Table 0-1. Revision History

<table>
<thead>
<tr>
<th>Rev</th>
<th>Date</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>10 Oct 2012</td>
<td></td>
<td>Initial release.</td>
</tr>
<tr>
<td>1.01</td>
<td>24 Jan 2013</td>
<td></td>
<td>Fixed typos and grammar errors throughout the document.</td>
</tr>
</tbody>
</table>
Chapter 6
Instruction Glossary

6.1 Introduction .......................................................... 77
6.2 Glossary ................................................................. 77

ABS — Absolute Value ................................................. 78
ADC — Add with Carry .................................................. 79
ADD — Add without Carry ............................................. 81
AND — Bitwise AND ..................................................... 83
ANDCC — Bitwise AND CCL with Immediate ....................... 85
ASL — Arithmetic Shift Left ........................................... 86
ASR — Arithmetic Shift Right ......................................... 92
BCC — Branch if Carry Clear .......................................... 98
BCLR — Test and Clear Bit ............................................ 100
BCS — Branch if Carry Set ............................................ 103
BEQ — Branch if Equal .................................................. 105
BFEXT — Bit Field Extract ............................................ 107
BFINS — Bit Field Insert ................................................ 111
BGE — Branch if Greater Than or Equal .............................. 115
BGND — Enter Background Debug Mode ............................ 117
BGT — Branch if Greater Than ......................................... 118
BHI — Branch if Higher ................................................. 120
BHS — Branch if Higher or Same ...................................... 122
BIT — Bit Test .............................................................. 124
BLE — Branch if Less Than or Equal ................................ 126
BLO — Branch if Lower .................................................. 128
BLS — Branch if Lower or Same ....................................... 130
BLT — Branch if Less Than ............................................. 132
BMI — Branch if Minus ................................................. 134
BNE — Branch if Not Equal ............................................ 136
BPL — Branch if Plus .................................................... 138
BRA — Branch Always .................................................... 140
BRCLR — Test Bit and Branch if Clear ............................... 142
BRSET — Test Bit and Branch if Set ................................ 147
BSET — Test and Set Bit ................................................. 152
BSR — Branch to Subroutine ........................................... 155
BTGL — Test and Toggle Bit ............................................ 156
BVC — Branch if Overflow Clear ...................................... 159
BVS — Branch if Overflow Set .......................................... 161
CLB — Count Leading Sign-Bits ...................................... 163
<table>
<thead>
<tr>
<th>Opcode</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLC</td>
<td>Clear Carry</td>
<td>164</td>
</tr>
<tr>
<td>CLI</td>
<td>Clear Interrupt Mask</td>
<td>165</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear Memory, Register, or Index Register</td>
<td>166</td>
</tr>
<tr>
<td>CLV</td>
<td>Clear Overflow</td>
<td>168</td>
</tr>
<tr>
<td>CMP</td>
<td>Compare</td>
<td>169</td>
</tr>
<tr>
<td>COM</td>
<td>Complement Memory</td>
<td>173</td>
</tr>
<tr>
<td>DBcc</td>
<td>Decrement and Branch</td>
<td>175</td>
</tr>
<tr>
<td>DEC</td>
<td>Decrement</td>
<td>178</td>
</tr>
<tr>
<td>DIVS</td>
<td>Signed Divide</td>
<td>180</td>
</tr>
<tr>
<td>DIVU</td>
<td>Unsigned Divide</td>
<td>184</td>
</tr>
<tr>
<td>EOR</td>
<td>Exclusive OR</td>
<td>188</td>
</tr>
<tr>
<td>EXG</td>
<td>Exchange Register Contents</td>
<td>190</td>
</tr>
<tr>
<td>INC</td>
<td>Increment</td>
<td>192</td>
</tr>
<tr>
<td>JMP</td>
<td>Jump</td>
<td>194</td>
</tr>
<tr>
<td>JSR</td>
<td>Jump to Subroutine</td>
<td>196</td>
</tr>
<tr>
<td>LD</td>
<td>Load</td>
<td>198</td>
</tr>
<tr>
<td>LEA</td>
<td>Load Effective Address</td>
<td>202</td>
</tr>
<tr>
<td>LSL</td>
<td>Logical Shift Left</td>
<td>205</td>
</tr>
<tr>
<td>LSR</td>
<td>Logical Shift Right</td>
<td>211</td>
</tr>
<tr>
<td>MACS</td>
<td>Signed Multiply and Accumulate</td>
<td>217</td>
</tr>
<tr>
<td>MACU</td>
<td>Unsigned Multiply and Accumulate</td>
<td>221</td>
</tr>
<tr>
<td>MAXS</td>
<td>Maximum of Two Signed Values to Di</td>
<td>225</td>
</tr>
<tr>
<td>MAXU</td>
<td>Maximum of Two Unsigned Values to Di</td>
<td>227</td>
</tr>
<tr>
<td>MINS</td>
<td>Minimum of Two Signed Values to Di</td>
<td>229</td>
</tr>
<tr>
<td>MINU</td>
<td>Minimum of Two Unsigned Values to Di</td>
<td>231</td>
</tr>
<tr>
<td>MODS</td>
<td>Signed Modulo</td>
<td>233</td>
</tr>
<tr>
<td>MODU</td>
<td>Unsigned Modulo</td>
<td>237</td>
</tr>
<tr>
<td>MOV</td>
<td>Move Data</td>
<td>241</td>
</tr>
<tr>
<td>MULS</td>
<td>Signed Multiply</td>
<td>246</td>
</tr>
<tr>
<td>MULU</td>
<td>Unsigned Multiply</td>
<td>250</td>
</tr>
<tr>
<td>NEG</td>
<td>Two’s Complement Negate</td>
<td>254</td>
</tr>
<tr>
<td>NOP</td>
<td>Null Operation</td>
<td>256</td>
</tr>
<tr>
<td>OR</td>
<td>Bitwise OR</td>
<td>257</td>
</tr>
<tr>
<td>ORCC</td>
<td>Bitwise OR CCL with Immediate</td>
<td>259</td>
</tr>
<tr>
<td>PSH</td>
<td>Push Registers onto Stack</td>
<td>260</td>
</tr>
<tr>
<td>PUL</td>
<td>Pull Registers from Stack</td>
<td>262</td>
</tr>
<tr>
<td>QMULS</td>
<td>Signed Fractional Multiply</td>
<td>264</td>
</tr>
<tr>
<td>QMULU</td>
<td>Unsigned Fractional Multiply</td>
<td>269</td>
</tr>
<tr>
<td>ROL</td>
<td>Rotate Left Through Carry</td>
<td>274</td>
</tr>
<tr>
<td>ROR</td>
<td>Rotate Right Through Carry</td>
<td>276</td>
</tr>
<tr>
<td>RTI</td>
<td>Return from Interrupt</td>
<td>278</td>
</tr>
<tr>
<td>RTS</td>
<td>Return from Subroutine</td>
<td>279</td>
</tr>
<tr>
<td>SAT</td>
<td>Saturate</td>
<td>280</td>
</tr>
<tr>
<td>SBC</td>
<td>Subtract with Borrow</td>
<td>281</td>
</tr>
<tr>
<td>SEC</td>
<td>Set Carry Flag</td>
<td>283</td>
</tr>
<tr>
<td>SEI</td>
<td>Set Interrupt Mask</td>
<td>284</td>
</tr>
<tr>
<td>SEV</td>
<td>Set Overflow Flag</td>
<td>285</td>
</tr>
<tr>
<td>SEX</td>
<td>Sign-Extend</td>
<td>286</td>
</tr>
<tr>
<td>SPARE</td>
<td>Unimplemented Page1 Opcode Trap</td>
<td>288</td>
</tr>
</tbody>
</table>
Chapter 7
Exceptions

7.1 Introduction .......................................................... 307
7.2 Types of Exceptions .................................................. 307
7.3 Exception Priority ...................................................... 308
7.3.1 Reset ............................................................... 308
7.3.2 Software Exceptions ................................................. 309
7.3.2.1 Unimplemented Op-code Traps (SPARE, TRAP) ........ 309
7.3.2.2 Software Interrupt and System Call Instructions (SWI, SYS) ................................................. 309
7.3.3 Machine Exception .................................................. 309
7.3.4 X-bit-Maskable Interrupt Request (XIRQ) ...................... 310
7.3.5 I-bit-Maskable Interrupt Requests ................................ 310
7.3.6 Return-from-Interrupt Instruction (RTI) ....................... 310
7.4 Interrupt Recognition .................................................. 310
7.5 Exception Processing Flow ............................................. 311
7.5.1 Vector Fetch ....................................................... 311
7.5.2 Reset Exception Processing ......................................... 313
7.5.3 Interrupt and Unimplemented Opcode Trap Exception Processing .......................................................... 313

Chapter 8
Instruction Execution Timing

8.1 Introduction .......................................................... 315
8.2 Instruction Execution Timing .......................................... 315
8.2.1 No Operation Instruction Execution Times (NOP) ............ 315
8.2.2 Move Instruction Execution Times (MOV) ....................... 315
8.2.3 Load Instruction Execution Times (LD) ......................... 316
8.2.4 Store Instruction Execution Times (ST) ......................... 317
8.2.5 Push Register(s) onto Stack Instruction Execution Times (PSH) .................................................. 317
8.2.6 Pull Register(s) from Stack Instruction Execution Times (PUL) .................................................. 317
8.2.7 Load Effective Address Instruction Execution Times (LEA) .................................................. 318
8.2.8 Clear Instruction Execution Times (CLR) ....................... 318
8.2.9 Register-To-Register Transfer and Exchange Execution Times (TFR, EXG, SEX, ZEX) ............ 318
8.2.10 Logical AND/OR Instruction Execution Times (AND, OR, BIT, EOR) .................................. 319
8.2.11 One’s Complement (Invert) Instruction Execution Times (COM) .................................................. 320
8.2.12 Increment and Decrement Instruction Execution Times (INC, DEC) .............................................. 320
8.2.13 Add and Subtract Instruction Execution Times (ADD, ADC, SUB, SBC, CMP) ...................... 321
8.2.14 Two’s Complement (Negate) Instruction Execution Times (NEG) .................................................. 321
Chapter 9
Data Bus Operation

9.1 Introduction .................................................... 343
9.2 Access Timing .................................................... 343
9.3 Data Transfer Alignment ...................................... 343

Appendix A
Instruction Reference

A.1 Introduction ...................................................... 345
A.2 S12Z Instruction Set Summary Table .......................... 346
A.3 S12Z Opcode Map ................................................ 363
A.4 Postbyte Coding ............................................... 365
A.4.1 General Operand (OPR) Addressing Postbyte (xb) ....... 365
A.4.2 Math Postbyte (mb) for MUL, MAC, DIV, MOD and QMUL 366
A.4.3 Loop Primitive Postbyte (lb) ................................. 368
A.4.4 Shift and Rotate Postbyte (sb) ............................... 368
A.4.5 Bit Manipulation Postbyte (bm) ............................. 370
A.4.6 Bitfield Postbyte (bb) for BFEXT and BFINS .......... 371
A.4.7 Transfer and Exchange Postbytes (tb) and (eb) ........ 373

Linear S12 Core Reference Manual, Rev. 1.01
Freescale Semiconductor
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** — 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).

- **opr5i** — Any label or expression that evaluates to an 8-bit immediate operand.

- **opr8i** — Any label or expression that evaluates to a 16-bit immediate operand.

- **opr16i** — 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.

- **opr1msz** — 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).
1.3.2 Operators

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).

[Di,xy] — Register offset from X or Y Indirect. D2, D3, D4, D5 treated as signed, D0, D1, D6, D7 treated as 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. opru18 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.

(opru24,xysp) — 24-bit offset from X, Y, S, or P. opru24 is any label or expression that evaluates to a 24-bit value (16M).

[opru24,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.

opru24 — Long Extended (16M). Reaches any address in the full 16M memory space.

[opru24] — 24-bit address Indirect.

opregs1 — 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.

opregs2 — 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
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:D6ML:D6L, and D7H:D7MH:D7ML:D7L because it is important to show the order that bytes are used. In all cases except Dp you may substitute any of the register numbers 0–7 in place of the subscript. In the case of Dp you are limited to the four 16-bit registers because the register is used for a 10-bit parameter.

- D0 — Any of the eight CPU data registers D0–D7.
- D1 — Any of the eight CPU data registers D0–D7. Used for a first operand.
- D2 — Any of the eight CPU data registers D0–D7. Used for a second operand.
- D3 — Any of the eight CPU data registers D0–D7. Used for a third operand.
- D4 — Any of the eight CPU data registers D0–D7. Used to specify an instruction parameter such as a number n or a number of bit positions for a shift.
- D5 — Any of the eight CPU data registers D0–D7. Used for a source operand.
- D6 — Any of the eight CPU data registers D0–D7. Used for a destination operand.
- D7 — 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.
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.

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 Di. 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.

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 (Di). 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 Di Indexed submode of general OPR addressing — A CPU data register (Di) is used as an index register in this indexed addressing mode variation.
  - **IDX3,REG** — 24b Offset from Di Indexed submode of general OPR addressing — A CPU data registers (Di) 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 (Di) 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<sub>i</sub> 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<sub>i</sub> 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...

```assembly
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 Di (IDX2,REG*)

An 18-bit unsigned constant is added to a CPU registers Di 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 Di (IDX3,REG*)

A 24-bit constant is added to a CPU registers Di 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 Di 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<sub>i</sub>), 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<sub>i</sub>.

#### 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<sub>i</sub>, 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.
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

<table>
<thead>
<tr>
<th>PC[1:0]</th>
<th>Minimum amount of bus-cycles required after Change-of-Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
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:
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
Chapter 5 Instruction Set Overview

— 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)

<table>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Load</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD Di,#opr24mz</td>
<td>Load Di from Memory</td>
<td>(M) (\Rightarrow) Di</td>
</tr>
<tr>
<td>LD Di,opr24a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD Di,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD xy,#opr18i</td>
<td>Load index register X or Y from Memory</td>
<td>(M:M+1:M+2) (\Rightarrow) X or Y</td>
</tr>
<tr>
<td>LD xy,opr24a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD xy,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD S,#opr24i</td>
<td>Load stack pointer SP from Memory</td>
<td>(M:M+1:M+2) (\Rightarrow) SP</td>
</tr>
<tr>
<td>LD S,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Pull (load from stack)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUL oprregs1</td>
<td>Pull specified CPU registers from Stack</td>
<td>(M(SP)–M(SP+n-1)) (\Rightarrow) regs; (SP) + n (\Rightarrow) SP</td>
</tr>
<tr>
<td>PUL oprregs2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUL ALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUL ALL16b</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Load Effective Address</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEA D6,oprmemreg</td>
<td>Load Effective Address into 32-bit D6 or D7</td>
<td>00:Effective Address (\Rightarrow) D6, or 00:Effective Address (\Rightarrow) D7</td>
</tr>
<tr>
<td>LEA D7,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEA S,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEA X,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEA Y,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Store</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST Di,opr24a</td>
<td>Store Di to Memory</td>
<td>(Di) (\Rightarrow) M</td>
</tr>
<tr>
<td>ST Di,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST xy,opr24a</td>
<td>Store index register X or Y to Memory</td>
<td>(X) (\Rightarrow) (M:M+1:M+2), or (Y) (\Rightarrow) (M:M+1:M+2)</td>
</tr>
<tr>
<td>ST xy,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST S,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Push (store to stack)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSH oprregs1</td>
<td>Push specified CPU registers onto Stack</td>
<td>(SP) – n (\Rightarrow) SP; (regs) (\Rightarrow) M(SP)–M(SP+n-1)</td>
</tr>
<tr>
<td>PSH oprregs2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSH ALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSH ALL16b</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Move (memory-to-memory; byte, word, pointer, or long-word)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV.B #opr8i,oprmemreg</td>
<td>Move Immediate to Memory MD, 8-bit operand</td>
<td># (\Rightarrow) MD</td>
</tr>
<tr>
<td>MOV.B oprmemreg,oprmemreg</td>
<td>Move memory to memory, 8-bit operand</td>
<td>(MS) (\Rightarrow) MD</td>
</tr>
<tr>
<td>MOV.W #opr16i,oprmemreg</td>
<td>Move Immediate to Memory MD, 16-bit operand</td>
<td># (\Rightarrow) MD</td>
</tr>
</tbody>
</table>
### Table 5-1. Load and Store Instructions (Sheet 2 of 3)

<table>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV.W oprmemreg,oprmemreg</td>
<td>Move memory to memory, 16-bit operand</td>
<td>(MS) ⇒ MD</td>
</tr>
<tr>
<td>MOV.P #opr24i,oprmemreg</td>
<td>Move Immediate to Memory MD, 24-bit operand</td>
<td># ⇒ MD</td>
</tr>
<tr>
<td>MOV.P oprmemreg,oprmemreg</td>
<td>Move memory to memory, 24-bit operand</td>
<td>(MS) ⇒ MD</td>
</tr>
<tr>
<td>MOV.L #opr32i,oprmemreg</td>
<td>Move Immediate to Memory MD, 32-bit operand</td>
<td># ⇒ MD</td>
</tr>
<tr>
<td>MOV.L oprmemreg,oprmemreg</td>
<td>Move memory to memory, 32-bit operand</td>
<td>(MS) ⇒ MD</td>
</tr>
</tbody>
</table>

**Transfer and Exchange**

<table>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFR cpureg,cpureg</td>
<td>Transfer CPU Register r1 to r2</td>
<td>(r1) ⇒ (r2)</td>
</tr>
<tr>
<td></td>
<td>D0–D7, X, Y, SP, CCH, CCL, or CCW</td>
<td></td>
</tr>
<tr>
<td></td>
<td>if same size, direct transfer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>if 1st smaller than 2nd, zero-extend 1st to 2nd</td>
<td></td>
</tr>
<tr>
<td>EXG cpureg,cpureg</td>
<td>Exchange contents of CPU Registers</td>
<td>(r1) ⇔ (r2)</td>
</tr>
<tr>
<td></td>
<td>D0–D7, X, Y, SP, CCH, CCL, or CCW</td>
<td></td>
</tr>
<tr>
<td></td>
<td>if same size, direct exchange</td>
<td></td>
</tr>
<tr>
<td></td>
<td>if 1st smaller than 2nd, sign extend 1st to 2nd</td>
<td></td>
</tr>
</tbody>
</table>
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.

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

<table>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clear (load with zero)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR Di</td>
<td>Clear data register Di , Memory, or Index Pointer</td>
<td>0 ⇒ Di, 0 ⇒ M, 0 ⇒ X, or 0 ⇒ Y</td>
</tr>
<tr>
<td>CLR (bwpl oprmemreg)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR Y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| **Set or Clear Bits (in memory or CCR)** | | |
| BSET Di,#opr5i | Set Bit n in Memory or in Di | (M) | bitn = M or (Di) | bitn = Di |
| BSET (bwpl oprmemreg,#opr5i) | C equal the original value of bitn in M or Di, (semaphore) | (M) | bitn = M or (Di) | bitn = Di |
| BCLR Dn,#opr5i | Clear Bit n in Memory or in Di | (M) & ~bitn = M or (Di) & ~bitn = Di |
| BCLR (bwpl oprmemreg,#opr5i) | C equal the original value of bitn in M or Di, (semaphore) | (M) & ~bitn = M or (Di) & ~bitn = Di |

| **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 #$02 | 1 ⇒ V |
| **CLC** | Clear Carry Bit | Translates to ANDCC #$FE | 0 ⇒ C |
| **CLI** | Clear I Bit; (I can only be changed in supervisor state) | Translates to ANDCC #$FE (enables I interrupts) | 0 ⇒ I |
| **CLV** | Clear Overflow Bit | Translates to ANDCC #$FD | 0 ⇒ V |
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>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Addition</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD $d_i$,#opr1mmss</td>
<td>Add without Carry to $d_i$</td>
<td>$(d_i) + (M) \Rightarrow d_i$</td>
</tr>
<tr>
<td>ADD $d_i$,oprmemreg</td>
<td>Add with Carry to $d_i$</td>
<td>$(d_i) + (M) + C \Rightarrow d_i$</td>
</tr>
<tr>
<td><strong>Increment</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC $d_i$</td>
<td>Increment data register $d_i$ or Memory</td>
<td>$(d_i) + 1 \Rightarrow d_i$, or $(M) + 1 \Rightarrow M$</td>
</tr>
<tr>
<td>INC.bwl oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Add 8-bit Signed Immediate to X, Y, or S (LEA)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEA $s$,#opr8i,S</td>
<td>Add sign-extended 8-bit Immediate to X, Y, or SP no change to CCR bits</td>
<td>$(SP) + \text{sign-extend} (M) \Rightarrow SP$, or $(X) + \text{sign-extend} (M) \Rightarrow X$, or $(Y) + \text{sign-extend} (M) \Rightarrow Y$</td>
</tr>
<tr>
<td>LEA $x$,#opr8i,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEA $y$,#opr8i,Y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Multiplication (MUL and MAC)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULS Dd,Dj,Dk</td>
<td>Signed Multiply</td>
<td>((Dj) \times (Dk) \Rightarrow D_d) or ((Dj) \times (M) \Rightarrow D_d), or ((M1) \times (M2) \Rightarrow D_d)</td>
</tr>
<tr>
<td>MULS Dd,Dj,#opr8i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULS Dd,Dj,#opr16i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULS Dd,Dj,#opr32i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULS.bwl Dd,Dj,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULS.bwpibwppl Dd,oprmemreg,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULU Dd,Dj,Dk</td>
<td>Unsigned Multiply</td>
<td>((Dj) \times (M) \Rightarrow D_d), or ((M1) \times (M2) \Rightarrow D_d)</td>
</tr>
<tr>
<td>MULU Dd,Dj,#opr8i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULU Dd,Dj,#opr16i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULU Dd,Dj,#opr32i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULU.bwl Dd,Dj,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULU.bwpibwppl Dd,oprmemreg,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACS Dd,Dj,Dk</td>
<td>Signed Multiply and Accumulate</td>
<td>((Dj) + (Dk) + D_d \Rightarrow D_d), or ((Dj) + (M) + D_d \Rightarrow D_d), or ((M1) + (M2) + D_d \Rightarrow D_d)</td>
</tr>
<tr>
<td>MACS Dd,Dj,#opr8i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACS Dd,Dj,#opr16i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACS Dd,Dj,#opr32i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACS.bwl Dd,Dj,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACS.bwpibwppl Dd,oprmemreg,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACU Dd,Dj,Dk</td>
<td>Unsigned Multiply and Accumulate</td>
<td>((Dj) + (Dk) + D_d \Rightarrow D_d), or ((Dj) + (M) + D_d \Rightarrow D_d), or ((M1) + (M2) + D_d \Rightarrow D_d)</td>
</tr>
<tr>
<td>MACU Dd,Dj,#opr8i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACU Dd,Dj,#opr16i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACU Dd,Dj,#opr32i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACU.bwl Dd,Dj,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACU.bwpibwppl Dd,oprmemreg,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Division (DIV and MOD)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVS Dd,Dj,Dk</td>
<td>Signed Divide</td>
<td>((Dj) \div (Dk) \Rightarrow D_d), or ((Dj) \div (M) \Rightarrow D_d), or ((M1) \div (M2) \Rightarrow D_d)</td>
</tr>
<tr>
<td>DIVS Dd,Dj,#opr8i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVS Dd,Dj,#opr16i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVS Dd,Dj,#opr32i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVS.bwl Dd,Dj,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVS.bwpibwppl Dd,oprmemreg,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVU Dd,Dj,Dk</td>
<td>Unsigned Divide</td>
<td>((Dj) \div (Dk) \Rightarrow D_d), or ((Dj) \div (M) \Rightarrow D_d), or ((M1) \div (M2) \Rightarrow D_d)</td>
</tr>
<tr>
<td>DIVU Dd,Dj,#opr8i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVU Dd,Dj,#opr16i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVU Dd,Dj,#opr32i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVU.bwl Dd,Dj,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVU.bwpibwppl Dd,oprmemreg,oprmemreg</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 5-3. Multiplication and Division Instructions (Sheet 2 of 2)

<table>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODS Dd,Dj,Dk</td>
<td>Signed Modulo result is always a register Dd</td>
<td>(Dj) % (Dk); remainder ⇒ Dd, or (Dj) % (D); remainder ⇒ Dp, or (M1) % (M2); remainder ⇒ Dd</td>
</tr>
<tr>
<td>MODS Dd,Dj,#opr8i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODS Dd,Dj,#opr16i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODS Dd,Dj,#opr32i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODS.bwl Dd,Dj,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODS.bwpibwp/ Dd,oprmemreg,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODU Dd,Dj,Dk</td>
<td>Unsigned Modulo result is always a register Dd</td>
<td>(Dj) % (Dk); remainder ⇒ Dd, or (Dj) % (D); remainder ⇒ Dp, or (M1) % (M2); remainder ⇒ Dd</td>
</tr>
<tr>
<td>MODU Dd,Dj,#opr8i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODU Dd,Dj,#opr16i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODU Dd,Dj,#opr32i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODU.bwl Dd,Dj,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODU.bwpibwp/ Dd,oprmemreg,oprmemreg</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.4.4.1 Fractional Multiply

These instructions 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.

---

Table 5-4. Fractional Math Instructions

<table>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fractional Multiplication</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QMULS $Dd, Dj, Dk$</td>
<td>Signed Fractional Multiply</td>
<td>$Dj \cdot (Dk) \Rightarrow Dd$, or $Dj \cdot (M) \Rightarrow Dd$, or $(M1) \cdot (M2) \Rightarrow Dd$</td>
</tr>
<tr>
<td>QMULS $Dd, Dj, #opr8i$</td>
<td>Signed Fractional Multiply</td>
<td>$Dj \cdot (Dk) \Rightarrow Dd$, or $Dj \cdot (M) \Rightarrow Dd$, or $(M1) \cdot (M2) \Rightarrow Dd$</td>
</tr>
<tr>
<td>QMULS $Dd, Dj, #opr16i$</td>
<td>Signed Fractional Multiply</td>
<td>$Dj \cdot (Dk) \Rightarrow Dd$, or $Dj \cdot (M) \Rightarrow Dd$, or $(M1) \cdot (M2) \Rightarrow Dd$</td>
</tr>
<tr>
<td>QMULS $Dd, Dj, #opr32i$</td>
<td>Signed Fractional Multiply</td>
<td>$Dj \cdot (Dk) \Rightarrow Dd$, or $Dj \cdot (M) \Rightarrow Dd$, or $(M1) \cdot (M2) \Rightarrow Dd$</td>
</tr>
<tr>
<td>QMULS $bw_Dd, Dj, opr_memreg$</td>
<td>Signed Fractional Multiply</td>
<td>$Dj \cdot (Dk) \Rightarrow Dd$, or $Dj \cdot (M) \Rightarrow Dd$, or $(M1) \cdot (M2) \Rightarrow Dd$</td>
</tr>
<tr>
<td>QMULS $bw_Dd, Dj, opr_memreg$</td>
<td>Signed Fractional Multiply</td>
<td>$Dj \cdot (Dk) \Rightarrow Dd$, or $Dj \cdot (M) \Rightarrow Dd$, or $(M1) \cdot (M2) \Rightarrow Dd$</td>
</tr>
<tr>
<td><strong>Unsigned Fractional Multiply</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QMULU $Dd, Dj, Dk$</td>
<td>Unsigned Fractional Multiply</td>
<td>$Dj \cdot (Dk) \Rightarrow Dd$, or $Dj \cdot (M) \Rightarrow Dd$, or $(M1) \cdot (M2) \Rightarrow Dd$</td>
</tr>
<tr>
<td>QMULU $Dd, Dj, #opr8i$</td>
<td>Unsigned Fractional Multiply</td>
<td>$Dj \cdot (Dk) \Rightarrow Dd$, or $Dj \cdot (M) \Rightarrow Dd$, or $(M1) \cdot (M2) \Rightarrow Dd$</td>
</tr>
<tr>
<td>QMULU $Dd, Dj, #opr16i$</td>
<td>Unsigned Fractional Multiply</td>
<td>$Dj \cdot (Dk) \Rightarrow Dd$, or $Dj \cdot (M) \Rightarrow Dd$, or $(M1) \cdot (M2) \Rightarrow Dd$</td>
</tr>
<tr>
<td>QMULU $Dd, Dj, #opr32i$</td>
<td>Unsigned Fractional Multiply</td>
<td>$Dj \cdot (Dk) \Rightarrow Dd$, or $Dj \cdot (M) \Rightarrow Dd$, or $(M1) \cdot (M2) \Rightarrow Dd$</td>
</tr>
<tr>
<td>QMULU $bw_Dd, Dj, opr_memreg$</td>
<td>Unsigned Fractional Multiply</td>
<td>$Dj \cdot (Dk) \Rightarrow Dd$, or $Dj \cdot (M) \Rightarrow Dd$, or $(M1) \cdot (M2) \Rightarrow Dd$</td>
</tr>
<tr>
<td>QMULU $bw_Dd, Dj, opr_memreg$</td>
<td>Unsigned Fractional Multiply</td>
<td>$Dj \cdot (Dk) \Rightarrow Dd$, or $Dj \cdot (M) \Rightarrow Dd$, or $(M1) \cdot (M2) \Rightarrow Dd$</td>
</tr>
</tbody>
</table>

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.
5.4.4.2 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

<table>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Logical AND</strong></td>
<td>AND Di, oprimm</td>
<td>Bitwise AND Di with Memory</td>
</tr>
<tr>
<td></td>
<td>ANDCC #opr8i</td>
<td>Bitwise AND CCL with immediate byte in Memory (S, X, and I can only be changed in supervisor state)</td>
</tr>
<tr>
<td></td>
<td>BIT Di, oprimm</td>
<td>Bitwise AND Di with Memory</td>
</tr>
<tr>
<td><strong>Logical OR</strong></td>
<td>OR Di, oprimm</td>
<td>Bitwise OR Di with Memory</td>
</tr>
<tr>
<td></td>
<td>ORCC #opr8i</td>
<td>Bitwise OR CCL with Immediate Mask (S, X, and I can only be changed in supervisor state)</td>
</tr>
<tr>
<td><strong>Logical Exclusive-OR</strong></td>
<td>EOR Di, oprimm</td>
<td>Exclusive OR Di with Memory</td>
</tr>
<tr>
<td><strong>Logical Invert (bit-by-bit ones complement)</strong></td>
<td>COM. bwl oprmemreg</td>
<td>1’s Complement Memory Location or Di</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arithmetic Shifts</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASL Dd,Ds,Dn</td>
<td>Arithmetic Shift Left D&lt;sub&gt;n&lt;/sub&gt; or memory, 0 to n positions</td>
<td><img src="image" alt="Diagram of Shift Left" /></td>
</tr>
<tr>
<td>ASL Dd,##opr1i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASL Dd,##opr5i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASL bwpl Dd,oprmemreg,##opr1i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASL bwpl Dd,oprmemreg,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASL bwpl oprmemreg,##opr1i</td>
<td>Arithmetic Shift Left memory by 1 or 2 positions</td>
<td><img src="image" alt="Diagram of Shift Left" /></td>
</tr>
<tr>
<td><strong>Logical Shifts</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSL Dd,Ds,Dn</td>
<td>Logical Shift Left D&lt;sub&gt;n&lt;/sub&gt; or memory, 0 to n positions.</td>
<td><img src="image" alt="Diagram of Shift Left" /></td>
</tr>
<tr>
<td>LSL Dd,##opr1i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSL Dd,##opr5i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSL bwpl Dd,oprmemreg,##opr1i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSL bwpl Dd,oprmemreg,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSL bwpl oprmemreg,##opr1i</td>
<td>Logical Shift Left memory by 1 or 2 position.</td>
<td><img src="image" alt="Diagram of Shift Left" /></td>
</tr>
<tr>
<td><strong>Rotate (one bit position through carry C-bit)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROL bwpl oprmemreg</td>
<td>Rotate Left through Carry D&lt;sub&gt;1&lt;/sub&gt; or memory, 1 bit position</td>
<td><img src="image" alt="Diagram of Rotate Left" /></td>
</tr>
<tr>
<td>ROR bwpl oprmemreg</td>
<td>Rotate Right through Carry D&lt;sub&gt;1&lt;/sub&gt; or memory, 1 bit position</td>
<td><img src="image" alt="Diagram of Rotate Right" /></td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Set, Clear, or Toggle Bits in Memory</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSET $Di,#opr5i</td>
<td>Set Bit n in Memory or in $Di</td>
<td>($M \lor \text{bitn} \Rightarrow M \lor (Di) \lor \text{bitn} \Rightarrow D_i$)</td>
</tr>
<tr>
<td>BSET.$bwl\ oprmemreg,#opr5i$</td>
<td>C equal the original value of bitn in M or D_i (semaphore)</td>
<td></td>
</tr>
<tr>
<td>BSET.$bwl\ oprmemreg,Di$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source Forms</td>
<td>Function</td>
<td>Operation</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>BCLR $Di, #opr5i</td>
<td>Clear Bit n in Memory or in $Di &lt;br&gt;C equal the original value of bitn in M or $Di &lt;br&gt;(semaphore)</td>
<td>$(M) \land \neg \text{bitn} \Rightarrow M$ or $(Di) \land \neg \text{bitn} \Rightarrow Di$</td>
</tr>
<tr>
<td>BCLR.bwl oprmemreg,$Di, #opr5i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCLR.bwl oprmemreg,$Di</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTGL $Di, #opr5i</td>
<td>Toggle Bit n in Memory or in $Di &lt;br&gt;C equal the original value of bitn in M or $Di &lt;br&gt;(semaphore)</td>
<td>$(M) \lor \text{bitn} \Rightarrow M$ or $(Di) \lor \text{bitn} \Rightarrow Di$</td>
</tr>
<tr>
<td>BTGL.bwl oprmemreg,$Di, #opr5i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTGL.bwl oprmemreg,$Di</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Set or Clear Bits in the CCR

<table>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANDCC $#opr8i$</td>
<td>Bitwise AND CCL with immediate byte in Memory &lt;br&gt;(Clear CCR bits that are 0 in the immediate mask) &lt;br&gt;(S, X, and I can only be changed in supervisor state)</td>
<td>$(CCL) \land (M) \Rightarrow CCL$</td>
</tr>
<tr>
<td>CLC</td>
<td>Clear Carry Bit &lt;br&gt;Translates to ANDCC #$FE</td>
<td>$0 \Rightarrow C$</td>
</tr>
<tr>
<td>CLI</td>
<td>Clear I Bit; (I can only be changed in supervisor state) &lt;br&gt;Translates to ANDCC #$EF &lt;br&gt;(enables I interrupts)</td>
<td>$0 \Rightarrow I$</td>
</tr>
<tr>
<td>CLV</td>
<td>Clear Overflow Bit &lt;br&gt;Translates to ANDCC #$FD</td>
<td>$0 \Rightarrow V$</td>
</tr>
<tr>
<td>ORCC $#opr8i$</td>
<td>Bitwise OR CCL with Immediate Mask &lt;br&gt;(Set CCR bits that are 1 in the immediate mask) &lt;br&gt;(S, X, and I can only be changed in supervisor state)</td>
<td>$(CCL) \lor (M) \Rightarrow CCL$</td>
</tr>
<tr>
<td>SEC</td>
<td>Set Carry Bit &lt;br&gt;Translates to ORCC #$01</td>
<td>$1 \Rightarrow C$</td>
</tr>
<tr>
<td>SEI</td>
<td>Set I Bit; (inhibit I interrupts) &lt;br&gt;Translates to ORCC #$10</td>
<td>$1 \Rightarrow I$</td>
</tr>
<tr>
<td>SEV</td>
<td>Set Overflow Bit &lt;br&gt;Translates to ORCC #$02</td>
<td>$1 \Rightarrow V$</td>
</tr>
</tbody>
</table>
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.

---

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

<table>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFEXT Dd,Ds,Dp</td>
<td>Bit Field Extract</td>
<td>Extract bit field with width w and offset o from Ds or a memory operand, and store it into the low order bits of Dd or memory (filling unused bits with 0). The source operand or destination operand must be a register (memory to memory not allowed)</td>
</tr>
<tr>
<td>BFEXT Dd,Ds,#width:offset</td>
<td>Bit Field Extract</td>
<td></td>
</tr>
<tr>
<td>BFEXT.bwpl Dd,oprmemreg,Dp</td>
<td>Bit Field Insert</td>
<td>Insert bit field with width w from the low order bits of Dp or a memory operand into Dd 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)</td>
</tr>
<tr>
<td>BFEXT.bwpl oprmemreg,Ds,#width:offset</td>
<td>Bit Field Insert</td>
<td></td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXS Di,oprmemreg</td>
<td>MAXimum of two signed operands replaces Di</td>
<td>MAX((D_i), (M)) ⇒ D_i</td>
</tr>
<tr>
<td>MAXU Di,oprmemreg</td>
<td>MAXimum of two unsigned operands replaces Di</td>
<td>MAX((D_i), (M)) ⇒ D_i</td>
</tr>
<tr>
<td>MINS Di,oprmemreg</td>
<td>MINimum of two signed operands replaces Di</td>
<td>MIN((D_i), (M)) ⇒ D_i</td>
</tr>
<tr>
<td>MINU Di,oprmemreg</td>
<td>MINimum of two unsigned operands replaces Di</td>
<td>MIN((D_i), (M)) ⇒ D_i</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Load</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD xy,#opr18i</td>
<td>Load index register X or Y from Memory</td>
<td>(M:M+1:M+2) ⇒ X or Y</td>
</tr>
<tr>
<td>LD xy,#opr24i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD xy,opr24a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD xy,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD S,#opr24i</td>
<td>Load stack pointer SP from Memory</td>
<td>(M:M+1:M+2) ⇒ SP</td>
</tr>
<tr>
<td>LD S,oprmemreg</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Pull (Load CPU Registers from Stack)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUL oprregs1</td>
<td>Pull specified CPU registers from Stack</td>
<td>(M(SP)−(M(SP)+n−1)) ⇒ regs; (SP) + n ⇒ SP</td>
</tr>
<tr>
<td>PUL oprregs2</td>
<td>mask 1 - CCH, CCL, D0, D1, D2, D3 (D3 in LSB)</td>
<td></td>
</tr>
<tr>
<td>PUL ALL</td>
<td>mask 2 - D4, D5, D6, D7, X, Y (Y in LSB)</td>
<td></td>
</tr>
<tr>
<td>PUL ALL16b</td>
<td>pulls all registers in the same order as RTI</td>
<td></td>
</tr>
<tr>
<td><strong>Restore CPU Registers after Interrupts (RTI)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTI</td>
<td>Return from Interrupt</td>
<td>(M(SP)−(M(SP)+3)) ⇒ CCH:CCL, D0, D1; (SP)+4 ⇒ SP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(M(SP)−(M(SP)+3)) ⇒ D2H:D2L, D3H:D3L; (SP)+4 ⇒ SP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(M(SP)−(M(SP)+3)) ⇒ D4H:D4L, D5H:D5L; (SP)+4 ⇒ SP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(M(SP)−(M(SP)+3)) ⇒ D6H:D6L; (SP)+4 ⇒ SP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(M(SP)−(M(SP)+3)) ⇒ D7H:D7L; (SP)+4 ⇒ SP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(M(SP)−(M(SP)+2)) ⇒ XH:XLM:XL; (SP)+3 ⇒ SP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(M(SP)−(M(SP)+2)) ⇒ YH:YLM:YL; (SP)+3 ⇒ SP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(M(SP)−(M(SP)+2)) ⇒ RTNH:RTNM; (SP)+3 ⇒ SP</td>
</tr>
</tbody>
</table>

Table 5-9. Index and Pointer Manipulation Instructions (Sheet 1 of 3)
### Table 5-9. Index and Pointer Manipulation Instructions (Sheet 2 of 3)

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

<table>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unconditional Branches</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRA oprdest</td>
<td>Branch Always</td>
<td>(if $1 = 1$) Branch offset is 7 bits or 15 bits</td>
</tr>
<tr>
<td>BSR oprdest</td>
<td>Branch to Subroutine</td>
<td>(SP) $\rightarrow$ SP; RTNH:RTNM:RTNL $\rightarrow$ M(SP):M(SP+1):M(SP+2); Subroutine Address $\rightarrow$ PC Branch offset is 7 bits or 15 bits</td>
</tr>
<tr>
<td><strong>Simple Branches</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCC oprdest</td>
<td>Branch if Carry Clear (same as BHS)</td>
<td>(if $C = 0$) Branch offset is 7 bits or 15 bits</td>
</tr>
<tr>
<td>BCS oprdest</td>
<td>Branch if Carry Set (same as BLO)</td>
<td>(if $C = 1$) Branch offset is 7 bits or 15 bits</td>
</tr>
<tr>
<td>BEQ oprdest</td>
<td>Branch if Equal</td>
<td>(if $Z = 1$) Branch offset is 7 bits or 15 bits</td>
</tr>
<tr>
<td>BMI oprdest</td>
<td>Branch if Minus</td>
<td>(if $N = 1$) Branch offset is 7 bits or 15 bits</td>
</tr>
<tr>
<td>BNE oprdest</td>
<td>Branch if Not Equal; $R \neq 0$</td>
<td>(if $Z = 0$) Branch offset is 7 bits or 15 bits</td>
</tr>
<tr>
<td>BPL oprdest</td>
<td>Branch if Plus</td>
<td>(if $N = 0$) Branch offset is 7 bits or 15 bits</td>
</tr>
<tr>
<td>BVC oprdest</td>
<td>Branch if Overflow Bit Clear</td>
<td>(if $V = 0$) Branch offset is 7 bits or 15 bits</td>
</tr>
<tr>
<td>BVS oprdest</td>
<td>Branch if Overflow Bit Set</td>
<td>(if $V = 1$) Branch offset is 7 bits or 15 bits</td>
</tr>
<tr>
<td><strong>Signed Branches</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BGE oprdest</td>
<td>Branch if Greater Than or Equal; signed $R \geq M$</td>
<td>(if $N \land V = 0$) Branch offset is 7 bits or 15 bits</td>
</tr>
<tr>
<td>BGT oprdest</td>
<td>Branch if Greater Than; signed $R &gt; M$</td>
<td>(if $Z \land (N \land V) = 0$) Branch offset is 7 bits or 15 bits</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP opr24a</td>
<td>Jump (unconditional)</td>
<td>Effective Address (\rightarrow) PC</td>
</tr>
<tr>
<td>JMP opmemreg</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSR opr24a</td>
<td>Jump to Subroutine</td>
<td>(SP) – 3 (\Rightarrow) SP; RTNH:RTNM:RTNL (\Rightarrow) M(SP):M(SP+1):M(SP+2); Subroutine Address (\rightarrow) PC</td>
</tr>
<tr>
<td>JSR opmemreg</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 5-12. Jump and Subroutine Instructions (Sheet 2 of 2)

<table>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BSR oprdest</strong></td>
<td>Branch to Subroutine</td>
<td>(SP) – 3 (\Rightarrow) SP; RTNH:RTNM:RTNL (\Rightarrow) M(SP):M(SP+1):M(SP+2); Subroutine Address (\Rightarrow) PC Branch offset is 7 bits or 15 bits</td>
</tr>
<tr>
<td><strong>RTS</strong></td>
<td>Return from Subroutine</td>
<td>(M(SP):M(SP+1):M(SP+2)) (\Rightarrow) PCH:PCM:PCL; (SP) + 3 (\Rightarrow) SP</td>
</tr>
</tbody>
</table>
### Table 5-13. Interrupt Instructions (Sheet 1 of 2)

<table>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interrupt Stacking</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SWI</strong></td>
<td>Software Interrupt</td>
<td>(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</td>
</tr>
<tr>
<td><strong>SYS</strong></td>
<td>System Call Software Interrupt</td>
<td>(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</td>
</tr>
<tr>
<td><strong>TRAP #trapnum</strong></td>
<td>Unimplemented (pg2) Opcode Trap Interrupt</td>
<td>(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</td>
</tr>
<tr>
<td><strong>SPARE</strong></td>
<td>Unimplemented pg1 Opcode Trap Interrupt</td>
<td>(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</td>
</tr>
</tbody>
</table>
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.

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

<table>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAI</td>
<td>Wait for Interrupt</td>
<td>(SP) – 3 ⇒ SP; RTINH:RTNM:RTNL ⇒ M(SP)–M(SP+2); (SP) – 3 ⇒ SP; YH:YM:YL ⇒ M(SP)–M(SP+2); (SP) – 3 ⇒ SP; 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</td>
</tr>
<tr>
<td>RTI</td>
<td>Return from Interrupt</td>
<td>(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)) ⇒ XM:XL; (SP)+3 ⇒ SP (M(SP)–M(SP+2)) ⇒ YM:YL; (SP)+3 ⇒ SP (M(SP)–M(SP+2)) ⇒ RTINH:RTNM:RTNL; (SP)+3 ⇒ SP</td>
</tr>
</tbody>
</table>

**Interrupt Enable and Disable**

<table>
<thead>
<tr>
<th>CLI</th>
<th>Clear I Bit; (I can only be changed in supervisor state)</th>
<th>0 ⇒ 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Translates to ANDCC #$EF (enables I interrupts)</td>
<td></td>
</tr>
<tr>
<td>SEI</td>
<td>Set I Bit; (inhibit I interrupts)</td>
<td>1 ⇒ 1</td>
</tr>
<tr>
<td></td>
<td>Translates to ORCC #$10</td>
<td></td>
</tr>
</tbody>
</table>

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.
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>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STOP</strong></td>
<td>STOP All Clocks and enter a low power state&lt;br&gt;If S control bit = 1, the STOP instruction is disabled and acts like a NOP.</td>
<td>(SP) – 3 ⇒ SP; RTNH; RTNM; RTNL ⇒ M(SP)–M(SP+2);&lt;br&gt;(SP) – 3 ⇒ SP; YH; YM; YL ⇒ M(SP)–M(SP+2);&lt;br&gt;(SP) – 3 ⇒ SP; XH; XM; XL ⇒ M(SP)–M(SP+2);&lt;br&gt;(SP) – 4 ⇒ SP; D7H; D7M; D7L; D7L ⇒ M(SP)–M(SP+3);&lt;br&gt;(SP) – 4 ⇒ SP; D6H; D6M; D6L; D6L ⇒ M(SP)–M(SP+3);&lt;br&gt;(SP) – 4 ⇒ SP; D5H; D5L; D5L ⇒ M(SP)–M(SP+3);&lt;br&gt;(SP) – 4 ⇒ SP; D4H; D4L; D4L ⇒ M(SP)–M(SP+3);&lt;br&gt;(SP) – 4 ⇒ SP; D3H; D3L; D3L ⇒ M(SP)–M(SP+3);&lt;br&gt;(SP) – 4 ⇒ SP; CCH, CCL, D0, D1 ⇒ M(SP)–M(SP+3);&lt;br&gt;STOP All Clocks</td>
</tr>
<tr>
<td><strong>WAI</strong></td>
<td>Wait for Interrupt</td>
<td>(SP) – 3 ⇒ SP; RTNH; RTNM; RTNL ⇒ M(SP)–M(SP+2);&lt;br&gt;(SP) – 3 ⇒ SP; YH; YM; YL ⇒ M(SP)–M(SP+2);&lt;br&gt;(SP) – 3 ⇒ SP; XH; XM; XL ⇒ M(SP)–M(SP+2);&lt;br&gt;(SP) – 4 ⇒ SP; D7H; D7M; D7L; D7L ⇒ M(SP)–M(SP+3);&lt;br&gt;(SP) – 4 ⇒ SP; D6H; D6M; D6L; D6L ⇒ M(SP)–M(SP+3);&lt;br&gt;(SP) – 4 ⇒ SP; D5H; D5L; D5L ⇒ M(SP)–M(SP+3);&lt;br&gt;(SP) – 4 ⇒ SP; D4H; D4L; D4L ⇒ M(SP)–M(SP+3);&lt;br&gt;(SP) – 4 ⇒ SP; D3H; D3L; D3L ⇒ M(SP)–M(SP+3);&lt;br&gt;(SP) – 4 ⇒ SP; CCH, CCL, D0, D1 ⇒ M(SP)–M(SP+3);&lt;br&gt;when interrupt occurs, $1 \Rightarrow i$; (Vector) ⇒ PC</td>
</tr>
<tr>
<td><strong>NOP</strong></td>
<td>No operation</td>
<td>–</td>
</tr>
</tbody>
</table>
Chapter 5 Instruction Set Overview

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.

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

<table>
<thead>
<tr>
<th>Source Forms</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGND</td>
<td>Enter active Background mode</td>
<td>enter Background if BDM enabled; else continue</td>
</tr>
</tbody>
</table>

Enter Active Background Debug Mode (BGND)
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

Operation

\[ |D_i| \Rightarrow D_i \]

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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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

\[
\begin{array}{cccccccccccc}
  7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & 1 & 4n \\
\hline
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & \text{SD REGISTER } D_i \\
0 & 1 & 0 & 0 & 0 & 0 & \text{SD REGISTER } D_i \\
\end{array}
\]

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

Operation

\[(D_i) + (M) + C \Rightarrow D_i\]

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC (D_i, #opr\text{immsz})</td>
<td>IMM1/2/4</td>
</tr>
<tr>
<td>ADC (D_i, \text{oprmemreg})</td>
<td>OPR/1/2/3</td>
</tr>
</tbody>
</table>

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, \(\text{oprmemreg}\) 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

<table>
<thead>
<tr>
<th></th>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
</tr>
</tbody>
</table>

- **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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 \(D_i, \#opr8i\); for \(D_i = 8\)-bit \(D_0\) or \(D_1\)
1B 5p i2 i1 ADC \(D_i, \#opr16i\); for \(D_i = 16\)-bit \(D_2, D_3, D_4, \) or \(D_5\)
1B 5p i4 i3 i2 i1 ADC \(D_i, \#opr32i\); for \(D_i = 32\)-bit \(D_6\) or \(D_7\)
### Instruction Fields

**SD REGISTER Di** - This field specifies the number of the data register Di 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 Di.

**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-indirect memory location.
ADD
Add without Carry

Operation
(Di) + (M) ⇒ Di

Syntax Variations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD Di,opr8msz</td>
<td>IMM1/2/4</td>
</tr>
<tr>
<td>ADD Di,oprmemreg</td>
<td>OPR/1/2/3</td>
</tr>
</tbody>
</table>

Description
Add without carry to register Di and store the result to Di. When the operand is an immediate value, it has the same size as register Di. 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 Di at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode.

CCR Details

<table>
<thead>
<tr>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SD REGISTER Di

IMMEDIATE DATA

(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF Di)

(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF Di)

(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF Di)

ADD Di,#opr8i ;for Di = 8-bit D0 or D1
ADD Di,#opr16i ;for Di = 16-bit D2, D3, D4, or D5
ADD Di,#opr32i ;for Di = 32-bit D6 or D7
**Instruction Fields**

SD REGISTER $Di$ - This field specifies the number of the data register $Di$ 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 $Di$.

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

**Operation**

\((D_i) & (M) \Rightarrow D_i\)

**Syntax Variations**

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND (D_i, #)opr(\text{mmsz})</td>
<td>IMM1/2/4</td>
</tr>
<tr>
<td>AND (D_i, \text{oprmemreg})</td>
<td>OPR1/2/3</td>
</tr>
</tbody>
</table>

**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, \(\text{oprmemreg}\) 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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
</tr>
</tbody>
</table>

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 1

SD REGISTER \(D_i\)
```

```
5p i1 AND \(D_i, \#\)opr8i ;for \(D_i\) = 8-bit D0 or D1
5p i2 i1 AND \(D_i, \#\)opr16i ;for \(D_i\) = 16-bit D2, D3, D4, or D5
5p i4 i3 i2 i1 AND \(D_i, \#\)opr32i ;for \(D_i\) = 32-bit D6 or D7
```
**Instruction Fields**

**SD REGISTER Di** - This field specifies the number of the data register Di 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=D6, 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 Di.

**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  

Operation  
(CCL) & (M) ⇒ CCL  

Syntax Variations  
```
ANDCC  #opr8i
```

Addressing Modes  
```
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  
```
<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
```

For supervisor 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 | 1 | 1 | 1 | 0 |
```

```
CE i1  
ANDCC  #opr8i
```
**ASL**

**Arithmetic Shift Left**

**Operation**

![Shift Left Diagram]

**Syntax Variations**

<table>
<thead>
<tr>
<th>ASL</th>
<th>Dd, Ds, Dn</th>
<th>REG-REG</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASL</td>
<td>Dd, Ds, #opr5i</td>
<td>REG-IMM (1-bit, or 5-bit)</td>
</tr>
<tr>
<td>ASL</td>
<td>Dd, Ds, opmemreg</td>
<td>REG-OPR/1/2/3</td>
</tr>
<tr>
<td>ASL.bwpl</td>
<td>Dd, opmemreg, #opr5i</td>
<td>OPR/1/2/3-IMM (1-bit, or 5-bit)</td>
</tr>
<tr>
<td>ASL.bwpl</td>
<td>Dd, opmemreg, opmemreg</td>
<td>OPR/1/2/3-OPR/1/2/3</td>
</tr>
<tr>
<td>ASL</td>
<td>Di, #oprli ; 2-operand, n=1 or 2</td>
<td>REG-IMM (2-operand)</td>
</tr>
<tr>
<td>ASL.bwpl</td>
<td>oprmemreg, #oprli ; 2-operand, n=1 or 2</td>
<td>OPR/1/2/3-IMM (2-operand)</td>
</tr>
</tbody>
</table>

**Addressing Modes**

**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, opmemreg 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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>DESTINATION REGISTER Dd</td>
<td>1n</td>
<td></td>
</tr>
<tr>
<td>A/L=1</td>
<td>L/R=1</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>SOURCE REGISTER Ds</td>
<td>sb</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>PARAMETER REGISTER Dn</td>
<td>xb</td>
<td></td>
</tr>
</tbody>
</table>

ln sb xb  
ASL  
Dd, Ds, Dn

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>DESTINATION REGISTER Dd</td>
<td>1n</td>
<td></td>
</tr>
<tr>
<td>A/L=1</td>
<td>L/R=1</td>
<td>0</td>
<td>0</td>
<td>N[0]</td>
<td>SOURCE REGISTER Ds</td>
<td>sb</td>
<td></td>
</tr>
</tbody>
</table>

ln sb  
ASL  
Dd, Ds, #opr5i

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>DESTINATION REGISTER Dd</td>
<td>1n</td>
<td></td>
</tr>
<tr>
<td>A/L=1</td>
<td>L/R=1</td>
<td>0</td>
<td>0</td>
<td>N[0]</td>
<td>SOURCE REGISTER Ds</td>
<td>sb</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>N[4:1]</td>
<td>xb</td>
<td></td>
</tr>
</tbody>
</table>

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

#### REG-OPR/1/2/3

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>DESTINATION REGISTER Dd</td>
<td>1n</td>
<td></td>
</tr>
<tr>
<td>A/L=1</td>
<td>L/R=1</td>
<td>0</td>
<td>0</td>
<td>x or N[0]</td>
<td>SOURCE REGISTER Ds</td>
<td>sb</td>
<td></td>
</tr>
</tbody>
</table>

OPR POSTBYTE (specifies number of shifts in byte-sized memory operand)  
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)  
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)  
(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)  

ln sb xb  
ASL  
Dd, Ds, Dn ; see REG-REG

ln sb xb  
ASL  
Dd, Ds, #opr5i ; see REG-IMM n=xb[3:0]:sb[3]

ln sb xb  
ASL  
Dd, Ds, (opr4, xys)

ln sb xb  
ASL  
Dd, Ds, {+-xy} | {-s} | (s+)

ln sb xb  
ASL  
Dd, Ds, (Di, xys)

ln sb xb  
ASL  
Dd, Ds, [Di, xy]

ln sb xb x1  
ASL  
Dd, Ds, (opra9, xysp)

ln sb xb x1  
ASL  
Dd, Ds, [opra9, xysp]

ln sb xb x1  
ASL  
Dd, Ds, oprul4

ln sb xb x2 x1  
ASL  
Dd, Ds, (oprul8, Di)

ln sb xb x2 x1  
ASL  
Dd, Ds, oprul8

ln sb xb x3 x2 x1  
ASL  
Dd, Ds, (opr24, xysp)

ln sb xb x3 x2 x1  
ASL  
Dd, Ds, [opr24, xysp]

ln sb xb x3 x2 x1  
ASL  
Dd, Ds, (opr24, Di)

ln sb xb x3 x2 x1  
ASL  
Dd, Ds, opr24

ln sb xb x3 x2 x1  
ASL  
Dd, Ds, [opr24]
Chapter 6 Instruction Glossary

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>DESTINATION REGISTER D7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A/L=1</td>
<td>L/R=1</td>
<td>1</td>
<td>0</td>
<td>N[0]</td>
<td>0</td>
<td>SIZE (.B, .W, .P, .L)</td>
<td></td>
</tr>
</tbody>
</table>

OPR POSTBYTE (specifies source operand to be shifted)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

ln sb xb ASL.bpl #oprxe4i,#opr1i
ln sb xb ASL.bpl 
Ds,#opr1i ;see more efficient REG-IMM version
ln sb xb ASL.bpl (opr4,xys),#opr1i
ln sb xb ASL.bpl ((-xy)|(xy+-)|(s-)|(s+)），#opr1i
ln sb xb ASL.bpl (Di,xys),#opr1i
ln sb xb ASL.bpl [Di,xy],#opr1i
ln sb xb x1 ASL.bpl (oprs9,xysp),#opr1i
ln sb xb x1 ASL.bpl [oprs9,xysp],#opr1i
ln sb xb x1 ASL.bpl opru14,#opr1i
ln sb xb x2 x1 ASL.bpl (oprul8,Di),#opr1i
ln sb xb x2 x1 ASL.bpl opru18,#opr1i
ln sb xb x3 x2 x1 ASL.bpl (opr24,xysp),#opr1i
ln sb xb x3 x2 x1 ASL.bpl [opru24,xysp],#opr1i
ln sb xb x3 x2 x1 ASL.bpl opr24,#opr1i
ln sb xb x3 x2 x1 ASL.bpl [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.

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>DESTINATION REGISTER D7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A/L=1</td>
<td>L/R=1</td>
<td>1</td>
<td>1</td>
<td>N[0]</td>
<td>0</td>
<td>SIZE (.B, .W, .P, .L)</td>
<td></td>
</tr>
</tbody>
</table>

OPR POSTBYTE (specifies source operand to be shifted)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

0 1 1 1 N[4:1] xb

ln sb xb xb ASL.bpl #oprsxe4i,#opr5i
ln sb xb xb ASL.bpl Di,#opr5i ;see more efficient REG-IMM version
ln sb xb xb ASL.bpl (opr4,xys),#opr5i
ln sb xb xb ASL.bpl ((-xy)|(xy+-)|(s-)|(s+)），#opr5i
ln sb xb xb ASL.bpl (Di,xys),#opr5i
ln sb xb xb ASL.bpl [Di,xy],#opr5i
ln sb xb x1 xb ASL.bpl (oprs9,xysp),#opr5i
ln sb xb x1 xb ASL.bpl [oprs9,xysp],#opr5i
ln sb xb x1 xb ASL.bpl opru14,#opr5i
ln sb xb x2 x1 xb ASL.bpl (oprul8,Di),#opr5i
ln sb xb x2 x1 xb ASL.bpl opru18,#opr5i
ln sb xb x3 x2 x1 xb ASL.bpl (opr24,xysp),#opr5i
ln sb xb x3 x2 x1 xb ASL.bpl [opru24,xysp],#opr5i
ln sb xb x3 x2 x1 xb ASL.bpl opr24,#opr5i
ln sb xb x3 x2 x1 xb ASL.bpl [opr24],#opr5i
ln sb xb x3 x2 x1 xb ASL.bpl opr24,#opr5i
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)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

A/L=1 | L/R=1 | 1 | 1 | N[0] | 1 | x:x or SIZE (.B, .W, .P, .L)

- OPR POSTBYTE (specifies source operand to be shifted)
- (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.bpl #oprse4i,#opr1i ;shifting IMM const not allowed
1n sb xb ASL Di,#opr1i ;2-operand register shift by 1 or 2
1n sb xb ASL.bpl (opru4,xys),#opr1i
1n sb xb ASL.bpl ((+-xy)|(xy+-)|(s)|(s+)),#opr1i
1n sb xb ASL.bpl (Di,xys),#opr1i
1n sb xb ASL.bpl [Di,xy],#opr1i
1n sb xb x1 ASL.bpl (oprs9,xysp),#opr1i
1n sb xb x1 ASL.bpl [oprs9,xysp],#opr1i
1n sb xb x1 ASL.bpl opru14,#opr1i
1n sb xb x2 x1 ASL.bpl (opru18,Di),#opr1i
1n sb xb x2 x1 ASL.bpl opru18,#opr1i
1n sb xb x3 x2 x1 ASL.bpl (opr24,xysp),#opr1i
1n sb xb x3 x2 x1 ASL.bpl [opr24,xysp],#opr1i
1n sb xb x3 x2 x1 ASL.bpl (opru24,Di),#opr1i
1n sb xb x3 x2 x1 ASL.bpl opru24,#opr1i
1n sb xb x3 x2 x1 ASL.bpl [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 Dd - This field specifies data register Dd (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 Ds - This field specifies data register Ds (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 Dn - This field specifies the number of the data register Dn (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 Di - This field specifies the number of the data register Di 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

### Operation

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, \( \text{oprmemreg} \) 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 \( \text{sb postbyte} \) and the higher four bits are encoded as a short-immediate value in the \( \text{xb postbyte} \). 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- **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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>DESTINATION REGISTER (D_d)</td>
<td>1n</td>
<td></td>
</tr>
<tr>
<td>A/L=1</td>
<td>L/R=0</td>
<td>1</td>
<td>0</td>
<td>(N[0])</td>
<td>SOURCE REGISTER (D_s)</td>
<td>sb</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>PARAMETER REGISTER (D_n)</td>
<td>xb</td>
<td></td>
</tr>
</tbody>
</table>

\(\text{ln, sb, xb} \quad \text{ASR} \quad D_d, D_s, D_n\)

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>DESTINATION REGISTER (D_d)</td>
<td>1n</td>
<td></td>
</tr>
<tr>
<td>A/L=1</td>
<td>L/R=0</td>
<td>0</td>
<td>1</td>
<td>(N[0])</td>
<td>SOURCE REGISTER (D_s)</td>
<td>sb</td>
<td></td>
</tr>
</tbody>
</table>

\(\text{ln, sb} \quad \text{ASR} \quad D_d, D_s, \#opr1i\)

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>DESTINATION REGISTER (D_d)</td>
<td>1n</td>
<td></td>
</tr>
<tr>
<td>A/L=1</td>
<td>L/R=0</td>
<td>0</td>
<td>1</td>
<td>(N[0])</td>
<td>SOURCE REGISTER (D_s)</td>
<td>sb</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(N[4:1])</td>
<td>(N[4:1])</td>
<td>xb</td>
<td></td>
</tr>
</tbody>
</table>

\(\text{ln, sb, xb} \quad \text{ASR} \quad D_d, D_s, \#opr5i \quad ; N[0] \text{ in sb, } N[4:1] \text{ in xb}\)

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>DESTINATION REGISTER (D_d)</td>
<td>1n</td>
<td></td>
</tr>
<tr>
<td>A/L=1</td>
<td>L/R=0</td>
<td>1</td>
<td>0</td>
<td>(N[0])</td>
<td>SOURCE REGISTER (D_s)</td>
<td>sb</td>
<td></td>
</tr>
</tbody>
</table>

\(\text{OPR POSTBYTE (specifies source operand to be shifted)}\)

\((\text{OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE})\)

\((\text{OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE})\)

\((\text{OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE})\)

\(\text{ln, sb, xb} \quad \text{ASR, bpl} \quad \#opr\text{xe4i}, \#opr1i\)

\(\text{ln, sb, xb} \quad \text{ASR, bpl} \quad D_s, \#opr1i \quad ; \text{see more efficient REG-IMM version}\)

\(\text{ln, sb, xb} \quad \text{ASR, bpl} \quad \text{OPR4, xys}, \#opr1i\)

\(\text{ln, sb, xb} \quad \text{ASR, bpl} \quad \{(+xy)|(-xy)|(-s)|(+s)\}, \#opr1i\)

\(\text{ln, sb, xb} \quad \text{ASR, bpl} \quad (\text{Di, xys}), \#opr1i\)

\(\text{ln, sb, xb} \quad \text{ASR, bpl} \quad [\text{Di, xy}], \#opr1i\)

\(\text{ln, sb, xb, x1} \quad \text{ASR, bpl} \quad (\text{oprs9, xysp}), \#opr1i\)

\(\text{ln, sb, xb, x1} \quad \text{ASR, bpl} \quad [\text{oprs9, xysp}], \#opr1i\)

\(\text{ln, sb, xb, x1} \quad \text{ASR, bpl} \quad \text{opru14}, \#opr1i\)

\(\text{ln, sb, xb, x2, x1} \quad \text{ASR, bpl} \quad (\text{opru18, Di}), \#opr1i\)

\(\text{ln, sb, xb, x2, x1} \quad \text{ASR, bpl} \quad \text{opru18}, \#opr1i\)

\(\text{ln, sb, xb, x3, x2, x1} \quad \text{ASR, bpl} \quad (\text{opru24, xysp}), \#opr1i\)

\(\text{ln, sb, xb, x3, x2, x1} \quad \text{ASR, bpl} \quad [\text{opru24, xysp}], \#opr1i\)

\(\text{ln, sb, xb, x3, x2, x1} \quad \text{ASR, bpl} \quad (\text{opru24, Di}), \#opr1i\)

\(\text{ln, sb, xb, x3, x2, x1} \quad \text{ASR, bpl} \quad \text{opru24}, \#opr1i\)

\(\text{ln, sb, xb, x3, x2, x1} \quad \text{ASR, bpl} \quad [\text{opru24}], \#opr1i\)
**OPR/1/2/3-IMM** (normal shift by 0 to 31 positions)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>DESTINATION REGISTER Dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A/L=1</td>
<td>L/R=1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>N[0]</td>
<td>0</td>
<td>SIZE (.B, .W, .P, .L)</td>
</tr>
</tbody>
</table>

**OPR POSTBYTE** (specifies source operand to be shifted)

**OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE**

**OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE**

0 0 0 0 1 1 1 1

---

1n sb xb xb ASR.nlm #oprxexi1,#opr5i
1n sb xb xb ASR.nlm Di,#opr5i ;see more efficient REG-IMM version
1n sb xb xb ASR.nlm (opr24,xys),#opr5i
1n sb xb xb ASR.nlm (opr24,xys),#opr5i
1n sb xb x2 x1 xb ASR.nlm (opr24,xys),#opr5i
1n sb xb x2 x1 xb ASR.nlm (opr24,xys),#opr5i
1n sb xb x3 x2 x1 xb ASR.nlm (opr24,xys),#opr5i
1n sb xb x3 x2 x1 xb ASR.nlm [opr24,xys],#opr5i
1n sb xb x3 x2 x1 xb ASR.nlm [opr24,xys],#opr5i
### OPR/1/2/3-OPR/1/2/3

<table>
<thead>
<tr>
<th>Opcode postbyte</th>
<th>Source operand object code</th>
<th>Parameter # of shifts object code</th>
<th>Instruction Mnemonic</th>
<th>Source Format for Source Operand (select 1 option in this col)</th>
<th>Source Format for Parameter (# of shifts) (select 1 option in this col)</th>
</tr>
</thead>
<tbody>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td></td>
<td>#oprsx4i,</td>
<td>#oprsx4i</td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td></td>
<td>Ds,</td>
<td>Dn</td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td></td>
<td>(opru4, xys),</td>
<td>(opru4, xys)</td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td></td>
<td>(+xy)</td>
<td>(+xy++)</td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td></td>
<td>(Dj, xys),</td>
<td>(Dk, xys)</td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td></td>
<td>[Dj, xy],</td>
<td>[Dk, xy]</td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td></td>
<td>(oprs9, xysp),</td>
<td>(oprs9, xysp)</td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td></td>
<td>[oprs9, xysp],</td>
<td>[oprs9, xysp]</td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td></td>
<td>opru14,</td>
<td>opru14</td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td></td>
<td>(opru18, Dj),</td>
<td>(opru18, Dk)</td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td></td>
<td>opru18,</td>
<td>opru18</td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td></td>
<td>(opru24, xysp),</td>
<td>(opru24, xysp)</td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td></td>
<td>[opru24, xysp],</td>
<td>[opru24, xysp]</td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td></td>
<td>(opru24, Dj),</td>
<td>(opru24, Dk)</td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td></td>
<td>opru24,</td>
<td>opru24</td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td></td>
<td>(opru24)</td>
<td>[opru24]</td>
</tr>
</tbody>
</table>

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.
Chapter 6 Instruction Glossary

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>A/L=1</td>
<td>L/R=0</td>
<td>1</td>
<td>1</td>
<td>N[0]</td>
<td>1</td>
<td>SIZE (.B, .W, .P, .L)</td>
<td></td>
</tr>
</tbody>
</table>

OPR POSTBYTE (specifies source operand to be shifted)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

ln sb xb ASR.bmpl #oprxe4i,#opr1i
ln sb xb ASR.bmpl #opr1i ; see more efficient REG-IMM version
ln sb xb ASR.bmpl (opr4,xys),#opr1i
ln sb xb ASR.bmpl (xy),(x+y),#opr1i
ln sb xb ASR.bmpl (Di,xys),#opr1i
ln sb xb ASR.bmpl [Di,xy],#opr1i
ln sb xb x1 ASR.bmpl (opr9,xys),#opr1i
ln sb xb x1 ASR.bmpl [opr9,xys],#opr1i
ln sb xb x1 ASR.bmpl opru14,#opr1i
ln sb xb x2 x1 ASR.bmpl (opr18,Di),#opr1i
ln sb xb x2 x1 ASR.bmpl opru18,#opr1i
ln sb xb x3 x2 x1 ASR.bmpl (opr24,xys),#opr1i
ln sb xb x3 x2 x1 ASR.bmpl [opr24,xys],#opr1i
ln sb xb x3 x2 x1 ASR.bmpl (opr24,Di),#opr1i
ln sb xb x3 x2 x1 ASR.bmpl opr24,#opr1i
ln sb xb x3 x2 x1 ASR.bmpl [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 Dd - This field specifies data register Dd (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 Ds - This field specifies data register Ds (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 Dn - This field specifies the number of the data register Dn (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*

**Operation**

If C = 0, then (PC) + REL ⇒ PC

Simple branch

**Syntax Variations**

<table>
<thead>
<tr>
<th>BCC</th>
<th>oprdest</th>
<th>REL</th>
</tr>
</thead>
</table>

**Addressing Modes**

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Detailed Instruction Formats**

**REL**

<table>
<thead>
<tr>
<th>REL_SIZE</th>
<th>7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)</td>
<td>r1</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Test</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Boolean</th>
<th>Test</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>r&gt;m</td>
<td>BGT</td>
<td>2E</td>
<td>Z</td>
<td>(N ^ V) = 0</td>
<td>r≤m</td>
<td>BLE</td>
<td>2F</td>
</tr>
<tr>
<td>r≥m</td>
<td>BGE</td>
<td>2C</td>
<td>N ^ V = 0</td>
<td>r&lt;m</td>
<td>BLT</td>
<td>2D</td>
<td>Signed</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r&gt;m</td>
<td>BNE</td>
<td>26</td>
<td>Signed</td>
</tr>
<tr>
<td>r≤m</td>
<td>BLE</td>
<td>2F</td>
<td>Z</td>
<td>(N ^ V) = 1</td>
<td>r&gt;m</td>
<td>BGT</td>
<td>2E</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLT</td>
<td>2D</td>
<td>N ^ V = 1</td>
<td>r&gt;m</td>
<td>BGE</td>
<td>2C</td>
<td>Signed</td>
</tr>
<tr>
<td>r&gt;m</td>
<td>BHI</td>
<td>22</td>
<td>C</td>
<td>Z = 0</td>
<td>r≤m</td>
<td>BLS</td>
<td>23</td>
</tr>
<tr>
<td>r≥m</td>
<td>BHS/BCC</td>
<td>24</td>
<td>C = 0</td>
<td>r&lt;m</td>
<td>BLO/BCS</td>
<td>25</td>
<td>Unsigned</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r&gt;m</td>
<td>BNE</td>
<td>26</td>
<td>Unsigned</td>
</tr>
<tr>
<td>r≤m</td>
<td>BLS</td>
<td>23</td>
<td>C</td>
<td>Z = 1</td>
<td>r&gt;m</td>
<td>BHI</td>
<td>22</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLO/BCS</td>
<td>25</td>
<td>C = 1</td>
<td>r≥m</td>
<td>BHS/BCC</td>
<td>24</td>
<td>Unsigned</td>
</tr>
<tr>
<td>Carry</td>
<td>BCS</td>
<td>25</td>
<td>C = 1</td>
<td>No Carry</td>
<td>BCC</td>
<td>24</td>
<td>Simple</td>
</tr>
<tr>
<td>Negative</td>
<td>BMI</td>
<td>2B</td>
<td>N = 1</td>
<td>Plus</td>
<td>BPL</td>
<td>2A</td>
<td>Simple</td>
</tr>
<tr>
<td>Overflow</td>
<td>BMI</td>
<td>2B</td>
<td>N = 1</td>
<td>Plus</td>
<td>BPL</td>
<td>2A</td>
<td>Simple</td>
</tr>
<tr>
<td>r=0</td>
<td>BEQ</td>
<td>27</td>
<td>V = 1</td>
<td>No Overflow</td>
<td>BVC</td>
<td>28</td>
<td>Simple</td>
</tr>
<tr>
<td>Always</td>
<td>BRA</td>
<td>20</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
BCLR

Test and Clear Bit

Operation

\[
\text{bit}_n \text{ of } D_i \Rightarrow C; \:\text{then} \:(D_i) \& \sim \text{bit}_n \Rightarrow D_i
\]

\[
\text{bit}_n \text{ of } M \Rightarrow C; \:\text{then} \:(M) \& \sim \text{bit}_n \Rightarrow M
\]

Syntax Variations

<table>
<thead>
<tr>
<th>BCLR</th>
<th>Di,#opr5i</th>
<th>REG-IMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCLR</td>
<td>Di,Dn</td>
<td>REG-REG</td>
</tr>
<tr>
<td>BCLR</td>
<td>bwl oprmemreg,#opr5i</td>
<td>OPR/1/2/3-IMM</td>
</tr>
<tr>
<td>BCLR</td>
<td>bwl oprmemreg,Dn</td>
<td>OPR/1/2/3-REG</td>
</tr>
</tbody>
</table>

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 Di 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 Dn. 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
<td>0</td>
<td>Δ</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

n[4.0] SD REGISTER Di

Linear S12 Core Reference Manual, Rev. 1.01

Freescale Semiconductor
### REG-REG

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>PARAMETER REGISTER Dn</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>SD REGISTER Di</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**EC** bm xb

**BCLR** Di, Dn

<table>
<thead>
<tr>
<th>Byte-sized operand (.B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

**OPR POSTBYTE**

(Optional ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

<table>
<thead>
<tr>
<th>Word-sized operand (.W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

**OPR POSTBYTE**

(Optional ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

<table>
<thead>
<tr>
<th>Long-word sized operand (.L)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

**OPR POSTBYTE**

(Optional ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

**EC** sb xb

**BCLR**.lw1 opru4, #opr5i; not appropriate for destination

**EC** sb xb

**BCLR**.lw1 Di, #opr5i; see more efficient REG-IMM1 version

**EC** sb xb

**BCLR**.lw1 opru4, #opr5i

**EC** sb xb

**BCLR**.lw1 \{(+xy)|(+xy-)|(-s)|(+s}\}, #opr5i

**EC** sb xb

**BCLR**.lw1 (Di, xy), #opr5i

**EC** sb xb

**BCLR**.lw1 [Di, xy], #opr5i

**EC** sb xb x1

**BCLR**.lw1 (opr9, xy), #opr5i

**EC** sb xb x1

**BCLR**.lw1 [opr9, xy], #opr5i

**EC** sb xb x1

**BCLR**.lw1 opru14, #opr5i

**EC** sb xb x2 x1

**BCLR**.lw1 (opr18, Di), #opr5i

**EC** sb xb x2 x1

**BCLR**.lw1 opru18, #opr5i

**EC** sb xb x3 x2 x1

**BCLR**.lw1 (opr24, xy), #opr5i

**EC** sb xb x3 x2 x1

**BCLR**.lw1 [opr24, xy], #opr5i

**EC** sb xb x3 x2 x1

**BCLR**.lw1 opru24, #opr5i

**EC** sb xb x3 x2 x1

**BCLR**.lw1 opr24, #opr5i

**EC** sb xb x3 x2 x1

**BCLR**.lw1 [opr24], #opr5i
Instruction Fields

SD REGISTER Di - This field specifies the number of the data register Di 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 Dn - This field specifies the number of the data register Dn (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

**Operation**

If C = 1, then (PC) + REL \(\Rightarrow\) PC

Simple branch

**Syntax Variations**

<table>
<thead>
<tr>
<th>BCS</th>
<th>oprdest</th>
<th>REL</th>
</tr>
</thead>
</table>

**Addressing Modes**

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Detailed Instruction Formats**

**REL**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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

**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.
<table>
<thead>
<tr>
<th>Test</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Boolean</th>
<th>Test</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>r&gt;m</td>
<td>BGT</td>
<td>2E</td>
<td>Z (N \land V = 0)</td>
<td>r≤m</td>
<td>BLE</td>
<td>2F</td>
<td>Signed</td>
</tr>
<tr>
<td>r≥m</td>
<td>BGE</td>
<td>2C</td>
<td>N \land V = 0</td>
<td>r&lt;m</td>
<td>BLT</td>
<td>2D</td>
<td>Signed</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r&gt;m</td>
<td>BNE</td>
<td>26</td>
<td>Signed</td>
</tr>
<tr>
<td>r≤m</td>
<td>BLE</td>
<td>2F</td>
<td>Z (N \land V = 1)</td>
<td>r&gt;m</td>
<td>BGT</td>
<td>2E</td>
<td>Signed</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLT</td>
<td>2D</td>
<td>N \land V = 1</td>
<td>r≥m</td>
<td>BGE</td>
<td>2C</td>
<td>Signed</td>
</tr>
<tr>
<td>r&gt;m</td>
<td>BHI</td>
<td>22</td>
<td>C \land Z = 0</td>
<td>r≤m</td>
<td>BLS</td>
<td>23</td>
<td>Unsigned</td>
</tr>
<tr>
<td>r≥m</td>
<td>BHS/BCC</td>
<td>24</td>
<td>C = 0</td>
<td>r&lt;m</td>
<td>BLO/BCS</td>
<td>25</td>
<td>Unsigned</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r&gt;m</td>
<td>BNE</td>
<td>26</td>
<td>Unsigned</td>
</tr>
<tr>
<td>r≤m</td>
<td>BLS</td>
<td>23</td>
<td>C \land Z = 1</td>
<td>r&gt;m</td>
<td>BHI</td>
<td>22</td>
<td>Unsigned</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLO/BCS</td>
<td>25</td>
<td>C = 1</td>
<td>r≥m</td>
<td>BHS/BCC</td>
<td>24</td>
<td>Unsigned</td>
</tr>
<tr>
<td>Carry</td>
<td>BCS</td>
<td>25</td>
<td>C = 1</td>
<td>No Carry</td>
<td>BCC</td>
<td>24</td>
<td>Simple</td>
</tr>
<tr>
<td>Negative</td>
<td>BMI</td>
<td>2B</td>
<td>N = 1</td>
<td>Plus</td>
<td>BPL</td>
<td>2A</td>
<td>Simple</td>
</tr>
<tr>
<td>Overflow</td>
<td>BVS</td>
<td>29</td>
<td>V = 1</td>
<td>No Overflow</td>
<td>BVC</td>
<td>28</td>
<td>Simple</td>
</tr>
<tr>
<td>r=0</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r\neq0</td>
<td>BNE</td>
<td>26</td>
<td>Simple</td>
</tr>
<tr>
<td>Always</td>
<td>BRA</td>
<td>20</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
BEQ  \hspace{1cm} \text{Branch if Equal}  \hspace{1cm} \text{BEQ}

\textbf{Operation}

If \( Z = 1 \), then \((PC) + \text{REL} \Rightarrow PC\)

Simple branch

\textbf{Syntax Variations} \hspace{1cm} \textbf{Addressing Modes}

\begin{tabular}{l|l}
BEQ & oprdest & REL \\
\end{tabular}

\textbf{Description}

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

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

\textbf{CCR Details}

\begin{tabular}{cccccccccccc}
\hline
\end{tabular}

\textbf{Detailed Instruction Formats}

\textbf{REL}

\begin{tabular}{cccccccc}
7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\hline
0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
\end{tabular}

\begin{tabular}{lll}
REL_SIZE & 7 \text{ bit DISPLACEMENT (REL_SIZE==0)} \text{ or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)} & 27 \\
\hline
Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1) & rb & \\
r1 & & \\
\end{tabular}

\begin{tabular}{lll}
27 & rb & BEQ \hspace{0.5cm} oprdest \hspace{0.5cm}; \text{Dest is within +63/-64 (7-bit offset)} \\
27 & rb & r1 & BEQ \hspace{0.5cm} oprdest \hspace{0.5cm}; \text{Dest is within \sim +/–16K (15-bit offset)} \\
\end{tabular}

\textbf{Instruction Fields}

REL_SIZE - This field specifies the size of the DISPLACEMENT \( 0=7\text{-bit}; \ 1=15\text{-bit} \).

DISPLACEMENT - This field specifies the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.
<table>
<thead>
<tr>
<th>Test</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Boolean</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>r&gt;m</td>
<td>BGT</td>
<td>2E</td>
<td>Z \cdot (N \cdot V) = 0</td>
<td>Signed</td>
</tr>
<tr>
<td>r\geq m</td>
<td>BGE</td>
<td>2C</td>
<td>N \cdot V = 0</td>
<td>Signed</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>Signed</td>
</tr>
<tr>
<td>r\leq m</td>
<td>BLE</td>
<td>2F</td>
<td>Z \cdot (N \cdot V) = 1</td>
<td>Signed</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLT</td>
<td>2D</td>
<td>N \cdot V = 1</td>
<td>Signed</td>
</tr>
<tr>
<td>r&gt;m</td>
<td>BHI</td>
<td>22</td>
<td>C \cdot Z = 0</td>
<td>Unsigned</td>
</tr>
<tr>
<td>r\geq m</td>
<td>BHS/BCC</td>
<td>24</td>
<td>C = 0</td>
<td>Unsigned</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>Unsigned</td>
</tr>
<tr>
<td>r\leq m</td>
<td>BLS</td>
<td>23</td>
<td>C \cdot Z = 1</td>
<td>Unsigned</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLO/BCS</td>
<td>25</td>
<td>C = 1</td>
<td>Unsigned</td>
</tr>
<tr>
<td>Carry</td>
<td>BCS</td>
<td>25</td>
<td>C = 1</td>
<td>Simple</td>
</tr>
<tr>
<td>Negative</td>
<td>BMI</td>
<td>2B</td>
<td>N = 1</td>
<td>Simple</td>
</tr>
<tr>
<td>Overflow</td>
<td>BVS</td>
<td>29</td>
<td>V = 1</td>
<td>Simple</td>
</tr>
<tr>
<td>r=0</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>Simple</td>
</tr>
<tr>
<td>Always</td>
<td>BRA</td>
<td>20</td>
<td>——</td>
<td>——</td>
</tr>
</tbody>
</table>
BFEXT  

**Bit Field Extract**

**Description**

Extracts a bit field from the specified source (register Ds or memory location), if necessary zero extends to the width of the destination, and stores the result to the destination (register Dd or memory location). The bit field width and offset are specified in the parameter (register Dp 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 Ds.

**CCR Details**

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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>DESTINATION REGISTER Dd</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1B 0q bb

BFEXT  Dd, Ds, Dp

**REG-REG-IMM**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>DESTINATION REGISTER Dd</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1B 0q bb

BFEXT  Dd, Ds, #width:offset

**Syntax Variations**

<table>
<thead>
<tr>
<th>BFEXT</th>
<th>Destination-Source-Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFEXT</td>
<td>Dd, Ds, Dp</td>
</tr>
<tr>
<td>BFEXT</td>
<td>Dd, Ds, #width:offset</td>
</tr>
<tr>
<td>BFEXT.bwp1Dd, oprmemreg, Dp</td>
<td>REG-OPR/1/2/3-REG</td>
</tr>
<tr>
<td>BFEXT.bwp1oprmemreg, Ds, Dp</td>
<td>REG-OPR/1/2/3-REG-REG</td>
</tr>
<tr>
<td>BFEXT.bwp1Dd, oprmemreg, #width:offset</td>
<td>REG-OPR/1/2/3-IMM</td>
</tr>
<tr>
<td>BFEXT.bwp1oprmemreg, Ds, #width:offset</td>
<td>OPR/1/2/3-REG-REG</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>U</th>
<th>I</th>
<th>P</th>
<th>L</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>Δ</td>
<td>Δ</td>
<td>0</td>
</tr>
</tbody>
</table>
### REG-OPR/1/2/3-REG

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**OPR POSTBYTE (specifies source)**

- (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)
- (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

1B 0q bb xb

\[\text{BFEXT.bpl} \quad \text{Dd, oprsxe4i, Dp} \quad \text{not appropriate for destination} \]

### OPR/1/2/3-REG-REG

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**OPR POSTBYTE (specifies destination)**

- (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)
- (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

1B 0q bb xb

\[\text{BFEXT.bpl} \quad \text{Dd, Ds, Dp} \quad \text{see more efficient REG-REG-REG version} \]

1B 0q bb xb

\[\text{BFEXT.bpl} \quad \text{Dd, (opru4, yxs), Dp} \]

1B 0q bb xb

\[\text{BFEXT.bpl} \quad \text{Dd, (-xy) | (xy-+)} \quad \text{Ds, Dp} \]

1B 0q bb xb

\[\text{BFEXT.bpl} \quad \text{Dd, [opru4, yxs], Dp} \]

1B 0q bb xb

\[\text{BFEXT.bpl} \quad \text{Dd, opru14, Dp} \]

1B 0q bb xb x1

\[\text{BFEXT.bpl} \quad \text{Dd, (opru18, Di), Dp} \]

1B 0q bb xb x1

\[\text{BFEXT.bpl} \quad \text{Dd, opru18, Dp} \]

1B 0q bb xb x1

\[\text{BFEXT.bpl} \quad \text{Dd, (opr24, yxs), Dp} \]

1B 0q bb xb x1

\[\text{BFEXT.bpl} \quad \text{Dd, [opr24, yxs], Dp} \]

1B 0q bb xb x1

\[\text{BFEXT.bpl} \quad \text{Dd, (opr24, Di), Dp} \]

1B 0q bb xb x1

\[\text{BFEXT.bpl} \quad \text{Dd, opr24, Dp} \]

1B 0q bb xb x1

\[\text{BFEXT.bpl} \quad \text{Dd, [opr24], Dp} \]
REG-OPR/1/2/3-IMM

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>SIZE (.B,.W,.P,.L)</td>
<td>WIDTH[4:3]</td>
<td></td>
</tr>
</tbody>
</table>

WIDTH[2:0] OFFSET[4:0]

OPR POSTBYTE (specifies source)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

1B 0q bb i1 xb BFEXT.bpl Dd,#oprse4i,#width:offset ;-1, +1, 2, 3...14, 15
1B 0q bb i1 xb BFEXT.bpl Dd,Ds,Dp ;see more efficient REG-REG-REG version
1B 0q bb i1 xb BFEXT.bpl Dd,(opru4,xys),#width:offset
1B 0q bb i1 xb x1 BFEXT.bpl Dd,(opr9,xys),#width:offset
1B 0q bb i1 xb BFEXT.bpl Dd,(Di,xys),#width:offset
1B 0q bb i1 xb x1 BFEXT.bpl Dd,opr14,#width:offset
1B 0q bb i1 xb x2 x1 BFEXT.bpl Dd,(opru18,Di),#width:offset
1B 0q bb i1 xb x3 x2 x1 BFEXT.bpl Dd,opr24,#width:offset
1B 0q bb i1 xb BFEXT.bpl [Di,xy],Ds,#width:offset
1B 0q bb i1 xb BFEXT.bpl [opr9,xys],Ds,#width:offset
1B 0q bb i1 xb BFEXT.bpl opr14,Ds,#width:offset
1B 0q bb i1 xb x1 BFEXT.bpl opru18,Di),Ds,#width:offset
1B 0q bb i1 xb x3 x2 x1 BFEXT.bpl [opr24,xys],Ds,#width:offset
1B 0q bb i1 xb x3 x2 x1 BFEXT.bpl [opr24,xys],Ds,#width:offset
1B 0q bb i1 xb BFEXT.bpl opru24,Di),Ds,#width:offset

OPR/1/2/3-REG-IMM

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>SIZE (.B,.W,.P,.L)</td>
<td>WIDTH[4:3]</td>
</tr>
</tbody>
</table>

WIDTH[2:0] OFFSET[4:0]

OPR POSTBYTE (specifies destination)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

1B 0q bb i1 xb BFEXT.bpl #oprse4i,Di,#width:offset ;don’t use for dest
1B 0q bb i1 xb BFEXT.bpl Dd,Di,#width:offset ;REG-REG-IMM more efficient
1B 0q bb i1 xb BFEXT.bpl (opr4,xys),Ds,#width:offset
1B 0q bb i1 xb BFEXT.bpl ((-xy) | (xy+-) | (-s) | (s+)),Ds,#width:offset
1B 0q bb i1 xb BFEXT.bpl (Di,xys),Ds,#width:offset
1B 0q bb i1 xb BFEXT.bpl [Di,xy],Ds,#width:offset
1B 0q bb i1 xb BFEXT.bpl (opr9,xys),Ds,#width:offset
1B 0q bb i1 xb BFEXT.bpl opru14,Ds,#width:offset
1B 0q bb i1 xb BFEXT.bpl opru18,Di),Ds,#width:offset
1B 0q bb i1 xb BFEXT.bpl [opr24,xys],Ds,#width:offset
1B 0q bb i1 xb BFEXT.bpl [opr24,xys],Ds,#width:offset
1B 0q bb i1 xb BFEXT.bpl opru24,Di),Ds,#width:offset

Linear S12 Core Reference Manual, Rev. 1.01
Instruction Fields

DESTINATION REGISTER Dd - This field specifies the number of the data register Dd 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 Ds - This field specifies the number of the data register Ds 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 Dp - 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

### Description
Inserts a bit field of specified width from the low-order bits of a specified source (register Ds or memory location), into the destination (register Dd or memory location), beginning at the specified offset. The bit field width and offset are specified in the parameter (register Dp 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Δ</td>
<td>Δ</td>
<td>0</td>
</tr>
</tbody>
</table>

- **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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

#### REG-REG-IMM

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

## Syntax Variations

<table>
<thead>
<tr>
<th>Destination-Source-Parameter</th>
<th>REG-REG-REG</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFINS Dd, Ds, Dp</td>
<td>REG-REG-REG</td>
</tr>
<tr>
<td>BFINS Dd, Ds, #width:offset</td>
<td>REG-REG-IMM</td>
</tr>
<tr>
<td>BFINS.bwpIdDd, oprmemreg, Dp</td>
<td>REG-OPR/1/2/3-REG</td>
</tr>
<tr>
<td>BFINS.bwploprmemreg, Ds, Dp</td>
<td>OPR/1/2/3-REG-REG</td>
</tr>
<tr>
<td>BFINS.bwpIdDd, oprmemreg, #width:offset</td>
<td>REG-OPR/1/2/3-IMM</td>
</tr>
<tr>
<td>BFINS.bwploprmemreg, Ds, #width:offset</td>
<td>OPR/1/2/3-REG-IMM</td>
</tr>
</tbody>
</table>

### CCR Details

- **U**:
- **P**:
- **L**:
- **S**:
- **X**:
- **I**:
- **N**:
- **Z**:
- **V**:
- **C**:

- **U**: Always 0.
- **P**: Always 0.
- **L**: Always 0.
- **S**: Always 0.
- **X**: Always 0.
- **I**: Always 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.
- **C**: Always 0.
### REG-OPR/1/2/3-REG

|   |   |   |   |   |   |   |   |   |   
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | DESTINATION REGISTER Dd |
| 1 | 1 | 0 | 0 | 0 | SIZE (B, W, P, L) PARAMETER REG Dp |

**OPR POSTBYTE** (specifies source)

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1B 0q bb xb</td>
<td>BFINS.bpl Dd, #opr+rsxe4i, Dp ; -1, +1, 2, 3...14, 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B 0q bb xb</td>
<td>BFINS.bpl Dd, Ds, Dp ; see more efficient REG-REG-REG version</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B 0q bb xb</td>
<td>BFINS.bpl Dd, (opru4, xys), Dp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B 0q bb xb</td>
<td>BFINS.bpl Dd, (sxy)</td>
<td>(xyz)</td>
<td>(sxy)</td>
<td>(sxy)</td>
<td>Dp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B 0q bb xb</td>
<td>BFINS.bpl Dd, opru14, Dp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B 0q bb xb x1</td>
<td>BFINS.bpl Dd, (opru18, Di), Dp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B 0q bb xb x2 x1</td>
<td>BFINS.bpl Dd, opru24, Dp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B 0q bb xb x3 x2 x1</td>
<td>BFINS.bpl Dd, [opr24, Dd, Dp]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**OPR/1/2/3-REG-REG**

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>SOURCE REGISTER Ds</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>SIZE (B, W, P, L) PARAMETER REG Dp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**OPR POSTBYTE** (specifies destination)

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1B 0q bb xb</td>
<td>BFINS.bpl #opr+rsxe4i, Ds, Dp ; not appropriate for destination</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B 0q bb xb</td>
<td>BFINS.bpl Dd, Ds, Dp ; see more efficient REG-REG-REG version</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B 0q bb xb</td>
<td>BFINS.bpl (opru4, xys), Ds, Dp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B 0q bb xb</td>
<td>BFINS.bpl (sxy)</td>
<td>(xyz)</td>
<td>(sxy)</td>
<td>(sxy)</td>
<td>Ds, Dp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B 0q bb xb</td>
<td>BFINS.bpl (Di, xys), Ds, Dp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B 0q bb xb</td>
<td>BFINS.bpl [Di, xys], Ds, Dp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B 0q bb xb x1</td>
<td>BFINS.bpl [opr9, xys], Ds, Dp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B 0q bb xb x2 x1</td>
<td>BFINS.bpl opru14, Ds, Dp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B 0q bb xb x3 x2 x1</td>
<td>BFINS.bpl opru18, Ds, Dp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B 0q bb xb x3 x2 x1</td>
<td>BFINS.bpl [opr24, Ds, Dp]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Linear S12 Core Reference Manual, Rev. 1.01

Freescale Semiconductor
### REG-OPR/1/2/3-IMM

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1B</th>
<th>0q</th>
<th>bb</th>
<th>i1</th>
<th>xb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>#</td>
<td>DESTINATION REGISTER Dd</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>SIZE (.B, .W, .P, .L)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OFFSET[4:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OPR POSTBYTE (specifies source)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

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

### OPR/1/2/3-REG-IMM

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1B</th>
<th>0q</th>
<th>bb</th>
<th>i1</th>
<th>xb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>#</td>
<td>SOURCE REGISTER Ds</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>SIZE (.B, .W, .P, .L)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OFFSET[4:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OPR POSTBYTE (specifies destination)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

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

---

Linear S12 Core Reference Manual, Rev. 1.01

Freescale Semiconductor
Instruction Fields

DESTINATION REGISTER D\textsubscript{d} - This field specifies the number of the data register D\textsubscript{d} used for the destination (0:0:0=D\textsubscript{2}, 0:0:1=D\textsubscript{3}, 0:1:0=D\textsubscript{4}, 0:1:1=D\textsubscript{5}, 1:0:0=D\textsubscript{0}, 1:0:1=D\textsubscript{1}, 1:1:0=D\textsubscript{6}, and 1:1:1=D\textsubscript{7}).

SOURCE REGISTER D\textsubscript{s} - This field specifies the number of the data register D\textsubscript{s} used for the source operand (0:0:0=D\textsubscript{2}, 0:0:1=D\textsubscript{3}, 0:1:0=D\textsubscript{4}, 0:1:1=D\textsubscript{5}, 1:0:0=D\textsubscript{0}, 1:0:1=D\textsubscript{1}, 1:1:0=D\textsubscript{6}, and 1:1:1=D\textsubscript{7}).

PARAMETER REG D\textsubscript{p} - This field specifies the number of the 16 bit data register which contains both width and offset parameters for the operation (0b00 = D\textsubscript{2}, 0b01 = D\textsubscript{3}, 0b10 = D\textsubscript{4}, and 0b11 = D\textsubscript{5}). 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)*

**Operation**

If \( N \land V = 0 \), then \((PC) + REL \Rightarrow PC\)

For signed two’s complement values

if \((\text{Accumulator}) \geq (\text{Memory})\), then branch

**Syntax Variations**

<table>
<thead>
<tr>
<th>BGE</th>
<th>oprdest</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td></td>
</tr>
</tbody>
</table>

**Addressing Modes**

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Detailed Instruction Formats**

**REL**

<table>
<thead>
<tr>
<th>REL_SIZE</th>
<th>7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REL_SIZE</th>
<th>DISPLACEMENT (REL_SIZE==0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rb</td>
</tr>
<tr>
<td>1</td>
<td>rb r1</td>
</tr>
</tbody>
</table>

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

Linear S12 Core Reference Manual, Rev. 1.01

Freescale Semiconductor
BGND  Enter Background Debug Mode

Operation
Enter Active Background Mode

Syntax Variations

<table>
<thead>
<tr>
<th>Syntax Variations</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGND</td>
</tr>
</tbody>
</table>

Addressing Modes

<table>
<thead>
<tr>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Detailed Instruction Formats

INH

<table>
<thead>
<tr>
<th>INH</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

00

00

BGND
BGT  
Branch if Greater Than
(Signed Branch)

Operation
If \( Z | (N \land V) = 0 \), then \((PC) + \text{REL} \Rightarrow PC\)
For signed two’s complement values
if \((\text{Accumulator}) > (\text{Memory})\), then branch

Syntax Variations

<table>
<thead>
<tr>
<th>BGT</th>
<th>oprdest</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td></td>
</tr>
</tbody>
</table>

Addressing Modes

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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Detailed Instruction Formats

REL

<table>
<thead>
<tr>
<th>REL_SIZE</th>
<th>7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)</th>
<th>Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 1 1 1 1 1 1 0</td>
<td>2E rb</td>
</tr>
<tr>
<td>2E rb</td>
<td>BGT ( \text{oprdest} ); Dest is within (+63/-64) (7-bit offset)</td>
<td></td>
</tr>
<tr>
<td>2E rb r1</td>
<td>BGT ( \text{oprdest} ); Dest is within (-16K/+16K) (15-bit offset)</td>
<td></td>
</tr>
</tbody>
</table>

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

**Branch if Higher**

(Unsigned Branch)

### Operation

If C | Z = 0, then (PC) + REL ⇒ PC

For unsigned values

if (Accumulator) > (Memory), then branch

### Syntax Variations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHI oprdest</td>
<td>REL</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td>-</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### Detailed Instruction Formats

REL

<table>
<thead>
<tr>
<th>REL_SIZE</th>
<th>DISPLACEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7 bit</td>
</tr>
<tr>
<td>1</td>
<td>high-order 7 bits</td>
</tr>
</tbody>
</table>

Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)

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

Linear S12 Core Reference Manual, Rev. 1.01

Freescale Semiconductor 119
BHS

Branch if Higher or Same
(Unsigned Branch; Same as BCC)

Operation
If C = 0, then (PC) + REL ⇒ PC
For unsigned values
if (Accumulator) ≥ (Memory), then branch

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

Detailed Instruction Formats

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

**Bit Test**

**Operation**

(Di) & (M)

**Syntax Variations**

<table>
<thead>
<tr>
<th>BIT</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Di,#oprimsz</td>
<td>IMM1/2/4</td>
</tr>
<tr>
<td>Di,oprmemreg</td>
<td>OPR/1/2/3</td>
</tr>
</tbody>
</table>

**Description**

Bitwise AND register Di 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 Di. 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 Di at a 14- 18- or 24-bit extended address, or a memory operand that is addressed with indexed or indirect addressing mode.

**CCR Details**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
<td>0</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1B</th>
<th>5p</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>SD REG D4</td>
<td></td>
</tr>
</tbody>
</table>

**IMMEDIATE DATA**

OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF Di

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

Linear S12 Core Reference Manual, Rev. 1.01

122 Freescale Semiconductor
**Instruction Fields**

SD REGISTER Di - This field specifies the number of the data register Di 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 Di.

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.
Branch if Less Than or Equal (Signed Branch)

**Operation**

If \( Z \land (N \lor V) = 1 \), then \((PC) + REL \Rightarrow PC\)

For signed two’s complement values
if \((\text{Accumulator}) \leq (\text{Memory})\), then branch

**Syntax Variations**

<table>
<thead>
<tr>
<th>REL</th>
<th>oprdest</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLE</td>
<td>REL</td>
</tr>
</tbody>
</table>

**Addressing Modes**

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Detailed Instruction Formats**

**REL**

```
<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REL_SIZE 7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)</td>
<td></td>
</tr>
<tr>
<td>2F</td>
<td>rb</td>
<td>BLE</td>
<td>oprdest</td>
<td>;Dest is within +63/-64 (7-bit offset)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2F</td>
<td>rb</td>
<td>r1</td>
<td>BLE</td>
<td>oprdest ;Dest is within ~ +/-16K (15-bit offset)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**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.
<table>
<thead>
<tr>
<th>Test</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Boolean</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r &gt; m$</td>
<td>BGT</td>
<td>2E</td>
<td>$Z \land (N \land V) = 0$</td>
<td>Signed</td>
</tr>
<tr>
<td>$r \geq m$</td>
<td>BGE</td>
<td>2C</td>
<td>$N \land V = 0$</td>
<td>Signed</td>
</tr>
<tr>
<td>$r = m$</td>
<td>BEQ</td>
<td>27</td>
<td>$Z = 1$</td>
<td>Signed</td>
</tr>
<tr>
<td>$r \leq m$</td>
<td>BLE</td>
<td>2F</td>
<td>$Z \land (N \land V) = 1$</td>
<td>Signed</td>
</tr>
<tr>
<td>$r &lt; m$</td>
<td>BLT</td>
<td>2D</td>
<td>$N \land V = 1$</td>
<td>Signed</td>
</tr>
<tr>
<td>$r &gt; m$</td>
<td>BHI</td>
<td>22</td>
<td>$C \land Z = 0$</td>
<td>Unsigned</td>
</tr>
<tr>
<td>$r = m$</td>
<td>BHS/BCC</td>
<td>24</td>
<td>$C = 0$</td>
<td>Unsigned</td>
</tr>
<tr>
<td>$r &lt; m$</td>
<td>BLO/BCS</td>
<td>25</td>
<td>$C = 1$</td>
<td>Unsigned</td>
</tr>
<tr>
<td>Carry</td>
<td>BCS</td>
<td>25</td>
<td>$C = 1$</td>
<td>Simple</td>
</tr>
<tr>
<td>Negative</td>
<td>BMI</td>
<td>2B</td>
<td>$N = 1$</td>
<td>Simple</td>
</tr>
<tr>
<td>Overflow</td>
<td>BVS</td>
<td>29</td>
<td>$V = 1$</td>
<td>Simple</td>
</tr>
<tr>
<td>$r = 0$</td>
<td>BEQ</td>
<td>27</td>
<td>$Z = 1$</td>
<td>Simple</td>
</tr>
<tr>
<td>Always</td>
<td>BRA</td>
<td>20</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
**BLO**

Branch if Lower  
(Unsigned Branch; same as BCS)

**Operation**
If \(C = 1\), then \((PC) + REL \Rightarrow PC\)
For unsigned values
if \((\text{Accumulator}) < (\text{Memory})\), then branch

**Syntax Variations**

<table>
<thead>
<tr>
<th>BLO</th>
<th>oprdest</th>
<th>REL</th>
</tr>
</thead>
</table>

**Addressing Modes**

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Detailed Instruction Formats**

**REL**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

REL_SIZE - This field specifies the size of the DISPLACEMENT \(0=7\text{-bit}; 1=15\text{-bit}\).

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

**Instruction Fields**

<p>| 25 rb | BLO | oprdest ; Dest is within +63/-64 (7-bit offset) |
| 25 rb r1 | BLO | oprdest ; Dest is within ~ +/-16K (15-bit offset) |</p>
<table>
<thead>
<tr>
<th>Branch</th>
<th>Complementary Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>Mnemonic</td>
</tr>
<tr>
<td>r&gt;m</td>
<td>BGT</td>
</tr>
<tr>
<td>r≥m</td>
<td>BGE</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
</tr>
<tr>
<td>r≤m</td>
<td>BLE</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLT</td>
</tr>
<tr>
<td>r&gt;m</td>
<td>BHI</td>
</tr>
<tr>
<td>r≥m</td>
<td>BHS/BCC</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
</tr>
<tr>
<td>r≤m</td>
<td>BLS</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLO/BCS</td>
</tr>
<tr>
<td>Carry</td>
<td>BCS</td>
</tr>
<tr>
<td>Negative</td>
<td>BMI</td>
</tr>
<tr>
<td>Overflow</td>
<td>BVS</td>
</tr>
<tr>
<td>r=0</td>
<td>BEQ</td>
</tr>
<tr>
<td>Always</td>
<td>BRA</td>
</tr>
</tbody>
</table>
BLS

Branch if Lower or Same
(Unsigned Branch)

Operation
If \( Z \mid C = 1 \), then \((PC) + REL \Rightarrow PC\)
For unsigned values
if \((\text{Accumulator}) \leq (\text{Memory})\), then branch

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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Detailed Instruction Formats

REL

<table>
<thead>
<tr>
<th>REL SIZE</th>
<th>7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1) r1

<table>
<thead>
<tr>
<th>REL SIZE</th>
<th>7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Test</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Boolean</th>
<th>Test</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>r&gt;m</td>
<td>BGT</td>
<td>2E</td>
<td>Z</td>
<td>(N ^ V) = 0</td>
<td>r≤m</td>
<td>BLE</td>
<td>2F</td>
</tr>
<tr>
<td>r≥m</td>
<td>BGE</td>
<td>2C</td>
<td>N ^ V = 0</td>
<td>r&lt;m</td>
<td>BLT</td>
<td>2D</td>
<td>Signed</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r&gt;m</td>
<td>BNE</td>
<td>26</td>
<td>Signed</td>
</tr>
<tr>
<td>r≤m</td>
<td>BLE</td>
<td>2F</td>
<td>Z</td>
<td>(N ^ V) = 1</td>
<td>r&gt;m</td>
<td>BGT</td>
<td>2E</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLT</td>
<td>2D</td>
<td>N ^ V = 1</td>
<td>r&gt;m</td>
<td>BGE</td>
<td>2C</td>
<td>Signed</td>
</tr>
<tr>
<td>r&gt;m</td>
<td>BHI</td>
<td>22</td>
<td>C</td>
<td>Z = 0</td>
<td>r≤m</td>
<td>BLS</td>
<td>23</td>
</tr>
<tr>
<td>r≥m</td>
<td>BHS/BCC</td>
<td>24</td>
<td>C = 0</td>
<td>r&lt;m</td>
<td>BLO/BCS</td>
<td>25</td>
<td>Unsigned</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r&gt;m</td>
<td>BNE</td>
<td>26</td>
<td>Unsigned</td>
</tr>
<tr>
<td>r≤m</td>
<td>BLS</td>
<td>23</td>
<td>C</td>
<td>Z = 1</td>
<td>r&gt;m</td>
<td>BHI</td>
<td>22</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLO/BCS</td>
<td>25</td>
<td>C = 1</td>
<td>r≥m</td>
<td>BHS/BCC</td>
<td>24</td>
<td>Unsigned</td>
</tr>
<tr>
<td>Carry</td>
<td>BCS</td>
<td>25</td>
<td>C = 1</td>
<td>No Carry</td>
<td>BCC</td>
<td>24</td>
<td>Simple</td>
</tr>
<tr>
<td>Negative</td>
<td>BMI</td>
<td>2B</td>
<td>N = 1</td>
<td>Plus</td>
<td>BPL</td>
<td>2A</td>
<td>Simple</td>
</tr>
<tr>
<td>Overflow</td>
<td>BVS</td>
<td>29</td>
<td>V = 1</td>
<td>No Overflow</td>
<td>BVC</td>
<td>28</td>
<td>Simple</td>
</tr>
<tr>
<td>r=0</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r≠0</td>
<td>BNE</td>
<td>26</td>
<td>Simple</td>
</tr>
<tr>
<td>Always</td>
<td>BRA</td>
<td>20</td>
<td>——</td>
<td>——</td>
<td>——</td>
<td>——</td>
<td>——</td>
</tr>
</tbody>
</table>
Chapter 6 Instruction Glossary

BLT
Branch if Less Than
(Signed Branch)

Operation
If \( N \land V = 1 \), then \( (PC) + REL \Rightarrow PC \)
For signed two’s complement values
if \((\text{Accumulator}) < (\text{Memory})\), then branch

Syntax Variations

<table>
<thead>
<tr>
<th>BLT</th>
<th>oprdest</th>
<th>REL</th>
</tr>
</thead>
</table>

Addressing Modes

Description
BLT 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Detailed Instruction Formats

REL

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REL_SIZE</td>
<td>7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)</td>
<td>rb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)</td>
<td>r1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Test</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Boolean</th>
<th>Test</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>r&gt;m</td>
<td>BGT</td>
<td>2E</td>
<td>Z</td>
<td>(N ^ V) = 0</td>
<td>≤m</td>
<td>BLE</td>
<td>2F</td>
</tr>
<tr>
<td>r≥m</td>
<td>BGE</td>
<td>2C</td>
<td>N ^ V = 0</td>
<td>&lt;m</td>
<td>BLT</td>
<td>2D</td>
<td>Signed</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>≥m</td>
<td>BNE</td>
<td>26</td>
<td>Signed</td>
</tr>
<tr>
<td>≤m</td>
<td>BLE</td>
<td>2F</td>
<td>Z</td>
<td>(N ^ V) = 1</td>
<td>&gt;m</td>
<td>BGT</td>
<td>2E</td>
</tr>
<tr>
<td>≤m</td>
<td>BLT</td>
<td>2D</td>
<td>N ^ V = 1</td>
<td>≥m</td>
<td>BGE</td>
<td>2C</td>
<td>Signed</td>
</tr>
<tr>
<td>r&gt;m</td>
<td>BHI</td>
<td>22</td>
<td>C</td>
<td>Z = 0</td>
<td>≤m</td>
<td>BLS</td>
<td>23</td>
</tr>
<tr>
<td>r≥m</td>
<td>BHS/BCC</td>
<td>24</td>
<td>C = 0</td>
<td>&lt;m</td>
<td>BLO/BCS</td>
<td>25</td>
<td>Unsigned</td>
</tr>
<tr>
<td>≤m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>≥m</td>
<td>BNE</td>
<td>26</td>
<td>Unsigned</td>
</tr>
<tr>
<td>&lt;m</td>
<td>BLS</td>
<td>23</td>
<td>C</td>
<td>Z = 1</td>
<td>&gt;m</td>
<td>BHI</td>
<td>22</td>
</tr>
<tr>
<td>≤m</td>
<td>BLO/BCS</td>
<td>25</td>
<td>C = 1</td>
<td>≥m</td>
<td>BHS/BCC</td>
<td>24</td>
<td>Unsigned</td>
</tr>
<tr>
<td>Carry</td>
<td>BCS</td>
<td>25</td>
<td>C = 1</td>
<td>No Carry</td>
<td>BCC</td>
<td>24</td>
<td>Simple</td>
</tr>
<tr>
<td>Negative</td>
<td>BMI</td>
<td>2B</td>
<td>N = 1</td>
<td>No</td>
<td>BPE</td>
<td>2A</td>
<td>Simple</td>
</tr>
<tr>
<td>Overflow</td>
<td>BVS</td>
<td>29</td>
<td>V = 1</td>
<td>No Overflow</td>
<td>BVC</td>
<td>28</td>
<td>Simple</td>
</tr>
<tr>
<td>≤0</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>≥0</td>
<td>BNE</td>
<td>26</td>
<td>Simple</td>
</tr>
<tr>
<td>Always</td>
<td>BRA</td>
<td>20</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
**BMI**  
**Branch if Minus**  

**Operation**  
If \( N = 1 \), then \( (PC) + REL \Rightarrow PC \)  
Simple branch

**Syntax Variations**

<table>
<thead>
<tr>
<th>BMI</th>
<th>oprdest</th>
<th>REL</th>
</tr>
</thead>
</table>

**Addressing Modes**

**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**

<p>| U - - - - | IPL | S | X | - | I | N | Z | V | C |</p>
<table>
<thead>
<tr>
<th>-----------</th>
<th>-----</th>
<th>---</th>
<th>---</th>
<th>---</th>
<th>---</th>
<th>---</th>
<th>---</th>
<th>---</th>
<th>---</th>
<th>---</th>
</tr>
</thead>
</table>

**Detailed Instruction Formats**

**REL**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**REL_SIZE**  
7 bit DISPLACEMENT \((REL\_SIZE==0)\) or high-order 7 bits of 15 bit DISPLACEMENT \((REL\_SIZE==1)\)

Optional low-order 8 bits of 15-bit DISPLACEMENT \((REL\_SIZE==1)\)

**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.
<table>
<thead>
<tr>
<th>Test</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Boolean</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>r&gt;m</td>
<td>BGT</td>
<td>2E</td>
<td>Z</td>
<td>(N ^ V) = 0</td>
</tr>
<tr>
<td>r≥m</td>
<td>BGE</td>
<td>2C</td>
<td>N ^ V = 0</td>
<td>r&lt;m</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r&gt;m</td>
</tr>
<tr>
<td>r≤m</td>
<td>BLE</td>
<td>2F</td>
<td>Z</td>
<td>(N ^ V) = 1</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLT</td>
<td>2D</td>
<td>N ^ V = 1</td>
<td>r≥m</td>
</tr>
<tr>
<td>r&gt;m</td>
<td>BHI</td>
<td>22</td>
<td>C</td>
<td>Z = 0</td>
</tr>
<tr>
<td>r≥m</td>
<td>BHS/BCC</td>
<td>24</td>
<td>C = 0</td>
<td>r&lt;m</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r&lt;m</td>
</tr>
<tr>
<td>r≤m</td>
<td>BLS</td>
<td>23</td>
<td>C</td>
<td>Z = 1</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLO/BCS</td>
<td>25</td>
<td>C = 1</td>
<td>r≥m</td>
</tr>
<tr>
<td>Carry</td>
<td>BCS</td>
<td>25</td>
<td>C = 1</td>
<td>No Carry</td>
</tr>
<tr>
<td>Negative</td>
<td>BMI</td>
<td>2B</td>
<td>N = 1</td>
<td>Plus</td>
</tr>
<tr>
<td>Overflow</td>
<td>BVS</td>
<td>29</td>
<td>N = 0</td>
<td>No Overflow</td>
</tr>
<tr>
<td>r=0</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r≠0</td>
</tr>
<tr>
<td>Always</td>
<td>BRA</td>
<td>20</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
BNE  
Branch if Not Equal  

Operation
If Z = 0, then (PC) + REL ⇒ PC
Simple branch

Syntax Variations

<table>
<thead>
<tr>
<th>BNE</th>
<th>oprdest</th>
<th>REL</th>
</tr>
</thead>
</table>

Addressing Modes

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

```
<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
```

Detailed Instruction Formats

**REL**

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

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.
```

Instruction Fields

26 rb  
BNE oprdest ;Dest is within +63/-64 (7-bit offset)

26 rb r1  
BNE oprdest ;Dest is within ~ +/-16K (15-bit offset)
<table>
<thead>
<tr>
<th>Test</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Boolean</th>
<th>Test</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>r&gt;m</td>
<td>BGT</td>
<td>2E</td>
<td>Z</td>
<td>(N ^ V) = 0</td>
<td>r≤m</td>
<td>BLE</td>
<td>2F</td>
</tr>
<tr>
<td>r≥m</td>
<td>BGE</td>
<td>2C</td>
<td>N ^ V = 0</td>
<td>r&lt;m</td>
<td>BLT</td>
<td>2D</td>
<td>Signed</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r&gt;m</td>
<td>BNE</td>
<td>26</td>
<td>Signed</td>
</tr>
<tr>
<td>r≤m</td>
<td>BLE</td>
<td>2F</td>
<td>Z</td>
<td>(N ^ V) = 1</td>
<td>r&gt;m</td>
<td>BGT</td>
<td>2E</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLT</td>
<td>2D</td>
<td>N ^ V = 1</td>
<td>r≥m</td>
<td>BGE</td>
<td>2C</td>
<td>Signed</td>
</tr>
<tr>
<td>r&gt;m</td>
<td>BHI</td>
<td>22</td>
<td>C</td>
<td>Z = 0</td>
<td>r≤m</td>
<td>BLS</td>
<td>23</td>
</tr>
<tr>
<td>r≥m</td>
<td>BHS/BCC</td>
<td>24</td>
<td>C = 0</td>
<td>r&lt;m</td>
<td>BLO/BCS</td>
<td>25</td>
<td>Unsigned</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r&gt;m</td>
<td>BNE</td>
<td>26</td>
<td>Unsigned</td>
</tr>
<tr>
<td>r≤m</td>
<td>BLS</td>
<td>23</td>
<td>C</td>
<td>Z = 1</td>
<td>r&gt;m</td>
<td>BHI</td>
<td>22</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLO/BCS</td>
<td>25</td>
<td>C = 1</td>
<td>r≥m</td>
<td>BHS/BCC</td>
<td>24</td>
<td>Unsigned</td>
</tr>
<tr>
<td>Carry</td>
<td>BCS</td>
<td>25</td>
<td>C = 1</td>
<td>No Carry</td>
<td>BCC</td>
<td>24</td>
<td>Simple</td>
</tr>
<tr>
<td>Negative</td>
<td>BMI</td>
<td>2B</td>
<td>N = 1</td>
<td>Plus</td>
<td>BPL</td>
<td>2A</td>
<td>Simple</td>
</tr>
<tr>
<td>Overflow</td>
<td>BVS</td>
<td>29</td>
<td>V = 1</td>
<td>No Overflow</td>
<td>BVC</td>
<td>28</td>
<td>Simple</td>
</tr>
<tr>
<td>r=0</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r≠0</td>
<td>BNE</td>
<td>26</td>
<td>Simple</td>
</tr>
<tr>
<td>Always</td>
<td>BRA</td>
<td>20</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
BPL  

Branch if Plus

**Operation**

If N = 0, then (PC) + REL ⇒ PC

Simple branch

**Syntax Variations**

<table>
<thead>
<tr>
<th>BPL</th>
<th>oprdest</th>
<th>REL</th>
</tr>
</thead>
</table>

**Addressing Modes**

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Detailed Instruction Formats**

**REL**

<table>
<thead>
<tr>
<th>REL_SIZE</th>
<th>7 bit DISPLACEMENT (REL_SIZE=0)</th>
<th>high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE=1)</th>
<th>Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE=1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>rb</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

Branch Always

Operation

(PC) + REL ⇒ PC

Simple unconditional branch

Syntax Variations

<table>
<thead>
<tr>
<th>BRA</th>
<th>oprdest</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Detailed Instruction Formats

REL

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

REL_SIZE: 7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)

Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)

20 rb RBRA 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

<table>
<thead>
<tr>
<th>Test</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Boolean</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>r&gt;m</td>
<td>BGT</td>
<td>2E</td>
<td>Z</td>
<td>(N ^ V) = 0</td>
</tr>
<tr>
<td>r≥m</td>
<td>BGE</td>
<td>2C</td>
<td>N ^ V = 0</td>
<td>r&lt;m</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r&gt;m</td>
</tr>
<tr>
<td>r≤m</td>
<td>BLE</td>
<td>2F</td>
<td>Z</td>
<td>(N ^ V) = 1</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLT</td>
<td>2D</td>
<td>N ^ V = 1</td>
<td>r=m</td>
</tr>
<tr>
<td>r&gt;m</td>
<td>BHI</td>
<td>22</td>
<td>C</td>
<td>Z = 0</td>
</tr>
<tr>
<td>r≥m</td>
<td>BHS/BCC</td>
<td>24</td>
<td>C = 0</td>
<td>r&lt;m</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r&gt;m</td>
</tr>
<tr>
<td>r≤m</td>
<td>BLS</td>
<td>23</td>
<td>C</td>
<td>Z = 1</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLO/BCS</td>
<td>25</td>
<td>C = 1</td>
<td>r≥m</td>
</tr>
<tr>
<td>Carry</td>
<td>BCS</td>
<td>25</td>
<td>C = 1</td>
<td>No Carry</td>
</tr>
<tr>
<td>Negative</td>
<td>BMI</td>
<td>2B</td>
<td>N = 1</td>
<td>Plus</td>
</tr>
<tr>
<td>Overflow</td>
<td>BVS</td>
<td>29</td>
<td>V = 1</td>
<td>No Overflow</td>
</tr>
<tr>
<td>r=0</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r≠0</td>
</tr>
<tr>
<td>Always</td>
<td>BRA</td>
<td>20</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Linear S12 Core Reference Manual, Rev. 1.01

Freescale Semiconductor
## BRCLR Test Bit and Branch if Clear

### 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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRCLR D_{i},#opr5i,oprdst</td>
<td>REG-IMM-REL</td>
</tr>
<tr>
<td>BRCLR D_{i},Dn,oprdst</td>
<td>REG-REG-REL</td>
</tr>
<tr>
<td>BRCLR.bwloprmreg,#opr5i,oprdst</td>
<td>OPR/1/2/3-IMM-REL</td>
</tr>
<tr>
<td>BRCLR.bwloprmreg,Dn,oprdst</td>
<td>OPR/1/2/3-REG-REL</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>U</th>
<th>I</th>
<th>P</th>
<th>L</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Δ</td>
</tr>
</tbody>
</table>

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

### Detailed Instruction Formats

#### REG-IMM-REL

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>bm</td>
<td>rb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
02 bm rb  BRCLR  Di,#opr5i,oprdst ;Dest is within +63/–64 (7-bit)
```

#### REG-REG-REL

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>xb</td>
<td>rb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
02 bm rb  BRCLR  Di,#opr5i,oprdst ;Dest is within +63/–64 (7-bit)
```

Linear S12 Core Reference Manual, Rev. 1.01

140 Freescale Semiconductor
# OPR/1/2/3-IMM-REL

**Byte-sized operand (.B)**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>n[2:0]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Word-sized operand (.W)**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>n[2:0]</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>n[3]</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Long-word sized operand (.L)**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>n[2:0]</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>n[4:3]</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**OPR POSTBYTE**

- (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)
- Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)
02 bm xb rb BRCLR.bw2 oprsxe4i, opr5i, oprdest ; (7-bit)
02 bm xb rb r1 BRCLR.bw2 oprsxe4i, opr5i, oprdest ; (15-bit)
02 bm xb rb BRCLR.bw2 opru4, xys, opr5i, oprdest ; (7-bit)
02 bm xb rb r1 BRCLR.bw2 opru4, xys, opr5i, oprdest ; (15-bit)
02 bm xb rb BRCLR.bw2 opru14, Di, opr5i, oprdest ; see efficient REG-IMM version
02 bm xb rb r1 BRCLR.bw2 opru14, Di, opr5i, oprdest ; see efficient REG-IMM version
02 bm xb rb BRCLR.bw2 opru24, Di, opr5i, oprdest ; (7-bit)
02 bm xb rb r1 BRCLR.bw2 opru24, Di, opr5i, oprdest ; (15-bit)
OPR/1/2/3-REG-REL

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 02</td>
</tr>
<tr>
<td>1</td>
<td>PARAMETER REGISTER Dn</td>
<td>SIZE (.B-0:0, .W-0:1, .L-1:1)</td>
<td>0</td>
<td>1</td>
<td>bm</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OPR POSTBYTE

(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  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  BRCLR.bwl  Di, Dn, oprdest ;see more efficient REG-REG version
02 bm xb rb  BRCLR.bwl  (opr4, xys), Dn, oprdest ;(7-bit)
02 bm xb rb  BRCLR.bwl  (opr4, xys), Dn, oprdest ;(15-bit)
02 bm xb rb  BRCLR.bwl  ((+-xy) | (xy+)-s| (s+)), Dn, oprdest ;(7-bit)
02 bm xb rb  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  BRCLR.bwl  (Di, xys), Dn, oprdest ;(15-bit)
02 bm xb rb  BRCLR.bwl  [Di, xy], Dn, oprdest ;(7-bit)
02 bm xb rb  BRCLR.bwl  [Di, xy], Dn, oprdest ;(15-bit)
02 bm xb rb  BRCLR.bwl  (opr9, xysp), Dn, oprdest ;(7-bit)
02 bm xb rb  BRCLR.bwl  (opr9, xysp), Dn, oprdest ;(15-bit)
02 bm xb rb  BRCLR.bwl  [opr9, xysp], Dn, oprdest ;(7-bit)
02 bm xb rb  BRCLR.bwl  [opr9, xysp], Dn, oprdest ;(15-bit)
02 bm xb rb  BRCLR.bwl  opru14, Dn, oprdest ;(7-bit)
02 bm xb rb  BRCLR.bwl  opru14, Dn, oprdest ;(15-bit)
02 bm xb rb  BRCLR.bwl  (opr18, Di), Dn, oprdest ;(7-bit)
02 bm xb rb  BRCLR.bwl  (opr18, Di), Dn, oprdest ;(15-bit)
02 bm xb rb  BRCLR.bwl  opr24, Dn, oprdest ;(7-bit)
02 bm xb rb  BRCLR.bwl  opr24, Dn, oprdest ;(15-bit)
02 bm xb rb  BRCLR.bwl  (opr24, xysp), Dn, oprdest ;(7-bit)
02 bm xb rb  BRCLR.bwl  (opr24, xysp), Dn, oprdest ;(15-bit)
02 bm xb rb  BRCLR.bwl  [opr24, xysp], Dn, oprdest ;(7-bit)
02 bm xb rb  BRCLR.bwl  [opr24, xysp], Dn, oprdest ;(15-bit)
02 bm xb rb  BRCLR.bwl  opr24, Dn, oprdest ;(7-bit)
02 bm xb rb  BRCLR.bwl  opr24, Dn, oprdest ;(15-bit)
02 bm xb rb  BRCLR.bwl  [opr24], Dn, oprdest ;(7-bit)
02 bm xb rb  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=$D_2$, 0:0:1=$D_3$, 0:1:0=$D_4$, 0:1:1=$D_5$, 1:0:0=$D_0$, 1:0:1=$D_1$, 1:1:0=$D_6$, and 1:1:1=$D_7$).

PARAMETER REGISTER $D_n$ - This field specifies the number of the data register $D_n$ (0:0:0=$D_2$, 0:0:1=$D_3$, 0:1:0=$D_4$, 0:1:1=$D_5$, 1:0:0=$D_0$, 1:0:1=$D_1$, 1:1:0=$D_6$, and 1:1:1=$D_7$) 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

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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRSET Di,#opr5i,oprdest</td>
<td>REG-IMM-REL</td>
</tr>
<tr>
<td>BRSET Di,Dn,oprdest</td>
<td>REG-REG-REL</td>
</tr>
<tr>
<td>BRSET.bwloprmemreg,#opr5i,oprdest</td>
<td>OPR/1/2/3-IMM-REL</td>
</tr>
<tr>
<td>BRSET.bwloprmemreg,Dn,oprdest</td>
<td>OPR/1/2/3-REG-REL</td>
</tr>
</tbody>
</table>

Description
Tests the specified bit in Di 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 Dn. 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

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

Detailed Instruction Formats

**REG-IMM-REL**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>bm</th>
<th>n[4:0]</th>
<th>SD REGISTER Dij</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>03</td>
<td>REL_SIZE</td>
<td>DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>03</td>
<td>rb</td>
<td>Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)</td>
</tr>
</tbody>
</table>

03 bm rb BRSET Di,#opr5i,oprdest ;Dest is within +63/-64 (7-bit)

03 bm rb rl BRSET Di,#opr5i,oprdest ;Dest within ~ +/-16K (15-bit)

**REG-REG-REL**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>bm</th>
<th>xb</th>
<th>n[4:0]</th>
<th>SD REGISTER Dij</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>03</td>
<td>REL_SIZE</td>
<td>DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>rb</td>
<td>Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)</td>
<td></td>
</tr>
</tbody>
</table>

03 bm xb rb BRSET Di,Dn,oprdest ;Dest is within +63/-64 (7-bit)
### OPR/1/2/3-IMM-REL

#### Byte-sized operand (.B)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>n[2:0]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**OPR POSTBYTE**

(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)

### Word-sized operand (.W)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**OPR POSTBYTE**

(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)

### Long-word sized operand (.L)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**OPR POSTBYTE**

(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)
Chapter 6 Instruction Glossary

03 bm xb rb  BRSET.lw2  #oprsxe4i,#opr5i,oprdest;(7-bit)
03 bm xb rb r1  BRSET.lw2  #oprsxe4i,#opr5i,oprdest;(15-bit)
03 bm xb rb  BRSET.lw2  Di,#opr5i,oprdest;see efficient REG-IMM version
03 bm xb rb r1  BRSET.lw2  (opr4,xy),#opr5i,oprdest;(7-bit)
03 bm xb rb r1  BRSET.lw2  (opr4,xy),#opr5i,oprdest;(15-bit)
03 bm xb rb  BRSET.lw2  (((-xy)|((xy)|(-s)|(+s))),#opr5i,oprdest;(7-bit)
03 bm xb rb r1  BRSET.lw2  (((-xy)|((xy)|(-s)|(+s))),#opr5i,oprdest;(15-bit)
03 bm xb rb  BRSET.lw2  (Di,xy),#opr5i,oprdest;(7-bit)
03 bm xb rb r1  BRSET.lw2  (Di,xy),#opr5i,oprdest;(15-bit)
03 bm xb x1 rb  BRSET.lw2  [Di,xy],#opr5i,oprdest;(7-bit)
03 bm xb x1 rb r1  BRSET.lw2  [Di,xy],#opr5i,oprdest;(15-bit)
03 bm xb x1 rb  BRSET.lw2  (opr5s,xy),#opr5i,oprdest;(7-bit)
03 bm xb x1 r1  BRSET.lw2  (opr5s,xy),#opr5i,oprdest;(15-bit)
03 bm xb x1 rb r1  BRSET.lw2  [opr5s,xy],#opr5i,oprdest;(7-bit)
03 bm xb x1 rb r1  BRSET.lw2  [opr5s,xy],#opr5i,oprdest;(15-bit)
03 bm xb x2 x1 rb  BRSET.lw2  opru14,#opr5i,oprdest;(7-bit)
03 bm xb x2 x1 rb r1  BRSET.lw2  opru14,#opr5i,oprdest;(15-bit)
03 bm xb x2 x1 rb  BRSET.lw2  opru18,Di,#opr5i,oprdest;(7-bit)
03 bm xb x2 x1 rb r1  BRSET.lw2  opru18,Di,#opr5i,oprdest;(15-bit)
03 bm xb x2 x1 rb  BRSET.lw2  opru18,#opr5i,oprdest;(7-bit)
03 bm xb x2 x1 rb r1  BRSET.lw2  opru18,#opr5i,oprdest;(15-bit)
03 bm xb x3 x2 x1 rb  BRSET.lw2  opru24,xy,xy,#opr5i,oprdest;(7-bit)
03 bm xb x3 x2 x1 rb  BRSET.lw2  opru24,xy,xy,#opr5i,oprdest;(15-bit)
03 bm xb x3 x2 x1 rb  BRSET.lw2  [opr24,xy],#opr5i,oprdest;(7-bit)
03 bm xb x3 x2 x1 r1  BRSET.lw2  [opr24,xy],#opr5i,oprdest;(15-bit)
03 bm xb x3 x2 x1 rb r1  BRSET.lw2  [opr24,xy],#opr5i,oprdest;(7-bit)
03 bm xb x3 x2 x1 rb r1  BRSET.lw2  [opr24,xy],#opr5i,oprdest;(15-bit)
03 bm xb x3 x2 x1 rb  BRSET.lw2  opru24,Di,#opr5i,oprdest;(7-bit)
03 bm xb x3 x2 x1 r1  BRSET.lw2  opru24,Di,#opr5i,oprdest;(15-bit)
03 bm xb x3 x2 x1 rb  BRSET.lw2  opru24,#opr5i,oprdest;(7-bit)
03 bm xb x3 x2 x1 r1  BRSET.lw2  opru24,#opr5i,oprdest;(15-bit)
03 bm xb x3 x2 x1 rb r1  BRSET.lw2  [opr24],#opr5i,oprdest;(7-bit)
03 bm xb x3 x2 x1 rb  BRSET.lw2  [opr24],#opr5i,oprdest;(15-bit)
<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PARAMETER REGISTER Dn**

**SIZE**: 
- **.B-0**: 0
- **.W-0**: 1
- **.L-1**: 1

**OPR POSTBYTE**

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

<table>
<thead>
<tr>
<th>REL_SIZE</th>
<th>7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000011</td>
<td>Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)</td>
<td>r1</td>
</tr>
</tbody>
</table>

**03 bm xb rb**

BRSET.bwl opru18,Dn,oprdest ;(7-bit)

**03 hm x3 x2 x1 rb**

BRSET.bwl opru24,Dn,oprdest ;(7-bit)

**03 hm x3 x2 x1 rb r1**

BRSET.bwl opru24,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

**Operation**

\[ \text{bitn of } D_i \Rightarrow C; \text{ then } (D_i) \mid (\text{bitn}) \Rightarrow D_i \]

\[ \text{bitn of } M \Rightarrow C; \text{ then } (M) \mid (\text{bitn}) \Rightarrow M \]

**Syntax Variations**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSET ( D_i, #\text{opr5i} )</td>
<td>REG-IMM</td>
</tr>
<tr>
<td>BSET ( D_i, D_n )</td>
<td>REG-REG</td>
</tr>
<tr>
<td>BSET.( \text{bw}l ) ( \text{oprmemreg}, #\text{opr5i} )</td>
<td>OPR/1/2/3-IMM</td>
</tr>
<tr>
<td>BSET.( \text{bw}l ) ( \text{oprmemreg}, D_n )</td>
<td>OPR/1/2/3-REG</td>
</tr>
</tbody>
</table>

**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, \( \text{oprmemreg} \) 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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- **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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**BSET** \( D_i, \#\text{opr5i} \)
### REG-REG

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>PARAMETER REGISTER Dn</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>SD REGISTER Di</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ED bm xb**  
**BSET**  
**Di, Dn**

### OPR/1/2/3-IMM

#### Byte-sized operand (.B)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>n[2:0]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ED bm xb**  
**OPR POSTBYTE**  
(Optional ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

#### Word-sized operand (.W)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>n[2:0]</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>n[3]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ED bm xb**  
**OPR POSTBYTE**  
(Optional ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

#### Long-word sized operand (.L)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>n[2:0]</td>
<td>1</td>
<td>0</td>
<td>n[4:3]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ED sb xb**  
**BSET.xwl**  
#opr51x4i,#opr5i; not appropriate for destination

**ED sb xb**  
**BSET.xwl**  
Di,#opr5i; see more efficient REG-IMM1 version

**ED sb xb**  
**BSET.xwl**  
 opru4,xys,#opr5i

**ED sb xb**  
**BSET.xwl**  
 (s+) or (s-),#opr5i

**ED sb xb**  
**BSET.xwl**  
 (Di,xys),#opr5i

**ED sb xb**  
**BSET.xwl**  
Di,xys,#opr5i

**ED sb xb x1**  
**BSET.xwl**  
 opru9,xys,#opr5i

**ED sb xb x1**  
**BSET.xwl**  
 opru9,xys,#opr5i

**ED sb xb x1**  
**BSET.xwl**  
 opru14,#opr5i

**ED sb xb x1**  
**BSET.xwl**  
 opru18,Di,#opr5i

**ED sb xb x2 x1**  
**BSET.xwl**  
 opru18,Di,#opr5i

**ED sb xb x2 x1**  
**BSET.xwl**  
 opru24,xys,#opr5i

**ED sb xb x3 x2 x1**  
**BSET.xwl**  
 opru24,xys,#opr5i

**ED sb xb x3 x2 x1**  
**BSET.xwl**  
 opru24,Di,#opr5i

**ED sb xb x3 x2 x1**  
**BSET.xwl**  
 opru24,#opr5i

**ED sb xb x3 x2 x1**  
**BSET.xwl**  
 opru24,#opr5i

**ED sb xb x3 x2 x1**  
**BSET.xwl**  
 opru24,#opr5i

**ED sb xb x3 x2 x1**  
**BSET.xwl**  
 opru24,#opr5i

**ED sb xb x3 x2 x1**  
**BSET.xwl**  
 opru24,#opr5i

**ED sb xb x3 x2 x1**  
**BSET.xwl**  
 opru24,#opr5i

**ED sb xb x3 x2 x1**  
**BSET.xwl**  
 opru24,#opr5i

**ED sb xb x3 x2 x1**  
**BSET.xwl**  
 opru24,#opr5i

**ED sb xb x3 x2 x1**  
**BSET.xwl**  
 opru24,#opr5i

**ED sb xb x3 x2 x1**  
**BSET.xwl**  
 opru24,#opr5i

**ED sb xb x3 x2 x1**  
**BSET.xwl**  
 opru24,#opr5i

**ED sb xb x3 x2 x1**  
**BSET.xwl**  
 opru24,#opr5i

**ED sb xb x3 x2 x1**  
**BSET.xwl**  
 opru24,#opr5i

**ED sb xb x3 x2 x1**  
**BSET.xwl**  
 opru24,#opr5i
### Instruction Fields

**SD REGISTER Di** - This field specifies the number of the data register Di 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 Dn** - This field specifies the number of the data register Dn (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**

**Operation**

\[
(SP) - 3 \Rightarrow SP \\
RTN[23:0] \Rightarrow M_{(SP)} : M_{(SP + 1)} : M_{(SP + 2)} \\
(PC) + REL \Rightarrow PC
\]

**Syntax Variations**

<table>
<thead>
<tr>
<th>BSR</th>
<th>oprdest</th>
<th>REL</th>
</tr>
</thead>
</table>

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Detailed Instruction Formats**

**REL**

<table>
<thead>
<tr>
<th>REL_SIZE</th>
<th>DISPLACEMENT</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REL_SIZE</td>
<td>7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)

**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.

**Linear S12 Core Reference Manual, Rev. 1.01**

Freescale Semiconductor
**BTGL**

**Test and Toggle Bit**

*(invert bit)*

**Operation**

\[
\text{bitn of } D_i \Rightarrow C; \text{ then } (D_i) \, ^\wedge \, \text{bitn} \Rightarrow D_i \\
\text{bitn of } M \Rightarrow C; \text{ then } (M) \, ^\wedge \, \text{bitn} \Rightarrow M
\]

**Syntax Variations**

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTGL \ D_i,#opr5i</td>
<td>REG-IMM</td>
</tr>
<tr>
<td>BTGL \ D_i,\ D_n</td>
<td>REG-REG</td>
</tr>
<tr>
<td>BTGL.\ Dwploprmemreg,#opr5i</td>
<td>OPR/1/2/3-IMM</td>
</tr>
<tr>
<td>BTGL.\ Dwploprmemreg,\ D_n</td>
<td>OPR/1/2/3-REG</td>
</tr>
</tbody>
</table>

**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, \( oprimemreg \) 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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>−−−−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>Δ</td>
<td>Δ</td>
<td>0</td>
<td>Δ</td>
</tr>
</tbody>
</table>

- **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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>EE</th>
<th>bm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>( n[4.0] )</td>
<td>SD REGISTER ( D_i )</td>
</tr>
</tbody>
</table>

\( EE \, \text{bm} \)
# REG-REG

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>PARAMETER REGISTER Dn</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>SD REGISTER Di</td>
</tr>
</tbody>
</table>

EE bm xb

BTGL Di, Dn

## OPR/1/2/3-IMM

### Byte-sized operand (.B)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>n[2:0]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OPR POSTBYTE

(Optional address-byte depending on address-mode)

### Word-sized operand (.W)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>n[2:0]</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>n[3]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OPR POSTBYTE

(Optional address-byte depending on address-mode)

### Long-word sized operand (.L)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>n[2:0]</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>n[4:3]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OPR POSTBYTE

(Optional address-byte depending on address-mode)

---

EE sb xb

BTGL lw1

# oprsxe4i,#opr5i ;not appropriate for destination

EE sb xb

BTGL lw1 Di,#opr5i ;see more efficient REG-IMM version

EE sb xb

BTGL lw1 (opru4, yxs),#opr5i

EE sb xb

BTGL lw1 ([xy]|(xy-+)|(-s)|(+s)),#opr5i

EE sb xb

BTGL lw1 (Di, yxs),#opr5i

EE sb xb

BTGL lw1 [Di, yxs],#opr5i

EE sb xb

BTGL lw1 (opr9, yxs),#opr5i

EE sb xb

BTGL lw1 opru14,#opr5i

EE sb xb x1

BTGL lw1 [opr9, yxs],#opr5i

EE sb xb x1

BTGL lw1 opru14,#opr5i

EE sb xb x2 x1

BTGL lw1 (opr18, Di),#opr5i

EE sb xb x2 x1

BTGL lw1 opru18,#opr5i

EE sb xb x3 x2 x1

BTGL lw1 (opr24, yxs),#opr5i

EE sb xb x3 x2 x1

BTGL lw1 [opr24, yxs],#opr5i

EE sb xb x3 x2 x1

BTGL lw1 (opr24, Di),#opr5i

EE sb xb x3 x2 x1

BTGL lw1 opr24,#opr5i

EE sb xb x3 x2 x1

BTGL lw1 [opr24],#opr5i
**OPR/1/2/3-REG**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>EE bm xb</th>
<th>BTGL.bwl</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### Instruction Fields

SD REGISTER Di - This field specifies the number of the data register Di 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 Dn - This field specifies the number of the data register Dn (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**

**Operation**
If $V = 0$, then $(PC) + REL \Rightarrow PC$

Simple branch

**Syntax Variations**

<table>
<thead>
<tr>
<th>BVC</th>
<th>oprdest</th>
<th>REL</th>
</tr>
</thead>
</table>

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Detailed Instruction Formats**

**REL**

<table>
<thead>
<tr>
<th>REL</th>
<th>oprdest</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>oprdest; Dest is within +63/-64 (7-bit offset)</td>
<td></td>
</tr>
<tr>
<td>28 rb</td>
<td>oprdest; Dest is within +/-16K (15-bit offset)</td>
<td></td>
</tr>
</tbody>
</table>

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

Operation
If V = 1, then (PC) + REL ⇒ PC
Simple branch

Syntax Variations

<table>
<thead>
<tr>
<th>BVS</th>
<th>oprdest</th>
<th>REL</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Detailed Instruction Formats

REL

<table>
<thead>
<tr>
<th>REL_SIZE</th>
<th>7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)</td>
<td>r1</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Test</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Boolean</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>r&gt;m</td>
<td>BGT</td>
<td>2E</td>
<td>Z</td>
<td>(N ^ V) = 0</td>
</tr>
<tr>
<td>r≥m</td>
<td>BGE</td>
<td>2C</td>
<td>N ^ V = 0</td>
<td>Signed</td>
</tr>
<tr>
<td>r=m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r&gt;m</td>
</tr>
<tr>
<td>r≤m</td>
<td>BLE</td>
<td>2F</td>
<td>Z</td>
<td>(N ^ V) = 1</td>
</tr>
<tr>
<td>r&lt;m</td>
<td>BLT</td>
<td>2D</td>
<td>N ^ V = 1</td>
<td>r&gt;m</td>
</tr>
<tr>
<td>r≥m</td>
<td>BHI</td>
<td>22</td>
<td>C</td>
<td>Z = 0</td>
</tr>
<tr>
<td>r=m</td>
<td>BHS/BCC</td>
<td>24</td>
<td>C = 0</td>
<td>r&lt;m</td>
</tr>
<tr>
<td>r≤m</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r&gt;m</td>
</tr>
<tr>
<td>r≥m</td>
<td>BLS</td>
<td>23</td>
<td>C</td>
<td>Z = 1</td>
</tr>
<tr>
<td>r≤m</td>
<td>BLO/BCS</td>
<td>25</td>
<td>C = 1</td>
<td>r&gt;m</td>
</tr>
<tr>
<td>Carry</td>
<td>BCS</td>
<td>25</td>
<td>C = 1</td>
<td>No Carry</td>
</tr>
<tr>
<td>Negative</td>
<td>BMI</td>
<td>2B</td>
<td>N = 1</td>
<td>Plus</td>
</tr>
<tr>
<td>Overflow</td>
<td>BVS</td>
<td>29</td>
<td>V = 1</td>
<td>No Overflow</td>
</tr>
<tr>
<td>r=0</td>
<td>BEQ</td>
<td>27</td>
<td>Z = 1</td>
<td>r≥0</td>
</tr>
<tr>
<td>Always</td>
<td>BRA</td>
<td>20</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
CLB

Count Leading Sign-Bits

Syntax Variations

<table>
<thead>
<tr>
<th>CLB</th>
<th>cpureg, cpureg</th>
</tr>
</thead>
</table>

Addressing Modes

| 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>A</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

N: 0, cleared.

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

V: 0, cleared.

Detailed Instruction Formats

INH

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

SOURCE REGISTER Di 0 DESTINATION REGISTER Di cb

Instruction Fields

SOURCE REGISTER Di - This field specifies the number of the data register Di 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 Di - This field specifies the number of the data register Di 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)

**Operation**

\[ 0 \Rightarrow C \text{ bit} \]

**Syntax Variations**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLC</td>
<td>IMM1</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

C: Cleared.

**Detailed Instruction Formats**

<table>
<thead>
<tr>
<th>IMM1</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CE</th>
<th>FE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CE</th>
<th>FE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
CLI

Clear Interrupt Mask
(Translates to ANDCC #$EF)

Operation

0 ⇒ I bit

Syntax Variations

<table>
<thead>
<tr>
<th>CLI</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM1</td>
</tr>
</tbody>
</table>

Addressing Modes

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

CE EF CLI
CLR

Clear Memory, Register, or Index Register

**Operation**

\[ 0 \Rightarrow M; \text{ or } 0 \Rightarrow D_i; \text{ or } 0 \Rightarrow X; \text{ or } 0 \Rightarrow Y \]

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR.bwpl oprmemreg</td>
<td>OPR/1/2/3</td>
</tr>
<tr>
<td>CLR D_i</td>
<td>INH</td>
</tr>
<tr>
<td>CLR X</td>
<td>INH</td>
</tr>
<tr>
<td>CLR Y</td>
<td>INH</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

N: 0, cleared.
Z: 1, set.
V: 0, cleared.
C: 0, cleared.

**Detailed Instruction Formats**

**INH**

```
INH

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

```
3q

CLR D_i
```

**INH**

```
INH

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

```
9p

9A
CLR X

9B
CLR Y
```

Linear S12 Core Reference Manual, Rev. 1.01
OPR/1/2/3

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**OPR POSTBYTE**

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

### Instruction Fields

- **SD REGISTER Di** - This field specifies the number of the data register Di 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)

Operation

\[ 0 \Rightarrow V \text{ bit} \]

Syntax Variations

| CLV | IMM1 |

Addressing Modes

| CLV |

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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

V: Cleared.

Detailed Instruction Formats

IMM1

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

CE FD CLV
Chapter 6 Instruction Glossary

CMP

Compare

Operation

\((D_i) \rightarrow (M); (X) \rightarrow (M); (Y) \rightarrow (M); (S) \rightarrow (M); \text{ or } (X) \rightarrow (Y)\)

Syntax Variations | Addressing Modes
--- | ---
CMP \(D_i,\#\text{opr}1\text{mmsz}\) | IMM1/2/4
CMP \(D_i,\text{oprmemreg}\) | OPR1/2/3
CMP \(xy,\#\text{opr24i}\) | IMM3
CMP \(xy,\text{oprmemreg}\) | OPR1/2/3
CMP \(S,\#\text{opr24i}\) | IMM3
CMP \(S,\text{oprmemreg}\) | OPR1/2/3
CMP \(X,Y\) | INH

Description

Compare register \(D_i, X, Y, \text{ or } S\) 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, \(\text{oprmemreg}\) 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

\[
\begin{array}{ccccccccccc}
\hline
\end{array}
\]

- **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 Di)**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ep</td>
<td>IMMEDIATE DATA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF Di)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF Di)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF Di)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 Di)**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fn xb</td>
<td>OPR POSTBYTE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fn xb  CMP  Di,#oprsex4i ; -1, +1, 2, 3...14, 15
Fn xb  CMP  Di,Dj
Fn xb  CMP  Di,(opr4,xy)
Fn xb  CMP  Di,(+-xy) | (xy+) | (-s) | (s+)
Fn xb  CMP  Di,(Dj,xy)
Fn xb x1  CMP  Di,(opr9,xy)
Fn xb x1  CMP  Di,[opr9,xy]
Fn xb x1  CMP  Di,opr14
Fn xb x2 x1  CMP  Di,(opr18,Dj)
Fn xb x2 x1  CMP  Di,opr18
Fn xb x3 x2 x1  CMP  Di,(opr24,xy)
Fn xb x3 x2 x1  CMP  Di,[opr24,xy]
Fn xb x3 x2 x1  CMP  Di,(opr24,Dj)
Fn xb x3 x2 x1  CMP  Di,opr24
Fn xb x3 x2 x1  CMP  Di,[opr24]

**IMM3 (for CMP X and Y)**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ep</td>
<td>IMMEDIATE DATA[23:16]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMMEDIATE DATA[15:8]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMMEDIATE DATA[7:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ep i3 i2 i1  CMP  xy,#opr24i
### OPR/1/2/3 (for CMP X and Y)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>( F_p ) ( x_b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Y/X</td>
<td></td>
</tr>
</tbody>
</table>

**OPR POSTBYTE**

- (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)
- (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

\( F_p \) \( x_b \) CMP \( x_y, \#\text{opr}sx4i \); \(-1, +1, 2, 3 \ldots 14, 15\)
\( F_p \) \( x_b \) CMP \( x_y, D_j \)
\( F_p \) \( x_b \) CMP \( x_y, (\text{opr}u4, x_y) \)
\( F_p \) \( x_b \) CMP \( x_y, ((+x-y) | (x+y-)) | (-s) | (s+) \)
\( F_p \) \( x_b \) CMP \( x_y, (D_j, x_y) \)
\( F_p \) \( x_b \) CMP \( x_y, [D_j, x_y] \)
\( F_p \) \( x_b \) \( x_l \) CMP \( x_y, (\text{opr}u9, x_y) \)
\( F_p \) \( x_b \) \( x_l \) CMP \( x_y, [\text{opr}u9, x_y] \)
\( F_p \) \( x_b \) \( x_l \) CMP \( x_y, \text{opr}u14 \)
\( F_p \) \( x_b \) \( x_l \) \( x_l \) CMP \( x_y, (\text{opr}u18, D_j) \)
\( F_p \) \( x_b \) \( x_l \) \( x_l \) CMP \( x_y, \text{opr}u18 \)
\( F_p \) \( x_b \) \( x_l \) \( x_l \) \( x_l \) CMP \( x_y, (\text{opr}u24, x_y) \)
\( F_p \) \( x_b \) \( x_l \) \( x_l \) \( x_l \) CMP \( x_y, [\text{opr}u24, D_j] \)
\( F_p \) \( x_b \) \( x_l \) \( x_l \) \( x_l \) CMP \( x_y, \text{opr}24 \)
\( F_p \) \( x_b \) \( x_l \) \( x_l \) \( x_l \) CMP \( x_y, [\text{opr}24] \)

### IMM3 (for CMP S)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>( \text{1B} ) ( \text{04} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**IMMEDIATE DATA[23:16]**
**IMMEDIATE DATA[15:8]**
**IMMEDIATE DATA[7:0]**

\( 1\text{B} \text{0}4 \ \text{i}3 \text{ i}2 \text{ i}1 \text{ x} \text{r} \text{m} \text{n} \text{s} \text{ u} \text{ d} \text{ a} \text{ t} \text{ a} \text{ r} \text{i} \text{ m} \text{ e} \text{[}2\text{3}:\text{16}] \)

### OPR/1/2/3 (for CMP S)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>( \text{1B} ) ( \text{02} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**OPR POSTBYTE**

- (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)
- (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

\( 1\text{B} \text{0}2 \ \text{x}b \) CMP \( S, \#\text{opr}sx4i \); \(-1, +1, 2, 3 \ldots 14, 15\)
\( 1\text{B} \text{0}2 \ \text{x}b \) CMP \( S, D_j \)
\( 1\text{B} \text{0}2 \ \text{x}b \) CMP \( S, (\text{opr}u4, x_y) \)
\( 1\text{B} \text{0}2 \ \text{x}b \) CMP \( S, ((+x-y) | (x+y-)) | (-s) | (s+) \)
\( 1\text{B} \text{0}2 \ \text{x}b \) CMP \( S, (D_j, x_y) \)
\( 1\text{B} \text{0}2 \ \text{x}b \) CMP \( S, [D_j, x_y] \)
\( 1\text{B} \text{0}2 \ \text{x}b \) \( x_l \) CMP \( S, (\text{opr}u9, x_y) \)
\( 1\text{B} \text{0}2 \ \text{x}b \) \( x_l \) CMP \( S, [\text{opr}u9] \)
\( 1\text{B} \text{0}2 \ \text{x}b \) \( x_l \) CMP \( S, \text{opr}u14 \)
INH (for CMP X,Y)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>FC</td>
</tr>
</tbody>
</table>

**Instruction Fields**

**SD REGISTER Di** - This field specifies the number of the data register Di 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.
Complement Memory

**Operation**

\[ \sim(M) \Rightarrow M \]

**Syntax Variations**

<table>
<thead>
<tr>
<th>COM.bwl oprmemreg</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPR/1/2/3</td>
</tr>
</tbody>
</table>

**Addressing Modes**

**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 \( b=8 \text{ bit byte}, w=16 \text{ bit word}, \text{ or } l=32 \text{ bit long-word} \). 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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**OPR POSTBYTE**

- (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)
- (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)
- (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

<table>
<thead>
<tr>
<th>( \text{Cp} ) ( \times b )</th>
<th>( \text{COM.bwl} )</th>
<th>#oprxe4i ; not appropriate for destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Cp} ) ( \times b )</td>
<td>COM.bwl</td>
<td>Di</td>
</tr>
<tr>
<td>( \text{Cp} ) ( \times b )</td>
<td>COM.bwl</td>
<td>(opru4, xys)</td>
</tr>
<tr>
<td>( \text{Cp} ) ( \times b )</td>
<td>COM.bwl</td>
<td>((+xy) \mid (xy^-) \mid (-s) \mid (s^+))</td>
</tr>
<tr>
<td>( \text{Cp} ) ( \times b )</td>
<td>COM.bwl</td>
<td>(Di, xys)</td>
</tr>
<tr>
<td>( \text{Cp} ) ( \times b )</td>
<td>COM.bwl</td>
<td>[Di, xy]</td>
</tr>
<tr>
<td>( \text{Cp} ) ( \times b \times 1 )</td>
<td>COM.bwl</td>
<td>(oprs9, xysp)</td>
</tr>
<tr>
<td>( \text{Cp} ) ( \times b \times 1 )</td>
<td>COM.bwl</td>
<td>[oprs9, xysp]</td>
</tr>
<tr>
<td>( \text{Cp} ) ( \times b \times 1 )</td>
<td>COM.bwl</td>
<td>opru4</td>
</tr>
<tr>
<td>( \text{Cp} ) ( \times b \times 2 \times 1 )</td>
<td>COM.bwl</td>
<td>(opru18, Di)</td>
</tr>
</tbody>
</table>
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

Operation

(D) – 1 ⇒ Di; 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)

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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>I</th>
<th>P</th>
<th>L</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Detailed Instruction Formats

REG-REL (Di)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REL_SIZE</th>
<th>DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)</td>
<td></td>
</tr>
</tbody>
</table>

Syntax Variations

<table>
<thead>
<tr>
<th>DBcc</th>
<th>Di, oprdest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Di, oprdest</td>
<td></td>
</tr>
<tr>
<td>X, oprdest</td>
<td></td>
</tr>
<tr>
<td>Y, oprdest</td>
<td></td>
</tr>
<tr>
<td>DBcc, bwploprmemreg, oprdest</td>
<td></td>
</tr>
</tbody>
</table>

Addressing Modes

REG-REL

OPR/1/2/3-REL

Linear S12 Core Reference Manual, Rev. 1.01
REG-REL (X, Y)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>CC (NE, EQ, PL, MI, GT, LE, –) 1</td>
<td>0</td>
<td>don't care</td>
<td>Y/X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REL_SIZE</td>
<td>7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rb</td>
<td>Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0B</th>
<th>lb</th>
<th>rb</th>
<th>DBCc</th>
<th>oprdest</th>
<th>;destination within ~+/-16k (15-bit)</th>
</tr>
</thead>
</table>

OPR/1/2/3-REL

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>CC (NE, EQ, PL, MI, GT, LE, –) 1</td>
<td>0</td>
<td>SIZE (.B, .W, .P, .L)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REL_SIZE</td>
<td>7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rb</td>
<td>Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0B</th>
<th>lb</th>
<th>xb</th>
<th>rb</th>
<th>DBCc, bpl</th>
<th>#oprsx4i, oprdest</th>
<th>;not appropriate for destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0B</td>
<td>lb</td>
<td>xb</td>
<td>r1</td>
<td>DBCc, bpl</td>
<td>#oprsx4i, oprdest</td>
<td>;not appropriate for destination</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>xb</td>
<td>rb</td>
<td>DBCc, bpl</td>
<td>Di, oprdest</td>
<td>;see efficient REG-REL version</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>xb</td>
<td>rb</td>
<td>DBCc, bpl</td>
<td>Di, oprdest</td>
<td>;see efficient REG-REL version</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>xb</td>
<td>rb</td>
<td>DBCc, bpl</td>
<td>opru4, xys, oprdest</td>
<td>;(7-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>xb</td>
<td>rb</td>
<td>DBCc, bpl</td>
<td>opru4, xys, oprdest</td>
<td>;(15-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>xb</td>
<td>rb</td>
<td>DBCc, bpl</td>
<td>{{+-xy}</td>
<td>xy+-}</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>xb</td>
<td>rb</td>
<td>DBCc, bpl</td>
<td>{{+-xy}</td>
<td>xy+-}</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>xb</td>
<td>rb</td>
<td>DBCc, bpl</td>
<td>(Di, xys), oprdest</td>
<td>;(7-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>xb</td>
<td>rb</td>
<td>DBCc, bpl</td>
<td>(Di, xys), oprdest</td>
<td>;(7-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>xb</td>
<td>r1</td>
<td>DBCc, bpl</td>
<td>[Di, xy], oprdest</td>
<td>;(15-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>xb</td>
<td>rb</td>
<td>DBCc, bpl</td>
<td>[Di, xy], oprdest</td>
<td>;(15-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>xb</td>
<td>rb</td>
<td>DBCc, bpl</td>
<td>opru9, xy, oprdest</td>
<td>;(7-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>xb</td>
<td>rb</td>
<td>DBCc, bpl</td>
<td>opru9, xy, oprdest</td>
<td>;(15-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>xb</td>
<td>rb</td>
<td>DBCc, bpl</td>
<td>opru14, oprdest</td>
<td>;(7-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>xb</td>
<td>r1</td>
<td>DBCc, bpl</td>
<td>opru14, oprdest</td>
<td>;(15-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>x2</td>
<td>x1</td>
<td>DBCc, bpl</td>
<td>opru18, Di, oprdest</td>
<td>;(7-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>x2</td>
<td>x1</td>
<td>DBCc, bpl</td>
<td>opru18, Di, oprdest</td>
<td>;(7-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>x2</td>
<td>x1</td>
<td>DBCc, bpl</td>
<td>opru18, Di, oprdest</td>
<td>;(15-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>x2</td>
<td>x1</td>
<td>DBCc, bpl</td>
<td>opru18, Di, oprdest</td>
<td>;(15-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>x2</td>
<td>x1</td>
<td>DBCb, bpl</td>
<td>opru18, Di, oprdest</td>
<td>;(15-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>x2</td>
<td>X</td>
<td>DBCb, bpl</td>
<td>opru24, xys, oprdest</td>
<td>;(7-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>x2</td>
<td>X</td>
<td>DBCb, bpl</td>
<td>opru24, xys, oprdest</td>
<td>;(15-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>x2</td>
<td>x</td>
<td>DBCb, bpl</td>
<td>opru24, xys, oprdest</td>
<td>;(7-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>x2</td>
<td>x</td>
<td>DBCb, bpl</td>
<td>opru24, xys, oprdest</td>
<td>;(15-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>x2</td>
<td>x</td>
<td>DBCb, bpl</td>
<td>opru24, xys, oprdest</td>
<td>;(7-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>x2</td>
<td>x</td>
<td>DBCb, bpl</td>
<td>opru24, xys, oprdest</td>
<td>;(15-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>x2</td>
<td>x</td>
<td>DBCb, bpl</td>
<td>opru24, xys, oprdest</td>
<td>;(15-bit)</td>
</tr>
<tr>
<td>0B</td>
<td>lb</td>
<td>x2</td>
<td>x</td>
<td>DBCb, bpl</td>
<td>opru24, xys, oprdest</td>
<td>;(7-bit)</td>
</tr>
</tbody>
</table>
Instruction Fields

CC - This field specifies the condition for the branch according to the table below:

<table>
<thead>
<tr>
<th>Test</th>
<th>Mnemonic</th>
<th>Condition</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>NE; r≠0</td>
<td>DBNE</td>
<td>000</td>
<td>Z = 0</td>
</tr>
<tr>
<td>EQ; r=0</td>
<td>DBEQ</td>
<td>001</td>
<td>Z = 1</td>
</tr>
<tr>
<td>PL; r≥0</td>
<td>DBPL</td>
<td>010</td>
<td>N = 0</td>
</tr>
<tr>
<td>MI; r&lt;0</td>
<td>DBMI</td>
<td>011</td>
<td>N = 1</td>
</tr>
<tr>
<td>GT; r&gt;0</td>
<td>DBGT</td>
<td>100</td>
<td>Z</td>
</tr>
<tr>
<td>LE; r≤0</td>
<td>DBLE</td>
<td>101</td>
<td>Z</td>
</tr>
<tr>
<td>reserved (Decrement and Branch Never)</td>
<td>110</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>111</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Operation

\[(Di) - 1 \Rightarrow Di\]
\[(M) - 1 \Rightarrow M\]

Syntax Variations

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC (Di)</td>
<td>INH</td>
</tr>
<tr>
<td>DEC.bwl (opremreg)</td>
<td>OPR/1/2/3</td>
</tr>
</tbody>
</table>

Description

Decrement a register \(Di\) or memory operand \(M\). The memory operand \(opremreg\) 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 \(Dj\), 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
<td>Δ</td>
</tr>
</tbody>
</table>

- \(N\): 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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>REGISTER Dj</td>
<td>4n</td>
</tr>
</tbody>
</table>

4n \ DEC \(Di\)
### Instruction Fields

**SIZE** - This field specifies 8-bit (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.

---

<table>
<thead>
<tr>
<th></th>
<th>SIZE</th>
<th>OPR POSTBYTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ap xb</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb x1</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb x1</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb x1</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb x1</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb x2 x1</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb x2 x1</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb x3 x2 x1</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb x3 x2 x1</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb x3 x2 x1</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb x3 x2 x1</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb x3 x2 x1</td>
<td>DEC.bwl</td>
</tr>
<tr>
<td>Ap xb x3 x2 x1</td>
<td>DEC.bwl</td>
</tr>
</tbody>
</table>

---

*Not appropriate for destination*

*INH version is more efficient*
DIVS  Signed Divide  DIVS

Operation

\[(D_j) \div (D_k) \Rightarrow D_d\]  
\[(D_j) \div \text{IMM} \Rightarrow D_d\]  
\[(D_j) \div (M) \Rightarrow D_d\]  
\[(M1) \div (M2) \Rightarrow D_d\]

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

<table>
<thead>
<tr>
<th>Regel</th>
<th>Dd,Dj,Dk</th>
<th>REG-REG</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIVS.B</td>
<td>Dd,Dj,#opr8i</td>
<td>REG-IMM1</td>
</tr>
<tr>
<td>DIVS.W</td>
<td>Dd,Dj,#opr16i</td>
<td>REG-IMM2</td>
</tr>
<tr>
<td>DIVS.L</td>
<td>Dd,Dj,#opr32i</td>
<td>REG-IMM4</td>
</tr>
<tr>
<td>DIVS.bwl</td>
<td>Dd,Dj,oprmemreg</td>
<td>REG-OPR/1/2/3</td>
</tr>
<tr>
<td>DIVS.bwpbwplDd,oprmemreg,oprmemreg</td>
<td>OPR/1/2/3-OPR/1/2/3</td>
<td></td>
</tr>
</tbody>
</table>

\(U\): 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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**DIVIDEND REGISTER Dj**

1B 3n mb

**DIVISOR REGISTER Dk**

1B 3n mb

#### REG-IMM1/2/4

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**IMMEDIATE DATA (Divisor)**

1B 3n mb

**DIVIDEND REGISTER Dj**

1B 3n mb

**IMM_SIZE (.B, .W, –, .L)**

1B 3n mb

#### REG-OPR/1/2/3

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**OPR POSTBYTE (for M2 divisor)**

1B 3n mb

**M2_SIZE (.B, .W, –, .L)**

1B 3n mb

### Example Instructions

- **DIVS.B**  
  - Dd, Dj, #opr8i
- **DIVS.W**  
  - Dd, Dj, #opr16i ; short-imm better for some values
- **DIVS.L**  
  - Dd, Dj, #opr32i ; short-imm better for some values

- **DIVS.bwl**  
  - Dd, Dj, Dk ; see more efficient REG-REG version
  - Dd, Dj, (#opr4, xys)
  - Dd, Dj, ((+-xy) | (xy+) | (-s) | (s))
  - Dd, Dj, ([Di, xys])
  - Dd, Dj, [Di, xy]
  - Dd, Dj, (#opr9, xys)
  - Dd, Dj, [opr9, xys]

- **DIVS.bwl**  
  - Dd, Dj, opru14
  - Dd, Dj, (opr18, Dd)

- **DIVS.bwl**  
  - Dd, Dj, opru18
  - Dd, Dj, opru18
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.

Linear S12 Core Reference Manual, Rev. 1.01

Freescale Semiconductor
Instruction Fields

QUOTIENT REGISTER - This field specifies the number of the data register Dd used for the result (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D6, 1:0:1=D7, 1:1:0=D0, and 1:1:1=D1).

DIVIDEND REGISTER - This field specifies the number of the data register Dj used as dividend (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D6, 1:0:1=D7).

DIVISOR REGISTER - This field specifies the number of the data register Dk used as divisor (0:0:0=D2, 0:0:1=D3, 0:1:0=D4, 0:1:1=D5, 1:0:0=D6, 1:0:1=D7, 1:1:0=D0, and 1:1:1=D1).

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

Operation

\[(D_j) \div (D_k) \Rightarrow D_d\]
\[(D_j) \div \text{IMM} \Rightarrow D_d\]
\[(D_j) \div (M) \Rightarrow D_d\]
\[(M1) \div (M2) \Rightarrow D_d\]

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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1B</th>
<th>3n</th>
<th>mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1B</td>
<td>3n</td>
<td>mb</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>DIVIDEND REGISTER Dj</td>
<td>QUOTIENT REGISTER Dd</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DIVISOR REGISTER Dk</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1B 3n mb

DIVU  Dd, Dj, Dk

#### REG-IMM1/2/4

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1B</th>
<th>3n</th>
<th>mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1B</td>
<td>3n</td>
<td>mb</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>DIVIDEND REGISTER Dj</td>
<td>QUOTIENT REGISTER Dd</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IMMEDIATE DATA (divisor)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1B</th>
<th>3n</th>
<th>mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1B</td>
<td>3n</td>
<td>mb</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>DIVIDEND REGISTER Dj</td>
<td>QUOTIENT REGISTER Dd</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OPR POSTBYTE (for M2 divisor)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(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, (opr4, 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]
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 ($0:0:0=D_2$, $0:0:1=D_3$, $0:1:0=D_4$, $0:1:1=D_5$, $1:0:0=D_0$, $1:0:1=D_1$, $1:1:0=D_6$, and $1:1:1=D_7$).

DIVIDEND REGISTER - This field specifies the number of the data register $D_j$ used as dividend ($0:0:0=D_2$, $0:0:1=D_3$, $0:1:0=D_4$, $0:1:1=D_5$, $1:0:0=D_0$, $1:0:1=D_1$, $1:1:0=D_6$, and $1:1:1=D_7$).

DIVISOR REGISTER - This field specifies the number of the data register $D_k$ used as divisor ($0:0:0=D_2$, $0:0:1=D_3$, $0:1:0=D_4$, $0:1:1=D_5$, $1:0:0=D_0$, $1:0:1=D_1$, $1:1:0=D_6$, and $1:1:1=D_7$).

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

Operation

\[(D_i) \ ^\ (M) \Rightarrow D_i\]

Syntax Variations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOR (D_i, #\text{oprmemsz})</td>
<td>IMM1/2/4</td>
</tr>
<tr>
<td>EOR (D_i, \text{oprmemreg})</td>
<td>OPR/1/2/3</td>
</tr>
</tbody>
</table>

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, \(\text{oprmemreg}\) 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>SD REGISTER (D_i)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **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 \(D_i\))

<table>
<thead>
<tr>
<th>1B</th>
<th>7p</th>
<th>i1</th>
<th>EOR</th>
<th>(D_i, #\text{opr8i}) ; for (D_i = 8)-bit (D_0) or (D_1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1B</td>
<td>7p</td>
<td>i2</td>
<td>EOR</td>
<td>(D_i, #\text{opr16i}) ; for (D_i = 16)-bit (D_2, D_3, D_4,) or (D_5)</td>
</tr>
<tr>
<td>1B</td>
<td>7p</td>
<td>i4</td>
<td>i3</td>
<td>i2</td>
</tr>
</tbody>
</table>
Chapter 6 Instruction Glossary

OPR/1/2/3

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

OPR POSTBYTE

%(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,#oprxe4i : -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,opru24
1B 8q xb x3 x2 x1  EOR  Di,[opr24]

Instruction Fields

SD REGISTER Di - This field specifies the number of the data register Di 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 Di.

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-indirect memory location.
EXG

Exchange Register Contents

**Syntax Variations**

<table>
<thead>
<tr>
<th>EXG</th>
<th>cpureg, cpureg</th>
</tr>
</thead>
</table>

**Addressing Modes**

<table>
<thead>
<tr>
<th>INH</th>
</tr>
</thead>
</table>

**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**

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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

AE eb

FIRST (SOURCE) REGISTER

SECOND (DESTINATION) REGISTER

Linear S12 Core Reference Manual, Rev. 1.01
Table 6-1. Exchange and Sign-Extend Postbyte (eb) Coding Map

<table>
<thead>
<tr>
<th>source</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>D5</th>
<th>D0</th>
<th>D1</th>
<th>D6</th>
<th>D7</th>
<th>X</th>
<th>Y</th>
<th>S</th>
<th>CCH</th>
<th>CCL</th>
<th>CCW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>D2</td>
<td>-0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
</tr>
<tr>
<td>D3</td>
<td>-1</td>
<td>D2</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>CCW</td>
</tr>
<tr>
<td>D4</td>
<td>-2</td>
<td>D2</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>CCW</td>
</tr>
<tr>
<td>D5</td>
<td>-3</td>
<td>D2</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>CCW</td>
</tr>
<tr>
<td>D0</td>
<td>-4</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>CCH</td>
<td>CCL</td>
<td>Big</td>
</tr>
<tr>
<td>D1</td>
<td>-5</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>CCH</td>
<td>CCL</td>
<td>Big</td>
</tr>
<tr>
<td>D6</td>
<td>-6</td>
<td>sex:D2</td>
<td>sex:D3</td>
<td>sex:D4</td>
<td>sex:D5</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>sex:CCW</td>
</tr>
<tr>
<td>D7</td>
<td>-7</td>
<td>sex:D2</td>
<td>sex:D3</td>
<td>sex:D4</td>
<td>sex:D5</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>sex:CCW</td>
</tr>
<tr>
<td>X</td>
<td>-8</td>
<td>sex:D2</td>
<td>sex:D3</td>
<td>sex:D4</td>
<td>sex:D5</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>sex:CCW</td>
</tr>
<tr>
<td>Y</td>
<td>-9</td>
<td>sex:D2</td>
<td>sex:D3</td>
<td>sex:D4</td>
<td>sex:D5</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>sex:CCW</td>
</tr>
<tr>
<td>S</td>
<td>-A</td>
<td>sex:D2</td>
<td>sex:D3</td>
<td>sex:D4</td>
<td>sex:D5</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>sex:CCW</td>
</tr>
<tr>
<td>reserved</td>
<td>-B</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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**

**Operation**

\((D_i) + 1 \Rightarrow D_i\)

\((M) + 1 \Rightarrow M\)

**Syntax Variations**

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>INC (D_i)</td>
<td>INH</td>
</tr>
<tr>
<td>INC.bwl (oprmemreg)</td>
<td>OPR/1/2/3</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
</tr>
</tbody>
</table>

- **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**

\[
\begin{array}{cccccccc}
7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\hline
0 & 0 & 1 & 1 & 0 & sd register \(D_i\) & \hline
3n & INC & \(D_i\) & \\
\end{array}
\]
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=\(D_2\), 0:0:1=\(D_3\), 0:1:0=\(D_4\), 0:1:1=\(D_5\), 1:0:0=\(D_6\), 1:0:1=\(D_1\), 1:1:0=\(D_6\), and 1:1:1=\(D_7\)).

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.
# Chapter 6 Instruction Glossary

## JMP

### Operation

Effective Address ⇒ PC

### Syntax Variations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP opr24a</td>
<td>EXT3</td>
</tr>
<tr>
<td>JMP oprmemreg</td>
<td>OPR/1/2/3</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Detailed Instruction Formats

#### EXT3

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- BA a3 a2 a1
- JMP opr24a

#### OPR/1/2/3

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- OPR POSTBYTE
- (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)
- (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

<table>
<thead>
<tr>
<th>AA xb</th>
<th>JMP</th>
<th>opr24a</th>
<th>#opr2xe4i ;not appropriate for destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA xb</td>
<td>JMP</td>
<td>opr4,xys</td>
<td>Di ;not appropriate for a jump destination</td>
</tr>
<tr>
<td>AA xb</td>
<td>JMP</td>
<td>opr4,xys</td>
<td>((+-xy)</td>
</tr>
<tr>
<td>AA xb</td>
<td>JMP</td>
<td>opr4,xys</td>
<td>(DI,xys)</td>
</tr>
<tr>
<td>AA xb</td>
<td>JMP</td>
<td>opr4,xys</td>
<td>[DI,xy]</td>
</tr>
<tr>
<td>AA xb</td>
<td>JMP</td>
<td>opr4,xys</td>
<td>(oprs9,xysp)</td>
</tr>
<tr>
<td>AA xb</td>
<td>JMP</td>
<td>opr4,xys</td>
<td>[oprs9,xysp]</td>
</tr>
<tr>
<td>AA xb</td>
<td>JMP</td>
<td>opr4,xys</td>
<td>opru14</td>
</tr>
</tbody>
</table>
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.

AA xb x2 x1    JMP (opr18, Di)
AA xb x2 x1    JMP opr18; 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 (opr24, Di)
AA xb x3 x2 x1 JMP opr24; EXT version is more efficient
AA xb x3 x2 x1 JMP [opr24]
**JSR**

**Jump to Subroutine**

**Operation**

\[
\text{(SP)} - 3 \Rightarrow \text{SP} \\
\text{RTN}[23:0] \Rightarrow \text{M}_{\text{SP}} : \text{M}_{\text{SP} + 1} : \text{M}_{\text{SP} + 2} \\
\text{Effective Address} \Rightarrow \text{PC}
\]

**Syntax Variations**

| JSR opr24a | EXT3 |
| JSR oprmem | 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.

Decrerals 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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Detailed Instruction Formats**

**EXT3**

\[
\begin{array}{cccccccccc}
7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
BB & a3 & a2 & a1 & JSR & opr24a
\end{array}
\]

**OPR/1/2/3**

\[
\begin{array}{cccccccccc}
7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
AB & xb & OPR POSTBYTE & (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE) & (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE) & (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)
\end{array}
\]

| AB xb | JSR | #opr2xe4i ;not appropriate for destination |
| AB xb | JSR | Di ;not appropriate for a jump destination |

Freescale Semiconductor
### 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.

<table>
<thead>
<tr>
<th>Format</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB xb</td>
<td>JSR (opru4, xys)</td>
</tr>
<tr>
<td>AB xb</td>
<td>JSR (+-xy)</td>
</tr>
<tr>
<td>AB xb</td>
<td>JSR (Di, xys)</td>
</tr>
<tr>
<td>AB xb</td>
<td>JSR [Di, xy]</td>
</tr>
<tr>
<td>AB xb x1</td>
<td>JSR (oprs9, xysp)</td>
</tr>
<tr>
<td>AB xb x1</td>
<td>JSR [oprs9, xysp]</td>
</tr>
<tr>
<td>AB xb x1</td>
<td>JSR opru14</td>
</tr>
<tr>
<td>AB xb x2 x1</td>
<td>JSR (opru18, Di)</td>
</tr>
<tr>
<td>AB xb x2 x1</td>
<td>JSR opru18 ;EXT version is just as efficient</td>
</tr>
<tr>
<td>AB xb x3 x2 x1</td>
<td>JSR (opr24, xysp)</td>
</tr>
<tr>
<td>AB xb x3 x2 x1</td>
<td>JSR [opr24, xysp]</td>
</tr>
<tr>
<td>AB xb x3 x2 x1</td>
<td>JSR (opru24, Di)</td>
</tr>
<tr>
<td>AB xb x3 x2 x1</td>
<td>JSR opr24 ;EXT version is more efficient</td>
</tr>
<tr>
<td>AB xb x3 x2 x1</td>
<td>JSR [opr24]</td>
</tr>
</tbody>
</table>
**LD**

**Load**

(Di, X, Y, or SP)

**Operation**

(M) ⇒ Di

(M) ⇒ X

(M) ⇒ Y

(M) ⇒ SP

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD  Di,#oprimmsz</td>
<td>IMM1/2/4 (same size as Di)</td>
</tr>
<tr>
<td>LD  Di,opr24a</td>
<td>EXT3 (24-bit address)</td>
</tr>
<tr>
<td>LD  Di,oprmemreg</td>
<td>OPR1/2/3</td>
</tr>
<tr>
<td>LD  xy,#opr18i</td>
<td>IMM2 (efficient 18-bit)</td>
</tr>
<tr>
<td>LD  xy,#opr24i</td>
<td>IMM3 (same size as X or Y)</td>
</tr>
<tr>
<td>LD  xy,opr24a</td>
<td>EXT3 (24-bit address)</td>
</tr>
<tr>
<td>LD  xy,oprmemreg</td>
<td>OPR1/2/3</td>
</tr>
<tr>
<td>LD  S,#opr24i</td>
<td>IMM3 (same size as SP)</td>
</tr>
<tr>
<td>LD  S,oprmemreg</td>
<td>OPR1/2/3</td>
</tr>
</tbody>
</table>

**Description**

Load a register Di, 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 Di, 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 Di, 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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>I</th>
<th>PL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Δ</td>
<td>Δ</td>
</tr>
</tbody>
</table>

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)\(^i\)

<table>
<thead>
<tr>
<th>7 6 5 4 3 2 1 0</th>
<th>REGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1 0 0 0 0</td>
<td>IMMEDIATE DATA</td>
</tr>
<tr>
<td></td>
<td>(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D)</td>
</tr>
<tr>
<td></td>
<td>(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D)</td>
</tr>
<tr>
<td></td>
<td>(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF D)</td>
</tr>
</tbody>
</table>

- **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)\(^i\)

<table>
<thead>
<tr>
<th>7 6 5 4 3 2 1 0</th>
<th>REGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0 0 0 0 0</td>
<td>Bn</td>
</tr>
<tr>
<td></td>
<td>ADDRESS[23:16]</td>
</tr>
<tr>
<td></td>
<td>ADDRESS[15:8]</td>
</tr>
<tr>
<td></td>
<td>ADDRESS[7:0]</td>
</tr>
</tbody>
</table>

- **Bn a3 a2 a1**: `LD Di,opr24a

#### OPR/1/2/3 (D)\(^i\)

<table>
<thead>
<tr>
<th>7 6 5 4 3 2 1 0</th>
<th>REGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0 0 0 0 0</td>
<td>An</td>
</tr>
<tr>
<td></td>
<td>OPR POSTBYTE</td>
</tr>
<tr>
<td></td>
<td>(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)</td>
</tr>
<tr>
<td></td>
<td>(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)</td>
</tr>
<tr>
<td></td>
<td>(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)</td>
</tr>
</tbody>
</table>

- **An xb**: `LD Di,#oprsxe4i ; -1, +1, 2, 3...14, 15
- **An xb**: `LD Di,Dj
- **An xb**: `LD Di,(opru4,xyz)
- **An xb**: `LD Di,((+-xy)|(xy+)|(-s)|(s+))
- **An xb**: `LD Di,(Dj,xyz)
- **An xb**: `LD Di,[Dj,xy]
- **An xb x1**: `LD Di,(oprs9,xyzp)
- **An xb x1**: `LD Di,[oprs9,xyzp]
- **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,(opru24,xyzp)
- **An xb x3 x2 x1**: `LD Di,[opru24,xyzp]
- **An xb x3 x2 x1**: `LD Di,opru24
- **An xb x3 x2 x1**: `LD Di,[opru24]

#### IMM2, IMM3 (X or Y)

<table>
<thead>
<tr>
<th>7 6 5 4 3 2 1 0</th>
<th>op</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 0 0 0 0 0</td>
<td>IMMEDIATE DATA[17:16]</td>
</tr>
<tr>
<td></td>
<td>IMMEDIATE DATA[15:8]</td>
</tr>
<tr>
<td></td>
<td>IMMEDIATE DATA[7:0]</td>
</tr>
</tbody>
</table>

- **op i2 i1**: `LD xy,#opr18i ; uses 4 opcodes ea. for X & Y
### EXT3 (X or Y)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-6</td>
<td>5-4</td>
</tr>
<tr>
<td>3-2</td>
<td>1-0</td>
</tr>
</tbody>
</table>

- **IMMEDIATE DATA[23:16]**
- **IMMEDIATE DATA[16:8]**
- **IMMEDIATE DATA[7:0]**

#### LD
- **9p i3 i2 i1**
- **xy, #opr24i**

### OPR/1/2/3 (X or Y)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-6</td>
<td>5-4</td>
</tr>
<tr>
<td>3-2</td>
<td>1-0</td>
</tr>
</tbody>
</table>

- **OPR POSTBYTE**
- **(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)**

#### LD
- **Bp a3 a2 a1**
- **xy, opr24a**

### IMM3 (S)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-6</td>
<td>5-4</td>
</tr>
<tr>
<td>3-2</td>
<td>1-0</td>
</tr>
</tbody>
</table>

- **IMMEDIATE DATA[23:16]**
- **IMMEDIATE DATA[16:8]**
- **IMMEDIATE DATA[7:0]**

#### LD
- **1B 03 i3 i2 i1**
- **S, #opr24i**
**OPR/1/2/3 (SP)**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**OPR POSTBYTE**

| 1B 00 xb | LD | S,#opraxe4i ; -1, +1, 2, 3...14, 15 |
| 1B 00 xb | LD | S,Dj |
| 1B 00 xb | LD | S,(opru4,sys) |
| 1B 00 xb | LD | S,((-xy) | (xy+-) | (-s) | (s+)) |
| 1B 00 xb | LD | S,(Dj,sys) |
| 1B 00 xb | LD | S,[Dj,xy] |
| 1B 00 xb x1 | LD | S,(oprs9,xyys) |
| 1B 00 xb x1 | LD | S,[oprs9,xyys] |
| 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,(opru24,xyys) |
| 1B 00 xb x3 x2 x1 | LD | S,[opru24,xyys] |
| 1B 00 xb x3 x2 x1 | LD | S,(opru24,Dj) |
| 1B 00 xb x3 x2 x1 | LD | S,opru24 |
| 1B 00 xb x3 x2 x1 | LD | S,[opru24] |

**Instruction Fields**

- **REGISTER** - This field specifies the number of the data register Di 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 Di, 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

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
<table>
<thead>
<tr>
<th>LEA</th>
<th>oprmemreg</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEA D67</td>
<td>oprmemreg</td>
<td>OPR/1/2/3</td>
</tr>
<tr>
<td>LEA S</td>
<td>oprmemreg</td>
<td>OPR/1/2/3</td>
</tr>
<tr>
<td>LEA xy</td>
<td>oprmemreg</td>
<td>OPR/1/2/3</td>
</tr>
<tr>
<td>LEA S (opr8i, S)</td>
<td>IMM1 (8-bit signed offset)</td>
<td></td>
</tr>
<tr>
<td>LEA xy (opr8i, xy)</td>
<td>IMM1 (8-bit signed offset)</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Detailed Instruction Formats

### OPR/1/2/3 (D6 or D7)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>D7/D6</td>
</tr>
</tbody>
</table>

#### OPR POSTBYTE

**(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)**

**(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)**

**(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)**

- **0p xb**
  - LEA \[D67,#\text{oprsxedi} \]; not appropriate for LEA
- **0p xb**
  - LEA \[D67,Dj \]; not appropriate for LEA
- **0p xb**
  - LEA \[D67,(\text{opru4},\text{sys})\]
- **0p xb**
  - LEA \[D67,((+-xy)|(xy+-)|(+-s)|(s+))\]
- **0p xb**
  - LEA \[D67,([Dj,\text{sys})\]
- **0p xb**
  - LEA \[D67,\text{opru14}\]
- **0p xb x1**
  - LEA \[D67,\text{oprus9,\text{sys}}\]
- **0p xb x1**
  - LEA \[D67,\text{oprus9,\text{sys}}\]
- **0p xb x1**
  - LEA \[D67,\text{opru14}\]
- **0p xb x2 x1**
  - LEA \[D67,\text{opru18,\text{dj}}\]
- **0p xb x2 x1**
  - LEA \[D67,\text{opru18}\]
- **0p xb x3 x2 x1**
  - LEA \[D67,\text{opru24,\text{sys}}\]
- **0p xb x3 x2 x1**
  - LEA \[D67,\text{opru24,\text{sys}}\]
- **0p xb x3 x2 x1**
  - LEA \[D67,\text{opru24,\text{dj}}\]
- **0p xb x3 x2 x1**
  - LEA \[D67,\text{opru24}\]

### OPR/1/2/3 (SP)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

#### OPR POSTBYTE

**(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)**

**(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)**

**(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)**

- **0A xb**
  - LEA \[S,#\text{oprsxedi} \]; not appropriate for LEA
- **0A xb**
  - LEA \[S,Dj \]; not appropriate for LEA
- **0A xb**
  - LEA \[S,(\text{opru4},\text{sys})\]
- **0A xb**
  - LEA \[S,((+-xy)|(xy+-)|(+-s)|(s+))\]
- **0A xb**
  - LEA \[S,([Dj,\text{sys})\]
- **0A xb**
  - LEA \[S,\text{opru14}\]
- **0A xb x1**
  - LEA \[S,\text{oprus9,\text{sys}}\]
- **0A xb x1**
  - LEA \[S,\text{oprus9,\text{sys}}\]
- **0A xb x1**
  - LEA \[S,\text{opru14}\]
- **0A xb x2 x1**
  - LEA \[S,\text{opru18,\text{dj}}\]
- **0A xb x2 x1**
  - LEA \[S,\text{opru18}\]
- **0A xb x3 x2 x1**
  - LEA \[S,\text{opru24,\text{sys}}\]
- **0A xb x3 x2 x1**
  - LEA \[S,\text{opru24,\text{sys}}\]
- **0A xb x3 x2 x1**
  - LEA \[S,\text{opru24,\text{dj}}\]
- **0A xb x3 x2 x1**
  - LEA \[S,\text{opru24}\]
- **0A xb x3 x2 x1**
  - LEA \[S,\text{opru24}\]
### 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 Di variations of OPR addressing are not appropriate for LEA because these values do not have an associated effective address.
Logical Shift Left

**Operation**

![Logical Shift Left Diagram]

**Syntax Variations**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSL</td>
<td>REG-REG</td>
</tr>
<tr>
<td>LSL</td>
<td>REG-IMM</td>
</tr>
<tr>
<td>LSL.bwpl</td>
<td>OPR/1/2/3-IMM</td>
</tr>
<tr>
<td>LSL.bwpl</td>
<td>OPR/1/2/3-OPR/1/2/3</td>
</tr>
<tr>
<td>LSL.bwpl</td>
<td>OPR/1/2/3-OPR/1/2/3</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>DESTINATION REGISTER ( D_d )</td>
<td></td>
</tr>
<tr>
<td>A/L=0</td>
<td>L/R=1</td>
<td>1</td>
<td>0</td>
<td>N[0]</td>
<td>SOURCE REGISTER ( D_s )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>PARAMETER REGISTER ( D_n )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>DESTINATION REGISTER ( D_d )</td>
<td></td>
</tr>
<tr>
<td>A/L=0</td>
<td>L/R=1</td>
<td>0</td>
<td>0</td>
<td>N[0]</td>
<td>SOURCE REGISTER ( D_s )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>N[0]</td>
<td></td>
</tr>
<tr>
<td>A/L=0</td>
<td>L/R=1</td>
<td>1</td>
<td>0</td>
<td>N[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>DESTINATION REGISTER ( D_d )</td>
<td></td>
</tr>
<tr>
<td>A/L=0</td>
<td>L/R=1</td>
<td>1</td>
<td>0</td>
<td>N[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPR POSTBYTE (specifies source operand to be shifted)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>DESTINATION REGISTER ( D_d )</td>
<td></td>
</tr>
<tr>
<td>A/L=0</td>
<td>L/R=1</td>
<td>1</td>
<td>0</td>
<td>N[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPR POSTBYTE (specifies source operand to be shifted)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Linear S12 Core Reference Manual, Rev. 1.01

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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>1n</th>
<th>1o</th>
<th>1f</th>
<th>1g</th>
<th>1h</th>
<th>Dest Register D&lt;sub&gt;d&lt;/sub&gt;</th>
</tr>
</thead>
</table>

**A/L=0**  
**L/R=1**  
**N[0]**  

**OPR POSTBYTE** (specifies source operand to be shifted)

**(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)**

**(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)**

| 0 | 1 | 1 | 1 | 1 | xb |

1n sb xb xb LSL.bpl #oprxe4i,#opr5i  
1n sb xb xb LSL.bpl Di,#opr5i ;see more efficient REG-IMM version  
1n sb xb xb LSL.bpl (opr4,xy),#opr5i  
1n sb xb xb LSL.bpl (xy+-)| (xy+-)| (s)| (s+),#opr5i  
1n sb xb xb LSL.bpl (Di,xy),#opr5i  
1n sb xb xb LSL.bpl [Di,xy],#opr5i  
1n sb xb x1 xb LSL.bpl (oprs9,xy),#opr5i  
1n sb xb x1 xb LSL.bpl [oprs9,xy],#opr5i  
1n sb xb x1 xb LSL.bpl opru14,#opr5i  
1n sb xb x2 x1 xb LSL.bpl (opru18,Di),#opr5i  
1n sb xb x2 x1 xb LSL.bpl opru18,#opr5i  
1n sb xb x3 x2 x1 xb LSL.bpl (opr24,xy),#opr5i  
1n sb xb x3 x2 x1 xb LSL.bpl [opr24,xy],#opr5i  
1n sb xb x3 x2 x1 xb LSL.bpl opru24,#opr5i  
1n sb xb x3 x2 x1 xb LSL.bpl opr24,#opr5i  
1n sb xb x3 x2 x1 xb LSL.bpl [opr24],#opr5i
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)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
</tr>
</tbody>
</table>

- **A/L=0**: Linear
- **L/R=1**: Right
- **N[0]**: Shift by 1 \((N[0]=0)\) or by 2 \((N[0]=1)\)

**OPR POSTBYTE**: Specifies source operand to be shifted

**OPL**: Postbyte

**xb**: (Optional address-byte depending on address-mode)

**sb**: (Optional address-byte depending on address-mode)

#### Examples

1. `ln sb xb`  
   `LSL.bpl #oprxe4i,#oprli`

2. `ln sb xb`  
   `LSL.bpl Ds,#oprli` \(\text{; see more efficient REG-IMM version}\)

3. `ln sb xb`  
   `LSL.bpl (opru4,xys),#oprli`

4. `ln sb xb`  
   `LSL.bpl ((+-xy)|(xy+-)|(s)|{(s)}),#oprli`

5. `ln sb xb`  
   `LSL.bpl (Di,xys),#oprli`

6. `ln sb xb`  
   `LSL.bpl [Di,xy],#oprli`

7. `ln sb xb x1`  
   `LSL.bpl (oprs9,xysp),#oprli`

8. `ln sb xb x1`  
   `LSL.bpl [oprs9,xysp],#oprli`

9. `ln sb xb x1`  
   `LSL.bpl opru14,#oprli`

10. `ln sb xb x2 x1`  
    `LSL.bpl (opru18,Di),#oprli`

11. `ln sb xb x2 x1`  
    `LSL.bpl opru18,#oprli`

12. `ln sb xb x3 x2 x1`  
    `LSL.bpl (opr24,xysp),#oprli`

13. `ln sb xb x3 x2 x1`  
    `LSL.bpl [opr24,xysp],#oprli`

14. `ln sb xb x3 x2 x1`  
    `LSL.bpl (opru24,Di),#oprli`

15. `ln sb xb x3 x2 x1`  
    `LSL.bpl opru24,#oprli`

16. `ln sb xb x3 x2 x1`  
    `LSL.bpl opr24,#oprli`

17. `ln sb xb x3 x2 x1`  
    `LSL.bpl [opr24],#oprli`
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 Dd - This field specifies data register Dd (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 Ds - This field specifies data register Ds (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 Dn - This field specifies the number of the data register Dn (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**

**Operation**

![Logical Shift Right Diagram]

**Syntax Variations**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSR D, Ds, Dn</td>
<td>REG-REG</td>
</tr>
<tr>
<td>LSR D, Ds, #opr5i</td>
<td>REG-IMM</td>
</tr>
<tr>
<td>LSR.bwpl D, oprmemreg, #opr5i</td>
<td>OPR/1/2/3-IMM</td>
</tr>
<tr>
<td>LSR.bwpl D, oprmemreg, oprmemreg</td>
<td>OPR/1/2/3-OPR/1/2/3</td>
</tr>
<tr>
<td>LSR.bwpl oprmemreg, #opr1i</td>
<td>OPR/1/2/3-IMM</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
<td>0</td>
<td>Δ</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**DESTINATION REGISTER D<sub>d</sub>**

**SOURCE REGISTER D<sub>s</sub>**

**PARAMETER REGISTER D<sub>n</sub>**

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**DESTINATION REGISTER D<sub>d</sub>**

**SOURCE REGISTER D<sub>s</sub>**

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**DESTINATION REGISTER D<sub>d</sub>**

**SOURCE REGISTER D<sub>s</sub>**

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**DESTINATION REGISTER D<sub>d</sub>**

**SOURCE REGISTER D<sub>s</sub>**


### OPR POSTBYTE (specifies source operand to be shifted)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

---

**Notes:**

- L/R=0
- A/L=0
- D<sub>d</sub>, D<sub>s</sub>, D<sub>n</sub>
- LSR
- opr1i
- oprs9
- opru4
- opru14
- opru18
- opr24
OPR/1/2/3-IMM (normal shift by 0 to 31 positions)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>DESTINATION REGISTER Dd</td>
<td>1n</td>
<td></td>
</tr>
<tr>
<td>A/L=0</td>
<td>L/R=0</td>
<td>1</td>
<td>1</td>
<td>N[0]</td>
<td>0</td>
<td>SIZE (.B, .W, .P, .L)</td>
<td>sb</td>
<td></td>
</tr>
</tbody>
</table>

OPR POSTBYTE (specifies source operand to be shifted)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N[4:1]</td>
<td>xb</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1n sb xb xb  LSR.bmpl  #oprsxe4i,#opr5i
1n sb xb xb  LSR.bmpl  Di,#opr5i ;see more efficient REG-IMM version
1n sb xb xb  LSR.bmpl  opru4,xy,#opr5i
1n sb xb xb  LSR.bmpl  ((+-xy)|(xy+-)|(s-)|(s+)),#opr5i
1n sb xb xb  LSR.bmpl  (Di,xy),#opr5i
1n sb xb xb  LSR.bmpl  [Di,xy],#opr5i
1n sb xb 1 x 1b  LSR.bmpl  opru9,xy,#opr5i
1n sb xb 1 x 1b  LSR.bmpl  opru9,xy,#opr5i
1n sb xb 1 x 1b  LSR.bmpl  opru14,#opr5i
1n sb xb x 2 x 1 xb  LSR.bmpl  opru18,Di,#opr5i
1n sb xb x 2 x 1 xb  LSR.bmpl  opru18,Di,#opr5i
1n sb xb x 3 x 2 x 1 xb  LSR.bmpl  opru24,xy,#opr5i
1n sb xb x 3 x 2 x 1 xb  LSR.bmpl  opru24,xy,#opr5i
1n sb xb x 3 x 2 x 1 xb  LSR.bmpl  opru24,Di,#opr5i
1n sb xb x 3 x 2 x 1 xb  LSR.bmpl  opru24,Di,#opr5i
1n sb xb x 3 x 2 x 1 xb  LSR.bmpl  opru24,Di,#opr5i
1n sb xb x 3 x 2 x 1 xb  LSR.bmpl  opru24,Di,#opr5i
1n sb xb x 3 x 2 x 1 xb  LSR.bmpl  [opr24],#opr5i
1n sb xb x 3 x 2 x 1 xb  LSR.bmpl  [opr24],#opr5i
### Opcode postbyte | Source operand | Parameter # of shifts | 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 | #oprsx4i, ~ #oprsx4i | ~ #oprsx4i |
 | 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 | (opru24, xysp), | (opru24, xysp) |
 | xb x3 x2 x1 | xb x3 x2 x1 | [opru24, xysp], | [opru24, xysp] |
 | xb x3 x2 x1 | xb x3 x2 x1 | (opru24, Dj), | (opru24, Dk) |
 | xb x3 x2 x1 | xb x3 x2 x1 | opru24, | opru24 |
 | xb x3 x2 x1 | xb x3 x2 x1 | [opru24], | [opru24] |

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)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>A/L=0</td>
<td>L/R=0</td>
<td>1</td>
<td>1</td>
<td>N[0]</td>
<td>1</td>
<td>SIZE (.B, .W, .P, .L)</td>
<td></td>
</tr>
</tbody>
</table>

**OPR POSTBYTE** (specifies source operand to be shifted)

**(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)**

**(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)**

- `ln sb xb` LSR.bpl `#oprsxe4i,#opr1i`
- `ln sb xb` LSR.bpl `Ds,#opr1i` ; see more efficient REG-IMM version
- `ln sb xb` LSR.bpl `(opru4,xys),#opr1i`
- `ln sb xb` LSR.bpl `{(+xy)|(xy+-)|(s)|(+s)},#opr1i`
- `ln sb xb` LSR.bpl `(Di,xys),#opr1i`
- `ln sb xb` LSR.bpl `[Di,xy],#opr1i`
- `ln sb xb x1` LSR.bpl `(oprs9,xysp),#opr1i`
- `ln sb xb x1` LSR.bpl `[oprs9,xysp],#opr1i`
- `ln sb xb x1` LSR.bpl `opru14,#opr1i`
- `ln sb xb x2 x1` LSR.bpl `opru18,Di),#opr1i`
- `ln sb xb x2 x1` LSR.bpl `opru18,#opr1i`
- `ln sb xb x3 x2 x1` LSR.bpl `(opru24,xysp),#opr1i`
- `ln sb xb x3 x2 x1` LSR.bpl `[opru24,xysp],#opr1i`
- `ln sb xb x3 x2 x1` LSR.bpl `(opru24,Di),#opr1i`
- `ln sb xb x3 x2 x1` LSR.bpl `opru24,#opr1i`
- `ln sb xb x3 x2 x1` LSR.bpl `[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 Dd - This field specifies data register Dd (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 Ds - This field specifies data register Ds (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 Dn - This field specifies the number of the data register Dn (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

Operation

\[ (D_j \times (D_k) + (D_d) \Rightarrow D_d ) \]
\[ (D_j \times IMM + (D_d) \Rightarrow D_d ) \]
\[ (D_j \times (M) + (D_d) \Rightarrow D_d ) \]
\[ (M1) \times (M2) + (D_d) \Rightarrow D_d \]

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

<table>
<thead>
<tr>
<th>U</th>
<th>I</th>
<th>P</th>
<th>L</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>RESULT REGISTER Dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>SOURCE 1 REGISTER Dj</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B</td>
<td>4q</td>
<td>mb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1B 4q mb

MACS

Dd, Dj, Dk

#### REG-IMM1/2/4

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>RESULT REGISTER Dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>SOURCE REGISTER Dj</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B</td>
<td>4q</td>
<td>mb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IMMEDIATE DATA

(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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>RESULT REGISTER Dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>SOURCE REGISTER Dj</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B</td>
<td>4q</td>
<td>mb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OPR POSTBYTE (for M2)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

1B 4q mb xb

MACS.bwl

Dd, Dj, #opru4

1B 4q mb xb

MACS.bwl

Dd, Dj, #opru18, Di

1B 4q mb xb

MACS.bwl

Dd, Dj, #opru14

1B 4q mb xb x1

MACS.bwl

Dd, Dj, [opru9, yxs]

1B 4q mb xb x1

MACS.bwl

Dd, Dj, [opru9, yxs]

1B 4q mb xb x1

MACS.bwl

Dd, Dj, opru4

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 x2 x1

MACS.bwl

Dd, Dj, (opru24, yxs)

1B 4q mb xb x2 x1

MACS.bwl

Dd, Dj, (opru24, Di)

1B 4q mb xb x2 x1

MACS.bwl

Dd, Dj, opru24

1B 4q mb xb x2 x1

MACS.bwl

Dd, Dj, opru24

1B 4q mb xb x3 x2 x1

MACS.bwl

Dd, Dj, (opru24, yxs)

1B 4q mb xb x3 x2 x1

MACS.bwl

Dd, Dj, (opru24, Di)

1B 4q mb xb x3 x2 x1

MACS.bwl

Dd, Dj, opru24

1B 4q mb xb x3 x2 x1

MACS.bwl

Dd, Dj, opru24

1B 4q mb xb x3 x2 x1

MACS.bwl

Dd, Dj, [opru24]
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.
Chapter 6 Instruction Glossary

MACU

Unsigned Multiply and Accumulate

Operation

\[(D_j) \times (D_k) + (D_d) \Rightarrow D_d\]
\[(D_j) \times \text{IMM} + (D_d) \Rightarrow D_d\]
\[(D_j) \times (M) + (D_d) \Rightarrow D_d\]
\[(M_1) \times (M_2) + (D_d) \Rightarrow D_d\]

Syntax Variations

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACU</td>
<td>Dd,Dj,Dk</td>
</tr>
<tr>
<td>MACU.B</td>
<td>Dd,Dj,#opr8i</td>
</tr>
<tr>
<td>MACU.W</td>
<td>Dd,Dj,#opr16i</td>
</tr>
<tr>
<td>MACU.L</td>
<td>Dd,Dj,#opr32i</td>
</tr>
<tr>
<td>MACU.bwl</td>
<td>Dd,Dj,oprmemreg</td>
</tr>
<tr>
<td>MACU.bwplbwpl</td>
<td>Dd,oprmemreg,oprmemreg</td>
</tr>
</tbody>
</table>

Addressing Modes

<table>
<thead>
<tr>
<th>REG-REG</th>
<th>REG-IMM1</th>
<th>REG-IMM2</th>
<th>REG-IMM4</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPR/1/2/3-OPR/1/2/3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description

Multiplies two unsigned operands, adds this product to a register Dd, and stores the accumulated result to register Dd. The first source operand may be a register Dj 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 Dk, 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

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 Dd 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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1B</th>
<th>4q</th>
<th>mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>RESULT REGISTER D&lt;d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>SOURCE 1 REGISTER Dj</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>SOURCE 2 REGISTER Dk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1B 4q mb
MACU Dd, Dj, Dk

#### REG-IMM1/2/4

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1B</th>
<th>4q</th>
<th>mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>RESULT REGISTER D&lt;d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>SOURCE REGISTER Dj</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 IMM_SIZE (.B, .W, −, .L)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1B</th>
<th>4q</th>
<th>mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>RESULT REGISTER D&lt;d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>SOURCE REGISTER Dj</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M2_SIZE (.B, .W, −, .L)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

1B 4q mb xb
MACU.bwl Dd, Dj, #opru4, #xy

1B 4q mb xb
MACU.bwl Dd, Dj, #opr18, Di

1B 4q mb xb
MACU.bwl Dd, Dj, (opr4, 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, xys)

1B 4q mb xb x1
MACU.bwl Dd, Dj, [oprs9, xys]

1B 4q mb xb x1
MACU.bwl Dd, Dj, opru4

1B 4q mb xb x2 x1
MACU.bwl Dd, Dj, (opr18, Di)

1B 4q mb xb x2 x1
MACU.bwl Dd, Dj, opru18

1B 4q mb xb x3 x2 x1
MACU.bwl Dd, Dj, (opr24, xys)

1B 4q mb xb x3 x2 x1
MACU.bwl Dd, Dj, [opr24, xys]

1B 4q mb xb x3 x2 x1
MACU.bwl Dd, Dj, (opr24, 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]
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 \( D_i \)

**Operation**

\[ \text{MAX}((D_i), (M)) \Rightarrow D_i \]

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- **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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1B 2q ( \text{xb} )</td>
<td>SD REGISTER ( D_i )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **OPR POSTBYTE (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)**
- **OPR POSTBYTE (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)**
- **OPR POSTBYTE (OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)**

1B 2q \( \text{xb} \)  MAXS \( D_i, \#\text{opr}x\text{e}4i \); -1, +1, 2, 3...14, 15
1B 2q \( \text{xb} \)  MAXS \( D_i, D_j \)
1B 2q \( \text{xb} \)  MAXS \( D_i, (\text{opru}4, \text{xys}) \)
1B 2q \( \text{xb} \)  MAXS \( D_i, (\{+-\text{xy}\} | (\text{xy}+-) | (-s) | (s+) \) \)
1B 2q \( \text{xb} \)  MAXS \( D_i, (D_j, \text{xys}) \)
1B 2q \( \text{xb} \)  MAXS \( D_i, [D_j, \text{xy}] \)
1B 2q \( \text{xb} \) \( x1 \)  MAXS \( D_i, (\text{oprs}9, \text{xy}p) \)
1B 2q \( \text{xb} \) \( x1 \)  MAXS \( D_i, [\text{oprs}9, \text{xy}p] \)
1B 2q \( \text{xb} \) \( x1 \)  MAXS \( D_i, \text{opru}14 \)
1B 2q \( \text{xb} \) \( x2 \) \( x1 \)  MAXS \( D_i, (\text{opru}18, D_j) \)
Instruction Fields

SD REGISTER Di - This field specifies the number of the data register Di 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 Di

**Operation**

\[
\text{MAX}((D_i, (M)) \Rightarrow D_i
\]

**Syntax Variations**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXU</td>
<td>OPR/1/2/3</td>
</tr>
<tr>
<td>Di, oprmemreg</td>
<td></td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[1B 1q xb\]

**OPR POSTBYTE**

(Optional Address-Byte Depending on Address-Mode)

\[1B 1q xb MAXU D_i, oprs9, xysp\]
\[1B 1q xb MAXU D_i, opru18, Dj\]
Instruction Fields

SD REGISTER Di - This field specifies the number of the data register Di 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 \( D_i \)

**Operation**

\[ \text{MIN}((D_i), (M)) \Rightarrow D_i \]

**Syntax Variations**

<table>
<thead>
<tr>
<th>MINS</th>
<th>( D_i, \text{opmemreg} )</th>
</tr>
</thead>
</table>

**Addressing Modes**

<table>
<thead>
<tr>
<th>OPR/1/2/3</th>
</tr>
</thead>
</table>

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- **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**

<table>
<thead>
<tr>
<th>( D_i, \text{opmemreg} )</th>
<th>( -1, +1, 2, 3...14, 15 )</th>
</tr>
</thead>
</table>

1B 2n | \( D_i, \text{#oprsex4i} \) |
1B 2n | \( D_i, D_j \) |
1B 2n | \( D_i, (\text{opru4}, \text{xy}) \) |
1B 2n | \( D_i, (\{+-\text{xy}\} | (xy+--) | (-s) | (s+) \) \) |
1B 2n | \( D_i, (D_j, \text{xy}) \) |
1B 2n | \( D_i, [\text{Dj, xy}] \) |
1B 2n | \( D_i, (\text{oprs9}, \text{yps}) \) |
1B 2n | \( D_i, [\text{oprs9}, \text{yps}] \) |
1B 2n | \( D_i, \text{opru14} \) |
1B 2n | \( D_i, (\text{opru18}, D_j) \) |
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 DI

Operation
MINU((DI), (M)) \(\Rightarrow\) DI

Syntax Variations

| MINU | DI, oprmemreg |

Addressing Modes

| OPR/1/2/3 |

Description
Subtracts the unsigned value of memory operand M from the unsigned value in register DI to determine which is smaller. The smaller of the two values is stored in register DI. The size of memory operand M is determined by the size of register DI.

CCR Details

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

1B 1n xb

Minu

DI, #oprSx6DI ; -1, +1, 2, 3...14, 15

DI, Dj

DI, (opr4, xys)

DI, ($(+-xy) \mid (xy+-) \mid (-s) \mid (s+)$)

DI, (Dj, xys)

DI, [Dj, xy]

DI, (opr9, xys)

DI, (opr9, xysp)

DI, [opr9, xys]

DI, [opr9, xysp]

DI, [opr14]

DI, (opr18, Dj)
SD REGISTER Di - This field specifies the number of the data register Di 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
(Dj) % (Dk) ⇒ Dd
(Dj) % IMM ⇒ Dd
(Dj) % (M) ⇒ Dd
(M1) % (M2) ⇒ Dd

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODS</td>
<td>REG-REG</td>
</tr>
<tr>
<td>MODS.B</td>
<td>REG-IMM1</td>
</tr>
<tr>
<td>MODS.W</td>
<td>REG-IMM2</td>
</tr>
<tr>
<td>MODS.L</td>
<td>REG-IMM4</td>
</tr>
<tr>
<td>MODS.bwl</td>
<td>REG-OPR/1/2/3</td>
</tr>
<tr>
<td>MODS.bwplbwplDd, oprimemreg, oprimemreg</td>
<td>OPR/1/2/3-OPR/1/2/3</td>
</tr>
</tbody>
</table>

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 Dd. The dividend may be a register Dj or an 8-bit (.B), 16-bit (.W), 24-bit (.P), or 32-bit (.L) memory operand M1. The divisor may be a register Dk, 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

<table>
<thead>
<tr>
<th>U</th>
<th>I</th>
<th>PL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
</tr>
</tbody>
</table>

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 Dd. Undefined after division by zero. Cleared otherwise.
C: Set if divisor was zero. Cleared otherwise. (Indicates division by zero).
### Detailed Instruction Formats

#### REG-REG

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **RESULT REGISTER Dd**
- **DIVIDEND REGISTER Dj**
- **DIVISOR REGISTER Dk**

#### REG-IMM1/2/4

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **RESULT REGISTER Dd**
- **IMMEDIATE DATA (Divisor)**

#### REG-OPR/1/2/3

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **RESULT REGISTER Dd**
- **OPR POSTBYTE (for M2 divisor)**

### Immediate Data Dependent on Size Specified in Suffix

- **IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX**

### OPR POSTBYTE (for M2 divisor)

- **OPR POSTBYTE DEPENDING ON ADDRESS-MODE**

---

**Linear S12 Core Reference Manual, Rev. 1.01**

232 Freescale Semiconductor
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 Dd 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 Dj 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 Dk 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

Operation

\((D_j) \% (D_k) \Rightarrow D_d\)
\((D_j) \% \text{IMM} \Rightarrow D_d\)
\((D_j) \% (M) \Rightarrow D_d\)
\((M1) \% (M2) \Rightarrow D_d\)

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODU (D_d,D_j,D_k)</td>
<td>REG-REG</td>
</tr>
<tr>
<td>MODU.B (D_d,D_j,#\text{opr8i})</td>
<td>REG-IMM1</td>
</tr>
<tr>
<td>MODU.W (D_d,D_j,#\text{opr16i})</td>
<td>REG-IMM2</td>
</tr>
<tr>
<td>MODU.L (D_d,D_j,#\text{opr32i})</td>
<td>REG-IMM4</td>
</tr>
<tr>
<td>MODU.W(W,D_j,\text{oprmemreg})</td>
<td>REG-OPR/1/2/3</td>
</tr>
<tr>
<td>MODU.W(W,D_d,\text{oprmemreg},\text{oprmemreg})</td>
<td>OPR/1/2/3-OPR/1/2/3</td>
</tr>
</tbody>
</table>

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 \(M_1\). 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 \(M_2\).

CCR Details

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Δ</td>
<td>Δ</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>DIVIDEND REGISTER Dj</td>
<td>DIVISOR REGISTER Dk</td>
<td></td>
</tr>
</tbody>
</table>

1B 3q mb MODU Dd, Dj, Dk

**REG-IMM1/2/4**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>IMMEDIATE DATA (divisor)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>DIVIDEND REGISTER Dj</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OPR POSTBYTE (for M2 divisor)

(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

1B 3q mb xb MODU.bwl Dd, Dj, #opru4
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, (opru24, xysp)
1B 3q mb xb x3 x2 x1 MODU.bwl Dd, Dj, [opru24, 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, opru24
1B 3q mb xb x3 x2 x1 MODU.bwl Dd, Dj, [opru24]
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 Dd 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 Dj 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 Dk 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)

**Operation**

\((M1) \Rightarrow M2\)

**Syntax Variations**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV.B #opr8i, oprmemreg</td>
<td>IMM1-OPR/1/2/3</td>
</tr>
<tr>
<td>MOV.W #opr16i, oprmemreg</td>
<td>IMM2-OPR/1/2/3</td>
</tr>
<tr>
<td>MOV.P #opr24i, oprmemreg</td>
<td>IMM3-OPR/1/2/3</td>
</tr>
<tr>
<td>MOV.L #opr32i, oprmemreg</td>
<td>IMM4-OPR/1/2/3</td>
</tr>
<tr>
<td>MOV.(bwpl) oprmemreg, oprmemreg</td>
<td>OPR/1/2/3-OPR/1/2/3</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
### Detailed Instruction Formats

**IMM1-OPR/1/2/3 (.B 8-bit byte)**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

0C i1 xb

MOV.B #opr8i,#opr8e4i ; 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,[opru9,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,(opru24,xysp)

0C i1 xb x3 x2 x1

MOV.B #opr8i,[opru24,xysp]

0C i1 xb x3 x2 x1

MOV.B #opr8i,(opru24,Di)

0C i1 xb x3 x2 x1

MOV.B #opr8i,opru24

0C i1 xb x3 x2 x1

MOV.B #opr8i,[opru24]

**IMM2-OPR/1/2/3 (.W 16-bit word)**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

0D i2 i1 xb

MOV.W #opr16i,#opr16e4i ; 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,xy]

0D i2 i1 xb

MOV.W #opr16i,[Di,xys]

0D i2 i1 xb x1

MOV.W #opr16i,(opru9,xysp)

0D i2 i1 xb x1

MOV.W #opr16i,[opru9,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,(opru24,xysp)

0D i2 i1 xb x3 x2 x1

MOV.W #opr16i,[opru24,xysp]

0D i2 i1 xb x3 x2 x1

MOV.W #opr16i,(opru24,Di)

0D i2 i1 xb x3 x2 x1

MOV.W #opr16i,opru24

0D i2 i1 xb x3 x2 x1

MOV.W #opr16i,[opru24]
**IMM3-OPR/1/2/3 (P 24-bit pointer)**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**IMMEDIATE DATA (source)**

**(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE)**

**(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE)**

**OPR POSTBYTE (destination)**

**(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 x1 MOV.P #opr24i,(opru18,Di)
0E i3 i2 i1 xb x1 MOV.P #opr24i,opru18
0E i3 i2 i1 xb x2 x1 MOV.P #opr24i,(opr24,xysp)
0E i3 i2 i1 xb x2 x1 MOV.P #opr24i,[opr24,xysp]
0E i3 i2 i1 xb x2 x1 MOV.P #opr24i,(opr24,Di)
0E i3 i2 i1 xb x2 x1 MOV.P #opr24i,opr24
0E i3 i2 i1 xb x2 x1 MOV.P #opr24i,[opr24]

**IMM4-OPR/1/2/3 (L 32-bit long-word)**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**IMMEDIATE DATA (source)**

**IMMEDIATE DATA[23:16]**

**IMMEDIATE DATA[15:8]**

**IMMEDIATE DATA[7:0]**

**OPR POSTBYTE (destination)**

**(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 x1 MOV.L #opr32i,(opru18,Di)
0F i4 i3 i2 i1 xb x2 x1 MOV.L #opr32i,(opr24,xysp)
0F i4 i3 i2 i1 xb x2 x1 MOV.L #opr32i,[opr24,xysp]
0F i4 i3 i2 i1 xb x2 x1 MOV.L #opr32i,(opr24,Di)
0F i4 i3 i2 i1 xb x2 x1 MOV.L #opr32i,opr24
0F i4 i3 i2 i1 xb x2 x1 MOV.L #opr32i,[opr24]
Short-immediate is not appropriate for the destination of a move instruction.

<table>
<thead>
<tr>
<th>Opcode</th>
<th>M1 object code</th>
<th>M2 object code</th>
<th>Instruction Mnemonic</th>
<th>Source Format for M1 (Source)</th>
<th>Source Format for M2 (Destination)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>xb</td>
<td>xb</td>
<td>MOV.L</td>
<td>opr32i,</td>
<td>opr24</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>opr32i, [opr24]</td>
<td></td>
</tr>
</tbody>
</table>

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

0F 14 i3 i2 i1 xb x3 x2 x1 MOV.L #opr32i, opr24
0F 14 i3 i2 i1 xb x3 x2 x1 MOV.L #opr32i, [opr24]

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]
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 instruction 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**

**Operation**

\[(D_j) \times (D_k) \Rightarrow D_d\]

\[(D_j) \times \text{IMM} \Rightarrow D_d\]

\[(D_j) \times (M) \Rightarrow D_d\]

\[(M1) \times (M2) \Rightarrow D_d\]

**Syntax Variations**

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULS</td>
<td>REG-REG</td>
</tr>
<tr>
<td>MULS.B</td>
<td>REG-IMM1</td>
</tr>
<tr>
<td>MULS.W</td>
<td>REG-IMM2</td>
</tr>
<tr>
<td>MULS.L</td>
<td>REG-IMM4</td>
</tr>
<tr>
<td>MULS.bwl Dd,Dj,oprmemreg</td>
<td>REG-OPR/1/2/3</td>
</tr>
<tr>
<td>MULS.bwplbwplDd,oprmemreg</td>
<td>OPR/1/2/3-OPR/1/2/3</td>
</tr>
</tbody>
</table>

**Description**

Multiplies two signed two’s complement operands and stores the signed two’s complement result to register Dd. The first source operand may be a register Dj 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 Dk, 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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
<td>Δ</td>
<td>0</td>
</tr>
</tbody>
</table>

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 Dd. Cleared otherwise.
C: Cleared.
### Detailed Instruction Formats

#### REG

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>RESULT REGISTER Dd</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SOURCE REGISTER Dj</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SOURCE 2 REGISTER Dk</td>
</tr>
</tbody>
</table>

#### REG-IMM1/2/4

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>IMMEDIATE DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IMMEDIATE DATA (OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IMMEDIATE DATA (OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)</td>
</tr>
</tbody>
</table>

#### REG-OPR/1/2/3

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>RESULT REGISTER Dd</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SOURCE REGISTER Dj</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>M2_SIZE (.B, .W, -.L)</td>
<td></td>
</tr>
</tbody>
</table>

**IMMEDIATE DATA**

#### REG-IMM1/2/4

- 4q mb i1 MULS.D 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

- 4q mb xb MULS.bwl Dd, Dj; see more efficient REG-REG version
- 4q mb xb MULS.bwl Dd, Dj, (opru4, xys)
- 4q mb xb MULS.bwl Dd, Dj, ([Di, xys])
- 4q mb xb MULS.bwl Dd, Dj, (opr9, xysp)
- 4q mb xb MULS.bwl Dd, Dj, (opr18, Di)
- 4q mb xb MULS.bwl Dd, Dj, (opr24, xysp)
- 4q mb xb x2 x1 MULS.bwl Dd, Dj, (opr24, xysp)
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

Operation

\[(D_j) \times (D_k) \Rightarrow D_d\]
\[(D_j) \times \text{IMM} \Rightarrow D_d\]
\[(D_j) \times (M) \Rightarrow D_d\]
\[(M1) \times (M2) \Rightarrow D_d\]

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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

- **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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>4q mb</th>
<th>4q mb</th>
<th>4q mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>RESULT REGISTER Dd</td>
<td></td>
<td>SOURCE 1 REGISTER Dj</td>
<td>SOURCE 2 REGISTER Dk</td>
</tr>
</tbody>
</table>

**REG-IMM1/2/4**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>4q mb</th>
<th>4q mb</th>
<th>4q mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>RESULT REGISTER Dd</td>
<td></td>
<td>SOURCE REGISTER Dj</td>
<td>IMMEDIATE DATA IMPLICIT IMMEDIATE (B, .W, –, .L)</td>
</tr>
</tbody>
</table>

**REG-OPR/1/2/3**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>4q mb</th>
<th>4q mb</th>
<th>4q mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>RESULT REGISTER Dd</td>
<td></td>
<td>SOURCE REGISTER Dj</td>
<td>IMMEDIATE DATA OPR POSTBYTE (for M2)</td>
</tr>
</tbody>
</table>

### Examples

**MULU**

- `Dd, Dj, Dk` - 4q mb
- `Dd, Dj, #opr8i` - 4q mb i1
- `Dd, Dj, #opr16i` - 4q mb i2 i1
- `Dd, Dj, #opr32i` - 4q mb i4 i3 i2 i1

**MULU.B, MULU.W, MULU.L**

- `Dd, Dj` - 4q mb xb
- `Dd, Dj, #opr8i`, `Dd, Dj, #opr16i`, `Dd, Dj, #opr32i` - 4q mb i1 i2 i3 i2 i1

**MULU.bwl**

- `Dd, Dj, #opru4`, `Dd, Dj, (Di, xys)` - 4q mb xb
- `Dd, Dj, (opr9, xysp)` - 4q mb xb x1
- `Dd, Dj, opr14` - 4q mb xb x2 x1
- `Dd, Dj, (opr24, Di)` - 4q mb xb x3 x2 x1
- `Dd, Dj, opr24` - 4q mb xb x3 x2 x1
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

\[ 0 - (M) = \neg(M) + 1 \Rightarrow M \]

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEG.bwl oprmemreg</td>
<td>OPR/1/2/3</td>
</tr>
</tbody>
</table>

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 Dj, 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>(\Delta)</td>
<td>(\Delta)</td>
<td>(\Delta)</td>
<td>(\Delta)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>OPR/1/2/3</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dp xb</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEG.bwl oprsxe4i ; not appropriate for destination</td>
</tr>
<tr>
<td>NEG.bwl Di</td>
</tr>
<tr>
<td>NEG.bwl (opr4, xys)</td>
</tr>
<tr>
<td>NEG.bwl ((+-xy) \mid (xy+-s) \mid (-s) \mid (s+))</td>
</tr>
<tr>
<td>NEG.bwl (Di, xys)</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>INH</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Detailed Instruction Formats**

**INH**

```
  7  6  5  4  3  2  1  0
  0  0  0  0  0  0  0  1  0  1
01
  NOP
```

Linear S12 Core Reference Manual, Rev. 1.01
**OR**

**Bitwise OR**

**Operation**

\[(D_i) \mid (M) \Rightarrow D_i\]

**Syntax Variations**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR (D_i,#opr_{immsz})</td>
<td>IMM1/2/4</td>
</tr>
<tr>
<td>OR (D_i,oprmemreg)</td>
<td>OPR/1/2/3</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>I</th>
<th>P</th>
<th>L</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Δ</td>
<td>Δ</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

\[
\begin{array}{cccccccccccc}
7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & & & & & & & & & 7p \\
\end{array}
\]

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 \(D_i,#opr_{8i}\) ; for \(D_i = 8\)-bit \(D_0\) or \(D_1\)

7p i2 i1 OR \(D_i,#opr_{16i}\) ; for \(D_i = 16\)-bit \(D_2\), \(D_3\), \(D_4\), or \(D_5\)

7p i4 i3 i2 i1 OR \(D_i,#opr_{32i}\) ; for \(D_i = 32\)-bit \(D_6\) or \(D_7\)
Instruction Fields

SD REGISTER Di - This field specifies the number of the data register Di 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 Di.

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**

**Operation**  
\[(CCL) \mid (M) \Rightarrow CCL\]

**Syntax Variations**

<table>
<thead>
<tr>
<th>ORCC</th>
<th>#opr8i</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM1</td>
<td></td>
</tr>
</tbody>
</table>

**Addressing Modes**

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>I</th>
<th>P</th>
<th>L</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>IMMEDIATE DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

DE i1  
ORCC #opr8i

---

Linear S12 Core Reference Manual, Rev. 1.01
Freescale Semiconductor 257
Operation
Push specified registers onto the stack.

for push mask oprregs2...

If Y specified: (SP) − 3 ⇒ SP; Y ⇒ M_{(SP)} : M_{(SP + 1)} : M_{(SP + 2)}
If X specified: (SP) − 3 ⇒ SP; X ⇒ M_{(SP)} : M_{(SP + 1)} : M_{(SP + 2)}
If D7 specified: (SP) − 4 ⇒ SP; D7 ⇒ M_{(SP)} : M_{(SP + 1)} : M_{(SP + 2)} : M_{(SP + 3)}
If D6 specified: (SP) − 4 ⇒ SP; D6 ⇒ M_{(SP)} : M_{(SP + 1)} : M_{(SP + 2)} : M_{(SP + 3)}
If D5 specified: (SP) − 2 ⇒ SP; D5 ⇒ M_{(SP)} : M_{(SP + 1)}
If D4 specified: (SP) − 2 ⇒ SP; D4 ⇒ M_{(SP)}
or for push mask oprregs1...

If D3 specified: (SP) − 2 ⇒ SP; D3 ⇒ M_{(SP)} : M_{(SP + 1)}
If D2 specified: (SP) − 2 ⇒ SP; D2 ⇒ M_{(SP)} : M_{(SP + 1)}
If D1 specified: (SP) − 1 ⇒ SP; D1 ⇒ M_{(SP)}
If D0 specified: (SP) − 1 ⇒ SP; D0 ⇒ M_{(SP)}
If CCL specified: (SP) − 1 ⇒ SP; CCL ⇒ M_{(SP)}
If CCH specified: (SP) − 1 ⇒ SP; CCH ⇒ M_{(SP)}

Syntax Variations

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSH oprregs1</td>
<td>INH</td>
</tr>
<tr>
<td>PSH oprregs2</td>
<td>INH</td>
</tr>
<tr>
<td>PSH ALL</td>
<td>INH</td>
</tr>
<tr>
<td>PSH ALL16b</td>
<td>INH</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Detailed Instruction Formats

INH

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

04 pb
04 pb
04 00
04 40

PSH oprregs1
PSH oprregs2
PSH ALL
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.

<table>
<thead>
<tr>
<th>MASK2/1</th>
<th>R5</th>
<th>R4</th>
<th>R3</th>
<th>R2</th>
<th>R1</th>
<th>R0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CCH</td>
<td>CCL</td>
<td>D0</td>
<td>D1</td>
<td>D2</td>
<td>D3</td>
</tr>
<tr>
<td>1</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>X</td>
<td>Y</td>
</tr>
</tbody>
</table>

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 registers 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

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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUL oprregs1</td>
<td>INH</td>
</tr>
<tr>
<td>PUL oprregs2</td>
<td>INH</td>
</tr>
<tr>
<td>PUL ALL</td>
<td>INH</td>
</tr>
<tr>
<td>PUL ALL16b</td>
<td>INH</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>MASK2/1</td>
<td>R5</td>
<td>R4</td>
<td>R3</td>
<td>R2</td>
<td>R1</td>
<td>R0</td>
</tr>
</tbody>
</table>

04 pb

04 pb

04 80

04 C0

04 pb

PUL oprregs1

PUL oprregs2

PUL ALL

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.

<table>
<thead>
<tr>
<th>MASK2/1</th>
<th>R5</th>
<th>R4</th>
<th>R3</th>
<th>R2</th>
<th>R1</th>
<th>R0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CCH</td>
<td>CCL</td>
<td>D0</td>
<td>D1</td>
<td>D2</td>
<td>D3</td>
</tr>
<tr>
<td>1</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>X</td>
<td>Y</td>
</tr>
</tbody>
</table>

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

Operation

\[(D_j) \times (D_k) \Rightarrow D_d\]
\[(D_j) \times \text{IMM} \Rightarrow D_d\]
\[(D_j) \times (M) \Rightarrow D_d\]
\[(M1) \times (M2) \Rightarrow D_d\]

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.

Syntax Variations

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>QMULS</td>
<td>(D_d, D_j, D_k)</td>
</tr>
<tr>
<td>QMULS.B</td>
<td>(D_d, D_j, #\text{opr8i})</td>
</tr>
<tr>
<td>QMULS.W</td>
<td>(D_d, D_j, #\text{opr16i})</td>
</tr>
<tr>
<td>QMULS.L</td>
<td>(D_d, D_j, #\text{opr32i})</td>
</tr>
<tr>
<td>QMULS.wplbwpl</td>
<td>(D_d, \text{oprmemreg}, \text{oprmemreg})</td>
</tr>
<tr>
<td>QMULS.bwplbwpl</td>
<td>(\text{OPR}/1/2/3-\text{OPR}/1/2/3)</td>
</tr>
</tbody>
</table>
### CCR Details

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
<td>Δ</td>
<td>0</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td>RESULT REGISTER Dd</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SOURCE 1 REGISTER Dj</td>
<td>SOURCE 2 REGISTER Dk</td>
</tr>
</tbody>
</table>

**REG-IMM1/2/4**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td>RESULT REGISTER Dd</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td>SOURCE REGISTER Dj</td>
<td>IMM_SIZE (.B, .W, –, .L)</td>
</tr>
</tbody>
</table>

**IMMEDIATE DATA**

(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)

(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE SPECIFIED IN SUFFIX)

1B Bn mb il 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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>RESULT REGISTER Dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>SOURCE REGISTER Dj</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

M2_SIZE (.B, .W, ¬, .L)

OPR POSTBYTE (for M2)

(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,xy)

1B Bn mb xb

QMULS.bwl Dd,Dj,(+-xy) | (xy+-) | (-x) | (+x)

1B Bn mb xb

QMULS.bwl Dd,Dj,(Di,xy)

1B Bn mb xb

QMULS.bwl Dd,Dj,[Di,xy]

1B Bn mb xb x1

QMULS.bwl Dd,Dj,(oprs9,xy)

1B Bn mb xb x1

QMULS.bwl Dd,Dj,[oprs9,xy]

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,(opru24,xy)

1B Bn mb xb x3 x2 x1

QMULS.bwl Dd,Dj,[opru24,xy]

1B Bn mb xb x3 x2 x1

QMULS.bwl Dd,Dj,(opru24,Di)

1B Bn mb xb x3 x2 x1

QMULS.bwl Dd,Dj,opru24

1B Bn mb xb x3 x2 x1

QMULS.bwl Dd,Dj,[opru24]
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.

<table>
<thead>
<tr>
<th>Opcode postbyte</th>
<th>M1 object code</th>
<th>M2 object code</th>
<th>Instruction Mnemonic</th>
<th>Source Format for M1 (select 1 option in this col)</th>
<th>Source Format for M2 (select 1 option in this col)</th>
</tr>
</thead>
<tbody>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td>#oprsxedi,</td>
<td>#oprsxedi1</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td>Dj,</td>
<td>Dk</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td>(opru4, xys),</td>
<td>(opru4, xys)</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td>(+-xy)</td>
<td>(xy--)</td>
<td>(-s)</td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td>[Dj, xy],</td>
<td>[Dk, xy]</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td>(oprs9, xysp),</td>
<td>(oprs9, xysp)</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td>[oprs9, xysp],</td>
<td>[oprs9, xysp]</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td>opru14,</td>
<td>opru14</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td>(opru18, Dj),</td>
<td>(opru18, Dk)</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td>opru18,</td>
<td>opru18</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td>(opru24, xysp),</td>
<td>(opru24, xysp)</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td>[opru24, xysp],</td>
<td>[opru24, xysp]</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td>(opru24, Dj),</td>
<td>(opru24, Dk)</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td>opru24,</td>
<td>opru24</td>
<td></td>
</tr>
<tr>
<td>xb</td>
<td>xb</td>
<td></td>
<td>[opru24],</td>
<td>[opru24]</td>
<td></td>
</tr>
</tbody>
</table>
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

Operation

(Dj) × (Dk) ⇒ Dd
(Dj) × IMM ⇒ Dd
(Dj) × (M) ⇒ Dd
(M1) × (M2) ⇒ Dd

Description

Multiplies two unsigned operands and stores the unsigned result to register Dd. The first source operand may be a register Dj 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 Dk, 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.

QMULU

Syntax Variations

| 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.bwpibwppl | Dd,oprmemreg,oprmemreg | OPR/1/2/3-OPR/1/2/3 |
### Detailed Instruction Formats

#### REG-REG

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>RESULT REGISTER Dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>SOURCE 1 REGISTER Dj</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>SOURCE 2 REGISTER Dk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>RESULT REGISTER Dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>SOURCE REGISTER Dj</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>IMM_SIZE (.B, .W, .L)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### REG-IMM1/2/4

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>RESULT REGISTER Dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>SOURCE REGISTER Dj</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>IMM_SIZE (.B, .W, .L)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### REG-OPR/1/2/3

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>RESULT REGISTER Dd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>SOURCE REGISTER Dj</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>M2_SIZE (.B, .W, .L)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### CCR Details

<table>
<thead>
<tr>
<th>U</th>
<th>I</th>
<th>P</th>
<th>L</th>
<th>S</th>
<th>X</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **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.
<table>
<thead>
<tr>
<th>Instruction Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1B Bn mb xb x1</td>
<td>QMULU.bw</td>
</tr>
<tr>
<td></td>
<td>Dd,Dj,opru14</td>
</tr>
<tr>
<td>1B Bn mb xb x2 x1</td>
<td>QMULU.bw</td>
</tr>
<tr>
<td></td>
<td>Dd,Dj,(opru18,Di)</td>
</tr>
<tr>
<td>1B Bn mb xb x2 x1</td>
<td>QMULU.bw</td>
</tr>
<tr>
<td></td>
<td>Dd,Dj,opru18</td>
</tr>
<tr>
<td>1B Bn mb xb x3 x2 x1</td>
<td>QMULU.bw</td>
</tr>
<tr>
<td></td>
<td>Dd,Dj,(opr24,xyse)</td>
</tr>
<tr>
<td>1B Bn mb xb x3 x2 x1</td>
<td>QMULU.bw</td>
</tr>
<tr>
<td></td>
<td>Dd,Dj,[opr24,xyse]</td>
</tr>
<tr>
<td>1B Bn mb xb x3 x2 x1</td>
<td>QMULU.bw</td>
</tr>
<tr>
<td></td>
<td>Dd,Dj,opr24</td>
</tr>
<tr>
<td>1B Bn mb xb x3 x2 x1</td>
<td>QMULU.bw</td>
</tr>
<tr>
<td></td>
<td>Dd,Dj,[opr24]</td>
</tr>
</tbody>
</table>
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.
Rotation 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>L/R=1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>SIZE (.B, .W, .P, .L)</td>
<td></td>
</tr>
</tbody>
</table>

OPR POSTBYTE (specifies source operand to be rotated)

(Optional Address-Byte Depending on Address-Mode)

(Optional Address-Byte Depending on Address-Mode)

1n sb xb

ln sb xb  ROL.bpl  #oprxed4i ;not appropriate for destination
ln sb xb  ROL.bpl  Di
ln sb xb  ROL.bpl  (opru4, xys)
ln sb xb  ROL.bpl  (+-xy) | (xy++) | (-s) | (s+)
ln sb xb  ROL.bpl  (Di, xys)
ln sb xb  ROL.bpl  [Di, xy]
ln sb xb  x1  ROL.bpl  (oprs9, xysp)
ln sb xb  x1  ROL.bpl  [oprs9, xysp]
ln sb xb  x1  ROL.bpl  opru14
ln sb xb  x2  x1  ROL.bpl  (opru18, Di)
ln sb xb  x2  x1  ROL.bpl  opru18
ln sb xb  x3  x2  x1  ROL.bpl  (opru24, xysp)
ln sb xb  x3  x2  x1  ROL.bpl  [opru24, xysp]
ln sb xb  x3  x2  x1  ROL.bpl  (opru24, Di)
ln sb xb  x3  x2  x1  ROL.bpl  opr24
ln sb xb  x3  x2  x1  ROL.bpl  [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
ROR.bwp1 oprmemreg

Addressing Modes
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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
<td>0</td>
<td>Δ</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>L/R=0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OPR POSTBYTE (specifies source operand to be rotated)

(Optional ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

(Optional ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)

| In sb xb | ROR.bpl | #oprxe4i ;not appropriate for destination |
| In sb xb | ROR.bpl | Di |
| In sb xb | ROR.bpl | (oprux,yys) |
| In sb xb | ROR.bpl | ((+-xy) | (xy+-) | (-s) | (s+)) |
| In sb xb | ROR.bpl | (Di,yys) |
| In sb xb | ROR.bpl | [Di,xy] |
| In sb xb x1 | ROR.bpl | (pr9s,yxsp) |
| In sb xb x1 | ROR.bpl | [pr9s,yxsp] |
| In sb xb x1 | ROR.bpl | opru14 |
| In sb xb x2 x1 | ROR.bpl | (opr18,Di) |
| In sb xb x2 x1 | ROR.bpl | opru18 |
| In sb xb x3 x2 x1 | ROR.bpl | (opr24,yxsp) |
| In sb xb x3 x2 x1 | ROR.bpl | [opr24,yxsp] |
| In sb xb x3 x2 x1 | ROR.bpl | (opr24,Di) |
| In sb xb x3 x2 x1 | ROR.bpl | opr24 |
| In sb xb x3 x2 x1 | ROR.bpl | [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

**Operation**

\[
\begin{align*}
M_{(SP)} &: M_{(SP + 1)} \Rightarrow \text{CCH:CCL}; (SP) + 2 \Rightarrow SP \\
M_{(SP)} &: D0; (SP) + 1 \Rightarrow SP \\
M_{(SP)} &: D1; (SP) + 1 \Rightarrow SP \\
M_{(SP)} &: M_{(SP + 1)} \Rightarrow D2; (SP) + 2 \Rightarrow SP \\
M_{(SP)} &: M_{(SP + 1)} \Rightarrow D3; (SP) + 2 \Rightarrow SP \\
M_{(SP)} &: M_{(SP + 1)} \Rightarrow D4; (SP) + 2 \Rightarrow SP \\
M_{(SP)} &: M_{(SP + 1)} \Rightarrow D5; (SP) + 2 \Rightarrow SP \\
M_{(SP)} &: M_{(SP + 1)}; M_{(SP + 2)}; M_{(SP + 3)} \Rightarrow D6; (SP) + 4 \Rightarrow SP \\
M_{(SP)} &: M_{(SP + 1)}; M_{(SP + 2)}; M_{(SP + 3)} \Rightarrow D7; (SP) + 4 \Rightarrow SP \\
M_{(SP)} &: M_{(SP + 1)}; M_{(SP + 2)} \Rightarrow X; (SP) + 3 \Rightarrow SP \\
M_{(SP)} &: M_{(SP + 1)}; M_{(SP + 2)} \Rightarrow Y; (SP) + 3 \Rightarrow SP \\
M_{(SP)} &: M_{(SP + 1)}; M_{(SP + 2)} \Rightarrow PC; (SP) + 3 \Rightarrow SP
\end{align*}
\]

**Syntax Variations**

<table>
<thead>
<tr>
<th>RTI</th>
<th>INH</th>
</tr>
</thead>
</table>

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>⇓</td>
<td>Δ</td>
<td>Δ</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
<td>Δ</td>
<td>Δ</td>
<td>Δ</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
</tr>
</tbody>
</table>

1B 90  

**Linear S12 Core Reference Manual, Rev. 1.01**

Freescale Semiconductor
**RTS**

**Return from Subroutine**

**Operation**

\[ M_{(SP)} : M_{(SP + 1)} : M_{(SP + 2)} \implies PC; (SP) + 3 \implies SP \]

**Syntax Variations**

| RTS |

**Addressing Modes**

| 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**

<table>
<thead>
<tr>
<th>U - - - -</th>
<th>IPL</th>
<th>S X - I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Detailed Instruction Format**

```
0 0 0 0 0 1 0 1 05
```

05 RTS
SAT

Saturate

Operation
saturated(Di) ⇒ Di

Syntax Variations

<table>
<thead>
<tr>
<th>SAT</th>
<th>Di</th>
</tr>
</thead>
</table>

Addressing Modes

INH

Description
Replace the content of Di with its saturated value. The operand is treated as a signed value. Operation size depends on (matches) the size of Di.

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 Di is replaced by a value representing the positive or the negative boundary of the signed value range defined by the size of Di.

If the overflow (V-) flag is set, the content of Di 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 Di 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

N: Set according to the MSB of the result.
Z: Set if the result is zero. Cleared otherwise.
V: Cleared.

Detailed Instruction Formats

INH

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>An</td>
</tr>
</tbody>
</table>

1B An SAT Di

Instruction Fields

SD REGISTER Di - This field specifies the number of the data register Di 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).

Linear S12 Core Reference Manual, Rev. 1.01

278 Freescale Semiconductor
SBC  Subtract with Borrow

Operation

\[(D_i) - (M) - C \Rightarrow D_i\]

Syntax Variations

<table>
<thead>
<tr>
<th>SBC</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBC</td>
<td>IMM1/2/4</td>
</tr>
<tr>
<td>SBC</td>
<td>OPR/1/2/3</td>
</tr>
</tbody>
</table>

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 (\(-1, 1, 2, 3, \ldots, 14, 15\)), 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>I</th>
<th>P</th>
<th>L</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Δ</td>
<td>Δ</td>
<td>Δ</td>
<td>Δ</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>SD REGISTER (D_i)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 \(D7\)
## Instruction Fields

**SD REGISTER Di** - This field specifies the number of the data register Di 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 Di.

**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-indirect memory location.

### OPRI/2/3

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**SD REGISTER Di**

### OPR POSTBYTE

- **(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,#opr#e4i ;-1, +1, 2, 3...14, 15
8n xb SBC Di,Dj
8n xb SBC Di,(opr4,ys)
8n xb SBC Di,((+-xy) | (xy+-) | (-s) | (s+))
8n xb SBC Di,(Dj,ys)
8n xb SBC Di,[Dj,xy]
8n xb x1 SBC Di,(opr9,ys)
8n xb x1 SBC Di,[opr9,ys]
8n xb x1 SBC Di,opr14
8n xb x2 x1 SBC Di,(opr18,Dj)
8n xb x2 x1 SBC Di,opr18
8n xb x3 x2 x1 SBC Di,(opr24,ys)
8n xb x3 x2 x1 SBC Di,[opr24,ys]
8n xb x3 x2 x1 SBC Di,(opr24,Dj)
8n xb x3 x2 x1 SBC Di,opr24
8n xb x3 x2 x1 SBC Di,[opr24]
SEC  
Set Carry Flag  
(Translates to ORCC #$01)

Operation
1 ⇒ 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

C: Set.

Detailed Instruction Formats

IMM1

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

DE 01  SEV
SEI
Set Interrupt Mask
(Translates to ORCC #$10)

Operation
1 ⇒ I bit

Syntax Variations

| 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

supervisor state
user state

Detailed Instruction Formats

IMM1

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

DE 10
SEI
SEV

Set Overflow Flag
(Translates to ORCC #$02)

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

\[
\begin{array}{cccccccccccc}
\hline
\end{array}
\]

V: Set.

Detailed Instruction Formats

IMM1

\[
\begin{array}{cccccccccccc}
7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\hline
1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]

DE 02 SEV
SEX
Sign-Extend
(smaller CPU register to a larger CPU register)

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
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

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEX cpureg, cpureg</td>
<td>INH</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INH</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td>IPL</td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td>S</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td>I</td>
</tr>
<tr>
<td>N</td>
</tr>
<tr>
<td>Z</td>
</tr>
<tr>
<td>V</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>–</td>
</tr>
<tr>
<td>–</td>
</tr>
<tr>
<td>–</td>
</tr>
<tr>
<td>–</td>
</tr>
<tr>
<td>–</td>
</tr>
<tr>
<td>–</td>
</tr>
<tr>
<td>–</td>
</tr>
<tr>
<td>–</td>
</tr>
<tr>
<td>–</td>
</tr>
<tr>
<td>–</td>
</tr>
<tr>
<td>–</td>
</tr>
</tbody>
</table>

FIRST (SOURCE) REGISTER
SECOND (DESTINATION) REGISTER

Linear S12 Core Reference Manual, Rev. 1.01
284 Freescale Semiconductor
### Table 6-2: Postbyte (eb) Coding for Sign-Extend Operations

<table>
<thead>
<tr>
<th>source</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>D5</th>
<th>D0</th>
<th>D1</th>
<th>D6</th>
<th>D7</th>
<th>X</th>
<th>Y</th>
<th>S</th>
<th>CCH</th>
<th>CCL</th>
<th>CCW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td>-0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D3</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D4</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D5</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D0</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D6</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D7</td>
<td>-7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>-9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>-A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reserved</td>
<td>-B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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).
Chapter 6 Instruction Glossary

**SPARE**  
*Unimplemented Page1 Opcode 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 1 TRAP Vector) ⇒ PC

**Syntax Variations**

| not a user instruction | - |

**Addressing Modes**

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

U: Cleared.

I: Set.

**Detailed Instruction Format**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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.
Store
(Di, X, Y, or SP)

Operation
(Di) ⇒ M
(X) ⇒ M
(Y) ⇒ M
(SP) ⇒ M

Description
Store a register Di, 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 Di, 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 Di, 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

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 (Di)

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Δ</td>
<td>Δ</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

Detailed Instruction Formats

EXT (Di)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>REGISTER</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ADDRESS[23:16]
ADDRESS[15:8]
ADDRESS[7:0]

Dn a3 a2 a1 ST Di, opr24a
### OPR/1/2/3 (Di)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>REGISTER</th>
<th>Cn xb</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**OPR POSTBYTE**

*(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)*

*(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)*

*(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)*

Cn xb  
ST  
\[Di, \#\text{opr}xse4i; \text{not appropriate for a destination}\]

Cn xb  
ST  
\[Di, Dj\]

Cn xb  
ST  
\[Di, (\text{opr}u4, xys)\]

Cn xb  
ST  
\[Di, ((+xy) | (xy+-) | (-s) | (s+))\]

Cn xb  
ST  
\[Di, (Dj, xys)\]

Cn xb  
ST  
\[Di, [Dj, xy]\]

Cn xb  
ST  
\[Di, (\text{opr}u9, xysp)\]

Cn xb  
ST  
\[Di, [\text{opr}9, xysp]\]

Cn xb  
ST  
\[Di, \text{opr}u14\]

Cn xb  
ST  
\[Di, (\text{opr}u18, Dj)\]

Cn xb  
ST  
\[Di, \text{opr}u18\]

Cn xb  
ST  
\[Di, (\text{opr}u24, xysp)\]

Cn xb  
ST  
\[Di, [\text{opr}u24, xysp]\]

Cn xb  
ST  
\[Di, (\text{opr}u24, Dj)\]

Cn xb  
ST  
\[Di, \text{opr}24\]

### EXT (X or Y)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Y/X</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**ADDRESS[23:16]**

**ADDRESS[15:8]**

**ADDRESS[7:0]**

Dp a3 a2 a1  
ST  
xy, opr24a

### OPR/1/2/3 (X or Y)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Y/X</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**OPR POSTBYTE**

*(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)*

*(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)*

*(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)*

Cp xb  
ST  
\[xy, \#\text{opr}xse4i; \text{not appropriate for a destination}\]

Cp xb  
ST  
\[xy, Dj\]

Cp xb  
ST  
\[xy, (\text{opr}u4, xys)\]

Cp xb  
ST  
\[xy, ((+xy) | (xy+-) | (-s) | (s+))\]

Cp xb  
ST  
\[xy, (Dj, xys)\]

Cp xb  
ST  
\[xy, [Dj, xy]\]

Cp xb  
ST  
\[xy, (\text{opr}u9, xysp)\]

Cp xb  
ST  
\[xy, [\text{opr}9, xysp]\]

Cp xb  
ST  
\[xy, \text{opr}u14\]

Cp xb  
ST  
\[xy, (\text{opr}u18, Dj)\]

Cp xb  
ST  
\[xy, \text{opr}u18\]

Cp xb  
ST  
\[xy, (\text{opr}u24, xysp)\]

Cp xb  
ST  
\[xy, [\text{opr}u24, xysp]\]
Chapter 6 Instruction Glossary

<table>
<thead>
<tr>
<th>OPR/1/2/3 (SP)</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**OPR POSTBYTE**

- (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,#opr2xe4i ;not appropriate for a destination
1B 01 xb ST S,Dj ;suggest using more efficient TFR
1B 01 xb ST S,(opr4,xy)
1B 01 xb ST S,((-xy) | (xy|(-s) | (s))
1B 01 xb ST S,(Dj,xy)
1B 01 xb ST S,[Dj,xy]
1B 01 xb x1 ST S,(opr9,xy)
1B 01 xb x1 ST S, opr9,xy)
1B 01 xb x1 ST S, opr14
1B 01 xb x2 x1 ST S,(opr18, Dj)
1B 01 xb x2 x1 ST S, opr18
1B 01 xb x3 x2 x1 ST S,(opr24,xy)
1B 01 xb x3 x2 x1 ST S, [opr24,xy]
1B 01 xb x3 x2 x1 ST S,(opr24, 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 Di 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=D6, 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)

Operation
If S bit = 1, treat STOP as a NOP; else if S = 0;
(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; CCW ⇒ 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 \textbf{RESET}, \textbf{XIRQ}, or \textbf{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

\begin{center}
\begin{tabular}{cccccccccccc}
\textbf{U} & - & - & - & - & \textbf{IPL} & \textbf{S} & \textbf{X} & - & \textbf{I} & \textbf{N} & \textbf{Z} & \textbf{V} & \textbf{C} \\
\end{tabular}
\end{center}

Detailed Instruction Format

\begin{center}
\begin{tabular}{cccccccccc}
7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & 1B \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 05 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
1B & 05 & STOP
\end{tabular}
\end{center}
**SUB** Subtract without Borrow

**Operation**

\[
(D_i) - (M) \implies D_i \\
(X) - (Y) \implies D_6 \\
(Y) - (X) \implies D_6
\]

**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 \(D_6\). 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 \(D_6,X,Y\) or SUB \(D_6,Y,X\) source operands \(X\) and \(Y\) are treated as unsigned and the result is a signed long int.

**CCR Details**

\[
\begin{array}{cccccccccccc}
  \hline
  - & - & - & - & - & - & - & - & - & \Delta & \Delta & \Delta & \Delta & \Delta
\end{array}
\]

- **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**

\[
\begin{array}{cccccccccc}
  7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
  \hline
  1 & 1 & 1 & 1 & 1 & 1 & 0 & 1
\end{array}
\]

FD

\[
\begin{array}{cccccccccc}
  7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
  \hline
  1 & 1 & 1 & 1 & 1 & 1 & 1 & 0
\end{array}
\]

**FD**

**SUB**

\[
\begin{array}{cccccccccc}
  7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
  \hline
  1 & 1 & 1 & 1 & 1 & 1 & 1 & 0
\end{array}
\]

**FE**

**SUB**

\[
\begin{array}{cccccccccc}
  7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
  \hline
  1 & 1 & 1 & 1 & 1 & 1 & 1 & 0
\end{array}
\]

**Linear S12 Core Reference Manual, Rev. 1.01**

Freescale Semiconductor
### Instruction Fields

**SD REGISTER Di** - This field specifies the number of the data register Di 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 Di.

**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.

### Table: Immediate Data

<table>
<thead>
<tr>
<th>SD REGISTER Di</th>
<th>IMMEDIATE DATA</th>
<th>(OPTIONAL IMMEDIATE DATA DEPENDING ON SIZE OF Di)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7p i1</td>
<td>SUB Di,#opr8i ; for Di = 8-bit D0 or D1</td>
<td></td>
</tr>
<tr>
<td>7p i2 i1</td>
<td>SUB Di,#opr16i ; for Di = 16-bit D2, D3, D4, or D5</td>
<td></td>
</tr>
<tr>
<td>7p i4 i3 i2 i1</td>
<td>SUB Di,#opr32i ; for Di = 32-bit D6 or D7</td>
<td></td>
</tr>
</tbody>
</table>

### Table: OPR/1/2/3

<table>
<thead>
<tr>
<th>SD REGISTER Di</th>
<th>OPR POSTBYTE</th>
<th>(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)</th>
<th>(OPTIONAL ADDRESS-BYTE DEPENDING ON ADDRESS-MODE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8n xb</td>
<td>SUB Di,#oprsxe4i ; -1, +1, 2, 3...14, 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8n xb</td>
<td>SUB Di,Dj</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8n xb</td>
<td>SUB Di,(opru4,sys)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8n xb</td>
<td>SUB Di,((+-xy)</td>
<td>(xy+))</td>
<td>(-s)</td>
</tr>
<tr>
<td>8n xb</td>
<td>SUB Di,(Dj,sys)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8n xb</td>
<td>SUB Di,[Dj,xy]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8n xb x1</td>
<td>SUB Di,(oprs9,sys)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8n xb x1</td>
<td>SUB Di,[oprs9,sys]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8n xb x1</td>
<td>SUB Di,opru14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8n xb x2 x1</td>
<td>SUB Di,(opru18,Dj)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8n xb x2 x1</td>
<td>SUB Di,opru18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8n xb x3 x2 x1</td>
<td>SUB Di,(opru24,sys)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8n xb x3 x2 x1</td>
<td>SUB Di,[opru24,sys]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8n xb x3 x2 x1</td>
<td>SUB Di,(opru24,Dj)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8n xb x3 x2 x1</td>
<td>SUB Di,opru24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8n xb x3 x2 x1</td>
<td>SUB Di,[opru24]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SWI

Software Interrupt

Operation

\( (SP) - 3 \Rightarrow SP; \text{RTN}[23:0] \Rightarrow M_{(SP)} : M_{(SP + 1)} : M_{(SP + 2)} \)

\( (SP) - 3 \Rightarrow SP; Y \Rightarrow M_{(SP)} : M_{(SP + 1)} : M_{(SP + 2)} \)

\( (SP) - 3 \Rightarrow SP; X \Rightarrow M_{(SP)} : M_{(SP + 1)} : M_{(SP + 2)} \)

\( (SP) - 4 \Rightarrow SP; D7 \Rightarrow M_{(SP)} : M_{(SP + 1)} : M_{(SP + 2)} : M_{(SP + 3)} \)

\( (SP) - 4 \Rightarrow SP; D6 \Rightarrow M_{(SP)} : M_{(SP + 1)} : M_{(SP + 2)} : M_{(SP + 3)} \)

\( (SP) - 2 \Rightarrow SP; D5 \Rightarrow M_{(SP)} : M_{(SP + 1)} \)

\( (SP) - 2 \Rightarrow SP; D4 \Rightarrow M_{(SP)} : M_{(SP + 1)} \)

\( (SP) - 2 \Rightarrow SP; D3 \Rightarrow M_{(SP)} : M_{(SP + 1)} \)

\( (SP) - 2 \Rightarrow SP; D2 \Rightarrow M_{(SP)} : M_{(SP + 1)} \)

\( (SP) - 1 \Rightarrow SP; D1 \Rightarrow M_{(SP)} \)

\( (SP) - 1 \Rightarrow SP; D0 \Rightarrow M_{(SP)} \)

\( (SP) - 2 \Rightarrow SP; \text{CCH:CCL} \Rightarrow M_{(SP)} : M_{(SP + 1)} \)

\( 0 \Rightarrow U; 1 \Rightarrow I; \text{(SWI vector)} \Rightarrow PC \)

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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

U: Cleared.
I: Set.

Detailed Instruction Format

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>FF</th>
</tr>
</thead>
</table>

Linear S12 Core Reference Manual, Rev. 1.01
Freescale Semiconductor
**SYS**

**System Call Software Interrupt**

**Operation**

\[
\begin{align*}
\text{(SP)} - 3 & \Rightarrow \text{SP}; \text{RTN}[23:0] \Rightarrow M_{\text{SP}} : M_{\text{SP} + 1} : M_{\text{SP} + 2} \\
\text{(SP)} - 3 & \Rightarrow \text{SP}; Y \Rightarrow M_{\text{SP}} : M_{\text{SP} + 1} : M_{\text{SP} + 2} \\
\text{(SP)} - 3 & \Rightarrow \text{SP}; X \Rightarrow M_{\text{SP}} : M_{\text{SP} + 1} : M_{\text{SP} + 2} \\
\text{(SP)} - 4 & \Rightarrow \text{SP}; D7 \Rightarrow M_{\text{SP}} : M_{\text{SP} + 1} : M_{\text{SP} + 2} : M_{\text{SP} + 3} \\
\text{(SP)} - 4 & \Rightarrow \text{SP}; D6 \Rightarrow M_{\text{SP}} : M_{\text{SP} + 1} : M_{\text{SP} + 2} : M_{\text{SP} + 3} \\
\text{(SP)} - 2 & \Rightarrow \text{SP}; D5 \Rightarrow M_{\text{SP}} : M_{\text{SP} + 1} \\
\text{(SP)} - 2 & \Rightarrow \text{SP}; D4 \Rightarrow M_{\text{SP}} : M_{\text{SP} + 1} \\
\text{(SP)} - 2 & \Rightarrow \text{SP}; D3 \Rightarrow M_{\text{SP}} : M_{\text{SP} + 1} \\
\text{(SP)} - 2 & \Rightarrow \text{SP}; D2 \Rightarrow M_{\text{SP}} : M_{\text{SP} + 1} \\
\text{(SP)} - 1 & \Rightarrow \text{SP}; D1 \Rightarrow M_{\text{SP}} \\
\text{(SP)} - 1 & \Rightarrow \text{SP}; D0 \Rightarrow M_{\text{SP}} \\
\text{(SP)} - 2 & \Rightarrow \text{SP}; \text{CCH:CCL} \Rightarrow M_{\text{SP}} : M_{\text{SP} + 1} \\
0 & \Rightarrow \text{U}; 1 \Rightarrow \text{I}; (\text{SYS Vector}) \Rightarrow \text{PC}
\end{align*}
\]

**Syntax Variations**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS</td>
<td>INH</td>
</tr>
</tbody>
</table>

**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**

\[
\begin{array}{cccccccc}
\text{IPL} & S & X & I & N & Z & V & C \\
\hline
0 & - & - & - & - & - & - & - \\
\end{array}
\]

U: Cleared.

I: Set.

**Detailed Instruction Format**

\[
\begin{array}{cccccccccc}
7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\hline
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
\end{array}
\]

\begin{align*}
\text{1B} & \quad \text{SYS} \\
\text{07} & \quad \text{SYS}
\end{align*}

Linear S12 Core Reference Manual, Rev. 1.01

Freescale Semiconductor
**TBcc** Test and Branch

**Operation**

(Di) \( \rightarrow \) \( Di; \) then Branch if (condition) true

(X) \( \rightarrow \) \( X; \) then Branch if (condition) true

(Y) \( \rightarrow \) \( Y; \) then Branch if (condition) true

(M) \( \rightarrow \) \( Y; \) 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**

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBcc ( Di, oprdest )</td>
<td>REG-REL</td>
</tr>
<tr>
<td>TBcc ( X, oprdest )</td>
<td>REG-REL</td>
</tr>
<tr>
<td>TBcc ( Y, oprdest )</td>
<td>REG-REL</td>
</tr>
<tr>
<td>TBcc.bwploprmemreg, oprdest</td>
<td>OPR/1/2/3-REL</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Detailed Instruction Formats**

**REG-REL (Di)**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REL_SIZE</th>
<th>7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)</td>
<td></td>
</tr>
</tbody>
</table>

0B \( lb \) \( rb \)

TBcc \( Di, oprdest \); destination within -64...+63 (7-bit)

0B \( lb \) \( rb \) \( rl \)

TBcc \( Di, oprdest \); destination within +/-16k (15-bit)
REG-REL (X, Y)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>CC (NE,EQ,PL,MI,GT,LE,=,-)</td>
<td>1</td>
<td>0</td>
<td>don't care</td>
<td>Y:X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

REPLACE SIZE 7 bit DISPLACEMENT (REL_SIZE==0) or high-order 7 bits of 15 bit DISPLACEMENT (REL_SIZE==1)

Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)

OPR/1/2/3-REL

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>CC (NE,EQ,PL,MI,GT,LE,=,-)</td>
<td>1</td>
<td>1</td>
<td>SIZE (.B, .W, .P, .L)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OPR POSTBYTE

(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)

Optional low-order 8 bits of 15-bit DISPLACEMENT (REL_SIZE==1)
Instruction Fields

CC - This field specifies the condition for the branch according to the table below:

<table>
<thead>
<tr>
<th>Test</th>
<th>Mnemonic</th>
<th>Condition</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>NE; r≠0</td>
<td>TBNE</td>
<td>000</td>
<td>Z = 0</td>
</tr>
<tr>
<td>EQ; r=0</td>
<td>TBEQ</td>
<td>001</td>
<td>Z = 1</td>
</tr>
<tr>
<td>PL; r≥0</td>
<td>TBPL</td>
<td>010</td>
<td>N = 0</td>
</tr>
<tr>
<td>MI; r&lt;0</td>
<td>TBMI</td>
<td>011</td>
<td>N = 1</td>
</tr>
<tr>
<td>GT; r&gt;0</td>
<td>TBGT</td>
<td>100</td>
<td>Z</td>
</tr>
<tr>
<td>LE; r≤0</td>
<td>TBLE</td>
<td>101</td>
<td>Z</td>
</tr>
<tr>
<td>reserved (Test and Branch Never)</td>
<td>110</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>111</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Syntax Variations</th>
<th>Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFR cpureg, cpureg</td>
<td>INH</td>
</tr>
</tbody>
</table>

**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**

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
```

FIRST (SOURCE) REGISTER | SECOND (DESTINATION) REGISTER

9E tb TFR cpureg, cpureg
### Table 6-3. Transfer Postbyte (tb) Coding Map

<table>
<thead>
<tr>
<th>Source</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D3</td>
<td>D3</td>
<td>D2</td>
</tr>
<tr>
<td>D3</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D3</td>
<td>D3</td>
<td>D2</td>
</tr>
<tr>
<td>D4</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D3</td>
<td>D3</td>
<td>D2</td>
</tr>
<tr>
<td>D5</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D3</td>
<td>D3</td>
<td>D2</td>
</tr>
<tr>
<td>D6</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D3</td>
<td>D3</td>
<td>D2</td>
</tr>
<tr>
<td>D7</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D3</td>
<td>D3</td>
<td>D2</td>
</tr>
</tbody>
</table>

**Note:** The table continues with more entries, indicating various combinations of source and destination bits, with corresponding actions for X, Y, S, and CCH, CCL, CCW codes.
**TRAP**

**Unimplemented Page2 Opcode Trap**

**Operation**

\[(SP) - 3 \Rightarrow SP; \text{RTN}[23:0] \Rightarrow M_{(SP)} : M_{(SP + 1)} : M_{(SP + 2)}\]

\[(SP) - 3 \Rightarrow SP; Y \Rightarrow M_{(SP)} : M_{(SP + 1)} : M_{(SP + 2)}\]

\[(SP) - 3 \Rightarrow SP; X \Rightarrow M_{(SP)} : M_{(SP + 1)} : M_{(SP + 2)}\]

\[(SP) - 4 \Rightarrow SP; D7 \Rightarrow M_{(SP)} : M_{(SP + 1)} : M_{(SP + 2)} : M_{(SP + 3)}\]

\[(SP) - 4 \Rightarrow SP; D6 \Rightarrow M_{(SP)} : M_{(SP + 1)} : M_{(SP + 2)} : M_{(SP + 3)}\]

\[(SP) - 2 \Rightarrow SP; D5 \Rightarrow M_{(SP)} : M_{(SP + 1)}\]

\[(SP) - 2 \Rightarrow SP; D4 \Rightarrow M_{(SP)} : M_{(SP + 1)}\]

\[(SP) - 2 \Rightarrow SP; D3 \Rightarrow M_{(SP)} : M_{(SP + 1)}\]

\[(SP) - 2 \Rightarrow SP; D2 \Rightarrow M_{(SP)} : M_{(SP + 1)}\]

\[(SP) - 1 \Rightarrow SP; D1 \Rightarrow M_{(SP)}\]

\[(SP) - 1 \Rightarrow SP; D0 \Rightarrow M_{(SP)}\]

\[(SP) - 2 \Rightarrow SP; \text{CCH}:\text{CCL} \Rightarrow M_{(SP)} : M_{(SP + 1)}\]

\[0 \Rightarrow U; 1 \Rightarrow I; \text{(Page 2 TRAP Vector)} \Rightarrow PC\]

**Syntax Variations**

```
<table>
<thead>
<tr>
<th>TRAP</th>
<th>#trapnum</th>
<th>INH</th>
</tr>
</thead>
</table>
```

**Addressing Modes**

**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: Cleared.
I: Set.
```

**Detailed Instruction Format**

```
<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Refer to the opcode map to identify unimplemented page 2 opcodes.
WAI

Wait for Interrupt

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)}

Stop CPU clock and wait for an interrupt

Syntax Variations

<table>
<thead>
<tr>
<th>WAI</th>
<th>INH</th>
</tr>
</thead>
</table>

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 RESET, XIRQ, or 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

<table>
<thead>
<tr>
<th>U</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IPL</th>
<th>S</th>
<th>X</th>
<th>-</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Detailed Instruction Format

<table>
<thead>
<tr>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Stop CPU clock and wait for an interrupt
Zero-Extend
(smaller CPU register to a larger CPU register)

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

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

```
<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

FIRST (SOURCE) REGISTER  SECOND (DESTINATION) REGISTER

9E tb ZEX cpureg, cpureg
<table>
<thead>
<tr>
<th>source</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>D5</th>
<th>D0</th>
<th>D1</th>
<th>D6</th>
<th>D7</th>
<th>X</th>
<th>Y</th>
<th>S</th>
<th>CCH</th>
<th>CCL</th>
<th>CCW</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination</td>
<td>0-</td>
<td>1-</td>
<td>2-</td>
<td>3-</td>
<td>4-</td>
<td>5-</td>
<td>6-</td>
<td>7-</td>
<td>8-</td>
<td>9-</td>
<td>A-</td>
<td>B-</td>
<td>C-</td>
<td>D-</td>
</tr>
<tr>
<td>D2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>00:D0</td>
<td>00:D1</td>
<td>D6L</td>
<td>D7L</td>
<td>XL</td>
<td>YL</td>
<td>SL</td>
<td>00:CCH</td>
<td>00:CCL</td>
<td>CCW</td>
</tr>
<tr>
<td>D3</td>
<td>-1</td>
<td>D2</td>
<td>-</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>00:D0</td>
<td>00:D1</td>
<td>D6L</td>
<td>D7L</td>
<td>XL</td>
<td>YL</td>
<td>SL</td>
<td>00:CCH</td>
</tr>
<tr>
<td>D4</td>
<td>-2</td>
<td>D2</td>
<td>-</td>
<td>D4</td>
<td>D5</td>
<td>-</td>
<td>D5</td>
<td>00:D0</td>
<td>00:D1</td>
<td>D6L</td>
<td>D7L</td>
<td>XL</td>
<td>YL</td>
<td>SL</td>
</tr>
<tr>
<td>D5</td>
<td>-3</td>
<td>D2</td>
<td>-</td>
<td>D5</td>
<td>D6</td>
<td>D7L</td>
<td>XL</td>
<td>YL</td>
<td>SL</td>
<td>00:CCH</td>
<td>00:CCL</td>
<td>CCW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D6</td>
<td>-4</td>
<td>D2</td>
<td>-</td>
<td>D6</td>
<td>D7</td>
<td>XL</td>
<td>YL</td>
<td>SL</td>
<td>00:CCH</td>
<td>00:CCL</td>
<td>CCW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D7</td>
<td>-5</td>
<td>D2</td>
<td>-</td>
<td>D7</td>
<td>XL</td>
<td>YL</td>
<td>SL</td>
<td>00:CCH</td>
<td>00:CCL</td>
<td>CCW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>-6</td>
<td>D2</td>
<td>-</td>
<td>D8</td>
<td>Y</td>
<td>S</td>
<td>00:CCH</td>
<td>00:CCL</td>
<td>CCW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>-7</td>
<td>D2</td>
<td>-</td>
<td>D9</td>
<td>S</td>
<td>S</td>
<td>00:CCH</td>
<td>00:CCL</td>
<td>CCW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.4: Transfer Postbyte (tb) Coding Map

Chapter 6 Instruction Glossary
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.
   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 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 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 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 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 (XIRQ)

The XIRQ function is disabled after system reset and upon entering the interrupt service routine for an 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 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>
<thead>
<tr>
<th>Memory Location</th>
<th>CPU12 Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP + 26</td>
<td>Return Address</td>
</tr>
<tr>
<td>SP + 23</td>
<td>Y</td>
</tr>
<tr>
<td>SP + 20</td>
<td>X</td>
</tr>
<tr>
<td>SP + 16</td>
<td>D7</td>
</tr>
<tr>
<td>SP + 12</td>
<td>D6</td>
</tr>
<tr>
<td>SP + 10</td>
<td>D5</td>
</tr>
<tr>
<td>SP + 8</td>
<td>D4</td>
</tr>
<tr>
<td>SP + 6</td>
<td>D3</td>
</tr>
<tr>
<td>SP + 4</td>
<td>D2</td>
</tr>
<tr>
<td>SP + 3</td>
<td>D1</td>
</tr>
<tr>
<td>SP + 2</td>
<td>D0</td>
</tr>
<tr>
<td>SP + 1</td>
<td>CCL</td>
</tr>
<tr>
<td>SP</td>
<td>CCH</td>
</tr>
</tbody>
</table>

Table 7-1. S12Z CPU Stacking Order on Entry to Interrupts

After the CCR is stacked, the I-bit (and the X-bit, if an 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.

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) | 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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>1(0/0)</td>
</tr>
</tbody>
</table>

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).
### 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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD Dn,#IMM1</td>
<td>1(0/0)</td>
<td>LD XY,#IMM18</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>LD Dn,#IMM2</td>
<td>1.5(0/0)</td>
<td>LD XY,#IMM3</td>
<td>1.5(0/0)</td>
</tr>
<tr>
<td>LD Dn,#IMM4</td>
<td>1.5(0/0)</td>
<td>LD S,#IMM3</td>
<td>1.5(0/0)</td>
</tr>
<tr>
<td>LD Dn,EXT24</td>
<td>2(0/0)</td>
<td>LD XY,EXT24</td>
<td>2.5(1/0)</td>
</tr>
<tr>
<td>LD Dn,REG</td>
<td>2.5(1/0)</td>
<td>LD XY,REG</td>
<td>2.5(1/0)</td>
</tr>
<tr>
<td>LD Dn,#IMM4</td>
<td>2.5(1/0)</td>
<td>LD XYS,REG</td>
<td>2.5(1/0)</td>
</tr>
<tr>
<td>LD Dn,(IDX)</td>
<td>3(1/0)</td>
<td>LD XYS,(IDX)</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>LD Dn,(++IDX)</td>
<td>3(1/0)</td>
<td>LD XYS,(++IDX)</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>LD Dn,(REG,IDX)</td>
<td>3(1/0)</td>
<td>LD XYS,REG,IDX</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>LD Dn,EXT2</td>
<td>3(1/0)</td>
<td>LD XY,EXT2</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>LD Dn,EXT3</td>
<td>3(1/0)</td>
<td>LD XY,EXT3</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>LD Dn,EXT1</td>
<td>3(1/0)</td>
<td>LD XY,EXT1</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>LD Dn,EXT2</td>
<td>3(1/0)</td>
<td>LD XY,EXT2</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>LD Dn,EXT3</td>
<td>3(1/0)</td>
<td>LD XY,EXT3</td>
<td>3(1/0)</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Dn,EXT24</td>
<td>2(0/1)</td>
<td>ST XY,EXT24</td>
<td>2(0/1)</td>
</tr>
<tr>
<td>ST Dn,REG</td>
<td>1(0/0)</td>
<td>ST XYS,REG</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>ST Dn,(IDX)</td>
<td>2(0/1)</td>
<td>ST XYS,(IDX)</td>
<td>2(0/1)</td>
</tr>
<tr>
<td>ST Dn,(++IDX)</td>
<td></td>
<td>ST XYS,(++IDX)</td>
<td></td>
</tr>
<tr>
<td>ST Dn,(REG,IDX)</td>
<td></td>
<td>ST XYS,(REG,IDX)</td>
<td></td>
</tr>
<tr>
<td>ST Dn,(IDX1)</td>
<td>2.5(0/1)</td>
<td>ST XYS,(IDX1)</td>
<td>2.5(0/1)</td>
</tr>
<tr>
<td>ST Dn,(IDX3)</td>
<td>2.5(0/1)</td>
<td>ST XYS,(IDX3)</td>
<td>2.5(0/1)</td>
</tr>
<tr>
<td>ST Dn,(IDX2,REG)</td>
<td></td>
<td>ST XYS,(IDX2,REG)</td>
<td></td>
</tr>
<tr>
<td>ST Dn,(IDX3,REG)</td>
<td></td>
<td>ST XYS,(IDX3,REG)</td>
<td></td>
</tr>
<tr>
<td>ST Dn,EXT1</td>
<td></td>
<td>ST XYS,EXT1</td>
<td></td>
</tr>
<tr>
<td>ST Dn,EXT2</td>
<td>2.5(0/1)</td>
<td>ST XYS,EXT2</td>
<td>2.5(0/1)</td>
</tr>
<tr>
<td>ST Dn,EXT3</td>
<td>2.5(0/1)</td>
<td>ST XYS,EXT3</td>
<td>2.5(0/1)</td>
</tr>
<tr>
<td>ST Dn,(REG,IDX)</td>
<td>3.5(1/1)</td>
<td>ST XYS,(REG,IDX)</td>
<td>3.5(1/1)</td>
</tr>
<tr>
<td>ST Dn,(IDX1)</td>
<td>4(1/1)</td>
<td>ST XYS,(IDX1)</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>ST Dn,(IDX3)</td>
<td>4(1/1)</td>
<td>ST XYS,(IDX3)</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>ST Dn,(EXT3)</td>
<td>4(1/1)</td>
<td>ST XYS,(EXT3)</td>
<td>4(1/1)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSH oprregs</td>
<td>1.5 + 0.5*n</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUL oprregs</td>
<td>2.5 + 0.5*n</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEA XYS,(IMMs8,XYS)</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>LEA D67XYS,(IDX)</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>LEA D67XYS,(++IDX)</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>LEA D67XYS,(REG,IDX)</td>
<td>1.5(0/0)</td>
</tr>
<tr>
<td>LEA D67XYS,(IDX1)</td>
<td>2(0/1)</td>
</tr>
<tr>
<td>LEA D67XYS,(IDX3)</td>
<td>2.5(0/1)</td>
</tr>
<tr>
<td>LEA D67XYS,(IDX2,REG)</td>
<td>3.5(0/1)</td>
</tr>
<tr>
<td>LEA D67XYS,(IDX3,REG)</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>LEA D67XYS,EXT1</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>LEA D67XYS,EXT2</td>
<td>3.5(1/1)</td>
</tr>
<tr>
<td>LEA D67XYS,EXT3</td>
<td>4(1/1)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR Dn</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>CLR XY</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>CLR REG</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>CLR.bwpl (IDX)</td>
<td>2(0/1)</td>
</tr>
<tr>
<td>CLR.bwpl (++IDX)</td>
<td>2.5(0/1)</td>
</tr>
<tr>
<td>CLR.bwpl (REG,IDX)</td>
<td>3.5(1/1)</td>
</tr>
<tr>
<td>CLR.bwpl [REG,IDX]</td>
<td>4(1/1)</td>
</tr>
</tbody>
</table>

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).
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 8-10. Register-To-Register Transfer (TFR, SEX, ZEX) Execution Timing

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Dn</th>
<th>XYS</th>
<th>CCL</th>
<th>CCH</th>
<th>CCW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dn</td>
<td>Dn</td>
<td>1(0/0)</td>
<td>1(0/0)</td>
<td>1(0/0)</td>
<td>1.5(0/0)</td>
<td>1.5(0/0)</td>
</tr>
<tr>
<td>XYS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.5(0/0)</td>
</tr>
<tr>
<td>CCH</td>
<td></td>
<td></td>
<td></td>
<td>1(0/0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8-11. Register-To-Register Exchange (EXG) Execution Timing

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Dn</th>
<th>XYS</th>
<th>CCL</th>
<th>CCH</th>
<th>CCW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dn</td>
<td>Dn</td>
<td>1(0/0)</td>
<td>1(0/0)</td>
<td>1(0/0)</td>
<td>1.5(0/0)</td>
<td>1.5(0/0)</td>
</tr>
<tr>
<td>XYS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.5(0/0)</td>
</tr>
<tr>
<td>CCH</td>
<td></td>
<td></td>
<td>1.5(0/0)</td>
<td>1.5(0/0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCW</td>
<td></td>
<td></td>
<td>1.5(0/0)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8-12. Logical Operation Execution Timing

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;OP&gt; Dn,#IMM1</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,#IMM2</td>
<td></td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,#IMM4</td>
<td>1.5(0/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,EXT24</td>
<td>2.5(1/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,REG</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,#IMMe4</td>
<td></td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,(IDX)</td>
<td>2.5(1/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,(++IDX)</td>
<td></td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,(REG,IDX)</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,(IDX1)</td>
<td></td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,(IDX3)</td>
<td></td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,(IDX2,REG)</td>
<td></td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,(IDX3,REG)</td>
<td></td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,EXT1</td>
<td></td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,EXT2</td>
<td></td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,EXT3</td>
<td></td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,REG,IDX</td>
<td>4(2/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,[IDX1]</td>
<td></td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,[IDX3]</td>
<td></td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,[EXT3]</td>
<td>4.5(2/0)</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM REG</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>COM.bwl IDX1</td>
<td>3.5(1/1)</td>
</tr>
<tr>
<td>COM.bwl (++IDX)</td>
<td>3.5(1/1)</td>
</tr>
<tr>
<td>COM.bwl REG,IDX</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>COM.bwl IDX1,REG</td>
<td>5(2/1)</td>
</tr>
<tr>
<td>COM.bwl [REG,IDX]</td>
<td>5.5(2/1)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;OP&gt; Dn</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; REG</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bwl IDX1</td>
<td>3.5(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bwl (++IDX)</td>
<td>3.5(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bwl REG,IDX</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bwl IDX1,REG</td>
<td>5(2/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bwl [REG,IDX]</td>
<td>5.5(2/1)</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;OP&gt; Dn,#IMM1</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,#IMM2</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,#IMM4</td>
<td>1.5(0/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,EXT24</td>
<td>2.5(1/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,REG</td>
<td>2.5(1/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,#IMMe4</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,(IDX)</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,(++IDX)</td>
<td>2.5(1/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,(REG,IDX)</td>
<td>2.5(1/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,(IDX1)</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,(IDX3)</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,(IDX2,REG)</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,(IDX3,REG)</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,EXT1</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,EXT2</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,EXT3</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,[REG,IDX]</td>
<td>4(2/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,[IDX1]</td>
<td>4.5(2/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,[IDX3]</td>
<td>4.5(2/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; Dn,[EXT3]</td>
<td>4.5(2/0)</td>
</tr>
</tbody>
</table>

**Table 8-16. Pointer Arithmetic Operation Execution Timing**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB D6,X,Y</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>CMP X,Y</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>CMP YX</td>
<td>1(0/0)</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEG REG</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>NEG.bwl (IDX)</td>
<td>3.5(1/1)</td>
</tr>
<tr>
<td>NEG.bwl (++IDX)</td>
<td>3.5(1/1)</td>
</tr>
<tr>
<td>NEG.bwl (REG,IDX)</td>
<td>3.5(1/1)</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS Dn</td>
<td>1(0/0)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT Dn</td>
<td>1(0/0)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLB Ds,Dd</td>
<td>1(0/0)</td>
</tr>
</tbody>
</table>

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\(^1\)

<table>
<thead>
<tr>
<th>Source2</th>
<th>Dn</th>
<th>REG</th>
<th>IMM</th>
<th>(IDX) (+IDX)</th>
<th>(IDX1) (IDX3)</th>
<th>(IDX2,REG) (IDX3,REG) EXT1</th>
<th>EXT2</th>
<th>EXT3</th>
<th>[REG,IDX]</th>
<th>[IDX1]</th>
<th>[IDX3]</th>
<th>[EXT3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dn</td>
<td>2(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>3.5(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IMM1</td>
<td>2(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IMM2</td>
<td>3.5(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IMM4</td>
<td>4(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>REG IMMe4</td>
<td>2(0/0)</td>
<td>3(0/0)</td>
<td>3.5(1/0)</td>
<td>5(1/0)</td>
<td>6(2/0)</td>
<td>6.5(2/0)</td>
<td>7.5(2/0)</td>
<td>8(2/0)</td>
<td>3.5(1/0)</td>
<td>4.5(1/0)</td>
<td>6(1/0)</td>
<td>6.5(1/0)</td>
</tr>
<tr>
<td></td>
<td>3.5(0/0)</td>
<td>4.5(0/0)</td>
<td>6(1/0)</td>
<td>6.5(1/0)</td>
<td>7.5(2/0)</td>
<td>8(2/0)</td>
<td>9(3/0)</td>
<td>9.5(3/0)</td>
<td>5(1/0)</td>
<td>6(1/0)</td>
<td>7.5(2/0)</td>
<td>8(2/0)</td>
</tr>
<tr>
<td>(IDX3) (IDX2,REG) (IDX3,REG) EXT1</td>
<td>4(1/0)</td>
<td>5(1/0)</td>
<td>6(2/0)</td>
<td>6.5(2/0)</td>
<td>7.5(3/0)</td>
<td>8(3/0)</td>
<td>5.5(1/0)</td>
<td>6.5(1/0)</td>
<td>7.5(2/0)</td>
<td>8(2/0)</td>
<td>9(3/0)</td>
<td>9.5(3/0)</td>
</tr>
<tr>
<td>EXT2</td>
<td>EXT3</td>
<td>5.5(1/0)</td>
<td>6.5(1/0)</td>
<td>7.5(2/0)</td>
<td>8(2/0)</td>
<td>9(3/0)</td>
<td>9.5(3/0)</td>
<td>5(2/0)</td>
<td>6(2/0)</td>
<td>7.5(3/0)</td>
<td>8(3/0)</td>
<td>9(4/0)</td>
</tr>
<tr>
<td>[REG,IDX]</td>
<td>6.5(2/0)</td>
<td>7.5(2/0)</td>
<td>9(3/0)</td>
<td>9.5(3/0)</td>
<td>10.5(4/0)</td>
<td>11(4/0)</td>
<td>7(2/0)</td>
<td>7.5(2/0)</td>
<td>9(3/0)</td>
<td>9.5(3/0)</td>
<td>10.5(4/0)</td>
<td>11(4/0)</td>
</tr>
</tbody>
</table>

Table 8-22. Unsigned Multiply (MULU) Execution Timing\(^1\)

<table>
<thead>
<tr>
<th>Source2</th>
<th>Dn</th>
<th>REG</th>
<th>IMM</th>
<th>(IDX) (+IDX)</th>
<th>(IDX1) (IDX3)</th>
<th>(IDX2,REG) (IDX3,REG) EXT1</th>
<th>EXT2</th>
<th>EXT3</th>
<th>[REG,IDX]</th>
<th>[IDX1]</th>
<th>[IDX3]</th>
<th>[EXT3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dn</td>
<td>1(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>2.5(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IMM1</td>
<td>1(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IMM2</td>
<td>2.5(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

\(^1\) 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\(^1\)

<table>
<thead>
<tr>
<th>Source2</th>
<th>Dn</th>
<th>REG</th>
<th>IMM4</th>
<th>(IDX) (++IDX)</th>
<th>(IDX1) (IDX3)</th>
<th>EXT1</th>
<th>EXT2</th>
<th>EXT3</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM4</td>
<td>3(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>REG</td>
<td>1(0/0)</td>
<td>2(0/0)</td>
<td>3.5(1/0)</td>
<td>4(1/0)</td>
<td>5(2/0)</td>
<td>5.5(2/0)</td>
<td>6.5(2/0)</td>
<td>7(2/0)</td>
</tr>
<tr>
<td>IMM4</td>
<td>2.5(0/0)</td>
<td>3.5(0/0)</td>
<td>5(1/0)</td>
<td>5.5(1/0)</td>
<td>6.5(2/0)</td>
<td>7(2/0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(IDX) (++IDX)</td>
<td>2.5(1/0)</td>
<td>3.5(1/0)</td>
<td>5(2/0)</td>
<td>5.5(2/0)</td>
<td>6.5(3/0)</td>
<td>7(3/0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REG</td>
<td>4(1/0)</td>
<td>5(1/0)</td>
<td>6.5(2/0)</td>
<td>7(2/0)</td>
<td>8(3/0)</td>
<td>8.5(3/0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMM4</td>
<td>(IDX1) (IDX3)</td>
<td>3(1/0)</td>
<td>3.5(1/0)</td>
<td>5(2/0)</td>
<td>5.5(2/0)</td>
<td>6.5(3/0)</td>
<td>7(3/0)</td>
<td></td>
</tr>
<tr>
<td>REG</td>
<td>(IDX1) (IDX3)</td>
<td>4.5(1/0)</td>
<td>5(1/0)</td>
<td>6.5(2/0)</td>
<td>7(2/0)</td>
<td>8(3/0)</td>
<td>8.5(3/0)</td>
<td></td>
</tr>
<tr>
<td>IMM4</td>
<td>[REG,IDX]</td>
<td>4(2/0)</td>
<td>5(2/0)</td>
<td>6.5(3/0)</td>
<td>7(3/0)</td>
<td>8(4/0)</td>
<td>8.5(4/0)</td>
<td></td>
</tr>
<tr>
<td>REG</td>
<td>[REG,IDX]</td>
<td>5.5(2/0)</td>
<td>6.5(2/0)</td>
<td>8(3/0)</td>
<td>8.5(3/0)</td>
<td>9.5(4/0)</td>
<td>10(4/0)</td>
<td></td>
</tr>
<tr>
<td>IMM4</td>
<td>[IDX1] [IDX2] [EXT3]</td>
<td>4.5(2/0)</td>
<td>5(2/0)</td>
<td>6.5(3/0)</td>
<td>7(3/0)</td>
<td>8(4/0)</td>
<td>8.5(4/0)</td>
<td></td>
</tr>
<tr>
<td>REG</td>
<td>[IDX1] [IDX2] [EXT3]</td>
<td>6(2/0)</td>
<td>6.5(2/0)</td>
<td>8(3/0)</td>
<td>8.5(3/0)</td>
<td>9.5(4/0)</td>
<td>10(4/0)</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) 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\(^1\)

<table>
<thead>
<tr>
<th>Source2</th>
<th>Dn</th>
<th>REG</th>
<th>IMM4</th>
<th>(IDX) (++IDX)</th>
<th>(IDX1) (IDX3)</th>
<th>EXT1</th>
<th>EXT2</th>
<th>EXT3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dn</td>
<td>3.5(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IMM1</td>
<td>3.5(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Dn</td>
<td>6.5(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IMM1</td>
<td>6.5(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

\(^1\) 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 8-23. Signed Fractional Multiply (QMULS) Execution Timing

<table>
<thead>
<tr>
<th>Source2</th>
<th>Source1</th>
<th>Dn</th>
<th>REG IMMe4</th>
<th>(IDX) (++IDX) (REG,IDX)</th>
<th>(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3</th>
<th>[REG,IDX]</th>
<th>[IDX1] [IDX3] [EXT3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM2</td>
<td></td>
<td>4(0/0)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>IMM4</td>
<td>7(0/0)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>REG IMMe4</td>
<td></td>
<td>3.5(0/0)</td>
<td>4.5(0/0)</td>
<td>6(1/0)</td>
<td>6.5(1/0)</td>
<td>7.5(2/0)</td>
<td>8(2/0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.5(0/0)</td>
<td>7.5(0/0)</td>
<td>9(1/0)</td>
<td>9.5(1/0)</td>
<td>10.5(2/0)</td>
<td>11(2/0)</td>
</tr>
<tr>
<td>(IDX) (+IDX) (REG,IDX)</td>
<td></td>
<td>5(1/0)</td>
<td>6(1/0)</td>
<td>7.5(2/0)</td>
<td>8(2/0)</td>
<td>9(3/0)</td>
<td>9.5(3/0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8(1/0)</td>
<td>9(1/0)</td>
<td>10.5(2/0)</td>
<td>11(2/0)</td>
<td>12(3/0)</td>
<td>12.5(3/0)</td>
</tr>
<tr>
<td>(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3</td>
<td></td>
<td>5.5(1/0)</td>
<td>6(1/0)</td>
<td>7.5(2/0)</td>
<td>8(2/0)</td>
<td>9(3/0)</td>
<td>9.5(3/0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8.5(1/0)</td>
<td>9(1/0)</td>
<td>10.5(2/0)</td>
<td>11(2/0)</td>
<td>12(3/0)</td>
<td>12.5(3/0)</td>
</tr>
<tr>
<td>[REG,IDX]</td>
<td></td>
<td>6.5(2/0)</td>
<td>7.5(2/0)</td>
<td>9(3/0)</td>
<td>9.5(3/0)</td>
<td>10.5(4/0)</td>
<td>11(4/0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9.5(2/0)</td>
<td>10.5(2/0)</td>
<td>12(3/0)</td>
<td>12.5(3/0)</td>
<td>13.5(4/0)</td>
<td>14(4/0)</td>
</tr>
<tr>
<td>[IDX1] [IDX2] [EXT3]</td>
<td></td>
<td>7(2/0)</td>
<td>7.5(2/0)</td>
<td>9(3/0)</td>
<td>9.5(3/0)</td>
<td>10.5(4/0)</td>
<td>11(4/0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10(2/0)</td>
<td>10.5(2/0)</td>
<td>12(3/0)</td>
<td>12.5(3/0)</td>
<td>13.5(4/0)</td>
<td>14(4/0)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Source2</th>
<th>Source1</th>
<th>Dn</th>
<th>REG IMMe4</th>
<th>(IDX) (+IDX) (REG,IDX)</th>
<th>(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3</th>
<th>[REG,IDX]</th>
<th>[IDX1] [IDX3] [EXT3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM1</td>
<td></td>
<td>3(0/0)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>IMM2</td>
<td>6(0/0)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.5(0/0)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.5(0/0)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Linear S12 Core Reference Manual, Rev. 1.01
Table 8-24. Unsigned Fractional Multiply (QMULU) Execution Timing

<table>
<thead>
<tr>
<th>Source2</th>
<th>Source1</th>
<th>Dn</th>
<th>REG IMMm4</th>
<th>++IDX</th>
<th>REG,IDX</th>
<th>[REG,IDX]</th>
<th>[IDX1]</th>
<th>[IDX3]</th>
<th>[EXT1]</th>
<th>[EXT2]</th>
<th>[EXT3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM4</td>
<td>6.5(0/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>REG IMMm4</td>
<td>3(0/0)</td>
<td>4(0/0)</td>
<td>5.5(1/0)</td>
<td>6(1/0)</td>
<td>7(2/0)</td>
<td>7.5(2/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>(IDX) (+IDX) (REG,IDX)</td>
<td>6(0/0)</td>
<td>7(0/0)</td>
<td>8.5(1/0)</td>
<td>9(1/0)</td>
<td>10(2/0)</td>
<td>10.5(2/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3</td>
<td>7.5(1/0)</td>
<td>8.5(1/0)</td>
<td>10(2/0)</td>
<td>10.5(2/0)</td>
<td>11.5(3/0)</td>
<td>12(3/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>[REG,IDX]</td>
<td>5(1/0)</td>
<td>5.5(1/0)</td>
<td>7(2/0)</td>
<td>7.5(2/0)</td>
<td>8.5(3/0)</td>
<td>9(3/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>[IDX1] [IDX3] [EXT1] EXT2 EXT3</td>
<td>8(1/0)</td>
<td>8.5(1/0)</td>
<td>10(2/0)</td>
<td>10.5(2/0)</td>
<td>11.5(3/0)</td>
<td>12(3/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>[REG,IDX]</td>
<td>6(2/0)</td>
<td>7(2/0)</td>
<td>8.5(3/0)</td>
<td>9(3/0)</td>
<td>10(4/0)</td>
<td>10.5(4/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>[IDX1] [IDX3] [EXT1] EXT2 EXT3</td>
<td>9(2/0)</td>
<td>10(2/0)</td>
<td>11.5(3/0)</td>
<td>12(3/0)</td>
<td>13(4/0)</td>
<td>13.5(4/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>[REG,IDX]</td>
<td>6.5(2/0)</td>
<td>8.5(2/0)</td>
<td>8.5(3/0)</td>
<td>9(3/0)</td>
<td>10(4/0)</td>
<td>10.5(4/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>[IDX1] [IDX2] [EXT3]</td>
<td>9.5(2/0)</td>
<td>10(2/0)</td>
<td>11.5(3/0)</td>
<td>12(3/0)</td>
<td>13(4/0)</td>
<td>13.5(4/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Source1</th>
<th>Source2</th>
<th>Dn</th>
<th>REG IMMm4</th>
<th>++IDX</th>
<th>REG,IDX</th>
<th>[REG,IDX]</th>
<th>[IDX1]</th>
<th>[IDX3]</th>
<th>[EXT1]</th>
<th>[EXT2]</th>
<th>[EXT3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM1</td>
<td>2.5(0/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>Dn</td>
<td>2.5(0/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>4(0/0)</td>
<td>2.5(0/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>4(0/0)</td>
<td>2.5(0/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>
### Table 8-25. Signed Multiply-and-Accumulate (MACS) Execution Timing\(^1\)

<table>
<thead>
<tr>
<th>Source2</th>
<th>Dn</th>
<th>REG IMMe4</th>
<th>(IDX) (+IDX)</th>
<th>(IDX1) (IDX3)</th>
<th>(IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3</th>
<th>REG,IDX</th>
<th>[IDX1] [IDX3] [EXT3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM2</td>
<td>3(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>4.5(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IMM4</td>
<td>4.5(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>REG IMMe4</td>
<td>2.5(0/0)</td>
<td>3.5(0/0)</td>
<td>5(1/0)</td>
<td>5.5(1/0)</td>
<td>6.5(2/0)</td>
<td>7(2/0)</td>
<td>8.5(2/0)</td>
</tr>
<tr>
<td></td>
<td>4(0/0)</td>
<td>5(0/0)</td>
<td>6.5(1/0)</td>
<td>7(1/0)</td>
<td>8(2/0)</td>
<td>9.5(3/0)</td>
<td>10(3/0)</td>
</tr>
<tr>
<td>(IDX) (+IDX) (REG,IDX)</td>
<td>4.5(1/0)</td>
<td>5(1/0)</td>
<td>6.5(2/0)</td>
<td>7(2/0)</td>
<td>8(3/0)</td>
<td>9.5(3/0)</td>
<td>10(3/0)</td>
</tr>
<tr>
<td></td>
<td>5.5(1/0)</td>
<td>6.5(1/0)</td>
<td>8(2/0)</td>
<td>8.5(2/0)</td>
<td>9.5(3/0)</td>
<td>10(3/0)</td>
<td></td>
</tr>
<tr>
<td>(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3</td>
<td>6(1/0)</td>
<td>7(1/0)</td>
<td>8(2/0)</td>
<td>8.5(2/0)</td>
<td>9.5(3/0)</td>
<td>10(3/0)</td>
<td></td>
</tr>
<tr>
<td>[REG,IDX]</td>
<td>5.5(2/0)</td>
<td>6.5(2/0)</td>
<td>8(3/0)</td>
<td>8.5(3/0)</td>
<td>9.5(4/0)</td>
<td>10(4/0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7(2/0)</td>
<td>8(2/0)</td>
<td>9.5(3/0)</td>
<td>10(3/0)</td>
<td>11(4/0)</td>
<td>11.5(4/0)</td>
<td></td>
</tr>
<tr>
<td>[IDX1] [IDX2] [EXT3]</td>
<td>6(2/0)</td>
<td>7(2/0)</td>
<td>8(3/0)</td>
<td>8.5(3/0)</td>
<td>9.5(4/0)</td>
<td>10(4/0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7.5(2/0)</td>
<td>8.5(2/0)</td>
<td>9.5(3/0)</td>
<td>10(3/0)</td>
<td>11(4/0)</td>
<td>11.5(4/0)</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) 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\(^1\)

<table>
<thead>
<tr>
<th>Source2</th>
<th>Dn</th>
<th>REG IMMe4</th>
<th>(IDX) (+IDX)</th>
<th>(IDX1) (IDX3)</th>
<th>(IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3</th>
<th>REG,IDX</th>
<th>[IDX1] [IDX3] [EXT3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dn</td>
<td>1.5(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>3(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IMM1</td>
<td>1.5(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>3(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IMM2</td>
<td>2(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>3.5(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

\(^1\) 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 8-27. Signed Divide/Modulo (DIVS/MODS) Execution Timing

<table>
<thead>
<tr>
<th>Source2</th>
<th>Source1</th>
<th>REG IMMe4</th>
<th>(IDX) (REG,IDX)</th>
<th>(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3</th>
<th>[REG,IDX]</th>
<th>[IDX1] [IDX3] [EXT3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM4</td>
<td>3.5(0/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>REG IMMMe4</td>
<td>1.5(0/0)</td>
<td>2.5(0/0)</td>
<td>4(1/0)</td>
<td>4.5(1/0)</td>
<td>5.5(2/0)</td>
<td>6(2/0)</td>
</tr>
<tr>
<td></td>
<td>3(0/0)</td>
<td>4(0/0)</td>
<td>5.5(1/0)</td>
<td>6(1/0)</td>
<td>7(2/0)</td>
<td>7.5(2/0)</td>
</tr>
<tr>
<td>(IDX) (+IDX) (REG,IDX)</td>
<td>3(1/0)</td>
<td>4(1/0)</td>
<td>5.5(2/0)</td>
<td>6(2/0)</td>
<td>7(3/0)</td>
<td>7.5(3/0)</td>
</tr>
<tr>
<td></td>
<td>4.5(1/0)</td>
<td>5.5(1/0)</td>
<td>7(2/0)</td>
<td>7.5(2/0)</td>
<td>8.5(3/0)</td>
<td>9(3/0)</td>
</tr>
<tr>
<td>(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3</td>
<td>3.5(1/0)</td>
<td>4(1/0)</td>
<td>5.5(2/0)</td>
<td>6(2/0)</td>
<td>7(3/0)</td>
<td>7.5(3/0)</td>
</tr>
<tr>
<td></td>
<td>5(1/0)</td>
<td>5.5(1/0)</td>
<td>7(2/0)</td>
<td>7.5(2/0)</td>
<td>8.5(3/0)</td>
<td>9(3/0)</td>
</tr>
<tr>
<td>[REG,IDX]</td>
<td>4.5(2/0)</td>
<td>5.5(2/0)</td>
<td>7(3/0)</td>
<td>7.5(3/0)</td>
<td>8.5(4/0)</td>
<td>9(4/0)</td>
</tr>
<tr>
<td></td>
<td>6(2/0)</td>
<td>7(2/0)</td>
<td>8.5(3/0)</td>
<td>9(3/0)</td>
<td>10(4/0)</td>
<td>10.5(4/0)</td>
</tr>
<tr>
<td>[IDX1] [IDX2] [EXT3]</td>
<td>5(2/0)</td>
<td>5.5(2/0)</td>
<td>7(3/0)</td>
<td>7.5(3/0)</td>
<td>8.5(4/0)</td>
<td>9(4/0)</td>
</tr>
<tr>
<td></td>
<td>6.5(2/0)</td>
<td>7(2/0)</td>
<td>8.5(3/0)</td>
<td>9(3/0)</td>
<td>10(4/0)</td>
<td>10.5(4/0)</td>
</tr>
</tbody>
</table>

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 8-26. Unsigned Multiply-and-Accumulate (MACU) Execution Timing

<table>
<thead>
<tr>
<th>Source2</th>
<th>Source1</th>
<th>REG IMMe4</th>
<th>(IDX) (REG,IDX)</th>
<th>(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3</th>
<th>[REG,IDX]</th>
<th>[IDX1] [IDX3] [EXT3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM4</td>
<td>3+n(0/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>IMM1</td>
<td>3+n(0/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>IMM2 IMM4</td>
<td>3.5+n(0/0)</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>

Linear S12 Core Reference Manual, Rev. 1.01

Freescale Semiconductor
### Table 8-27. Signed Divide/Modulo (DIVS/MODS) Execution Timing

<table>
<thead>
<tr>
<th>Source2</th>
<th>Dn</th>
<th>REG IMMe4</th>
<th>(IDX) (++IDX)</th>
<th>(IDX1) (IDX3)</th>
<th>[REG,IDX]</th>
<th>[IDX1] [IDX3] [EXT3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>REG IMMe4</td>
<td>3+n(0/0)</td>
<td>4+n(0/0)</td>
<td>5.5+n(1/0)</td>
<td>6+n(1/0)</td>
<td>7+n(2/0)</td>
<td>7.5+n(2/0)</td>
</tr>
<tr>
<td>(++IDX)</td>
<td>4.5+n(1/0)</td>
<td>5.5+n(1/0)</td>
<td>7+n(2/0)</td>
<td>7.5+n(2/0)</td>
<td>8.5+n(3/0)</td>
<td>9+n(3/0)</td>
</tr>
<tr>
<td>(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3</td>
<td>5+n(1/0)</td>
<td>5.5+n(1/0)</td>
<td>7+n(2/0)</td>
<td>7.5+n(2/0)</td>
<td>8.5+n(3/0)</td>
<td>9+n(3/0)</td>
</tr>
<tr>
<td>[REG,IDX]</td>
<td>6+n(2/0)</td>
<td>7+n(2/0)</td>
<td>8.5+n(3/0)</td>
<td>9+n(3/0)</td>
<td>10+n(4/0)</td>
<td>10.5+n(4/0)</td>
</tr>
<tr>
<td>[IDX1] [IDX2] [EXT3]</td>
<td>6.5+n(2/0)</td>
<td>7+n(2/0)</td>
<td>8.5+n(3/0)</td>
<td>9+n(3/0)</td>
<td>10+n(4/0)</td>
<td>10.5+n(4/0)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Source2</th>
<th>Dn</th>
<th>REG IMMe4</th>
<th>(IDX) (++IDX)</th>
<th>(IDX1) (IDX3)</th>
<th>[REG,IDX]</th>
<th>[IDX1] [IDX3] [EXT3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dn</td>
<td>2.5+n(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IMM1</td>
<td>2.5+n(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IMM2 IMM4</td>
<td>3+n(0/0)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>REG IMMe4</td>
<td>2.5+n(0/0)</td>
<td>3.5+n(0/0)</td>
<td>5+n(1/0)</td>
<td>5.5+n(1/0)</td>
<td>6.5+n(2/0)</td>
<td>7+n(2/0)</td>
</tr>
<tr>
<td>(++IDX) (REG,IDX)</td>
<td>4+n(1/0)</td>
<td>5+n(1/0)</td>
<td>6.5+n(2/0)</td>
<td>7+n(2/0)</td>
<td>8+n(3/0)</td>
<td>8.5+n(3/0)</td>
</tr>
<tr>
<td>(IDX1) (IDX3) (IDX2,REG) (IDX3,REG) EXT1 EXT2 EXT3</td>
<td>4.5+n(1/0)</td>
<td>5+n(1/0)</td>
<td>6.5+n(2/0)</td>
<td>7+n(2/0)</td>
<td>8+n(3/0)</td>
<td>8.5+n(3/0)</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Source2</th>
<th>Dn</th>
<th>REG</th>
<th>IMM4</th>
<th>(IDX)</th>
<th>(IDX1)</th>
<th>(IDX2,REG)</th>
<th>(IDX3,REG)</th>
<th>EXT1</th>
<th>EXT2</th>
<th>EXT3</th>
<th>[REG,IDX]</th>
<th>[IDX1]</th>
<th>[IDX2]</th>
<th>[EXT3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[REG,IDX]</td>
<td>5.5+n(2/0)</td>
<td>6.5+n(2/0)</td>
<td>8+n(3/0)</td>
<td>8.5+n(3/0)</td>
<td>9.5+n(4/0)</td>
<td>10+n(4/0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[IDX1]</td>
<td>[IDX2]</td>
<td>[EXT3]</td>
<td>6+n(2/0)</td>
<td>6.5+n(2/0)</td>
<td>8+n(3/0)</td>
<td>8.5+n(3/0)</td>
<td>9.5+n(4/0)</td>
<td>10+n(4/0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

8.2.23 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

<table>
<thead>
<tr>
<th>Source2 (shift width)</th>
<th>Ds</th>
<th>REG IMM04</th>
<th>(IDX)</th>
<th>(IDX1)</th>
<th>(IDX3)</th>
<th>[IDX1]</th>
<th>[IDX2]</th>
<th>[EXT]</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM (1 or 2)</td>
<td></td>
<td>1(0/0)</td>
<td>1.5(0/0)</td>
<td>2.5(1/0)</td>
<td>3(1/0)</td>
<td>4(2/0)</td>
<td>4.5(2/0)</td>
<td></td>
</tr>
<tr>
<td>IMM (3..31) REG</td>
<td></td>
<td>1(0/0)</td>
<td>2(0/0)</td>
<td>3.5(1/0)</td>
<td>4(1/0)</td>
<td>5(2/0)</td>
<td>5.5(2/0)</td>
<td></td>
</tr>
<tr>
<td>(IDX) (++IDX) (REG,IDX)</td>
<td>2.5(1/0)</td>
<td>3.5(1/0)</td>
<td>5(2/0)</td>
<td>5.5(2/0)</td>
<td>6.5(3/0)</td>
<td>7(3/0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(IDX3) (IDX2,REG) EXT1 EXT2 EXT3</td>
<td>3(1/0)</td>
<td>3.5(1/0)</td>
<td>5(2/0)</td>
<td>5.5(2/0)</td>
<td>6.5(3/0)</td>
<td>7(3/0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[REG,IDX]</td>
<td>4(2/0)</td>
<td>5(2/0)</td>
<td>6.5(3/0)</td>
<td>7(3/0)</td>
<td>8.5(4/0)</td>
<td>9(4/0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[IDX1] [IDX2] [EXT3]</td>
<td>4.5(2/0)</td>
<td>5(2/0)</td>
<td>6.5(3/0)</td>
<td>7(3/0)</td>
<td>8.5(4/0)</td>
<td>9(4/0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[IDX1] [IDX1] [REG,IDX]</td>
<td>4(2/0)</td>
<td>5(2/0)</td>
<td>6.5(3/0)</td>
<td>7(3/0)</td>
<td>8.5(4/0)</td>
<td>9(4/0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[IDX1] [IDX1] [IDX1] [IDX1] [REG,IDX]</td>
<td>4.5(2/0)</td>
<td>5(2/0)</td>
<td>6.5(3/0)</td>
<td>7(3/0)</td>
<td>8.5(4/0)</td>
<td>9(4/0)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8-31. Execution Timing for Shifting a Memory Operand by 1 or 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;OP&gt;. REG</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>&lt;OP&gt;. bwpl (IDX)</td>
<td>3.5(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;. bwpl (++IDX)</td>
<td>3.5(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;. bwpl (REG,IDX)</td>
<td>3.5(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;. bwpl (IDX1)</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;. bwpl (IDX3)</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;. bwpl (IDX1)</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;. bwpl (IDX3)</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;. bwpl EXT1</td>
<td>5(2/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;. bwpl EXT2</td>
<td>5(2/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;. bwpl EXT3</td>
<td>5(2/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;. bwpl [REG,IDX]</td>
<td>5(2/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;. bwpl [IDX1]</td>
<td>5(2/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;. bwpl [IDX3]</td>
<td>5(2/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;. bwpl [EXT3]</td>
<td>5(2/1)</td>
</tr>
</tbody>
</table>

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).
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 8-32. Rotate (ROL, ROR) Execution Timing

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;OP&gt; REG</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bwpl (IDX)</td>
<td>3.5(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bwpl (++IDX)</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bwpl (REG,IDX)</td>
<td>5(2/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bwpl (IDX1)</td>
<td>5.5(2/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bwpl (IDX3)</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bwpl (IDX2,REG)</td>
<td>4.5(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bwpl (IDX3,REG)</td>
<td>5.5(2/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bwpl EXT1</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bwpl EXT2</td>
<td>4.5(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bwpl EXT3</td>
<td>5.5(2/1)</td>
</tr>
</tbody>
</table>

### Table 8-33. Bit-Manipulation (BCLR, BSET, BTGL) Execution Timing

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;OP&gt; Di,#opr5i</td>
<td>1.5(0/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; REG,#opr5i</td>
<td>1.5(0/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; REG,Dn</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bw1 (IDX),#opr5i</td>
<td>4.5(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bw1 (++IDX),#opr5i</td>
<td>5.5(2/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bw1 (REG,IDX),#opr5i</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bw1 (IDX1),Dn</td>
<td>4.5(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bw1 (IDX3),Dn</td>
<td>5.5(2/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bw1 (IDX2,REG),#opr5i</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bw1 (IDX3,REG),#opr5i</td>
<td>4.5(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bw1 EXT1,#opr5i</td>
<td>5.5(2/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bw1 EXT2,#opr5i</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bw1 EXT3,#opr5i</td>
<td>4.5(1/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bw1 [REG,IDX],Dn</td>
<td>5.5(2/1)</td>
</tr>
<tr>
<td>&lt;OP&gt;.bw1 [REG,IDX],#opr5i</td>
<td>4(1/1)</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;OP&gt;.bwl [IDX1],#opr5i</code></td>
<td>6(2/1)</td>
</tr>
<tr>
<td><code>&lt;OP&gt;.bwl [IDX3],#opr5i</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;OP&gt;.bwl [EXT3],#opr5i</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;OP&gt;.bwl [IDX1],Dn</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;OP&gt;.bwl [IDX3],Dn</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;OP&gt;.bwl [EXT3],Dn</code></td>
<td></td>
</tr>
</tbody>
</table>

Table 8-33. Bit-Manipulation (BCLR, BSET, BTGL) Execution Timing

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFEXT Dd,Ds,#width:offset</td>
<td>2(0/0)</td>
<td>BFEXT REG,Ds,#width:offset</td>
<td>2(0/0)</td>
</tr>
<tr>
<td>BFEXT Dd,Ds,Dp</td>
<td></td>
<td>BFEXT REG,Ds,REG,Dp</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,REG,#width:offset</td>
<td>2.5(0/0)</td>
<td>BFEXT Dd,REG,REG,Dp</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,#IMMe4,#width:offset</td>
<td></td>
<td>BFEXT Dd,#IMMe4,Dp</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,REG,Dp</td>
<td></td>
<td>BFEXT Dd,REG,REG,Dp</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,#IMMe4,Dp</td>
<td></td>
<td>BFEXT Dd,#IMMe4,Dp</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,(IDX),Ds,#width:offset</td>
<td>4(1/0)</td>
<td>BFEXT Dd,(++IDX),Ds,#width:offset</td>
<td>3.5(0/1)</td>
</tr>
<tr>
<td>BFEXT Dd,(REG,IDX),#width:offset</td>
<td></td>
<td>BFEXT Dd,(REG,IDX),Ds,#width:offset</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,(IDX),Dp</td>
<td></td>
<td>BFEXT Dd,(REG,IDX),Ds,Dp</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,(++IDX),Dp</td>
<td></td>
<td>BFEXT Dd,(REG,IDX),Ds,Dp</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,(REG,IDX),Dp</td>
<td></td>
<td>BFEXT Dd,(REG,IDX),Ds,Dp</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,(IDX1),#width:offset</td>
<td>4(1/0)</td>
<td>BFEXT Dd,(IDX1),#width:offset</td>
<td>3.5(0/1)</td>
</tr>
<tr>
<td>BFEXT Dd,(IDX3),#width:offset</td>
<td></td>
<td>BFEXT Dd,(IDX1),#width:offset</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,(IDX2,REG),#width:offset</td>
<td></td>
<td>BFEXT Dd,(IDX1),#width:offset</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,(IDX3,REG),#width:offset</td>
<td></td>
<td>BFEXT Dd,(IDX1),#width:offset</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,EXT1,#width:offset</td>
<td></td>
<td>BFEXT Dd,(IDX1),#width:offset</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,EXT1,Ds,#width:offset</td>
<td></td>
<td>BFEXT Dd,(IDX1),#width:offset</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,EXT2,#width:offset</td>
<td></td>
<td>BFEXT Dd,(IDX1),#width:offset</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,EXT3,#width:offset</td>
<td></td>
<td>BFEXT Dd,(IDX1),#width:offset</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,EXT1,Ds,Dp</td>
<td></td>
<td>BFEXT Dd,(IDX1),#width:offset</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,EXT2,Ds,Dp</td>
<td></td>
<td>BFEXT Dd,(IDX1),#width:offset</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,EXT3,Ds,Dp</td>
<td></td>
<td>BFEXT Dd,(IDX1),#width:offset</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,[REG,IDX],#width:offset</td>
<td>5.5(2/0)</td>
<td>BFEXT Dd,[REG,IDX],#width:offset</td>
<td>5(1/1)</td>
</tr>
<tr>
<td>BFEXT Dd,[REG,IDX],Dp</td>
<td></td>
<td>BFEXT Dd,[REG,IDX],#width:offset</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,[IDX1],#width:offset</td>
<td>5.5(2/0)</td>
<td>BFEXT Dd,[IDX1],#width:offset</td>
<td>5(1/1)</td>
</tr>
<tr>
<td>BFEXT Dd,[IDX3],#width:offset</td>
<td></td>
<td>BFEXT Dd,[IDX1],#width:offset</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,[EXT3],#width:offset</td>
<td></td>
<td>BFEXT Dd,[IDX1],#width:offset</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,[IDX1],Dp</td>
<td></td>
<td>BFEXT Dd,[IDX1],Dp</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,[IDX3],Dp</td>
<td></td>
<td>BFEXT Dd,[IDX1],Dp</td>
<td></td>
</tr>
<tr>
<td>BFEXT Dd,[EXT3],Dp</td>
<td></td>
<td>BFEXT Dd,[IDX1],Dp</td>
<td></td>
</tr>
</tbody>
</table>
Table 8-36. Unconditional Branch Execution Timing

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRA oprdest</td>
<td>1.5(0/0)</td>
</tr>
</tbody>
</table>

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”).
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>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP EXT24</td>
<td>1.5(0/0)</td>
</tr>
<tr>
<td>JMP (IDX)</td>
<td>2.0(0/0)</td>
</tr>
<tr>
<td>JMP (+IDX)</td>
<td></td>
</tr>
<tr>
<td>JMP (REG,IDX)</td>
<td></td>
</tr>
<tr>
<td>JMP (IDX1)</td>
<td></td>
</tr>
<tr>
<td>JMP (IDX3)</td>
<td></td>
</tr>
<tr>
<td>JMP (IDX2,REG)</td>
<td>2.5(0/0)</td>
</tr>
<tr>
<td>JMP (IDX3,REG)</td>
<td></td>
</tr>
<tr>
<td>JMP EXT1</td>
<td></td>
</tr>
<tr>
<td>JMP EXT2</td>
<td></td>
</tr>
<tr>
<td>JMP EXT3</td>
<td></td>
</tr>
<tr>
<td>JMP [REG,IDX]</td>
<td>3(1/0)</td>
</tr>
<tr>
<td>JMP [IDX1]</td>
<td>3.5(1/0)</td>
</tr>
<tr>
<td>JMP [IDX3]</td>
<td></td>
</tr>
<tr>
<td>JMP [EXT3]</td>
<td></td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Operation</th>
<th>Cycles (taken)</th>
<th>Cycles (not taken)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bcc oprdest</td>
<td>1.5(0/0)</td>
<td>1(0/0)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles (taken)</th>
<th>Cycles (not taken)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;OP&gt; Di,#opr5i,oprdest</td>
<td>3(0/0)</td>
<td>2.5(0/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; REG,#opr5i,oprdest</td>
<td>3(0/0)</td>
<td>2.5(0/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; REG,Dn,oprdest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;OP&gt; bw! (IDX),#opr5i,oprdest</td>
<td>4.5(1/0)</td>
<td>4(1/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; bw! (IDX),Dn,oprdest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;OP&gt; bw! (REG,IDX),#opr5i,oprdest</td>
<td>5(1/0)</td>
<td>4.5(1/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; bw! (REG,IDX),Dn,oprdest</td>
<td>6(2/0)</td>
<td>5.5(2/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; bw! [IDX1],#opr5i,oprdest</td>
<td>6.5(2/0)</td>
<td>6(2/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; bw! [IDX1],Dn,oprdest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;OP&gt; bw! [IDX3],#opr5i,oprdest</td>
<td>7.5(2/0)</td>
<td>6(2/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; bw! [IDX3],Dn,oprdest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;OP&gt; bw! [EXT3],#opr5i,oprdest</td>
<td>8.5(2/0)</td>
<td>7(2/0)</td>
</tr>
<tr>
<td>&lt;OP&gt; bw! [EXT3],Dn,oprdest</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles (taken)</th>
<th>Cycles (not taken)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBcc Di,oprdest</td>
<td>2.5(0/0)</td>
<td>2(0/0)</td>
</tr>
<tr>
<td>DBcc xy,oprdest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBcc REG,oprdest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBcc bwpl (IDX),oprdest</td>
<td>4.5(1/1)</td>
<td>4(1/1)</td>
</tr>
<tr>
<td>DBcc bwpl (+IDX),oprdest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBcc bwpl (REG,IDX),oprdest</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 8-40. Decrement and Branch Execution Timing

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles (taken)</th>
<th>Cycles (not taken)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBcc Di, opdest</td>
<td>2.5(0/0)</td>
<td>2(0/0)</td>
</tr>
<tr>
<td>TBcc xy, opdest</td>
<td>2(0/0)</td>
<td>2(0/0)</td>
</tr>
<tr>
<td>TBcc REG, opdest</td>
<td>3(1/0)</td>
<td>2(0/0)</td>
</tr>
<tr>
<td>TBcc bwpl (IDX), opdest</td>
<td>4.5(1/0)</td>
<td>4(1/0)</td>
</tr>
<tr>
<td>TBcc bwpl (++IDX), opdest</td>
<td>4(1/0)</td>
<td>3.5(1/0)</td>
</tr>
<tr>
<td>TBcc bwpl (REG,IDX), opdest</td>
<td>4.5(1/0)</td>
<td>4(1/0)</td>
</tr>
<tr>
<td>TBcc bwpl (IDX1), opdest</td>
<td>4.5(1/0)</td>
<td>4(1/0)</td>
</tr>
<tr>
<td>TBcc bwpl (IDX3), opdest</td>
<td>4.5(1/0)</td>
<td>4(1/0)</td>
</tr>
<tr>
<td>TBcc bwpl EXT1, opdest</td>
<td>5(2/0)</td>
<td>5.5(2/0)</td>
</tr>
<tr>
<td>TBcc bwpl EXT2, opdest</td>
<td>5(2/0)</td>
<td>5.5(2/0)</td>
</tr>
<tr>
<td>TBcc bwpl EXT3, opdest</td>
<td>5(2/0)</td>
<td>5.5(2/0)</td>
</tr>
<tr>
<td>DBcc bwpl [REG,IDX], opdest</td>
<td>6(2/1)</td>
<td>5.5(2/1)</td>
</tr>
<tr>
<td>DBcc bwpl [IDX1], opdest</td>
<td>6.5(2/1)</td>
<td>6(2/1)</td>
</tr>
<tr>
<td>DBcc bwpl [IDX3], opdest</td>
<td>6.5(2/1)</td>
<td>6(2/1)</td>
</tr>
<tr>
<td>DBcc bwpl [EXT3], opdest</td>
<td>6.5(2/1)</td>
<td>6(2/1)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles (taken)</th>
<th>Cycles (not taken)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBcc bwpl (IDX1), opdest</td>
<td>5(1/1)</td>
<td>4.5(1/1)</td>
</tr>
<tr>
<td>DBcc bwpl (IDX3), opdest</td>
<td>5(1/1)</td>
<td>4.5(1/1)</td>
</tr>
<tr>
<td>DBcc bwpl (IDX2,REG), opdest</td>
<td>5(1/1)</td>
<td>4.5(1/1)</td>
</tr>
<tr>
<td>DBcc bwpl (IDX3,REG), opdest</td>
<td>5(1/1)</td>
<td>4.5(1/1)</td>
</tr>
<tr>
<td>DBcc bwpl EXT1, opdest</td>
<td>5(1/1)</td>
<td>4.5(1/1)</td>
</tr>
<tr>
<td>DBcc bwpl EXT2, opdest</td>
<td>5(1/1)</td>
<td>4.5(1/1)</td>
</tr>
<tr>
<td>DBcc bwpl EXT3, opdest</td>
<td>5(1/1)</td>
<td>4.5(1/1)</td>
</tr>
<tr>
<td>DBcc bwpl [REG,IDX], opdest</td>
<td>6(2/1)</td>
<td>5.5(2/1)</td>
</tr>
<tr>
<td>DBcc bwpl [IDX1], opdest</td>
<td>6.5(2/1)</td>
<td>6(2/1)</td>
</tr>
<tr>
<td>DBcc bwpl [IDX3], opdest</td>
<td>6.5(2/1)</td>
<td>6(2/1)</td>
</tr>
<tr>
<td>DBcc bwpl [EXT3], opdest</td>
<td>6.5(2/1)</td>
<td>6(2/1)</td>
</tr>
</tbody>
</table>

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”).
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>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSR oprdest</td>
<td>2.5(0/1)</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTS</td>
<td>3(1/0)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Machine Exception&gt;</td>
<td>4(1/0)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Hardware Interrupt&gt;</td>
<td>8(1/8)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARE</td>
<td>8(1/8)</td>
</tr>
<tr>
<td>TRAP num</td>
<td>8(1/8)</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWI</td>
<td>8(1/8)</td>
</tr>
<tr>
<td>SYS</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTI (no pending interrupt)</td>
<td>6.5(8/0)</td>
</tr>
<tr>
<td>RTI (pending interrupt)</td>
<td>8.5(9/0)</td>
</tr>
</tbody>
</table>

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”).
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

<table>
<thead>
<tr>
<th>Operation (BGND)</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGND (BDC disabled)</td>
<td>1(0/0)</td>
</tr>
<tr>
<td>BGND (BDC enabled)</td>
<td>S12Z CPU halted until BDC “Go” command is executed</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Transfer Size</th>
<th>Address Alignment</th>
<th>Number of Accesses Required</th>
<th>Data Alignment&lt;sup&gt;1&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[A1:A0]</td>
<td>[D31:D24] [D23:D16] [D15:D8] [D7:D0]</td>
<td></td>
</tr>
<tr>
<td><strong>Byte (8-bit)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>[OP7:OP0]</td>
<td>−</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>−</td>
<td>[OP7:OP0]</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td><strong>Word (16-bit)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>[OP15:OP8]</td>
<td>[OP7:OP0]</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>−</td>
<td>[OP15:OP8]</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>[OP7:OP0]</td>
<td>−</td>
</tr>
<tr>
<td><strong>Pointer (24-bit)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>[OP23:OP16]</td>
<td>[OP15:OP8]</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>−</td>
<td>[OP23:OP16]</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>[OP7:OP0]</td>
<td>−</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>[OP15:OP8]</td>
<td>[OP7:OP0]</td>
</tr>
<tr>
<td><strong>Long Word (32-bit)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>[OP31:OP24]</td>
<td>[OP23:OP16]</td>
</tr>
<tr>
<td>01</td>
<td>2</td>
<td>[OP7:OP0]</td>
<td>[OP31:OP24]</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>[OP15:OP8]</td>
<td>[OP7:OP0]</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>[OP23:OP16]</td>
<td>[OP15:OP8]</td>
</tr>
</tbody>
</table>

<sup>1</sup> 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)**

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Operation</th>
<th>Address Mode(s)</th>
<th>Machine Coding (hex)</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS Di</td>
<td>(Di) := D</td>
<td>INM</td>
<td>1B 4n</td>
<td></td>
</tr>
<tr>
<td>ADC Di,Diopl</td>
<td>(Di) + (M) + C := Di</td>
<td>IMM1, IMM2, IMM4</td>
<td>1B 5p 11, 1B 5p 12 i1, 1B 5p 14 i3 i2 i1</td>
<td></td>
</tr>
<tr>
<td>ADC Di,Dioplreg</td>
<td>M can be a memory operand or another register Di</td>
<td>OPR</td>
<td>1B 6n xb</td>
<td></td>
</tr>
<tr>
<td>AND Di,Diopl</td>
<td>(Di) &amp; (M) := Di</td>
<td>IMM1, IMM2, IMM4</td>
<td>1B 5p 11, 1B 5p 12 i1, 1B 5p 14 i3 i2 i1</td>
<td></td>
</tr>
<tr>
<td>AND Di,Dioplreg</td>
<td>M can be a memory operand or another register Di</td>
<td>OPR</td>
<td>6n xb</td>
<td></td>
</tr>
<tr>
<td>ANDCC Di,#opr8i</td>
<td>(CCL) &amp; (M) := CCL</td>
<td>IMM1</td>
<td>5p i1</td>
<td></td>
</tr>
<tr>
<td>ASL Dd,Di, Dd</td>
<td>Bitwise Shift Left Dd or memory, 0 to n positions</td>
<td>REG-REG, REG-IMM</td>
<td>in sb xb</td>
<td></td>
</tr>
<tr>
<td>ASL Dd,Ds, Dn</td>
<td>Arithmetic Shift Left Dd, where n is the number of bits in the operand</td>
<td>REG-REG, REG-IMM</td>
<td>in sb</td>
<td></td>
</tr>
<tr>
<td>ASL Dd,Dioplreg,#opr8i</td>
<td>Bitwise AND CCL with immediate byte in Memory</td>
<td>OPR</td>
<td>in sb xb</td>
<td></td>
</tr>
<tr>
<td>ASL Dd,Dioplreg,#opr5i</td>
<td>Bitwise AND CCL with immediate byte in Memory</td>
<td>OPR</td>
<td>in sb xb</td>
<td></td>
</tr>
<tr>
<td>ASL Dd,Dioplreg,#opr1i</td>
<td>Bitwise AND CCL with immediate byte in Memory</td>
<td>OPR</td>
<td>in sb xb</td>
<td></td>
</tr>
<tr>
<td>ASL Dd,Dioplreg,#opr1i</td>
<td>Bitwise AND CCL with immediate byte in Memory</td>
<td>OPR</td>
<td>in sb xb</td>
<td></td>
</tr>
<tr>
<td>ASL Dd,Dioplreg,#opr5i</td>
<td>Bitwise AND CCL with immediate byte in Memory</td>
<td>OPR</td>
<td>in sb xb</td>
<td></td>
</tr>
<tr>
<td>ASL Dd,Dioplreg,#opr1i</td>
<td>Bitwise AND CCL with immediate byte in Memory</td>
<td>OPR</td>
<td>in sb xb</td>
<td></td>
</tr>
</tbody>
</table>

**Linear S12 Core Reference Manual, Rev. 1.01**

344 Freescale Semiconductor
### Table A-1. S12Z Instruction Set Summary (Sheet 2 of 17)

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Operation</th>
<th>Address Mode(s)</th>
<th>Machine Coding (hex)</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASR Dd,DS,##</td>
<td>Arithmetic Shift Right Dd or memory, 0 to n positions where n+1 is the number of bits in the operand. The result is saved in register Dd (encoded in the opcode).</td>
<td>REG-REG</td>
<td>in sb xb</td>
<td>---...X...X...</td>
</tr>
<tr>
<td>ASR Dd,DS,##</td>
<td>If the destination is wider than the source, sign-extend to the width of the destination.</td>
<td>REG-REG</td>
<td>in sb xb</td>
<td>---...X...X...</td>
</tr>
<tr>
<td>ASR Dd,DS,##</td>
<td>If the destination is narrower than the source, shift and then truncate to the width of the destination.</td>
<td>REG-REG</td>
<td>in sb xb</td>
<td>---...X...X...</td>
</tr>
<tr>
<td>ASR Dd,OPR,OPR</td>
<td>In the case of two OPR operands, the parameter n operand is the last operand in the source form and the object code.</td>
<td>OPX-OPX</td>
<td>in sb xb</td>
<td>---...X...X...</td>
</tr>
<tr>
<td>ASR Dd,OPR,OPR</td>
<td>Shift memory location by 1 or 2 position. Source and destination are the same memory operand</td>
<td>OPX-OPX</td>
<td>in sb xb</td>
<td>---...X...X...</td>
</tr>
<tr>
<td>BCC oprdest</td>
<td>Branch if Carry Clear (if $C = 0$)</td>
<td>R7</td>
<td>24 rb</td>
<td>---...X...X...</td>
</tr>
<tr>
<td>BCLR oprmemreg</td>
<td>Clear Bit n in Memory or in $D_n$ 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 Di (semaphore)</td>
<td>REG-REG</td>
<td>in sb xb</td>
<td>---...X...X...</td>
</tr>
<tr>
<td>BCLR oprmemreg</td>
<td>Branch if Carry Set (if $C = 1$)</td>
<td>R7</td>
<td>25 rb</td>
<td>---...X...X...</td>
</tr>
<tr>
<td>BEQ oprdest</td>
<td>Branch if Equal (if $Z = 1$)</td>
<td>R7</td>
<td>27 rb</td>
<td>---...X...X...</td>
</tr>
</tbody>
</table>
### Table A-1. S12Z Instruction Set Summary (Sheet 3 of 17)

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Operation</th>
<th>Address Mode(s)</th>
<th>Machine Coding (hex)</th>
<th>Condition Codes</th>
</tr>
</thead>
</table>
| BFEXT |Extract bit field with width w and offset o from Dp or a memory operand, and store it into the low order bits of Dp or memory operand (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 Dp. Dp is an immediate operand as two 5-bit values w:o
w=0 is treated as 32
(000000 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-RM | 1B 0q bb | S X – I N Z V C | Δ A 6 |
| BFEXT |Extract bit field with width w and offset o from Dp or a memory operand (same function as BCC) | RG-RG-MM | 1B 0q bb 11 | Δ A 6 |
| BFEXT.bwpl Dp,opmemreg,Di,Dp |BFEXT.bwpl Dp,opmemreg,Di,Dp | RG-OP-RG | 1B 0q bb xb | Δ A 6 |
| BFEXT.bwpl Dp,opmemreg,Di,Ds |BFEXT.bwpl Dp,opmemreg,Di,Ds | OP-RG-RG | 1B 0q bb x3 x2 x1 | Δ A 6 |
| BFEXT.bwpl opmemreg,Di,Ds |BFEXT.bwpl opmemreg,Di,Ds | OP-RG-MM | 1B 0q bb i1 x3 x2 x1 | Δ A 6 |
| BFEXT.bwpl opmemreg,Di,Ds |BFEXT.bwpl opmemreg,Di,Ds | OP1-RG-MM | 1B 0q bb i1 x3 x2 x1 | Δ A 6 |
| BFEXT.bwpl opmemreg,Di,Ds |BFEXT.bwpl opmemreg,Di,Ds | OP2-RG-MM | 1B 0q bb i1 x3 x2 x1 | Δ A 6 |
| BFEXT.bwpl opmemreg,Di,Ds |BFEXT.bwpl opmemreg,Di,Ds | OP3-RG-MM | 1B 0q bb i1 x3 x2 x1 | Δ A 6 |
| BFNS |Insert bit field with width w from the low order bits of Ds or a memory operand into Dp 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 Dp or an immediate operand as two 5-bit values w:o w=0 is treated as 32
(000000 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 A 6 |
| BFNS |Insert bit field with width w and offset o from Dp or a memory operand into Dp 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 Dp. Dp is an immediate operand as two 5-bit values w:o
w=0 is treated as 32
(000000 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 A 6 |
| BGE |branch If Greater Than or Equal (if Z = 0) | R7 | 2C rb | Δ A 6 |
| BGE |branch If Greater Than or Equal (if Z = 0) | R15 | 2C rb r1 | Δ A 6 |
| BGD |branch If Greater Than (if Z = 0) (signed) | R7 | 2E rb | Δ A 6 |
| BGD |branch If Greater Than (if Z = 0) (signed) | R15 | 2E rb r1 | Δ A 6 |
| BGT |branch If Greater Than (if Z = 0) (unsigned) | R7 | 2F rb | Δ A 6 |
| BGT |branch If Greater Than (if Z = 0) (unsigned) | R15 | 2F rb r1 | Δ A 6 |
| BR |branch If Greater or Same (same function as BCC) (if C = 0) (unsigned) | R7 | 24 rb | Δ A 6 |
| BR |branch If Greater or Same (same function as BCC) (if C = 0) (unsigned) | R15 | 24 rb r1 | Δ A 6 |
| BLT |branch If Higher (if C = 1) (unsigned) | R7 | 22 rb | Δ A 6 |
| BLT |branch If Higher (if C = 1) (unsigned) | R15 | 22 rb r1 | Δ A 6 |
| BLS |branch If Higher or Same (same function as BCC) (if C = 0) (unsigned) | R7 | 24 rb | Δ A 6 |
| BLS |branch If Higher or Same (same function as BCC) (if C = 0) (unsigned) | R15 | 24 rb r1 | Δ A 6 |
| BLT |branch If Greater or Same (same function as BCC) (if C = 0) (unsigned) | R7 | 23 rb | Δ A 6 |
| BLT |branch If Greater or Same (same function as BCC) (if C = 0) (unsigned) | R15 | 23 rb r1 | Δ A 6 |
| BMI |branch If Minus (if N = 1) | R7 | 2B rb | Δ A 6 |
| BMI |branch If Minus (if N = 1) | R15 | 2B rb r1 | Δ A 6 |
| BNE |branch If Not Equal (if Z = 0) | R7 | 26 rb | Δ A 6 |
| BNE |branch If Not Equal (if Z = 0) | R15 | 26 rb r1 | Δ A 6 |
### Linear S12Z Instruction Set Summary (Sheet 4 of 17)

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Operation</th>
<th>Address Mode(s)</th>
<th>Machine Coding (hex)</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPL oprdest</td>
<td>Branch if Plus (if N = 0)</td>
<td>R7</td>
<td>A rb</td>
<td>---</td>
</tr>
<tr>
<td>BRA oprdest</td>
<td>Branch Always (if 1 = 1)</td>
<td>R7</td>
<td>0 rb</td>
<td>---</td>
</tr>
<tr>
<td>BRCR.D oprreg.opr5,opr6</td>
<td>Branch if (M) &amp; bitn = 0 or if (D) &amp; bitn = 0</td>
<td>REG-IMM-R7</td>
<td>02 bm rb</td>
<td>---</td>
</tr>
<tr>
<td>BRCR.D,opr5,opr6</td>
<td>Test Bit n in Memory or in D and branch if clear n is specified in an immediate value or D n is encoded in the postbyte (sb) bitn is a mask with only bit n set Branch offset is 7 bits or 15 bits</td>
<td>REG-IMM-R15</td>
<td>02 bm rb</td>
<td>---</td>
</tr>
<tr>
<td>BRCR.D,opr5,opr6</td>
<td>Branch if (M) &amp; bitn = 0 or if (D) &amp; bitn = 0</td>
<td>OP-IMM-R7</td>
<td>02 bm sd rb r1</td>
<td>---</td>
</tr>
<tr>
<td>BRCR.D,opr5,opr6</td>
<td>Test Bit n in Memory or in D and branch if clear n is specified in an immediate value or D n is encoded in the postbyte (sb) bitn is a mask with only bit n set Branch offset is 7 bits or 15 bits</td>
<td>OP-IMM-R15</td>
<td>02 bm sd x1 rb</td>
<td>---</td>
</tr>
<tr>
<td>BRCR.D,opr5,opr6</td>
<td>Branch if (M) &amp; bitn = 0 or if (D) &amp; bitn = 0</td>
<td>OP1-IMM-R7</td>
<td>02 bm sb x1 rb r1</td>
<td>---</td>
</tr>
<tr>
<td>BRCR.D,opr5,opr6</td>
<td>Test Bit n in Memory or in D and branch if clear n is specified in an immediate value or D n is encoded in the postbyte (sb) bitn is a mask with only bit n set Branch offset is 7 bits or 15 bits</td>
<td>OP1-IMM-R15</td>
<td>02 bm sb x1 rb</td>
<td>---</td>
</tr>
<tr>
<td>BRCR.D,opr5,opr6</td>
<td>Branch if (M) &amp; bitn = 0 or if (D) &amp; bitn = 0</td>
<td>OP2-IMM-R7</td>
<td>02 bm sb x1 rb r1</td>
<td>---</td>
</tr>
<tr>
<td>BRCR.D,opr5,opr6</td>
<td>Test Bit n in Memory or in D and branch if clear n is specified in an immediate value or D n is encoded in the postbyte (sb) bitn is a mask with only bit n set Branch offset is 7 bits or 15 bits</td>
<td>OP2-IMM-R15</td>
<td>02 bm sb x1 rb</td>
<td>---</td>
</tr>
<tr>
<td>BRCR.D,opr5,opr6</td>
<td>Branch if (M) &amp; bitn = 0 or if (D) &amp; bitn = 0</td>
<td>OP3-IMM-R7</td>
<td>02 bm sb x1 x1 rb r1</td>
<td>---</td>
</tr>
<tr>
<td>BRCR.D,opr5,opr6</td>
<td>Test Bit n in Memory or in D and branch if clear n is specified in an immediate value or D n is encoded in the postbyte (sb) bitn is a mask with only bit n set Branch offset is 7 bits or 15 bits</td>
<td>OP3-IMM-R15</td>
<td>02 bm sb x1 x1 rb</td>
<td>---</td>
</tr>
<tr>
<td>BSR oprdest</td>
<td>Branch if Overflow Bit Clear (if V = 0)</td>
<td>R7</td>
<td>0 rb</td>
<td>---</td>
</tr>
<tr>
<td>BTG oprreg</td>
<td>Branch if (M) &amp; bitn = 0 or if (D) &amp; bitn = 0</td>
<td>REG-IMM-R7</td>
<td>02 bm rb</td>
<td>---</td>
</tr>
<tr>
<td>BTG oprreg</td>
<td>Branch if (M) &amp; bitn = 0 or if (D) &amp; bitn = 0</td>
<td>REG-IMM-R15</td>
<td>02 bm rb</td>
<td>---</td>
</tr>
<tr>
<td>BTG oprreg</td>
<td>Toggle Bit n in Memory or in D where n is the number of the bit to be changed n is specified in an immediate value or D n is encoded in the postbyte (sb) bitn is a mask with only bit n set Branch offset is 7 bits or 15 bits</td>
<td>OP-IMM-R7</td>
<td>02 bm sb x1 rb r1</td>
<td>---</td>
</tr>
<tr>
<td>BTG oprreg</td>
<td>Toggle Bit n in Memory or in D where n is the number of the bit to be changed n is specified in an immediate value or D n is encoded in the postbyte (sb) bitn is a mask with only bit n set Branch offset is 7 bits or 15 bits</td>
<td>OP-IMM-R15</td>
<td>02 bm sb x1 rb</td>
<td>---</td>
</tr>
<tr>
<td>BTG oprreg</td>
<td>Toggle Bit n in Memory or in D where n is the number of the bit to be changed n is specified in an immediate value or D n is encoded in the postbyte (sb) bitn is a mask with only bit n set Branch offset is 7 bits or 15 bits</td>
<td>OP1-IMM-R7</td>
<td>02 bm sb x1 x1 rb r1</td>
<td>---</td>
</tr>
<tr>
<td>BTG oprreg</td>
<td>Toggle Bit n in Memory or in D where n is the number of the bit to be changed n is specified in an immediate value or D n is encoded in the postbyte (sb) bitn is a mask with only bit n set Branch offset is 7 bits or 15 bits</td>
<td>OP1-IMM-R15</td>
<td>02 bm sb x1 x1 rb</td>
<td>---</td>
</tr>
<tr>
<td>BTG oprreg</td>
<td>Toggle Bit n in Memory or in D where n is the number of the bit to be changed n is specified in an immediate value or D n is encoded in the postbyte (sb) bitn is a mask with only bit n set Branch offset is 7 bits or 15 bits</td>
<td>OP2-IMM-R7</td>
<td>02 bm sb x1 x1 x1 rb r1</td>
<td>---</td>
</tr>
<tr>
<td>BTG oprreg</td>
<td>Toggle Bit n in Memory or in D where n is the number of the bit to be changed n is specified in an immediate value or D n is encoded in the postbyte (sb) bitn is a mask with only bit n set Branch offset is 7 bits or 15 bits</td>
<td>OP2-IMM-R15</td>
<td>02 bm sb x1 x1 x1 rb</td>
<td>---</td>
</tr>
<tr>
<td>BTG oprreg</td>
<td>Toggle Bit n in Memory or in D where n is the number of the bit to be changed n is specified in an immediate value or D n is encoded in the postbyte (sb) bitn is a mask with only bit n set Branch offset is 7 bits or 15 bits</td>
<td>OP3-IMM-R7</td>
<td>02 bm sb x1 x1 x1 x1 rb r1</td>
<td>---</td>
</tr>
<tr>
<td>BTG oprreg</td>
<td>Toggle Bit n in Memory or in D where n is the number of the bit to be changed n is specified in an immediate value or D n is encoded in the postbyte (sb) bitn is a mask with only bit n set Branch offset is 7 bits or 15 bits</td>
<td>OP3-IMM-R15</td>
<td>02 bm sb x1 x1 x1 x1 rb</td>
<td>---</td>
</tr>
</tbody>
</table>

---

**APPENDIX A INSTRUCTION REFERENCE**

Freescale Semiconductor 347
Table A-1. S12Z Instruction Set Summary (Sheet 5 of 17)

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Operation</th>
<th>Address Mode(s)</th>
<th>Machine Coding (hex)</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BVS oprdest</td>
<td>Branch if Overflow Bit Set (if V = 1)</td>
<td>R7</td>
<td>09 rb</td>
<td>- - - - - -</td>
</tr>
<tr>
<td>CLS oprreg,oprreg</td>
<td>count leading sign bits</td>
<td>REG-REG</td>
<td>1B 91 cb</td>
<td>- - - 0 A 0 -</td>
</tr>
<tr>
<td>CLC</td>
<td>0  =&gt; C</td>
<td>IMM1</td>
<td>CE FE</td>
<td>- - - - - -</td>
</tr>
<tr>
<td>CLI</td>
<td>0  =&gt; l (l can only be changed in supervisor state)</td>
<td>IMM1</td>
<td>CE EP</td>
<td>- - - - - -</td>
</tr>
<tr>
<td>CLR D7</td>
<td>0  =&gt; D7, Clear data register D7</td>
<td>INH</td>
<td>3q</td>
<td>- - - 0 1 0 0</td>
</tr>
<tr>
<td>CLR,any opregreg</td>
<td>0  =&gt; M Clear Memory operand M</td>
<td>OPR</td>
<td>Bp xb</td>
<td>- - - 0 0 0 0</td>
</tr>
<tr>
<td>CLV</td>
<td>0  =&gt; Y Clear index register Y</td>
<td>INH</td>
<td>1B 02 xb x1</td>
<td>- - - 0 0 0 0</td>
</tr>
<tr>
<td>CMP Di#,opregreg</td>
<td>(Di) – (M)</td>
<td>OPR</td>
<td>Bp xb x1</td>
<td>- - - 0 0 0 0</td>
</tr>
<tr>
<td>CMP D7,opregreg</td>
<td>M can be a memory operand or another register D7</td>
<td>OPR</td>
<td>Bp xb x1 x2 x1</td>
<td>- - - 0 0 0 0</td>
</tr>
<tr>
<td>CMP xy#,opregreg</td>
<td>(xy) – (M+1:M+2)</td>
<td>OPR</td>
<td>Bp x2 x1 x2 x1</td>
<td>- - - 0 0 0 0</td>
</tr>
<tr>
<td>CMP S,#opr24i</td>
<td>(SP) – (M:M+1:M+2)</td>
<td>OPR</td>
<td>Bp x3 x2 x1 x2 x1</td>
<td>- - - 0 0 0 0</td>
</tr>
<tr>
<td>CMP X,Y</td>
<td>(X) – (Y)</td>
<td>OPR</td>
<td>Bp x4 x3 x2 x1 x2 x1</td>
<td>- - - 0 0 0 0</td>
</tr>
<tr>
<td>COM,any opregreg</td>
<td>-(M) =&gt; M equivalent to SF; (M) =&gt; M</td>
<td>OPR</td>
<td>Bp x5 x4 x3 x2 x1 x2 x1</td>
<td>- - - 0 0 0 0</td>
</tr>
<tr>
<td>DBc,Di,opregdest</td>
<td>(Di) – 1  =&gt; Di Decrement and branch</td>
<td>REG-R7</td>
<td>0B lb rb</td>
<td>- - - - - -</td>
</tr>
<tr>
<td>DBc,xy,opregdest</td>
<td>(X) – 1  =&gt; X or (Y) – 1  =&gt; Y Decrement and branch</td>
<td>REG-R7</td>
<td>0B lb rb r1</td>
<td>- - - - - -</td>
</tr>
<tr>
<td>DBc,any opregreg,opregdest</td>
<td>(M) – 1  =&gt; M Decrement M, X, Y, or memory operand M, and branch if condition cc is true.</td>
<td>OPR</td>
<td>0B lb x2 x1 x2 x1 x2 r1</td>
<td>- - - - - -</td>
</tr>
<tr>
<td>DEC D7</td>
<td>0  =&gt; D7, Decrement data register D7</td>
<td>INH</td>
<td>4n</td>
<td>- - - 0 0 0 0</td>
</tr>
<tr>
<td>DEC,any opregreg</td>
<td>(M) – 1  =&gt; M Decrement Memory operand M can be 1, 2, or 4 bytes</td>
<td>OPR</td>
<td>Ap x1 x2 x1 x2 x1 x2</td>
<td>- - - 0 0 0 0</td>
</tr>
</tbody>
</table>
### Table A-1. S12Z Instruction Set Summary (Sheet 6 of 17)

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Operation</th>
<th>Address Mode(s)</th>
<th>Machine Coding (hex)</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIVS DLD0,AN</td>
<td>(D3) (M) → D3 signed Divide result is always a register D3</td>
<td>REG-REG</td>
<td>1B 3n mb</td>
<td>– – – – ± ± ± ±</td>
</tr>
<tr>
<td>DIVS DLD0,#opr8</td>
<td>(D3) (M) → D3 Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-IMM1</td>
<td>1B 3n mb i1</td>
<td>– – – – ± ± ± ±</td>
</tr>
<tr>
<td>DIVS DLD0,#opr16</td>
<td>(D3) (M) → D3 Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-IMM2</td>
<td>1B 3n mb i2 i1</td>
<td>– – – – ± ± ± ±</td>
</tr>
<tr>
<td>DIVS DLD0,#opr32</td>
<td>(D3) (M) → D3 Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-IMM4</td>
<td>1B 3n mb i4 i3 i2 i1</td>
<td>– – – – ± ± ± ±</td>
</tr>
<tr>
<td>DIVS.Orm DL/Z,oprmemreg</td>
<td>(D3) (M) → D3 Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-OPR</td>
<td>1B 3n mb xb</td>
<td>– – – – ± ± ± ±</td>
</tr>
<tr>
<td>DIVS.Orm DL/Z,oprmemreg</td>
<td>(D3) (M) → D3 Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-OPR1</td>
<td>1B 3n mb xb x1</td>
<td>– – – – ± ± ± ±</td>
</tr>
<tr>
<td>DIVS.Orm DL/Z,oprmemreg</td>
<td>(D3) (M) → D3 Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-OPR2</td>
<td>1B 3n mb xb x1 x1</td>
<td>– – – – ± ± ± ±</td>
</tr>
<tr>
<td>DIVS.Orm DL/Z,oprmemreg</td>
<td>(D3) (M) → D3 Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-OPR3</td>
<td>1B 3n mb xb x1 x2 x1</td>
<td>– – – – ± ± ± ±</td>
</tr>
<tr>
<td>EOR DL/#oprmmsz</td>
<td>(D3) (M) → D3 Exclusive OR D3 with Memory operand M is the same size as D3</td>
<td>IMM1</td>
<td>1B 7p i1</td>
<td>– – – – ± ± ± ±</td>
</tr>
<tr>
<td>EOR DL/oprmemreg</td>
<td>M can be a memory operand or another register Dj if Dj wider, OR D3 with low portion of Dj if Dj narrower, OR D3 with zero-extended Dj</td>
<td>IMM2</td>
<td>1B 7p i2 i1</td>
<td>– – – – ± ± ± ±</td>
</tr>
<tr>
<td>EOR DL/oprmemreg</td>
<td>M can be a memory operand or another register Dj if Dj wider, OR D3 with low portion of Dj if Dj narrower, OR D3 with zero-extended Dj</td>
<td>IMM4</td>
<td>1B 7p i4 i3 i2 i1</td>
<td>– – – – ± ± ± ±</td>
</tr>
<tr>
<td>EXG cpureg,cpuavg</td>
<td>(D1) → (D2) Exchange contents of CPU Registers D0–DT, 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 CCL act as NOP.</td>
<td>REG-REG</td>
<td>AE 0b</td>
<td>– – – – ± ± ± ±</td>
</tr>
</tbody>
</table>
### Table A-1. S12Z Instruction Set Summary (Sheet 7 of 17)

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Operation</th>
<th>Address Mode(s)</th>
<th>Machine Coding (hex)</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>INC Di</td>
<td>(Di) + 1 → Di</td>
<td>INH</td>
<td>R</td>
<td>S X – Il N Z V C</td>
</tr>
<tr>
<td>INC Di,op</td>
<td>(Di) + 1 → M</td>
<td>CMPR</td>
<td>R</td>
<td>S X – Il N Z V C</td>
</tr>
<tr>
<td>JMP opr24a</td>
<td>Effective Address → PC</td>
<td>EXT24</td>
<td>R</td>
<td>S X – Il N Z V C</td>
</tr>
<tr>
<td>ORR opr24a</td>
<td>(SP) – 3 → SP;</td>
<td>EXT24</td>
<td>R</td>
<td>S X – Il N Z V C</td>
</tr>
<tr>
<td>LD Di,opr16z</td>
<td>Load Di</td>
<td>IMM1</td>
<td>R</td>
<td>S X – Il N Z V C</td>
</tr>
<tr>
<td>LD Di,opr24a</td>
<td>Load Di</td>
<td>IMM2</td>
<td>R</td>
<td>S X – Il N Z V C</td>
</tr>
<tr>
<td>LEA D6,op</td>
<td>Effective Address → D6</td>
<td>IMM1</td>
<td>R</td>
<td>S X – Il N Z V C</td>
</tr>
<tr>
<td>LEA D7,op</td>
<td>Effective Address → D7</td>
<td>IMM1</td>
<td>R</td>
<td>S X – Il N Z V C</td>
</tr>
<tr>
<td>LEA S,op</td>
<td>Effective Address → SP</td>
<td>IMM1</td>
<td>R</td>
<td>S X – Il N Z V C</td>
</tr>
<tr>
<td>LEA X,op</td>
<td>Effective Address → X</td>
<td>IMM1</td>
<td>R</td>
<td>S X – Il N Z V C</td>
</tr>
<tr>
<td>LEA Y,op</td>
<td>Effective Address → Y</td>
<td>IMM1</td>
<td>R</td>
<td>S X – Il N Z V C</td>
</tr>
<tr>
<td>LEA S,(opr8i,S)</td>
<td>(SP) + sign-extend (M) → SP</td>
<td>IMM1</td>
<td>R</td>
<td>S X – Il N Z V C</td>
</tr>
<tr>
<td>LEA X,(opr8i,X)</td>
<td>(X) + sign-extend (M) → X</td>
<td>IMM1</td>
<td>R</td>
<td>S X – Il N Z V C</td>
</tr>
<tr>
<td>LEA Y,(opr8i,Y)</td>
<td>(Y) + sign-extend (M) → Y</td>
<td>IMM1</td>
<td>R</td>
<td>S X – Il N Z V C</td>
</tr>
</tbody>
</table>
Logical Shift Left Dd or memory, 0 to n positions
The result is saved in register Dd (encoded in the opcode).

n is specified in postbyte sb, xb, a byte-sized memory
operand, or register Dn

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.

Logical Shift Right Dd or memory, 0 to n positions
The result is saved in register Dd (encoded in the opcode).

n is specified in postbyte sb, xb, a byte-sized memory
operand, or register Dn

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.

Shift memory location by 1 or 2 position.
Source and destination are the same memory operand

Shift memory location by 1 or 2 position.
Source and destination are the same memory operand

Table A-1. S12Z Instruction Set Summary (Sheet 8 of 17)
### Table A-1. S12Z Instruction Set Summary (Sheet 9 of 17)

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Operation</th>
<th>Address Mode(s)</th>
<th>Machine Coding (hex)</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACS D,Di,Dx</td>
<td>((D) \quad (D) + D_{ij} \Rightarrow D_j) signed multiply and accumulate result is always a register (D_j)</td>
<td>REG-REG</td>
<td>1B 4q mb</td>
<td><strong>A</strong> <strong>A</strong> <strong>A</strong> <strong>A</strong></td>
</tr>
<tr>
<td>MACS D,Dx,#opr/16</td>
<td>((D) \quad (M) + Q \Rightarrow D_j) Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-IMM</td>
<td>1B 4q mb i1</td>
<td><strong>N</strong> <strong>Z</strong> <strong>V</strong> <strong>C</strong></td>
</tr>
<tr>
<td>MACS D1,Dx,#opr/32</td>
<td>((D) \quad (M) + D_{ij} \Rightarrow D_j) Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-OPR</td>
<td>1B 4q mb xb</td>
<td><strong>A</strong> <strong>A</strong> <strong>A</strong> <strong>A</strong></td>
</tr>
<tr>
<td>MACS D1,Dx,#oprmemreg</td>
<td>((D1) \quad (M) + Q \Rightarrow D_j) Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-OPR1-OPR2</td>
<td>1B 4q mb xdi x1</td>
<td><strong>A</strong> <strong>A</strong> <strong>A</strong> <strong>A</strong></td>
</tr>
<tr>
<td>MACS D1,Dx,#oprmemreg</td>
<td>((M1) \quad (M2) + Q \Rightarrow D_j) Memory operands M1 and M2 can be 8, 16, 24, or 32 bits 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</td>
<td>OPR</td>
<td>1B 4q mb xb</td>
<td><strong>A</strong> <strong>A</strong> <strong>A</strong> <strong>A</strong></td>
</tr>
<tr>
<td>MACS D1,Dx,#oprmemreg</td>
<td>((D1) \quad (M) + Q \Rightarrow D_j) Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-OPR</td>
<td>1B 4q mb xb</td>
<td><strong>A</strong> <strong>A</strong> <strong>A</strong> <strong>A</strong></td>
</tr>
<tr>
<td>MACS D1,Dx,#oprmemreg</td>
<td>((M1) \quad (M2) + Q \Rightarrow D_j) Memory operands M1 and M2 can be 8, 16, 24, or 32 bits 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</td>
<td>OPR-OPR</td>
<td>1B 4q mb xb</td>
<td><strong>A</strong> <strong>A</strong> <strong>A</strong> <strong>A</strong></td>
</tr>
<tr>
<td>MACS D,Dx</td>
<td>((D) \quad (D) + D_{ij} \Rightarrow D_j) unsigned multiply and accumulate result is always a register (D_j)</td>
<td>REG-REG</td>
<td>1B 4q mb</td>
<td><strong>A</strong> <strong>A</strong> <strong>A</strong> <strong>A</strong></td>
</tr>
<tr>
<td>MACU D1,Dx,#opr16</td>
<td>((D) \quad (M) + Q \Rightarrow D_j) Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-IMM</td>
<td>1B 4q mb i1</td>
<td><strong>N</strong> <strong>Z</strong> <strong>V</strong> <strong>C</strong></td>
</tr>
<tr>
<td>MACU D1,Dx,#opr32</td>
<td>((D) \quad (M) + D_{ij} \Rightarrow D_j) Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-OPR</td>
<td>1B 4q mb xdi x1</td>
<td><strong>A</strong> <strong>A</strong> <strong>A</strong> <strong>A</strong></td>
</tr>
<tr>
<td>MACU b1w D1,Dx,#oprmemreg</td>
<td>((D) \quad (M) + Q \Rightarrow D_j) Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-OPR</td>
<td>1B 4q mb xdi x1</td>
<td><strong>A</strong> <strong>A</strong> <strong>A</strong> <strong>A</strong></td>
</tr>
<tr>
<td>MACU b2w D1,Dx,#oprmemreg</td>
<td>((M1) \quad (M2) + Q \Rightarrow D_j) Memory operands M1 and M2 can be 8, 16, 24, or 32 bits 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</td>
<td>OPR-OPR</td>
<td>1B 4q mb xdi x1</td>
<td><strong>A</strong> <strong>A</strong> <strong>A</strong> <strong>A</strong></td>
</tr>
<tr>
<td>MAXS D,#oprmemreg</td>
<td>MANI((D)<em>{ij}, (M)</em>{ij} \Rightarrow D_{ij}) Memory operand M is the same size as (D_{ij})</td>
<td>OPR</td>
<td>1B 2q xdi</td>
<td><strong>A</strong> <strong>A</strong> <strong>A</strong> <strong>A</strong></td>
</tr>
<tr>
<td>MAXU D,#oprmemreg</td>
<td>MAXU((D)<em>{ij}, (M)</em>{ij} \Rightarrow D_{ij}) Memory operand M is the same size as (D_{ij})</td>
<td>OPR</td>
<td>1B 2q xdi</td>
<td><strong>A</strong> <strong>A</strong> <strong>A</strong> <strong>A</strong></td>
</tr>
<tr>
<td>MINS D,#oprmemreg</td>
<td>MINS((D)<em>{ij}, (M)</em>{ij} \Rightarrow D_{ij}) Memory operand M is the same size as (D_{ij})</td>
<td>OPR</td>
<td>1B 2q xdi</td>
<td><strong>A</strong> <strong>A</strong> <strong>A</strong> <strong>A</strong></td>
</tr>
</tbody>
</table>

**Linear S12 Core Reference Manual, Rev. 1.01**

352 Freescale Semiconductor
<table>
<thead>
<tr>
<th>Source Form</th>
<th>Operation</th>
<th>Address Mode(s)</th>
<th>Machine Coding (hex)</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINU Ds,oprmemreg</td>
<td>MIN:Di = Di. Minimum of two unsigned operands replaces Di. Memory operand M is the same size as Di.</td>
<td>OPR</td>
<td>1B 1n xb</td>
<td>- - - - A A A</td>
</tr>
<tr>
<td>MODS Ds,Dj,LA</td>
<td>(Di) (Lj): remainder = Dj. signed modulo operation result is always a register Dj.</td>
<td>REG-REG</td>
<td>1B 3j mb</td>
<td>- - - - A A A</td>
</tr>
<tr>
<td>MODS Ds,Dj,#opr18</td>
<td>(Di) (Mj): remainder = Dj. Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-MM1, REG-MM2, REG-MM4</td>
<td>1B 3j mb i1 11, 1B 3j mb i2 11, 1B 3q mb i4 11 i2 i1</td>
<td></td>
</tr>
<tr>
<td>MODS Ds,Dj,#opr16i</td>
<td>(Di) (Mj): remainder = Dj. Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-OPR1, REG-OPR2, REG-OPR3</td>
<td>1B 3q mb xb x1, 1B 3q mb xb x2 x1, 1B 3q mb xb x3 x2 x1</td>
<td></td>
</tr>
<tr>
<td>MODS lw Ds,Dj,oprmemreg</td>
<td>(Di) (Mj): remainder = Dj. Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-OPR1, REG-OPR2, REG-OPR3</td>
<td>1B 3q mb xb x1, 1B 3q mb xb x2 x1, 1B 3q mb xb x3 x2 x1</td>
<td></td>
</tr>
<tr>
<td>MODS.lwbp Ds,Dj,oprmemreg,oprmemreg</td>
<td>(Di) (Mj): remainder = Dj. Memory operands M1 and M2 can be 8, 16, 24, or 32 bits.</td>
<td>REG-OPR1, REG-OPR2, REG-OPR3</td>
<td>1B 3q mb xb x1, 1B 3q mb xb x2 x1, 1B 3q mb xb x3 x2 x1</td>
<td></td>
</tr>
<tr>
<td>MODS lw Ds,Dj,LA</td>
<td>(Di) (Lj): remainder = Dj. unsigned modulo operation result is always a register Dj.</td>
<td>REG-REG</td>
<td>1B 3j mb</td>
<td>- - - - A A A</td>
</tr>
<tr>
<td>MODS lwbp Ds,Dj,oprmemreg,oprmemreg</td>
<td>(Di) (Mj): remainder = Dj. Memory operands M1 and M2 can be 8, 16, 24, or 32 bits.</td>
<td>REG-OPR1, REG-OPR2, REG-OPR3</td>
<td>1B 3q mb xb x1, 1B 3q mb xb x2 x1, 1B 3q mb xb x3 x2 x1</td>
<td></td>
</tr>
<tr>
<td>MODS lwbp Ds,Dj,oprmemreg</td>
<td>(Di) (Mj): remainder = Dj. Memory operands M1 and M2 can be 8, 16, 24, or 32 bits.</td>
<td>REG-OPR1, REG-OPR2, REG-OPR3</td>
<td>1B 3q mb xb x1, 1B 3q mb xb x2 x1, 1B 3q mb xb x3 x2 x1</td>
<td></td>
</tr>
<tr>
<td>MODS lw Ds,Dj,LA</td>
<td>(Di) (Lj): remainder = Dj. unsigned modulo operation result is always a register Dj.</td>
<td>REG-REG</td>
<td>1B 3j mb</td>
<td>- - - - A A A</td>
</tr>
<tr>
<td>MODS lwbp Ds,Dj,oprmemreg</td>
<td>(Di) (Mj): remainder = Dj. Memory operands M1 and M2 can be 8, 16, 24, or 32 bits.</td>
<td>REG-OPR1, REG-OPR2, REG-OPR3</td>
<td>1B 3q mb xb x1, 1B 3q mb xb x2 x1, 1B 3q mb xb x3 x2 x1</td>
<td></td>
</tr>
</tbody>
</table>

Table A-1. S12Z Instruction Set Summary (Sheet 10 of 17)
### Table A-1. S12Z Instruction Set Summary (Sheet 11 of 17)

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Operation</th>
<th>Address Mode(s)</th>
<th>Machine Coding (hex)</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV.B #opr8i,oprmemreg</td>
<td>Move Immediate to Memory MD, 8-bit operands</td>
<td>IMM7-OPR</td>
<td>0 C i1 xb</td>
<td>SX IN ZV C</td>
</tr>
<tr>
<td>MOV.B oprmemreg,oprmemreg</td>
<td>Memory to memory, 8-bit operands</td>
<td>OPR-OPR, OPR1-OPR, OPR2-OPR, OPR3-OPR</td>
<td>OF i4 i3 i2 i1 xb</td>
<td>SX IN ZV C</td>
</tr>
<tr>
<td>MOV.L #opr32i,oprmemreg</td>
<td>Move Immediate to Memory MD, 32-bit operands</td>
<td>IMM4-OPR</td>
<td>0 F i4 i3 i2 i1 xb</td>
<td>SX IN ZV C</td>
</tr>
<tr>
<td>MOV.L oprmemreg,oprmemreg</td>
<td>Memory to memory, 32-bit operands</td>
<td>OPR-OPR, OPR1-OPR, OPR2-OPR, OPR3-OPR</td>
<td>OF i4 i3 i2 i1 xb x2 x1</td>
<td>SX IN ZV C</td>
</tr>
<tr>
<td>MOV.P #opr24i,oprmemreg</td>
<td>Move Immediate to Memory MD, 24-bit operands</td>
<td>IMM3-OPR</td>
<td>0 E i3 i2 i1 xb</td>
<td>SX IN ZV C</td>
</tr>
<tr>
<td>MOV.P oprmemreg,oprmemreg</td>
<td>Memory to memory, 24-bit operands</td>
<td>OPR-OPR, OPR1-OPR, OPR2-OPR, OPR3-OPR</td>
<td>OF i3 i2 i1 xb x2 x1</td>
<td>SX IN ZV C</td>
</tr>
<tr>
<td>Source Form</td>
<td>Operation</td>
<td>Address Mode(s)</td>
<td>Machine Coding (hex)</td>
<td>Condition Codes</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------------------------------------------</td>
<td>----------------------------------</td>
<td>----------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>MOV.W #opr16i,oprmemreg</td>
<td># → MD Move Immediate to Memory MD, 16-bit operands suggest load or transfer for register operands</td>
<td>IMM2-OPR1 0D i2 i1 xb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV.W oprmemreg,oprmemreg</td>
<td>(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</td>
<td>IMM2-OPR1 0D i2 i1 xb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULS Dd,Dj,Dk</td>
<td>(Dj) (Dk) → Dd signed multiply result is always a register Dd</td>
<td>REG-REG 4q mb</td>
<td></td>
<td>−−−−−ΔΔΔΑ0</td>
</tr>
<tr>
<td>MULS Dd,Dj,#opr8i</td>
<td>Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-IMM1 4q mb i1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULS Dd,Dj,#opr16i</td>
<td>Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-IMM2 4q mb i2 i1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULS Dd,Dj,#opr32i</td>
<td>Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-IMM4 4q mb i4 i3 i2 i1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULS.D(j),(M)</td>
<td>(Dj) (M) → Dd Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-OPR 4q mb xb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULS.bwl.D(j),oprmemreg</td>
<td>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 although memory operands could be registers, the register and register-memory versions are more efficient</td>
<td>OPR-OPR 4q mb xb x1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULS.bwpawp.D(j),oprmemreg</td>
<td>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</td>
<td>OPR1-OPR 4q mb x2 x1 x1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULS.bwpawp.D(j),oprmemreg</td>
<td>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</td>
<td>OPR2-OPR 4q mb x2 x1 x2 x1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULS.bwpawp.D(j),oprmemreg</td>
<td>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</td>
<td>OPR3-OPR 4q mb x2 x1 x3 x2 x1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table A-1. S12Z Instruction Set Summary (Sheet 13 of 17)

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Operation</th>
<th>Address Mode(s)</th>
<th>Machine Coding (hex)</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULU Ds(Dx,Op)</td>
<td>unsigned multiply</td>
<td>REG-REG</td>
<td>4q mb</td>
<td>$- - - - \Delta \Delta \Delta \Delta$</td>
</tr>
<tr>
<td>MULU Ds(Dx,Op)</td>
<td>Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-REG</td>
<td>4q mb i1</td>
<td>$- - - - \Delta \Delta \Delta \Delta$</td>
</tr>
<tr>
<td>MULU Ds(Dx,Op)</td>
<td>Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-REG</td>
<td>4q mb x2 x1</td>
<td>$- - - - \Delta \Delta \Delta \Delta$</td>
</tr>
<tr>
<td>MULU Ds(Dx,Op)</td>
<td>Memory operand M can be 1, 2, or 4 bytes</td>
<td>REG-REG</td>
<td>4q mb x3 x2 x1</td>
<td>$- - - - \Delta \Delta \Delta \Delta$</td>
</tr>
</tbody>
</table>

### Linear S12 Core Reference Manual, Rev. 1.01

356 Freescale Semiconductor
Table A-1. S12Z Instruction Set Summary (Sheet 14 of 17)

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Operation</th>
<th>Address Mode(s)</th>
<th>Machine Coding (hex)</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>QMULS Dd,Dj,DI</td>
<td>(Dj) = Dd signed fractional multiply result is always a register Dd</td>
<td>REG-REG</td>
<td>1B Bn mb</td>
<td>- - - - A A X 0</td>
</tr>
<tr>
<td>QMULS Dd,Dj,DI,apr8</td>
<td>(Dj) = Dd Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-MM1</td>
<td>1B Bn mb 11</td>
<td>- - - - A A 0 0</td>
</tr>
<tr>
<td>QMULS Dd,Dj,DI,apr16</td>
<td>(Dj) = Dd Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-MM2</td>
<td>1B Bn mb i2 i1</td>
<td>- - - - A A 0 0</td>
</tr>
<tr>
<td>QMULS Dd,Dj,DI,apr32</td>
<td>(Dj) = Dd Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-MM4</td>
<td>1B Bn mb i4 i3 i2 i1</td>
<td>- - - - A A 0 0</td>
</tr>
<tr>
<td>QMULS QMULS Dd,Dj,oprmemreg</td>
<td>(Dj) = Dd Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-OPR</td>
<td>1B Bn mb xb</td>
<td>- - - - A A X 0</td>
</tr>
<tr>
<td>QMULS QMULS Dd,Dj,oprmemreg.oprmemreg</td>
<td>(Dj) = Dd Memory operands M1 and M2 can be 8, 16, 24, or 32 bits</td>
<td>REG-OPR1</td>
<td>1B Bn mb xb xb</td>
<td>- - - - A A X 0</td>
</tr>
<tr>
<td>QMULS QMULS Dd,Dj,oprmemreg.oprmemreg</td>
<td>(Dj) = Dd Memory operands M1 and M2 can be 8, 16, 24, or 32 bits</td>
<td>REG-OPR2</td>
<td>1B Bn mb xb xb xb x1</td>
<td>- - - - A A X 0</td>
</tr>
<tr>
<td>QMULS QMULS Dd,Dj,oprmemreg.oprmemreg</td>
<td>(Dj) = Dd Memory operands M1 and M2 can be 8, 16, 24, or 32 bits</td>
<td>REG-OPR3</td>
<td>1B Bn mb xb xb xb x2 x1</td>
<td>- - - - A A X 0</td>
</tr>
<tr>
<td>QMULS QMULU Dd,Dj,DI</td>
<td>(Dj) = Dd unsigned fractional multiply result is always a register Dd</td>
<td>REG-REG</td>
<td>1B Bn mb</td>
<td>- - - - A A X 0</td>
</tr>
<tr>
<td>QMULS QMULU Dd,Dj,DI,apr8</td>
<td>(Dj) = Dd Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-MM1</td>
<td>1B Bn mb 11</td>
<td>- - - - A A 0 0</td>
</tr>
<tr>
<td>QMULS QMULU Dd,Dj,DI,apr16</td>
<td>(Dj) = Dd Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-MM2</td>
<td>1B Bn mb i2 i1</td>
<td>- - - - A A 0 0</td>
</tr>
<tr>
<td>QMULS QMULU Dd,Dj,DI,apr32</td>
<td>(Dj) = Dd Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-MM4</td>
<td>1B Bn mb i4 i3 i2 i1</td>
<td>- - - - A A 0 0</td>
</tr>
<tr>
<td>QMULS QMULU Dd,Dj,oprmemreg</td>
<td>(Dj) = Dd Memory operand M can be 8, 16, or 32 bits</td>
<td>REG-OPR</td>
<td>1B Bn mb xb</td>
<td>- - - - A A X 0</td>
</tr>
<tr>
<td>QMULS QMULU Dd,Dj,oprmemreg.oprmemreg</td>
<td>(Dj) = Dd Memory operands M1 and M2 can be 8, 16, 24, or 32 bits</td>
<td>REG-OPR1</td>
<td>1B Bn mb xb xb</td>
<td>- - - - A A X 0</td>
</tr>
<tr>
<td>QMULS QMULU Dd,Dj,oprmemreg.oprmemreg</td>
<td>(Dj) = Dd Memory operands M1 and M2 can be 8, 16, 24, or 32 bits</td>
<td>REG-OPR2</td>
<td>1B Bn mb xb xb xb x1</td>
<td>- - - - A A X 0</td>
</tr>
<tr>
<td>QMULS QMULU Dd,Dj,oprmemreg.oprmemreg</td>
<td>(Dj) = Dd Memory operands M1 and M2 can be 8, 16, 24, or 32 bits</td>
<td>REG-OPR3</td>
<td>1B Bn mb xb xb xb x2 x1</td>
<td>- - - - A A X 0</td>
</tr>
</tbody>
</table>

**Source Form**: Various operational formats for the QMULS and QMULU instructions, including direct register (Dj), register-memory (M), and memory-memory operands (M1 and M2).

**Operation**: Fractional multiply operations, both signed (QMULS) and unsigned (QMULU), with results always stored in a register (Dd).

**Address Mode(s)**: Various addressing modes, including register-only (REG), register-memory (REG-IMM), and memory-memory (REG-OPR).

**Machine Coding (hex)**: Hexadecimal representations of the machine codes for each instruction, indicating the size and type of operands used.

**Condition Codes**: Standard condition codes (S X – I NZVC) typically used in microcontroller programming to indicate the outcome of arithmetic operations.

**Note**: The table includes detailed instructions and their respective machine code formats, highlighting the efficiency of register and register-memory versions over memory-memory versions. It covers a range of operand sizes and addressing modes, ensuring comprehensive instruction set coverage for the S12Z architecture.
### Table A-1. S12Z Instruction Set Summary (Sheet 15 of 17)

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Operation</th>
<th>Address Mode(s)</th>
<th>Machine Coding (hex)</th>
<th>Condition Codes</th>
<th>Mode(s)</th>
<th>Source Form</th>
<th>Operation</th>
<th>Address Mode(s)</th>
<th>Machine Coding (hex)</th>
<th>Condition Codes</th>
<th>Mode(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTS</td>
<td>(M(SP) M(SP-1)):M(SP-2) → PCH,PCM,PCL; (SP) + 3; (SP) + 4; (SP) + 5; (SP) + 6</td>
<td>INH 20</td>
<td>8</td>
<td>-</td>
<td>A A A A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAT D from C</td>
<td>saturate( ) = D0</td>
<td>INH 1B 90</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEC</td>
<td>(M(SP) M(SP-1)):M(SP-2) → PCH,PCM,PCL; (SP) + 3; (SP) + 4; (SP) + 5; (SP) + 6</td>
<td>INH 05</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEL</td>
<td>(M(SP) M(SP-1)):M(SP-2) → PCH,PCM,PCL; (SP) + 3; (SP) + 4; (SP) + 5; (SP) + 6</td>
<td>INH 1B 10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEV</td>
<td>(M(SP) M(SP-1)):M(SP-2) → PCH,PCM,PCL; (SP) + 3; (SP) + 4; (SP) + 5; (SP) + 6</td>
<td>INH 02</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBC</td>
<td>(M(SP) M(SP-1)):M(SP-2) → PCH,PCM,PCL; (SP) + 3; (SP) + 4; (SP) + 5; (SP) + 6</td>
<td>INH 1B 12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEC</td>
<td>(M(SP) M(SP-1)):M(SP-2) → PCH,PCM,PCL; (SP) + 3; (SP) + 4; (SP) + 5; (SP) + 6</td>
<td>INH 1B 10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEV</td>
<td>(M(SP) M(SP-1)):M(SP-2) → PCH,PCM,PCL; (SP) + 3; (SP) + 4; (SP) + 5; (SP) + 6</td>
<td>INH 02</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBC</td>
<td>(M(SP) M(SP-1)):M(SP-2) → PCH,PCM,PCL; (SP) + 3; (SP) + 4; (SP) + 5; (SP) + 6</td>
<td>INH 1B 12</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEC</td>
<td>(M(SP) M(SP-1)):M(SP-2) → PCH,PCM,PCL; (SP) + 3; (SP) + 4; (SP) + 5; (SP) + 6</td>
<td>INH 1B 10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEV</td>
<td>(M(SP) M(SP-1)):M(SP-2) → PCH,PCM,PCL; (SP) + 3; (SP) + 4; (SP) + 5; (SP) + 6</td>
<td>INH 02</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### Linear S12 Core Reference Manual, Rev. 1.01

358 Freescale Semiconductor
### Table A-1. S12Z Instruction Set Summary (Sheet 16 of 17)

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Operation</th>
<th>Address Mode(s)</th>
<th>Machine Coding (hex)</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>STOP</em></td>
<td>(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:D7M:D7L:D7OL → M(SP)–M(SP+3); (SP) – 4 → SP; D5H:D5M:D5L:D5OL → M(SP)–M(SP+3); (SP) – 4 → SP; D7H:D7M:D7L:D7OL → M(SP)–M(SP+3); STOP All Clocks</td>
<td>INH</td>
<td>1B 05</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If S control bit = 1, the STOP instruction is disabled and acts like a NOP.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>SUB D,X,Y</em></td>
<td>(X) → (M) = M \ Subtract without Carry to D, M can be a memory operand or another register D</td>
<td>IMM1</td>
<td>7p i1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Memory operand M is the same size as D,</td>
<td>IMM2</td>
<td>7p i2 ii</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>IMM4</td>
<td>7p i4 i3 i2 ii</td>
<td></td>
</tr>
<tr>
<td><em>SWI</em></td>
<td>(SP) – 4 → SP; YL, RTNH,RTNM,RTNL → M(SP)–M(SP+3); (SP) – 4 → SP; XH:XM:XL → M(SP)–M(SP+2); (SP) – 4 → SP; D7H:D7M:D7L:D7OL → M(SP)–M(SP+3); (SP) – 4 → SP; D5H:D5M:D5L:D5OL → M(SP)–M(SP+3); (SP) – 4 → SP; D7H:D7M:D7L:D7OL → M(SP)–M(SP+3); (SP) – 4 → SP; D7H:D7M:D7L:D7OL → M(SP)–M(SP+3); (SP) – 1 → SP; (CCCH) = M(SP); 1 ← i; (SWI Vector) ← PC Software Interrupt</td>
<td>INH</td>
<td>FE</td>
<td></td>
</tr>
<tr>
<td><em>SYS</em></td>
<td>(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:D7M:D7L:D7OL → M(SP)–M(SP+3); (SP) – 4 → SP; D5H:D5M:D5L:D5OL → M(SP)–M(SP+3); (SP) – 4 → SP; D7H:D7M:D7L:D7OL → M(SP)–M(SP+3); (SP) – 4 → SP; D7H:D7M:D7L:D7OL → M(SP)–M(SP+3); (SP) – 1 → SP; (CCCH) = M(SP); 1 ← i; (SYS Vector) ← PC System Call Software Interrupt</td>
<td>INH</td>
<td>FF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 bit can only be changed in supervisor state</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>TBC M,C</em></td>
<td>(M) – 1 → D, Test and branch</td>
<td>REG-R7</td>
<td>0B lb rb</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0B lb rr r1</td>
<td>REG-R15</td>
<td>0B lb rr r1</td>
<td></td>
</tr>
<tr>
<td><em>TBC x,y</em></td>
<td>(X) – 1 → X or (Y) – 1 → Y Test and branch</td>
<td>REG-R7</td>
<td>0B lb rb</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0B lb rr r1</td>
<td>REG-R15</td>
<td>0B lb rr r1</td>
<td></td>
</tr>
<tr>
<td><em>TBC lw,x</em></td>
<td>(M) – 1 → M Test D, 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</td>
<td>OPR-R7</td>
<td>0B lb xb rb</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OPR-R15</td>
<td>0B lb xb rb r1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OPR1-R7</td>
<td>0B lb xb x1 rb</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OPR1-R15</td>
<td>0B lb xb x2 x1 rb</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OPR2-R15</td>
<td>0B lb xb x3 x2 x1 rb</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OPR3-R7</td>
<td>0B lb xb x3 x2 x1 rb</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OPR3-R15</td>
<td>0B lb xb x3 x2 x1 x1 rb</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>TPR opr,x</em></td>
<td>(X) – (Y) or (T) Transfer CPU Register r1 to r2 D0-D7, X, Y, S, CCH, CCL, or CCW if same size, direct transfer</td>
<td>REG-REG</td>
<td>9E tb</td>
<td></td>
</tr>
<tr>
<td></td>
<td>if 1st smaller than 2nd, zero-extend 1st to 2nd</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>if 1st larger than 2nd, transfer low portion of 1st to 2nd (X, and bits can only be changed in supervisor state) (r1:r2 = CCL or CCW, CCR bit may be written directly)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Table A-1. S12Z Instruction Set Summary (Sheet 17 of 17)

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Operation</th>
<th>Address Mode(s)</th>
<th>Machine Coding (hex)</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TRAP #trapnum</strong></td>
<td>(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; 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</td>
<td>INH IB tn</td>
<td>S X INZV C</td>
<td></td>
</tr>
<tr>
<td><strong>WAI</strong></td>
<td>(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; 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)</td>
<td>INH IB 06</td>
<td>S X INZV C</td>
<td></td>
</tr>
<tr>
<td><strong>ZEX cpureg,cpureg</strong></td>
<td>(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; 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)</td>
<td>REG-REG 9E td</td>
<td>S X INZV C</td>
<td></td>
</tr>
</tbody>
</table>

### Source Form
- **TRAP #trapnum**
- **WAI**
- **ZEX cpureg,cpureg**

### Operation
- (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; 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
- (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; 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)
- (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; 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)

### Address Mode(s)
- INH
- IB
- 06
- 9E
- td

### Machine Coding (hex)
- IB tn
- IB 06
- 9E td

### Condition Codes
- S X INZV C

### Notes
- Unimplemented Opcode Trap Interrupt
- (S, X, and I bits can only be changed in supervisor state)
- Wait for Interrupt
- (X and I set depending on interrupt source)
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)

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Opcode</th>
<th>Instruction Mnemonic</th>
<th>Addressing Mode(s) or Postbyte</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>BGO N</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>01</td>
<td>NOP</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>04</td>
<td>BRCLR</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>05</td>
<td>BRSET</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>09</td>
<td>RTS</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>0A</td>
<td>LEA D6</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>0B</td>
<td>LEA D7</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>0C</td>
<td>LEA X</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>0D</td>
<td>LEA Y</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>0E</td>
<td>Dbcc/TBcc</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>0F</td>
<td>MOV B</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>24</td>
<td>SUB</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>25</td>
<td>ADD</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>26</td>
<td>OR</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>27</td>
<td>AND</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>28</td>
<td>MOV</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>29</td>
<td>ADD</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>2A</td>
<td>OR</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>2B</td>
<td>MOV</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>2C</td>
<td>ADD</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>2D</td>
<td>OR</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>2E</td>
<td>MOV</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
<tr>
<td>2F</td>
<td>ADD</td>
<td>INC D2</td>
<td>REL</td>
<td>pg2</td>
</tr>
</tbody>
</table>

 Opcode in Hexadecimal
Instruction Mnemonic
Addressing Mode(s) or Postbyte
Location
<table>
<thead>
<tr>
<th>Opcode in Hexadecimal</th>
<th>Instruction Mnemonic</th>
<th>Addressing Mode(s) or Postbyte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1B 0D</td>
<td>LD S OPR</td>
<td></td>
</tr>
<tr>
<td>1B 10</td>
<td>MINU D2 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 12</td>
<td>DIV D2 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B 14</td>
<td>ADD D2 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B 16</td>
<td>SUB D2 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 18</td>
<td>ADC D2 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B 1A</td>
<td>RTI OPR</td>
<td></td>
</tr>
<tr>
<td>1B 1C</td>
<td>SAT D2 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 1E</td>
<td>OMLU D2 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B 20</td>
<td>TRAP INH</td>
<td></td>
</tr>
<tr>
<td>1B 22</td>
<td>CMP S OPR</td>
<td></td>
</tr>
<tr>
<td>1B 24</td>
<td>MINU D3 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 26</td>
<td>DIV D3 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B 28</td>
<td>ADD D3 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B 2A</td>
<td>ADC D3 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B 2C</td>
<td>RCI OPR</td>
<td></td>
</tr>
<tr>
<td>1B 2E</td>
<td>OMLU D3 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B 30</td>
<td>TRAP INH</td>
<td></td>
</tr>
<tr>
<td>1B 32</td>
<td>CMP S OPR</td>
<td></td>
</tr>
<tr>
<td>1B 34</td>
<td>MINU D4 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 36</td>
<td>DIV D4 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B 38</td>
<td>ADD D4 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B 3A</td>
<td>ADC D4 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B 3C</td>
<td>OLMU D4 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B 3E</td>
<td>TRAP INH</td>
<td></td>
</tr>
<tr>
<td>1B 40</td>
<td>STOP INH</td>
<td></td>
</tr>
<tr>
<td>1B 42</td>
<td>MINU D5 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 44</td>
<td>DIV D5 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B 46</td>
<td>ADD D5 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B 48</td>
<td>ADC D5 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B 4A</td>
<td>TRAP INH</td>
<td></td>
</tr>
<tr>
<td>1B 4C</td>
<td>IMM3 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 4E</td>
<td>IMM4 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 50</td>
<td>MOD D7 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 52</td>
<td>MOD D6 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 54</td>
<td>MOD D5 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 56</td>
<td>MOD D4 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 58</td>
<td>MOD D3 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 5A</td>
<td>MOD D2 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 5C</td>
<td>MOD D1 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 5E</td>
<td>MOD D0 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 60</td>
<td>MAXU D2 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 62</td>
<td>MAXS D2 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 64</td>
<td>MINU D2 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 66</td>
<td>DIV D2 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B 68</td>
<td>ADD D2 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B 6A</td>
<td>ADC D2 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B 6C</td>
<td>RCI OPR</td>
<td></td>
</tr>
<tr>
<td>1B 6E</td>
<td>OMLU D2 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B 70</td>
<td>TRAP INH</td>
<td></td>
</tr>
<tr>
<td>1B 72</td>
<td>CMP S OPR</td>
<td></td>
</tr>
<tr>
<td>1B 74</td>
<td>MINU D3 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 76</td>
<td>DIV D3 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B 78</td>
<td>ADD D3 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B 7A</td>
<td>ADC D3 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B 7C</td>
<td>OLMU D3 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B 7E</td>
<td>TRAP INH</td>
<td></td>
</tr>
<tr>
<td>1B 80</td>
<td>STOP INH</td>
<td></td>
</tr>
<tr>
<td>1B 82</td>
<td>MINU D4 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 84</td>
<td>DIV D4 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B 86</td>
<td>ADD D4 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B 88</td>
<td>ADC D4 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B 8A</td>
<td>TRAP INH</td>
<td></td>
</tr>
<tr>
<td>1B 8C</td>
<td>IMM3 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 8E</td>
<td>IMM4 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 90</td>
<td>MOD D7 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 92</td>
<td>MOD D6 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 94</td>
<td>MOD D5 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 96</td>
<td>MOD D4 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 98</td>
<td>MOD D3 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 9A</td>
<td>MOD D2 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 9C</td>
<td>MOD D1 OPR</td>
<td></td>
</tr>
<tr>
<td>1B 9E</td>
<td>MOD D0 OPR</td>
<td></td>
</tr>
<tr>
<td>1B A0</td>
<td>MAXU D3 OPR</td>
<td></td>
</tr>
<tr>
<td>1B A2</td>
<td>MAXS D3 OPR</td>
<td></td>
</tr>
<tr>
<td>1B A4</td>
<td>MINU D3 OPR</td>
<td></td>
</tr>
<tr>
<td>1B A6</td>
<td>DIV D3 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B A8</td>
<td>ADD D3 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B AA</td>
<td>ADC D3 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B AC</td>
<td>RCI OPR</td>
<td></td>
</tr>
<tr>
<td>1B AD</td>
<td>OLMU D3 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B AF</td>
<td>TRAP INH</td>
<td></td>
</tr>
<tr>
<td>1B B0</td>
<td>MAXU D4 OPR</td>
<td></td>
</tr>
<tr>
<td>1B B2</td>
<td>MAXS D4 OPR</td>
<td></td>
</tr>
<tr>
<td>1B B4</td>
<td>MINU D4 OPR</td>
<td></td>
</tr>
<tr>
<td>1B B6</td>
<td>DIV D4 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B B8</td>
<td>ADD D4 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B BA</td>
<td>ADC D4 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B BC</td>
<td>OLMU D4 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B BD</td>
<td>TRAP INH</td>
<td></td>
</tr>
<tr>
<td>1B C0</td>
<td>MAXU D5 OPR</td>
<td></td>
</tr>
<tr>
<td>1B C2</td>
<td>MAXS D5 OPR</td>
<td></td>
</tr>
<tr>
<td>1B C4</td>
<td>MINU D5 OPR</td>
<td></td>
</tr>
<tr>
<td>1B C6</td>
<td>DIV D5 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B C8</td>
<td>ADD D5 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B CA</td>
<td>ADC D5 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B CE</td>
<td>OLMU D5 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B CF</td>
<td>TRAP INH</td>
<td></td>
</tr>
<tr>
<td>1B D0</td>
<td>MAXU D6 OPR</td>
<td></td>
</tr>
<tr>
<td>1B D2</td>
<td>MAXS D6 OPR</td>
<td></td>
</tr>
<tr>
<td>1B D4</td>
<td>MINU D6 OPR</td>
<td></td>
</tr>
<tr>
<td>1B D6</td>
<td>DIV D6 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B D8</td>
<td>ADD D6 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B DA</td>
<td>ADC D6 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B DC</td>
<td>OLMU D6 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B E0</td>
<td>TRAP INH</td>
<td></td>
</tr>
<tr>
<td>1B E2</td>
<td>MAXU D7 OPR</td>
<td></td>
</tr>
<tr>
<td>1B E4</td>
<td>MAXS D7 OPR</td>
<td></td>
</tr>
<tr>
<td>1B E6</td>
<td>MINU D7 OPR</td>
<td></td>
</tr>
<tr>
<td>1B E8</td>
<td>DIV D7 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B EA</td>
<td>ADD D7 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B EC</td>
<td>ADC D7 IMM2</td>
<td></td>
</tr>
<tr>
<td>1B EE</td>
<td>OLMU D7 postbyte mb</td>
<td></td>
</tr>
<tr>
<td>1B EF</td>
<td>TRAP INH</td>
<td></td>
</tr>
</tbody>
</table>

Table A-2. Opcode Map (Sheet 2 of 2)
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>
<thead>
<tr>
<th>xb postbyte bitwise encoding</th>
<th>Summary Source Form</th>
<th>Summary Address Mode</th>
<th>Operand Machine Coding</th>
<th>Detailed Source Form</th>
<th>Detailed Addressing Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>b7 b6 b5 b4 b3 b2 b1 b0</td>
<td></td>
<td>xb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 1 1 e4 IMM (–1, 1, 2 ... 14, 15)</td>
<td>INST oprmemreg</td>
<td>OPR</td>
<td>xb</td>
<td>INST #oprmemreg:</td>
<td>IMM: Short Immediate (~1, 1, 2, .. 14, 15)</td>
</tr>
<tr>
<td>0 1 1 1 1 D[2:0]</td>
<td></td>
<td></td>
<td></td>
<td>INST [Di]</td>
<td>REG - Register as operand</td>
</tr>
<tr>
<td>0 1 XYS u4 (0...15)</td>
<td></td>
<td></td>
<td></td>
<td>INST (opr4,xys)</td>
<td>IDX - u4 Constant offset from xys</td>
</tr>
<tr>
<td>1 1 XY 0 0 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td>INST (xy)</td>
<td></td>
</tr>
<tr>
<td>1 1 XY 0 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td>INST (–xy)</td>
<td></td>
</tr>
<tr>
<td>1 0 XY 0 0 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td>INST (+xy)</td>
<td></td>
</tr>
<tr>
<td>1 1 0 XY 0 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td>INST (+–xy)</td>
<td>Pre/post inc/dec –xy+-s</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td>INST (+s)</td>
<td></td>
</tr>
<tr>
<td>1 0 XYS 1 D[2:0]</td>
<td></td>
<td></td>
<td></td>
<td>INST (Di,xys)</td>
<td>D[0,1,2,3,4,7] unsigned; D[2–4] signed</td>
</tr>
<tr>
<td>1 1 XYS 0 0 0 0 0 sign</td>
<td></td>
<td></td>
<td></td>
<td>INST [D[0,1,2,3,4,7],xys]</td>
<td>REG,IDX - Register offset from yys D[0,1,6,7] unsigned; D[2–4] signed</td>
</tr>
<tr>
<td>1 1 XYS 0 1 0 0 0 sign</td>
<td></td>
<td></td>
<td></td>
<td>INST [D[0,1,2,3,4,7],xys]</td>
<td>REG,IDX - Register offset from yys Indirect D[0,1,6,7] unsigned; D[2–4] signed</td>
</tr>
<tr>
<td>0 0 0 Add[17:8]</td>
<td></td>
<td></td>
<td></td>
<td>INST [opru18]</td>
<td>U1 Short Extended (LIM 16K)</td>
</tr>
<tr>
<td>1 1 1 1 1 1 A[7] 0 A[8]</td>
<td></td>
<td></td>
<td></td>
<td>INST [opru18,D]</td>
<td>U1 Extended (LIM 256K)</td>
</tr>
<tr>
<td>1 1 XYS 0 0 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td>INST [opru18,D]</td>
<td>U1 Extended (LIM 256K)</td>
</tr>
<tr>
<td>1 1 XYS 0 1 0 1 0</td>
<td></td>
<td></td>
<td></td>
<td>INST [opru18,D]</td>
<td>U1 Extended (LIM 256K)</td>
</tr>
<tr>
<td>1 1 1 1 0 1 D[2:0]</td>
<td></td>
<td></td>
<td></td>
<td>INST [opru18,D]</td>
<td>U1 Extended (LIM 256K)</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td>INST [opru18,D]</td>
<td>U1 Extended (LIM 256K)</td>
</tr>
</tbody>
</table>

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=X, 1=Y, 1=S, and 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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Y</td>
<td>S</td>
<td>–1</td>
<td>INT4</td>
<td>n.D2</td>
<td>n.S</td>
<td>n.X</td>
<td>n.Y</td>
<td>n.DX</td>
<td>n.SX</td>
<td>n.YX</td>
</tr>
<tr>
<td>0</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Y</td>
<td>S</td>
<td>–1</td>
<td>INT4</td>
<td>n.D2</td>
<td>n.S</td>
<td>n.X</td>
<td>n.Y</td>
<td>n.DX</td>
<td>n.SX</td>
<td>n.YX</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>Y</td>
<td>S</td>
<td>n.X</td>
<td>n.Y</td>
<td>n.DX</td>
<td>n.SX</td>
<td>n.YX</td>
<td>n.XX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>Y</td>
<td>S</td>
<td>n.X</td>
<td>n.Y</td>
<td>n.DX</td>
<td>n.SX</td>
<td>n.YX</td>
<td>n.XX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Y</td>
<td>S</td>
<td>n.X</td>
<td>n.Y</td>
<td>n.DX</td>
<td>n.SX</td>
<td>n.YX</td>
<td>n.XX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Appendix A Instruction Reference

**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 Decode

<table>
<thead>
<tr>
<th>mb postbyte bitwise encoding</th>
<th>Addressing modes for Operand 1, Operand 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>b7 b6 b5 b4 b3 b2 b1 b0</td>
<td>Registers, Register, Register, OPR.B, OPR.W, OPR.L, OPR.R</td>
</tr>
<tr>
<td>0</td>
<td>Rs1</td>
</tr>
<tr>
<td>1</td>
<td>Rs1</td>
</tr>
<tr>
<td>1</td>
<td>Rs1</td>
</tr>
<tr>
<td>1</td>
<td>Rs1</td>
</tr>
<tr>
<td>1</td>
<td>Rs1</td>
</tr>
<tr>
<td>1</td>
<td>Rs1</td>
</tr>
<tr>
<td>1</td>
<td>Rs1</td>
</tr>
<tr>
<td>1</td>
<td>Rs1</td>
</tr>
</tbody>
</table>

1 = Signed, 0 = Unsigned

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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2</td>
<td>D1</td>
<td>D0</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
</tr>
<tr>
<td>D2</td>
<td>D1</td>
<td>D0</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
</tr>
<tr>
<td>D2</td>
<td>D1</td>
<td>D0</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
</tr>
<tr>
<td>D2</td>
<td>D1</td>
<td>D0</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
</tr>
<tr>
<td>D2</td>
<td>D1</td>
<td>D0</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
</tr>
<tr>
<td>D2</td>
<td>D1</td>
<td>D0</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D3</td>
</tr>
</tbody>
</table>

Linear S12 Core Reference Manual, Rev. 1.01

Freescale Semiconductor 365


A.4.3  Loop Primitive Postbyte (lb)

The lb postbyte shows the coding for the DBcc and TBcc loop primitive instructions. Decrement or Test Di, 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

<table>
<thead>
<tr>
<th>Ib postbyte bitwise encoding</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Test or Decrement Di and then branch based on condition CC (NE, EQ, PL, MI, GT, or LE)</td>
</tr>
<tr>
<td>0x01</td>
<td>Test or Decrement X or Y and then branch based on condition CC (NE, EQ, PL, MI, GT, or LE)</td>
</tr>
<tr>
<td>0x02</td>
<td>Test or Decrement Memory location (B, W, P, or L) and then branch based on condition CC</td>
</tr>
</tbody>
</table>

Table A-8. Loop Postbyte (lb) Coding Map

<table>
<thead>
<tr>
<th>Test and Branch</th>
<th>Decrement and Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>TBNE.D2</td>
</tr>
<tr>
<td>0x01</td>
<td>TBNE.D3</td>
</tr>
<tr>
<td>0x02</td>
<td>TBNE.D4</td>
</tr>
<tr>
<td>0x03</td>
<td>TBNE.D5</td>
</tr>
<tr>
<td>0x04</td>
<td>TBNE.D6</td>
</tr>
<tr>
<td>0x05</td>
<td>TBNE.D7</td>
</tr>
<tr>
<td>0x06</td>
<td>TBNE.X</td>
</tr>
<tr>
<td>0x07</td>
<td>TBNE.Y</td>
</tr>
<tr>
<td>0x08</td>
<td>TBNE.B</td>
</tr>
<tr>
<td>0x09</td>
<td>TBNE.W</td>
</tr>
<tr>
<td>0x0A</td>
<td>TBNE.L</td>
</tr>
<tr>
<td>0x0B</td>
<td>TBNE.P</td>
</tr>
<tr>
<td>0x0C</td>
<td>TBNE.L</td>
</tr>
<tr>
<td>0x0D</td>
<td>TBNE.W</td>
</tr>
<tr>
<td>0x0E</td>
<td>TBNE.B</td>
</tr>
<tr>
<td>0x0F</td>
<td>TBNE.P</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>sb postbyte bitwise encoding</th>
<th>Comments</th>
<th>Source Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>b7 b6 b5 b4 b3 b2 b1 b0</td>
<td>Dd &lt;= Ds &lt;&lt;&gt;&gt; #n Efficient shift by n=1 or 2</td>
<td>SHIFT Dd, Ds, #opr1i</td>
</tr>
<tr>
<td>AL L/R 0 1 N[0] 0</td>
<td>Dd &lt;= A/L L/R 0 1 N[0] 0 size[1:0]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>AL L/R 1 0 x N[0] 0 size[1:0]</td>
<td>Dd &lt;= Rotate x L/R 1 0 x 1 size[1:0]</td>
<td>SHIFT oprmemreg :ROL or ROR</td>
</tr>
<tr>
<td>AL L/R 1 1 N[0] 0 size[1:0]</td>
<td>Dd &lt;= postbyte xb</td>
<td>SHIFT Dd, oprmemreg, #opr1i</td>
</tr>
</tbody>
</table>

Table A.10. Shift Postbyte (sb) Coding Map

<table>
<thead>
<tr>
<th>Logical</th>
<th>Arithmetic</th>
<th>Rights</th>
<th>Lefts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rights</th>
<th>Lefts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Appendix A Instruction Reference

Linear S12 Core Reference Manual, Rev. 1.01

Freescale Semiconductor

367
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

<table>
<thead>
<tr>
<th>bm postbyte bitwise encoding</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:1:1=0</td>
<td>D0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0:1:0=1</td>
<td>D1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0:0:1=0</td>
<td>D2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0:0:0=0</td>
<td>D3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0:0:1=1</td>
<td>D4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0:0:0=1</td>
<td>D5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0:0:1=0</td>
<td>D6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0:0:0=0</td>
<td>D7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A-12. Bit Manipulation Postbyte (bm) Coding Map

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>D2</td>
<td>D1</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D8</td>
<td>D9</td>
<td>D10</td>
<td>D11</td>
<td>D12</td>
<td>D13</td>
<td>D14</td>
</tr>
</tbody>
</table>

Comments

- bit n (0-7) in 8-bit register D0 or D1 (Di[2:0] = 1:0:0 or 1:0:1) reserved, like above but d6 is don't care and acts as d6=0
- bit n (0-15) in 16-bit register D2..D7 (Di[2:0] = 0:0:0, 0:0:1, 0:1:0, or 0:1:1) reserved, like above but d6 is don't care and acts like d6=0
- Operands in memory (size[1:0]=byte, word=1, long=1) n (in Dn) = bit number
- Operands in memory operand OPR.L
- Operands in memory operand OPR.W
- Operands in memory operand OPR.B

Linear S12 Core Reference Manual, Rev. 1.01
Freescale Semiconductor
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>
<thead>
<tr>
<th>bb postbyte bitwise encoding</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>b7</td>
<td>b6</td>
</tr>
<tr>
<td>0 0</td>
<td>Ds[2:0]</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>0_</td>
<td>1_</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>BFEXT Dd,Dd</td>
<td>BFEXT Dd,Dd</td>
</tr>
<tr>
<td>BFINS Dd,Dd</td>
<td>BFINS Dd,Dd</td>
</tr>
<tr>
<td>BFEXT Dd,Dd</td>
<td>BFEXT Dd,Dd</td>
</tr>
<tr>
<td>BFINS Dd,Dd</td>
<td>BFINS Dd,Dd</td>
</tr>
<tr>
<td>BFEXT Dd,Dd</td>
<td>BFEXT Dd,Dd</td>
</tr>
<tr>
<td>BFINS Dd,Dd</td>
<td>BFINS Dd,Dd</td>
</tr>
<tr>
<td>BFEXT Dd,Dd</td>
<td>BFEXT Dd,Dd</td>
</tr>
<tr>
<td>BFINS Dd,Dd</td>
<td>BFINS Dd,Dd</td>
</tr>
<tr>
<td>BFEXT Dd,Dd</td>
<td>BFEXT Dd,Dd</td>
</tr>
<tr>
<td>BFINS Dd,Dd</td>
<td>BFINS Dd,Dd</td>
</tr>
<tr>
<td>BFEXT Dd,Dd</td>
<td>BFEXT Dd,Dd</td>
</tr>
<tr>
<td>BFINS Dd,Dd</td>
<td>BFINS Dd,Dd</td>
</tr>
<tr>
<td>BFEXT Dd,Dd</td>
<td>BFEXT Dd,Dd</td>
</tr>
<tr>
<td>BFINS Dd,Dd</td>
<td>BFINS Dd,Dd</td>
</tr>
<tr>
<td>BFEXT Dd,Dd</td>
<td>BFEXT Dd,Dd</td>
</tr>
<tr>
<td>BFINS Dd,Dd</td>
<td>BFINS Dd,Dd</td>
</tr>
</tbody>
</table>

Table A-14. Bitfield Extract/Insert Postbyte (bb) Coding Map
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.

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>
<thead>
<tr>
<th>eb and tb postbyte bitwise encoding</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>b7 b6 b5 b4 b3 b2 b1 b0</td>
<td>Refer to coding maps for exchange and transfer to see how the registers are assigned to 4-bit codes</td>
</tr>
</tbody>
</table>

Refer to 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

<table>
<thead>
<tr>
<th>source</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>D5</th>
<th>D0</th>
<th>D1</th>
<th>D6</th>
<th>D7</th>
<th>X</th>
<th>Y</th>
<th>S</th>
<th>-</th>
<th>CCH</th>
<th>CCL</th>
<th>CCW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0-</td>
<td>1-</td>
<td>2-</td>
<td>3-</td>
<td>4-</td>
<td>5-</td>
<td>6-</td>
<td>7-</td>
<td>8-</td>
<td>9-</td>
<td>A-</td>
<td>B-</td>
<td>C-</td>
<td>D-</td>
<td>E-</td>
</tr>
<tr>
<td>D2</td>
<td>-</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>CCW</td>
<td>D2</td>
</tr>
<tr>
<td>D3</td>
<td>-</td>
<td>D2</td>
<td>-</td>
<td>D3</td>
<td>D5</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>CCW</td>
</tr>
<tr>
<td>D4</td>
<td>-</td>
<td>D2</td>
<td>D3</td>
<td>-</td>
<td>D5</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>CCW</td>
</tr>
<tr>
<td>D5</td>
<td>-</td>
<td>D2</td>
<td>D3</td>
<td>D4</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>CCW</td>
<td>D5</td>
</tr>
<tr>
<td>D0</td>
<td>-</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>CCW</td>
<td>D0</td>
</tr>
<tr>
<td>D1</td>
<td>-</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>CCW</td>
<td>D1</td>
</tr>
<tr>
<td>D6</td>
<td>-</td>
<td>sex:D2</td>
<td>sex:D3</td>
<td>sex:D4</td>
<td>sex:D5</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>CCW</td>
</tr>
<tr>
<td>X</td>
<td>-</td>
<td>sex:D2</td>
<td>sex:D3</td>
<td>sex:D4</td>
<td>sex:D5</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>CCW</td>
</tr>
<tr>
<td>Y</td>
<td>-</td>
<td>sex:D2</td>
<td>sex:D3</td>
<td>sex:D4</td>
<td>sex:D5</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>CCW</td>
</tr>
<tr>
<td>S</td>
<td>-</td>
<td>sex:D2</td>
<td>sex:D3</td>
<td>sex:D4</td>
<td>sex:D5</td>
<td>sex:D0</td>
<td>sex:D1</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>Big</td>
<td>sex:CCH</td>
<td>sex:CCL</td>
<td>CCW</td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>source</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>D5</th>
<th>D0</th>
<th>D1</th>
<th>D6</th>
<th>D7</th>
<th>X</th>
<th>Y</th>
<th>S</th>
<th>-</th>
<th>CCH</th>
<th>CCL</th>
<th>CCW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0-</td>
<td>1-</td>
<td>2-</td>
<td>3-</td>
<td>4-</td>
<td>5-</td>
<td>6-</td>
<td>7-</td>
<td>8-</td>
<td>9-</td>
<td>A-</td>
<td>B-</td>
<td>C-</td>
<td>D-</td>
<td>E-</td>
</tr>
<tr>
<td>D2</td>
<td>=&gt; D0</td>
<td>=&gt; D0</td>
<td>=&gt; D0</td>
<td>=&gt; D0</td>
<td>=&gt; D0</td>
<td>=&gt; D0</td>
<td>=&gt; D0</td>
<td>=&gt; D0</td>
<td>=&gt; D0</td>
<td>=&gt; D0</td>
<td>=&gt; D0</td>
<td>=&gt; D0</td>
<td>=&gt; D0</td>
<td>=&gt; D0</td>
<td>=&gt; D0</td>
</tr>
<tr>
<td>D3</td>
<td>=&gt; D3</td>
<td>=&gt; D3</td>
<td>=&gt; D3</td>
<td>=&gt; D3</td>
<td>=&gt; D3</td>
<td>=&gt; D3</td>
<td>=&gt; D3</td>
<td>=&gt; D3</td>
<td>=&gt; D3</td>
<td>=&gt; D3</td>
<td>=&gt; D3</td>
<td>=&gt; D3</td>
<td>=&gt; D3</td>
<td>=&gt; D3</td>
<td>=&gt; D3</td>
</tr>
<tr>
<td>D4</td>
<td>=&gt; D4</td>
<td>=&gt; D4</td>
<td>=&gt; D4</td>
<td>=&gt; D4</td>
<td>=&gt; D4</td>
<td>=&gt; D4</td>
<td>=&gt; D4</td>
<td>=&gt; D4</td>
<td>=&gt; D4</td>
<td>=&gt; D4</td>
<td>=&gt; D4</td>
<td>=&gt; D4</td>
<td>=&gt; D4</td>
<td>=&gt; D4</td>
<td>=&gt; D4</td>
</tr>
<tr>
<td>D5</td>
<td>=&gt; D5</td>
<td>=&gt; D5</td>
<td>=&gt; D5</td>
<td>=&gt; D5</td>
<td>=&gt; D5</td>
<td>=&gt; D5</td>
<td>=&gt; D5</td>
<td>=&gt; D5</td>
<td>=&gt; D5</td>
<td>=&gt; D5</td>
<td>=&gt; D5</td>
<td>=&gt; D5</td>
<td>=&gt; D5</td>
<td>=&gt; D5</td>
<td>=&gt; D5</td>
</tr>
<tr>
<td>D6</td>
<td>=&gt; D6</td>
<td>=&gt; D6</td>
<td>=&gt; D6</td>
<td>=&gt; D6</td>
<td>=&gt; D6</td>
<td>=&gt; D6</td>
<td>=&gt; D6</td>
<td>=&gt; D6</td>
<td>=&gt; D6</td>
<td>=&gt; D6</td>
<td>=&gt; D6</td>
<td>=&gt; D6</td>
<td>=&gt; D6</td>
<td>=&gt; D6</td>
<td>=&gt; D6</td>
</tr>
<tr>
<td>D7</td>
<td>=&gt; D7</td>
<td>=&gt; D7</td>
<td>=&gt; D7</td>
<td>=&gt; D7</td>
<td>=&gt; D7</td>
<td>=&gt; D7</td>
<td>=&gt; D7</td>
<td>=&gt; D7</td>
<td>=&gt; D7</td>
<td>=&gt; D7</td>
<td>=&gt; D7</td>
<td>=&gt; D7</td>
<td>=&gt; D7</td>
<td>=&gt; D7</td>
<td>=&gt; D7</td>
</tr>
<tr>
<td>X</td>
<td>=&gt; X</td>
<td>=&gt; X</td>
<td>=&gt; X</td>
<td>=&gt; X</td>
<td>=&gt; X</td>
<td>=&gt; X</td>
<td>=&gt; X</td>
<td>=&gt; X</td>
<td>=&gt; X</td>
<td>=&gt; X</td>
<td>=&gt; X</td>
<td>=&gt; X</td>
<td>=&gt; X</td>
<td>=&gt; X</td>
<td>=&gt; X</td>
</tr>
<tr>
<td>Y</td>
<td>=&gt; Y</td>
<td>=&gt; Y</td>
<td>=&gt; Y</td>
<td>=&gt; Y</td>
<td>=&gt; Y</td>
<td>=&gt; Y</td>
<td>=&gt; Y</td>
<td>=&gt; Y</td>
<td>=&gt; Y</td>
<td>=&gt; Y</td>
<td>=&gt; Y</td>
<td>=&gt; Y</td>
<td>=&gt; Y</td>
<td>=&gt; Y</td>
<td>=&gt; Y</td>
</tr>
<tr>
<td>S</td>
<td>=&gt; S</td>
<td>=&gt; S</td>
<td>=&gt; S</td>
<td>=&gt; S</td>
<td>=&gt; S</td>
<td>=&gt; S</td>
<td>=&gt; S</td>
<td>=&gt; S</td>
<td>=&gt; S</td>
<td>=&gt; S</td>
<td>=&gt; S</td>
<td>=&gt; S</td>
<td>=&gt; S</td>
<td>=&gt; S</td>
<td>=&gt; S</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>reserved</th>
<th>-F</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCH</td>
<td>=&gt; CCH</td>
</tr>
<tr>
<td>CCL</td>
<td>=&gt; CCL</td>
</tr>
<tr>
<td>CCW</td>
<td>=&gt; CCW</td>
</tr>
</tbody>
</table>

Table A-17. Transfer Postbyte (tb) Coding Map
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.

Refer to the Count Leading Sign-Bits coding map below (shaded fields are reserved).

Table A-18. Count Leading Sign-Bits (cb) Decode

<table>
<thead>
<tr>
<th>eb and tb postbyte bitwise encoding</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>b7 b6 b5 b4 b3 b2 b1 b0</td>
<td></td>
</tr>
<tr>
<td>0 1 SOURCE[2:0]</td>
<td></td>
</tr>
<tr>
<td>0 0 DEST[2:0]</td>
<td></td>
</tr>
</tbody>
</table>

Table A-19. Count Leading Sign-Bits (cb) Coding Map

<table>
<thead>
<tr>
<th>eb and tb postbyte bitwise encoding</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>b7 b6 b5 b4 b3 b2 b1 b0</td>
<td></td>
</tr>
<tr>
<td>0 1 SOURCE[2:0]</td>
<td></td>
</tr>
<tr>
<td>0 0 DEST[2:0]</td>
<td></td>
</tr>
</tbody>
</table>

Refer to coding map for Count Leading Sign-Bits to see how the registers are assigned to 3-bit codes.

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

<table>
<thead>
<tr>
<th>pb postbyte bitwise encoding</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>b7 b6 b5 b4 b3 b2 b1 b0</td>
<td></td>
</tr>
<tr>
<td>0 1 D4 D5 D6 D7 X Y</td>
<td>Push selected registers in register mask 1 list (pb = 0x00 = PSH ALL)</td>
</tr>
<tr>
<td>1 0 CCH CCL D0 D1 D2 D3</td>
<td>Push selected registers in register mask 2 list (pb = 0x40 = PSH ALL16b)</td>
</tr>
<tr>
<td>1 1 D4 D5 D6 D7 X Y</td>
<td>Pull selected registers in register mask 1 list (pb = 0x80 = PUL ALL)</td>
</tr>
<tr>
<td>YL YL X X</td>
<td>Pull selected registers in register mask 2 list (pb = 0xC0 = PUL ALL16b)</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>D7L</th>
<th>D7</th>
<th>D7H</th>
</tr>
</thead>
<tbody>
<tr>
<td>D6L</td>
<td>D6</td>
<td>D6H</td>
</tr>
<tr>
<td>D5L</td>
<td>D5</td>
<td>D5H</td>
</tr>
<tr>
<td>D4L</td>
<td>D4</td>
<td>D4H</td>
</tr>
<tr>
<td>D3L</td>
<td>D3</td>
<td>D3H</td>
</tr>
<tr>
<td>D2L</td>
<td>D2</td>
<td>D2H</td>
</tr>
<tr>
<td>D1</td>
<td>D1</td>
<td></td>
</tr>
<tr>
<td>D0</td>
<td>D0</td>
<td></td>
</tr>
<tr>
<td>CCL</td>
<td>CCR</td>
<td></td>
</tr>
<tr>
<td>CCH</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>