CLIPPER™
32-Bit Microprocessor Module

INSTRUCTION SET
CLIPPER™
32-Bit Microprocessor Module

INSTRUCTION SET

ADVANCE INFORMATION
Fairchild reserves the right to make changes in the circuitry or specifications at any time without notice.

CLIPPER is a trademark of Fairchild Camera and Instrument Corporation.

UNIX is a trademark of AT&T Bell Laboratories.

# CLIPPER™
# 32-BIT MICROPROCESSOR MODULE
# INSTRUCTION SET

## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER 1. INSTRUCTION FORMATS</th>
<th>cnvdw</th>
<th>Convert Double to Word</th>
<th>2-27</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Introduction</td>
<td>cnvrdw</td>
<td>Convert Rounding Double</td>
<td>2-28</td>
</tr>
<tr>
<td>1.2 Instruction Formats — Without Address</td>
<td>cnrsw</td>
<td>Convert Rounding Single</td>
<td>2-28</td>
</tr>
<tr>
<td>1.2.1 Register</td>
<td>cnvrdw</td>
<td>to Word</td>
<td>2-28</td>
</tr>
<tr>
<td>1.2.2 Quick</td>
<td>cnvsd</td>
<td>Convert Single to</td>
<td>2-29</td>
</tr>
<tr>
<td>1.2.3 16-Bit Immediate</td>
<td>cnvsd</td>
<td>Double Floating</td>
<td>2-30</td>
</tr>
<tr>
<td>1.2.4 32-Bit Immediate</td>
<td>cnvsd</td>
<td>to Word</td>
<td>2-29</td>
</tr>
<tr>
<td>1.2.5 Control</td>
<td>cnvsw</td>
<td>Convert Single Floating</td>
<td>2-31</td>
</tr>
<tr>
<td>1.2.6 Macro</td>
<td>cnvsw</td>
<td>to Word</td>
<td>2-31</td>
</tr>
<tr>
<td>1.3 Instruction Formats — With Address</td>
<td>cnvtsw</td>
<td>Convert Truncating Double</td>
<td>2-32</td>
</tr>
<tr>
<td>1.3.1 Relative</td>
<td>cnvtsw</td>
<td>to Word</td>
<td>2-32</td>
</tr>
<tr>
<td>1.3.2 Relative With 12-Bit Displacement</td>
<td>cnvtdw</td>
<td>Convert Truncating Single</td>
<td>2-33</td>
</tr>
<tr>
<td>1.3.3 Relative With 32-Bit Displacement</td>
<td>cnvsw</td>
<td>to Word</td>
<td>2-33</td>
</tr>
<tr>
<td>1.3.4 16-Bit Absolute</td>
<td>cnvsw</td>
<td>Double Floating</td>
<td>2-34</td>
</tr>
<tr>
<td>1.3.5 32-Bit Absolute</td>
<td>cnvsw</td>
<td>to Single Floating</td>
<td>2-35</td>
</tr>
<tr>
<td>1.3.6 PC Relative With 16-Bit Displacement</td>
<td>divd</td>
<td>Divide Double Floating</td>
<td>2-36</td>
</tr>
<tr>
<td>1.3.7 PC Relative With 32-Bit Displacement</td>
<td>divs</td>
<td>Divide Single Floating</td>
<td>2-37</td>
</tr>
<tr>
<td>1.3.8 Relative Indexed</td>
<td>divw</td>
<td>Divide Word</td>
<td>2-38</td>
</tr>
<tr>
<td>1.3.9 PC Indexed</td>
<td>divwu</td>
<td>Divide Word Unsigned</td>
<td>2-39</td>
</tr>
<tr>
<td></td>
<td>initc</td>
<td>Initialize Characters</td>
<td>2-40</td>
</tr>
<tr>
<td></td>
<td>loada</td>
<td>Load Address</td>
<td>2-41</td>
</tr>
<tr>
<td></td>
<td>loadb</td>
<td>Load Byte</td>
<td>2-42</td>
</tr>
<tr>
<td></td>
<td>loadbu</td>
<td>Load Byte Unsigned</td>
<td>2-43</td>
</tr>
<tr>
<td></td>
<td>loadd</td>
<td>Load Double Floating</td>
<td>2-44</td>
</tr>
<tr>
<td></td>
<td>loadfs</td>
<td>Load Floating Status</td>
<td>2-45</td>
</tr>
<tr>
<td></td>
<td>loadh</td>
<td>Load Halfword</td>
<td>2-46</td>
</tr>
<tr>
<td></td>
<td>loadhu</td>
<td>Load Halfword Unsigned</td>
<td>2-47</td>
</tr>
<tr>
<td></td>
<td>loadi</td>
<td>Load Immediate</td>
<td>2-48</td>
</tr>
<tr>
<td></td>
<td>loadq</td>
<td>Load Quick</td>
<td>2-49</td>
</tr>
<tr>
<td></td>
<td>loads</td>
<td>Load Single Floating</td>
<td>2-50</td>
</tr>
<tr>
<td></td>
<td>loadw</td>
<td>Load Word</td>
<td>2-51</td>
</tr>
<tr>
<td></td>
<td>modw</td>
<td>Modulus Word</td>
<td>2-52</td>
</tr>
<tr>
<td></td>
<td>modwu</td>
<td>Modulus Word Unsigned</td>
<td>2-53</td>
</tr>
<tr>
<td></td>
<td>movc</td>
<td>Move Characters</td>
<td>2-54</td>
</tr>
<tr>
<td></td>
<td>movd</td>
<td>Move Double Floating</td>
<td>2-55</td>
</tr>
<tr>
<td></td>
<td>movdl</td>
<td>to Longword</td>
<td>2-56</td>
</tr>
<tr>
<td></td>
<td>movld</td>
<td>Move Longword to Double</td>
<td>2-57</td>
</tr>
<tr>
<td></td>
<td>movpw</td>
<td>Move Processor Register</td>
<td>to Word</td>
</tr>
<tr>
<td></td>
<td>moves</td>
<td>Move Single Floating</td>
<td>2-59</td>
</tr>
</tbody>
</table>

## CHAPTER 2. INSTRUCTION SET

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>addd</td>
<td>Add Double Floating</td>
<td>2-7</td>
</tr>
<tr>
<td>addi</td>
<td>Add Immediate</td>
<td>2-8</td>
</tr>
<tr>
<td>addq</td>
<td>Add Quick</td>
<td>2-9</td>
</tr>
<tr>
<td>adds</td>
<td>Add Single Floating</td>
<td>2-10</td>
</tr>
<tr>
<td>addw</td>
<td>Add Word</td>
<td>2-11</td>
</tr>
<tr>
<td>addwc</td>
<td>Add Word With Carry</td>
<td>2-12</td>
</tr>
<tr>
<td>andi</td>
<td>And Immediate</td>
<td>2-13</td>
</tr>
<tr>
<td>andw</td>
<td>And Word</td>
<td>2-14</td>
</tr>
<tr>
<td>b*</td>
<td>Branch On Condition</td>
<td>2-15</td>
</tr>
<tr>
<td>bf*</td>
<td>Branch On Floating Exception</td>
<td>2-17</td>
</tr>
<tr>
<td>call</td>
<td>Call Subroutine</td>
<td>2-18</td>
</tr>
<tr>
<td>calls</td>
<td>Call Supervisor</td>
<td>2-19</td>
</tr>
<tr>
<td>cmpc</td>
<td>Compare Characters</td>
<td>2-20</td>
</tr>
<tr>
<td>cmpd</td>
<td>Compare Double Floating</td>
<td>2-21</td>
</tr>
<tr>
<td>cmpi</td>
<td>Compare Immediate</td>
<td>2-22</td>
</tr>
<tr>
<td>cmpq</td>
<td>Compare Quick</td>
<td>2-23</td>
</tr>
<tr>
<td>cmps</td>
<td>Compare Single Floating</td>
<td>2-24</td>
</tr>
<tr>
<td>cmpw</td>
<td>Compare Word</td>
<td>2-25</td>
</tr>
<tr>
<td>cnvds</td>
<td>Convert Double to Single Floating</td>
<td>2-26</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS (CONTINUED)

movsu  Move Supervisor to User ....................................... 2-60
movsw  Move Single to Word ........................................... 2-61
movus  Move User to Supervisor ....................................... 2-62
movw  Move Word ............................................................. 2-63
movwp  Move Word to Processor
        Register ......................................................... 2-64
movws  Move Word to Single Floating ................................. 2-65
muld  Multiply Double Floating ...................................... 2-66
muls  Multiply Single Floating ....................................... 2-67
mulw  Multiply Word ..................................................... 2-68
mulwu  Multiply Word Unsigned ....................................... 2-69
mulwux  Multiply Word Unsigned
        Extended ....................................................... 2-70
mulwx  Multiply Word Extended ...................................... 2-71
negd  Negate Double Floating ........................................ 2-72
negs  Negate Single Floating ......................................... 2-73
negw  Negate Word ....................................................... 2-74
noop  No Operation ........................................................ 2-75
notq  Not Quick ............................................................ 2-76
notw  Not Word ............................................................. 2-77
ori  Or Immediate .......................................................... 2-78
orw  Or Word ................................................................. 2-79
popw  Pop Word ............................................................. 2-80
pushw  Push Word ........................................................... 2-81
restdn  Restore Registers fn–f7 .................................... 2-82
restur  Restore User Registers ...................................... 2-83
restwn  Restore Registers mn–r12 .................................. 2-84
ret  Return From Subroutine .......................................... 2-85
reti  Return From Interrupt ........................................ 2-86
roti  Rotate Immediate .................................................. 2-87
rotl  Rotate Longword .................................................... 2-88
rotli  Rotate Long Immediate ......................................... 2-89
rotw  Rotate Word ........................................................ 2-90
savedn  Save Registers fn–f7 ....................................... 2-91
saveur  Save User Registers ........................................... 2-92
savewn  Save Registers mn–r14 ...................................... 2-93
scalbd  Scale by Double Floating .................................. 2-94
scalbs  Scale by Single Floating .................................... 2-95
shai  Shift Arithmetic Immediate .................................. 2-96
shal  Shift Arithmetic Longword ..................................... 2-97
shali  Shift Arithmetic Longword
        Immediate ....................................................... 2-98
shaw  Shift Arithmetic Word .......................................... 2-99
shli  Shift Logical Immediate ....................................... 2-100
shll  Shift Logical Longword ......................................... 2-101
shlli  Shift Logical Longword
        Immediate ....................................................... 2-102
shlw  Shift Logical Word ................................................. 2-103
storb  Store Byte .......................................................... 2-104
stord  Store Double Floating ........................................ 2-105
stord  Store Halfword .................................................. 2-106
stors  Store Single Floating ......................................... 2-107
storw  Store Word ........................................................ 2-108
subd  Subtract Double Floating ..................................... 2-109
subi  Subtract Immediate ............................................. 2-110
subq  Subtract Quick .................................................... 2-111
subw  Subtract Single Floating ..................................... 2-112
subw  Subtract Word .................................................... 2-113
subwc  Subtract Word With Carry ................................... 2-114
trapsn  Trap Floating Unordered ................................... 2-115
tsts  Test and Set ...................................................... 2-116
wait  Wait for Interrupt ............................................... 2-117
xor  Exclusive-Or Immediate ......................................... 2-118
xorw  Exclusive-Or Word ............................................... 2-119

APPENDICES

A  ASCII Character Set ................................................. A-1

B  Instruction Summary .............................................. B-1

FIGURES

1-1  Instruction Formats ............................................... 1-2
2-1  Instruction Description Example ................................ 2-3

TABLES

1-1  Memory Addressing Modes ........................................ 1-5
2-1  Alphabetical Listing of the
        Instructions .................................................... 2-2
2-2  Syntax Field Operands ........................................... 2-4
2-3  Operation Field Operands ....................................... 2-4
2-4  Operation Field Operators ...................................... 2-5
2-5  Format Field Operands ........................................... 2-5
2-6  Example Field Operands .......................................... 2-6
2-7  Integer Branch Conditions ...................................... 2-16
2-8  Floating Branch Conditions ..................................... 2-17
A-1  ASCII Character Set .............................................. A-1
A-2  Nonprintable Characters ......................................... A-2
B-1  Functional Instruction Set ....................................... B-2
B-2  Instruction Opcode/Mnemonic
        Summary ........................................................... B-6
B-3  Macro Instruction Code Field ................................... B-7
B-4  Privileged Macro Instruction
        Code Field ........................................................ B-7

ADVANCE INFORMATION
PREFACE

This document describes the CLIPPER instruction set. Component information (e.g., internal register description, programming model, exception processing, bus operation, etc.) is available in the CLIPPER Module Product Description.
CHAPTER ONE
INSTRUCTION FORMATS

This chapter discusses:
• Instruction formats without addresses
• Instruction formats with addresses

1.1 INTRODUCTION
The information encoded in each instruction specifies the operation to be performed, the type and number of operands to use, and the location of the operands. These operands can be located in a register or in memory. For example, the loadb instruction contains operands that reference memory and a register. If an operand is located in memory, the instruction must calculate the address of the operand according to the address mode specified in the instruction format.

The immediate and quick instructions use an operand encoded within the instruction for fast efficient operation.

All instructions are constructed in multiples of halfwords called parcels (see the general instruction format below). The size of instructions varies from one to four parcels.
Figure 1-1 shows the instruction formats used in the CLIPPER architecture. Notice that the formats are divided into two main categories, non-memory referencing instructions (NO ADDRESS) and memory referencing instructions (WITH ADDRESS).

### INSTRUCTION FORMATS — NO ADDRESS

**REGISTER**
```
15  8  7  4  3  0
    OPCODE  R1   R2
```

**QUICK**
```
15  8  7  4  3  0
    OPCODE  QUICK  R2
```

**16-BIT IMMEDIATE**
```
15  8  7  4  3  0
    OPCODE  1 0 1  1  R2

S  IMMEDIATE
31  30                      16
```

**32-BIT IMMEDIATE**
```
15  8  7  4  3  0
    OPCODE  0 0 1  1  R2

S  IMMEDIATE LOW
47  46
```

**CONTROL**
```
15  8  7  0
    OPCODE  BYTE
```

**MACRO**
```
15  9  8  7  6  0
    OPCODE  P  0  0  0  CODE

0 0 0 0 0 0 0 0  R1  R2
31  24  23  20  19  16
```

### INSTRUCTION FORMATS — WITH ADDRESS

**RELATIVE**
```
15  8  7  4  3  0
    OPCODE  0   R1  R2
```

**RELATIVE PLUS 12-BIT DISPLACEMENT**
```
15  8  7  4  3  0
    OPCODE  1  0 1  0  R1

S  DISPLACEMENT  R2
31  20  19          16
```

**RELATIVE PLUS 32-BIT DISPLACEMENT**
```
15  8  7  4  3  0
    OPCODE  1 0 1  1  0

S  DISPLACEMENT LOW
63  62
```

**PC-RELATIVE PLUS 16-BIT DISPLACEMENT**
```
15  8  7  4  3  0
    OPCODE  1 1 0  1  R2

S  DISPLACEMENT
31  30
```

**PC-RELATIVE PLUS 32-BIT DISPLACEMENT**
```
15  8  7  4  3  0
    OPCODE  1 0 0  1  R2

S  DISPLACEMENT LOW
47  46
```

**16-BIT ABSOLUTE**
```
15  8  7  4  3  0
    OPCODE  1 1 0  1  1

S  ADDRESS
31  30
```

**32-BIT ABSOLUTE**
```
15  8  7  4  3  0
    OPCODE  1 0 0  1  1

S  ADDRESS LOW
47  46
```

**PC INDEXED**
```
15  8  7  4  3  0
    OPCODE  1 1 0  1  0

0 0 0 0 0 0 0 0  RX  R2
31  24  23  20  19  16
```

**RELATIVE INDEXED**
```
15  8  7  4  3  0
    OPCODE  1 1 1  0  R1

0 0 0 0 0 0 0 0  RX  R2
31  24  23  20  19  16
```

*Figure 1-1 Instruction Formats*
1.2 INSTRUCTION FORMATS — NO ADDRESS

1.2.1 REGISTER
The Register format is used for most instructions that take just one or two register arguments.

The opcode specifies the interpretation of the R1 and R2 fields. Usually the R1 field contains the source operand register number, and R2 contains the destination operand register number. For example, in the `movsw` instruction, the R1 field contains the number of the single-precision floating-point register containing the source operand, and the R2 field contains the number of the general register in which to store the result.

1.2.2 QUICK
The Quick format encodes constant, 4-bit unsigned source operands directly in the instruction. The quick value is always zero-filled at the left before use.

1.2.3 16-BIT IMMEDIATE
The 16-bit Immediate format encodes a 16-bit source operand constant directly in the instruction. The immediate value is always sign-extended before use.
1.2.4 32-BIT IMMEDIATE
The 32-bit Immediate format encodes a constant, 32-bit source operand directly in the instruction.

1.2.5 CONTROL
The Control format encodes up to 8 bits of a constant value that is used by a few special instructions. For example, the byte operand specifies the system call number in the calls instruction.

1.2.6 MACRO
The Macro format is used by those instructions that are implemented in MI ROM rather than directly in the hardware. The P bit in the opcode, bit 9 of the format instruction parcel, selects a privileged macro.
1.3 INSTRUCTION FORMATS — WITH ADDRESS

The rest of the instruction formats specify an address operand and a register operand. Several address formats, or modes, are provided to support typical high-level language operations. The address mode is selected first by the opcode (bit 8 of the first instruction parcel), and if necessary, by the AM field (bits 7:4 of the first instruction parcel). Displacements and absolute addresses are always sign extended.

The address modes used in the memory referencing instructions are summarized in Table 1-1 and explained on the following pages.

<table>
<thead>
<tr>
<th>Memory Addressing Mode</th>
<th>Address Formation</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative</td>
<td>Address $\leftarrow (R1)$</td>
<td>1-7</td>
</tr>
<tr>
<td>Relative with 12-bit displacement</td>
<td>Address $\leftarrow (R1) + 12$-bit displacement</td>
<td>1-7</td>
</tr>
<tr>
<td>Relative with 32-bit displacement</td>
<td>Address $\leftarrow (R1) + 32$-bit displacement</td>
<td>1-8</td>
</tr>
<tr>
<td>16-bit Absolute</td>
<td>Address $\leftarrow 16$-bit displacement</td>
<td>1-8</td>
</tr>
<tr>
<td>32-bit Absolute</td>
<td>Address $\leftarrow 32$-bit displacement</td>
<td>1-9</td>
</tr>
<tr>
<td>PC Relative with 16-bit displacement</td>
<td>Address $\leftarrow (PC) + 16$-bit displacement</td>
<td>1-9</td>
</tr>
<tr>
<td>PC Relative with 32-bit displacement</td>
<td>Address $\leftarrow (PC) + 32$-bit displacement</td>
<td>1-10</td>
</tr>
<tr>
<td>Relative Indexed</td>
<td>Address $\leftarrow (R1) + (RX)$</td>
<td>1-10</td>
</tr>
<tr>
<td>PC Indexed</td>
<td>Address $\leftarrow (PC) + (RX)$</td>
<td>1-11</td>
</tr>
</tbody>
</table>

Notes:
All displacements are signed.
PC addresses the first parcel of the current instruction.
RX is any general register containing the index modifying the effect of the source register.

1.3.1 RELATIVE
The Relative format uses the address in a register (R1) to compute an address.

[Diagram of instruction format with labels]
### 1.3.2 RELATIVE WITH 12-BIT DISPLACEMENT

The Relative Plus 12-bit Displacement format uses the address in a register (R1), plus a signed 12-bit displacement, to compute an address. The displacement is sign-extended to 32 bits before the address calculation.

![Diagram](image)

### 1.3.3 RELATIVE WITH 32-BIT DISPLACEMENT

The Relative Plus 32-bit Displacement format uses the address in a register (R1), plus a signed 32-bit displacement, to compute an address.

![Diagram](image)
1.3.4 16-BIT ABSOLUTE
The 16-bit Absolute format uses the address in register (R2). The address is sign-extended to 32 bits before being used. Because the address field is signed, the range of address that can be accessed with this format is: $0xffff8000 \leq \text{address} \leq 0xffffffff$.

1.3.5 32-BIT ABSOLUTE
The 32-bit Absolute format uses the signed 32-bit displacement portion of the instruction as an address.

1.3.6 PC RELATIVE WITH 16-BIT DISPLACEMENT
The 16-bit PC Relative format adds a signed 16-bit displacement to the contents of the Program Counter (PC) to compute an address.
1.3.7 PC RELATIVE WITH 32-BIT DISPLACEMENT
The 32-bit PC Relative format adds a signed 32-bit displacement to the contents of the Program Counter (PC) to compute the address.

EXAMPLE INSTRUCTION: call sp., far (pc)

<table>
<thead>
<tr>
<th>OPCODE</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1</td>
<td></td>
</tr>
</tbody>
</table>

DISPLACEMENT LOW

S
DISPLACEMENT HIGH

ADDRESS FORMATION

31
ADDRESS FROM PROGRAM COUNTER

+ 0

SIGNED DISPLACEMENT

31
ADDRESS

1.3.8 RELATIVE Indexed
The Relative indexed format uses the address in a register (R1), plus the contents of an index register (RX), to compute an address.

EXAMPLE INSTRUCTION: loadbu [r3] (fp), r1

<table>
<thead>
<tr>
<th>OPCODE</th>
<th>RX</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 0 0 0 0 0 0 0

ADDRESS FORMATION

31
ADDRESS FROM REGISTER

+ 0

ADDRESS FROM REGISTER

31
ADDRESS
1.3.9 PC INDEXED
The PC Indexed format adds the contents of an index register (RX) to the contents of the PC to compute an address.

EXAMPLE INSTRUCTION
storeh r9. [r3] (pc)

INSTRUCTION FORMAT

ADDRESS FORMATION

OPCODE  1 1 1 0 1 0 0 0 0
0 0 0 0 0 0 0 0 RX R2

ADDRESS FROM PROGRAM COUNTER

+ 0

ADDRESS FROM REGISTER

ADDRESS
CHAPTER TWO
INSTRUCTION SET

This chapter contains detailed descriptions of the CLIPPER instructions. The instructions are listed in alphabetical order.

The CLIPPER instruction set contains 101 basic instructions and 67 macro instructions. This reduced instruction set is especially useful to high-level language optimizing compilers.

Memory access in a CLIPPER environment is through load/store instructions to minimize bus traffic and memory-dependent execution delays. Data operations are performed in registers.

There are two units in the CLIPPER CPU that execute instructions: the Integer Execution Unit (IEU) and the Floating-Point Execution Unit (FPU). The integer instructions (with the exception of integer multiplies and divides) are executed by the IEU. Floating-point instructions (and the integer multiplies and divides) are executed by the FPU.

The basic instructions are fetched from main memory, through the instruction cache, decoded and then executed, either by the IEU or by the FPU. The macro instructions are executed from the Macro Instruction ROM (MI ROM).

A macro instruction opcode is actually a reference to a sequence of instructions in the MI ROM. When a macro instruction is fetched, execution control is switched to the MI ROM and the macrocoded sequence of the macro instruction is executed.

The instructions are listed in Table 2-1.
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Page</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>addd</td>
<td>Add Double Floating</td>
<td>2-7</td>
<td>movwp</td>
<td>Move Word to Processor Register</td>
<td>2-64</td>
</tr>
<tr>
<td>addi</td>
<td>Add Immediate</td>
<td>2-8</td>
<td>movws</td>
<td>Move Word to Single Floating</td>
<td>2-65</td>
</tr>
<tr>
<td>addq</td>
<td>Add Quick</td>
<td>2-9</td>
<td>muld</td>
<td>Multiply Double Floating</td>
<td>2-66</td>
</tr>
<tr>
<td>addds</td>
<td>Add Single Floating</td>
<td>2-10</td>
<td>muls</td>
<td>Multiply Single Floating</td>
<td>2-67</td>
</tr>
<tr>
<td>addw</td>
<td>Add Word</td>
<td>2-11</td>
<td>mulw</td>
<td>Multiply Word</td>
<td>2-68</td>
</tr>
<tr>
<td>addwc</td>
<td>Add Word with Carry</td>
<td>2-12</td>
<td>mulwu</td>
<td>Multiply Word Unsigned</td>
<td>2-69</td>
</tr>
<tr>
<td>andi</td>
<td>And Immediate</td>
<td>2-13</td>
<td>mulwux</td>
<td>Multiply Word Unsigned Extended</td>
<td>2-70</td>
</tr>
<tr>
<td>andw</td>
<td>And Word</td>
<td>2-14</td>
<td>mulwx</td>
<td>Multiply Word Extended</td>
<td>2-71</td>
</tr>
<tr>
<td>b*</td>
<td>Branch on Condition</td>
<td>2-15</td>
<td>negd</td>
<td>Negate Double Floating</td>
<td>2-72</td>
</tr>
<tr>
<td>bf*</td>
<td>Branch on Floating Exception</td>
<td>2-17</td>
<td>negs</td>
<td>Negate Single Floating</td>
<td>2-73</td>
</tr>
<tr>
<td>call</td>
<td>Call Subroutine</td>
<td>2-18</td>
<td>negw</td>
<td>Negate Word</td>
<td>2-74</td>
</tr>
<tr>
<td>calls</td>
<td>Call Supervisor</td>
<td>2-19</td>
<td>noo</td>
<td>No Operation</td>
<td>2-75</td>
</tr>
<tr>
<td>cmpc</td>
<td>Compare Characters</td>
<td>2-20</td>
<td>nq</td>
<td>Not Quick</td>
<td>2-76</td>
</tr>
<tr>
<td>cmpd</td>
<td>Compare Double Floating</td>
<td>2-21</td>
<td>nw</td>
<td>Not Word</td>
<td>2-77</td>
</tr>
<tr>
<td>cmpi</td>
<td>Compare Immediate</td>
<td>2-22</td>
<td>Ori</td>
<td>Or Immediate</td>
<td>2-78</td>
</tr>
<tr>
<td>cmpq</td>
<td>Compare Quick</td>
<td>2-23</td>
<td>ow</td>
<td>Or Word</td>
<td>2-79</td>
</tr>
<tr>
<td>cmps</td>
<td>Compare Single Floating</td>
<td>2-24</td>
<td>popw</td>
<td>Pop Word</td>
<td>2-80</td>
</tr>
<tr>
<td>cmpw</td>
<td>Compare Word</td>
<td>2-25</td>
<td>pushw</td>
<td>Push Word</td>
<td>2-81</td>
</tr>
<tr>
<td>cnvds</td>
<td>Convert Double to Single Floating</td>
<td>2-26</td>
<td>restdn</td>
<td>Restore Registers fn: 0 ≤ n ≤ 7</td>
<td>2-82</td>
</tr>
<tr>
<td>cnvdw</td>
<td>Convert Double to Word</td>
<td>2-27</td>
<td>restur</td>
<td>Restore User Registers</td>
<td>2-83</td>
</tr>
<tr>
<td>cnvrdw</td>
<td>Convert Rounding Double to Word</td>
<td>2-28</td>
<td>restwr</td>
<td>Restore Registers m: 0 ≤ n ≤ 12</td>
<td>2-84</td>
</tr>
<tr>
<td>cnvrsw</td>
<td>Convert Rounding Single to Word</td>
<td>2-29</td>
<td>ret</td>
<td>Return From Subroutine</td>
<td>2-85</td>
</tr>
<tr>
<td>cnvsd</td>
<td>Convert Single to Floating</td>
<td>2-30</td>
<td>rot</td>
<td>Return From Interrupt</td>
<td>2-86</td>
</tr>
<tr>
<td>cnvsw</td>
<td>Convert Single Floating</td>
<td>2-31</td>
<td>roti</td>
<td>Rotate Immediate</td>
<td>2-87</td>
</tr>
<tr>
<td>cnvtdw</td>
<td>Convert Truncating Double to Word</td>
<td>2-32</td>
<td>rotw</td>
<td>Rotate Longword</td>
<td>2-88</td>
</tr>
<tr>
<td>cnvtsw</td>
<td>Convert Truncating Single to Word</td>
<td>2-33</td>
<td>rotw</td>
<td>Rotate Long Immediate</td>
<td>2-89</td>
</tr>
<tr>
<td>cnvw</td>
<td>Convert Word to Double Floating</td>
<td>2-34</td>
<td>rotw</td>
<td>Rotate Word</td>
<td>2-90</td>
</tr>
<tr>
<td>cnwvs</td>
<td>Convert Word to Single Floating</td>
<td>2-35</td>
<td>savedn</td>
<td>Save Registers fn: 0 ≤ n ≤ 7</td>
<td>2-91</td>
</tr>
<tr>
<td>divd</td>
<td>Divide Double Floating</td>
<td>2-36</td>
<td>saveur</td>
<td>Save User Registers</td>
<td>2-92</td>
</tr>
<tr>
<td>divs</td>
<td>Divide Single Floating</td>
<td>2-37</td>
<td>savewn</td>
<td>Save Registers m: 0 ≤ n ≤ 12</td>
<td>2-93</td>
</tr>
<tr>
<td>divw</td>
<td>Divide Word</td>
<td>2-38</td>
<td>scalebd</td>
<td>Scale by Double Floating</td>
<td>2-94</td>
</tr>
<tr>
<td>divwu</td>
<td>Divide Word Unsigned</td>
<td>2-39</td>
<td>scalebs</td>
<td>Scale by Single Floating</td>
<td>2-95</td>
</tr>
<tr>
<td>initc</td>
<td>Initialize Characters</td>
<td>2-40</td>
<td>shai</td>
<td>Shift Arithmetic Immediate</td>
<td>2-96</td>
</tr>
<tr>
<td>loada</td>
<td>Load Address</td>
<td>2-41</td>
<td>shal</td>
<td>Shift Arithmetic Longword</td>
<td>2-97</td>
</tr>
<tr>
<td>loadb</td>
<td>Load Byte</td>
<td>2-42</td>
<td>shaw</td>
<td>Shift Arithmetic Longword Immediate</td>
<td>2-98</td>
</tr>
<tr>
<td>loadbu</td>
<td>Load Byte Unsigned</td>
<td>2-43</td>
<td>shaw</td>
<td>Shift Arithmetic Word</td>
<td>2-99</td>
</tr>
<tr>
<td>loadd</td>
<td>Load Double Floating</td>
<td>2-44</td>
<td>shl</td>
<td>Shift Logical Immediate</td>
<td>2-100</td>
</tr>
<tr>
<td>loadfs</td>
<td>Load Floating Status</td>
<td>2-45</td>
<td>shl</td>
<td>Shift Logical Longword</td>
<td>2-101</td>
</tr>
<tr>
<td>loadh</td>
<td>Load Halfword</td>
<td>2-46</td>
<td>shll</td>
<td>Shift Logical Longword Immediate</td>
<td>2-102</td>
</tr>
<tr>
<td>loadhu</td>
<td>Load Halfword Unsigned</td>
<td>2-47</td>
<td>shlw</td>
<td>Shift Logical Word</td>
<td>2-103</td>
</tr>
<tr>
<td>loadi</td>
<td>Load Immediate</td>
<td>2-48</td>
<td>storb</td>
<td>Store Byte</td>
<td>2-104</td>
</tr>
<tr>
<td>loadq</td>
<td>Load Quick</td>
<td>2-49</td>
<td>stord</td>
<td>Store Double Floating</td>
<td>2-105</td>
</tr>
<tr>
<td>loads</td>
<td>Load Single Floating</td>
<td>2-50</td>
<td>storh</td>
<td>Store Halfword</td>
<td>2-106</td>
</tr>
<tr>
<td>loadw</td>
<td>Load Word</td>
<td>2-51</td>
<td>stors</td>
<td>Store Single Floating</td>
<td>2-107</td>
</tr>
<tr>
<td>mod</td>
<td>Modulus Word</td>
<td>2-52</td>
<td>storw</td>
<td>Store Word</td>
<td>2-108</td>
</tr>
<tr>
<td>modw</td>
<td>Modulus Word Unsigned</td>
<td>2-53</td>
<td>subd</td>
<td>Subtract Double Floating</td>
<td>2-109</td>
</tr>
<tr>
<td>movc</td>
<td>Move Characters</td>
<td>2-54</td>
<td>subi</td>
<td>Subtract Immediate</td>
<td>2-110</td>
</tr>
<tr>
<td>movd</td>
<td>Move Double Floating</td>
<td>2-55</td>
<td>subq</td>
<td>Subtract Quick</td>
<td>2-111</td>
</tr>
<tr>
<td>movdl</td>
<td>Move Double Floating to Longword</td>
<td>2-56</td>
<td>subs</td>
<td>Subtract Single Floating</td>
<td>2-112</td>
</tr>
<tr>
<td>movld</td>
<td>Move Longword to Double</td>
<td>2-57</td>
<td>subw</td>
<td>Subtract Word</td>
<td>2-113</td>
</tr>
<tr>
<td>movpw</td>
<td>Move Processor Register to Word</td>
<td>2-58</td>
<td>subwc</td>
<td>Subtract Word with Carry</td>
<td>2-114</td>
</tr>
<tr>
<td>movs</td>
<td>Move Single Floating</td>
<td>2-59</td>
<td>trapfn</td>
<td>Trap On Floating Unordered</td>
<td>2-115</td>
</tr>
<tr>
<td>movsu</td>
<td>Move Supervisor to User</td>
<td>2-60</td>
<td>tsts</td>
<td>Test and Set</td>
<td>2-116</td>
</tr>
<tr>
<td>movsw</td>
<td>Move Single to Word</td>
<td>2-61</td>
<td>wait</td>
<td>Wait for Interrupt</td>
<td>2-117</td>
</tr>
<tr>
<td>movus</td>
<td>Move User to Supervisor</td>
<td>2-62</td>
<td>xor</td>
<td>Exclusive-OR Immediate</td>
<td>2-118</td>
</tr>
<tr>
<td>movw</td>
<td>Move Word</td>
<td>2-63</td>
<td>xorw</td>
<td>Exclusive-OR Word</td>
<td>2-119</td>
</tr>
</tbody>
</table>
The format of each instruction is described in detail in the following pages. Figure 2-1 below illustrates the information presented.

Instruction Name — mnemonic and descriptive name of the instruction.

Assembler Syntax — Bold means enter item exactly as shown. Italic means substitute correct value. Punctuation must be included as shown. See Table 2-2 for operands.

Description — the text that describes the operation of the instruction, functionally equivalent to Operation.

Operation — this is an equation describing the operation of the instruction. Tables 2-3 and 2-4 list operands used. When setting PSW flags, the value of the expression to the right of the “←” replaces the flag.

Exceptions — conditions under which an exception may occur. Many traps have trap enables in the PSW or SSW. See Chapter 5 for details.

Instruction Format — the class, bit patterns, and fields of the instruction. See Table 2-5 for operand and operator descriptions.

Instruction Example — an example of the source statement entered. See Table 2-6 for a description of the terms used.

Figure 2-1 Instruction Description Example
Table 2-2 lists meanings of the operands used in the Syntax field of the instruction descriptions. The first character of the notation column specifies the operand’s type and size:

\[ b = \text{byte} \quad w = \text{word} \quad s = \text{single-precision floating-point} \]
\[ h = \text{halfword} \quad l = \text{longword} \quad d = \text{double-precision floating-point} \]
\[ p = \text{processor register} \]

The second character of the notation column specifies the operand’s field within the instruction and its location in the machine (immediate value, register, memory, etc.):

\[ a = \text{R1} \quad q = \text{quick} \quad a = \text{address} \]
\[ 2 = \text{R2} \quad i = \text{immediate} \quad b = \text{byte} \]

Table 2-2  Syntax Field Operands

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ba</td>
<td>Byte address (32-bit address on byte boundary)</td>
</tr>
<tr>
<td>bb</td>
<td>Unsigned byte data (0 — 256 decimal)</td>
</tr>
<tr>
<td>da</td>
<td>Double floating address (32-bit address on doubleword boundary)</td>
</tr>
<tr>
<td>d1</td>
<td>Double floating register operand 1 (f0 — f7)</td>
</tr>
<tr>
<td>d2</td>
<td>Double floating register operand 2 (f0 — f7)</td>
</tr>
<tr>
<td>ha</td>
<td>Halfword address (32 bit address on halfword boundary)</td>
</tr>
<tr>
<td>l1</td>
<td>Longword register operand 1 (even register pair r0,r1 — r14, r15)</td>
</tr>
<tr>
<td>l2</td>
<td>Longword register operand 2 (even register pair r0,r1 — r14, r15)</td>
</tr>
<tr>
<td>p1</td>
<td>Processor register operand 1 (0 = psw, 1 = ssw)</td>
</tr>
<tr>
<td>sa</td>
<td>Single floating address (32-bit address on word boundary)</td>
</tr>
<tr>
<td>s1</td>
<td>Single floating register operand 1 (f0 — f7)</td>
</tr>
<tr>
<td>s2</td>
<td>Single floating register operand 2 (f0 — f7)</td>
</tr>
<tr>
<td>wa</td>
<td>Word address (32 bit address on word boundary)</td>
</tr>
<tr>
<td>wi</td>
<td>Signed 16-bit immediate data (sign extended to 32-bits) or signed 32-bit immediate data</td>
</tr>
<tr>
<td>wq</td>
<td>Unsigned 4-bit quick data (zero extended to 32-bits)</td>
</tr>
<tr>
<td>w1</td>
<td>General register operand 1 (r0 — r15)</td>
</tr>
<tr>
<td>w2</td>
<td>General register operand 2 (r0 — r15)</td>
</tr>
</tbody>
</table>

Tables 2-2 and 2-3 list the meanings of the operands used in the Operation field of each description. Table 2-4 lists the meanings of the operators used in the Operation field of each description.

Table 2-3  Operation Field Operands

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>PSW carry out flag</td>
</tr>
<tr>
<td>FP dest</td>
<td>Original floating-point destination</td>
</tr>
<tr>
<td>FP PC</td>
<td>Floating-point program counter</td>
</tr>
<tr>
<td>N</td>
<td>PSW integer negative flag</td>
</tr>
<tr>
<td>n</td>
<td>Number (usually register number)</td>
</tr>
<tr>
<td>PC</td>
<td>Program counter</td>
</tr>
<tr>
<td>r0 — r15</td>
<td>General registers 0 through 15</td>
</tr>
<tr>
<td>V</td>
<td>PSW integer overflow flag</td>
</tr>
<tr>
<td>Z</td>
<td>PSW integer zero flag</td>
</tr>
</tbody>
</table>
### Table 2-4  Operation Field Operators

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>rot</td>
<td>Rotate operator</td>
</tr>
<tr>
<td>sha</td>
<td>Shift arithmetic operator</td>
</tr>
<tr>
<td>shl</td>
<td>Shift logical operator</td>
</tr>
<tr>
<td>+</td>
<td>Add operator</td>
</tr>
<tr>
<td>-</td>
<td>Subtract operator or negate unary operator</td>
</tr>
<tr>
<td>×</td>
<td>Multiply operator</td>
</tr>
<tr>
<td>÷</td>
<td>Divide operator</td>
</tr>
<tr>
<td>mod</td>
<td>Modulus operator</td>
</tr>
<tr>
<td>~</td>
<td>Complement operator</td>
</tr>
<tr>
<td>=</td>
<td>Equal operator</td>
</tr>
<tr>
<td>≠</td>
<td>Not equal operator</td>
</tr>
<tr>
<td>←</td>
<td>Assignment operator</td>
</tr>
<tr>
<td>&amp;</td>
<td>AND logical operator</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>⊕</td>
<td>Exclusive-OR logical operator</td>
</tr>
<tr>
<td>( )</td>
<td>Contents of operand within</td>
</tr>
<tr>
<td>[ ]</td>
<td>Separators used to indicate value inside in a unit</td>
</tr>
<tr>
<td>&lt; &gt;</td>
<td>Value inside symbols indicates bit(s) of a register</td>
</tr>
<tr>
<td>:</td>
<td>Indicates a range of values</td>
</tr>
</tbody>
</table>

Tables 2-2 and 2-5 list meanings of the operands used in the Format field of the instruction descriptions.

### Table 2-5  Format Field Operands

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr mode</td>
<td>A four bit code for the type of addressing used</td>
</tr>
<tr>
<td>cond</td>
<td>Condition code for branch conditions (see Tables 2-7 and 2-8)</td>
</tr>
<tr>
<td>high</td>
<td>Most-significant portion of an address or value</td>
</tr>
<tr>
<td>low</td>
<td>Least-significant portion of an address or value</td>
</tr>
<tr>
<td>R1</td>
<td>Register field 1 within the instruction format</td>
</tr>
<tr>
<td>R2</td>
<td>Register field 2 within the instruction format</td>
</tr>
</tbody>
</table>
Table 2-6 lists the assembler instruction operands used in the Example field of the instruction descriptions. Assembler instruction operands are generally given in source, destination order, independent of their positions in the machine representation.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0 : r15</td>
<td>General registers. Even general registers address longword operands. sp, fp, and ap are synonyms for r15, r14, and r13. Not to be confused with R1 or R2, which are register fields within an instruction.</td>
</tr>
<tr>
<td>f0 : f7</td>
<td>Floating registers. Each register may contain either a single or double floating value</td>
</tr>
<tr>
<td>psw, ssw</td>
<td>Processor registers 0 and 1</td>
</tr>
<tr>
<td>$n</td>
<td>Quick, byte or immediate value (decimal)</td>
</tr>
<tr>
<td>$O \times n</td>
<td>Quick, byte or immediate value (hexadecimal)</td>
</tr>
<tr>
<td>n</td>
<td>Absolute address</td>
</tr>
<tr>
<td>n(m)</td>
<td>Relative or relative with displacement address. n may be 0 or absent.</td>
</tr>
<tr>
<td><a href="rn">rx</a></td>
<td>Relative indexed address</td>
</tr>
<tr>
<td>n(pc)</td>
<td>PC relative address</td>
</tr>
<tr>
<td>. ± n</td>
<td>PC relative address</td>
</tr>
<tr>
<td><a href="pc">rx</a></td>
<td>PC indexed address</td>
</tr>
<tr>
<td>#</td>
<td>Indicates a comment field</td>
</tr>
</tbody>
</table>
Add Double Floating

Syntax: \texttt{add} \ d1,d2

Description: Add the double-precision contents of floating register d1 to the double-precision contents of floating register d2 and put the result in d2. On a trap, the PC and the original value in d2 can be obtained by using the \texttt{loadfs} instruction.

Operation: \begin{align*}
d2 & \leftarrow (d2) + (d1) \\
FX & \leftarrow \text{floating inexact result} \\
FU & \leftarrow \text{floating underflow} \\
FV & \leftarrow \text{floating overflow} \\
FI & \leftarrow \text{floating invalid}
\end{align*}

Traps: Floating inexact result
Floating invalid operation
Floating overflow
Floating underflow

Format: Register

\begin{center}
\begin{tabular}{|cccccc|}
\hline
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
\hline
\end{tabular}
\end{center}

Example: \begin{align*}
\text{loadd} & \quad \text{dvalue}, f0 & \# \text{Load double floating value at dvalue into f0} \\
\text{loadd} & \quad (r2), f1 & \# \text{Load double floating value at (r2) into f1} \\
\text{add} & \quad f0, f1 & \# \text{Add floating regs f0 and f1 and put the result in f1}
\end{align*}
ADDI

Add Immediate

Syntax: \texttt{addi \ texttt{wi,w2}}

Description: Add the immediate value \texttt{wi} to the contents of general register \texttt{w2} and put the result in \texttt{w2}. The operands may be signed or unsigned integers. Overflow is set if the operands (which are treated as signed integers) have the same sign and the result has the opposite sign.

Operation: \begin{align*}
\texttt{w2} & \leftarrow (\texttt{w2} + \texttt{wi}) \\
\texttt{N} & \leftarrow (\texttt{w2} <_{31} >) \\
\texttt{Z} & \leftarrow (\texttt{w2}) = 0 \\
\texttt{V} & \leftarrow \text{integer overflow} \\
\texttt{C} & \leftarrow \text{integer carry out}
\end{align*}

Traps: none

Format: Immediate

If $-2^{15} \leq \texttt{wi} \leq +2^{15} - 1$, this format is used:

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|}
\hline
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 1 \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{|c|c|}
\hline
31 & 30 \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|}
\hline
16 & 15 & 8 & 7 & 4 & 3 \\
\hline
\texttt{S} & \texttt{wi} \\
\hline
\end{tabular}
\end{center}

If $+2^{15} \leq \texttt{wi} \leq +2^{31} - 1$ or $-2^{15} \leq \texttt{wi} \leq -2^{15} - 1$, this format is used:

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|}
\hline
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 1 \\
\hline
\texttt{S} & \texttt{wi low} \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|}
\hline
32 & 47 & 46 & 16 & \\
\hline
\texttt{wi high} \\
\hline
\end{tabular}
\end{center}

Example: \begin{align*}
\texttt{addi} & \quad \texttt{\$180,r1} \quad \# \ Add \ 16\text{-}bit \ value \ to \ r1 \\
\texttt{addi} & \quad \texttt{\$99999,r0} \quad \# \ Add \ 32\text{-}bit \ value \ to \ r0 \\
\texttt{addi} & \quad \texttt{\$-1,r2} \quad \# \ Add \ 0xffff \ to \ r2
\end{align*}
ADDQ

Add Quick

Syntax: \textbf{addq} \ wq,w2

Description: Add the unsigned quick value \( wq \) to the contents of general register \( w2 \) and put the result in \( w2 \). The contents of \( w2 \) may be a signed or unsigned integer. Overflow is set if the contents of \( w2 \) (which is treated as a signed integer) is positive and the result is negative.

Operation: \( w2 \leftarrow (w2) + wq \)
\( N \leftarrow (w2 < 31 >) \)
\( Z \leftarrow (w2) = 0 \)
\( V \leftarrow \) integer overflow
\( C \leftarrow \) integer carry out

Traps: none

Format: Quick

\begin{center}
\begin{tabular}{cccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\hline
& \text{wq} & \text{w2}
\end{tabular}
\end{center}

Example: \textbf{addq} \ $4,r1$ \ # Add 4 to (r1)
**ADD**

Add Single Floating

**Syntax:**

```
adds s1, s2
```

**Description:**

Add the single-precision contents of floating register s1 to the single-precision contents of floating register s2 and put the result in s2. On a trap, the PC and the original value in s2 can be obtained by using the **loadfs** instruction.

**Operation:**

```
s2 <- (s2) + (s1)
FX <- floating inexact result
FU <- floating underflow
FV <- floating overflow
FI <- floating invalid
```

**Traps:**

- Floating inexact result
- Floating invalid operation
- Floating overflow
- Floating underflow

**Format:**

Register

```
  15  8  7  4  3  0
  0  0  1  0  0  0  0  s1  s2
```

**Example:**

```
loads $value1,f0  # Load single floating value at $value1 into f0
loads (r3),f2     # Load single floating value at (r3) into f2
adds f2,f0        # Add floating regs f0 and f2 and put the result in f0
```
ADDW

Add Word

Syntax: \texttt{addw w1,w2}

Description: Add the contents of general register \( w1 \) to the contents of general register \( w2 \) and put the result in \( w2 \). Operands may be signed or unsigned integers. Overflow is set if the operands (which are treated as signed integers) have the same sign and the result has the opposite sign.

Operation: \( w2 \leftarrow (w2) + (w1) \)
\( N \leftarrow (w2 < 31 >) \)
\( Z \leftarrow (w2 = 0) \)
\( V \leftarrow \text{integer overflow} \)
\( C \leftarrow \text{integer carry out} \)

Traps: none

Format: Register

\[
\begin{array}{cccccc}
  & & & & & \\
  & 1 & 0 & 0 & 0 & 0 & 0 & 0 & w1 & w2 & 0 & 3 & 4 & 7 & 8 & 15
\end{array}
\]

Example: \texttt{addw r12,r0}  # Add contents of \( r12 \) and \( r0 \), put the result in \( r0 \)
ADDWC

Add Word With Carry

Syntax:

```
addwc  w1, w2
```

Description:
Add the contents of general register w1 and the carry flag to the contents of general register w2 and put the result in w2. Operands may be signed or unsigned integers.

Operation:

```
w2 ← (w2) + (w1) + C
N ← (w2[31])
Z ← (w2) = 0
V ← integer overflow
C ← integer carry out
```

Traps:

none

Format:

Register

```
  1 0 0 1 0 0 0 0 | w1 |
  1                | w2 |
```

Example:
Assume (r0,r1) and (r4,r5) contain two longwords and the operation to be performed is:

```
(r4,r5) ← (r4,r5) + (r0,r1).
```

```
addw r0, r4       # Add low words, get carry
addwc r1, r5      # Add high words using carry
```
Syntax: \texttt{andi \ wi, w2}

Description: Bitwise AND the immediate value \( \text{wi} \) with the contents of general register \( \text{w2} \) and put the result in \( \text{w2} \).

Operation:
\begin{align*}
\text{w2} & \leftarrow (\text{w2}) \& \text{wi} \\
N & \leftarrow (\text{w2} < 31) \\
Z & \leftarrow (\text{w2}) = 0 \\
V & \leftarrow 0 \\
C & \leftarrow 0
\end{align*}

Traps: none

Format: Immediate

- If \(-2^{15} \leq \text{wi} \leq +2^{15} - 1\), this format is used:

\begin{center}
\begin{tabular}{c|c|c|c|c|c}
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & w2 \\
S & wi \\
31 & 30 & 16
\end{tabular}
\end{center}

- If \(+2^{15} \leq \text{wi} \leq +2^{31} - 1\) or \(-2^{15} \leq \text{wi} \leq -2^{31} - 1\), this format is used:

\begin{center}
\begin{tabular}{c|c|c|c|c|c|c|c|c|c|c|c|c|c}
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & w2 \\
S & wi low \\
47 & 46 & 32
\end{tabular}
\end{center}

Examples:

Assume \( \text{r8} \) contains 0x001100ff.

\begin{verbatim}
andi $0xffff,r8  # AND 32-bit value with r8
\end{verbatim}

The result put in \( \text{r8} \) is 0x000000ff.

Assume \( \text{r2} \) contains 0xffff0000.

\begin{verbatim}
andi $023,r2      # AND 16-bit value with r2
\end{verbatim}

The result put in \( \text{r2} \) is 0x00000000.
ANDW

And Word

Syntax: \texttt{andw \ w1,w2}

Description: Bitwise AND the contents of general register \(w1\) with the contents of general register \(w2\) and put the result in \(w2\).

Operation: \begin{align*}
  w2 &\leftarrow (w2) \& (w1) \\
  N &\leftarrow (w2 < 31) \\
  Z &\leftarrow (w2) = 0 \\
  V &\leftarrow 0 \\
  C &\leftarrow 0
\end{align*}

Traps: none

Format: Register

\begin{verbatim}
+-------+-------+-------+
| 15    |  8    |  7    |  4    |  3    |  0    |
| 1 0 0 0 1 0 0 0 | w1    | w2    |
+-------+-------+-------+
\end{verbatim}

Example: Assume \(r2\) contains \texttt{0x7788ffff} and \(r3\) contains \texttt{0xffff0000}.

\begin{verbatim}
andw \ r2,r3 # AND (r2) with (r3), put result in r3
\end{verbatim}

The result put in \(r3\) is \texttt{0x77880000}. 
Branch On Condition

Syntax: \[ b^* \quad ha \]

Description: If, for the selected condition cond, PSW flags C, V, Z, and N match one of the patterns shown in Table 2-7, then the program branches to address ha; otherwise it does not branch. Cond is set by selecting one of the operands listed in Table 2-7.

When a choice of mnemonics is shown, use the mnemonics beginning with bc if the conditions to be tested were set by a compare instruction; otherwise use the mnemonics beginning with br.

Operation: if selected condition,
then PC \( \leftarrow \) ha

Traps: none

Format: Address

If addressing is relative, this format is used:

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & R1 & cond
\end{array}
\]

For all other addressing, this format is used:

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & addr \ mode
\end{array}
\]

See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63

Examples: Assume (r3) contains 17, (r6) contains \(-19\).

- \texttt{movw} r3,r4
- \texttt{brgt} label1 \hfill \# Branches
- \texttt{cmpw} r3,r6
- \texttt{bcge} label2 \hfill \# Branches
- \texttt{cmpw} r3,r6
- \texttt{bcgeu} label3 \hfill \# Doesn't branch
## Branch On Condition (Continued)

### Table 2-7 Integer Branch Conditions

<table>
<thead>
<tr>
<th>cond</th>
<th>PSW Flags</th>
<th>Name</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X X X X</td>
<td>b</td>
<td>Branch always</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cond</th>
<th>PSW Flags</th>
<th>Name</th>
<th>Compare R1:R2</th>
<th>Name</th>
<th>Result R2:0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X 0 0 0</td>
<td>bclt</td>
<td>Less Than</td>
<td>brgt</td>
<td>Greater Than</td>
</tr>
<tr>
<td></td>
<td>X 1 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>X 0 X 0</td>
<td>bcle</td>
<td>Less or Equal</td>
<td>brge</td>
<td>Greater or Equal</td>
</tr>
<tr>
<td></td>
<td>X 1 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>X X 1 0</td>
<td>bceq</td>
<td>Equal</td>
<td>breq</td>
<td>Equal</td>
</tr>
<tr>
<td>4</td>
<td>X 0 0 1</td>
<td>bcgt</td>
<td>Greater Than</td>
<td>brlt</td>
<td>Less Than</td>
</tr>
<tr>
<td></td>
<td>X 1 X 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>X 1 X 0</td>
<td>bcge</td>
<td>Greater or Equal</td>
<td>brle</td>
<td>Less or Equal</td>
</tr>
<tr>
<td></td>
<td>X 0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X X 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>X X 0 X</td>
<td>bcne</td>
<td>Not Equal</td>
<td>brne</td>
<td>Not Equal</td>
</tr>
<tr>
<td></td>
<td>X X 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0 X X X</td>
<td>bcltu</td>
<td>Less Than Unsigned</td>
<td>brgtu</td>
<td>Greater Than Unsigned</td>
</tr>
<tr>
<td>8</td>
<td>0 X X X</td>
<td>bcleu</td>
<td>Less or Equal Unsigned</td>
<td>brgeu</td>
<td>Greater Than or Equal Unsigned</td>
</tr>
<tr>
<td>9</td>
<td>1 X X X</td>
<td>bcgtu</td>
<td>Greater Than Unsigned</td>
<td>bcltu</td>
<td>Less Than Unsigned</td>
</tr>
<tr>
<td>A</td>
<td>1 X X X</td>
<td>bcgeu</td>
<td>Greater or Equal Unsigned</td>
<td>brleu</td>
<td>Less or Equal Unsigned</td>
</tr>
<tr>
<td></td>
<td>X X 1 X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The R2 field of the branch on condition instruction selects the conditions on which to branch. When a choice of mnemonics is shown, use the ones beginning with `bc` if the condition codes to be tested were set by a compare instruction. Use the mnemonics beginning with `br` if they were set by move or logical instructions (those instructions that set only N or Z).

Legend:
X = don’t care

---

**ADVANCE INFORMATION**

2-16
Branch On Floating Exception

Syntax: \textbf{bf}^* \quad ha

Description: If the exceptions selected by the mnemonic are met, put ha in the PC. The cond field selects exceptions on which to branch as shown in Table 2-8.

<table>
<thead>
<tr>
<th>cond</th>
<th>Name</th>
<th>Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>bfany</td>
<td>Floating ANY exception</td>
</tr>
<tr>
<td>1</td>
<td>bfbad</td>
<td>Floating BAD result</td>
</tr>
<tr>
<td>2-F</td>
<td></td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Operation: if selection condition, then PC $\leftarrow$ ha

Traps: none

Format: Address

If addressing is relative, this format is used:

\begin{center}
\begin{tabular}{ccccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & R1 & cond
\end{tabular}
\end{center}

For all other addressing, this format is used:

\begin{center}
\begin{tabular}{ccccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & addr mode
\end{tabular}
\end{center}

See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63

Example: Assume (f0) contains 1.0, (f1) contains 0.0, and there are no traps enabled.

\begin{verbatim}
divd f0,f1 # Divide by zero
bfdz label1 # branches
\end{verbatim}
CALL

Syntax: call w2,ha

Description: The return address (the address of the instruction following the call instruction) is pushed onto the stack. The stack pointer is contained in w2. Control is then transferred to the specified address. On an exception, the stack pointer has not been modified.

Operation: w2 ← (w2)  
           (w2) ← (PC)  
           PC ← ha

Traps: Page fault  
       Write protect fault  
       Memory fault

Format: Address

If addressing is relative, this format is used:

```
  15  8  7  4  3  0
  0 1 0 0 0 1 0 0  R1  w2
```

For all other addressing, this format is used:

```
  15  8  7  4  3  0
  0 1 0 0 0 1 0 1  addr mode
```

See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63

Example: call sp,print # Call the print routine
CALLS

Call Supervisor

Syntax: \texttt{calls \ bb}

Description: Call the supervisor. Cause one of the 128 unique traps through the indicated supervisor trap vector bb. The current status registers (SSW, PSW and PC) are pushed on the supervisor's stack. The new PC and SSW are taken from the trap vector and the PSW is set to zero.

Operation: \texttt{trap 400 + 8 \times bb}

Traps: none

Format: Control

\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
15 & 8 & 7 & 0 \\
\hline
0 & 0 & 0 & 1 \\
& 0 & 0 & 1 \\
& 0 & bb \\
\hline
\end{tabular}
\end{center}

Example: \texttt{calls \ $6 \ # \ Invoke \ supervisor \ call \ # \ 6}
CMPC

Compare Characters

Syntax: cmpc

Description: Compare a string of bytes. The string length is in r0, the address of the first string is in r1, and the address of the second string is in r2. On an exception, the registers are updated so that restarting the instruction compares the remaining portions of the strings. The bytes are signed extended to 32 bits, then compared as words.

Operation:

```
while [(r0) ≠ 0] & [(r2) = (r1)],
    r0 ← (r0) - 1
    r1 ← (r1) + 1
    r2 ← (r2) + 1

if (r0) = 0,
    then N ← 0
        Z ← 1
        V ← 0
        C ← 0
    else temp ← (r2) - (r1)
        N ← (temp <31>)
        Z ← (temp) = 0
        V ← overflow
        C ← carry out
```

Traps:
Page fault
Read protect fault
Memory fault

Format:
Macro

```
<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</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>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</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>1</td>
<td>1</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>
<td>----</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>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>
```

Example: Assume the string “ABCD” is at label str1 and the string “ABXY” is at label str2.

```
loadq $4,r0  # Load length into r0
loada str1,r1 # Load addr of str1 into r1
loada str2,r2 # Load addr of str2 into r2
cmpc         # Compare the strings
bcge label1  # Does not branch, (r0) = 2,
             # (r1) points to “C”, str1 + 3
             # (r2) points to “X”, str2 + 3
```
CMPD

Compare Double Floating

Syntax:          cmpd    d1,d2

Description:    Compare the double-precision contents of floating register d1 with the double-precision contents of floating register d2. Only the condition codes are affected.

        NOTE

        Although +0.0 and −0.0 are represented differently, they compare equal.

Operation:      (d2) − (d1)

N ← [(d2) − (d1)] ≤ 0] OR [(d2),(d1) unordered]

Z ← [(d2) − (d1)] = 0] OR [(d2),(d1) unordered]

V ← 0

C ← 0

Traps:          none

Format:         Register

Example:        cmpd    f1,f2    # Compare contents of floating regs 1 & 2
Syntax: \texttt{cmpl} \ w, w2

Description: Compare the immediate value wi with the contents of general register w2. Only the condition codes are affected.

Operation: \[(w2) - wi\]
\[N \leftarrow [(w2) - wi] < 0\]
\[Z \leftarrow [(w2) - wi] = 0\]
\[V \leftarrow \text{integer overflow}\]
\[C \leftarrow \text{integer carry out}\]

Traps: none

Format: Immediate

If \(-2^{15} \leq wi \leq +2^{15} - 1\), this format is used:

```
  15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
   1 0 1 0 0 1 1 1 0 0 1 1 w2
   S wi
   31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
```

If \(+2^{15} \leq wi \leq +2^{31} - 1\) or \(-2^{15} \leq wi \leq -2^{31} - 1\), this format is used:

```
  15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
   1 0 1 0 0 1 1 1 0 0 1 1 w2
   S wi low
   47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32
   S wi high
```

Example: \texttt{cmpl} \ $0x1f,r2 \ # \text{Compare 0x1f to the contents of r2}
Syntax: \texttt{cmpq \ wq,w2}

Description: Compare the quick value wq with the contents of general register w2. Only the condition codes are affected.

Operation:
- \((w2) - wq\)
- \(N \leftarrow [(w2) - wq] < 0\)
- \(Z \leftarrow [(w2) - wq] = 0\)
- \(V \leftarrow \text{integer overflow}\)
- \(C \leftarrow \text{integer carry out}\)

Traps: none

Format: Quick

Example: \texttt{cmpq \ $0,r3 \# \text{Compare 0 to the contents of r3}}
CMPS

Compare Single Floating

Syntax: \texttt{cmps} \hspace{5pt} s1,s2

Description: Compare the single-precision contents of floating register \( s1 \) with the single-precision contents of floating register \( s2 \). Only the condition codes are affected.

\textbf{NOTE}
Although +0.0 and -0.0 are represented differently, they compare equal.

Operation:
\begin{align*}
(s2) - (s1) \\
N & \leftarrow [[[d2] - (d1)] < 0] \text{ OR } [[[d2],(d1) \text{ unordered}] \\
Z & \leftarrow [[[d2] - (d1)] = 0] \text{ OR } [[[d2],(d1) \text{ unordered}] \\
V & \leftarrow 0 \\
C & \leftarrow 0
\end{align*}

Traps: none

Format: Register

\begin{tabular}{c|c|c|c|c|c|c|c}
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
0 & 0 & 1 & 0 & 1 & 0 & 1 & s1 & s2
\end{tabular}

Example: \texttt{cmps} \hspace{5pt} f0,f3 \hspace{5pt} # Compare the contents of f0 to the contents of f3
Syntax: cmpw w1,w2

Description: Compare the contents of general register w1 with the contents of general register w2. Only the condition codes are affected.

Operation: (w2) – (w1)
N ← [(w2) – (w1)] < 0
Z ← [(w2) – (w1)] = 0
V ← integer overflow
C ← integer carry out

Traps: none

Format: Register

Example: cmpw r0,r2 # Compare the contents of r0 to the contents of r2
CNVDS
Convert Double to Single Floating

Syntax: \texttt{cnvds d1,s2}

Description: Convert the double-precision contents of floating register d1 to single-precision and put the result in floating register s2. On a trap, the PC and the original value in s2 can be obtained by using the \texttt{loadfs} instruction. The source and destination registers may be the same register.

Operation: \begin{align*}
\text{\texttt{s2} } &\leftarrow \text{(d1)} \\
\text{FX} &\leftarrow \text{floating inexact result} \\
\text{FU} &\leftarrow \text{floating underflow} \\
\text{FV} &\leftarrow \text{floating overflow} \\
\text{FI} &\leftarrow \text{floating invalid}
\end{align*}

Traps: Floating inexact result
Floating invalid operation
Floating overflow
Floating underflow

Format: Macro

\begin{center}
\begin{tabular}{cccc|cccc}
15 & 8 & 7 & 0 \\
\hline
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
31 & 24 & 23 & 20 & 19 & 16 \\
\end{tabular}
\end{center}

Example: \begin{align*}
\text{cnvds f0,f2} & \quad \# \text{Convert double in f0 to single, put in f2} \\
\text{cnvds f1,f1} & \quad \# \text{Convert double in f1 to single}
\end{align*}
**CNVDW**

Convert Double Floating to Word

**Syntax:**
cnvdw  d1, w2

**Description:** Convert the double-precision contents of floating register d1 to a signed integer using the IEEE rounding mode given in the PSW and put the result in general register w2.

Attempting to convert $\pm \infty$, a NaN, or a number too large to fit in a word causes an invalid operation exception, and the stored result is undefined. Attempting to convert a number that is not an exact integer but is small enough for conversion causes a floating inexact result exception.

On a trap, the PC can be obtained with the loadfs instruction. The destination original value, normally returned by the loadfs instruction, is undefined.

**Operation:**
w2 $\leftarrow$ (d1)
FX $\leftarrow$ floating inexact result
FI $\leftarrow$ floating invalid

**Traps:**
Floating invalid operation
Floating inexact result

**Format:**
Macro

<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>31</th>
<th>24</th>
<th>23</th>
<th>20</th>
<th>19</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</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>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d1</td>
<td>w2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example:**
cnvdw  f1, r2  # Convert double in f1 to integer, put in r2
CNVRDW

Convert Rounding Double Floating to Word

Syntax: \( \text{cnrvdw} \ d1,w2 \)

Description: Convert the double-precision contents of floating register \( d1 \) to the nearest signed integer by adding a properly signed 0.5, truncating toward 0.0, and putting the result in general register \( w2 \). This is the round operation in ISO Pascal, regardless of the IEEE rounding mode given in the PSW, and differs from the IEEE round-to-nearest mode.

Attempting to convert \( \pm \infty \), a NaN, or a number too large to fit in a word causes an invalid operation exception, and the stored result is undefined. Attempting to convert a number for which the truncation portion of the operation is inexact causes a floating inexact result exception.

On a trap, the PC can be obtained with the \text{loadfs} instruction. The destination original value, normally returned by the \text{loadfs} instruction, is undefined.

Operation: \( w2 \leftarrow (d1) \)
FX \( \leftarrow \) floating inexact result
FI \( \leftarrow \) floating invalid

Traps: Floating invalid operation
Floating inexact result

Format: Macro

```
  15  8  7  0
1  0  1  1  0  1  0  0  0  0  1  1  0  1  0  1
  0  0  0  0  0  0  0  0  s1  s1
  31  24 23 20 19 16
```

Example: \( \text{cnrvdw} \ f0,r2 \) \# Convert double in f0 to integer, put in r2
CNVRSW

Convert Rounding Single Floating to Word

Syntax: \texttt{cnvrs\ w1,w2}

Description: Convert the single-precision contents of floating register s1 to the nearest signed integer by adding a properly signed 0.5, truncating toward 0.0, and putting the result in general register w2. This is the round operation in ISO Pascal, independent of the IEEE rounding mode in the PSW, and differs from the IEEE round to nearest mode.

Attempting to convert $\pm \infty$, a NaN, or a number too large to fit in a word causes an invalid operation exception, and the stored result is undefined. Attempting to convert a number for which the truncation portion of the operation is inexact causes a floating inexact result exception.

On a trap, the PC can be obtained with the \texttt{loadfs} instruction. The destination original value, normally returned by the \texttt{loadfs} instruction, is undefined.

Operation:

\begin{align*}
\text{w2} & \leftarrow (s1) \\
\text{FX} & \leftarrow \text{floating inexact result} \\
\text{FI} & \leftarrow \text{floating invalid}
\end{align*}

Traps: Floating invalid operation Floating inexact result

Format: Macro

\begin{figure}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
15 & 8 & 7 & 0 \\
\hline
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
\hline
0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
31 & 24 & 23 & 20 & 19 & 16 \\
\hline
\end{tabular}
\end{figure}

Example: \texttt{cnvrs\ f1,r0}  # Convert single in f1 to integer, put in r0
**CNVSD**

**Convert Single to Double Floating**

**Syntax:**

```
cnvsd s1,d2
```

**Description:** Convert the single-precision contents of floating register s1 to double-precision and put the result in floating register d2. The source and destination registers may be the same register.

**Operation:**

- d2 ← (s1)
- Fi ← floating invalid

**Traps:**

Floating invalid operation

**Format:**

Macro

```
15 8 7 0

1 0 1 1 0 1 0 0 0 0 1 1 0 0 0

0 0 0 0 0 0 0 0 s1 d2
```

31 24 23 20 19 16

**Example:**

```
cnvsd f0,f2  # Convert single in f2 to double, put in f2
```
CNVSW

Convert Single Floating to Word

Syntax: \texttt{cnvsw s1\_w2}

Description: Convert the single-precision contents of floating register \( s2 \) to a signed integer using the IEEE rounding mode given in th PSW, and put the result in general register \( w2 \).

Attempting to convert \( \pm \infty \), a NaN, or a number too large to fit in a word causes an invalid operation exception, and the stored result is undefined. Attempting to convert a number that is not an exact integer but is small enough for conversion causes a floating inexact result exception.

On a trap, the PC can be obtained with the \texttt{loadfs} instruction. The destination original value, normally returned by the \texttt{loadfs} instruction, is undefined.

Operation: \( w2 \leftarrow (s1) \)

\( \text{FX} \leftarrow \text{floating inexact result} \)

\( \text{FI} \leftarrow \text{floating invalid} \)

Traps: Floating invalid operation

Floating inexact result

Format: Macro

\[
\begin{array}{cccccccc}
15 & 8 & 7 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
31 & 24 & 23 & 20 & 19 & 16 \\
\end{array}
\]

Example: \texttt{cnvsw f0\_r3} \quad \# \text{Convert single in f0 to integer, result in w2}
**CNVTWD**

**Convert Truncating Double Floating to Word**

**Syntax:**

\[ \text{cnvtwd} \quad d1,w2 \]

**Description:**
Conver the double-precision contents of floating register \( d1 \) to a signed integer by truncating toward 0.0 and putting the result in general register \( w2 \). This is the INT operation in ANSI FORTRAN 77 and the trunc operation in ISO Pascal, regardless of the setting of the IEEE rounding mode in the PSW.

Attempting to convert \( \pm \infty \), a NaN, or a number too large to fit in a word causes an invalid operation exception, and the stored result is undefined. Attempting to convert a number that is not an exact integer but is small enough for conversion causes a floating inexact result exception.

On a trap, the PC can be obtained with the \textbf{loadfs} instruction. The destination original value, normally returned by the \textbf{loadfs} instruction, is undefined.

**Operation:**

\[ \begin{align*}
  &w2 \leftarrow (d1) \\
  &FX \leftarrow \text{floating inexact result} \\
  &FI \leftarrow \text{floating invalid}
\end{align*} \]

**Traps:**

Floating invalid operation
Floating inexact result

**Format:**

Macro

```
+---+---+---+---+---+---+---+---+
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
+---+---+---+---+---+---+---+---+
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
+---+---+---+---+---+---+---+---+
| d1 | w2 |
+---+---+---+---+---+---+---+---+
```

**Example:**

\[ \text{cnvtwd} \quad f1,r1 \quad \# \text{Truncate double in } f1 \text{ to integer, put in } r2 \]
Convert Truncating Single Floating to Word

Syntax: \texttt{cnvtsw s1, w2}

Description: Convert the single-precision contents of floating register \texttt{s1} to a signed integer by truncating toward 0.0 and putting the result in general register \texttt{w2}. This is the INT operation in ANSI FORTRAN 77 and the trunc operation in ISO Pascal, regardless of the setting of the IEEE rounding mode in the PSW.

Attempting to convert \( \pm \infty \), a NaN, or a number too large to fit in a word causes an invalid operation exception, and the stored result is undefined. Attempting to convert a number that is not an exact integer but is small enough for conversion causes a floating inexact result exception.

On a trap, the PC can be obtained with the \texttt{loadfs} instruction. The destination original value, normally returned by the \texttt{loadfs} instruction, is undefined.

Operation:
\begin{align*}
& w2 \leftarrow (s1) \\
& FX \leftarrow \text{floating inexact result} \\
& FI \leftarrow \text{floating invalid}
\end{align*}

Traps:
Floating invalid operation
Floating inexact result

Format:
Macro

\begin{center}
\begin{tabular}{cccccc}
15 & 8 & 7 & 0 \\
\hline
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & s1 & w2 \\
\hline
31 & 24 & 23 & 20 & 19 & 16
\end{tabular}
\end{center}

Example: \texttt{cnvtsw f0, r2} # Truncate \texttt{f0} to integer, store in \texttt{r2}
CNVWD  Convert Word to Double Floating

Syntax:  \texttt{cnvwd} \( w1,d2 \)

Description:  Convert the contents of general register \( w1 \) to double-precision and put the result in floating register \( d2 \).

Operation:  \( d2 \leftarrow (w1) \)

Traps:  none

Format:  Macro

\[
\begin{array}{cccccc}
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{array}
\]

31 24 19 16

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

w1  d2

Example:  \texttt{cnvwd r1,f0} \# Convert integer in r1 to double, result in f0

ADVANCE INFORMATION  2-34
### CNVWS

**Convert Word to Single Floating**

**Syntax:**
```
cnvws  w1,s2
```

**Description:** Convert the contents of general register \( w1 \) to single-precision floating-point and put the result in floating register \( s2 \). If the conversion is not exact, the result is rounded according to the IEEE rounding mode in the PSW, and floating inexact result is signalled.

**Operation:**
- \( s2 \leftarrow (w1) \)
- \( FX \leftarrow \) floating inexact result

**Traps:** Floating inexact result

**Format:**
```
<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Example:
```
cnvws  r3,f2  
# Convert integer in r3 to single, result in f2
```
DIVD

Divide Double Floating

**Syntax:**
\[ \text{divd} \ d1,d2 \]

**Description:**
Divide the double-precision contents of floating register d2 by the double-precision contents of floating register d1 and put the result in d2. On a trap, the PC and the original value in d2 can be obtained by using the `loadfs` instruction.

**Operation:**
- \( d2 \leftarrow (d2) \div (d1) \)
- \( FX \leftarrow \text{floating inexact result} \)
- \( FU \leftarrow \text{floating underflow} \)
- \( FD \leftarrow \text{floating divide-by-zero} \)
- \( FV \leftarrow \text{floating overflow} \)
- \( FI \leftarrow \text{floating invalid} \)

**Traps:**
- Floating inexact result
- Floating invalid operation
- Floating overflow
- Floating underflow
- Floating divide-by-zero

**Format:**
- Register
  
  \[
  \begin{array}{cccccccc}
  & & & & & & & \\
  & & & & & & & \\
  & & & & & & & \\
  & & & & & & & \\
  & & & & & & & \\
  & & & & & & & \\
  & & & & & & & \\
  & & & & & & & \\
  & & & & & & & \\
  & & & & & & & \\
  \hline
  0 & 0 & 1 & 0 & 1 & 1 & d1 & d2 \\
  \end{array}
  \]

**Example:**
\[ \text{divd} \ f2,f0 \]  # Divide double in f0 by double in f2, put result in f0
**DIVS**

**Divide Single Floating**

**Syntax:**

```
divs   s1, s2
```

**Description:** Divide the single-precision contents of floating register s2 by the single-precision contents of floating register s1 and put the result in s2. On a trap, the PC and the original value in s2 can be obtained by using the `loadfs` instruction.

**Operation:**

```
(s2) ← (s2) ÷ (s1)
FX ← floating inexact result
FU ← floating underflow
FD ← floating divide-by-zero
FV ← floating overflow
FI ← floating invalid
```

**Traps:**

Floating inexact result
Floating invalid operation
Floating overflow
Floating underflow
Floating divide-by-zero

**Format:**

```
<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>3</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>
</tr>
</tbody>
</table>
```

**Example:**

```
divs   f1, f2  # Divide single in f2 by single in f1, put result in f2
```
DIVW

Divide Word

Syntax: \texttt{divw w1,w2}

Description: Divide the contents of general register \( w2 \) by the contents of general register \( w1 \) and put the \textit{quotient} in \( w2 \). The operands are treated as signed integers. The quotient is positive if the signs of the divisor and dividend are the same, and negative if they are different. Overflow is set if the largest negative number \((-2^{31})\) is divided by \(-1\).

\textbf{NOTE}

For the same dividend and divisor, if the quotient returned by \texttt{divw} is multiplied by the divisor and then added to the remainder returned by \texttt{modw}, the result would be the original dividend.

Operation: \( w2 \leftarrow (w2) \div (w1) \)
\( N \leftarrow 0 \)
\( Z \leftarrow 0 \)
\( V \leftarrow \text{integer overflow} \)
\( C \leftarrow 0 \)

Traps: Divide-by-zero

Format: Register

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|}
\hline
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
1 & 0 & 0 & 1 & 1 & 0 & 0 \text{ w1} & \text{ w2} \\
\hline
\end{tabular}
\end{center}

Example: \texttt{divw r10,r4} \# Divide \( (r4) \) by \( (r10) \), put result in \( r4 \)
DIVWU

Divide Word Unsigned

Syntax: \texttt{divwu} \; w1,w2

Description: Divide the contents of general register w2 by the contents of general register w1 and put the quotient in w2. The operands are treated as unsigned integers. Since the quotient is always positive, overflow cannot occur.

\textbf{NOTE}

For the same dividend and divisor, if the quotient returned by \texttt{divwu} and the remainder returned by \texttt{modwu} are multiplied, the result would be the original dividend.

Operation: \begin{align*}
w2 & \leftarrow (w2) \div (w1) \\
N & \leftarrow 0 \\
Z & \leftarrow 0 \\
V & \leftarrow 0 \\
C & \leftarrow 0
\end{align*}

Traps: Divide-by-zero

Format: Register

\begin{center}
\begin{tabular}{|cccccc|}
\hline
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
1 & 0 & 0 & 1 & 1 & 1 & 0 & w1 & w2 \\
\hline
\end{tabular}
\end{center}

Example: \texttt{divwu \; r0,r5} \; \#\; Divide \; (r5) \; by \; (r0), \; put \; result \; in \; r5
initialize characters

**Syntax:**

`initc`

**Description:**

Initialize a byte, halfword, word, or single-precision floating string with a constant value. The length of the string (in bytes) is in the r0 and the address is in r1. The pattern to be stored is in r2. For word and single-precision floating strings, r2 contains the initial value. For halfword strings, both halfwords of r2 must contain the initial value. For byte strings, all four bytes of r2 must contain the desired value. If a trap occurs, the registers are updated so that restarting the instruction initializes the remaining portion of the string.

**Operation:**

```
while (r0) ≠ 0,
   (r1) ← (r2 <7:0>)
   r0 ← (r0) - 1
   r1 ← (r1) + 1
   r2 ← (r2) ROT - 8
```

**Traps:**

Page fault
Write protect fault
Memory fault

**Format:**

Macro

```
  15  8  7  0
  1  0  1  1  0  0  0  0  0  1  1  1  0

  31  24  23  20  19  16
  0  0  0  0  0  0  0  0  0  0  0  0
```

**Example:**

Set the 17 bytes beginning at label `str` to 19.

```
  loadi  $17,r0  # String length
  loada  str,r1  # String address
  loadi  $0x13131313,r2  # 19 decimal = 13 hex
  initc
  # Initialize characters
```
**LOADA**

**Syntax:** \texttt{loada \ ba,w2}

**Description:** Load the memory address \( ba \) into general register \( w2 \).

**Operation:** \( w2 \leftarrow ba \)

**Traps:** none

**Formats:** Address

If addressing is relative, this format is used:

<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

For all other addressing, this format is used:

<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63

**Examples:**

- \( \texttt{loada \ value,r0} \)  
  # Address of value into r0
- \( \texttt{loada \ 8(sp),fp} \)  
  # (sp) + 8 into fp
- \( \texttt{loada \ addr(r4),r0} \)  
  # (r4) + addr into r0
- \( \texttt{loada \ [r1](r2),r3} \)  
  # 3 operand addr that sets no flags
LOADB

**Syntax:** \texttt{loadb \, ba,w2}

**Description:** Load the byte at memory address \( ba \), sign-extended, into the least-significant byte of general register \( w2 \).

**Operation:** \( w2 \leftarrow (ba) \)

**Traps:** Page fault
Read protect fault
Memory fault

**Format:** Address

If addressing is relative, this format is used:

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 \\
\end{array}
\]

For all other addressing, this format is used:

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 \\
\end{array}
\]

See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63

**Example:** \texttt{loadb \, 3(r7),r2}  \hspace{1em} \# Load byte at 3(r7) into r2
Load Byte Unsigned

Syntax: \texttt{loadbu} \hspace{5pt} ba,w2

Description: Load the byte at memory address \( ba \), zero-extended, into the least-significant byte of general register \( w2 \).

Operation: \( w2 \leftarrow (ba) \)

Traps: Page fault
Read protect fault
Memory fault

Format: Address

If addressing is relative, this format is used:

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & \text{R1} & w2
\end{array}
\]

For all other addressing, this format is used:

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & \text{addr mode}
\end{array}
\]

See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63

Example: \texttt{loadbu 4(r5),r2} \hspace{5pt} \# \hspace{5pt} \text{Load byte at 4(r5) into r2}
Syntax: \texttt{load} \quad da,d2

Description: Load the double-precision floating-point value at memory addresses \textit{da} and \textit{da} + 1 into floating register \textit{d2}.

Operation: \textit{d2} \leftarrow (\textit{da})

Traps: Page fault
Read protect fault
Memory fault

Format: Address

If addressing is relative, this format is used:

\begin{center}
\begin{tabular}{|c|c|c|c|c|}
\hline
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & R1 & d2 \\
\hline
\end{tabular}
\end{center}

For all other addressing, this format is used:

\begin{center}
\begin{tabular}{|c|c|c|c|c|}
\hline
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
\hline
\end{tabular}
\end{center}

\textbf{See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63}

Examples:
\begin{itemize}
\item \texttt{load} \quad \textit{addr},f2 \quad \# \text{ Load double floating value at \textit{addr} into \textit{f2}}
\item \texttt{load} \quad 4(r2),f2 \quad \# \text{ Load double floating value at address specified by 4(r2)}
\end{itemize}
LOADFS

Load Floating Status

Syntax: \texttt{loadfs} \hspace{0.5em} w1, d2

Description: Load the floating status following a floating trap. The PC of the offending floating instruction is put in general register \( w1 \). The original value of the destination register is put in floating register \( d2 \). This information allows a trap handler to determine the original operation and its operands.

Operation: \( w1 \leftarrow (\text{FP PC}) \)
\( d2 \leftarrow (\text{FP dest}) \)

Traps: none

Format: Macro

\begin{center}
\begin{tabular}{cccccccc}
\hline
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \%
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
15 & 8 & 7 & 0 & 31 & 24 & 23 & 19 & 16 \\
\end{tabular}
\end{center}

Example: \texttt{loadfs} \hspace{0.5em} r1, f2 \hspace{0.5em} \# Load FP PC into r1, and original destination into f2
LOADH

Load Halfword

Syntax: \text{loadh } ha,w2

Description: Load the halfword at memory address ha, sign-extended, into the least-significant halfword of general register w2.

Operation: w2 \leftarrow (ha)

Traps: Page fault
Read protect fault
Memory fault

Format: Address

If addressing is relative, this format is used:

\begin{center}
\begin{tabular}{c c c c c}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & R1 & w2
\end{tabular}
\end{center}

For all other addressing, this format is used:

\begin{center}
\begin{tabular}{c c c c c c}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & \text{addr mode}
\end{tabular}
\end{center}

See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63

Example: \text{loadh (r15),r12} \# Load the halfword at (r15) into r12
LOADHU

Load Halfword Unsigned

Syntax: \texttt{loadhu ha,w2}

Description: Load the halfword at memory address ha, zero-extended, into the least-significant halfword of general register w2.

Operation: \( w2 \leftarrow (ha) \)

Traps: Page fault
Read protect fault
Memory fault

Format: Address

If addressing is relative, this format is used:

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & R1 & w2 \\
\end{array}
\]

For all other addressing, this format is used:

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 1 & \text{addr mode} \\
\end{array}
\]

See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63

Example: \texttt{loadhu new,r11} \# Load the halfword at new into r11
Syntax: \texttt{loadi} \hspace{0.5em} \texttt{wi,w2}

Description: Load the immediate value \texttt{wi}, sign-extended, into general register \texttt{w2}.

Operation:
\begin{align*}
\texttt{w2} & \leftarrow \texttt{wi} \\
\texttt{N} & \leftarrow (\texttt{w2} < 31 >) \\
\texttt{Z} & \leftarrow (\texttt{w2}) = 0 \\
\texttt{V} & \leftarrow 0 \\
\texttt{C} & \leftarrow 0
\end{align*}

Traps: none

Format: Immediate

If $-2^{15} \leq \texttt{wi} \leq +2^{15} - 1$, this format is used:

\begin{center}
\begin{tabular}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
& & & & & 1 & 0 & 1 & 1 \\
& S & & & & & \multicolumn{5}{c}{\texttt{w2}} \\
& & & & & \texttt{wi} & & & & \\
31 & 30 & & & & & 16 & & & & \\
\end{tabular}
\end{center}

If $+2^{15} \leq \texttt{wi} \leq +2^{31} - 1$ or $-2^{15} \leq \texttt{wi} \leq -2^{15} - 1$, this format is used:

\begin{center}
\begin{tabular}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
& & & & & 0 & 0 & 1 & 1 \\
& S & & & & & \multicolumn{5}{c}{\texttt{w2}} \\
& wi low & & & & & & & & & \\
& & & & wi high & & & & & & \\
47 & 46 & & & & & 32 & & & & \\
\end{tabular}
\end{center}

Example: \texttt{loadi $21,r2} \quad \# \text{ Load 21 into r2}
LOADQ

Load Quick

Syntax: \texttt{loadq wq, w2}

Description: Load the quick value \(wq\), zero-extended, into general register \(w2\).

Operation:
\[
\begin{align*}
w2 & \leftarrow wq \\
N & \leftarrow 0 \\
Z & \leftarrow (w2) = 0 \\
V & \leftarrow 0 \\
C & \leftarrow 0
\end{align*}
\]

Traps: none

Format: Quick

\[
\begin{array}{|cccc|}
\hline
1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & \text{wq} & \text{w2} \\
\hline
\end{array}
\]

Example: \texttt{loadq 000110 000110}  # Load f hex into r1
LOADS

Load Single Floating

Syntax: \texttt{loads \ sa, s2}

Description: Load the single-precision floating-point value at memory address \( sa \) into floating register \( s2 \).

Operation: \( s2 \leftarrow (sa) \)

Traps: Page fault
Read protect fault
Memory fault

Format: Address

If addressing is relative, this format is used:

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
& 0 & 1 & 1 & 0 & 0 & 1 \ 0 \ 0 & R1 & s2 \\
\end{array}
\]

For all other addressing, this format is used:

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
& 0 & 1 & 1 & 0 & 0 & 1 \ 0 \ 1 & \text{addr mode} \\
\end{array}
\]

See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63

Example:
\texttt{loads \ clock, f2}  \# Load single floating value at clock into f2
\texttt{loads \ (r3), f3}  \# Load single floating value at (r3) into f3
LOADW

Load Word

Syntax: \( \text{loadw} \quad wa,w2 \)

Description: Load the contents of memory address \( wa \) into general register \( w2 \).

Operation: \( w2 \leftarrow (wa) \)

Traps: Page fault
Read protect fault
memory fault

Format: Address

If addressing is relative, this format is used:

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & \quad \text{R1} & \quad w2
\end{array}
\]

For all other addressing, this format is used:

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & \quad \text{addr mode}
\end{array}
\]

See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63

Example:

\( \text{loadw} \quad \text{abc,r2} \quad \# \text{ Load word at abc into r2} \)

\( \text{loadw} \quad \$0xfca2,r1 \quad \# \text{ Load word at address fca2 hex into r1} \)
modw \ w1,w2

Description: Divide the contents of general register w2 by the contents of general register w1 and put the remainder in w2. The operands are treated as signed integers. The quotient is positive if the signs of the divisor and dividend are the same, and negative if they are different. If the remainder is non-zero, it has the same sign as the dividend. (Overflow is set if the largest negative number \((-2^{31})\) is divided by \(-1\).)

NOTE
For the same dividend and divisor, if the quotient returned by divw is multiplied by the divisor and then added to the remainder returned by modw, the result would be the original dividend.

Operation: \[\text{w2} \leftarrow (\text{w2}) \mod (\text{w1})\]
N \leftarrow 0
Z \leftarrow 0
V \leftarrow \text{integer overflow}
C \leftarrow 0

Traps: Divide-by-zero

Format: Register

<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1 1 0 1</td>
<td>w1</td>
<td>w2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:
loadi $1234,r2  # Load 1234 into r2
loadq $11,r1    # Load 11 into r1
modw r1,r2      # Divide r2 by r1, save remainder in r2
MODWU
Modulus Word Unsigned

Syntax: \texttt{modwu} \quad \texttt{w1,w2}

Description: Divide the contents of general register \texttt{w2} by the contents of general register \texttt{w1} and put the remainder in \texttt{w2}. The operands are treated as unsigned integers. If the remainder is non-zero, then it is positive.

\textbf{NOTE}
For the same dividend and divisor, if the quotient returned by \texttt{divwu} and the remainder returned by \texttt{modwu} are multiplied, the result would be the original dividend.

Operation: \( w2 \leftarrow (w2) \mod (w1) \)
\( N \leftarrow 0 \)
\( Z \leftarrow 0 \)
\( V \leftarrow 0 \)
\( C \leftarrow 0 \)

Traps: Divide-by-zero

Format: Register

\begin{center}
\begin{tabular}{cccccc}
\hline
1 & 0 & 0 & 1 & 1 & 1 \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{|c|c|}
\hline
w1 & w2 \\
\hline
\end{tabular}
\end{center}

Example:
\begin{itemize}
\item loadi \texttt{$0xffe,r5} \quad \# \text{ Load ffe hex into r5}
\item loadq \texttt{$0x11,r0} \quad \# \text{ Load 11 hex into r0}
\item modw \texttt{r0,r5} \quad \# \text{ Divide r5 by r0, save remainder in r5}
\end{itemize}
MOVC

Move Characters

Syntax: movc

Description: Move a string of bytes. The length of the strings is in r0, the address of the source string is in r1, and the address of the destination string is in r2. On an exception, the registers are updated so that restarting the instruction moves the remaining portion of the string.

The source and destination strings may not overlap. The initc instruction should be used to initialize memory or the source string may be moved in blocks.

Operation:

while (r0) ≠ 0
    (r2) ← ((r1))
    r0 ← (r0) - 1
    r1 ← (r1) + 1
    r2 ← (r2) + 1

Traps:
Page fault
Read protect fault
Write protect fault
Memory fault

Format: Macro

<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 1 0 1 0 0</td>
<td>0 0 0 0 1 1 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example: Move a string of 124 characters at label **string1** to **string2**.

```assembly
loadi $124,r0  # Load string length into r0
loada string1,r1  # Load address of source string into r1
loada string2,r2  # Load address of destination string into r2
mовc             # Move the characters
```
MOVD

Move Double Floating

Syntax: \texttt{movd} \hspace{1em} d1,d2

Description: Move the double-precision contents of floating register d1 to floating register d2. On a trap, the PC and the original value in d2 can be obtained by using the \texttt{loadfs} instruction.

Operation: \hspace{1em} d2 \leftarrow (d1)

Traps: none

Format: Register

\begin{center}
\begin{tabular}{cccccc}
15 & 14 & 13 & 12 & 11 & 10 \\
0 & 0 & 1 & 0 & 0 & 1 \hspace{1em} d1 \hspace{1em} 1 & 0 \hspace{1em} d2
\end{tabular}
\end{center}

Example: \texttt{movd f1,f2} \hspace{1em} \# Move (f1) to f2
**MOVDL**

*Move Double Floating to Longword*

**Syntax:**

```
MOVDL d1, l2
```

**Description:**

Move the double-precision contents of floating register `d1` to longword register pair `l2` without conversion.

**Operation:**

`l2 ← (d1)`

**Traps:**

`none`

**Format:**

Register

```
  15  14  13  12  11  10   8   7   4   3   0
  0  0  1  0  1  1  1  0   d1   l2
```

**Example:**

```
MOVDL f1, r2    # Move f1 to reg pair (r2,r3)
```
MOVLD

Move Longword to Double Floating

Syntax: \texttt{movld} \hspace{1em} l1,d2

Description: Move the contents of longword register pair l1 to floating register d2 without conversion.

Operation: \((d2) \leftarrow (l1)\)

Traps: none

Format: Register

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & d2
\end{array}
\]

Example: \texttt{movld} \hspace{1em} r0,f2 \hspace{1em} \# Move \((r2,r3)\) to double floating reg f2
MOVPW
Move Processor Register to Word

Syntax: \texttt{movpw \ p1,w2}

Description: Move the contents of processor register \( p1 \) to general register \( w2 \). The \( p1 \) value is interpreted as follows:

\begin{center}
\begin{tabular}{ccl}
\textbf{p1} & \textbf{Name} & \textbf{Meaning} \\
0 & PSW & Program status word \\
1 & SSW & Supervisor status word \\
2-15 & --- & (Reserved)
\end{tabular}
\end{center}

Operation: \( w2 \leftarrow (p1) \)

Traps: none

Format: Register

\begin{center}
\begin{tabular}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & p1 & w2
\end{tabular}
\end{center}

Examples:

\begin{itemize}
\item \texttt{movpw psw,r2} \ # Move (PSW) to r2
\item \texttt{movpw ssw,r2} \ # Move (SSW) to r2
\end{itemize}
MOVS

Move Single Floating

Syntax: \texttt{movs} \texttt{s1,s2}

Description: Move the single-precision contents of floating register s1 to floating register s2.

Operation: \texttt{s2} \leftarrow (\texttt{s1})

Traps: none

Format: Register

\begin{center}
\begin{tabular}{cccc}
15 & 14 & 13 & 8 & 7 & 4 & 3 & 0 \\
\hline
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & s1 & s2
\end{tabular}
\end{center}

Example: \texttt{movs} \texttt{f1,f2} \hspace{1em} \# Move (f1) to f2
MOVSU

Move Supervisor to User (Privileged)

Syntax: \texttt{movsu \ w1,\ w2}

Description: Move the contents of supervisor general register \( \text{w1} \) to user general register \( \text{w2} \). A privileged instruction trap occurs if this instruction is executed in user mode.

Operation:

\[
\begin{align*}
\text{w2}_{\text{usr}} & \leftarrow (\text{w1})_{\text{sup}} \\
\text{N} & \leftarrow (\text{w2} < 31 >)_{\text{usr}} \\
\text{Z} & \leftarrow (\text{w2})_{\text{usr}} = 0 \\
\text{V} & \leftarrow 0 \\
\text{C} & \leftarrow 0
\end{align*}
\]

Traps: Privileged instruction

Format: Macro

\[
\begin{array}{cc|cc|cccc|c}
15 & 8 & 7 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
31 & 24 & 23 & 20 & 19 & 16 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w1 & w2
\end{array}
\]

Example: \texttt{movsu \ r0,r11} \quad \# \text{Move (r0) to r11}
MOVSU

Move Single Floating to Word

Syntax: \texttt{movsw } s1, w2

Description: Move the single-precision contents of floating register s1 to general register w2 without conversion.

Operation: \( w2 \leftarrow (s1) \)

Traps: none

Format: Register

\[
\begin{array}{ccccccc}
15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & s1 & & w2 \\
\end{array}
\]

Example: \texttt{movsw f1, r2 } \quad \# \text {Move (f1) to r2}
**MOVUS**

Move User to Supervisor (Privileged)

**Syntax:**

\[
\text{movus } w1, w2
\]

**Description:**

Move the contents of user general register \( w1 \) to supervisor general register \( w2 \).

**Operation:**

\[
\begin{align*}
\text{w2}_{\text{sup}} & \leftarrow (w1)_{\text{usr}} \\
\text{N} & \leftarrow (w2_{\text{<31>}})_{\text{sup}} \\
\text{Z} & \leftarrow (w2)_{\text{sup}} = 0 \\
\text{V} & \leftarrow 0 \\
\text{C} & \leftarrow 0
\end{align*}
\]

**Traps:**

Privileged instruction

**Format:**

Macro

<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<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>

\[
\begin{align*}
0 & 0 0 0 0 0 0 0 0 \\
0 & 0 0 0 0 0 0 0 0
\end{align*}
\]

**Example:**

\[
\text{movus } r1, r2 \quad \# \text{ Move (r1) to r2}
\]
MOVW

Move Word

Syntax: \texttt{movw} \ w1,w2

Description: Move the contents of general register \( w1 \) to general register \( w2 \).

Operation: \[
\begin{align*}
w2 & \leftarrow (w1) \\
N & \leftarrow (w2 \ll 31) \\
Z & \leftarrow (w2) = 0 \\
V & \leftarrow 0 \\
C & \leftarrow 0
\end{align*}
\]

Traps: none

Format: Register

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & w1 & w2
\end{array}
\]

Example: \texttt{movw r3,r0} \quad \# \text{ Move (r3) to r0}
MOVWP

Move Word to Processor Register

Syntax: \texttt{movwp \textit{w2,p1}}

Description: Move the contents of general register \textit{w2} to processor register \textit{p1}. The \textit{p1} value is interpreted as follows:

\begin{center}
\begin{tabular}{c c l}
\textbf{p1} & \textbf{Name} & \textbf{Meaning} \\
0 & PSW & Program status word \\
1 & SSW & Supervisor status word \\
2–15 & — & (Reserved)
\end{tabular}
\end{center}

If \textit{p1} represents the PSW, then the condition codes are also set. Attempting to modify the SSW in user mode will cause a \texttt{noop}.

Operation: \begin{align*}
\textit{p1} & \leftarrow (\textit{w2}) \\
\textit{N} & \leftarrow [\textit{p1} = \textit{psw}] \text{ AND PSW}<\textit{N}> \\
\textit{Z} & \leftarrow [\textit{p1} = \textit{psw}] \text{ AND PSW}<\textit{Z}> \\
\textit{V} & \leftarrow [\textit{p1} = \textit{psw}] \text{ AND PSW}<\textit{V}> \\
\textit{C} & \leftarrow [\textit{p1} = \textit{psw}] \text{ AND PSW}<\textit{C}>
\end{align*}

Traps: none

Format: Register

\begin{center}
\begin{tabular}{ccccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & p1 & w2
\end{tabular}
\end{center}

Examples: \texttt{movwp \textit{r2,psw}} # Move (r2) to PSW

The following example only works in supervisor mode:

\texttt{movwp \textit{r2,ssw}} # Move (r2) to SSW
MOVWS

Move Word to Single Floating

Syntax:   movws  w1,s2

Description:   Move the contents of general register w1 to floating register s2 without conversion.

Operation:   s2 ← (w1)

Traps:   none

Format:   Register

Example:   movws  r1,f2  # Move (r1) to f2
MULD
Multiply Double Floating

Syntax:  
\text{muld} \quad d1,d2

Description:  
Multiply the double-precision contents of floating register d2 by the double-precision contents of floating register d1 and put the result in d2. On a trap, the PC and the original value in d2 can be obtained by using the loadfs instruction.

Operation:  
d2 ← (d2) × (d1)
FX ← floating inexact result
FU ← floating underflow
FV ← floating overflow
FI ← floating invalid

Traps:  
Floating inexact result
Floating invalid operation
Floating overflow
Floating underflow

Format:  
Register

<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>3</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>
</tr>
</tbody>
</table>
\hspace{1cm} d1 \hspace{1cm} d2

Example:  
Assume fpval1 contains 1239237.1234 and fpval2 contains 8989.44334.

\begin{verbatim}
loadd fpval1,f1  # Load double floating value
loadd fpval2,f2  # Load double floating value
muld   f1,f2     # Multiply the values
\end{verbatim}

The result is 11140051905.62889.
Syntax: \texttt{muls s1,s2}

Description: Multiply the single-precision contents of floating register \texttt{s2} by the single-precision contents of floating register \texttt{s1} and put the result in \texttt{s2}. On a trap, the PC and the original value in \texttt{d2} can be obtained by using the \texttt{loadfs} instruction.

Operation:
\begin{itemize}
  \item \texttt{s2} $\leftarrow (s2) \times (s1)$
  \item \texttt{FX} $\leftarrow$ floating inexact result
  \item \texttt{FU} $\leftarrow$ floating underflow
  \item \texttt{FV} $\leftarrow$ floating overflow
  \item \texttt{FI} $\leftarrow$ floating invalid
\end{itemize}

Traps:
\begin{itemize}
  \item Floating inexact result
  \item Floating invalid operation
  \item Floating overflow
  \item Floating underflow
\end{itemize}

Format: Register

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|}
\hline
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
0 & 0 & 1 & 0 & 1 & 0 0 0 \\
\hline
\end{tabular}
\end{center}

\begin{itemize}
  \item \texttt{loadi} $0xf8ccd,r0$ # Load r0 with value
  \item \texttt{movws} r0,f0 # Transfer to f0
  \item \texttt{loadi} $0xc27dd,r0$ # Load r0 with value
  \item \texttt{movws} r0,f1 # Transfer to f1
  \item \texttt{muls} f0,f1 # Multiply the numbers
  \item \texttt{stors} f1,place1 # Save the result
  \item \texttt{.}
  \item \texttt{.}
  \item \texttt{.}
\texttt{place1: .space}
\end{itemize}
Syntax: \texttt{mulw w1,w2}

Description: Multiply the contents of general register w2 by general register w1 and put the least-significant word of the product in w2. The operands are treated as signed integers. Overflow is set if the product cannot be represented in one word.

Operation: \begin{align*}
\text{w2} & \leftarrow (w2) \times (w1) \\
\text{N} & \leftarrow 0 \\
\text{Z} & \leftarrow 0 \\
\text{V} & \leftarrow \text{integer overflow} \\
\text{C} & \leftarrow 0
\end{align*}

Traps: none

Format: Register

\begin{table}[h]
\centering
\begin{tabular}{cccccc}
\hline
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & w1 & w2 \\
\hline
\end{tabular}
\end{table}

Example:
\begin{align*}
\text{loadi} & \quad \$999,r4 & \quad \# \text{ Load 999 into r4} \\
\text{loadw} & \quad \text{wdata},r2 & \quad \# \text{ Get value at wdata} \\
\text{mulw} & \quad r2,r4 & \quad \# \text{ Multiply the numbers}
\end{align*}
**MULWU**

Multiply Word Unsigned

**Syntax:**

\[ \text{mulw} \ w1, w2 \]

**Description:**

Multiply the contents of general register \( w2 \) by the contents of general register \( w1 \) and put the least-significant word of the product in \( w2 \). The operands are treated as unsigned integers. Overflow is set if the result cannot be represented in one word.

**Operation:**

\[
\begin{align*}
w2 & \leftarrow (w2) \times (w1) \\
N & \leftarrow 0 \\
Z & \leftarrow 0 \\
V & \leftarrow \text{integer overflow} \\
C & \leftarrow 0
\end{align*}
\]

**Traps:**

none

**Format:**

Register

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</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>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</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>

\( w1 \quad w2 \)

**Example:**

\[
\begin{align*}
\text{loadhu} & \quad \text{hwval,r0} \quad \# \text{Load the value at hwval} \\
\text{loadq} & \quad \$33,r1 \quad \# \text{Load 33 into r1} \\
\text{mulw} & \quad r0,r1 \quad \# \text{Multiply the numbers}
\end{align*}
\]
MULWUX

Multiply Word Unsigned Extended

Syntax: \[ \text{mulwux} \ w1, l2 \]

Description: Multiply the contents of general register \( w1 \) by the contents of general register \( w2 \) and put the product in longword register pair \( l2 \). The operands are treated as unsigned integers. Overflow is set if the product cannot be represented in one word.

Operation:
\[
\begin{align*}
l2 & \leftarrow (w2) \times (w1) \\
N & \leftarrow 0 \\
Z & \leftarrow 0 \\
V & \leftarrow \text{result requires a longword} \\
C & \leftarrow 0
\end{align*}
\]

Traps: none

Format: Register

\[
\begin{array}{cccccc}
15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\hline
1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & w1 & 0 & l2
\end{array}
\]

Example:
\[
\begin{align*}
\text{loadhu} & \quad \text{uhwval}, r3 \\
\text{loadi} & \quad \$0xf, r6 \\
\text{mulwux} & \quad r3, r6
\end{align*}
\]

# Load the value at uhwval
# Load ff hex into r6
# Multiply the numbers
MULWX

Multiply Word-Extended

Syntax: \texttt{mulwx} \hspace{0.5cm} w1,l2

Description: Multiply the contents of general register w1 by the contents of general register w2 and put the product in longword register pair l2. Overflow is set if the product cannot be represented in one word.

Operation: \begin{align*}
l2 &\leftarrow (w2) \times (w1) \\
N &\leftarrow 0 \\
Z &\leftarrow 0 \\
V &\leftarrow \text{product requires a longword} \\
C &\leftarrow 0
\end{align*}

Traps: none

Format: Register

\begin{center}
\begin{array}{cccccc}
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & w1 & l2 & 0 \\
\end{array}
\end{center}

Example:
\begin{align*}
\text{loadhu} & \hspace{0.5cm} \text{wxval},r0 & \# \text{Load the value at wxval} \\
\text{loadi} & \hspace{0.5cm} \$1023,r4 & \# \text{Load 1023 into r4} \\
\text{mulwx} & \hspace{0.5cm} r0,r4 & \# \text{Multiply the numbers}
\end{align*}
Syntax: \texttt{negd} \hspace{5pt} d1, d2

Description: Negate the double-precision contents of floating register d1 and put the result in floating register d2. The sign of d1 is reversed so that \pm 0.0 and NaNs are handled properly.

Operation: d2 ← − (d1)

Traps: none

Format: Macro

```
15  8  7  0
  1 0 1 1 0 1 0 0  0 0 1 1 1 0 1 1
  0 0 0 0 0 0 0 0    d1    d2
31 24 23 20 19 16
```

Example: \texttt{negd} \hspace{5pt} f0,f0 \hspace{5pt} # Negate the register
**NEGS**

**Negate Single Floating**

**Syntax:** negs  s1,s2

**Description:** Negate the single-precision contents of floating register s1 and put the result in floating register s2. The sign of s1 is reversed so that ±0.0 and NaNs are handled properly.

**Operation:** s2 ← -(s1)

**Traps:** none

**Format:** Macro

<table>
<thead>
<tr>
<th>31</th>
<th>24</th>
<th>23</th>
<th>20</th>
<th>19</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td>s1</td>
<td></td>
<td>s2</td>
</tr>
<tr>
<td>1 0 1 1 0 1 0 0</td>
<td>0 0 1 1 1 0 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example:** negs  f1,f2  # Negate the value in f1, store in f2
Negate Word

Syntax: \texttt{negw} \ w1,w2

Description: Two's complement the contents of general register \( w1 \) and put the result in general register \( w2 \). Overflow is set if \( w1 \) contains the largest negative number (\(-2^{31}\)). Carry is set if \( w1 \) does not equal 0.

Operation:
\begin{align*}
\text{w2} \leftarrow & \ - (w1) \\
\text{N} \leftarrow & \ (w2 < 31) \\
\text{Z} \leftarrow & \ (w2) = 0 \\
\text{V} \leftarrow & \ \text{integer overflow} \\
\text{C} \leftarrow & \ \text{borrow in}
\end{align*}

Traps: none

Format: Register

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
15 & 8 7 4 3 0 \\
\hline
1 0 0 1 0 0 1 1 & w1 & w2 \\
\hline
\end{tabular}
\end{table}

Example: \texttt{negw} \ r1,r1 \ # Two's complement (r1)
NOOP

No Operation

Syntax: \texttt{noop} \hspace{5pt} \texttt{bb}

Description: No operation is performed. bb is ignored.

Operation: none

Traps: none

Format: Control

\begin{center}
\begin{tabular}{cccc}
15 & 8 & 7 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \text{bb} \\
\end{tabular}
\end{center}

Examples:
- \texttt{noop} \hspace{5pt} $0$ \hspace{5pt} # No operation
- \texttt{noop} \hspace{5pt} # Operand is optional
NOTQ

Syntax:  \texttt{notq} \ wq,w2

Description:  Zero fill the quick value \( wq \) on the left, take its one's complement and put the result in general register \( w2 \).

Operation:  \( w2 \leftarrow \neg wq \)
N \leftarrow 1
Z \leftarrow 0
V \leftarrow 0
C \leftarrow 0

Traps:  none

Format:  Quick

Example:  \texttt{notq} \ $4,r0
# Load -5 into r0
<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</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>Value</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>
<td></td>
</tr>
<tr>
<td>Register</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Syntax</td>
<td>notw</td>
<td>w1, w2</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>Operation</td>
<td>N ← (w2 &lt; 31)</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>
<tr>
<td>Z ← (w2 = 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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V ← 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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C ← 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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Traps</td>
<td>none</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>
<tr>
<td>Format</td>
<td>NOTW</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>

Example:

```
notw r1, r2

# One's complement r1, put in r2
```

Take the one's complement of the contents of general register w1 and put the result in general register w2.
Syntax: \texttt{ori \ w1,w2}

Description: Bitwise OR the contents of general register \( w2 \) with immediate value \( w1 \), and put the result in \( w2 \). A 16-bit immediate value is sign-extended first.

Operation:
\[
\begin{align*}
  w2 &\leftarrow (w2) \mid w1 \\
  N &\leftarrow (w2 < 31) \\
  Z &\leftarrow (w2) = 0 \\
  V &\leftarrow 0 \\
  C &\leftarrow 0
\end{align*}
\]

Traps: none

Formats: Immediate

If \(-2^{15} \leq w1 \leq 2^{15} - 1\), this format is used:

<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( S )</td>
<td>( wi )</td>
<td>( w2 )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If \(+2^{15} \leq w1 \leq 2^{31} - 1\) or \(-2^{15} \leq w1 \leq -2^{15} - 1\), this format is used:

<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( w1 ) low</td>
<td>( w2 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( S )</td>
<td>( wi ) high</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example: Assume \( r0 \) contains 0x00ff00ff.

\[
\text{ori} \quad 0x00ff, r0 \quad \# \text{Or 0xff with } (r0)
\]

The result put in \( r0 \) is 0x00ff00ff.
Syntax: \texttt{orw \ w1,w2}

Description: Bitwise OR the contents of general register \( w2 \) with the contents of general register \( w1 \) and put the result in \( w2 \).

Operation: \( w2 \leftarrow (w2) \ | (w1) \)
\( N \leftarrow (w2 < 31) \)
\( Z \leftarrow (w2) = 0 \)
\( V \leftarrow 0 \)
\( C \leftarrow 0 \)

Traps: none

Format: Register

\[
\begin{array}{cccccc}
1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & w1 & w2
\end{array}
\]

Example: Assume \( r0 \) contains 0x7770088f and \( r1 \) contains 0x001100ff.

\[
\text{orw \ r1,r0} \quad \# \text{ Or (r0) with (r1)}
\]

The result put in \( r0 \) is 0x777108ff.
POPW

Pop Word

Syntax: \texttt{popw w1,w2}

Description: Pop the word on the top of the stack into general register w2. General register w1 contains the stack address. The stack grows from high to low addresses.

On a data page fault, the contents of w1 will have been incremented. The page fault handler must check for this case and decrement the stack pointer by 4 before restarting the instruction.

Operation: \begin{align*}
w1 & \leftarrow (w1) + 4 \\
w2 & \leftarrow ((w1) - 4)
\end{align*}

Traps: Page fault  
Read protect fault  
Memory fault

Format: Register

\begin{center}
\begin{tabular}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & w1 & w2
\end{tabular}
\end{center}

Example: Assume sp = r15 = stack pointer.

\texttt{popw sp,r0}  \# Pop word into r0
PUSHW
Push Word

Syntax: \texttt{pushw w2,\textit{w1}}

Description: Push the contents of general register \(w2\) onto the stack. General register \(w1\) contains the stack address. The stack grows from high to low addresses.

On a data page fault, the contents of \(w1\) will have been decremented. The page fault handler must check for this case and increment the stack pointer by 4 before restarting the instruction.

Operation: \begin{align*}
\text{w1} & \leftarrow (w1) - 4 \\
(w1) & \leftarrow (w2)
\end{align*}

Traps: Page fault
Write protect fault
Memory fault

Format: Register

\begin{center}
\begin{array}{cccccccc}
15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
\multicolumn{8}{c}{\text{w1}} \\
\multicolumn{8}{c}{\text{w2}} \\
\end{array}
\end{center}

Example: Assume \(\text{sp} = r15\) = stack pointer.

\texttt{pushw \textit{r2},\textit{sp}} \quad \# \text{Push (r2) onto stack}
RESTDn

Syntax:
```
restdn
```

Description:
This description represents the eight instructions restd0 through restd7. The restdn instruction restores the double-precision, floating registers fn through f7 from the stack. The stack pointer is assumed to be in register r15. On a data page fault, the stack pointer is unchanged to permit restarting. A floating register containing a single-precision value will be restored properly by the appropriate restdn instruction, but will not appear in IEEE single-precision format while in memory.

Operation:
```
dn : d7 ← ((r15)) : ((r15) + 8 × [7 − n])
r15 ← (r15) + 8 × [8 − n]
```

Traps:
Page fault
Read protect fault
Memory fault

Format:
```
Macro
```

Example:
```
restd3  # Restore floating registers f3 : f7
restd7  # Restore floating register f7
```
Syntax: `restur w1`

Description: Restore all the user registers. Restore the contents of all user registers r0 through r15 from supervisor memory addressed by supervisor general register w1. Register w1 may be the supervisor stack pointer, r15. On a data page fault, w1 is unchanged to permit restarting.

A privileged instruction trap occurs if this instruction is executed in user mode.

Operation: 
\[
\begin{align*}
    r0 & : r15_{\text{usr}} \leftarrow ((w1)) : ((w1) + 60)_{\text{sup}} \\
    w1 & \leftarrow (w1) + 64
\end{align*}
\]

Traps: Privileged instruction
Page fault
Read protect fault
Memory fault

Format: Macro

\[
\begin{array}{cccccc}
\hline
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & w1 & 0 & 0 & 0 & 0 \\
\hline
31 & 24 & 23 & 20 & 19 & 16
\end{array}
\]

Example: `restur r1` # Restore user's registers
Syntax: \texttt{restwn}

Description: This description represents the 13 instructions \texttt{restw0} through \texttt{restw12}. The \texttt{restwn} instruction restores the general registers \texttt{rn} through \texttt{r14} from the stack. The stack pointer is assumed to be in register \texttt{r15}. On a data page fault, the stack pointer is unchanged to permit restarting.

Operation: \begin{align*}
\texttt{rn} & : \texttt{r14} \leftarrow ((\texttt{r15}) : ((\texttt{r15}) + 4 \times (14 - n))) \\
\texttt{r15} & \leftarrow (\texttt{r15}) + 4 \times [15 - n]
\end{align*}

Traps: Page fault 
Read protect fault
memory fault

Format: Macro

\begin{center}
\begin{array}{c|c|c|c|c|c}
15 & 14 & 13 & 8 & 7 & 32 \\
\hline
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & n \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
31 & 30 & 29 & 24 & 23 & 20 & 19 & 16 \\
\end{array}
\end{center}

Example: \texttt{restw5} \quad \# \text{ Restore registers r5 : r14}
RET

Return from Subroutine

Syntax:
ret w2

Description: Pop the word on top of the stack into the PC. General register w2 contains the stack address. This undoes the effect of the call instruction.

Operation:
PC ← ((w2))
w2 ← (w2) + 4

Traps:
Page fault
Read protect fault
Memory fault

Format:
Register

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</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>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Example: Assume sp = r15 = stack pointer.

    ret sp    # Return to calling program
Return from Interrupt (Privileged)

Syntax: `reti w1`

Description: Return from interrupt or trap. Restore the contents of the SSW, PSW, and PC from the supervisor stack addressed by the contents of general register w1. The supervisor must assure that the stack references do not cause page or protect faults.

A privileged instruction trap occurs if this instruction is executed in user mode.

Operation:
- `SSW ← ((w1))`
- `PSW ← ((w1) + 4)`
- `PC ← ((w1) + 8)`
- `w1 ← (w1) + 12`

Traps: Privileged instruction

Format: Macro

```
  15  8  7  0
  1  0  1  1  0  0  0  0  0  1  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Example: Assume `sp = r15` = stack pointer.

```
  reti  sp  # Return from interrupt
```
**ROT I**

**Rotate Immediate**

**Syntax:**

\[
\text{roti} \ w_1, w_2
\]

**Description:**
Rotate the contents of general register \( w_2 \) by the number of bits given in the 16-bit immediate value \( w_1 \). A positive count rotates the contents of \( w_2 \) to the left, moving bit 31 into bit 0:

![Diagram of left rotation](image)

A negative count rotates to the right, moving bit 0 into bit 31:

![Diagram of right rotation](image)

**Operation:**

\[
w_2 \leftarrow (w_2) \text{ ROT } w_1
\]

\[
N \leftarrow (w_2 < 31 >)
\]

\[
Z \leftarrow (w_2) = 0
\]

\[
V \leftarrow 0
\]

\[
C \leftarrow 0
\]

**Traps:** none

**Format:** Immediate

![Immediate format](image)

**Example:**
Assume \( r_1 \) contains 0x3333ffff.

\[
\text{roti} \quad \$1, r_1 \quad \# \text{ Rotate } r_1 \text{ left 1 place}
\]

The result put in \( r_1 \) is 0x6667ffffe.

Assume \( r_1 \) contains 0x3333ffff.

\[
\text{roti} \quad \$ - 1, r_1 \quad \# \text{ Rotate } r_1 \text{ right 1 place}
\]

The result put in \( r_1 \) is 0x9999ffff.
Syntax: \texttt{rotl } \textit{w1,l2}

Description: Rotate the contents of longword register pair \texttt{l2} by the number of bits given in general register \texttt{w1}. A positive count rotates the contents of \texttt{l2} to the left, moving bit 63 into bit 0:

A negative count rotates the contents of \texttt{w2} to the right, moving bit 0 into bit 63:

Operation:
\begin{align*}
\textit{l2} & \leftarrow (\textit{l2} \text{ ROT (w1)}) \\
\text{N} & \leftarrow (\textit{l2} < 63 >) \\
\text{Z} & \leftarrow (\textit{l2} = 0) \\
\text{V} & \leftarrow 0 \\
\text{C} & \leftarrow 0
\end{align*}

Traps: none

Format: Register

Example: Assume \texttt{r0} contains 2, \texttt{r4,r5} contains \texttt{0x3333ffff 77770000}.

\begin{verbatim}
    rotl     r0,r4     # Rotate r4,r5 left 2 place
\end{verbatim}

The result put in \texttt{r4,r5} is \texttt{0xcccccccccccccccccc80000}.

Assume \texttt{r0} contains \texttt{-2}, \texttt{r4,r5} contains \texttt{0x3333ffff 77770000}.

\begin{verbatim}
    rotl     r0,r4     # Rotate r4,r5 right 2 places
\end{verbatim}

The result put in \texttt{r4,r5} is \texttt{0x0cccccccccccccccc80000}.
**Syntax:**

\[ \text{rotli} \; w_i, l_2 \]

**Description:**
Rotate the contents of longword register pair \( l_2 \) by the number of bits given in the 16-bit immediate value \( w_i \). A positive count rotates the contents of \( l_2 \) to the left, moving bit 63 into bit 0:

A negative count rotates the contents of \( l_2 \) to the right, moving bit 0 into bit 63:

**Operation:**

\[ l_2 \leftarrow (l_2) \text{ ROT } w_i \]

\[ N \leftarrow (l_2 < 63) \]

\[ Z \leftarrow (l_2) = 0 \]

\[ V \leftarrow 0 \]

\[ C \leftarrow 0 \]

**Traps:**

none

**Formats:**

```
  15  8  7  4  3  0
0 0 1 1 1 1 0 1 1 1 1 0 0
```

Example:

\[ \text{rotli} \; \$8, r_2 \]  # Rotate left 1 byte
**Syntax:** \( \text{rotw} \ w1, w2 \)

**Description:** Rotate the contents of general register \( w2 \) by the number of bits given in \( w1 \). A positive count rotates the contents of general register \( w2 \) to the left, moving bit 31 into bit 0:

![Diagram showing rotation left]

A negative count rotates right, moving bit 0 into bit 31:

![Diagram showing rotation right]

**Operation:**
- \( w2 \leftarrow (w2) \text{ ROT} (w1) \)
- \( N \leftarrow (w2 < 31 >) \)
- \( Z \leftarrow (w2) = 0 \)
- \( V \leftarrow 0 \)
- \( C \leftarrow 0 \)

**Traps:** none

**Format:** Register

<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Example:** Assume \( r0 \) contains 8 and \( r2 \) contains 0x000f0071.

\[
\text{rotw} \quad r0, r2 \quad \# \text{Rotate } r2 \text{ left 8 bits}
\]

The result put in \( r2 \) is 0x0f007100.
SAVEDn

Save Registers fn : f7

Syntax: savedn

Description: This description represents the eight instructions saved0 through saved7. The savedn
instructions save the double-precision floating registers dn through d7 on the stack. The
stack pointer is assumed to be in register r15. On a data page fault, the stack pointer is
unchanged to permit restarting. A floating register that contains a single-precision value will
be restored properly by the appropriate restdn instruction, but it will not appear in IEEE
single-precision format while in memory.

Operation: (r15) – 8 × [8 – n] : (r15) ← (dn) : (d7)
r15 ← (r15) – 8 × [8 – n]

Traps: Page fault
Write protect fault
Memory fault

Format: Macro

```
15  8  7  3  2  0
1 0 1 1 0 1 0 0 0 0 1 0 0 n
0 0 0 0 0 0 0 0 0 0 0 0 0
```

Example: saved4 # Save floating registers f4 : f7
Save User Registers (Privileged)

Syntax:  
\texttt{saveur \ w1}

Description: Save the contents of user general registers r0 through r15 in supervisor memory addressed by supervisor general register w1. Register w1 may be the supervisor stack pointer, r15. On a data page fault, w1 is unchanged to permit restarting.

A privileged instruction trap occurs if this instruction is executed outside of supervisor mode.

Operation:  
\[(w1) - 4 : (w1) - 64_{\text{sup}} \leftarrow (r15) : (r0)_{\text{usr}}\]

\[w1 \leftarrow (w1) - 64\]

Traps:  
Privileged instruction
Page fault
Write protect fault
Memory fault

Format: Macro

\[
\begin{array}{cccccccc}
15 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
31 & 24 & 23 & 22 & 21 & 20 & 19 & 18 & 17 & 16
\end{array}
\]

Example:  
\texttt{saveur r1}  \quad \# \text{ Save user's registers r1 : r15}
SAVEWn

Save Registers r{n : r14}

Syntax:  
savewn

Description:  
This description represents the 13 instructions savew0, through savew12. The savewn instruction saves the general registers r{n through r14 on the stack. The stack pointer is assumed to be in register r15. On a data page fault, the stack pointer is unchanged to permit restarting.

Operation:  
(r15) – 4 x [15 – n] : (r15) ← (rn) : (r14)  
r15 ← (r15) – 4 x [15 – n]

Traps:  
Page fault  
Write protect fault  
Memory fault

Format:  
Macro

```
  1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Example:  
savew6  # Save registers r6 : r14
SCALBD

Scale By, Double Floating

Syntax: \( \text{scalbd} \ w1,d2 \)

Description: Multiply the double-precision contents of floating register \( d2 \) by two raised to the integer contents of general register \( w1 \) and put the result in \( d2 \). In the normal case, just the exponent is modified; a multiply operation is not performed. On a trap, the PC and the original value in \( d2 \) can be obtained by using the \text{loadfs} instruction.

Operation: \( d2 \leftarrow (d2) \times 2^{(w1)} \)

FX ← floating inexact result
FU ← floating underflow
FV ← floating overflow
FI ← floating invalid

Traps: Floating inexact result
Floating invalid operation
Floating overflow
Floating underflow

Format: Macro

\[
\begin{array}{cccccccc}
15 & 14 & 13 & 12 & 11 & 10 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
24 & 23 & 22 & 21 & 20 & 19 & 18 & 17 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
31 & 30 & 29 & 28 & 27 & 26 & 25 & 24 \\
1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\
\end{array}
\]

Example: \( \text{scalbd} \ r1,f0 \)  # Scale the floating number
SCALBS

Scale By, Single Floating

Syntax: \texttt{scalbs \ w1,s2}

Description: Multiply the contents of single-precision floating register \( s2 \) by two raised to the integer contents of general register \( w1 \) and put the result in \( s2 \). In the normal case, just the exponent is modified; a multiply operation is not performed. On a trap, the PC and the original value in \( d2 \) can be obtained by using the \texttt{loadfs} instruction.

Operation: \[ s2 \leftarrow (s2) \times 2^{(w1)} \]

\( \text{FX} \leftarrow \text{floating inexact result} \)
\( \text{FU} \leftarrow \text{floating underflow} \)
\( \text{FV} \leftarrow \text{floating overflow} \)
\( \text{FI} \leftarrow \text{floating invalid} \)

Traps: Floating inexact result

Floating invalid operation

Floating overflow

Floating underflow

Format: Macro

\begin{verbatim}
  15  8  7  0
  1 0 1 1 0 1 0 0
  0 0 1 1 1 0 0
  0 0 0 0 0 0 0 0
  w1  s2
\end{verbatim}

Example: \texttt{scalbs \ r1,10} \ # Scale the double floating value
Syntax: \texttt{shai \ wh, w2}

Description: Shift arithmetically the contents of general register \( w2 \) by the number of bits given by the 16-bit immediate value \( wi \). Overflow is set at the end of the operation if the sign of the result changes at any time during the operation. A positive count shifts the contents of general register \( w2 \) left, bringing zeros into bit 0 and shifting the bit 31 out:

A negative count shifts right, bringing in copies of bit 31:

Operation:
\[
\begin{align*}
w2 & \leftarrow (w2) \text{ SHA } wi \\
N & \leftarrow (w2 < 31 >) \\
Z & \leftarrow (w2) = 0 \\
V & \leftarrow \text{ integer overflow} \\
C & \leftarrow 0
\end{align*}
\]

Traps: none

Format: Immediate

Example: Assume \( r2 \) contains \( 0xffffff00 \).

\[
\text{shaw } $8,r2 \\
# \text{ Shift } r2 \text{ left 8 places}
\]

The result put in \( r2 \) is \( 0xff000000 \).

Assume \( r2 \) contains \( 0xffffff00 \).

\[
\text{shaw } $ - 8,r2 \\
# \text{ Shift } r2 \text{ right 8 places}
\]

The result put in \( r2 \) is \( 0xffffff00 \).
Syntax: \texttt{shal \ w1,\l2}

Description: Shift arithmetically the contents of longword register pair \l2 by the number of bits given in \w1. Overflow is set at the end of the operation if the sign of the result changes at any time during the operation. A positive count shifts the contents of \l2 to the left, bringing zeros into bit 0:

A negative count shifts right, bringing in copies of bit 63:

Operation:
\begin{align*}
\l2 & \leftarrow (\l2) \text{ SHA (w1)} \\
N & \leftarrow (\l2 < 63) \\
Z & \leftarrow (\l2) = 0 \\
V & \leftarrow \text{integer overflow} \\
C & \leftarrow 0
\end{align*}

Traps: none

Format: Register

<table>
<thead>
<tr>
<th>15</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Example:
Assume \( r1 \) contains 5, \( r6,r7 \) contains 0x1111ffff00ff00ff.

\begin{verbatim}
shal r1,6 # Shift r6,r7 left 5 places
\end{verbatim}

The result put in \( r6,r7 \) is 0x223ffe01fe01fe0.

Assume \( r1 = -5, r6,r7 = 0xffff111100ff00ff. \)

\begin{verbatim}
shal r1,6 # Shift r6,r7 right 5 places
\end{verbatim}

The result put in \( r6,r7 \) is 0xffff8888 8807f807.
### SHALI

**Shift Arithmetic Longword Immediate**

**Syntax:**

`shali w[i,2]`

**Description:**

Shift arithmetically the contents of longword register pair l2 by the number of bits given in the 16-bit immediate value wi. Overflow is set at the end of the operation if the sign of the result changes at any time during the operation. A positive count shifts the contents of l2 to the left, bringing zeros into bit 0:

![Diagram of left shift]

A negative count shifts right, bringing in copies of bit 63:

![Diagram of right shift]

**Operation:**

- `l2 ← (l2) SHA wi`
- `N ← (w2 < 31 >)`
- `Z ← (w2) = 0`
- `V ← integer overflow`
- `C ← 0`

**Traps:**

None

**Format:**

Immediate

```
  15  8  7  4  3  0
 0 0 1 1 1 0 0 1 1 0 1 1  l2 0
```

**Example:**

Assume r8 contains 0x1110ffff.

```
  shali  $ - 4, r8  # Shift r8 right 4 bits
```

The result put in r8 is 0x01110ff.
Shift Arithmetic Word

Syntax: \texttt{shaw \ w1, w2}

Description: Shift arithmetically the contents of general register \( w2 \) by the number of bits given in general register \( w1 \). Overflow is set at the end of the operation if the sign of the result changes at any time during the operation. A positive count shifts the contents of general register \( w2 \) to the left, bringing zeros into bit 0:

\[
\begin{array}{c}
\text{31} \\
\text{0} \downarrow \\
\text{0} \\
\end{array}
\]

A negative count shifts right, bringing in copies of bit 31:

\[
\begin{array}{c}
\text{31} \\
\text{0} \uparrow \\
\end{array}
\]

Operation:
- \( w2 \leftarrow (w2) \text{ SHA } (w1) \)
- \( N \leftarrow (w2 < 31 >) \)
- \( Z \leftarrow (w2) = 0 \)
- \( V \leftarrow \text{integer overflow} \)
- \( C \leftarrow 0 \)

Traps: none

Format: Register

\[
\begin{array}{cccccccc}
\text{15} & \text{8} & \text{7} & \text{4} & \text{3} & \text{0} \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\quad w1 \quad w2
\]

Example: Assume \( r10 \) contains 8, \( r2 \) contains 0xffff0000.

\[
\text{shaw} \quad r10, r2 \quad \# \text{ Shift } r2 \text{ left 8 places}
\]

The result put in \( r2 \) is 0xff000000.

Assume \( r10 \) contains \(-8\), \( r2 \) contains 0xffff0000.

\[
\text{shaw} \quad r10, r2 \quad \# \text{ Shift } r2 \text{ right 8 places}
\]

The result put in \( r2 \) is 0xfffff00.
SHLI

Shift Logical Immediate

Syntax: \texttt{shli \; wi,\; w2}

Description: Shift logically the contents of general register \( w2 \) by the number of bits given in the 16-bit immediate value \( wi \). A positive count shifts the contents of \( w2 \) out the left; bringing zeros into bit 0:

\[
\begin{array}{c}
31 \\
\vdots \\
0
\end{array}
\]

A negative count shifts the contents of \( w2 \) out the right; bringing zeros into bit 31:

\[
\begin{array}{c}
31 \\
\vdots \\
0
\end{array}
\]

Operation: \( w2 \leftarrow (w2) \text{ SHL } wi \)
\( N \leftarrow (w2 < 31 >) \)
\( Z \leftarrow (w2) = 0 \)
\( V \leftarrow 0 \)
\( C \leftarrow 0 \)

Traps: none

Format: Immediate

Example: Assume \( r4 \) contains 0xffff0000.

\texttt{shli \; \$-8,\; r4} \quad \# \text{Shift \( r4 \) right 8 places}

The result put in \( r4 \) is 0x00ffff00.
Shift Logical Longword

Syntax: \texttt{shll w1,l2}

Description: Shift logically the contents of longword register pair l2 by the number of bits given in general register w1. A positive count shifts the contents of l2 to the left, bringing zeros into bit 0:

![Diagram of shifting left]

A negative count shifts right, bringing zeros into bit 63:

![Diagram of shifting right]

Operation:
\[
\begin{align*}
  l2 &\leftarrow (l2) \text{ SHL} (w1) \\
  N &\leftarrow (l2 < 63>) \\
  Z &\leftarrow (l2) = 0 \\
  V &\leftarrow 0 \\
  C &\leftarrow 0
\end{align*}
\]

Traps: none

Format: Register

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Examples:
Assume r1 contains 16 and r2,r3 contains 0x123456780000ffff.

\texttt{shll r1,r2} \quad \# Shift (r2,r3) by (r1)

The result put in r2,r3 is 0x567800000ffff0000.

Assume r1 contains \(-16\) and r2,r3 contains 0x123456780000ffff.

\texttt{shll r1,r2} \quad \# Shift (r2,r3) by (r1)

The result put in r2,r3 is 0x0000000012345678.
**SHLLI**

**Shift Logical Longword Immediate**

**Syntax:**  
\[ \text{shlli} \quad wi, l2 \]

**Description:**  
Shift logically the contents of longword register pair \( l2 \) by the number of bits given in the 16-bit immediate value \( wi \). A positive count shifts the contents of \( l2 \) to the left, bringing zeros into bit 0:

![Diagram of left shift]

A negative count shifts right, bringing zeros into bit 63:

![Diagram of right shift]

**Operation:**

\[ l2 \leftarrow (l2) \text{ SHL } wi \]
\[ N \leftarrow (l2 < 63 >) \]
\[ V \leftarrow (l2) = 0 \]
\[ Z \leftarrow 0 \]
\[ C \leftarrow 0 \]

**Traps:** none

**Format:** Immediate

![Format Diagram]

**Example:**  
Assume \( r2, r3 \) contains \( 0x777fffffff77777ff \).

\[ \text{shlli} \quad $8, r2 \quad \# \text{ Shift } (r2, r3) \text{ left } 8 \text{ places} \]

The result put in \( r2, r3 \) is \( 0x77ff7777fffffff00 \).

Assume \( r2, r3 \) contains \( 0x7777f00077777ff \).

\[ \text{shlli} \quad $- 8, r2 \quad \# \text{ Shift } (r2, r3) \text{ right } 8 \]

The result put in \( r2, r3 \) is \( 0x00777777777ff \).
**SHLW**  
*Shift Logical Word*

**Syntax:**  
```
shlw w1, w2
```

**Description:**  
Shift logically the contents of general register w2 by the number of bits given in general register w1. A positive count shifts the contents of general register w2 to the left, bringing zeros into bit 0:

![Diagram of left shift]

A negative count shifts right, bringing zeros into bit 31:

![Diagram of right shift]

**Operation:**  
w2 ← (w2) SHL (w1)  
N ← (w2 < 31>)  
Z ← (w2) = 0  
V ← 0  
C ← 0

**Traps:** none

**Format:** Register

```
  15  8  7  4  3  0
  0  0  1  1  0  0  1  0  w1  w2
```

**Examples:**  
Assume r1 contains 4 and r0 contains 0xff.

```
  shl r1, r0  # Shift (r0) by (r1)
```

The result put in r0 is 0x000000ff0.

Assume r1 contains –4 and r0 contains 0xff.

```
  shl r1, r0  # Shift (r0) by (r1)
```

The result put in r0 is 0x000000f0.
STORB

Store Byte

Syntax: \texttt{storb \ w2,ba}

Description: Store the least-significant byte of the contents of general register \texttt{w2} into memory address \texttt{ba}.

Operation: \texttt{ba ← (w2)}

Traps: Page fault
Write protect fault
Memory fault

Format: Address

If addressing is relative, this format is used:

\begin{verbatim}
 15 8 7 4 3 0
 0 1 1 1 1 0 0 0 \texttt{R1} \texttt{w2}
\end{verbatim}

For all other addressing, this format is used:

\begin{verbatim}
 15 8 7 4 3 0
 0 1 1 1 1 0 0 1 \texttt{addr mode}
\end{verbatim}

See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63

Example: \texttt{storb r2,label1}  \# Store (r2) at label1
STORD

STORD

Syntax: stord d2,da

Description: Store the double-precision contents of floating register d2 into memory addresses da.

Operation: da ← (d2)

Traps: Page fault
Write protect fault
Memory fault

Format: Address

If addressing is relative, this format is used:

```
       15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0
     ┌────────────────────────────────┐
     │ 0 1 1 1 0 1 1 0 ┊ R1 ┊ w2 ┊
     └────────────────────────────┘
```

For all other addressing, this format is used:

```
       15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0
     ┌────────────────────────────────┐
     │ 0 1 1 1 0 1 1 1 ┊ addr mode ┊
     └────────────────────────────┘
```

See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63

Example: stord f0,fpsave # Store (f0) at fpsave
STORH

Store Halfword

Syntax:  \texttt{storph \ w2,ha}

Description:  Store the least-significant halfword of the contents of general register \( w2 \) into memory address \( ha \).

Operation:  \( ha \leftarrow (w2) \)

Traps:  Page fault  Write protect fault  Memory fault

Format:  Address

If addressing is relative, this format is used:

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 & R1 & w2
\end{array}
\]

For all other addressing, this format is used:

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & \text{addr mode}
\end{array}
\]

See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63

Example:  \texttt{storph \ r12,hwsav1}  \# Store (r12) at hwsav1
STORS

Store Single Floating

Syntax: \texttt{stors} \texttt{s2,sa}

Description: Store the single-precision contents of floating register \texttt{s2} into memory address \texttt{sa}.

Operation: \texttt{sa} \leftarrow (\texttt{s2})

Traps: Page fault  
Write protect fault  
Memory fault

Format: Address

If addressing is relative, this format is used:

\[
\begin{array}{cccccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
& & & & & R1 & & w2 \\
\end{array}
\]

For all other addressing, this format is used:

\[
\begin{array}{cccccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\
& & & & & \text{addr mode} & & \\
\end{array}
\]

See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63

Example: \texttt{stors} \texttt{f0,sbuff}  
\# Store (f0) at sbuff
Syntax: \texttt{storw} \ w2,wa

Description: Store the contents of general register w2 into memory address wa.

Operation: \ wa \rightarrow (w2)

Traps: Page fault
Write protect fault
Memory fault

Format: Address

If addressing is relative, this format is used:

\begin{center}
\begin{tabular}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
0 & 1 & 1 & 1 & 0 & 0 \\
\end{tabular}
\end{center}

R1 w2

For all other addressing, this format is used:

\begin{center}
\begin{tabular}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
0 & 1 & 1 & 1 & 0 & 0 \\
\end{tabular}
\end{center}

\texttt{addr mode}

See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63

Example: \texttt{storw} r0,waddr \ # Store (r0) at waddr
SUBD

Subtract Double Floating

Syntax: \texttt{subd \ d1,d2}

Description: Subtract the double-precision contents of floating register d1 from the double-precision contents of floating register d2 and put the result in d2.

Operation: \[ d2 \leftarrow (d2) - (d1) \]
FX \leftarrow floating inexact result
FU \leftarrow floating underflow
FV \leftarrow floating overflow
FI \leftarrow floating invalid

Traps: Floating inexact result
Floating invalid operation
Floating overflow
Floating underflow

Format: Register

\[
\begin{array}{ccccccc}
15 & 14 & 13 & 12 & 11 & 10 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & d1 & d2
\end{array}
\]

Example: \texttt{subd f1,f0} # Subtract (f1) from (f0)
Syntax: subi \( w_1, w_2 \)

Description: Subtract the immediate value \( w_1 \) from the contents of general register \( w_2 \) and put the result in \( w_2 \). Operands may be signed or unsigned integers. Overflow is set if the input operands (which are treated as signed integers) have different signs and the sign of the result has the same sign as \( w_1 \).

Operation:
- \( w_2 \leftarrow (w_2) - w_1 \)
- \( N \leftarrow (w_2 < 31 >) \)
- \( Z \leftarrow (w_2) = 0 \)
- \( V \leftarrow \) integer overflow
- \( C \leftarrow \) borrow in

Traps: none

Format: Immediate

If \( -2^{15} \leq w_1 \leq +2^{15} - 1 \), this format is used:

\[
\begin{array}{cccccc}
\hline
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & w_2 \\
\hline
S & wi \\
31 & 30 & & & & & & 16 \\
\end{array}
\]

If \( +2^{15} \leq w_1 \leq +2^{31} - 1 \) or \( -2^{15} \leq w_1 \leq -2^{15} - 1 \), this format is used:

\[
\begin{array}{cccccc}
\hline
15 & 8 & 7 & 4 & 3 & 0 \\
\hline
1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & w_2 \\
\hline
\hline
wi low & & & & & & & & & & & & \\
S & \hline
47 & 46 & & & & & & & & 32 \\
\hline
\end{array}
\]

Example: subi \$0x18,r2 \# Subtract 18 hex from (r2)
SUBQ

Subtract Quick

Syntax: \texttt{subq \ wq, w2}

Description: Subtract the quick value wq from the contents of general register w2 and put the result in w2. Operands may be signed or unsigned integers. Overflow is set if the input operands (which are treated as signed integers) have different signs and the result is positive.

Operation:
- \( w2 \leftarrow (w2) - wq \)
- \( N \leftarrow (w2 < 31 >) \)
- \( Z \leftarrow (w2) = 0 \)
- \( V \leftarrow \text{integer overflow} \)
- \( C \leftarrow \text{borrow in} \)

Traps: none

Format: Quick

\[
\begin{array}{cccccccc}
15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 \\
& 0 & 0 & 0 & 0 & 1 & 0 \\
8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\
& \text{wq} & \text{w2} & \\
0 & & & & & & & \\
\end{array}
\]

Example: \texttt{subq \$10, r3} \quad \# \text{Subtract 10 from (r3)}
Subtract Single Floating

Syntax:  \texttt{subs} \quad s1,s2

Description: Subtract the single-precision contents of floating register \( s1 \) from the single-precision contents of floating register \( s2 \) and put the result in \( d2 \).

Operation: \( s2 \leftarrow (s2) - (s1) \)
FX \( \leftarrow \) floating inexact result
FU \( \leftarrow \) floating underflow
FV \( \leftarrow \) floating overflow
FI \( \leftarrow \) floating invalid

Traps: Floating inexact result
Floating invalid operation
Floating overflow
Floating underflow

Format: Register

\[\begin{array}{ccccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & s1 & s2 \\
\end{array}\]

Example: \texttt{subs f1,f2} \quad \# \text{ Subtract} \ (f1) \text{ from} \ (f2)
Syntax: \texttt{subw \ w1,\ w2}

Description: Subtract the contents of general register \( w1 \) from the contents of general register \( w2 \) and put the result in \( w2 \). Operands may be signed or unsigned integers. Overflow is set if the operands (which are treated as signed integers) have different signs and the sign of the result is the same as the subtrahend \( (w1) \).

Operation: \( w2 \leftarrow (w2) - (w1) \)
\( N \leftarrow (w2 < 31) \)
\( Z \leftarrow (w2) = 0 \)
\( V \leftarrow \text{integer overflow} \)
\( C \leftarrow \text{borrow in} \)

Traps: none

Format: Register

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & w1 & w2
\end{array}
\]

Example: \texttt{subw \ r0,r2} \ # Subtract \( (r0) \) from \( (r2) \)
Syntax: subwc \ w1,w2

Description: Subtract the contents of general register \( w1 \) and the carry condition code from the contents of general register \( w2 \) and put the result in \( w2 \).

Operation: \[ w2 \leftarrow (w2) - (w1) - C \]

\[ N \leftarrow (w2 < 31) \]

\[ Z \leftarrow (w2 = 0) \]

\[ V \leftarrow \text{integer overflow} \]

\[ C \leftarrow \text{borrow in} \]

Traps: none

Format: Register

\[
\begin{array}{cccccc}
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & w1 & w2
\end{array}
\]

Example: subwc r1,r2  # Subtract \((r1) + C\) from \((r2)\)
Syntax: \( \text{trapfn} \)

Description: Causes an illegal instruction trap if a floating unordered condition exists. The IEEE draft standard specifies that the six predicates \( =, \neq, >, \geq, <, \leq \) shall cause floating invalid exceptions on unordered comparisons. The \( \text{trapfn} \) instruction, put before a branch instruction, supports the IEEE predicates. The supervisor trap handler must interpret an illegal instruction trap from a \( \text{trapfn} \) instruction as if it were a floating invalid operation trap.

Operation: if PSW \( <ZN> \) indicates unordered, then illegal instruction trap

Traps: Illegal instruction

Format: Macro

```
15 8 7 4 3 0

1 0 1 1 0 1 0 0 0 0 1 1 1 1 1 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

31 24 23 20 19 16
```

Example: The following sequence branches to \( \text{addr} \) if the IEEE \( \leq \) predicate is satisfied.

```
cmps f3,f5    # Compare single floating values
trapfn       # Checks for floating unordered
bclt addr    # branches
```
TSTS

Test and Set

Syntax:  
tsts  wa,w2

Description:  
Test and set a software lock. Load the contents of memory address wa into general register w2 and set bit 31 in wa (the lock). The operation is indivisible and can be used in a multi-processor configuration. The lock has been acquired if \( (w2_{<31}) = 0 \) after the instruction has executed.

This instruction may not refer to Boot ROM space or I/O space (System Tag 4 or 5).

Operation:  
w2 \rightarrow (wa)
(wa_{<31}) \rightarrow 1

Traps:  
Page fault
Read protect fault
Write protect fault
Memory fault

Format:  
Address

If addressing is relative, this format is used:

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & R1 & w2
\end{array}
\]

For all other addressing, this format is used:

\[
\begin{array}{cccccc}
15 & 8 & 7 & 4 & 3 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & addr \ mode
\end{array}
\]

See Section 1.3, INSTRUCTION FORMATS — WITH ADDRESS for details on bits 0–3 and 16–63

Example:  
tsts  addr,r2  # Set software lock
Wait for Interrupt (Privileged)

Syntax: wait

Description: Wait for an interrupt. When an enabled interrupt occurs, the instruction terminates and the interrupt is taken. The interrupt routine may then decide to whether or not to continue the interrupted instruction stream at the instruction following the wait instruction.

A privileged instruction trap occurs if this instruction is executed in user mode.

Operation: while no interrupt pending
do nothing

Traps: Privileged instruction

Format: Macro

```
  15 14 13 12 11 10  8  7  6  5  4  3  2  1  0
1 0 1 1 0 1 1 0 0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Example: wait  # Wait for interrupt
**XORI**

**Exclusive-OR Immediate**

**Syntax:**
\[ \text{xori} \; \text{wi}, \text{w2} \]

**Description:**
Bitwise exclusive-OR the contents of general register \( \text{w2} \) with the immediate value \( \text{wi} \) and put the result in \( \text{w2} \).

**Operation:**
\[
\begin{align*}
\text{w2} & \leftarrow (\text{w2}) \oplus \text{wi} \\
\text{N} & \leftarrow (\text{w2} < 31) \\
\text{Z} & \leftarrow (\text{w2}) = 0 \\
\text{V} & \leftarrow 0 \\
\text{C} & \leftarrow 0
\end{align*}
\]

**Traps:**
none

**Formats:**
Immediate

If \(-2^{15} \leq \text{wi} \leq +2^{15} - 1\), this format is used:

\[
\begin{array}{cccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
\text{S} & & & & & & & \\
\hline
1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
\text{w2} & & & & & & &
\end{array}
\]

If \(+2^{15} \leq \text{wi} \leq +2^{31} - 1\) or \(-2^{15} \leq \text{wi} \leq -2^{15} - 1\), this format is used:

\[
\begin{array}{cccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
\text{s} & & & & & & & \\
\hline
1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
\text{wi low} & & & & & & &
\end{array}
\]

\[
\begin{array}{cccccccc}
1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
\text{s} & & & & & & & \\
\hline
0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
\text{wi high} & & & & & & &
\end{array}
\]

**Examples:**
Assume \( r0 \) contains 0xff00.
\[
\text{xori} \; \text{0xa07f601f}, \text{r0} \quad \# \; \text{XOR} \; 0x7f601f \text{ with } (\text{r0})
\]
The result put in \( r0 \) is 0xa07f9f1f.

Assume \( r2 \) contains 0xc0ffee12.
\[
\text{xori} \; \text{0xfe}, \text{r2} \quad \# \; \text{XOR} \; 0xfe \text{ with } (\text{r2})
\]
The result put in \( r2 \) is 0xc0ff10.
**XORW**

**Exclusive-OR Word**

**Syntax:** \( \text{xorw} \quad w1,w2 \)

**Description:** Bitwise exclusive-OR the contents of general register \( w1 \) with the contents of general register \( w2 \) and put the result in \( w2 \).

**Operation:**
\[
\begin{align*}
w2 & \leftarrow (w2) \oplus (w1) \\
N & \leftarrow (w2 < 31 >) \\
Z & \leftarrow (w2) = 0 \\
V & \leftarrow 0 \\
C & \leftarrow 0
\end{align*}
\]

**Traps:** none

**Format:** Register

```
15 8 7 4 3 0
1 0 1 0 1 0 0 0  w1  w2
```

**Example:** Assume \( r1 \) contains 0xffff0000 and \( r2 \) contains 0x00ff00ff.

\[
\text{xorw} \quad r1,r2 \quad \# \text{ XOR regs, result in } r2
\]

The result put in \( r2 \) is 0xff0000ff.
# APPENDIX A

## ASCII CHARACTER SET

This appendix lists:
- The ASCII character set
- A description of the non-printing ASCII characters

### Table A-1 ASCII Character Set

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00</td>
<td>NUL (*)</td>
<td>032</td>
<td>20</td>
<td>SP</td>
<td>064</td>
<td>40</td>
<td>@</td>
<td>096</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>01</td>
<td>SOH (*)</td>
<td>033</td>
<td>21</td>
<td>!</td>
<td>065</td>
<td>41</td>
<td>A</td>
<td>097</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>002</td>
<td>02</td>
<td>STX (*)</td>
<td>034</td>
<td>22</td>
<td>&quot;</td>
<td>066</td>
<td>42</td>
<td>B</td>
<td>098</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>003</td>
<td>03</td>
<td>ETX (*)</td>
<td>035</td>
<td>23</td>
<td>#</td>
<td>067</td>
<td>43</td>
<td>C</td>
<td>099</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>004</td>
<td>04</td>
<td>EOT (*)</td>
<td>036</td>
<td>24</td>
<td>$</td>
<td>068</td>
<td>44</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>005</td>
<td>05</td>
<td>ENQ (*)</td>
<td>037</td>
<td>25</td>
<td>%</td>
<td>069</td>
<td>45</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>006</td>
<td>06</td>
<td>ACK (*)</td>
<td>038</td>
<td>26</td>
<td>&amp;</td>
<td>070</td>
<td>46</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>007</td>
<td>07</td>
<td>BEL (*)</td>
<td>039</td>
<td>27</td>
<td>'</td>
<td>071</td>
<td>47</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>008</td>
<td>08</td>
<td>BS (*)</td>
<td>040</td>
<td>28</td>
<td>(</td>
<td>072</td>
<td>48</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>009</td>
<td>09</td>
<td>HT (*)</td>
<td>041</td>
<td>29</td>
<td>)</td>
<td>073</td>
<td>49</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>010</td>
<td>0A</td>
<td>LF (*)</td>
<td>042</td>
<td>2A</td>
<td>*</td>
<td>074</td>
<td>4A</td>
<td>J</td>
<td>106</td>
<td>6A</td>
<td>j</td>
</tr>
<tr>
<td>011</td>
<td>0B</td>
<td>VT (*)</td>
<td>043</td>
<td>2B</td>
<td>+</td>
<td>075</td>
<td>4B</td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>012</td>
<td>0C</td>
<td>FF (*)</td>
<td>044</td>
<td>2C</td>
<td>,</td>
<td>076</td>
<td>4C</td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>013</td>
<td>0D</td>
<td>CR (*)</td>
<td>045</td>
<td>2D</td>
<td>-</td>
<td>077</td>
<td>4D</td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>014</td>
<td>0E</td>
<td>SO (*)</td>
<td>046</td>
<td>2E</td>
<td>.</td>
<td>078</td>
<td>4E</td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>015</td>
<td>0F</td>
<td>SI (*)</td>
<td>047</td>
<td>2F</td>
<td>/</td>
<td>079</td>
<td>4F</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>016</td>
<td>10</td>
<td>DLE (*)</td>
<td>048</td>
<td>30</td>
<td>0</td>
<td>080</td>
<td>50</td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>017</td>
<td>11</td>
<td>DC1 (*)</td>
<td>049</td>
<td>31</td>
<td>1</td>
<td>081</td>
<td>51</td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>018</td>
<td>12</td>
<td>DC2 (*)</td>
<td>050</td>
<td>32</td>
<td>2</td>
<td>082</td>
<td>52</td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>019</td>
<td>13</td>
<td>DC3 (*)</td>
<td>051</td>
<td>33</td>
<td>3</td>
<td>083</td>
<td>53</td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>020</td>
<td>14</td>
<td>DC4 (*)</td>
<td>052</td>
<td>34</td>
<td>4</td>
<td>084</td>
<td>54</td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>021</td>
<td>15</td>
<td>NAK (*)</td>
<td>053</td>
<td>35</td>
<td>5</td>
<td>085</td>
<td>55</td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>022</td>
<td>16</td>
<td>SYN (*)</td>
<td>054</td>
<td>36</td>
<td>6</td>
<td>086</td>
<td>56</td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>023</td>
<td>17</td>
<td>ETB (*)</td>
<td>055</td>
<td>37</td>
<td>7</td>
<td>087</td>
<td>57</td>
<td>W</td>
<td>119</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>024</td>
<td>18</td>
<td>CAN (*)</td>
<td>056</td>
<td>38</td>
<td>8</td>
<td>088</td>
<td>58</td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>025</td>
<td>19</td>
<td>EM (*)</td>
<td>057</td>
<td>39</td>
<td>9</td>
<td>089</td>
<td>59</td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>026</td>
<td>1A</td>
<td>SUB (*)</td>
<td>058</td>
<td>3A</td>
<td>:</td>
<td>090</td>
<td>5A</td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>027</td>
<td>1B</td>
<td>ESC (*)</td>
<td>059</td>
<td>3B</td>
<td>;</td>
<td>091</td>
<td>5B</td>
<td>]</td>
<td>123</td>
<td>7B</td>
<td>{</td>
</tr>
<tr>
<td>028</td>
<td>1C</td>
<td>FS (*)</td>
<td>060</td>
<td>3C</td>
<td>&lt;</td>
<td>092</td>
<td>5C</td>
<td>\</td>
<td>124</td>
<td>7C</td>
<td>ALT</td>
</tr>
<tr>
<td>029</td>
<td>1D</td>
<td>GS (*)</td>
<td>061</td>
<td>3D</td>
<td>=</td>
<td>093</td>
<td>5D</td>
<td>}</td>
<td>125</td>
<td>7D</td>
<td>ESC</td>
</tr>
<tr>
<td>030</td>
<td>1E</td>
<td>RS (*)</td>
<td>062</td>
<td>3E</td>
<td>&gt;</td>
<td>094</td>
<td>5E</td>
<td>^</td>
<td>126</td>
<td>7E</td>
<td>DEL</td>
</tr>
<tr>
<td>031</td>
<td>1F</td>
<td>US (*)</td>
<td>063</td>
<td>3F</td>
<td>?</td>
<td>095</td>
<td>5F</td>
<td>_</td>
<td>127</td>
<td>7F</td>
<td></td>
</tr>
<tr>
<td>Char</td>
<td>Definition</td>
<td>Char</td>
<td>Definition</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>
</tr>
<tr>
<td>NUL</td>
<td>Null</td>
<td>DC2</td>
<td>Device Control 2 (TAPE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOH</td>
<td>Start of message</td>
<td>DC3</td>
<td>Device Control 3 (X-OFF)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STX</td>
<td>Start of Text</td>
<td>DC4</td>
<td>Device Control 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ETX</td>
<td>End of Text</td>
<td>NAK</td>
<td>Negative Acknowledge</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EOT</td>
<td>End of Transmission</td>
<td>SYN</td>
<td>Synchronous idle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENQ</td>
<td>Enquiry</td>
<td>ETB</td>
<td>End of Transmission Block</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACK</td>
<td>Acknowledge</td>
<td>CAN</td>
<td>Cancel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEL</td>
<td>Bell</td>
<td>EM</td>
<td>End of Medium</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td>Back Space</td>
<td>SUB</td>
<td>Substitute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HT</td>
<td>Horizontal Tab</td>
<td>ESC</td>
<td>Escape</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LF</td>
<td>Line Feed</td>
<td>FS</td>
<td>File Separator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VT</td>
<td>Vertical Tab</td>
<td>GS</td>
<td>Group Separator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>Form Feed</td>
<td>RS</td>
<td>Record Separator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CR</td>
<td>Carriage Return</td>
<td>US</td>
<td>Unit Separator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SO</td>
<td>Shift Out</td>
<td>SP</td>
<td>Space</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SI</td>
<td>Shift In</td>
<td>ALT</td>
<td>Alt mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLE</td>
<td>Data Link Escape</td>
<td>ESC</td>
<td>Escape prefix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC1</td>
<td>Device Control 1 (X-ON)</td>
<td>DEL</td>
<td>Delete, Rubout</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX B
INSTRUCTION SUMMARY

This appendix summarizes the instruction set. For a detailed description of each instruction, see Chapter 2.

### Table B-1  Functional Instruction Set

<table>
<thead>
<tr>
<th>Instruction Name</th>
<th>Syntax</th>
<th>Opcode</th>
<th>Format</th>
<th>Parcels</th>
<th>Operation</th>
<th>PSW Flags</th>
<th>IVDUX</th>
<th>CVZN</th>
<th>Traps</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LOAD/STORE INSTRUCTIONS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load Address</td>
<td>loada</td>
<td>ba,w2</td>
<td>62,63</td>
<td>Address</td>
<td>1-4 w2 ← ba</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load Byte</td>
<td>loadb</td>
<td>ba,w2</td>
<td>68,69</td>
<td>Address</td>
<td>1-4 w2 ← (ba)</td>
<td></td>
<td></td>
<td></td>
<td>PR,M,</td>
</tr>
<tr>
<td>Load Byte Unsigned</td>
<td>loadbu</td>
<td>ba,w2</td>
<td>6a,6b</td>
<td>Address</td>
<td>1-4 w2 ← (ba)</td>
<td></td>
<td></td>
<td></td>
<td>PR,M,</td>
</tr>
<tr>
<td>Load Double Floating</td>
<td>loadd</td>
<td>da,d2</td>
<td>66,67</td>
<td>Address</td>
<td>1-4 d2 ← (da)</td>
<td></td>
<td></td>
<td></td>
<td>PR,M,</td>
</tr>
<tr>
<td>Load Floating Status</td>
<td>loadfs</td>
<td>w1,d2</td>
<td>b4,3f</td>
<td>Macro</td>
<td>2 w1 ← (FP PC), d2 ← (FP dest)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load Halfword</td>
<td>loadh</td>
<td>ha,w2</td>
<td>6c,6d</td>
<td>Address</td>
<td>1-4 w2 ← (ha)</td>
<td></td>
<td></td>
<td></td>
<td>PR,M,</td>
</tr>
<tr>
<td>Load Halfword Unsigned</td>
<td>loadhu</td>
<td>ha,w2</td>
<td>6e,6f</td>
<td>Address</td>
<td>1-4 w2 ← (ha)</td>
<td></td>
<td></td>
<td></td>
<td>PR,M,</td>
</tr>
<tr>
<td>Load Immediate</td>
<td>loadi</td>
<td>wi,w2</td>
<td>87</td>
<td>Immediate</td>
<td>2,3 w2 ← wi</td>
<td></td>
<td></td>
<td>0**</td>
<td></td>
</tr>
<tr>
<td>Load Quick</td>
<td>loadq</td>
<td>wq,w2</td>
<td>86</td>
<td>Quick</td>
<td>1 w2 ← wq</td>
<td></td>
<td></td>
<td>0**</td>
<td></td>
</tr>
<tr>
<td>Load Single Floating</td>
<td>loads</td>
<td>sa,s2</td>
<td>64,65</td>
<td>Address</td>
<td>1-4 s2 ← (sa)</td>
<td></td>
<td></td>
<td></td>
<td>PWM,</td>
</tr>
<tr>
<td>Load Word</td>
<td>loadw</td>
<td>wa,w2</td>
<td>60,61</td>
<td>Address</td>
<td>1-4 w2 ← (wa)</td>
<td></td>
<td></td>
<td></td>
<td>PWM,</td>
</tr>
<tr>
<td>Store Byte</td>
<td>storb</td>
<td>w2,ba</td>
<td>78,79</td>
<td>Address</td>
<td>1-4 ba ← (w2)</td>
<td></td>
<td></td>
<td></td>
<td>PWM,</td>
</tr>
<tr>
<td>Store Double Floating</td>
<td>stord</td>
<td>d2,de</td>
<td>76,77</td>
<td>Address</td>
<td>1-4 da ← (d2)</td>
<td></td>
<td></td>
<td></td>
<td>PWM,</td>
</tr>
<tr>
<td>Store Halfword</td>
<td>storh</td>
<td>w2,ha</td>
<td>7c,7d</td>
<td>Address</td>
<td>1-4 ha ← (w2)</td>
<td></td>
<td></td>
<td></td>
<td>PWM,</td>
</tr>
<tr>
<td>Store Single Floating</td>
<td>stors</td>
<td>s2,sa</td>
<td>74,75</td>
<td>Address</td>
<td>1-4 sa ← (s2)</td>
<td></td>
<td></td>
<td></td>
<td>PWM,</td>
</tr>
<tr>
<td>Store Word</td>
<td>storw</td>
<td>w2,wa</td>
<td>70,71</td>
<td>Address</td>
<td>1-4 wa ← (w2)</td>
<td></td>
<td></td>
<td></td>
<td>PWM,</td>
</tr>
</tbody>
</table>

**DATA MOVEMENT INSTRUCTIONS**

<table>
<thead>
<tr>
<th>Instruction Name</th>
<th>Syntax</th>
<th>Opcode</th>
<th>Format</th>
<th>Parcels</th>
<th>Operation</th>
<th>PSW Flags</th>
<th>IVDUX</th>
<th>CVZN</th>
<th>Traps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move Double Floating</td>
<td>movd</td>
<td>d1,d2</td>
<td>26</td>
<td>Register</td>
<td>1 d2 ← (d1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move Double Floating to Longword</td>
<td>movdl</td>
<td>d1,l2</td>
<td>2a</td>
<td>Register</td>
<td>1 l2 ← (d1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move Longword to Double Floating</td>
<td>movld</td>
<td>l1,d2</td>
<td>2f</td>
<td>Register</td>
<td>1 d2 ← (l1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move Processor Register to Word</td>
<td>movpw</td>
<td>p1,w2</td>
<td>11</td>
<td>Register</td>
<td>1 w2 ← (p1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move Single Floating</td>
<td>moves</td>
<td>s1,s2</td>
<td>24</td>
<td>Register</td>
<td>1 s2 ← (s1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move Supervisor to User (privileged)</td>
<td>movsu</td>
<td>w1,w2</td>
<td>b6,01</td>
<td>Macro</td>
<td>2 w2 ← (w1)</td>
<td></td>
<td></td>
<td>0**</td>
<td>S</td>
</tr>
<tr>
<td>Move Single Floating to Word</td>
<td>movsw</td>
<td>s1,w2</td>
<td>2c</td>
<td>Register</td>
<td>1 w2 ← (s1)</td>
<td></td>
<td></td>
<td>0**</td>
<td>S</td>
</tr>
<tr>
<td>Move User to Supervisor (privileged)</td>
<td>movusu</td>
<td>w1,b2</td>
<td>b6,00</td>
<td>Macro</td>
<td>2 w2 ← (w1)</td>
<td></td>
<td></td>
<td>0**</td>
<td>S</td>
</tr>
<tr>
<td>Move Word</td>
<td>movw</td>
<td>w1,w2</td>
<td>84</td>
<td>Register</td>
<td>1 w2 ← (w1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move Word to Processor Register</td>
<td>movwp</td>
<td>w2,p2</td>
<td>10</td>
<td>Register</td>
<td>1 p2 ← (w2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move Word to Single Floating</td>
<td>movws</td>
<td>w1,s2</td>
<td>2d</td>
<td>Register</td>
<td>1 s2 ← (w1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend:  
PSW Flags Field  
D = Divide-by-zero  
I = Illegal instruction  
M = Memory fault  
P = Page fault  
R = Read protect fault  
S = Supervisor only (privileged) instruction  
W = Write protect fault
### Table B-1 Functional Instruction Set (Continued)

<table>
<thead>
<tr>
<th>Instruction Name</th>
<th>Syntax</th>
<th>Opcode</th>
<th>Format</th>
<th>Parcels</th>
<th>Operation</th>
<th>PSW Flags</th>
<th>Traps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Double Floating</td>
<td><code>add d1,d2</code></td>
<td>22</td>
<td>Register</td>
<td>1</td>
<td><code>d2 ← (d2) + (d1)</code></td>
<td><strong>. .</strong></td>
<td></td>
</tr>
<tr>
<td>Add Immediate</td>
<td><code>add w1,w2</code></td>
<td>83</td>
<td>Immediate</td>
<td>2,3</td>
<td><code>w2 ← (w2) + wi</code></td>
<td>. . . .</td>
<td></td>
</tr>
<tr>
<td>Add Quick</td>
<td><code>addq wq,w2</code></td>
<td>62</td>
<td>Quick</td>
<td>1</td>
<td><code>w2 ← (w2) + wq</code></td>
<td>. . . .</td>
<td></td>
</tr>
<tr>
<td>Add Single Floating</td>
<td><code>adds s1,s2</code></td>
<td>20</td>
<td>Register</td>
<td>1</td>
<td><code>s2 ← (s2) + (s1)</code></td>
<td><strong>. .</strong></td>
<td></td>
</tr>
<tr>
<td>Add Word</td>
<td><code>addw w1,w2</code></td>
<td>80</td>
<td>Register</td>
<td>1</td>
<td><code>w2 ← (w2) + (w1)</code></td>
<td>. . . .</td>
<td></td>
</tr>
<tr>
<td>Add Word with Carry</td>
<td><code>addwc w1,w2</code></td>
<td>90</td>
<td>Register</td>
<td>1</td>
<td><code>w2 ← (w2) + (w1) + C</code></td>
<td><strong>. .</strong></td>
<td></td>
</tr>
<tr>
<td>Subtract Double Floating</td>
<td><code>subd d1,d2</code></td>
<td>23</td>
<td>Register</td>
<td>1</td>
<td><code>d2 ← (d2) − (d1)</code></td>
<td>. . . .</td>
<td></td>
</tr>
<tr>
<td>Subtract Immediate</td>
<td><code>subi w1,w2</code></td>
<td>a3</td>
<td>Immediate</td>
<td>2,3</td>
<td><code>w2 ← (w2) − wi</code></td>
<td><strong>. .</strong></td>
<td></td>
</tr>
<tr>
<td>Subtract Quick</td>
<td><code>subq wq,w2</code></td>
<td>a2</td>
<td>Quick</td>
<td>1</td>
<td><code>w2 ← (w2) − wq</code></td>
<td>. . . .</td>
<td></td>
</tr>
<tr>
<td>Subtract Single Floating</td>
<td><code>subw s1,s2</code></td>
<td>21</td>
<td>Register</td>
<td>1</td>
<td><code>s2 ← (s2) − (s1)</code></td>
<td><strong>. .</strong></td>
<td></td>
</tr>
<tr>
<td>Subtract Word</td>
<td><code>subw w1,w2</code></td>
<td>a0</td>
<td>Register</td>
<td>1</td>
<td><code>w2 ← (w2) − (w1)</code></td>
<td>. . . .</td>
<td></td>
</tr>
<tr>
<td>Subtract Word with Carry</td>
<td><code>subwc w1,w2</code></td>
<td>91</td>
<td>Register</td>
<td>1</td>
<td><code>w2 ← (w2) − (w1) − C</code></td>
<td>. . . .</td>
<td></td>
</tr>
<tr>
<td>Multiply Double Floating</td>
<td><code>mul d1,d2</code></td>
<td>2a</td>
<td>Register</td>
<td>1</td>
<td><code>d2 ← (d2) × (d1)</code></td>
<td>***** .</td>
<td></td>
</tr>
<tr>
<td>Multiply Single Floating</td>
<td><code>mul s1,s2</code></td>
<td>28</td>
<td>Register</td>
<td>1</td>
<td><code>s2 ← (s2) × (s1)</code></td>
<td>. . . .</td>
<td>.0000</td>
</tr>
<tr>
<td>Multiply Word</td>
<td><code>mulw w1,w2</code></td>
<td>98</td>
<td>Register</td>
<td>1</td>
<td><code>w2 ← (w2) × (w1)</code></td>
<td>.0000</td>
<td>.0000</td>
</tr>
<tr>
<td>Multiply Word Unsigned</td>
<td><code>mulwu w1,w2</code></td>
<td>9a</td>
<td>Register</td>
<td>1</td>
<td><code>w2 ← (w2) × (w1)</code></td>
<td>.0000</td>
<td>.0000</td>
</tr>
<tr>
<td>Multiply Word Unsigned Extended</td>
<td><code>mulwux w1,l2</code></td>
<td>9b</td>
<td>Register</td>
<td>1</td>
<td><code>l2 ← (w2) × (w1)</code></td>
<td>.0000</td>
<td>.0000</td>
</tr>
<tr>
<td>Divide Double Floating</td>
<td><code>divd d1,d2</code></td>
<td>2b</td>
<td>Register</td>
<td>1</td>
<td><code>d2 ← (d2) ÷ (d1)</code></td>
<td>***** .</td>
<td></td>
</tr>
<tr>
<td>Divide Single Floating</td>
<td><code>divs s1,s2</code></td>
<td>29</td>
<td>Register</td>
<td>1</td>
<td><code>s2 ← (s2) ÷ (s1)</code></td>
<td><strong>. .</strong></td>
<td>.0000</td>
</tr>
<tr>
<td>Divide Word</td>
<td><code>divw w1,w2</code></td>
<td>9c</td>
<td>Register</td>
<td>1</td>
<td><code>w2 ← (w2) ÷ (w1)</code></td>
<td>.0000</td>
<td>.0000</td>
</tr>
<tr>
<td>Divide WordUnsigned</td>
<td><code>divwu w1,l2</code></td>
<td>9e</td>
<td>Register</td>
<td>1</td>
<td><code>w2 ← (w2) ÷ (w1)</code></td>
<td>.0000</td>
<td>.0000</td>
</tr>
<tr>
<td>Negate Double Floating</td>
<td><code>negd d1,d2</code></td>
<td>b4</td>
<td>3b</td>
<td>Macro</td>
<td>2 <code>d2 ← (d1)</code></td>
<td>. . . .</td>
<td></td>
</tr>
<tr>
<td>Negate Single Floating</td>
<td><code>negs s1,s2</code></td>
<td>b4</td>
<td>3a</td>
<td>Macro</td>
<td>2 <code>s2 ← (s1)</code></td>
<td>. . . .</td>
<td></td>
</tr>
<tr>
<td>Negate Word</td>
<td><code>negw w1,w2</code></td>
<td>93</td>
<td>Register</td>
<td>1</td>
<td><code>w2 ← (w1)</code></td>
<td>. . . .</td>
<td></td>
</tr>
<tr>
<td>Modulus Word</td>
<td><code>mod w1,w2</code></td>
<td>9d</td>
<td>Register</td>
<td>1</td>
<td><code>w2 ← (w2) MOD (w1)</code></td>
<td>.0000</td>
<td>.0000</td>
</tr>
<tr>
<td>Modulus Word Unsigned</td>
<td><code>modwu w1,w2</code></td>
<td>9f</td>
<td>Register</td>
<td>1</td>
<td><code>w2 ← (w2) MOD (w1)</code></td>
<td>.0000</td>
<td>.0000</td>
</tr>
<tr>
<td>Scale by, Double Floating</td>
<td><code>scalbd w1,d2</code></td>
<td>b4</td>
<td>3d</td>
<td>Macro</td>
<td>2 <code>d2 ← (d2) × 2(w1)</code></td>
<td><strong>. .</strong></td>
<td></td>
</tr>
<tr>
<td>Scale by, Single Floating</td>
<td><code>scals s1,s2</code></td>
<td>b4</td>
<td>3c</td>
<td>Macro</td>
<td>2 <code>s2 ← (s2) × 2(w1)</code></td>
<td><strong>. .</strong></td>
<td></td>
</tr>
</tbody>
</table>

Legend: PSW Flags Field

. = Flag not affected by instruction
* = Flag set according to operation
0 = Flag set to 0
1 = Flag set to 1

Traps Field

D = Divide-by-zero
I = Illegal instruction
M = Memory fault
P = Page fault
R = Read protect fault
S = Supervisor only (privileged) instruction
W = Write protect fault
<table>
<thead>
<tr>
<th>Instruction Name</th>
<th>Syntax</th>
<th>Opcode</th>
<th>Format</th>
<th>Parcels</th>
<th>Operation</th>
<th>PSW Flags</th>
<th>Traps</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGICAL INSTRUCTIONS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VIDUX CVZN Traps</td>
<td></td>
</tr>
<tr>
<td>And Immediate</td>
<td>andi</td>
<td>wi,w2</td>
<td>8b</td>
<td>Immediate</td>
<td>2,3 w2 ← (w2) &amp; wi</td>
<td>00**</td>
<td></td>
</tr>
<tr>
<td>And Word</td>
<td>andw</td>
<td>w1,w2</td>
<td>88</td>
<td>Register</td>
<td>1 w2 ← (w2) &amp; (w1)</td>
<td>00**</td>
<td></td>
</tr>
<tr>
<td>Or Immediate</td>
<td>ori</td>
<td>wi,w2</td>
<td>8f</td>
<td>Immediate</td>
<td>2,3 w2 ← (w2)</td>
<td>wi</td>
<td>00**</td>
</tr>
<tr>
<td>Or Word</td>
<td>orw</td>
<td>w1,w2</td>
<td>8c</td>
<td>Register</td>
<td>1 w2 ← (w2)</td>
<td>(w1)</td>
<td>00**</td>
</tr>
<tr>
<td>Exclusive-OR Immediate</td>
<td>xorl</td>
<td>wi,w2</td>
<td>ab</td>
<td>Immediate</td>
<td>2,3 w2 ← (w2) &amp; wi</td>
<td>00**</td>
<td></td>
</tr>
<tr>
<td>Exclusive-OR Word</td>
<td>xorw</td>
<td>w1,w2</td>
<td>a8</td>
<td>Register</td>
<td>1 w2 ← (w2) &amp; (w1)</td>
<td>00**</td>
<td></td>
</tr>
<tr>
<td>Not Word</td>
<td>notb</td>
<td>w1,w2</td>
<td>ac</td>
<td>Register</td>
<td>1 w2 ← ~ (w1)</td>
<td>00**</td>
<td></td>
</tr>
<tr>
<td>Not Quick</td>
<td>notq</td>
<td>wq,w2</td>
<td>ae</td>
<td>Quick</td>
<td>1 w2 ← ~ wq</td>
<td>0001</td>
<td></td>
</tr>
<tr>
<td>SHIFT/ROTATE INSTRUCTIONS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VIDUX CVZN Traps</td>
<td></td>
</tr>
<tr>
<td>Shift Arithmetic Immediate</td>
<td>shal</td>
<td>wi,w2</td>
<td>38</td>
<td>Immediate</td>
<td>2 w2 ← (w2) SHA wi</td>
<td>00**</td>
<td></td>
</tr>
<tr>
<td>Shift Arithmetic Longword</td>
<td>shal</td>
<td>w1,l2</td>
<td>31</td>
<td>Register</td>
<td>1 l2 ← (l2) SHA (w1)</td>
<td>00**</td>
<td></td>
</tr>
<tr>
<td>Shift Arithmetic Longword Immediate</td>
<td>shali</td>
<td>wi,l2</td>
<td>39</td>
<td>Immediate</td>
<td>2 l2 ← (l2) SHA wi</td>
<td>00**</td>
<td></td>
</tr>
<tr>
<td>Shift Arithmetic Word</td>
<td>shaw</td>
<td>w1,w2</td>
<td>30</td>
<td>Register</td>
<td>1 w2 ← (w2) SHA (w1)</td>
<td>00**</td>
<td></td>
</tr>
<tr>
<td>Shift Logical Immediate</td>
<td>shli</td>
<td>wi,w2</td>
<td>3a</td>
<td>Immediate</td>
<td>2 w2 ← (w2) SHL wi</td>
<td>00**</td>
<td></td>
</tr>
<tr>
<td>Shift Logical Word</td>
<td>shll</td>
<td>w1,l2</td>
<td>33</td>
<td>Register</td>
<td>1 l2 ← (l2) SHL (w1)</td>
<td>00**</td>
<td></td>
</tr>
<tr>
<td>Shift Logical Longword</td>
<td>shlli</td>
<td>wi,l2</td>
<td>3b</td>
<td>Immediate</td>
<td>2 l2 ← (l2) SHL wi</td>
<td>00**</td>
<td></td>
</tr>
<tr>
<td>Rotate Immediate</td>
<td>rotl</td>
<td>wi,w2</td>
<td>3c</td>
<td>Immediate</td>
<td>2 w2 ← (w2) ROT wi</td>
<td>00**</td>
<td></td>
</tr>
<tr>
<td>Rotate Longword</td>
<td>rotw</td>
<td>w1,l2</td>
<td>35</td>
<td>Register</td>
<td>1 l2 ← (l2) ROT (w1)</td>
<td>00**</td>
<td></td>
</tr>
<tr>
<td>Rotate Longword Immediate</td>
<td>rotli</td>
<td>wi,l2</td>
<td>3d</td>
<td>Immediate</td>
<td>2 l2 ← (l2) ROT wi</td>
<td>00**</td>
<td></td>
</tr>
<tr>
<td>Rotate Word</td>
<td>rotw</td>
<td>w1,w2</td>
<td>34</td>
<td>Register</td>
<td>1 w2 ← (w2) ROT (w1)</td>
<td>00**</td>
<td></td>
</tr>
<tr>
<td>CONVERSION INSTRUCTIONS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VIDUX CVZN Traps</td>
<td></td>
</tr>
<tr>
<td>Convert Double Floating to Single</td>
<td>cnvds</td>
<td>d1,s2</td>
<td>b4 39</td>
<td>Macro</td>
<td>2 s2 ← (d1)</td>
<td>0**</td>
<td></td>
</tr>
<tr>
<td>Convert Double Floating to Word</td>
<td>cnvwd</td>
<td>d1,w2</td>
<td>b4 34</td>
<td>Macro</td>
<td>2 w2 ← (d1)</td>
<td>0**</td>
<td></td>
</tr>
<tr>
<td>Convert Rounding Double to Word</td>
<td>cnrvd</td>
<td>d1,w2</td>
<td>b4 35</td>
<td>Macro</td>
<td>2 w2 ← (d1)</td>
<td>0**</td>
<td></td>
</tr>
<tr>
<td>Convert Single Floating to Double</td>
<td>cnrvd</td>
<td>s1,d2</td>
<td>b4 38</td>
<td>Macro</td>
<td>2 d2 ← (s1)</td>
<td>0**</td>
<td></td>
</tr>
<tr>
<td>Convert Single Floating to Word</td>
<td>cnrvw</td>
<td>s1,w2</td>
<td>b4 30</td>
<td>Macro</td>
<td>2 w2 ← (s1)</td>
<td>0**</td>
<td></td>
</tr>
<tr>
<td>Convert Truncating Double to word</td>
<td>cnrvtd</td>
<td>d1,w2</td>
<td>b4 36</td>
<td>Macro</td>
<td>2 w2 ← (d1)</td>
<td>0**</td>
<td></td>
</tr>
<tr>
<td>Convert Truncating Single to word</td>
<td>cnrvtw</td>
<td>s1,w2</td>
<td>b4 32</td>
<td>Macro</td>
<td>2 w2 ← (s1)</td>
<td>0**</td>
<td></td>
</tr>
<tr>
<td>Convert Word to Double Floating</td>
<td>cnrvwd</td>
<td>w1,d2</td>
<td>b4 37</td>
<td>Macro</td>
<td>2 d2 ← (w1)</td>
<td>0**</td>
<td></td>
</tr>
<tr>
<td>Convert Word to Single Floating</td>
<td>cnrvws</td>
<td>w1,s2</td>
<td>b4 33</td>
<td>Macro</td>
<td>2 s2 ← (w1)</td>
<td>0**</td>
<td></td>
</tr>
</tbody>
</table>

Legend:  
PSW Flags Field  
D = Divide-by-zero  
I = Illegal instruction  
M = Memory fault  
P = Page fault  
R = Read protect fault  
S = Supervisor only (privileged) instruction  
W = Write protect fault
## Table B-1 Functional Instruction Set (Continued)

### COMPARE AND TEST INSTRUCTIONS

<table>
<thead>
<tr>
<th>Instruction Name</th>
<th>Syntax</th>
<th>Opcode</th>
<th>Format</th>
<th>Parcels</th>
<th>Operation</th>
<th>PSW Flags</th>
<th>IVDUX</th>
<th>CVZN</th>
<th>Traps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare Double Floating</td>
<td>cmpd</td>
<td>d1,d2</td>
<td>27</td>
<td>Register</td>
<td>$1 \ (d2) - (d1)$</td>
<td>. . . 0 * *</td>
<td></td>
<td></td>
<td>P, R, W, M</td>
</tr>
<tr>
<td>Compare Immediate</td>
<td>cmpl</td>
<td>w1,w2</td>
<td>a7</td>
<td>Immediate</td>
<td>$2,3 \ (w2) - w1$</td>
<td>. . . * * *</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compare Quick</td>
<td>cmpq</td>
<td>wq,w2</td>
<td>a6</td>
<td>Quick</td>
<td>$1 \ (w2) - wq$</td>
<td>. . . 0 * *</td>
<td></td>
<td></td>
<td>P, R, W, M</td>
</tr>
<tr>
<td>Compare Single Floating</td>
<td>cmps</td>
<td>s1,s2</td>
<td>25</td>
<td>Register</td>
<td>$1 \ (s2) - (s1)$</td>
<td>. . . * * *</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compare Word</td>
<td>cmpw</td>
<td>w1,w2</td>
<td>a4</td>
<td>Register</td>
<td>$1 \ (w2) - (w1)$</td>
<td>. . . * * *</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test and Set</td>
<td>tsts</td>
<td>wa,w2</td>
<td>72,73</td>
<td>Address</td>
<td>$1 \ w2 \leftarrow (w_a), w_a &lt; 31 &gt; \leftarrow 1$</td>
<td>. . .</td>
<td></td>
<td></td>
<td>P, R, W, M</td>
</tr>
</tbody>
</table>

### CHARACTER STRING INSTRUCTIONS

<table>
<thead>
<tr>
<th>Instruction Name</th>
<th>Syntax</th>
<th>Opcode</th>
<th>Format</th>
<th>Parcels</th>
<th>Operation</th>
<th>PSW Flags</th>
<th>IVDUX</th>
<th>CVZN</th>
<th>Traps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare Characters</td>
<td>cmpc</td>
<td>b4 0f</td>
<td>Macro</td>
<td>2</td>
<td>WHILE [(r0) = 0] &amp; [(r2) = (r1)], \ r0 \leftarrow \ (r0) - 1, r1 \leftarrow \ (r1) + 1, r2 \leftarrow \ (r2) + 1</td>
<td>. . . * * *</td>
<td>P, R, M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initialize Characters</td>
<td>initc</td>
<td>b4 0e</td>
<td>Macro</td>
<td>2</td>
<td>WHILE \ (r1) \leftarrow \ (r1 &lt; 7:0 &gt;), \ r0 \leftarrow \ (r0) - 1, r1 \leftarrow \ (r1) + 1, r2 \leftarrow \ (r2) \text{ROT - 8}</td>
<td>. . .</td>
<td>P, W, M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move Characters</td>
<td>movc</td>
<td>b4 0d</td>
<td>Macro</td>
<td>2</td>
<td>WHILE \ (r0) = 0, \ (r2) \leftarrow \ (r1), \ r0 \leftarrow \ (r0) - 1, r1 \leftarrow \ (r1) + 1, r2 \leftarrow \ (r2) + 1</td>
<td>. . .</td>
<td>P, R, W, M</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### STACK MANIPULATION INSTRUCTIONS

<table>
<thead>
<tr>
<th>Instruction Name</th>
<th>Syntax</th>
<th>Opcode</th>
<th>Format</th>
<th>Parcels</th>
<th>Operation</th>
<th>PSW Flags</th>
<th>IVDUX</th>
<th>CVZN</th>
<th>Traps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pop Word</td>
<td>popw</td>
<td>w1,w2</td>
<td>16</td>
<td>Register</td>
<td>$1 \ w1 \leftarrow \ (w1) + 4, \ w2 \leftarrow \ (w2) - 4$</td>
<td>. . .</td>
<td>P, R, M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push Word</td>
<td>pushw</td>
<td>w2,w1</td>
<td>14</td>
<td>Register</td>
<td>$1 \ w1 \leftarrow \ (w1) - 4, (w1) \leftarrow \ (w2)$</td>
<td>. . .</td>
<td>P, W, M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restore Registers fn: 17</td>
<td>restdn</td>
<td>b4 28</td>
<td>Macro</td>
<td>2</td>
<td>\text{. . .} \ (r15) \leftarrow 8 \times [7-n], \ r15 \leftarrow (r15) + 8 \times [8-n], \ r15 \leftarrow (r15) + 4 \times [14-n], \ r15 \leftarrow (r15) + 4 \times [15-n]</td>
<td>. . .</td>
<td>P, R, M, S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restore User Registers (privileged)</td>
<td>restur</td>
<td>w1</td>
<td>b6 03</td>
<td>Macro</td>
<td>$2 \ r0 \leftarrow \ (r15) = (w1) + 60$, (w1) \leftarrow \ (w1)$</td>
<td>. . .</td>
<td>P, R, M, S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restore Registers mn: r14</td>
<td>restwn</td>
<td>b4 10</td>
<td>Macro</td>
<td>2</td>
<td>$(r15) \leftarrow (r15) + 4 \times [14-n], \ r15 \leftarrow (r15) + 8 \times [8-n], \ r15 \leftarrow (r15) + 8 \times [8-n]$</td>
<td>. . .</td>
<td>P, W, M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Save Registers fn: 17</td>
<td>savedn</td>
<td>b4 20</td>
<td>Macro</td>
<td>2</td>
<td>$(r15) \leftarrow 8 \times [8-n], (r15) \leftarrow 8 \times [14-n], \ r15 \leftarrow (r15) + (14-n), \ r15 \leftarrow (r15) + 8 \times [8-n], \ r15 \leftarrow (r15) + 8 \times [15-n]$</td>
<td>. . .</td>
<td>P, W, M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Save User Registers (privileged)</td>
<td>saveur</td>
<td>w1</td>
<td>b6 02</td>
<td>Macro</td>
<td>$2 \ (w1) \leftarrow 4 \times (w1) + 64 \times (r15) \leftarrow (r0)$</td>
<td>. . .</td>
<td>P, W, M, S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Save Registers mn: r14</td>
<td>savewn</td>
<td>b4 00</td>
<td>Macro</td>
<td>2</td>
<td>$(r15) \leftarrow 4 \times [15-n], \ r15 \leftarrow (r15) + 4 \times [15-n]$</td>
<td>. . .</td>
<td>P, W, M</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend: PSW Flags Field
- . = Flag not affected by instruction
- * = Flag set according to operation
- 0 = Flag set to 0
- 1 = Flag set to 1

Traps Field
- D = Divide-by-zero
- I = Illegal instruction
- M = Memory fault
- P = Page fault
- R = Read protect fault
- S = Supervisor only (privileged) instruction
- W = Write protect fault

ADVANCE INFORMATION B-4
Table B-1  Functional Instruction Set (Continued)

<table>
<thead>
<tr>
<th>Instruction Name</th>
<th>Syntax</th>
<th>Opcode</th>
<th>Format</th>
<th>Parcels</th>
<th>Operation</th>
<th>PSW Flags</th>
<th>Traps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch Conditional</td>
<td>b*</td>
<td>ha</td>
<td>48,49</td>
<td>1-4</td>
<td>IF cond, PC ← ha</td>
<td>FFFF</td>
<td></td>
</tr>
<tr>
<td>Branch Floating Exception</td>
<td>bf*</td>
<td>ha</td>
<td>4c,4d</td>
<td>1-4</td>
<td>IF cond, PC ← ha</td>
<td>FFFFFFF</td>
<td></td>
</tr>
<tr>
<td>Call Subroutine</td>
<td>call</td>
<td>w2,ha</td>
<td>44,45</td>
<td>1-4</td>
<td>w2 ← (w2) - 4, (w2) ← (PC), PC ← ha</td>
<td>........</td>
<td></td>
</tr>
<tr>
<td>Call Supervisor</td>
<td>calls</td>
<td>bb</td>
<td>12</td>
<td>1</td>
<td>trap 400 + 8 x bb &lt; 7:0&gt;</td>
<td>........</td>
<td></td>
</tr>
<tr>
<td>No Operation</td>
<td>noop</td>
<td>bb</td>
<td>00</td>
<td>1</td>
<td>none</td>
<td>........</td>
<td></td>
</tr>
<tr>
<td>Return From Subroutine</td>
<td>ret</td>
<td>w2</td>
<td>13</td>
<td>1</td>
<td>PC ← ((w2)), w2 ← (w2) + 4</td>
<td>........</td>
<td></td>
</tr>
<tr>
<td>Return From Interrupt (privileged)</td>
<td>reti</td>
<td>w1</td>
<td>b6 04</td>
<td>2</td>
<td>Restore SSW, PSW and PC</td>
<td>........</td>
<td></td>
</tr>
<tr>
<td>Trap on Floating Unordered</td>
<td>trapfn</td>
<td>b4 3e</td>
<td>Macro</td>
<td>2</td>
<td>IF PSW&lt;ZN&gt; indicated unordered, illegal</td>
<td>........</td>
<td>I</td>
</tr>
<tr>
<td>Wait for Interrupt (privileged)</td>
<td>wait</td>
<td>b6 05</td>
<td>Macro</td>
<td>2</td>
<td>Wait for interrupt</td>
<td>........</td>
<td>S</td>
</tr>
</tbody>
</table>

Legend:  PSW Flags Field
. = Flag not affected by instruction
* = Flag set according to operation
0 = Flag set to 0
1 = Flag set to 1

Traps Field
D = Divide-by-zero
I = Illegal instruction
M = Memory fault
P = Page fault
R = Read protect fault
S = Supervisor only (privileged) instruction
W = Write protect fault
### Table B-2 Instruction Opcode/Mnemonic Summary

<table>
<thead>
<tr>
<th>LSB</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>
<th>F</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>
<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>
<td></td>
<td></td>
</tr>
<tr>
<td>MSB</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>
<td></td>
</tr>
<tr>
<td>0</td>
<td>noop</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>
<tr>
<td>1</td>
<td>movwp movpw</td>
<td>calls ret pushw</td>
<td>popw</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>2</td>
<td>adds</td>
<td>sube</td>
<td>addd</td>
<td>subd</td>
<td>movs</td>
<td>cmps</td>
<td>movd</td>
<td>cmpd</td>
<td>mulis</td>
<td>divs</td>
<td>muld</td>
<td>divd</td>
<td>movew</td>
<td>movws</td>
<td>movdl</td>
<td>movid</td>
</tr>
<tr>
<td>3</td>
<td>shaw</td>
<td>shai</td>
<td>shiw</td>
<td>shil</td>
<td>rotw</td>
<td>rotl</td>
<td></td>
<td></td>
<td>shai</td>
<td>shali</td>
<td>shii</td>
<td>shilli</td>
<td>roti</td>
<td>rotli</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>call</td>
<td>b*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>see Table 2-7</td>
<td></td>
<td></td>
<td>b**</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>loadw</td>
<td>loada</td>
<td>loads</td>
<td>loadd</td>
<td>loadb</td>
<td>loadbu</td>
<td>loadh</td>
<td>loadhu</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>storw</td>
<td>tst</td>
<td>stors</td>
<td>stord</td>
<td>storb</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>8</td>
<td>addw</td>
<td>addq</td>
<td>addi</td>
<td>movw</td>
<td>loadq</td>
<td>loadi</td>
<td>andw</td>
<td>andl</td>
<td>ori</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>addwc</td>
<td>subwc</td>
<td>negw</td>
<td></td>
<td>mulw</td>
<td>mulwx</td>
<td>mulwu</td>
<td>mulwux</td>
<td>divw</td>
<td>modw</td>
<td>divwu</td>
<td>modwu</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>subw</td>
<td>subq</td>
<td>subi</td>
<td>cmpw</td>
<td>cmpq</td>
<td>cmpi</td>
<td>xorw</td>
<td>xori</td>
<td>notw</td>
<td>notq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>macros see Table B-3</td>
<td></td>
<td></td>
<td>privileged macros see Table B-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</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>
<td></td>
</tr>
<tr>
<td>D</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>
<td></td>
</tr>
<tr>
<td>E</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>
<td></td>
</tr>
<tr>
<td>F</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>
<td></td>
</tr>
</tbody>
</table>
### Table B-3  Macro Instruction Code Field (opcode B4)

<table>
<thead>
<tr>
<th>LSB</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>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>savew0</td>
<td>savew1</td>
<td>savew2</td>
<td>savew3</td>
<td>savew4</td>
<td>savew5</td>
<td>savew6</td>
<td>savew7</td>
<td>savew8</td>
<td>savew9</td>
<td>savew10</td>
<td>savew11</td>
<td>savew12</td>
<td>movc</td>
<td>initc</td>
<td>cmpc</td>
</tr>
<tr>
<td>1</td>
<td>restw0</td>
<td>restw1</td>
<td>restw2</td>
<td>restw3</td>
<td>restw4</td>
<td>restw5</td>
<td>restw6</td>
<td>restw7</td>
<td>restw8</td>
<td>restw9</td>
<td>restw10</td>
<td>restw11</td>
<td>restw12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>saved0</td>
<td>saved1</td>
<td>saved2</td>
<td>saved3</td>
<td>saved4</td>
<td>saved5</td>
<td>saved6</td>
<td>saved7</td>
<td>restd0</td>
<td>restd1</td>
<td>restd2</td>
<td>restd3</td>
<td>restd4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>cnvsw</td>
<td>cnvrsw</td>
<td>cnvtsw</td>
<td>cnvws</td>
<td>cnvrdw</td>
<td>cnvrdw</td>
<td>cnvwd</td>
<td>cnvwsd</td>
<td>cnvds</td>
<td>negs</td>
<td>negds</td>
<td>scalbs</td>
<td>scalbd</td>
<td>trapfn</td>
<td>loadfs</td>
<td></td>
</tr>
</tbody>
</table>

### Table B-4  Privileged Macro Instruction Code Field (opcode B6)

<table>
<thead>
<tr>
<th>LSB</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>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>movus</td>
<td>movsu</td>
<td>savexr</td>
<td>restur</td>
<td>reti</td>
<td>wait</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</td>
<td>movus</td>
<td>movsu</td>
<td>savexr</td>
<td>restur</td>
<td>reti</td>
<td>wait</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>