in supervisor state)	INH	FF	– – – 1 s	– – – –
SYS	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; YH:YM:YL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; XH:XM:XL ⇒ M(SP)~M(SP+2); (SP) – 4 ⇒ SP; D7H:D7MH:D7ML:D7L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D6H:D6MH:D6ML:D6L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D4H:D4L, D5H:D5L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D2H:D2L, D3H:D3L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; CCH, CCL, D0, D1 ⇒ M(SP)~M(SP+3); 1 ⇒ I; (SYS Vector) ⇒ PC System Call Software Interrupt (I bit can only be changed in supervisor state)	INH	1B 07	– – – 1 s	– – – –
TB*cc Di*,*oprmemreg*,*oprdest*	(D_i) – 1 ⇒ D_i Test and branch	REG-R7 REG-R15	0B lb rb 0B lb rb r1	– – – –	– – – –
TB*cc xy*,*oprmemreg*,*oprdest*	(X) – 1 ⇒ X or (Y) – 1 ⇒ Y Test and branch	REG-R7 REG-R15	0B lb rb 0B lb rb r1		
TB*cc*.*bwpl oprmemreg*,*oprdest*	(M) – 1 ⇒ M Test D_i, X, Y, or memory operand M, and branch if condition cc is true. Memory operand M may be 8, 16, 24, or 32 bits long cc can be Not Equal-TBNE, Equal-TBEQ, Plus-TBPL, Minus-TBMI, Greater Than-TBGT, or Less Than or Equal-TBLE (encoded in postbyte lb) Branch offset is 7 or 15 bits	OPR-R7 OPR-R15 OPR1-R7 OPR1-R15 OPR2-R7 OPR2-R15 OPR3-R7 OPR3-R15	0B lb xb rb 0B lb xb rb r1 0B lb xb x1 rb 0B lb xb x1 rb r1 0B lb xb x2 x1 rb 0B lb xb x2 x1 rb r1 0B lb xb x3 x2 x1 rb 0B lb xb x3 x2 x1 rb r1		
TFR *cpureg*,*cpureg*	(r1) ⇒ (r2) Transfer CPU Register r1 to r2 D0~D7, X, Y, S, CCH, CCL, or CCW if same size, direct transfer if 1st smaller than 2nd, zero-extend 1st to 2nd if 1st larger than 2nd, transfer low portion of 1st to 2nd (S, X, and I bits can only be changed in supervisor state) (If r2 = CCL or CCW, CCR bit may be written directly)	REG-REG	9E tb	– – – – c c c s s s	– – – – c c c c

Table A-1. S12Z Instruction Set Summary (Sheet 17 of 17)

Source Form	Operation	Address Mode(s)	Machine Coding (hex)	Condition Codes S X – I	Condition Codes N Z V C
TRAP *#trapnum*	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; YH:YM:YL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; XH:XM:XL ⇒ M(SP)~M(SP+2); (SP) – 4 ⇒ SP; D7H:D7MH:D7ML:D7L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D6H:D6MH:D6ML:D6L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D4H:D4L, D5H:D5L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D2H:D2L, D3H:D3L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; CCH, CCL, D0, D1 ⇒ M(SP)~M(SP+3); 1 ⇒ I; (TRAP Vector) ⇒ PC Unimplemented Opcode Trap Interrupt (I bit can only be changed in supervisor state)	INH	1B tn	– – – 1 s	– – – –
WAI	(SP) – 3 ⇒ SP; RTNH:RTNM:RTNL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; YH:YM:YL ⇒ M(SP)~M(SP+2); (SP) – 3 ⇒ SP; XH:XM:XL ⇒ M(SP)~M(SP+2); (SP) – 4 ⇒ SP; D7H:D7MH:D7ML:D7L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D6H:D6MH:D6ML:D6L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D4H:D4L, D5H:D5L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; D2H:D2L, D3H:D3L ⇒ M(SP)~M(SP+3); (SP) – 4 ⇒ SP; CCH, CCL, D0, D1 ⇒ M(SP)~M(SP+3); Wait for Interrupt (X and I set depending on interrupt source)	INH	1B 06	– – – – v v	– – – –
ZEX *cpureg,cpureg*	(r1) ⇒ (r2) Zero-extend CPU Register r1 to r2 D0~D7, X, Y, S, CCH, CCL, or CCW same as transfer TFR except r1 is smaller than r2 (S, X, and I bits can only be changed in supervisor state) (If r2 = CCL or CCW, CCR bit may be written directly)	REG-REG	9E tb	– – – – c c c s s s	– – – – c c c c

A.3 S12Z Opcode Map

Instruction opcodes are organized in two pages of 256 codes each. The opcodes on the first page are the most efficient because they require only one byte of object code. One of the codes on the first opcode page (code 0x1B) is used to select a second opcode page with 256 more instruction opcodes (not all are used). The instructions on this second opcode page require the pg2 prebyte plus an 8-bit opcode so they require a minimum of two bytes of object code. Opcode page 2 includes less-frequently-used instructions.

A few instructions span several opcodes because the opcode includes part of an address or a register code. There are 18-bit immediate versions of load X and load Y which use 2 bits of the opcode as the two highest order bits of the 18-bit immediate value. The other 16 bits of the immediate value are provided in two more bytes of object code after the opcode. The bit field instructions use three bits in the opcode to specify a source or destination register so these instructions span eight opcodes.

Table A-2. Opcode Map (Sheet 1 of 2)

00 BGND INH	10 SHIFT D2 postbyte sb	20 BRA REL	30 INC D2 INH	40 DEC D2 INH	50 ADD D2 IMM2	60 ADD D2 OPR	70 SUB D2 IMM2	80 SUB D2 OPR	90 LD D2 IMM2	A0 LD D2 OPR	B0 LD D2 EXT24	C0 ST D2 OPR	D0 ST D2 EXT24	E0 CMP D2 IMM2	F0 CMP D2 OPR
01 NOP INH	11 SHIFT D3 postbyte sb	21 BSR REL	31 INC D3 INH	41 DEC D3 INH	51 ADD D3 IMM2	61 ADD D3 OPR	71 SUB D3 IMM2	81 SUB D3 OPR	91 LD D3 IMM2	A1 LD D3 OPR	B1 LD D3 EXT24	C1 ST D3 OPR	D1 ST D3 EXT24	E1 CMP D3 IMM2	F1 CMP D3 OPR
02 BRCLR postbyte bm	12 SHIFT D4 postbyte sb	22 BHI REL	32 INC D4 INH	42 DEC D4 INH	52 ADD D4 IMM2	62 ADD D4 OPR	72 SUB D4 IMM2	82 SUB D4 OPR	92 LD D4 IMM2	A2 LD D4 OPR	B2 LD D4 EXT24	C2 ST D4 OPR	D2 ST D4 EXT24	E2 CMP D4 IMM2	F2 CMP D4 OPR
03 BRSET postbyte bm	13 SHIFT D5 postbyte sb	23 BLS REL	33 INC D5 INH	43 DEC D5 INH	53 ADD D5 IMM2	63 ADD D5 OPR	73 SUB D5 IMM2	83 SUB D5 OPR	93 LD D5 IMM2	A3 LD D5 OPR	B3 LD D5 EXT24	C3 ST D5 OPR	D3 ST D5 EXT24	E3 CMP D5 IMM2	F3 CMP D5 OPR
04 PSH/PUL postbyte pb	14 SHIFT D0 postbyte sb	24 BCC REL	34 INC D0 INH	44 DEC D0 INH	54 ADD D0 IMM1	64 ADD D0 OPR	74 SUB D0 IMM1	84 SUB D0 OPR	94 LD D0 IMM1	A4 LD D0 OPR	B4 LD D0 EXT24	C4 ST D0 OPR	D4 ST D0 EXT24	E4 CMP D0 IMM1	F4 CMP D0 OPR
05 RTS INH	15 SHIFT D1 postbyte sb	25 BCS REL	35 INC D1 INH	45 DEC D1 INH	55 ADD D1 IMM1	65 ADD D1 OPR	75 SUB D1 IMM1	85 SUB D1 OPR	95 LD D1 IMM1	A5 LD D1 OPR	B5 LD D1 EXT24	C5 ST D1 OPR	D5 ST D1 EXT24	E5 CMP D1 IMM1	F5 CMP D1 OPR
06 LEA D6 OPR	16 SHIFT D6 postbyte sb	26 BNE REL	36 INC D6 INH	46 DEC D6 INH	56 ADD D6 IMM4	66 ADD D6 OPR	76 SUB D6 IMM4	86 SUB D6 OPR	96 LD D6 IMM4	A6 LD D6 OPR	B6 LD D6 EXT24	C6 ST D6 OPR	D6 ST D6 EXT24	E6 CMP D6 IMM4	F6 CMP D6 OPR
07 LEA D7 OPR	17 SHIFT D7 postbyte sb	27 BEQ REL	37 INC D7 INH	47 DEC D7 INH	57 ADD D7 IMM4	67 ADD D7 OPR	77 SUB D7 IMM4	87 SUB D7 OPR	97 LD D7 IMM4	A7 LD D7 OPR	B7 LD D7 EXT24	C7 ST D7 OPR	D7 ST D7 EXT24	E7 CMP D7 IMM4	F7 CMP D7 OPR
08 LEA X OPR	18 LEA X (IMMs8,X)	28 BVC REL	38 CLR D2 INH	48 MUL D2 postbyte mb	58 AND D2 IMM2	68 AND D2 OPR	78 OR D2 IMM2	88 OR D2 OPR	98 LD X IMM3	A8 LD X OPR	B8 LD X EXT24	C8 ST X OPR	D8 ST X EXT24	E8 CMP X IMM3	F8 CMP X OPR
09 LEA Y OPR	19 LEA Y (IMMs8,Y)	29 BVS REL	39 CLR D3 INH	49 MUL D3 postbyte mb	59 AND D3 IMM2	69 AND D3 OPR	79 OR D3 IMM2	89 OR D3 OPR	99 LD Y IMM3	A9 LD Y OPR	B9 LD Y EXT24	C9 ST Y OPR	D9 ST Y EXT24	E9 CMP Y IMM3	F9 CMP Y OPR
0A LEA S OPR	1A LEA S (IMMs8,S)	2A BPL REL	3A CLR D4 INH	4A MUL D4 postbyte mb	5A AND D4 IMM2	6A AND D4 OPR	7A OR D4 IMM2	8A OR D4 OPR	9A CLR X INH	AA JMP OPR	BA JMP EXT24	CA,DA,EA,FA LD X IMMu18			
0B DBcc/TBcc postbyte lb	1B pg2	2B BMI REL	3B CLR D5 INH	4B MUL D5 postbyte mb	5B AND D5 IMM2	6B AND D5 OPR	7B OR D5 IMM2	8B OR D5 OPR	9B CLR Y INH	AB JSR OPR	BB JSR EXT24	CB,DB,EB,FB LD Y IMMu18			
0C MOV.B IMM1-OPR	1C MOV.B OPR-OPR	2C BGE REL	3C CLR D0 INH	4C MUL D0 postbyte mb	5C AND D0 IMM1	6C AND D0 OPR	7C OR D0 IMM1	8C OR D0 OPR	9C INC.B OPR	AC DEC.B OPR	BC CLR.B OPR	CC COM.B OPR	DC NEG.B OPR	EC BCLR postbyte bm	FC CMP X,Y INH
0D MOV.W IMM2-OPR	1D MOV.W OPR-OPR	2D BLT REL	3D CLR D1 INH	4D MUL D1 postbyte mb	5D AND D1 IMM1	6D AND D1 OPR	7D OR D1 IMM1	8D OR D1 OPR	9D INC.W OPR	AD DEC.W OPR	BD CLR.W OPR	CD COM.W OPR	DD NEG.W OPR	ED BSET postbyte bm	FD SUB D6,X,Y INH
0E MOV.P IMM3-OPR	1E MOV.P OPR-OPR	2E BGT REL	3E CLR D6 INH	4E MUL D6 postbyte mb	5E AND D6 IMM4	6E AND D6 OPR	7E OR D6 IMM4	8E OR D6 OPR	9E TFR postbyte tb	AE EXG/SEX postbyte eb	BE CLR.P OPR	CE ANDCC IMM1	DE ORCC IMM1	EE BTGL postbyte bm	FE SUB D6,Y,X INH
0F MOV.L IMM4-OPR	1F MOV.L OPR-OPR	2F BLE REL	3F CLR D7 INH	4F MUL D7 postbyte mb	5F AND D7 IMM4	6F AND D7 OPR	7F OR D7 IMM4	8F OR D7 OPR	9F INC.L OPR	AF DEC.L OPR	BF CLR.L OPR	CF COM.L OPR	DF NEG.L OPR	EF SPARE	FF SWI INH

Opcode in Hexadecimal	F0 BRA REL	Instruction Mnemonic Addressing Mode(s) or Postbyte

Table A-2. Opcode Map (Sheet 2 of 2)

1B 00 LD S OPR	1B 10 MINU D2 OPR	1B 20 MINS D2 OPR	1B 30 DIV D2 postbyte mb	1B 40 ABS D2 INH	1B 50 ADC D2 IMM2	1B 60 ADC D2 OPR	1B 70 SBC D2 IMM2	1B 80 SBC D2 OPR	1B 90 RTI INH	1B A0 SAT D2 INH	1B B0 QMUL D2 postbyte mb	1B C0 TRAP INH	1B D0 TRAP INH	1B E0 TRAP INH	1B F0 TRAP INH
1B 01 ST S OPR	1B 11 MINU D3 OPR	1B 21 MINS D3 OPR	1B 31 DIV D3 postbyte mb	1B 41 ABS D3 INH	1B 51 ADC D3 IMM2	1B 61 ADC D3 OPR	1B 71 SBC D3 IMM2	1B 81 SBC D3 OPR	1B 91 CLB postbyte cb	1B A1 SAT D3 INH	1B B1 QMUL D3 postbyte mb	1B C1 TRAP INH	1B D1 TRAP INH	1B E1 TRAP INH	1B F1 TRAP INH
1B 02 CMP S OPR	1B 12 MINU D4 OPR	1B 22 MINS D4 OPR	1B 32 DIV D4 postbyte mb	1B 42 ABS D4 INH	1B 52 ADC D4 IMM2	1B 62 ADC D4 OPR	1B 72 SBC D4 IMM2	1B 82 SBC D4 OPR	1B 92 TRAP INH	1B A2 SAT D4 INH	1B B2 QMUL D4 postbyte mb	1B C2 TRAP INH	1B D2 TRAP INH	1B E2 TRAP INH	1B F2 TRAP INH
1B 03 LD S IMM3	1B 13 MINU D5 OPR	1B 23 MINS D5 OPR	1B 33 DIV D5 postbyte mb	1B 43 ABS D5 INH	1B 53 ADC D5 IMM2	1B 63 ADC D5 OPR	1B 73 SBC D5 IMM2	1B 83 SBC D5 OPR	1B 93 TRAP INH	1B A3 SAT D5 INH	1B B3 QMUL D5 postbyte mb	1B C3 TRAP INH	1B D3 TRAP INH	1B E3 TRAP INH	1B F3 TRAP INH
1B 04 CMP S IMM3	1B 14 MINU D0 OPR	1B 24 MINS D0 OPR	1B 34 DIV D0 postbyte mb	1B 44 ABS D0 INH	1B 54 ADC D0 IMM1	1B 64 ADC D0 OPR	1B 74 SBC D0 IMM1	1B 84 SBC D0 OPR	1B 94 TRAP INH	1B A4 SAT D0 INH	1B B4 QMUL D0 postbyte mb	1B C4 TRAP INH	1B D4 TRAP INH	1B E4 TRAP INH	1B F4 TRAP INH
1B 05 STOP INH	1B 15 MINU D1 OPR	1B 25 MINS D1 OPR	1B 35 DIV D1 postbyte mb	1B 45 ABS D1 INH	1B 55 ADC D1 IMM1	1B 65 ADC D1 OPR	1B 75 SBC D1 IMM1	1B 85 SBC D1 OPR	1B 95 TRAP INH	1B A5 SAT D1 INH	1B B5 QMUL D1 postbyte mb	1B C5 TRAP INH	1B D5 TRAP INH	1B E5 TRAP INH	1B F5 TRAP INH
1B 06 WAI INH	1B 16 MINU D6 OPR	1B 26 MINS D6 OPR	1B 36 DIV D6 postbyte mb	1B 46 ABS D6 INH	1B 56 ADC D6 IMM4	1B 66 ADC D6 OPR	1B 76 SBC D6 IMM4	1B 86 SBC D6 OPR	1B 96 TRAP INH	1B A6 SAT D6 INH	1B B6 QMUL D6 postbyte mb	1B C6 TRAP INH	1B D6 TRAP INH	1B E6 TRAP INH	1B F6 TRAP INH
1B 07 SYS INH	1B 17 MINU D7 OPR	1B 27 MINS D7 OPR	1B 37 DIV D7 postbyte mb	1B 47 ABS D7 INH	1B 57 ADC D7 IMM4	1B 67 ADC D7 OPR	1B 77 SBC D7 IMM4	1B 87 SBC D7 OPR	1B 97 TRAP INH	1B A7 SAT D7 INH	1B B7 QMUL D7 postbyte mb	1B C7 TRAP INH	1B D7 TRAP INH	1B E7 TRAP INH	1B F7 TRAP INH
1B 08 1B 09 1B 0A 1B 0B 1B 0C 1B 0D 1B 0E 1B 0F	1B 18 MAXU D2 OPR	1B 28 MAXS D2 OPR	1B 38 MOD D2 postbyte mb	1B 48 MAC D2 postbyte mb	1B 58 BIT D2 IMM2	1B 68 BIT D2 OPR	1B 78 EOR D2 IMM2	1B 88 EOR D2 OPR	1B 98 TRAP INH	1B A8 TRAP INH	1B B8 TRAP INH	1B C8 TRAP INH	1B D8 TRAP INH	1B E8 TRAP INH	1B F8 TRAP INH
	1B 19 MAXU D3 OPR	1B 29 MAXS D3 OPR	1B 39 MOD D3 postbyte mb	1B 49 MAC D3 postbyte mb	1B 59 BIT D3 IMM2	1B 69 BIT D3 OPR	1B 79 EOR D3 IMM2	1B 89 EOR D3 OPR	1B 99 TRAP INH	1B A9 TRAP INH	1B B9 TRAP INH	1B C9 TRAP INH	1B D9 TRAP INH	1B E9 TRAP INH	1B F9 TRAP INH
	1B 1A MAXU D4 OPR	1B 2A MAXS D4 OPR	1B 3A MOD D4 postbyte mb	1B 4A MAC D4 postbyte mb	1B 5A BIT D4 IMM2	1B 6A BIT D4 OPR	1B 7A EOR D4 IMM2	1B 8A EOR D4 OPR	1B 9A TRAP INH	1B AA TRAP INH	1B BA TRAP INH	1B CA TRAP INH	1B DA TRAP INH	1B EA TRAP INH	1B FA TRAP INH
	1B 1B MAXU D5 OPR	1B 2B MAXS D5 OPR	1B 3B MOD D5 postbyte mb	1B 4B MAC D5 postbyte mb	1B 5B BIT D5 IMM2	1B 6B BIT D5 OPR	1B 7B EOR D5 IMM2	1B 8B EOR D5 OPR	1B 9B TRAP INH	1B AB TRAP INH	1B BB TRAP INH	1B CB TRAP INH	1B DB TRAP INH	1B EB TRAP INH	1B FB TRAP INH
BFEXT BFINS postbyte bb	1B 1C MAXU D0 OPR	1B 2C MAXS D0 OPR	1B 3C MOD D0 postbyte mb	1B 4C MAC D0 postbyte mb	1B 5C BIT D0 IMM1	1B 6C BIT D0 OPR	1B 7C EOR D0 IMM1	1B 8C EOR D0 OPR	1B 9C TRAP INH	1B AC TRAP INH	1B BC TRAP INH	1B CC TRAP INH	1B DC TRAP INH	1B EC TRAP INH	1B FC TRAP INH
	1B 1D MAXU D1 OPR	1B 2D MAXS D1 OPR	1B 3D MOD D1 postbyte mb	1B 4D MAC D1 postbyte mb	1B 5D BIT D1 IMM1	1B 6D BIT D1 OPR	1B 7D EOR D1 IMM1	1B 8D EOR D1 OPR	1B 9D TRAP INH	1B AD TRAP INH	1B BD TRAP INH	1B CD TRAP INH	1B DD TRAP INH	1B ED TRAP INH	1B FD TRAP INH
	1B 1E MAXU D6 OPR	1B 2E MAXS D6 OPR	1B 3E MOD D6 postbyte mb	1B 4E MAC D6 postbyte mb	1B 5E BIT D6 IMM4	1B 6E BIT D6 OPR	1B 7E EOR D6 IMM4	1B 8E EOR D6 OPR	1B 9E TRAP INH	1B AE TRAP INH	1B BE TRAP INH	1B CE TRAP INH	1B DE TRAP INH	1B EE TRAP INH	1B FE TRAP INH
	1B 1F MAXU D7 OPR	1B 2F MAXS D7 OPR	1B 3F MOD D7 postbyte mb	1B 4F MAC D7 postbyte mb	1B 5F BIT D7 IMM4	1B 6F BIT D7 OPR	1B 7F EOR D7 IMM4	1B 8F EOR D7 OPR	1B 9F TRAP INH	1B AF TRAP INH	1B BF TRAP INH	1B CF TRAP INH	1B DF TRAP INH	1B EF TRAP INH	1B FF TRAP INH

Opcode in Hexadecimal	1B 00 LD S OPR	Instruction Mnemonic Addressing Mode(s) or Postbyte

A.4 Postbyte Coding

Many instructions use a postbyte to provide variations of the instructions including various addressing mode combinations for instructions with two or more operands. Refer to the tables and explanations on the following pages for a complete description of postbyte coding.

A.4.1 General Operand (OPR) Addressing Postbyte (xb)

Instead of having separate opcodes for every possible addressing mode, instructions such as load (LD), store (ST), and ADD use a postbyte to specify the addressing mode that is used to access an instruction operand. Some instructions such as the math instructions MUL, MAC, DIV, and MOD or the move instructions, have two operands and each of these operands can use a separate xb postbyte to specify the memory location or register to be used in the instruction.

The xb postbyte allows 16 submodes as shown in the following table. These include indexed addressing modes, three variations of extended addressing mode, register-as-memory, and a short-immediate mode for quickly initializing registers with common constants such as –1 or +2.

Table A-3. General Operand Addressing Postbyte (xb) Decode

xb postbyte bitwise encoding								Summary Source Form	Summary Address Mode	Operand Machine Coding	Detailed Source Form	Detailed Addressing Modes
b7	b6	b5	b4	b3	b2	b1	b0					
0	1	1	1	e4 IMM (–1, 1, 2 ... 14, 15)				INST *oprmemreg*	OPR	xb	INST #*oprsxe4i*	**IMMe4** - Short Immediate (–1, 1, 2, 3..14, 15)*
1	0	1	1	1	D[2:0]						INST *Di*	**REG** - Register as operand
0	1	XYS		u4 (0...15)							INST (*opru4*,*xys*)	**IDX** - u4 Constant offset from xys
1	1	1	XY	0	0	1	1				INST (+*xy*)	**++IDX** - Pre/post inc/dec +–xy+–,–s+
1	1	1	XY	0	1	1	1				INST (*xy*+)	
1	1	0	XY	0	0	1	1				INST (–*xy*)	
1	1	0	XY	0	1	1	1				INST (*xy*–)	
1	1	1	1	1	0	1	1				INST (–*s*)	
1	1	1	1	1	1	1	1				INST (*s*+)	
1	0	XYS		1	D[2:0]						INST (*Di*,*xys*)	**REG,IDX** - Register offset from xys D0,D1,D6,D7 unsigned; D2~D5 signed
1	1	0	XY	1	D[2:0]						INST [*Di*,*xy*]	**[REG,IDX]** - Register offset from xys **Indirect** D0,D1,D6,D7 unsigned; D2~D5 signed
1	1	XYSP		0	0	0	sign		OPR1	xb x1	INST (*oprs9*,*xysp*)	**IDX1** - s9 Constant offset from xysp; –256 to +255
1	1	XYSP		0	1	0	sign				INST [*oprs9*,*xysp*]	**[IDX1]** - s9 offset from xysp **Indirect**; –256 to +255
0	0	Addr[13:8]									INST *opru14*	**EXT1** - u14 Short Extended (first 16K)
1	0	Addr[17:16]		0	D[2:0]				OPR2	xb x2 x1	INST (*opru18*,*Di*)	**IDX2,REG** - u18 offset from Di (256K) D0,D1,D6,D7 unsigned; D2~D5 signed
1	1	1	1	1	A17	0	A16				INST *opru18*	**EXT2** - u18 Extended (256K)
1	1	XYSP		0	0	1	0		OPR3	xb x3 x2 x1	INST (*opr24*,*xysp*)	**IDX3** - 24b constant offset from xysp
1	1	XYSP		0	1	1	0				INST [*opr24*,*xysp*]	**[IDX3]** - 24b offset from xysp **Indirect**
1	1	1	0	1	D[2:0]						INST (*opru24*,*Di*)	**IDX3,REG** - 24b offset from Di D0,D1,D6,D7 unsigned; D2~D5 signed
1	1	1	1	1	0	1	0				INST *opr24*	**EXT3** - 24b Extended (full 16M)
1	1	1	1	1	1	1	0				INST [*opr24*]	**[EXT3]** - 24b address **Indirect**

The IMMe4 short immediate mode uses an enumerated 4-bit code to select 1-of-16 constants where 0:0:0:0 indicates –1 and the remaining 15 codes indicate the values 1, 2, ...14, 15. These constants are automatically sign-extended to the size of the operation. For example, the instruction LD X #–1 is an efficient 2-byte instruction which loads 0xFFFFFF into the 24-bit index register.

* Shift instructions treat the 4-bit short immediate value as the upper four bits of a 5-bit immediate value where the least significant bit of the 5-bit value is located in the shift postbyte.

D[2:0] selects 1-of-8 CPU data registers 0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7. For XY, 0=X and 1=Y. For XYSP, 0:0=X, 0:1=Y, 1:0=S, and 1:1=PC. For XYS,

0:0=X, 0:1=Y, 1:0=S, and the remaining 1:1 code corresponds to another row in the decode table. The bit labeled sign holds the high-order 9th (or sign bit) of a 9-bit signed value.

The following table shows the coding map for the xb postbyte, that results from the above decode.

Table A-4. General Operand Addressing Postbyte (xb) Coding Map

	0_	1_	2_	3_	4_	5_	6_	7_	8_	9_	A_	B_	C_	D_	E_	F_
_0	n postbyte + 1 extension byte for low-order 8 address bits EXT1 u14 May be used to access first 16K of address space which includes all I/O and control registers plus ~14K of RAM				0,X IDX u4	0,Y IDX u4	0,S IDX u4	–1 IMMe4	n,D2 IDX2,REG u18				n,X IDX1 s9	n,Y IDX1 s9	n,S IDX1 s9	n,PC IDX1 s9
_1					1,X IDX u4	1,Y IDX u4	1,S IDX u4	1 IMMe4	n,D3 IDX2,REG u18							
_2					2,X IDX u4	2,Y IDX u4	2,S IDX u4	2 IMMe4	n,D4 IDX2,REG u18				n,X IDX3 24b	n,Y IDX3 24b	n,S IDX3 24b	n,PC IDX3 24b
_3					3,X IDX u4	3,Y IDX u4	3,S IDX u4	3 IMMe4	n,D5 IDX2,REG u18				auto,–X ++IDX	auto,–Y ++IDX	auto,+X ++IDX	auto,+Y ++IDX
_4					4,X IDX u4	4,Y IDX u4	4,S IDX u4	4 IMMe4	n,D0 IDX2,REG u18				[n,X] [IDX1] s9	[n,Y] [IDX1] s9	[n,S] [IDX1] s9	[n,PC] [IDX1] s9
_5					5,X IDX u4	5,Y IDX u4	5,S IDX u4	5 IMMe4	n,D1 IDX2,REG u18							
_6					6,X IDX u4	6,Y IDX u4	6,S IDX u4	6 IMMe4	n,D6 IDX2,REG u18				[n,X] [IDX3] 24b	[n,Y] [IDX3] 24b	[n,S] [IDX3] 24b	[n,PC] [IDX3] 24b
_7					7,X IDX u4	7,Y IDX u4	7,S IDX u4	7 IMMe4	n,D7 IDX2,REG u18				auto,X– ++IDX	auto,Y– ++IDX	auto,X+ ++IDX	auto,Y+ ++IDX
_8					8,X IDX u4	8,Y IDX u4	8,S IDX u4	8 IMMe4	D2,X REG,IDX	D2,Y REG,IDX	D2,S REG,IDX	D2 REG	[D2,X] [REG,IDX]	[D2,Y] [REG,IDX]	n,D2 IDX3,REG 24b	n EXT2 u18
_9					9,X IDX u4	9,Y IDX u4	9,S IDX u4	9 IMMe4	D3,X REG,IDX	D3,Y REG,IDX	D3,S REG,IDX	D3 REG	[D3,X] [REG,IDX]	[D3,Y] [REG,IDX]	n,D3 IDX3,REG 24b	
_A					10,X IDX u4	10,Y IDX u4	10,S IDX u4	10 IMMe4	D4,X REG,IDX	D4,Y REG,IDX	D4,S REG,IDX	D4 REG	[D4,X] [REG,IDX]	[D4,Y] [REG,IDX]	n,D4 IDX3,REG 24b	n EXT3 24b
_B					11,X IDX u4	11,Y IDX u4	11,S IDX u4	11 IMMe4	D5,X REG,IDX	D5,Y REG,IDX	D5,S REG,IDX	D5 REG	[D5,X] [REG,IDX]	[D5,Y] [REG,IDX]	n,D5 IDX3,REG 24b	auto,–S ++IDX
_C					12,X IDX u4	12,Y IDX u4	12,S IDX u4	12 IMMe4	D0,X REG,IDX	D0,Y REG,IDX	D0,S REG,IDX	D0 REG	[D0,X] [REG,IDX]	[D0,Y] [REG,IDX]	n,D0 IDX3,REG 24b	n EXT2 u18
_D					13,X IDX u4	13,Y IDX u4	13,S IDX u4	13 IMMe4	D1,X REG,IDX	D1,Y REG,IDX	D1,S REG,IDX	D1 REG	[D1,X] [REG,IDX]	[D1,Y] [REG,IDX]	n,D1 IDX3,REG 24b	
_E					14,X IDX u4	14,Y IDX u4	14,S IDX u4	14 IMMe4	D6,X REG,IDX	D6,Y REG,IDX	D6,S REG,IDX	D6 REG	[D6,X] [REG,IDX]	[D6,Y] [REG,IDX]	n,D6 IDX3,REG 24b	[n] [EXT3] 24b
_F					15,X IDX u4	15,Y IDX u4	15,S IDX u4	15 IMMe4	D7,X REG,IDX	D7,Y REG,IDX	D7,S REG,IDX	D7 REG	[D7,X] [REG,IDX]	[D7,Y] [REG,IDX]	n,D7 IDX3,REG 24b	auto,S+ ++IDX

A.4.2 Math Postbyte (mb) for MUL, MAC, DIV, MOD and QMUL

For math instructions MUL, MAC, DIV, MOD, and QMUL, the destination is specified in bits 2:0 of the opcode and the mb postbyte specifies the addressing modes for the two source operands. OPR addressing modes support 16 general operand addressing sub-modes including indexed, extended, register, and auto increment modes.

In the following decode table, Rs1 and Rs2 refer to source operand registers for the first and second operands. The 3-bit codes select 1-of-8 CPU data registers 0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7. Memory operand size options include 8-bit byte, 16-bit word, 24-bit pointer, and 32-bit long word.

Table A-5. MUL, MAC, DIV, MOD, and QMUL Postbyte (mb) Postbyte Decode

mb postbyte bitwise encoding								Addressing modes for Operand 1 , Operand 2
b7	b6	b5	b4	b3	b2	b1	b0	
1 = Signed, 0 = Unsigned	0	Rs1			Rs2			Register , Register
	1	Rs1			0	Size 2 = 0:0 byte		Register , OPR.B
	1	Rs1			0	Size 2 = 0:1 word		Register , OPR.W
	1	Rs1			0	Size 2 = 1:1 long		Register , OPR.L
	1	Rs1			1	Size 2 = 0:0 byte		Register , IMM1
	1	Rs1			1	Size 2 = 0:1 word		Register , IMM2
	1	Rs1			1	Size 2 = 1:1 long		Register , IMM4
	1	Size 1 = 0:0 byte		Size 2 = 0:0 byte		1	0	OPR.B , OPR.B
	1	Size 1 = 0:0 byte		Size 2 = 0:1 word		1	0	OPR.B , OPR.W
	1	Size 1 = 0:0 byte		Size 2 = 1:0 pointer		1	0	OPR.B , OPR.P
	1	Size 1 = 0:0 byte		Size 2 = 1:1 long		1	0	OPR.B , OPR.L
	1	Size 1 = 0:1 word		Size 2 = 0:0 byte		1	0	OPR.W , OPR.B
	1	Size 1 = 0:1 word		Size 2 = 0:1 word		1	0	OPR.W , OPR.W
	1	Size 1 = 0:1 word		Size 2 = 1:0 pointer		1	0	OPR.W , OPR.P
	1	Size 1 = 0:1 word		Size 2 = 1:1 long		1	0	OPR.W , OPR.L
	1	Size 1 = 1:0 pointer		Size 2 = 0:0 byte		1	0	OPR.P , OPR.B
	1	Size 1 = 1:0 pointer		Size 2 = 0:1 word		1	0	OPR.P , OPR.W
	1	Size 1 = 1:0 pointer		Size 2 = 1:0 pointer		1	0	OPR.P , OPR.P
	1	Size 1 = 1:0 pointer		Size 2 = 1:1 long		1	0	OPR.P , OPR.L
	1	Size 1 = 1:1 long		Size 2 = 0:0 byte		1	0	OPR.L , OPR.B
	1	Size 1 = 1:1 long		Size 2 = 0:1 word		1	0	OPR.L , OPR.W
	1	Size 1 = 1:1 long		Size 2 = 1:0 pointer		1	0	OPR.L , OPR.P
	1	Size 1 = 1:1 long		Size 2 = 1:1 long		1	0	OPR.L , OPR.L

The following table shows the coding map for the mb postbyte, that results from the above decode.

Table A-6. MUL, MAC, DIV, MOD, and QMUL Postbyte (mb) Coding Map

	Unsigned								Signed							
	0_	1_	2_	3_	4_	5_	6_	7_	8_	9_	A_	B_	C_	D_	E_	F_
_0	D2, D2	D4, D2	D0, D2	D6, D2	D2, OPR.B	D4, OPR.B	D0, OPR.B	D6, OPR.B	D2, D2	D4, D2	D0, D2	D6, D2	D2, OPR.B	D4, OPR.B	D0, OPR.B	D6, OPR.B
_1	D2, D3	D4, D3	D0, D3	D6, D3	D2, OPR.W	D4, OPR.W	D0, OPR.W	D6, OPR.W	D2, D3	D4, D3	D0, D3	D6, D3	D2, OPR.W	D4, OPR.W	D0, OPR.W	D6, OPR.W
_2	D2, D4	D4, D4	D0, D4	D6, D4	OPR.B, OPR.B	OPR.W, OPR.B	OPR.P, OPR.B	OPR.L, OPR.B	D2, D4	D4, D4	D0, D4	D6, D4	OPR.B, OPR.B	OPR.W, OPR.B	OPR.P, OPR.B	OPR.L, OPR.B
_3	D2, D5	D4, D5	D0, D5	D6, D5	D2, OPR.L	D4, OPR.L	D0, OPR.L	D6, OPR.L	D2, D5	D4, D5	D0, D5	D6, D5	D2, OPR.L	D4, OPR.L	D0, OPR.L	D6, OPR.L
_4	D2, D0	D4, D0	D0, D0	D6, D0	D2, IMM1	D4, IMM1	D0, IMM1	D6, IMM1	D2, D0	D4, D0	D0, D0	D6, D0	D2, IMM1	D4, IMM1	D0, IMM1	D6, IMM1
_5	D2, D1	D4, D1	D0, D1	D6, D1	D2, IMM2	D4, IMM2	D0, IMM2	D6, IMM2	D2, D1	D4, D1	D0, D1	D6, D1	D2, IMM2	D4, IMM2	D0, IMM2	D6, IMM2
_6	D2, D6	D4, D6	D0, D6	D6, D6	OPR.B, OPR.W	OPR.W, OPR.W	OPR.P, OPR.W	OPR.L, OPR.W	D2, D6	D4, D6	D0, D6	D6, D6	OPR.B, OPR.W	OPR.W, OPR.W	OPR.P, OPR.W	OPR.L, OPR.W
_7	D2, D7	D4, D7	D0, D7	D6, D7	D2, IMM4	D4, IMM4	D0, IMM4	D6, IMM4	D2, D7	D4, D7	D0, D7	D6, D7	D2, IMM4	D4, IMM4	D0, IMM4	D6, IMM4
_8	D3, D2	D5, D2	D1, D2	D7, D2	D3, OPR.B	D5, OPR.B	D1, OPR.B	D7, OPR.B	D3, D2	D5, D2	D1, D2	D7, D2	D3, OPR.B	D5, OPR.B	D1, OPR.B	D7, OPR.B
_9	D3, D3	D5, D3	D1, D3	D7, D3	D3, OPR.W	D5, OPR.W	D1, OPR.W	D7, OPR.W	D3, D3	D5, D3	D1, D3	D7, D3	D3, OPR.W	D5, OPR.W	D1, OPR.W	D7, OPR.W
_A	D3, D4	D5, D4	D1, D4	D7, D4	OPR.B, OPR.P	OPR.W, OPR.P	OPR.P, OPR.P	OPR.L, OPR.P	D3, D4	D5, D4	D1, D4	D7, D4	OPR.B, OPR.P	OPR.W, OPR.P	OPR.P, OPR.P	OPR.L, OPR.P
_B	D3, D5	D5, D5	D1, D5	D7, D5	D3, OPR.L	D5, OPR.L	D1, OPR.L	D7, OPR.L	D3, D5	D5, D5	D1, D5	D7, D5	D3, OPR.L	D5, OPR.L	D1, OPR.L	D7, OPR.L
_C	D3, D0	D5, D0	D1, D0	D7, D0	D3, IMM1	D5, IMM1	D1, IMM1	D7, IMM1	D3, D0	D5, D0	D1, D0	D7, D0	D3, IMM1	D5, IMM1	D1, IMM1	D7, IMM1
_D	D3, D1	D5, D1	D1, D1	D7, D1	D3, IMM2	D5, IMM2	D1, IMM2	D7, IMM2	D3, D1	D5, D1	D1, D1	D7, D1	D3, IMM2	D5, IMM2	D1, IMM2	D7, IMM2
_E	D3, D6	D5, D6	D1, D6	D7, D6	OPR.B, OPR.L	OPR.W, OPR.L	OPR.P, OPR.L	OPR.L, OPR.L	D3, D6	D5, D6	D1, D6	D7, D6	OPR.B, OPR.L	OPR.W, OPR.L	OPR.P, OPR.L	OPR.L, OPR.L
_F	D3, D7	D5, D7	D1, D7	D7, D7	D3, IMM4	D5, IMM4	D1, IMM4	D7, IMM4	D3, D7	D5, D7	D1, D7	D7, D7	D3, IMM4	D5, IMM4	D1, IMM4	D7, IMM4

A.4.3 Loop Primitive Postbyte (lb)

The lb postbyte shows the coding for the DBcc and TBcc loop primitive instructions. Decrement or Test D_i, X, Y, or a byte, word, pointer, or long memory location and then branch based on EQ, NE, PL, MI, GT, or LE. Unused codes for CC test or decrement, but do not branch or change the CCR bits.

Table A-7. Loop Instruction Postbyte (lb) Decode

lb postbyte bitwise encoding								Comments
b7	b6	b5	b4	b3	b2	b1	b0	
D/T	CC			0	Di[2:0]			Test or Decrement Di and then branch based on condition CC (NE,EQ,PL,MI,GT,or LE)
D/T	CC			1	0	x	Y/X	Test or Decrement X or Y and then branch based on condition CC (NE,EQ,PL,MI,GT,or LE)
D/T	CC			1	1	SIZE		Test or Decrement Memory location (.B,.W,.P, or .L) and then branch based on condition CC

Table A-8. Loop Postbyte (lb) Coding Map

	Test and Branch								Decrement and Branch							
	0_	1_	2_	3_	4_	5_	6_	7_	8_	9_	A_	B_	C_	D_	E_	F_
_0	TBNE D2	TBEQ D2	TBPL D2	TBMI D2	TBGT D2	TBLE D2	reserved TBRN	reserved TBRN	DBNE D2	DBEQ D2	DBPL D2	DBMI D2	DBGT D2	DBLE D2	reserved DBRN	reserved DBRN
_1	TBNE D3	TBEQ D3	TBPL D3	TBMI D3	TBGT D3	TBLE D3	reserved TBRN	reserved TBRN	DBNE D3	DBEQ D3	DBPL D3	DBMI D3	DBGT D3	DBLE D3	reserved DBRN	reserved DBRN
_2	TBNE D4	TBEQ D4	TBPL D4	TBMI D4	TBGT D4	TBLE D4	reserved TBRN	reserved TBRN	DBNE D4	DBEQ D4	DBPL D4	DBMI D4	DBGT D4	DBLE D4	reserved DBRN	reserved DBRN
_3	TBNE D5	TBEQ D5	TBPL D5	TBMI D5	TBGT D5	TBLE D5	reserved TBRN	reserved TBRN	DBNE D5	DBEQ D5	DBPL D5	DBMI D5	DBGT D5	DBLE D5	reserved DBRN	reserved DBRN
_4	TBNE D0	TBEQ D0	TBPL D0	TBMI D0	TBGT D0	TBLE D0	reserved TBRN	reserved TBRN	DBNE D0	DBEQ D0	DBPL D0	DBMI D0	DBGT D0	DBLE D0	reserved DBRN	reserved DBRN
_5	TBNE D1	TBEQ D1	TBPL D1	TBMI D1	TBGT D1	TBLE D1	reserved TBRN	reserved TBRN	DBNE D1	DBEQ D1	DBPL D1	DBMI D1	DBGT D1	DBLE D1	reserved DBRN	reserved DBRN
_6	TBNE D6	TBEQ D6	TBPL D6	TBMI D6	TBGT D6	TBLE D6	reserved TBRN	reserved TBRN	DBNE D6	DBEQ D6	DBPL D6	DBMI D6	DBGT D6	DBLE D6	reserved DBRN	reserved DBRN
_7	TBNE D7	TBEQ D7	TBPL D7	TBMI D7	TBGT D7	TBLE D7	reserved TBRN	reserved TBRN	DBNE D7	DBEQ D7	DBPL D7	DBMI D7	DBGT D7	DBLE D7	reserved DBRN	reserved DBRN
_8	TBNE X	TBEQ X	TBPL X	TBMI X	TBGT X	TBLE X	reserved TBRN	reserved TBRN	DBNE X	DBEQ X	DBPL X	DBMI X	DBGT X	DBLE X	reserved DBRN	reserved DBRN
_9	TBNE Y	TBEQ Y	TBPL Y	TBMI Y	TBGT Y	TBLE Y	reserved TBRN	reserved TBRN	DBNE Y	DBEQ Y	DBPL Y	DBMI Y	DBGT Y	DBLE Y	reserved DBRN	reserved DBRN
_A	TBNE X	TBEQ X	TBPL X	TBMI X	TBGT X	TBLE X	reserved TBRN	reserved TBRN	DBNE X	DBEQ X	DBPL X	DBMI X	DBGT X	DBLE X	reserved DBRN	reserved DBRN
_B	TBNE Y	TBEQ Y	TBPL Y	TBMI Y	TBGT Y	TBLE Y	reserved TBRN	reserved TBRN	DBNE Y	DBEQ Y	DBPL Y	DBMI Y	DBGT Y	DBLE Y	reserved DBRN	reserved DBRN
_C	TBNE.B mem	TBEQ.B mem	TBPL.B mem	TBMI.B mem	TBGT.B mem	TBLE.B mem	reserved TBRN	reserved TBRN	DBNE.B mem	DBEQ.B mem	DBPL.B mem	DBMI.B mem	DBGT.B mem	DBLE.B mem	reserved DBRN	reserved DBRN
_D	TBNE.W mem	TBEQ.W mem	TBPL.W mem	TBMI.W mem	TBGT.W mem	TBLE.W mem	reserved TBRN	reserved TBRN	DBNE.W mem	DBEQ.W mem	DBPL.W mem	DBMI.W mem	TBGT.W mem	TBLE.W mem	reserved DBRN	reserved DBRN
_E	TBNE.P mem	TBEQ.P mem	TBPL.P mem	TBMI.P mem	TBGT.P mem	TBLE.P mem	reserved TBRN	reserved TBRN	DBNE.P mem	DBEQ.P mem	DBPL.P mem	DBMI.P mem	TBGT.P mem	TBLE.P mem	reserved DBRN	reserved DBRN
_F	TBNE.L mem	TBEQ.L mem	TBPL.L mem	TBMI.L mem	TBGT.L mem	TBLE.L mem	reserved TBRN	reserved TBRN	DBNE.L mem	DBEQ.L mem	DBPL.L mem	DBMI.L mem	TBGT.L mem	TBLE.L mem	reserved DBRN	reserved DBRN

A.4.4 Shift and Rotate Postbyte (sb)

The sb postbyte selects arithmetic or logical shift (A/L), direction (L/R), the low-order bit of the shift count, and the source register or memory operand size. The destination of 3-operand shift instructions is one of the eight CPU data registers Dd (encoded in the opcode) except in the case of 2-operand memory shifts. The source can be a CPU data register Ds or a byte, word, pointer, or long-word memory operand. There are efficient 2-byte instructions for shifting by 1 or 2 bit positions and versions with another postbyte (xb) for shifting by up to 31 bit positions and specifying that the shift count is in another register or memory location. 2-operand memory shifts allow a byte, word, pointer, or long-word memory location to be shifted by n=1 or n=2. Rotate instructions allow a register or memory location to be rotated left or right by one bit position. Shaded codes in the coding map are reserved for future use but default as shown.

Table A-9. Shift Postbyte (sb) Decode

sb postbyte bitwise encoding								Comments	Source Syntax
b7	b6	b5	b4	b3	b2	b1	b0		
A/L	L/R	0	0	N[0]	Ds[2:0]			Dd <= Ds <<>> #n Efficient shift by n=1 or 2	`SHFT Dd,Ds,#opr1i`

A/L	L/R	0	1	N[0]
				x
				x
					Ds[2:0]			Dd <= Ds <<>> *oprmemreg* N[4:1], Dn, or byte-sized n value specified using xb postbyte	`SHFT Dd,Ds,#opr5i`
									`SHFT Dd,Ds,Dn`
									`SHFT Dd,Ds,oprmemreg`

A/L	L/R	1	0	N[0]	0	size[1:0]		Dd <= *oprmemreg* <<>> #n Efficient shift by n=1 or 2	`SHFT.bwpl Dd,oprmemreg,#opr1i`
x	L/R	1	0	x	1	size[1:0]		Rotate *oprmemreg* left or right by n=1	`SHFT oprmemreg :ROL or ROR`

A/L	L/R	1	1	N[0]
				x
				x
					0	size[1:0]		Dd <= *oprmemreg* <<>> *oprmemreg* N[4:1], Dn, or byte-sized n value specified using second xb postbyte	`SHFT.bwpl Dd,oprmemreg,#opr5i`
									`SHFT.bwpl Dd,oprmemreg,Dn`
									`SHFT.bwpl Dd,oprmemreg,oprmemreg`

A/L	L/R	1	1	N[0]	1	size[1:0]		Shift *oprmemreg* left or right by n=1 or 2	`SHFT Di,#opr1i` `SHFT.bwpl oprmemreg,#opr1i`

Table A-10. Shift Postbyte (sb) Coding Map

	Logical								Arithmetic							
	Right				Left				Right				Left			
	0_	1_	2_	3_	4_	5_	6_	7_	8_	9_	A_	B_	C_	D_	E_	F_
_0	LSR Dd D2 IMM n=1	LSR Dd D2 postbyte xb	LSR Dd OPR.B IMM n=1	LSR Dd OPR.B postbyte xb	LSL Dd D2 IMM n=1	LSL Dd D2 postbyte xb	LSL Dd OPR.B IMM n=1	LSL Dd OPR.B postbyte xb	ASR Dd D2 IMM n=1	ASR Dd D2 postbyte xb	ASR Dd OPR.B IMM n=1	ASR Dd OPR.B postbyte xb	ASL Dd D2 IMM n=1	ASL Dd D2 postbyte xb	ASL Dd OPR.B IMM n=1	ASL Dd OPR.B postbyte xb
_1	LSR Dd D3 IMM n=1	LSR Dd D3 postbyte xb	LSR Dd OPR.W IMM n=1	LSR Dd OPR.W postbyte xb	LSL Dd D3 IMM n=1	LSL Dd D3 postbyte xb	LSL Dd OPR.W IMM n=1	LSL Dd OPR.W postbyte xb	ASR Dd D3 IMM n=1	ASR Dd D3 postbyte xb	ASR Dd OPR.W IMM n=1	ASR Dd OPR.W postbyte xb	ASL Dd D3 IMM n=1	ASL Dd D3 postbyte xb	ASL Dd OPR.W IMM n=1	ASL Dd OPR.W postbyte xb
_2	LSR Dd D4 IMM n=1	LSR Dd D4 postbyte xb	LSR Dd OPR.P IMM n=1	LSR Dd OPR.P postbyte xb	LSL Dd D4 IMM n=1	LSL Dd D4 postbyte xb	LSL Dd OPR.P IMM n=1	LSL Dd OPR.P postbyte xb	ASR Dd D4 IMM n=1	ASR Dd D4 postbyte xb	ASR Dd OPR.P IMM n=1	ASR Dd OPR.P postbyte xb	ASL Dd D4 IMM n=1	ASL Dd D4 postbyte xb	ASL Dd OPR.P IMM n=1	ASL Dd OPR.P postbyte xb
_3	LSR Dd D5 IMM n=1	LSR Dd D5 postbyte xb	LSR Dd OPR.L IMM n=1	LSR Dd OPR.L postbyte xb	LSL Dd D5 IMM n=1	LSL Dd D5 postbyte xb	LSL Dd OPR.L IMM n=1	LSL Dd OPR.L postbyte xb	ASR Dd D5 IMM n=1	ASR Dd D5 postbyte xb	ASR Dd OPR.L IMM n=1	ASR Dd OPR.L postbyte xb	ASL Dd D5 IMM n=1	ASL Dd D5 postbyte xb	ASL Dd OPR.L IMM n=1	ASL Dd OPR.L postbyte xb
_4	LSR Dd D0 IMM n=1	LSR Dd D0 postbyte xb	ROR OPR.B n=1	LSR OPR.B n=1	LSL Dd D0 IMM n=1	LSL Dd D0 postbyte xb	ROL OPR.B n=1	LSL OPR.B n=1	ASR Dd D0 IMM n=1	ASR Dd D0 postbyte xb	ROR OPR.B n=1	ASR OPR.B n=1	ASL Dd D0 IMM n=1	ASL Dd D0 postbyte xb	ROL OPR.B n=1	ASL OPR.B n=1
_5	LSR Dd D1 IMM n=1	LSR Dd D1 postbyte xb	ROR OPR.W n=1	LSR OPR.W n=1	LSL Dd D1 IMM n=1	LSL Dd D1 postbyte xb	ROL OPR.W n=1	LSL OPR.W n=1	ASR Dd D1 IMM n=1	ASR Dd D1 postbyte xb	ROR OPR.W n=1	ASR OPR.W n=1	ASL Dd D1 IMM n=1	ASL Dd D1 postbyte xb	ROL OPR.W n=1	ASL OPR.W n=1
_6	LSR Dd D6 IMM n=1	LSR Dd D6 postbyte xb	ROR OPR.P n=1	LSR OPR.P n=1	LSL Dd D6 IMM n=1	LSL Dd D6 postbyte xb	ROL OPR.P n=1	LSL OPR.P n=1	ASR Dd D6 IMM n=1	ASR Dd D6 postbyte xb	ROR OPR.P n=1	ASR OPR.P n=1	ASL Dd D6 IMM n=1	ASL Dd D6 postbyte xb	ROL OPR.P n=1	ASL OPR.P n=1
_7	LSR Dd D7 IMM n=1	LSR Dd D7 postbyte xb	ROR OPR.L n=1	LSR OPR.L n=1	LSL Dd D7 IMM n=1	LSL Dd D7 postbyte xb	ROL OPR.L n=1	LSL OPR.L n=1	ASR Dd D7 IMM n=1	ASR Dd D7 postbyte xb	ROR OPR.L n=1	ASR OPR.L n=1	ASL Dd D7 IMM n=1	ASL Dd D7 postbyte xb	ROL OPR.L n=1	ASL OPR.L n=1
_8	LSR Dd D2 IMM n=2	LSR Dd D2 postbyte xb	LSR Dd OPR.B IMM n=2	LSR Dd OPR.B postbyte xb	LSL Dd D2 IMM n=2	LSL Dd D2 postbyte xb	LSL Dd OPR.B IMM n=2	LSL Dd OPR.B postbyte xb	ASR Dd D2 IMM n=2	ASR Dd D2 postbyte xb	ASR Dd OPR.B IMM n=2	ASR Dd OPR.B postbyte xb	ASL Dd D2 IMM n=2	ASL Dd D2 postbyte xb	ASL Dd OPR.B IMM n=2	ASL Dd OPR.B postbyte xb
_9	LSR Dd D3 IMM n=2	LSR Dd D3 postbyte xb	LSR Dd OPR.W IMM n=2	LSR Dd OPR.W postbyte xb	LSL Dd D3 IMM n=2	LSL Dd D3 postbyte xb	LSL Dd OPR.W IMM n=2	LSL Dd OPR.W postbyte xb	ASR Dd D3 IMM n=2	ASR Dd D3 postbyte xb	ASR Dd OPR.W IMM n=2	ASR Dd OPR.W postbyte xb	ASL Dd D3 IMM n=2	ASL Dd D3 postbyte xb	ASL Dd OPR.W IMM n=2	ASL Dd OPR.W postbyte xb
_A	LSR Dd D4 IMM n=2	LSR Dd D4 postbyte xb	LSR Dd OPR.P IMM n=2	LSR Dd OPR.P postbyte xb	LSL Dd D4 IMM n=2	LSL Dd D4 postbyte xb	LSL Dd OPR.P IMM n=2	LSL Dd OPR.P postbyte xb	ASR Dd D4 IMM n=2	ASR Dd D4 postbyte xb	ASR Dd OPR.P IMM n=2	ASR Dd OPR.P postbyte xb	ASL Dd D4 IMM n=2	ASL Dd D4 postbyte xb	ASL Dd OPR.P IMM n=2	ASL Dd OPR.P postbyte xb
_B	LSR Dd D5 IMM n=2	LSR Dd D5 postbyte xb	LSR Dd OPR.L IMM n=2	LSR Dd OPR.L postbyte xb	LSL Dd D5 IMM n=2	LSL Dd D5 postbyte xb	LSL Dd OPR.L IMM n=2	LSL Dd OPR.L postbyte xb	ASR Dd D5 IMM n=2	ASR Dd D5 postbyte xb	ASR Dd OPR.L IMM n=2	ASR Dd OPR.L postbyte xb	ASL Dd D5 IMM n=2	ASL Dd D5 postbyte xb	ASL Dd OPR.L IMM n=2	ASL Dd OPR.L postbyte xb
_C	LSR Dd D0 IMM n=2	LSR Dd D0 postbyte xb	ROR OPR.B n=1	LSR OPR.B n=2	LSL Dd D0 IMM n=2	LSL Dd D0 postbyte xb	ROL OPR.B n=1	LSL OPR.B n=2	ASR Dd D0 IMM n=2	ASR Dd D0 postbyte xb	ROR OPR.B n=1	ASR OPR.B n=2	ASL Dd D0 IMM n=2	ASL Dd D0 postbyte xb	ROL OPR.B n=1	ASL OPR.B n=2
_D	LSR Dd D1 IMM n=2	LSR Dd D1 postbyte xb	ROR OPR.W n=1	LSR OPR.W n=2	LSL Dd D1 IMM n=2	LSL Dd D1 postbyte xb	ROL OPR.W n=1	LSL OPR.W n=2	ASR Dd D1 IMM n=2	ASR Dd D1 postbyte xb	ROR OPR.W n=1	ASR OPR.W n=2	ASL Dd D1 IMM n=2	ASL Dd D1 postbyte xb	ROL OPR.W n=1	ASL OPR.W n=2
_E	LSR Dd D6 IMM n=2	LSR Dd D6 postbyte xb	ROR OPR.P n=1	LSR OPR.P n=2	LSL Dd D6 IMM n=2	LSL Dd D6 postbyte xb	ROL OPR.P n=1	LSL OPR.P n=2	ASR Dd D6 IMM n=2	ASR Dd D6 postbyte xb	ROR OPR.P n=1	ASR OPR.P n=2	ASL Dd D6 IMM n=2	ASL Dd D6 postbyte xb	ROL OPR.P n=1	ASL OPR.P n=2
_F	LSR Dd D7 IMM n=2	LSR Dd D7 postbyte xb	ROR OPR.L n=1	LSR OPR.L n=2	LSL Dd D7 IMM n=2	LSL Dd D7 postbyte xb	ROL OPR.L n=1	LSL OPR.L n=2	ASR Dd D7 IMM n=2	ASR Dd D7 postbyte xb	ROR OPR.L n=1	ASR OPR.L n=2	ASL Dd D7 IMM n=2	ASL Dd D7 postbyte xb	ROL OPR.L n=1	ASL OPR.L n=2

A.4.5 Bit Manipulation Postbyte (bm)

The (bm) postbyte is for bit instructions where the operand is a register Di or an 8-bit byte, 16-bit word, or 32-bit long word in memory. The bit number to be changed or tested is specified in an immediate value (coded in the postbyte) or in a register Dn. Shaded codes in the coding map are reserved for future use.

The 3-bit codes for Di[2:0] and Dn[2:0] select 1-of-8 CPU data registers 0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7. Size[1:0] specifies the size of a memory operand where 0:0=byte, 0:1=16-bit word, and 1:1=32-bit long word.

Table A-11. Bit Manipulation Postbyte (bm) Decode

bm postbyte bitwise encoding								Comments
b7	b6	b5	b4	b3	b2	b1	b0	
0	0	n[2:0]			Di[2:0]			bit n (0-7) in 8-bit register D0 or D1 (Di[2:0] = 1:0:0 or 1:0:1)
0	1	x	x	x	1	0	x	reserved; like above but d6 is don't care and acts as b6=0
0	n[3:0]				Di[2:0]			bit n (0-15) in 16-bit register D2, D3, D4, or D5 (Di[2:0] = 0:0:0, 0:0:1, 0:1:0, or 0:1:1)
n[4:0]					Di[2:0]			bit n (0-31) in 32-bit register D6 or D7 (Di[2:0] = 1:1:0 or 1:1:1)
1	n[2:0]			0	0	0	0	bit n (0-7) in 8-bit memory operand OPR.B
1	n[2:0]			0	0	1	n[3]	bit n (0-15) in 16-bit memory operand OPR.W
1	n[2:0]			1	0	n[4:3]		bit n (0-31) in 32-bit memory operand OPR.L
1	Dn[2:0]			size[1:0]		0	1	Operand in memory (size[1:0]= byte-0:0, word-0:1, long-1:1); n (in Dn) = bit number
1	x	x	x	x	1	0	0	reserved; like above but d0 is don't care and acts like d0=1

Table A-12. Bit Manipulation Postbyte (bm) Coding Map

	0_	1_	2_	3_	4_	5_	6_	7_	8_	9_	A_	B_	C_	D_	E_	F_
_0	D2, n=0	D2, n=2	D2, n=4	D2, n=6	D2, n=8	D2, n=10	D2, n=12	D2, n=14	OPR.B, n=0	OPR.B, n=1	OPR.B, n=2	OPR.B, n=3	OPR.B, n=4	OPR.B, n=5	OPR.B, n=6	OPR.B, n=7
_1	D3, n=0	D3, n=2	D3, n=4	D3, n=6	D3, n=8	D3, n=10	D3, n=12	D3, n=14	OPR.B, n=D2	OPR.B, n=D3	OPR.B, n=D4	OPR.B, n=D5	OPR.B, n=D0	OPR.B, n=D1	OPR.B, n=D6	OPR.B, n=D7
_2	D4, n=0	D4, n=2	D4, n=4	D4, n=6	D4, n=8	D4, n=10	D4, n=12	D4, n=14	OPR.W, n=0	OPR.W, n=1	OPR.W, n=2	OPR.W, n=3	OPR.W, n=4	OPR.W, n=5	OPR.W, n=6	OPR.W, n=7
_3	D5, n=0	D5, n=2	D5, n=4	D5, n=6	D5, n=8	D5, n=10	D5, n=12	D5, n=14	OPR.W, n=8	OPR.W, n=9	OPR.W, n=10	OPR.W, n=11	OPR.W, n=12	OPR.W, n=13	OPR.W, n=14	OPR.W, n=15
_4	D0, n=0	D0, n=2	D0, n=4	D0, n=6	D0, n=0	D0, n=2	D0, n=4	D0, n=6	OPR.W, n=D2	OPR.W, n=D3	OPR.W, n=D4	OPR.W, n=D5	OPR.W, n=D0	OPR.W, n=D1	OPR.W, n=D6	OPR.W, n=D7
_5	D1, n=0	D1, n=2	D1, n=4	D1, n=6	D1, n=0	D1, n=2	D1, n=4	D1, n=6	OPR.W, n=D2	OPR.W, n=D3	OPR.W, n=D4	OPR.W, n=D5	OPR.W, n=D0	OPR.W, n=D1	OPR.W, n=D6	OPR.W, n=D7
_6	D6, n=0	D6, n=2	D6, n=4	D6, n=6	D6, n=8	D6, n=10	D6, n=12	D6, n=14	D6, n=16	D6, n=18	D6, n=20	D6, n=22	D6, n=24	D6, n=26	D6, n=28	D6, n=30
_7	D7, n=0	D7, n=2	D7, n=4	D7, n=6	D7, n=8	D7, n=10	D7, n=12	D7, n=14	D7, n=16	D7, n=18	D7, n=20	D7, n=22	D7, n=24	D7, n=26	D7, n=28	D7, n=30
_8	D2, n=1	D2, n=3	D2, n=5	D2, n=7	D2, n=9	D2, n=11	D2, n=13	D2, n=15	OPR.L, n=0	OPR.L, n=1	OPR.L, n=2	OPR.L, n=3	OPR.L, n=4	OPR.L, n=5	OPR.L, n=6	OPR.L, n=7
_9	D3, n=1	D3, n=3	D3, n=5	D3, n=7	D3, n=9	D3, n=11	D3, n=13	D3, n=15	OPR.L, n=8	OPR.L, n=9	OPR.L, n=10	OPR.L, n=11	OPR.L, n=12	OPR.L, n=13	OPR.L, n=14	OPR.L, n=15
_A	D4, n=1	D4, n=3	D4, n=5	D4, n=7	D4, n=9	D4, n=11	D4, n=13	D4, n=15	OPR.L, n=16	OPR.L, n=17	OPR.L, n=18	OPR.L, n=19	OPR.L, n=20	OPR.L, n=21	OPR.L, n=22	OPR.L, n=23
_B	D5, n=1	D5, n=3	D5, n=5	D5, n=7	D5, n=9	D5, n=11	D5, n=13	D5, n=15	OPR.L, n=24	OPR.L, n=25	OPR.L, n=26	OPR.L, n=27	OPR.L, n=28	OPR.L, n=29	OPR.L, n=30	OPR.L, n=31
_C	D0, n=1	D0, n=3	D0, n=5	D0, n=7	D0, n=1	D0, n=3	D0, n=5	D0, n=7	OPR.L, n=D2	OPR.L, n=D3	OPR.L, n=D4	OPR.L, n=D5	OPR.L, n=D0	OPR.L, n=D1	OPR.L, n=D6	OPR.L, n=D7
_D	D1, n=1	D1, n=3	D1, n=5	D1, n=7	D1, n=1	D1, n=3	D1, n=5	D1, n=7	OPR.L, n=D2	OPR.L, n=D3	OPR.L, n=D4	OPR.L, n=D5	OPR.L, n=D0	OPR.L, n=D1	OPR.L, n=D6	OPR.L, n=D7
_E	D6, n=1	D6, n=3	D6, n=5	D6, n=7	D6, n=9	D6, n=11	D6, n=13	D6, n=15	D6, n=17	D6, n=19	D6, n=21	D6, n=23	D6, n=25	D6, n=27	D6, n=29	D6, n=31
_F	D7, n=1	D7, n=3	D7, n=5	D7, n=7	D7, n=9	D7, n=11	D7, n=13	D7, n=15	D7, n=17	D7, n=19	D7, n=21	D7, n=23	D7, n=25	D7, n=27	D7, n=29	D7, n=31

A.4.6 Bitfield Postbyte (bb) for BFEXT and BFINS

The BFEXT and BFINS instructions share 8 opcodes where the extract/insert property is controlled by a bit in the bb postbyte. Eight opcodes are used because three bits in the opcode select one of the eight CPU data registers for the destination or source operand.

The 3-bit code for Ds[2:0] selects 1-of-8 CPU data registers 0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D0, 1:0:1=D1, 1:1:0=D6, and 1:1:1=D7. The 2-bit code for Dp[1:0] selects 1-of-4 16-bit CPU data registers 0:0=D2, 0:1=D3, 1:0=D4, and 1:1=D5. Only 16-bit registers are allowed for Dp because the width and offset parameters take 10 bits. Size[1:0] specifies the size of a memory operand where 0:0=byte, 0:1=16-bit word, 1:0=24-bit pointer, and 1:1=32-bit long word. w[4:3] holds the two high-order bits of the 5-bit width parameter. The low 3 bits of w and the 5-bit offset are supplied in an additional byte of object code after the postbyte.

Table A-13. Bitfield Extract/Insert Postbyte (bb) Decode

bb postbyte bitwise encoding								Comments
b7	b6	b5	b4	b3	b2	b1	b0	
1 = Insert 0 = Extract	0	0	Ds[2:0]			Dp[1:0]		Dd in opcode[2:0]; Source in Ds; parameters w and o in low 10 bits of 16-bit Dp
	0	1	Ds[2:0]			w[4:3]		Dd in opcode[2:0]; Source in Ds; parameter w[4:3] in postbyte, w[2:0], o[4:0] in extension byte `i1`
	1	0	0	size[1:0]		Dp[1:0]		Dd in opcode[2:0]; Source in memory; parameters w and o in low 10 bits of 16-bit Dp
	1	0	1	size[1:0]		Dp[1:0]		Destination in memory; Source in opcode[2:0]; parameters w and o in low 10 bits of 16-bit Dp
	1	1	0	size[1:0]		w[4:3]		Dd in opcode[2:0]; Source in memory; parameter w[4:3] in postbyte, w[2:0], o[4:0] in extension byte `i1`
	1	1	1	size[1:0]		w[4:3]		Destination in memory; Source in opcode[2:0]; parameters w[4:3] in postbyte, w[2:0], o[4:0] in extension byte `i1`

Table A-14. Bitfield Extract/Insert Postbyte (bb) Coding Map

	Bitfield Extract								Bitfield Insert							
	0_	1_	2_	3_	4_	5_	6_	7_	8_	9_	A_	B_	C_	D_	E_	F_
_0	BFEXT Dd D2,D2	BFEXT Dd D0,D2	BFEXT Dd D2,#w:o	BFEXT Dd D0,#w:o	BFEXT Dd OPR.B,D2	BFEXT Ds OPR.B,D2	BFEXT Dd OPR.B,#w:o	BFEXT Ds OPR.B,#w:o	BFINS Dd D2,D2	BFINS Dd D0,D2	BFINS Dd D2,#w:o	BFINS Dd D0,#w:o	BFINS Dd OPR.B,D2	BFINS Ds OPR.B,D2	BFINS Dd OPR.B,#w:o	BFINS Ds OPR.B,#w:o
_1	BFEXT Dd D2,D3	BFEXT Dd D0,D3	BFEXT Dd D2,#w:o	BFEXT Dd D0,#w:o	BFEXT Dd OPR.B,D3	BFEXT Ds OPR.B,D3			BFINS Dd D2,D3	BFINS Dd D0,D3	BFINS Dd D2,#w:o	BFINS Dd D0,#w:o	BFINS Dd OPR.B,D3	BFINS Ds OPR.B,D3		
_2	BFEXT Dd D2,D4	BFEXT Dd D0,D4	BFEXT Dd D2,#w:o	BFEXT Dd D0,#w:o	BFEXT Dd OPR.B,D4	BFEXT Ds OPR.B,D4			BFINS Dd D2,D4	BFINS Dd D0,D4	BFINS Dd D2,#w:o	BFINS Dd D0,#w:o	BFINS Dd OPR.B,D4	BFINS Ds OPR.B,D4		
_3	BFEXT Dd D2,D5	BFEXT Dd D0,D5	BFEXT Dd D2,#w:o	BFEXT Dd D0,#w:o	BFEXT Dd OPR.B,D5	BFEXT Ds OPR.B,D5			BFINS Dd D2,D5	BFINS Dd D0,D5	BFINS Dd D2,#w:o	BFINS Dd D0,#w:o	BFINS Dd OPR.B,D5	BFINS Ds OPR.B,D5		
_4	BFEXT Dd D3,D2	BFEXT Dd D1,D2	BFEXT Dd D3,#w:o	BFEXT Dd D1,#w:o	BFEXT Dd OPR.W,D2	BFEXT Ds OPR.W,D2	BFEXT Dd OPR.W,#w:o	BFEXT Ds OPR.W,#w:o	BFINS Dd D3,D2	BFINS Dd D1,D2	BFINS Dd D3,#w:o	BFINS Dd D1,#w:o	BFINS Dd OPR.W,D2	BFINS Ds OPR.W,D2	BFINS Dd OPR.W,#w:o	BFINS Ds OPR.W,#w:o
_5	BFEXT Dd D3,D3	BFEXT Dd D1,D3	BFEXT Dd D3,#w:o	BFEXT Dd D1,#w:o	BFEXT Dd OPR.W,D3	BFEXT Ds OPR.W,D3			BFINS Dd D3,D3	BFINS Dd D1,D3	BFINS Dd D3,#w:o	BFINS Dd D1,#w:o	BFINS Dd OPR.W,D3	BFINS Ds OPR.W,D3		
_6	BFEXT Dd D3,D4	BFEXT Dd D1,D4	BFEXT Dd D3,#w:o	BFEXT Dd D1,#w:o	BFEXT Dd OPR.W,D4	BFEXT Ds OPR.W,D4			BFINS Dd D3,D4	BFINS Dd D1,D4	BFINS Dd D3,#w:o	BFINS Dd D1,#w:o	BFINS Dd OPR.W,D4	BFINS Ds OPR.W,D4		
_7	BFEXT Dd D3,D5	BFEXT Dd D1,D5	BFEXT Dd D3,#w:o	BFEXT Dd D1,#w:o	BFEXT Dd OPR.W,D5	BFEXT Ds OPR.W,D5			BFINS Dd D3,D5	BFINS Dd D1,D5	BFINS Dd D3,#w:o	BFINS Dd D1,#w:o	BFINS Dd OPR.W,D5	BFINS Ds OPR.W,D5		
_8	BFEXT Dd D4,D2	BFEXT Dd D6,D2	BFEXT Dd D4,#w:o	BFEXT Dd D6,#w:o	BFEXT Dd OPR.P,D2	BFEXT Ds OPR.P,D2	BFEXT Dd OPR.P,#w:o	BFEXT Ds OPR.P,#w:o	BFINS Dd D4,D2	BFINS Dd D6,D2	BFINS Dd D4,#w:o	BFINS Dd D6,#w:o	BFINS Dd OPR.P,D2	BFINS Ds OPR.P,D2	BFINS Dd OPR.P,#w:o	BFINS Ds OPR.P,#w:o
_9	BFEXT Dd D4,D3	BFEXT Dd D6,D3	BFEXT Dd D4,#w:o	BFEXT Dd D6,#w:o	BFEXT Dd OPR.P,D3	BFEXT Ds OPR.P,D3			BFINS Dd D4,D3	BFINS Dd D6,D3	BFINS Dd D4,#w:o	BFINS Dd D6,#w:o	BFINS Dd OPR.P,D3	BFINS Ds OPR.P,D3		
_A	BFEXT Dd D4,D4	BFEXT Dd D6,D4	BFEXT Dd D4,#w:o	BFEXT Dd D6,#w:o	BFEXT Dd OPR.P,D4	BFEXT Ds OPR.P,D4			BFINS Dd D4,D4	BFINS Dd D6,D4	BFINS Dd D4,#w:o	BFINS Dd D6,#w:o	BFINS Dd OPR.P,D4	BFINS Ds OPR.P,D4		
_B	BFEXT Dd D4,D5	BFEXT Dd D6,D5	BFEXT Dd D4,#w:o	BFEXT Dd D6,#w:o	BFEXT Dd OPR.P,D5	BFEXT Ds OPR.P,D5			BFINS Dd D4,D5	BFINS Dd D6,D5	BFINS Dd D4,#w:o	BFINS Dd D6,#w:o	BFINS Dd OPR.P,D5	BFINS Ds OPR.P,D5		
_C	BFEXT Dd D5,D2	BFEXT Dd D7,D2	BFEXT Dd D5,#w:o	BFEXT Dd D7,#w:o	BFEXT Dd OPR.L,D2	BFEXT Ds OPR.L,D2	BFEXT Dd OPR.L,#w:o	BFEXT Ds OPR.L,#w:o	BFINS Dd D5,D2	BFINS Dd D7,D2	BFINS Dd D5,#w:o	BFINS Dd D7,#w:o	BFINS Dd OPR.L,D2	BFINS Ds OPR.L,D2	BFINS Dd OPR.L,#w:o	BFINS Ds OPR.L,#w:o
_D	BFEXT Dd D5,D3	BFEXT Dd D7,D3	BFEXT Dd D5,#w:o	BFEXT Dd D7,#w:o	BFEXT Dd OPR.L,D3	BFEXT Ds OPR.L,D3			BFINS Dd D5,D3	BFINS Dd D7,D3	BFINS Dd D5,#w:o	BFINS Dd D7,#w:o	BFINS Dd OPR.L,D3	BFINS Ds OPR.L,D3		
_E	BFEXT Dd D5,D4	BFEXT Dd D7,D4	BFEXT Dd D5,#w:o	BFEXT Dd D7,#w:o	BFEXT Dd OPR.L,D4	BFEXT Ds OPR.L,D4			BFINS Dd D5,D4	BFINS Dd D7,D4	BFINS Dd D5,#w:o	BFINS Dd D7,#w:o	BFINS Dd OPR.L,D4	BFINS Ds OPR.L,D4		
_F	BFEXT Dd D5,D5	BFEXT Dd D7,D5	BFEXT Dd D5,#w:o	BFEXT Dd D7,#w:o	BFEXT Dd OPR.L,D5	BFEXT Ds OPR.L,D5			BFINS Dd D5,D5	BFINS Dd D7,D5	BFINS Dd D5,#w:o	BFINS Dd D7,#w:o	BFINS Dd OPR.L,D5	BFINS Ds OPR.L,D5		

A.4.7 Transfer and Exchange Postbytes (tb) and (eb)

Although transfer and exchange use the same postbyte mapping, they have separate opcodes and there are subtle effects when registers of different width are involved in the transfer or exchange. Separate coding maps show these effects in the cells of the coding maps.

Table A-15. Transfer and Exchange Postbyte (eb) and (tb) Decode

eb and tb postbyte bitwise encoding		Comments
b7 b6 b5 b4	b3 b2 b1 b0	
SOURCE[3:0]	DEST[3:0]	Refer to coding maps for exchange and transfer to see how the registers are assigned to 4-bit codes

Refer to the exchange and sign-extend coding map below. When the source register is narrower than the destination register, the smaller source register is sign-extended as it is copied into the larger destination register and the source register is unchanged. When the source register is wider than the destination register, the narrower register is sign-extended as it is transferred into the wider register and the wider register is truncated during the transfer into the narrower register. These are not considered useful operations, this description simply documents what would happen if these unexpected combinations occur.

The two special cases EXG CCW,CCL and EXG CCW,CCH are ambiguous so CCW is not changed (this is equivalent to a NOP instruction).

Refer to the transfer coding map below. When the source register is narrower than the destination register, the smaller source register is zero-extended as it is transferred into the wider destination register. When the source register is wider than the destination register, the lower portion of the source register is transferred to the destination register.

Table A-16. Exchange and Sign-Extend Postbyte (eb) Coding Map

source		D2	D3	D4	D5	D0	D1	D6	D7	X	Y	S	–	CCH	CCL	CCW	
destination		0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
D2	-0	–	D3 ⇔ D2	D4 ⇔ D2	D5 ⇔ D2	sex:D0 ⇒ D2	sex:D1 ⇒ D2	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		sex:CCH ⇒ D2	sex:CCL ⇒ D2	CCW ⇔ D2	
D3	-1	D2 ⇔ D3	–	D4 ⇔ D3	D5 ⇔ D3	sex:D0 ⇒ D3	sex:D1 ⇒ D3	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		sex:CCH ⇒ D3	sex:CCL ⇒ D3	CCW ⇔ D3	
D4	-2	D2 ⇔ D4	D3 ⇔ D4	–	D5 ⇔ D4	sex:D0 ⇒ D4	sex:D1 ⇒ D4	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		sex:CCH ⇒ D4	sex:CCL ⇒ D4	CCW ⇔ D4	
D5	-3	D2 ⇔ D5	D3 ⇔ D5	D4 ⇔ D5	–	sex:D0 ⇒ D5	sex:D1 ⇒ D5	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		sex:CCH ⇒ D5	sex:CCL ⇒ D5	CCW ⇔ D5	
D0	-4	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	–	D1 ⇔ D0	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		CCH ⇔ D0	CCL ⇔ D0	Big ⇔Small	
D1	-5	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	D0 ⇔ D1	–	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		CCH ⇔ D1	CCL ⇔ D1	Big ⇔Small	
D6	-6	sex:D2 ⇒ D6	sex:D3 ⇒ D6	sex:D4 ⇒ D6	sex:D5 ⇒ D6	sex:D0 ⇒ D6	sex:D1 ⇒ D6	–	D7 ⇔ D6	sex:X ⇒ D6	sex:Y ⇒ D6	sex:S ⇒ D6		sex:CCH ⇒ D6	sex:CCL ⇒ D6	sex:CCW ⇒ D6	
D7	-7	sex:D2 ⇒ D7	sex:D3 ⇒ D7	sex:D4 ⇒ D7	sex:D5 ⇒ D7	sex:D0 ⇒ D7	sex:D1 ⇒ D7	D6 ⇔ D7	–	sex:X ⇒ D7	sex:Y ⇒ D7	sex:S ⇒ D7		sex:CCH ⇒ D7	sex:CCL ⇒ D7	sex:CCW ⇒ D7	
X	-8	sex:D2 ⇒ X	sex:D3 ⇒ X	sex:D4 ⇒ X	sex:D5 ⇒ X	sex:D0 ⇒ X	sex:D1 ⇒ X	Big ⇔Small	Big ⇔Small	–	Y ⇔ X	S ⇔ X		sex:CCH ⇒ X	sex:CCL ⇒ X	sex:CCW ⇒ X	
Y	-9	sex:D2 ⇒ Y	sex:D3 ⇒ Y	sex:D4 ⇒ Y	sex:D5 ⇒ Y	sex:D0 ⇒ Y	sex:D1 ⇒ Y	Big ⇔Small	Big ⇔Small	X ⇔ Y	–	S ⇔ Y		sex:CCH ⇒ Y	sex:CCL ⇒ Y	sex:CCW ⇒ Y	
S	-A	sex:D2 ⇒ S	sex:D3 ⇒ S	sex:D4 ⇒ S	sex:D5 ⇒ S	sex:D0 ⇒ S	sex:D1 ⇒ S	Big ⇔Small	Big ⇔Small	X ⇔ S	Y ⇔ S	–		sex:CCH ⇒ S	sex:CCL ⇒ S	sex:CCW ⇒ S	
reserved	-B																
CCH	-C	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	D0 ⇔ CCH	D1 ⇔ CCH	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		–	CCL ⇔ CCH	NOP	
CCL	-D	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	D0 ⇔ CCL	D1 ⇔ CCL	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		CCH ⇔ CCL	–	NOP	
CCW	-E	D2 ⇔ CCW	D3 ⇔ CCW	D4 ⇔ CCW	D5 ⇔ CCW	sex:D0 ⇒ CCW	sex:D1 ⇒ CCW	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small	Big ⇔Small		sex:CCH ⇒ CCW	sex:CCL ⇒ CCW	–	
	-F																–

EXG Big,Small: Small register gets low part of Big register, Big register gets sign-extended Small register. These cases are not expected to be useful in application programs.
EXG CCW,CCH and EXG CCW,CCL are ambiguous cases so CCW is not changed (equivalent to NOP)

Table A-17. Transfer Postbyte (tb) Coding Map

	source	D2	D3	D4	D5	D0	D1	D6	D7	X	Y	S	–	CCH	CCL	CCW	
destination		0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
D2	-0	–	D3 ⇒ D2	D4 ⇒ D2	D5 ⇒ D2	00:D0 ⇒ D2	00:D1 ⇒ D2	D6L ⇒ D2	D7L ⇒ D2	XL ⇒ D2	YL ⇒ D2	SL ⇒ D2		00:CCH ⇒ D2	00:CCL ⇒ D2	CCW ⇒ D2	
D3	-1	D2 ⇒ D3	–	D4 ⇒ D3	D5 ⇒ D3	00:D0 ⇒ D3	00:D1 ⇒ D3	D6L ⇒ D3	D7L ⇒ D3	XL ⇒ D3	YL ⇒ D3	SL ⇒ D3		00:CCH ⇒ D3	00:CCL ⇒ D3	CCW ⇒ D3	
D4	-2	D2 ⇒ D4	D3 ⇒ D4	–	D5 ⇒ D4	00:D0 ⇒ D4	00:D1 ⇒ D4	D6L ⇒ D4	D7L ⇒ D4	XL ⇒ D4	YL ⇒ D4	SL ⇒ D4		00:CCH ⇒ D4	00:CCL ⇒ D4	CCW ⇒ D4	
D5	-3	D2 ⇒ D5	D3 ⇒ D5	D4 ⇒ D5	–	00:D0 ⇒ D5	00:D1 ⇒ D5	D6L ⇒ D5	D7L ⇒ D5	XL ⇒ D5	YL ⇒ D5	SL ⇒ D5		00:CCH ⇒ D5	00:CCL ⇒ D5	CCW ⇒ D5	
D0	-4	D2L ⇒ D0	D3L ⇒ D0	D4L ⇒ D0	D5L ⇒ D0	–	D1 ⇒ D0	D6L ⇒ D0	D7L ⇒ D0	XL ⇒ D0	YL ⇒ D0	SL ⇒ D0		CCH ⇒ D0	CCL ⇒ D0	CCL ⇒ D0	
D1	-5	D2L ⇒ D1	D3L ⇒ D1	D4L ⇒ D1	D5L ⇒ D1	D0 ⇒ D1	–	D6L ⇒ D1	D7L ⇒ D1	XL ⇒ D1	YL ⇒ D1	SL ⇒ D1		CCH ⇒ D1	CCL ⇒ D1	CCL ⇒ D1	
D6	-6	0000:D2 ⇒ D6	0000:D3 ⇒ D6	0000:D4 ⇒ D6	0000:D5 ⇒ D6	000000:D0 ⇒ D6	000000:D1 ⇒ D6	–	D7 ⇒ D6	00:X ⇒ D6	00:Y ⇒ D6	00:S ⇒ D6		000000:CCH ⇒ D6	000000:CCL ⇒ D6	0000:CCW ⇒ D6	
D7	-7	0000:D2 ⇒ D7	0000:D3 ⇒ D7	0000:D4 ⇒ D7	0000:D5 ⇒ D7	000000:D0 ⇒ D7	000000:D1 ⇒ D7	D6 ⇒ D7	–	00:X ⇒ D7	00:Y ⇒ D7	00:S ⇒ D7		000000:CCH ⇒ D7	000000:CCL ⇒ D7	0000:CCW ⇒ D7	
X	-8	00:D2 ⇒ X	00:D3 ⇒ X	00:D4 ⇒ X	00:D5 ⇒ X	0000:D0 ⇒ X	0000:D1 ⇒ X	D6L ⇒ X	D7L ⇒ X	–	Y ⇒ X	S ⇒ X		0000:CCH ⇒ X	0000:CCL ⇒ X	00:CCW ⇒ X	
Y	-9	00:D2 ⇒ Y	00:D3 ⇒ Y	00:D4 ⇒ Y	00:D5 ⇒ Y	0000:D0 ⇒ Y	0000:D1 ⇒ Y	D6L ⇒ Y	D7L ⇒ Y	X ⇒ Y	–	S ⇒ Y		0000:CCH ⇒ Y	0000:CCL ⇒ Y	00:CCW ⇒ Y	
S	-A	00:D2 ⇒ S	00:D3 ⇒ S	00:D4 ⇒ S	00:D5 ⇒ S	0000:D0 ⇒ S	0000:D1 ⇒ S	D6L ⇒ S	D7L ⇒ S	X ⇒ S	Y ⇒ S	–		0000:CCH ⇒ S	0000:CCL ⇒ S	00:CCW ⇒ S	
reserved	-B																
CCH	-C	D2L ⇒ CCH	D3L ⇒ CCH	D4L ⇒ CCH	D5L ⇒ CCH	D0 ⇒ CCH	D1 ⇒ CCH	D6L ⇒ CCH	D7L ⇒ CCH	XL ⇒ CCH	YL ⇒ CCH	SL ⇒ CCH		–	CCL ⇒ CCH	CCL ⇒ CCH	
CCL	-D	D2L ⇒ CCL	D3L ⇒ CCL	D4L ⇒ CCL	D5L ⇒ CCL	D0 ⇒ CCL	D1 ⇒ CLL	D6L ⇒ CCL	D7L ⇒ CCL	XL ⇒ CCL	YL ⇒ CCL	SL ⇒ CCL		CCH ⇒ CCL	–	CCL ⇒ CCL	
CCW	-E	D2 ⇒ CCW	D3 ⇒ CCW	D4 ⇒ CCW	D5 ⇒ CCW	00:D0 ⇒ CCW	00:D1 ⇒ CCW	D6L ⇒ CCW	D7L ⇒ CCW	XL ⇒ CCW	YL ⇒ CCW	SL ⇒ CCW		00:CCH ⇒ CCW	00:CCL ⇒ CCW	–	
	-F																–

A.4.8 Count Leading Sign-Bits Postbyte (cb)

This is a variant of the transfer postbyte (tb) but limited to the 8 data-registers D0..D7.

Table A-18. Count Leading Sign-Bits (cb) Decode

eb and tb postbyte bitwise encoding								Comments
b7	b6	b5	b4	b3	b2	b1	b0	
0	SOURCE[2:0]			0	DEST[2:0]			Refer to coding map for Count Leading Sign-Bits to see how the registers are assigned to 3-bit codes

Refer to the Count Leading Sign-Bits coding map below (shaded fields are reserved).

Table A-19. Count Leading Sign-Bits (cb) Coding Map

	0_	1_	2_	3_	4_	5_	6_	7_	8_	9_	A_	B_	C_	D_	E_	F_
_0	D2,D2	D3,D2	D4,D2	D5,D2	D0,D2	D1,D2	D6,D2	D7,D2	D2,D2	D3,D2	D4,D2	D5,D2	D0,D2	D1,D2	D6,D2	D7,D2
_1	D2,D3	D3,D3	D4,D3	D5,D3	D0,D3	D1,D3	D6,D3	D7,D3	D2,D3	D3,D3	D4,D3	D5,D3	D0,D3	D1,D3	D6,D3	D7,D3
_2	D2,D4	D3,D4	D4,D4	D5,D4	D0,D4	D1,D4	D6,D4	D7,D4	D2,D4	D3,D4	D4,D4	D5,D4	D0,D4	D1,D4	D6,D4	D7,D4
_3	D2,D5	D3,D5	D4,D5	D5,D5	D0,D5	D1,D5	D6,D5	D7,D5	D2,D5	D3,D5	D4,D5	D5,D5	D0,D5	D1,D5	D6,D5	D7,D5
_4	D2,D0	D3,D0	D4,D0	D5,D0	D0,D0	D1,D0	D6,D0	D7,D0	D2,D0	D3,D0	D4,D0	D5,D0	D0,D0	D1,D0	D6,D0	D7,D0
_5	D2,D1	D3,D1	D4,D1	D5,D1	D0,D1	D1,D1	D6,D1	D7,D1	D2,D1	D3,D1	D4,D1	D5,D1	D0,D1	D1,D1	D6,D1	D7,D1
_6	D2,D6	D3,D6	D4,D6	D5,D6	D0,D6	D1,D6	D6,D6	D7,D6	D2,D6	D3,D6	D4,D6	D5,D6	D0,D6	D1,D6	D6,D6	D7,D6
_7	D2,D7	D3,D7	D4,D7	D5,D7	D0,D7	D1,D7	D6,D7	D7,D7	D2,D7	D3,D7	D4,D7	D5,D7	D0,D7	D1,D7	D6,D7	D7,D7
_8	D2,D2	D3,D2	D4,D2	D5,D2	D0,D2	D1,D2	D6,D2	D7,D2	D2,D2	D3,D2	D4,D2	D5,D2	D0,D2	D1,D2	D6,D2	D7,D2
_9	D2,D3	D3,D3	D4,D3	D5,D3	D0,D3	D1,D3	D6,D3	D7,D3	D2,D3	D3,D3	D4,D3	D5,D3	D0,D3	D1,D3	D6,D3	D7,D3
_A	D2,D4	D3,D4	D4,D4	D5,D4	D0,D4	D1,D4	D6,D4	D7,D4	D2,D4	D3,D4	D4,D4	D5,D4	D0,D4	D1,D4	D6,D4	D7,D4
_B	D2,D5	D3,D5	D4,D5	D5,D5	D0,D5	D1,D5	D6,D5	D7,D5	D2,D5	D3,D5	D4,D5	D5,D5	D0,D5	D1,D5	D6,D5	D7,D5
_C	D2,D0	D3,D0	D4,D0	D5,D0	D0,D0	D1,D0	D6,D0	D7,D0	D2,D0	D3,D0	D4,D0	D5,D0	D0,D0	D1,D0	D6,D0	D7,D0
_D	D2,D1	D3,D1	D4,D1	D5,D1	D0,D1	D1,D1	D6,D1	D7,D1	D2,D1	D3,D1	D4,D1	D5,D1	D0,D1	D1,D1	D6,D1	D7,D1
_E	D2,D6	D3,D6	D4,D6	D5,D6	D0,D6	D1,D6	D6,D6	D7,D6	D2,D6	D3,D6	D4,D6	D5,D6	D0,D6	D1,D6	D6,D6	D7,D6
_F	D2,D7	D3,D7	D4,D7	D5,D7	D0,D7	D1,D7	D6,D7	D7,D7	D2,D7	D3,D7	D4,D7	D5,D7	D0,D7	D1,D7	D6,D7	D7,D7

A.4.9 Push and Pull Postbyte (pb)

Push and pull instructions are used to store CPU registers onto the stack or read them from the stack, respectively. These instructions use an opcode and the pb postbyte to specify which registers to save onto or restore from the stack. Up to 6 registers may be specified in the register mask in the low six bits of the pb postbyte and there are two variations of this mask to allow a second group of CPU registers to pushed and pulled.

In the special case where the low-order six bits of the mask are all zero, it indicates that all 12 CPU registers (CCH, CCL, D0, D1, D2, D3, D4, D5, D6, D7, X, and Y) or all four 16-bit registers (D2, D3, D4, and D5) should be pushed or pulled as indicated by b[7:6] of postbyte pb.

Table A-20. Push/Pull Postbyte (pb) Decode

pb postbyte bitwise encoding								Comments
b7	b6	b5	b4	b3	b2	b1	b0	
PUSH/PULL	MASK2/1	Specify which registers to push/pull						
0	0	CCH	CCL	D0	D1	D2	D3	Push selected registers in register mask 1 list (pb = 0x00 = PSH ALL)
0	1	D4	D5	D6	D7	X	Y	Push selected registers in register mask 2 list (pb = 0x40 = PSH ALL16b)
1	0	CCH	CCL	D0	D1	D2	D3	Pull selected registers in register mask 1 list (pb = 0x80 = PUL ALL)
1	1	D4	D5	D6	D7	X	Y	Pull selected registers in register mask 2 list (pb = 0xC0 = PUL ALL16b)

YL	Y
YM	
YH	
XL	X
XM	
XH	

Push order is top to bottom (Higher memory addresses are at the top)
Pull order is bottom to top

If the mask bit corresponding to a register is 0, skip that register in the push or pull operation.

D7L	D7
D7ML	
D7MH	
D7H	
D6L	D6
D6ML	
D6MH	
D6H	
D5L	D5
D5H	
D4L	D4
D4H	
D3L	D3
D3H	
D2L	D2
D2H	
D1	D1
D0	D0
CCL	CCR
CCH	

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

<size-12>
Rev. 1.01, <size-12>