<table>
<thead>
<tr>
<th>OP CODE</th>
<th>MNEMONIC</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>ACB8</td>
<td>LOAD BYTE</td>
</tr>
<tr>
<td>0010</td>
<td>ACB9</td>
<td>LOAD HALFWORD</td>
</tr>
<tr>
<td>0020</td>
<td>ACBA</td>
<td>LOAD DOUBLEWORD</td>
</tr>
<tr>
<td>0030</td>
<td>ACBB</td>
<td>LOAD MASKED HALFWORD</td>
</tr>
<tr>
<td>0040</td>
<td>ACBC</td>
<td>LOAD MASKED DOUBLEWORD</td>
</tr>
<tr>
<td>0050</td>
<td>ACBD</td>
<td>LOAD MASKED EFFECTIVE ADDRESS</td>
</tr>
<tr>
<td>0060</td>
<td>ACBE</td>
<td>LOAD MASKED ADDRESS</td>
</tr>
<tr>
<td>0070</td>
<td>ACBF</td>
<td>LOAD PROCESSOR ADDRESS</td>
</tr>
<tr>
<td>0080</td>
<td>ACBG</td>
<td>LOAD PROCESSOR ADDRESS MASKED</td>
</tr>
<tr>
<td>0090</td>
<td>ACBH</td>
<td>LOAD FILE</td>
</tr>
<tr>
<td>00A0</td>
<td>ACBI</td>
<td>LOAD BYTE</td>
</tr>
<tr>
<td>00B0</td>
<td>ACBJ</td>
<td>LOAD HALFWORD</td>
</tr>
<tr>
<td>00C0</td>
<td>ACBK</td>
<td>LOAD DOUBLEWORD</td>
</tr>
<tr>
<td>00D0</td>
<td>ACBL</td>
<td>LOAD EFFECTIVE ADDRESS</td>
</tr>
<tr>
<td>00E0</td>
<td>ACBM</td>
<td>LOAD EFFECTIVE ADDRESS MASKED</td>
</tr>
<tr>
<td>00F0</td>
<td>ACBN</td>
<td>LOAD PROCESSOR ADDRESS</td>
</tr>
<tr>
<td>0100</td>
<td>ACBO</td>
<td>LOAD PROCESSOR ADDRESS MASKED</td>
</tr>
<tr>
<td>0110</td>
<td>ACBP</td>
<td>STORE BYTE</td>
</tr>
<tr>
<td>0120</td>
<td>ACBQ</td>
<td>STORE HALFWORD</td>
</tr>
<tr>
<td>0130</td>
<td>ACBR</td>
<td>STORE DOUBLEWORD</td>
</tr>
<tr>
<td>0140</td>
<td>ACBS</td>
<td>STORE MASKED BYTE</td>
</tr>
<tr>
<td>0150</td>
<td>ACBT</td>
<td>STORE MASKED HALFWORD</td>
</tr>
<tr>
<td>0160</td>
<td>ACBU</td>
<td>STORE MASKED DOUBLEWORD</td>
</tr>
<tr>
<td>0170</td>
<td>ACBV</td>
<td>STORE MASKED EFFECTIVE ADDRESS</td>
</tr>
<tr>
<td>0180</td>
<td>ACBW</td>
<td>STORE MASKED ADDRESS</td>
</tr>
<tr>
<td>0190</td>
<td>ACBX</td>
<td>STORE MASKED ADDRESS MASKED</td>
</tr>
<tr>
<td>01A0</td>
<td>ACBY</td>
<td>STORE PROCESSOR ADDRESS</td>
</tr>
<tr>
<td>01B0</td>
<td>ACBZ</td>
<td>STORE PROCESSOR ADDRESS MASKED</td>
</tr>
<tr>
<td>01C0</td>
<td>ACBP</td>
<td>TRANSFER TO REGISTER</td>
</tr>
<tr>
<td>01D0</td>
<td>ACBQ</td>
<td>TRANSFER TO REGISTER</td>
</tr>
<tr>
<td>01E0</td>
<td>ACBR</td>
<td>TRANSFER TO REGISTER</td>
</tr>
<tr>
<td>01F0</td>
<td>ACBS</td>
<td>TRANSFER TO REGISTER</td>
</tr>
<tr>
<td>0200</td>
<td>ACBT</td>
<td>TRANSFER TO REGISTER</td>
</tr>
<tr>
<td>0210</td>
<td>ACBU</td>
<td>TRANSFER TO REGISTER</td>
</tr>
<tr>
<td>0220</td>
<td>ACBV</td>
<td>TRANSFER TO REGISTER</td>
</tr>
<tr>
<td>0230</td>
<td>ACBW</td>
<td>TRANSFER TO REGISTER</td>
</tr>
<tr>
<td>0240</td>
<td>ACBX</td>
<td>TRANSFER TO REGISTER</td>
</tr>
<tr>
<td>0250</td>
<td>ACBY</td>
<td>TRANSFER TO REGISTER</td>
</tr>
<tr>
<td>0260</td>
<td>ACBZ</td>
<td>TRANSFER TO REGISTER</td>
</tr>
<tr>
<td>0270</td>
<td>ACBP</td>
<td>TRANSFER TO MEMORY</td>
</tr>
<tr>
<td>0280</td>
<td>ACBQ</td>
<td>TRANSFER TO MEMORY</td>
</tr>
<tr>
<td>0290</td>
<td>ACBR</td>
<td>TRANSFER TO MEMORY</td>
</tr>
<tr>
<td>02A0</td>
<td>ACBS</td>
<td>TRANSFER TO MEMORY</td>
</tr>
<tr>
<td>02B0</td>
<td>ACBT</td>
<td>TRANSFER TO MEMORY</td>
</tr>
<tr>
<td>02C0</td>
<td>ACBU</td>
<td>TRANSFER TO MEMORY</td>
</tr>
<tr>
<td>02D0</td>
<td>ACBV</td>
<td>TRANSFER TO MEMORY</td>
</tr>
<tr>
<td>02E0</td>
<td>ACBW</td>
<td>TRANSFER TO MEMORY</td>
</tr>
<tr>
<td>02F0</td>
<td>ACBX</td>
<td>TRANSFER TO MEMORY</td>
</tr>
<tr>
<td>0300</td>
<td>ACBY</td>
<td>TRANSFER TO MEMORY</td>
</tr>
<tr>
<td>0310</td>
<td>ACBZ</td>
<td>TRANSFER TO MEMORY</td>
</tr>
<tr>
<td>0320</td>
<td>ACBP</td>
<td>SHIFT INSTRUCTIONS</td>
</tr>
<tr>
<td>0330</td>
<td>ACBQ</td>
<td>SHIFT INSTRUCTIONS</td>
</tr>
<tr>
<td>0340</td>
<td>ACBR</td>
<td>SHIFT INSTRUCTIONS</td>
</tr>
<tr>
<td>0350</td>
<td>ACBS</td>
<td>SHIFT INSTRUCTIONS</td>
</tr>
<tr>
<td>0360</td>
<td>ACBT</td>
<td>SHIFT INSTRUCTIONS</td>
</tr>
<tr>
<td>0370</td>
<td>ACBU</td>
<td>SHIFT INSTRUCTIONS</td>
</tr>
<tr>
<td>0380</td>
<td>ACBV</td>
<td>SHIFT INSTRUCTIONS</td>
</tr>
<tr>
<td>0390</td>
<td>ACBW</td>
<td>SHIFT INSTRUCTIONS</td>
</tr>
<tr>
<td>03A0</td>
<td>ACBX</td>
<td>SHIFT INSTRUCTIONS</td>
</tr>
<tr>
<td>03B0</td>
<td>ACBY</td>
<td>SHIFT INSTRUCTIONS</td>
</tr>
<tr>
<td>03C0</td>
<td>ACBZ</td>
<td>SHIFT INSTRUCTIONS</td>
</tr>
<tr>
<td>03D0</td>
<td>ACBP</td>
<td>BRANCH INSTRUCTIONS</td>
</tr>
<tr>
<td>03E0</td>
<td>ACBQ</td>
<td>BRANCH INSTRUCTIONS</td>
</tr>
<tr>
<td>03F0</td>
<td>ACBR</td>
<td>BRANCH INSTRUCTIONS</td>
</tr>
<tr>
<td>0400</td>
<td>ACBS</td>
<td>BRANCH INSTRUCTIONS</td>
</tr>
<tr>
<td>0410</td>
<td>ACBT</td>
<td>BRANCH INSTRUCTIONS</td>
</tr>
<tr>
<td>0420</td>
<td>ACBU</td>
<td>BRANCH INSTRUCTIONS</td>
</tr>
<tr>
<td>0430</td>
<td>ACBV</td>
<td>BRANCH INSTRUCTIONS</td>
</tr>
<tr>
<td>0440</td>
<td>ACBW</td>
<td>BRANCH INSTRUCTIONS</td>
</tr>
<tr>
<td>0450</td>
<td>ACBX</td>
<td>BRANCH INSTRUCTIONS</td>
</tr>
<tr>
<td>0460</td>
<td>ACBY</td>
<td>BRANCH INSTRUCTIONS</td>
</tr>
<tr>
<td>0470</td>
<td>ACBZ</td>
<td>BRANCH INSTRUCTIONS</td>
</tr>
<tr>
<td>0480</td>
<td>ACBP</td>
<td>LOGICAL INSTRUCTIONS</td>
</tr>
<tr>
<td>0490</td>
<td>ACBQ</td>
<td>LOGICAL INSTRUCTIONS</td>
</tr>
<tr>
<td>04A0</td>
<td>ACBR</td>
<td>LOGICAL INSTRUCTIONS</td>
</tr>
<tr>
<td>04B0</td>
<td>ACBS</td>
<td>LOGICAL INSTRUCTIONS</td>
</tr>
<tr>
<td>04C0</td>
<td>ACBT</td>
<td>LOGICAL INSTRUCTIONS</td>
</tr>
<tr>
<td>04D0</td>
<td>ACBU</td>
<td>LOGICAL INSTRUCTIONS</td>
</tr>
<tr>
<td>04E0</td>
<td>ACBV</td>
<td>LOGICAL INSTRUCTIONS</td>
</tr>
<tr>
<td>04F0</td>
<td>ACBW</td>
<td>LOGICAL INSTRUCTIONS</td>
</tr>
<tr>
<td>0500</td>
<td>ACBX</td>
<td>LOGICAL INSTRUCTIONS</td>
</tr>
<tr>
<td>0510</td>
<td>ACBY</td>
<td>LOGICAL INSTRUCTIONS</td>
</tr>
<tr>
<td>0520</td>
<td>ACBZ</td>
<td>LOGICAL INSTRUCTIONS</td>
</tr>
<tr>
<td>0530</td>
<td>ACBP</td>
<td>INTERRUPT INSTRUCTIONS</td>
</tr>
<tr>
<td>0540</td>
<td>ACBQ</td>
<td>INTERRUPT INSTRUCTIONS</td>
</tr>
<tr>
<td>0550</td>
<td>ACBR</td>
<td>INTERRUPT INSTRUCTIONS</td>
</tr>
<tr>
<td>0560</td>
<td>ACBS</td>
<td>INTERRUPT INSTRUCTIONS</td>
</tr>
<tr>
<td>0570</td>
<td>ACBT</td>
<td>INTERRUPT INSTRUCTIONS</td>
</tr>
<tr>
<td>0580</td>
<td>ACBU</td>
<td>INTERRUPT INSTRUCTIONS</td>
</tr>
<tr>
<td>0590</td>
<td>ACBV</td>
<td>INTERRUPT INSTRUCTIONS</td>
</tr>
<tr>
<td>05A0</td>
<td>ACBW</td>
<td>INTERRUPT INSTRUCTIONS</td>
</tr>
<tr>
<td>05B0</td>
<td>ACBX</td>
<td>INTERRUPT INSTRUCTIONS</td>
</tr>
<tr>
<td>05C0</td>
<td>ACBY</td>
<td>INTERRUPT INSTRUCTIONS</td>
</tr>
<tr>
<td>05D0</td>
<td>ACBZ</td>
<td>INTERRUPT INSTRUCTIONS</td>
</tr>
<tr>
<td>05E0</td>
<td>ACBP</td>
<td>INPUT/OUTPUT INSTRUCTIONS</td>
</tr>
<tr>
<td>05F0</td>
<td>ACBQ</td>
<td>INPUT/OUTPUT INSTRUCTIONS</td>
</tr>
<tr>
<td>0600</td>
<td>ACBR</td>
<td>INPUT/OUTPUT INSTRUCTIONS</td>
</tr>
<tr>
<td>0610</td>
<td>ACBS</td>
<td>INPUT/OUTPUT INSTRUCTIONS</td>
</tr>
<tr>
<td>0620</td>
<td>ACBT</td>
<td>INPUT/OUTPUT INSTRUCTIONS</td>
</tr>
<tr>
<td>0630</td>
<td>ACBU</td>
<td>INPUT/OUTPUT INSTRUCTIONS</td>
</tr>
<tr>
<td>0640</td>
<td>ACBV</td>
<td>INPUT/OUTPUT INSTRUCTIONS</td>
</tr>
<tr>
<td>0650</td>
<td>ACBW</td>
<td>INPUT/OUTPUT INSTRUCTIONS</td>
</tr>
<tr>
<td>0660</td>
<td>ACBX</td>
<td>INPUT/OUTPUT INSTRUCTIONS</td>
</tr>
<tr>
<td>0670</td>
<td>ACBY</td>
<td>INPUT/OUTPUT INSTRUCTIONS</td>
</tr>
<tr>
<td>0680</td>
<td>ACBZ</td>
<td>INPUT/OUTPUT INSTRUCTIONS</td>
</tr>
</tbody>
</table>

** 注：仅列出部分指令。
REVISION INSTRUCTIONS AND MANUAL HISTORY

EQUIPMENT: 32/70 Series Computer

PURPOSE: This reissue upgrades the manual reflecting the requirements of the Model 2005 Internal Processing Unit (IPU)

REVISION INSTRUCTIONS: Delete and add pages as shown on the following table.

<table>
<thead>
<tr>
<th>DELETE</th>
<th>ADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

NOTE: Revised pages are marked with the Rev. No. in the upper unbound corner. Revised areas are marked with a vertical bar.

MANUAL HISTORY

<table>
<thead>
<tr>
<th>REV. TYPE</th>
<th>REV. NO.</th>
<th>DATE ISSUED</th>
<th>CONTROL DOC. NO.</th>
<th>ECL</th>
<th>REV. TYPE</th>
<th>REV. NO.</th>
<th>DATE ISSUED</th>
<th>CONTROL DOC. NO.</th>
<th>ECL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Ptg</td>
<td>-</td>
<td>1/79</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>-</td>
<td>1/80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

I = INTERIM REVISION
F = FORMAL REVISION
R = REISSUE
A = ADDENDUM
LIST OF EFFECTIVE PAGES

The total number of pages is in this manual is 398 consisting of the following:

<table>
<thead>
<tr>
<th>Page</th>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Original</td>
</tr>
<tr>
<td>Instructions</td>
<td>Original</td>
</tr>
<tr>
<td>iii through xiv</td>
<td>Original</td>
</tr>
<tr>
<td>Frontispiece</td>
<td>Original</td>
</tr>
<tr>
<td>1-1 through 1-18</td>
<td>Original</td>
</tr>
<tr>
<td>2-1 through 2-32</td>
<td>Original</td>
</tr>
<tr>
<td>3-1 through 3-12</td>
<td>Original</td>
</tr>
<tr>
<td>4-1 through 4-22</td>
<td>Original</td>
</tr>
<tr>
<td>5-1 through 5-22</td>
<td>Original</td>
</tr>
<tr>
<td>6-1 through 6-228</td>
<td>Original</td>
</tr>
<tr>
<td>7-1 through 7-18</td>
<td>Original</td>
</tr>
<tr>
<td>8-1 through 8-6</td>
<td>Original</td>
</tr>
<tr>
<td>A-1 through A-6</td>
<td>Original</td>
</tr>
<tr>
<td>B-1 through B-8</td>
<td>Original</td>
</tr>
<tr>
<td>C-1 through C-2</td>
<td>Original</td>
</tr>
<tr>
<td>D-1 through D-2</td>
<td>Original</td>
</tr>
<tr>
<td>E-1 through E-2</td>
<td>Original</td>
</tr>
<tr>
<td>F-1 through F-2</td>
<td>Original</td>
</tr>
<tr>
<td>G-1 through G-2</td>
<td>Original</td>
</tr>
<tr>
<td>OP Codes</td>
<td>Original</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS

SECTION I  GENERAL DESCRIPTION

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1-1</td>
</tr>
<tr>
<td>System Overview</td>
<td>1-1</td>
</tr>
<tr>
<td>General Characteristics</td>
<td>1-1</td>
</tr>
<tr>
<td>Standard and Optional Features</td>
<td>1-4</td>
</tr>
<tr>
<td>General Purpose Features</td>
<td>1-5</td>
</tr>
<tr>
<td>Real-Time Features</td>
<td>1-6</td>
</tr>
<tr>
<td>Multiusage Features</td>
<td>1-7</td>
</tr>
<tr>
<td>Multiprocessing Features</td>
<td>1-8</td>
</tr>
<tr>
<td>Functional Description</td>
<td>1-8</td>
</tr>
<tr>
<td>Major System Elements</td>
<td>1-8</td>
</tr>
<tr>
<td>SelBUS</td>
<td>1-11</td>
</tr>
<tr>
<td>Central Processor Unit</td>
<td>1-11</td>
</tr>
<tr>
<td>General Purpose Registers</td>
<td>1-11</td>
</tr>
<tr>
<td>Floating-Point Arithmetic Processor</td>
<td>1-11</td>
</tr>
<tr>
<td>CPU Modes</td>
<td>1-11</td>
</tr>
<tr>
<td>Control Modes</td>
<td>1-13</td>
</tr>
<tr>
<td>Addressing Modes</td>
<td>1-13</td>
</tr>
<tr>
<td>Address Submodes</td>
<td>1-13</td>
</tr>
<tr>
<td>Hardware Memory Management</td>
<td>1-14</td>
</tr>
<tr>
<td>Memory Map</td>
<td>1-14</td>
</tr>
<tr>
<td>Write Protection</td>
<td>1-14</td>
</tr>
<tr>
<td>Optional Writable Control Storage</td>
<td>1-15</td>
</tr>
<tr>
<td>Optional High-Speed Floating-Point Unit</td>
<td>1-15</td>
</tr>
<tr>
<td>Real-Time Option Module</td>
<td>1-15</td>
</tr>
<tr>
<td>Interval Timer</td>
<td>1-15</td>
</tr>
<tr>
<td>Main Memory</td>
<td>1-15</td>
</tr>
<tr>
<td>Memory Unit</td>
<td>1-15</td>
</tr>
<tr>
<td>Memory Module</td>
<td>1-16</td>
</tr>
<tr>
<td>Memory Interleaving</td>
<td>1-16</td>
</tr>
<tr>
<td>Memory Unit Address Identity</td>
<td>1-16</td>
</tr>
<tr>
<td>Memory Bus Controllers</td>
<td>1-16</td>
</tr>
<tr>
<td>Memory Lock and Unlock</td>
<td>1-17</td>
</tr>
<tr>
<td>Private Memory</td>
<td>1-17</td>
</tr>
<tr>
<td>Input/Output Subsystem</td>
<td>1-17</td>
</tr>
<tr>
<td>IOM</td>
<td>1-17</td>
</tr>
<tr>
<td>Regional Processing Unit</td>
<td>1-17</td>
</tr>
<tr>
<td>General Purpose Multiplexer Controller</td>
<td>1-18</td>
</tr>
</tbody>
</table>

SECTION II  CENTRAL PROCESSOR

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>2-1</td>
</tr>
<tr>
<td>Instruction Repertoire</td>
<td>2-1</td>
</tr>
<tr>
<td>General Purpose Registers</td>
<td>2-2</td>
</tr>
<tr>
<td>CPU Control Modes</td>
<td>2-2</td>
</tr>
</tbody>
</table>
### TABLE OF CONTENTS (Cont'd)

#### SECTION II CENTRAL PROCESSOR (Cont'd)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Status Word</td>
<td>2-2</td>
</tr>
<tr>
<td>Program Status Doubleword</td>
<td>2-2</td>
</tr>
<tr>
<td>Condition Codes</td>
<td>2-2</td>
</tr>
<tr>
<td>Privileged and Unprivileged Operation</td>
<td>2-2</td>
</tr>
<tr>
<td>CPU Addressing Modes</td>
<td>2-6</td>
</tr>
<tr>
<td>512 KB Mode</td>
<td>2-7</td>
</tr>
<tr>
<td>512 KB Extended Mode</td>
<td>2-7</td>
</tr>
<tr>
<td>512 KB Mapped Mode</td>
<td>2-7</td>
</tr>
<tr>
<td>Mapped Extended Mode</td>
<td>2-7</td>
</tr>
<tr>
<td>CPU Major Elements</td>
<td>2-7</td>
</tr>
<tr>
<td>CPU Data Structure</td>
<td>2-7</td>
</tr>
<tr>
<td>CPU Microprogrammable Processor</td>
<td>2-9</td>
</tr>
<tr>
<td>Implementation Logic</td>
<td>2-9</td>
</tr>
<tr>
<td>SelBUS Interface</td>
<td>2-9</td>
</tr>
<tr>
<td>Optional Writable Control Storage</td>
<td>2-9</td>
</tr>
<tr>
<td>Optional High-Speed Floating-Point Unit</td>
<td>2-12</td>
</tr>
<tr>
<td>Internal Processing Unit</td>
<td>2-12</td>
</tr>
<tr>
<td>Introduction</td>
<td>2-12</td>
</tr>
<tr>
<td>General</td>
<td>2-14</td>
</tr>
<tr>
<td>General Characteristics</td>
<td></td>
</tr>
<tr>
<td>Instruction Repertoire</td>
<td>2-15</td>
</tr>
<tr>
<td>General Purpose Registers</td>
<td>2-16</td>
</tr>
<tr>
<td>IPU Control Mode</td>
<td>2-16</td>
</tr>
<tr>
<td>Program Status Doubleword</td>
<td>2-16</td>
</tr>
<tr>
<td>Condition Codes</td>
<td>2-16</td>
</tr>
<tr>
<td>Privileged and Unprivileged Operation</td>
<td>2-16</td>
</tr>
<tr>
<td>IPU Addressing Modes</td>
<td>2-19</td>
</tr>
<tr>
<td>512-KB Mode</td>
<td>2-19</td>
</tr>
<tr>
<td>512-KB Extended Mode</td>
<td>2-19</td>
</tr>
<tr>
<td>512-KB Mapped Mode</td>
<td>2-19</td>
</tr>
<tr>
<td>Mapped Extended Mode</td>
<td>2-19</td>
</tr>
<tr>
<td>Functional Description</td>
<td>2-19</td>
</tr>
<tr>
<td>Major System Elements</td>
<td>2-19</td>
</tr>
<tr>
<td>Central Processing Unit</td>
<td>2-21</td>
</tr>
<tr>
<td>IPU Major Elements</td>
<td>2-21</td>
</tr>
<tr>
<td>IPU Data Structure</td>
<td>2-21</td>
</tr>
<tr>
<td>IPU Microprogrammable Processor</td>
<td>2-22</td>
</tr>
<tr>
<td>Implementation Logic</td>
<td>2-22</td>
</tr>
<tr>
<td>SelBUS Interface</td>
<td>2-22</td>
</tr>
<tr>
<td>Optional High-Speed Floating-Point Unit</td>
<td>2-22</td>
</tr>
<tr>
<td>Optional Scientific Accelerator</td>
<td>2-24</td>
</tr>
<tr>
<td>Optional Writable Control Storage</td>
<td>2-24</td>
</tr>
<tr>
<td>Traps</td>
<td>2-24</td>
</tr>
<tr>
<td>New Traps</td>
<td>2-24</td>
</tr>
<tr>
<td>Operating Mode</td>
<td>2-24</td>
</tr>
<tr>
<td>Trap Context Switching</td>
<td>2-24</td>
</tr>
<tr>
<td>Trap Format</td>
<td>2-24</td>
</tr>
<tr>
<td>IPU Status Word</td>
<td>2-27</td>
</tr>
<tr>
<td>CPU/IPU Interface Operation</td>
<td>2-27</td>
</tr>
<tr>
<td>Start IPU Trap (Vector Address 2E4)</td>
<td>2-27</td>
</tr>
<tr>
<td>Restart IPU</td>
<td>2-28</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS (Cont'd)

SECTION II CENTRAL PROCESSOR (Cont'd)

<table>
<thead>
<tr>
<th>Traps and Interrupts</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPU Error Condition Trap (Vector Address 2EC)</td>
<td>2-28</td>
</tr>
<tr>
<td>IPU Call Monitor Trap (Vector Address 2FD)</td>
<td>2-31</td>
</tr>
<tr>
<td>IPU Supervisor Call Trap (Vector Address 2EF)</td>
<td>2-31</td>
</tr>
<tr>
<td>Stop IPU Trap Vector Address 2F4</td>
<td>2-31</td>
</tr>
<tr>
<td>CPU (End IPU Processing) Trap (Vector Address 2EO)</td>
<td>2-31</td>
</tr>
<tr>
<td>Memory Management</td>
<td>2-31</td>
</tr>
<tr>
<td>Input/Output System</td>
<td>2-32</td>
</tr>
<tr>
<td>Scratchpad Memory</td>
<td>2-32</td>
</tr>
<tr>
<td>Initialization</td>
<td>2-32</td>
</tr>
<tr>
<td>Introduction</td>
<td>2-32</td>
</tr>
<tr>
<td>Initial Program Load</td>
<td>2-32</td>
</tr>
<tr>
<td>Power Fail-Safe Feature</td>
<td>2-32</td>
</tr>
<tr>
<td>Machine Check Trap</td>
<td>3-10</td>
</tr>
<tr>
<td>System Check Trap</td>
<td>3-10</td>
</tr>
<tr>
<td>Block Mode Time-Out Trap</td>
<td>3-11</td>
</tr>
<tr>
<td>PSD Trap Halt Implementation</td>
<td>3-11</td>
</tr>
</tbody>
</table>
## TABLE OF CONTENTS (Cont'd)

### SECTION IV MEMORY MANAGEMENT

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>4-1</td>
</tr>
<tr>
<td>Overview</td>
<td>4-1</td>
</tr>
<tr>
<td>MOS and Core Memory</td>
<td>4-1</td>
</tr>
<tr>
<td>600/900 ns Core Memory Modules</td>
<td>4-2</td>
</tr>
<tr>
<td>Mixed Memory Rules</td>
<td>4-2</td>
</tr>
<tr>
<td>Memory Reference Instructions</td>
<td>4-3</td>
</tr>
<tr>
<td>F- and C-Bits</td>
<td>4-4</td>
</tr>
<tr>
<td>Direct Addressing</td>
<td>4-4</td>
</tr>
<tr>
<td>Indirect and Indexed Addressing</td>
<td>4-5</td>
</tr>
<tr>
<td>Indexed Addressing</td>
<td>4-5</td>
</tr>
<tr>
<td>Indirect Addressing</td>
<td>4-6</td>
</tr>
<tr>
<td>Words, Halfwords, and Bytes</td>
<td>4-6</td>
</tr>
<tr>
<td>Word and Doubleword Operands</td>
<td>4-6</td>
</tr>
<tr>
<td>Hardware Memory Management</td>
<td>4-8</td>
</tr>
<tr>
<td>Addressing Modes</td>
<td>4-8</td>
</tr>
<tr>
<td>512 KB Mode</td>
<td>4-8</td>
</tr>
<tr>
<td>512 KB Extended Mode</td>
<td>4-8</td>
</tr>
<tr>
<td>512 KB Mapped Mode</td>
<td>4-9</td>
</tr>
<tr>
<td>Mapped Extended Mode</td>
<td>4-9</td>
</tr>
<tr>
<td>Memory Mapping</td>
<td>4-9</td>
</tr>
<tr>
<td>Memory Protection</td>
<td>4-12</td>
</tr>
<tr>
<td>Program Status Doubleword</td>
<td>4-12</td>
</tr>
<tr>
<td>PSD Fields</td>
<td>4-12</td>
</tr>
<tr>
<td>Condition Codes</td>
<td>4-14</td>
</tr>
<tr>
<td>MAP Description</td>
<td>4-15</td>
</tr>
<tr>
<td>Master Process List</td>
<td>4-15</td>
</tr>
<tr>
<td>Address Generation</td>
<td>4-17</td>
</tr>
</tbody>
</table>

### SECTION V INPUT/OUTPUT SYSTEM

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>5-1</td>
</tr>
<tr>
<td>Definitions</td>
<td>5-1</td>
</tr>
<tr>
<td>I/O Processor Classifications</td>
<td>5-4</td>
</tr>
<tr>
<td>Operation With Class 0, 1, 2, and E I/O Processors</td>
<td>5-4</td>
</tr>
<tr>
<td>Command Device Instruction</td>
<td>5-5</td>
</tr>
<tr>
<td>Transfer Control Word</td>
<td>5-5</td>
</tr>
<tr>
<td>Test Device Instruction</td>
<td>5-10</td>
</tr>
<tr>
<td>Input/Output Processor</td>
<td>5-10</td>
</tr>
<tr>
<td>SelBUS Interface</td>
<td>5-10</td>
</tr>
<tr>
<td>Transfer Responses</td>
<td>5-11</td>
</tr>
<tr>
<td>IOM Data Structure</td>
<td>5-11</td>
</tr>
<tr>
<td>Arithmetic Logic Unit</td>
<td>5-11</td>
</tr>
<tr>
<td>Data Structure Control</td>
<td>5-11</td>
</tr>
<tr>
<td>Test Structure</td>
<td>5-11</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS (Cont'd)

SECTION V  INPUT/OUTPUT SYSTEM (Cont'd)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intermits</td>
<td>5-11</td>
</tr>
<tr>
<td>Class F I/O Operation</td>
<td>5-11</td>
</tr>
<tr>
<td>Class F I/O Processor</td>
<td>5-13</td>
</tr>
<tr>
<td>Memory Addressing Method</td>
<td>5-13</td>
</tr>
<tr>
<td>PSD Mode I/O Instructions</td>
<td>5-16</td>
</tr>
<tr>
<td>Start I/O</td>
<td>5-16</td>
</tr>
<tr>
<td>Test I/O</td>
<td>5-16</td>
</tr>
<tr>
<td>Halt I/O</td>
<td>5-16</td>
</tr>
<tr>
<td>Enable Channel WCS Load</td>
<td>5-16</td>
</tr>
<tr>
<td>Write Channel WCS</td>
<td>5-16</td>
</tr>
<tr>
<td>Enable Channel Interrupt</td>
<td>5-17</td>
</tr>
<tr>
<td>Disable Channel Interrupt</td>
<td>5-17</td>
</tr>
<tr>
<td>Activate Channel Interrupt</td>
<td>5-17</td>
</tr>
<tr>
<td>Deactivate Channel Interrupt</td>
<td>5-17</td>
</tr>
<tr>
<td>Reset Channel Interrupt</td>
<td>5-17</td>
</tr>
<tr>
<td>Stop I/O</td>
<td>5-17</td>
</tr>
<tr>
<td>Reset Controller</td>
<td>5-17</td>
</tr>
<tr>
<td>Grab Controller</td>
<td>5-17</td>
</tr>
<tr>
<td>Input/Output Command List Address</td>
<td>5-17</td>
</tr>
<tr>
<td>Input/Output Command Doubleword</td>
<td>5-17</td>
</tr>
<tr>
<td>Input/Output Commands</td>
<td>5-18</td>
</tr>
<tr>
<td>Write</td>
<td>5-18</td>
</tr>
<tr>
<td>Read</td>
<td>5-18</td>
</tr>
<tr>
<td>Read Backword</td>
<td>5-18</td>
</tr>
<tr>
<td>Control</td>
<td>5-18</td>
</tr>
<tr>
<td>Sense</td>
<td>5-18</td>
</tr>
<tr>
<td>Transfer In Channel</td>
<td>5-18</td>
</tr>
<tr>
<td>Channel Control</td>
<td>5-18</td>
</tr>
<tr>
<td>Input/Output Termination</td>
<td>5-18</td>
</tr>
<tr>
<td>Input/Output Status Words</td>
<td>5-20</td>
</tr>
<tr>
<td>Input/Output Interrupts</td>
<td>5-20</td>
</tr>
</tbody>
</table>

SECTION VI  INSTRUCTION REPertoire

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>6-1</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>6-1</td>
</tr>
<tr>
<td>Instruction Name</td>
<td>6-1</td>
</tr>
<tr>
<td>Operation Code</td>
<td>6-1</td>
</tr>
<tr>
<td>Format</td>
<td>6-1</td>
</tr>
<tr>
<td>Definition</td>
<td>6-1</td>
</tr>
<tr>
<td>Summary Expression</td>
<td>6-1</td>
</tr>
<tr>
<td>Assembly Coding Conventions</td>
<td>6-1</td>
</tr>
<tr>
<td>Condition Code Results</td>
<td>6-4</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS (Cont'd)

SECTION VI INSTRUCTION REPERTOIRE (Cont'd)

<table>
<thead>
<tr>
<th>Example</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examp 1 es</td>
<td>6-4</td>
</tr>
<tr>
<td>Instruction Mnemonics</td>
<td>6-4</td>
</tr>
<tr>
<td>Assembler Coding Conventions</td>
<td>6-5</td>
</tr>
<tr>
<td>Instruction Definition Format</td>
<td>6-5</td>
</tr>
<tr>
<td>Load/Store Instructions</td>
<td>6-7</td>
</tr>
<tr>
<td>Register Transfer Instructions</td>
<td>6-44</td>
</tr>
<tr>
<td>Memory Management Instructions</td>
<td>6-58</td>
</tr>
<tr>
<td>Writable Control Storage (WCS) Instructions</td>
<td>6-63</td>
</tr>
<tr>
<td>Branch Instructions</td>
<td>6-68</td>
</tr>
<tr>
<td>Branch Programming</td>
<td>6-69</td>
</tr>
<tr>
<td>Compare Instructions</td>
<td>6-81</td>
</tr>
<tr>
<td>Logical Instructions</td>
<td>6-94</td>
</tr>
<tr>
<td>Shift Operation Instructions</td>
<td>6-112</td>
</tr>
<tr>
<td>Bit Manipulation Instructions</td>
<td>6-126</td>
</tr>
<tr>
<td>Fixed-Point Arithmetic Instructions</td>
<td>6-136</td>
</tr>
<tr>
<td>Floating-Point Arithmetic Instructions</td>
<td>6-170</td>
</tr>
<tr>
<td>Control Instructions</td>
<td>6-180</td>
</tr>
<tr>
<td>Interrupt Instructions</td>
<td>6-199</td>
</tr>
<tr>
<td>Input/Output Instructions</td>
<td>6-213</td>
</tr>
<tr>
<td>Class F I/O Instructions</td>
<td></td>
</tr>
<tr>
<td>IOCD Format for Class F I/O WCS.</td>
<td>6-214</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SECTION VII CONTROL PANEL

| General                           | 7-1  |
|                                  | 7-1  |
| Panel Lock                        | 7-1  |
| Power                             | 7-1  |
| Run/Halt                          | 7-1  |
| System Reset                      | 7-1  |
| Attention                         | 7-1  |
| Initial Program Load              | 7-1  |
| Clock Override                    | 7-1  |
| Operation/Mode Indicators         | 7-1  |
| Parity Error                      | 7-1  |
| Interrupt Active                  | 7-3  |
| Clock Override                    | 7-3  |
| Run                               | 7-3  |
| Halt                              | 7-3  |
| Wait                              | 7-3  |
| Keyboards                         | 7-3  |
| Hexadecimal Keyboard              | 7-3  |
| Function Keyboard                 | 7-4  |
TABLE OF CONTENTS (Cont'd)

SECTION VII CONTROL PANEL (Cont'd)

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRITE Key</td>
<td>7-4</td>
</tr>
<tr>
<td>X</td>
<td></td>
</tr>
<tr>
<td>READ Key</td>
<td>7-4</td>
</tr>
<tr>
<td>X</td>
<td></td>
</tr>
<tr>
<td>WRITE &amp; INC 'A' Key</td>
<td>7-4</td>
</tr>
<tr>
<td>INC 'A' &amp; READ Key</td>
<td>7-4</td>
</tr>
<tr>
<td>EXT FUNCT Key</td>
<td>7-5</td>
</tr>
<tr>
<td>INSTR STOP Key</td>
<td>7-5</td>
</tr>
<tr>
<td>OPRND R STOP Key</td>
<td>7-5</td>
</tr>
<tr>
<td>OPRND W STOP Key</td>
<td>7-5</td>
</tr>
<tr>
<td>INSTR STEP Key</td>
<td>7-5</td>
</tr>
<tr>
<td>KEYBOARD Key</td>
<td>7-5</td>
</tr>
<tr>
<td>Panel Displays</td>
<td></td>
</tr>
<tr>
<td>A-Display</td>
<td>7-6</td>
</tr>
<tr>
<td>B-Display</td>
<td>7-7</td>
</tr>
<tr>
<td>Odd/Even Indicators</td>
<td></td>
</tr>
<tr>
<td>EVEN REGISTER Indicator</td>
<td>7-8</td>
</tr>
<tr>
<td>ODD REGISTER Indicator</td>
<td>7-8</td>
</tr>
<tr>
<td>Miscellaneous Indicators</td>
<td></td>
</tr>
<tr>
<td>MEMORY ADDRESS Indicator</td>
<td>7-8</td>
</tr>
<tr>
<td>PSW Indicator</td>
<td>7-8</td>
</tr>
<tr>
<td>PROGRAM COUNTER Indicator</td>
<td>7-8</td>
</tr>
<tr>
<td>OPERATOR FAULT Indicator</td>
<td>7-8</td>
</tr>
<tr>
<td>MEMORY DATA Indicator</td>
<td>7-9</td>
</tr>
<tr>
<td>EFFECTIVE ADDRESS Indicator</td>
<td>7-9</td>
</tr>
<tr>
<td>ERROR Indicator</td>
<td>7-9</td>
</tr>
<tr>
<td>CONTROL SWITCHES Indicator</td>
<td>7-9</td>
</tr>
<tr>
<td>KEYBOARD Indicator</td>
<td>7-9</td>
</tr>
<tr>
<td>INSTRUCTION Indicator</td>
<td>7-9</td>
</tr>
<tr>
<td>STOP Indicator</td>
<td>7-9</td>
</tr>
<tr>
<td>INSTR STOP Indicator</td>
<td>7-10</td>
</tr>
<tr>
<td>OPERAND READ STOP Indicator</td>
<td>7-10</td>
</tr>
<tr>
<td>OPERAND WRITE STOP Indicator</td>
<td>7-10</td>
</tr>
<tr>
<td>OPERATOR FAULT Indicator</td>
<td>7-10</td>
</tr>
<tr>
<td>ERROR Indicator</td>
<td>7-10</td>
</tr>
<tr>
<td>Miscellaneous Indications</td>
<td>7-11</td>
</tr>
<tr>
<td>Operating Instructions</td>
<td></td>
</tr>
<tr>
<td>Load B-Display From Hex Keyboard</td>
<td>7-11</td>
</tr>
<tr>
<td>Load A-Display</td>
<td>7-11</td>
</tr>
<tr>
<td>Write Memory Address</td>
<td>7-12</td>
</tr>
<tr>
<td>Write PSW</td>
<td>7-12</td>
</tr>
<tr>
<td>Read PSW</td>
<td>7-12</td>
</tr>
<tr>
<td>Write PSD2</td>
<td>7-13</td>
</tr>
<tr>
<td>Read PSD2</td>
<td>7-13</td>
</tr>
<tr>
<td>Write Program Counter</td>
<td>7-13</td>
</tr>
<tr>
<td>Read Program Counter</td>
<td>7-13</td>
</tr>
<tr>
<td>Write Memory (Single Address)</td>
<td>7-14</td>
</tr>
<tr>
<td>Read Memory (Single Address)</td>
<td>7-14</td>
</tr>
<tr>
<td>Instruction Step</td>
<td>7-15</td>
</tr>
<tr>
<td>Read Effective Address</td>
<td>7-15</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS (Cont'd)

SECTION VII CONTROL PANEL (Cont'd)

Convert Address ............................................................... 7-16
Stop Sequence ........................................................................ 7-16
Control Switches Sequence ..................................................... 7-17
Write Control Switches ........................................................... 7-17
Read Control Switches ............................................................ 7-17
Initial Program Load Sequence .............................................. 7-18

SECTION VIII SYSTEM INITIALIZATION

Initial Program Load (IPL) ....................................................... 8-1
Formats of the Initial Configuration Load (ICL) ......................... 8-1
Format #1 ............................................................................. 8-2
Format #2 ............................................................................. 8-3
Format #3 ............................................................................. 8-3
Examples of Initial Configuration Load (ICL) Records) ................. 8-3

APPENDICES

APPENDIX A Instruction Set (Functionally Grouped) ....................... A-1
APPENDIX B Hexadecimal-Decimal Conversion Table ...................... B-1
APPENDIX C Hexadecimal Conversion Table ................................ C-1
APPENDIX D Hexadecimal Additions ........................................... D-1
APPENDIX E Numerical Information .......................................... E-1
APPENDIX F Table of Powers of Sixteen and Tables of Powers of Ten .. F-1
APPENDIX G ASCII Interchange Code Set with Card Punch Codes ...... G-1
LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>System Block Diagram Example: Typical 32/70 Series System with Core Memory</td>
<td>1-18</td>
</tr>
<tr>
<td>1-2</td>
<td>System Block Diagram Example: Typical 32/70 Series System with MOS Memory</td>
<td>1-19</td>
</tr>
<tr>
<td>2-1</td>
<td>Program Status Word (PSW) Format</td>
<td>2-4</td>
</tr>
<tr>
<td>2-2</td>
<td>Program Status Doubleword (PSD) Format</td>
<td>2-6</td>
</tr>
<tr>
<td>2-3</td>
<td>Trap Context Block Format - Class F I/O Interrupts</td>
<td>2-8</td>
</tr>
<tr>
<td>2-4</td>
<td>Microinstruction Format</td>
<td>2-10</td>
</tr>
<tr>
<td>2-5</td>
<td>Functional Interrelationship: CPU, WCS, and High-Speed FPU</td>
<td>2-11</td>
</tr>
<tr>
<td>2-6</td>
<td>Optional High-Speed Floating-Point Unit</td>
<td>2-13</td>
</tr>
<tr>
<td>2-7</td>
<td>Program Status Doubleword (PSD) Format</td>
<td>2-17</td>
</tr>
<tr>
<td>2-8</td>
<td>System Configuration With Class F I/O Organization</td>
<td>2-25</td>
</tr>
<tr>
<td>2-9</td>
<td>Optional High-Speed Floating-Point Unit</td>
<td>2-25</td>
</tr>
<tr>
<td>2-10</td>
<td>Microinstruction Format</td>
<td>2-25</td>
</tr>
<tr>
<td>2-11</td>
<td>System Configuration With Class F I/O Organization</td>
<td>2-25</td>
</tr>
<tr>
<td>2-12</td>
<td>Functional Interrelationship of the IPU, WCS, and High-Speed Floating Point Unit</td>
<td>2-26</td>
</tr>
<tr>
<td>2-13</td>
<td>Trap Context Block Format - (Internal Processing Unit)</td>
<td>2-28</td>
</tr>
<tr>
<td>2-14</td>
<td>Trap Context Block Format - (Class F I/O Interrupts)</td>
<td>2-28</td>
</tr>
<tr>
<td>3-1</td>
<td>Trap Context Block Format - External Interrupts and Non-Class</td>
<td>3-7</td>
</tr>
<tr>
<td>3-2</td>
<td>Trap Context Block Format</td>
<td>3-7</td>
</tr>
<tr>
<td>3-3</td>
<td>Trap Context Block Format - Class F I/O Interrupts</td>
<td>3-9</td>
</tr>
<tr>
<td>4-1</td>
<td>Information Boundaries in Memory</td>
<td>4-7</td>
</tr>
<tr>
<td>4-2</td>
<td>Map Image Descriptor List</td>
<td>4-10</td>
</tr>
<tr>
<td>4-3</td>
<td>Memory Management Components</td>
<td>4-11</td>
</tr>
<tr>
<td>4-4</td>
<td>Formats for SDI and PSD2</td>
<td>4-13</td>
</tr>
<tr>
<td>4-5</td>
<td>Map Segment Control Descriptor (MSCD)</td>
<td>4-18</td>
</tr>
<tr>
<td>4-6</td>
<td>Map Segment Descriptor (MSD)</td>
<td>4-18</td>
</tr>
<tr>
<td>4-7</td>
<td>Map Image Descriptor (MID)</td>
<td>4-18</td>
</tr>
<tr>
<td>4-8</td>
<td>Address Generation (512 KB Mode)</td>
<td>4-19</td>
</tr>
<tr>
<td>4-9</td>
<td>Address Generation (512 KB Extended Mode)</td>
<td>4-20</td>
</tr>
<tr>
<td>4-10</td>
<td>Address Generation (512 KB Mapped Mode)</td>
<td>4-21</td>
</tr>
<tr>
<td>4-11</td>
<td>Address Generation (Mapped, Extended Mode)</td>
<td>4-22</td>
</tr>
<tr>
<td>5-1</td>
<td>32/70 Series Input/Output Organization</td>
<td>5-2</td>
</tr>
<tr>
<td>5-2</td>
<td>Block Diagram - Regional Processing Unit (RPU)</td>
<td>5-3</td>
</tr>
<tr>
<td>5-3</td>
<td>Class 0, 1, 2, and E I/O Organization</td>
<td>5-6</td>
</tr>
<tr>
<td>5-4</td>
<td>Command Device Instruction Format</td>
<td>5-6</td>
</tr>
<tr>
<td>5-5</td>
<td>Command Device Instruction Format for Peripheral Devices</td>
<td>5-7</td>
</tr>
<tr>
<td>5-6</td>
<td>Transfer Control Word Format</td>
<td>5-8</td>
</tr>
<tr>
<td>5-7</td>
<td>Test Device Instruction Format</td>
<td>5-9</td>
</tr>
<tr>
<td>5-8</td>
<td>Test Device 2000 Status Information</td>
<td>5-9</td>
</tr>
<tr>
<td>5-9</td>
<td>Block Diagram - I/O Microprogrammable Processor</td>
<td>5-12</td>
</tr>
<tr>
<td>5-10</td>
<td>System Configuration With Class F I/O Processor</td>
<td>5-14</td>
</tr>
<tr>
<td>5-11</td>
<td>I/O Control Words (Class F)</td>
<td>5-15</td>
</tr>
<tr>
<td>5-12</td>
<td>Input/Output Command Doubleword (I0CD)</td>
<td>5-19</td>
</tr>
<tr>
<td>6-1</td>
<td>Positioning of Information Transferred Between Memory and Registers</td>
<td>6-9</td>
</tr>
<tr>
<td>7-1</td>
<td>32/70 Series Serial Control Panel</td>
<td>7-2</td>
</tr>
<tr>
<td>8-1</td>
<td>Transfer Control Word Format</td>
<td>8-8</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Relationship of CPU Modes</td>
</tr>
<tr>
<td>2-1</td>
<td>PSW and PSD Modes: Functional Differences</td>
</tr>
<tr>
<td>2-2</td>
<td>PSD Mode (IPU)</td>
</tr>
<tr>
<td>2-3</td>
<td>CPU/IPU Communication Traps</td>
</tr>
<tr>
<td>2-4</td>
<td>IPU Status Word Bit Definitions</td>
</tr>
<tr>
<td>3-1</td>
<td>PSW/PSD Mode Relative Trap/Interrupt Priorities</td>
</tr>
<tr>
<td>5-1</td>
<td>Transfer Control Word Format Code</td>
</tr>
<tr>
<td>6-1</td>
<td>Symbol Definitions</td>
</tr>
<tr>
<td>6-2</td>
<td>Assembler Coding Symbols</td>
</tr>
<tr>
<td>6-3</td>
<td>32/70 Series Relative Trap/Interrupt Priorities</td>
</tr>
</tbody>
</table>
This equipment generates, uses, and can radiate radio frequency energy, and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. As temporarily permitted by regulation it has not been tested for compliance with the limits for Class A computing devices pursuant to subpart J of part 15 of FCC rules, which are designated to provide reasonable protection against such interference. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.
SECTION I
GENERAL DESCRIPTION

INTRODUCTION

SYSTEM OVERVIEW

The 32/70 Series computer systems are high-speed, general purpose, digital systems that are designed for a variety of scientific, data acquisition, and real-time applications. A basic system includes a central processor, main memory subsystem, and microprogrammed input/output controllers. Each major system element operates semi-independently with respect to the other elements.

The basic system can be readily expanded to accommodate the user's requirements. Main memory (Core or MOS) has addressing space for 16 million bytes. In a multiprocessor environment, memory can be configured with up to 20 access routes. Input/output capability can be increased by adding more I/O Micro-programmable Processors (IOMs), Regional Processing Units (RPUs), multiplexers, device controllers, and I/O devices.

The CPU has a large instruction set that includes fixed- and floating-point arithmetic instructions. A special lookahead feature enables the CPU to overlap instruction execution with memory accessing, thereby reducing program execution time. A large main memory of up to 16 million bytes (4M words) is available. The memory can consist of up to 16 module increments on each of up to 16 memory buses. Memory can be shared by up to eight CPUs and their associated I/O processors.

Each memory module operates independently of all others and address interleaving can be provided between adjacent modules. This multiaccess memory subsystem with interleaving provides system performance far superior to other design concepts. A 32/70 Series system can support up to 16 independent I/O processors of four types - IOMs, RPUs, multiplexers, and high-speed data interfaces - with a maximum aggregate data transfer rate of up to 16.67 million bytes per second, concurrent with CPU instruction execution.

The existing 32/35 and 32/55 programs can be run on a 32/70 Series computer in the PSW mode. The upward compatibility of the software (assemblers, compilers, mathematical and utility routines, and application packages) virtually eliminates reprogramming.

GENERAL CHARACTERISTICS

All 32/70 Series computer systems contain features and functional characteristics that promote efficient operation in general purpose, multiprocessing, real-time, and multiusage environments.

- Byte-oriented memory (8-bit byte plus one parity bit) which can be addressed and altered as bit, byte (8-bit), halfword (2-byte), word (4-byte), and doubleword (8-byte) quantities.
- 600- or 900-nanosecond core memory.
- 900-nanosecond MOS memory with error checking and correction.
- Both core and MOS memory expandable to 16,777,216 (16M) bytes in some models.
- Indexed addressing capability (PSW or PSD mode with extended addressing) of entire memory.
- Multilevel indirect addressing with indexing at each level.
- Immediate operand instructions for greater storage efficiency and increased speed.
- Eight general purpose registers that may be used for arithmetic, logical, and shift operations, as well as masking, linking, and indexing.
- Hardware memory mapping to reduce memory fragmentation and to provide dynamic program relocation.
- Memory write protection to prevent inadvertent destruction of critical areas of memory.
- Real-time priority interrupt system of up to 112 levels with automatic identification and priority assignment; external interrupt levels which can be individually enabled, disabled and requested by program.
- Automatic traps (for error or fault conditions) that have masking capability and maximum recoverability under program control.
- Power fail-safe for automatic shutdown in the event of power failure and resumption of processing after power is restored.
- Multiple interval timers with a choice of resolutions for independent time bases.
- Privileged instruction logic for program integrity in multiusage environments.
- A complete instruction set that includes the following:
  - Bit, byte, halfword, word, and doubleword operations.
  - Register-to-register operations with halfword instructions to improve program execution time.
  - Fixed-point integer arithmetic operations on byte, halfword, word, and doubleword operands.
  - Floating-point arithmetic operations in single and double precision formats.
  - Full complement of logical operations (AND, OR, Exclusive OR) for bytes, halfwords, words, and doublewords.
  - Comparison operations for bit, byte, halfword, word, and doubleword operands.
- Call Monitor and Supervisory Call instructions that allow a program access to operating system functions.
- Shift operations (left and right) of word or doubleword, including logical, circular, and arithmetic shifts.

- Built-in reliability and maintainability features:
  - Full parity checking of all memory accesses.
  - Address stop feature that permits operator or maintenance personnel to:
    - Stop on any instruction address.
    - Stop on any memory read reference address.
    - Stop on any memory write reference address.
  - CPU traps, which provide for detection of a variety of CPU and system fault conditions, designed to enable a high degree of system recoverability.

- Independently operating I/O system with up to 16 I/O processors per CPU.

- General Purpose Multiplexer Controller (GPMC) that provides for the concurrent operation of up to 16 devices on one I/O processor.

- High-Speed Data interface (HSD) for use with high-speed devices, that allows data transfer rates of up to 3.2 million bytes per second.

- Comprehensive software that is upward program compatible with the 32/35 and 32/55 computers.
  - Expands in capability and speed as system grows.
  - Real-Time Monitor (RTM and Mapped Programming Executive (MPX32)).
  - Language processors that include: Extended FORTRAN IV, ANS COBOL, BASIC, assembler, utilities, and applications software for real-time and scientific users.

- Standard and special purpose peripheral equipment:*  
  - Cartridge Disc Units - 10 million byte capacity per unit, peak transfer rate of 312K bytes per second, average access time of 35 milliseconds.
  - Moving-Head Fixed Media Disc - 24 million byte capacity per unit, transfer rates of 1.2 million bytes per second, average access time of 40 milliseconds.
  - Moving-Head Disc - Units available with 40, 80, or 300 million byte per unit capacity, transfer rates of 1.2 million bytes per second, average access time of 30 milliseconds.
- Magnetic Tape Units 9-track, 800/1600 bpi, IBM compatible, high-speed units operating at 75 inches per second with transfer rates up to 120,000 bytes per second; other units operating at 45 inches per second with transfer rates up to 72,000 bytes per second.

- Card Equipment Reading speeds up to 1,000 cards per minute.

- Line Printers Fully buffered with speeds up to 900 lines per minute, 132 print positions with 64 characters.

- Keyboard/Printers 30 characters per second.

- Paper Tape Equipment Readers with speeds up to 300 characters per second, punches with speeds up to 120 characters per second.

- Data Communications Equipment Asynchronous, synchronous, and bisynchronous communications equipment to connect remote user terminals to the computer system via common carrier lines and local terminals directly.

* Some packaged 32/70 Series systems are restricted in regard to peripherals due to environmental requirements.

A basic 32/70 Series System has the following standard features:

- A CPU that includes:
  - Floating-point arithmetic
  - Memory map with access protection
  - Memory write protection
  - Power fail-safe

- Real-Time Option Module that includes:
  - A real-time clock
  - A programmable interval timer
  - Sixteen interrupt levels

- Core or MOS memory (maximum amount and type varies depending on model).

- Teletype, Line Printer, and Card Reader (TLC) controller with three subchannels.

A 32/70 Series system can have the following optional features:

- High-Speed Floating-Point option with up to four times the performance of the standard unit for both single and double precision operands.
GENERAL PURPOSE FEATURES

- Six additional Real-Time Option Modules
- Writable Control Storage (WCS): up to 4,096 64-bit words.
- An additional 96 external priority interrupts per CPU.
- Up to 13 High-Speed Data interfaces (HSD)
- Up to five General Purpose Multiplexer Controllers (GPMCs).
- Memory shared by up to eight CPUs.
- Up to 16 device controllers with each GPMC.
- Up to 13 user-microprogrammable General Purpose I/O modules (GPIOs) and Regional Processing Units (RPUs).
- Up to 13 high-speed controllers, such as magnetic tape and disc.

Floating-point instructions are available in both single (32-bit) and double (64-bit) precision formats.

Indirect addressing facilitates table linkages and permits keeping data sections of a program separate from procedure sections for ease of maintenance.

The large instruction set (up to 189 instructions in some models) permits short, highly optimized programs to be written that minimize both program space and execution time.

Monitor and Supervisory Call instructions permit access to specified operating system services.

A four-bit condition code simplifies the checking of results by automatically providing information on instruction execution. It includes indicators for arithmetic exception, zero, minus, and plus, as appropriate.

Regional Processing Units (RPU) implement intelligent I/O controllers. Once initialized, an RPU operates independently of the CPU, leaving it free to provide fast response to system needs. The RPU requires minimal interaction with the CPU. Thus, many I/O devices can operate simultaneously without overloading the CPU.

The High-Speed Data Interface (HSD) is a single channel parallel controller that interfaces directly to the SelBUS. Once initiated, I/O operations proceed independently of the CPU. The HSD sustains a data transfer rate of up to three million bytes per second.

All 32/70 Series Computer systems include the following general purpose features:
Hardware Memory Management of 32/70 Series core or MOS memory - which is available in sizes up to 16 million bytes and provides the needed capacity while assuring the potential for expansion - makes efficient use of available memory. The memory map hardware permits storing a user's program in segments of 8,192 words, wherever space is available. All segments appear as a single, contiguous block of storage at execution time. The memory map also automatically handles dynamic program relocation so the program appears to be stored in a standard way at execution time. Actually, it can be stored in a different set of locations each time it is brought into memory.

Real-time applications require: (1) hardware to respond quickly to an external environment, (2) speed to keep up with the real-time process and (3) input/output flexibility to handle a wide variety of data types at varying speeds. A 32/70 Series system provides the following real-time computing features:

Multilevel, Priority Interrupt Structure of the real-time oriented 32/70 Series systems provides a quick response to interrupts with a maximum of 112 interrupt levels. The source of each interrupt is automatically identified and responded to according to its priority. For further flexibility, each level can be individually disabled to discontinue input acceptance and to defer responses.

The way interrupt levels are programmed is not affected by the priority assignment.

Programs that deal with interrupts from special purpose devices often require checkout before the equipment is actually available. To simulate special equipment, any external interrupt level can be requested by the CPU by executing a single Request Interrupt (RI) instruction. This capability is also useful in establishing a modified hierarchy of responses. For example, when servicing a high-priority interrupt and the urgent processing is finished, it is often desirable to assign a lower priority to the rest of the service routine so that the interrupt system can respond to other critical stimuli. A service routine can do this by requesting a lower-priority interrupt level, and thereby process the remaining data after other interrupts have been serviced.

Real-Time Clocks are needed to handle the real-time functions that must be timed to occur at specific instants. Other timing information is also needed, such as elapsed time since a given event or the current time of day. Clocks also allow easy handling of separate time bases and relative time priorities. A 32/70 can support up to seven real-time clocks synchronized to a line frequency of 50 Hz or 60 Hz. The clocks can also run at twice the line frequency, 100 Hz or 120 Hz, or on an external source.

Programmable Interval Timers can be set to request an interrupt after any specified time period with a 300-nanosecond resolution. In addition to the real-time clocks, the system can support seven programmable interval timers.

Context Switching must be done quickly with a minimum of time overhead. When responding to a new set of interrupt-initiated circumstances, a computer system must preserve the current operating environment, so the program can continue later, while setting up the new environment. In a 32/70 Series system, all relevant information about the current environment (instruction address, privilege state, condition codes, address modes, etc.) is kept in a 32-bit Program Status Word (PSW) or 64-bit Doubleword (PSD).
When an interrupt occurs, the CPU stores the current PSW or PSD in the memory location(s) selected by the interrupt level and loads a new PSW or PSD to establish a new environment.

Every 32/70 Series system also includes a Load File and Store File instruction so that the entire set of general purpose registers can be loaded or stored with one instruction. These instructions help make context switching fast and easy.

Quick Response is a 32/70 Series feature which involves the following combination: rapid context switching, store file and load file instructions, and a priority interrupt system. These features benefit all users because more of the system's resources are available for useful work at any given time.

Memory Protection features that protect each user from every unprivileged user also guarantee the integrity of programs essential to critical real-time applications.

Input/Output requirements are available for a wide range of capacities and speeds. The 32/70 Series I/O system satisfies the needs of many different application areas economically and efficiently in terms of equipment and programming.

A 32/70 Series system can run programs from two or more computer application areas concurrently. The most difficult general computing problem is the real-time application because it has several requirements. The most difficult multiusage problem is a terminal-oriented application that includes one or more real-time processes. Because the 32/70 Series systems have been designed on a real-time base, they are uniquely qualified for a mixture of applications in a multiusage environment. Many hardware features that prove valuable for one application area are useful in others, although in different ways. This multiple capability makes a 32/70 Series system particularly effective in multiusage applications.

The Instruction Set is large enough to provide the computational and data-handling capabilities required for widely differing application areas. This allows user programs to be short and fast.

Memory Protection makes it possible to run both real-time and batch programs concurrently in a 32/70 Series system. Real-time programs are protected against destruction by unchecked batch programs. Under Real-Time Monitor Control, the memory write-protection feature prevents destruction of information in protected memory.

Variable Precision Arithmetic is important in real-time systems where the data encountered is often 16 bits or less. To process this data efficiently, as well as the data in a batch environment, the 32/70 Series computers provide bit, byte, halfword, word, and doubleword arithmetic.

Priority Interrupts are especially useful because they make it possible for many elements to operate simultaneously and asynchronously. An interrupt system allows the computer to respond quickly and in proper sequence to the many demands made upon it.
Every 32/70 Series computer is designed to function as a shared-memory, multiprocessor system. It can support up to 20 Central Processor Units that share memory, and may have up to 16 Input/Output Microprogrammable Processors per CPU. All processors in a 32/70 Series system can address shared memory using identical addresses.

The 32/70 Series computers have the following major features that allow expansion of a single processor to a multiprocessor system:

Multicore Interlock. In a multiprocessor system, a Central Processor Unit (CPU) often needs exclusive control of a system resource. This resource can be a region of memory, a particular peripheral device, or in some cases, a specific software routine. The 32/70 Series computers have a special set of instructions to provide this required multiprocessor interlock. The special instructions are Set Bit in Memory, Reset Bit in Memory, Test Bit in Memory, and Add Bit in Memory. The Set Bit in Memory instruction sets a bit in the selected position of the referenced memory location before other CPUs are allowed to access that memory location. If this bit had been previously set by another CPU, the interlock is set and the testing program proceeds to another task. On the other hand, if the bit of the tested location is a zero, the resource is allocated to the testing CPU. Simultaneously, the interlock can be set to lock out any other CPU.

Private Memory. Each CPU in a multiprocessor system must retain some private memory for its trap and interrupt locations, I/O communication locations, and other dedicated locations. This private memory consists of at least 8,192 words for each CPU. This private memory must begin with real address zero. The implicitly assigned trap locations and interrupt locations occupy the first 1,096 words of private memory. The remaining words in private memory can be used as private, independent storage by the CPU.

The major elements of a typical 32/70 Series computer system include: the SelBUS, a Central Processor Unit, a Real-Time Option Module, main memory, an input/output subsystem, and a System Control Panel (Figures 1-1 and 1-2 for system block diagram examples). The overall computer system can be viewed as a group of program-controlled subsystems communicating with a common memory. Each subsystem operates semi-independently with automatic overlap of subsystem operation occurring when conditions permit. This overlap greatly enhances the speed of operation. The major elements are listed below along with a brief functional description.

1. SelBUS - provides for high-speed communication between the major system elements.

2. Central Processor Unit - performs overall control and data reduction tasks.

3. Real-Time Option Module - implements internal and external interrupts and traps.

4. Main Memory - provides for private and shared storage.
Figure 1-1: System Block Diagram Example:

Typical 32/70 Series System

- CORE MEMORY MODULE NO 1
- CORE MEMORY MODULE NO 4
- CORE MEMORY MODULE NO 16
- MEMORY BUS NO 1
- MEMORY BUS NO 2
- SERIAL CONTROL PANEL
- CENTRAL PROCESSOR UNIT (CPU)
- WRITABLE CONTROL STORAGE (WCS)
- HIGH-SPEED (HSFP) FLOATING-POINT OPTION
- MEMORY BUS SERIAL CONTROLLER (MBC)
- USER FURNISHED DEVICE
- REGIONAL PROCESSING UNIT (RPU)
- RANDOM ACCESS MEMORY
- INPUT/OUTPUT MICROPROGRAMMABLE PROCESSOR (IOM)
- MEMORY BUS CONTROLLER (MBC)
- TTY LP CR CONTROLLER (TLC)
- TTY LP CR
- STANDARD PERIPHERAL DEVICES
  - DATA ACQUISITION SYSTEMS
  - COMMUNICATIONS EQUIPMENT
  - MULTIPLEXED I/O BUS
  - 6 EXT INTERRUPTS

- REAL-TIME OPTION MODULE (RTOM)
  - 16 EXT INTERRUPTS
Figure 1-2. System Block Diagram Example.
Typical 32/70 Series System with MOS Memory

- MOS MEM MOD 256 KB (1)
- MOS MEM MOD 256 KB (2)
- MOS MEM MOD 256 KB (6)
- MEMORY BUS
- MOS MEMORY BUS CONTROLLER
- CONTROL PANEL W/HEX DISP
- HIGH SPEED FLOATING POINT
- CENTRAL PROCESSOR UNIT
- WRITABLE CONTROL STORAGE
- MEMORY BUS
- MOS MEM MOD 256 KB (1)
- MOS MEM MOD 256 KB (7)
- MOS MEM MOD 256 KB (8)
- SELBUS (26.67 MB/SEC)
- ADDITIONAL REAL-TIME OPTION MODULE
- 16 EXT INTERRUPTS
- FIRST REAL-TIME OPTION MODULE
- 6 EXT INTERRUPTS
- MAGNETIC TAPE CONTROLLER
- TLC CONTROLLER
- CARTRIDGE DISC CONTROLLER
- REGIONAL PROCESSING UNIT
- USER FURNISHED DEVICE
- 80 MB DISC
- 80 MB DISC (1)
- CARDS READER (300 CPM)
- A/N OR CRT
- LINE PRINTER (600 LPM)
- CARD READER (300 CPM)
- 75 IPS MTU (1)

OPTIONAL EQUIPMENT
SelBUS

5. Input/Output Subsystem - enables information exchange between memory and selected peripheral devices.

6. System Control Panel - provides for user interaction with the system.

SelBUS is a 184-line bidirectional bus that sends and receives data between the CPU, the memory subsystem, the Regional Processing Unit (RPU), the Input/Output Microprogrammable Processors (IOMs) on 32 data lines at a continuous data rate of 26.67 million bytes per second. Twenty-four address lines are used to address the selected IOM or memory interface for a read or write operation. Both data and address lines operate concurrently, and the transfers occur every 150 nanoseconds.

In a multiprocessor or special system configurations, remote memory subsystems, dual-processor shared-memory options, and memory ports may be connected to the SelBUS to support remote, shared, or private memory.

The SelBUS is a 184-line bidirectional bus that sends and receives data between the CPU, the memory subsystem, the Regional Processing Unit (RPU), the Input/Output Microprogrammable Processors (IOMs) on 32 data lines at a continuous data rate of 26.67 million bytes per second. Twenty-four address lines are used to address the selected IOM or memory interface for a read or write operation. Both data and address lines operate concurrently, and the transfers occur every 150 nanoseconds.

In a multiprocessor or special system configurations, remote memory subsystems, dual-processor shared-memory options, and memory ports may be connected to the SelBUS to support remote, shared, or private memory.

Central Processor Unit

The 32/70 Series Central Processor Unit (CPU) is contained on three plug-in circuit boards. Two of the boards are the Micro Arithmetic/Logic Unit. The third board is the Micro Control Unit, which is sometimes referred to as the personality board.

Instructions on a 32/70 Series computer are continuously and automatically fetched for processing. This occurs concurrently with execution and decoding of previous instructions. Decoding is by proprietary parsing logic which employs parallel Read-Only Memories (ROMs) for high-speed decoding.

General Purpose Registers

Eight integrated-circuit, 32-bit general purpose registers (GPRs) are used by the CPU. These eight registers of fast memory are referred to as the general purpose file.

Each general purpose register is identified by a 3-bit code in the range 000 through 111 (0 through 7 in decimal). Any general purpose register can be used as a fixed-point accumulator, floating-point accumulator, or temporary data storage location. A register can also contain control information such as a data address, count, or pointer. General purpose registers 1 through 3 can be used as index registers. Register 4 can be used as a mask register. Register 0 is a link register and an interval timer count.

Floating-Point Arithmetic Processor

A firmware floating-point arithmetic processor is standard with the Central Processor Units. The firmware floating-point arithmetic processor executes all floating-point instructions significantly faster than normal software floating-point routines.

CPU Modes

A 32/70 Series computer can operate in eight different modes: four control modes (PSW-Privileged, PSW-Unprivileged, PSD-Privileged, PSD-Unprivileged) and four addressing modes (512 KB, 512 KB Extended, 512 KB Mapped, Mapped Extended).

The Extended mode can mean either 1 megabyte or 16 megabytes depending on the mapping mode. Table 1-1 shows the interrelationships among the control and address modes.
Table 1-1. Relationship of CPU Modes

<table>
<thead>
<tr>
<th>Control Modes</th>
<th>Addressing Modes</th>
<th>PSW</th>
<th>PSD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Privileged</td>
<td>Unprivileged</td>
<td>Privileged</td>
</tr>
<tr>
<td>Unmapped</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>512 KB</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>512 KB Extended</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Mapped</td>
<td>NA</td>
<td>NA</td>
<td>X</td>
</tr>
<tr>
<td>512 KB</td>
<td>NA</td>
<td>NA</td>
<td>X</td>
</tr>
<tr>
<td>Extended</td>
<td>NA</td>
<td>NA</td>
<td>X</td>
</tr>
</tbody>
</table>
**Control Modes**

The basic control mode is designated either Program Status Word (PSW) or Program Status Doubleword (PSD) mode. The PSW mode allows a 32/70 Series computer to emulate the environment required to run the Real-Time Monitor (RTM); whereas the PSD mode makes it possible to create the environment required to run the Mapped Programming Executive (MPX).

The CPU, when in the PSW mode or PSD mode, can run in either the Privileged or Unprivileged mode.

Privileged operation allows the CPU to perform all of its control functions and to modify any part of the system. It is assumed that the resident operating system (operating in the Privileged mode) controls and supports the execution of other programs (which can operate in the Privileged or Unprivileged mode).

Unprivileged operation is the problem-solving mode of the CPU. In this mode, memory protection is in effect, and all privileged operations are prohibited. Privileged operations are those relating to input/output and to changes in the basic control state of the computer. All privileged operations are performed by a group of privileged instructions. Any attempt by a program to execute a privileged instruction while the computer is in the Unprivileged mode results in a trap.

The Privileged/Unprivileged mode control bit can be changed when the computer is in the Privileged mode. An Unprivileged mode program can gain direct access to certain executive program operations by means of Supervisory Call or Call Monitor instructions. The operations available through these instructions are established by the resident operating system.

**Addressing Modes**

The basic addressing modes are designated either Unmapped or Mapped. Addressing submodes are 512 KB or extended addressing (refer to Table 1-1).

Unmapped addressing establishes a one-to-one relationship between the effective virtual address of each operand or instruction and the physical address in memory.

Mapped addressing uses the memory management hardware to convert effective virtual operand and instruction addresses into physical (real) memory addresses located anywhere in up to 16 megabytes of physical memory. The memory management hardware contains a MAP which allows the privileged user to define how virtual addresses are converted to real addresses.

The MAP contains thirty-two 16-bit registers; the first 16 registers contain the Primary MAP to define a 512 KB primary logical address space, and the second 16 registers contain the Extended Operand Map to define an additional 512 KB extended operand address space for additional data storage.

**Addressing Submodes**

The addressing submodes are 512 KB and extended addressing. 512 KB addressing allows direct addressing of 512K bytes (128K words) of memory. In the 512 KB mode, this address space consists of the first 512K bytes in memory. In the 512 KB Mapped mode, this address space is the 512K bytes of primary logical address space for each user.
Extended Addressing allows a program through indexing to extend the address space beyond 512K bytes. In the Unmapped Extended mode, the extension is to 16 megabytes. In Mapped-Extended mode, provision is made for up to 1 megabyte of logical address space for each user. The mapping hardware can locate this 512 KB space in 8,192-word segments anywhere in up to 16 megabytes of physical memory.

The Hardware Memory Management feature of 32/70 Series computers use dynamic Memory Allocation and Protection (MAP). This allows programs to be loaded in one area of physical memory, rolled out to disc, rolled back into another area of memory, and to continue execution without requiring time-consuming software relocation biasing. In addition, user programs may be write protected and distributed throughout physical memory in 32K-byte blocks. Thus, the full utilization of available memory is a practical possibility.

A memory map deals with virtual and real addresses. A virtual address pertains to the logical space used by a machine-level program and is normally derived from programmer-supplied labels through an assembly (or compilation) process followed by a loading process. Virtual addresses may be used to designate an element of data, the location of an instruction, and either an indirect or immediate (explicit) address. A real (physical) address is the address a processor sends to the memory address register to access a specific physical memory location for storage or retrieval of information. Real addresses are determined by the hardware, whereas virtual addresses include all addresses.

The memory map provides dynamic program relocation into discontinuous segments of memory. When the CPU is operating in Mapped mode, a program can be segmented into an integral number of 8,192-word blocks and distributed throughout memory in whatever space is available. The memory map transforms virtual addresses, as seen by the individual program, into real addresses, as seen by the memory system.

When the CPU is not in the Mapped mode, as determined by a control bit in the Program Status Doubleword (PSD), all virtual addresses are used by the CPU as real addresses. When the CPU is operating in the Mapped mode, all virtual addresses are transformed into real addresses by replacing the high-order four or five bits (dependent upon extended addressing) of the virtual address with a 9-bit value obtained from the memory map register.

The memory protection system provides write protection for individual memory pages. When the CPU is in the Mapped mode (either 512 KB or Extended), each 32 KB memory block of logical program address space may be write protected. Write protection for a 32 KB memory block is selected by setting the protect/unprotect bit that is stored, along with the block address, in the MAP register of the CPU.

When the CPU is in either the Unmapped or Mapped mode (either 512 KB or Extended), 512-word memory pages may be write protected. Up to 256 pages (128K words) can be protected at a time. Sixteen 16-bit Page Protect registers are provided in the CPU for write protection in the Unmapped or Mapped mode.

Write protection may be overridden by a CPU operating in the Privileged mode.
The optional Writable Control Storage (WCS) may be used to expand the 32/70 Series computer instruction repertoire and to enhance the performance of user programs. By microprogramming a 32/70 Series computer with firmware subroutines, the optional Writable Control Storage (WCS) can tailor the computer to perform specific applications such as Fourier transforms, coordinate transformation, polynomial evaluation, and number system conversion.

Further improvement in overall performance is achieved by using microprograms for frequently executed subroutines in the FORTRAN Run-Time Package, the FORTRAN Compiler, the BASIC Interpreter, and the 32/70 operating system. All high-speed firmware subroutines can be invoked from main memory for execution as needed.

Up to 4,096 64-bit words of Writable Control Storage (WCS) can be added to a 32/70 Series computer in increments of 2,048 64-bit words. Each increment plugs into the SelBUS for power and clock. However, communication with the CPU is independent of SelBUS operation.

The optional High-Speed Floating-Point Unit functions as an extension of the 32/70 Series central processor to perform high-speed execution of floating-point arithmetic instructions. Addition, subtraction, multiplication and division of single-precision (32-bit) or double-precision (64-bit) operands are possible with execution times that are significantly greater than with the standard floating-point feature of the CPU.

The first RTOM in the system provides the 10 basic interrupts and traps which comprise the system integrity features. These basic interrupts and traps include: Power Fail-Safe, System Override, Memory Parity, Non-present Memory, Undefined Instruction, Privilege Violation, Attention, Call Monitor, Real-Time Clock, and Arithmetic Exception.

The first RTOM also provides the six highest external interrupt levels, one of which may be used for the standard interval timer.

The programmable interval timer provides a 32-bit counter that can be loaded, examined, started, or stopped by way of a Command Device (CD) instruction. The Command Device (CD) enables the counter at one of four program-selectable rates. When the counter is decremented to zero, the interval timer requests a priority interrupt.

An introduction to the basic organization and operation of the main memory subsystem is provided in the paragraphs that follow.

A 32/70 Series system may have either core or MOS memory. Packaged systems are sold with one or the other but not both for the same system. The user may elect to mix the two types of memory, but only if it is done in accordance with the configuration rules specified in Section III of this manual.

The main memory for a 32/70 Series system is physically organized as a group of units. A memory unit is the smallest logically complete part of the system, and the smallest part that can be logically isolated from the rest of the memory system. A memory unit consists of 1 or 2 memory chassis, a power supply, 1 to 4 Memory Bus Controllers (MBCs), and 1 to 16 memory modules. Memory units with MOS memory also include a Refresh board.
A memory module is the basic functionally independent element of the memory system. Each module can operate concurrently with all others in a memory unit. A memory module consists of storage elements, drive and sense electronics, control timing, and data registers. Core and MOS memory modules are described separately, as follows:

1. Core memory modules have either 8,192-word (32K-byte) locations with a 600-nanosecond cycle time or 16,384-word (64K-byte) locations with a 900-nanosecond cycle time. Each word contains a total of 36 bits: 32 data bits and 4 parity bits (1 parity bit per byte). Byte, halfword, word, or doubleword addresses may be used to access memory.

2. MOS memory modules have either 65,536-word (256K-byte) or 131,072-word (512K-byte) locations; both have a cycle time of 900 nanoseconds. MOS memory is organized into 39-bit words: 32 data bits plus 7 error checking and correction (ECC) bits. The seven error correction bits report and correct single-bit errors. The ECC bits also detect and report (but do not correct) double-bit errors.

When a system consists of two memory modules (or a multiple thereof), memory can be two-way interleaved. If a system has four modules (or a multiple thereof), memory can be four-way interleaved. Memory interleaving is a built-in hardware feature that distributes sequential addresses into independently operating memory modules. Interleaving increases the probability that a processor can gain access to a given memory location without encountering interference from other processors. Thus, interleaving significantly reduces cycle time and increases the throughput rate.

With two-way interleaving, even addresses are assigned to even-numbered memory modules and odd addresses to odd-numbered memory modules. Four-way interleaving assigns every fourth address to its respective memory module and can occur when a multiple of four memory modules are included in a unit.

Each memory unit in a 32/70 Series system is provided with an individual identity by means of address range switches. These switches define the range of addresses to which the unit responds when servicing memory requests. All addresses, including the starting address, for a given unit should be the same for all Memory Bus Controllers (MBCs) in that unit; that is, the address of a given byte remains the same regardless of the MBC used to access the byte. The starting address of a unit must be on a boundary equal to a multiple of the size of the memory modules in the unit. If the unit is interleaved, the unit must contain a multiple of the memory modules'size times the number of interleaves.

The Memory Bus Controllers (MBCs) in a memory unit act as an interface between the processing units (CPUs, IOMs, and RPUs) on the SelBUS and the memory modules. Each memory unit can have from one to four MBCs. Each MBC is capable of managing up to 16 memory modules with overlapped operation. All memory modules assigned to one MBC must be of the same type (either MOS or core but not both) and have the same cycle and access time.

MBCs examine incoming addresses to determine if the request is for a memory module within the memory unit. In addition, an MBC determines the priority of memory requests that are received simultaneously. Computer memory requests can be initiated every 150 nanoseconds due to the overlapped memory design.
The 32/70 Series systems can include from one to eight MBCs per SelBUS. All processors, either CPUs or I/O processors, must interface to memory by way of an MBC. MBCs are located, along with the memory modules, in a separate chassis from the CPU and I/O processors. Depending on the particular system and the needs of the user, an MBC may be configured in a variety of ways. For example, an MBC can connect directly to the SelBUS; or, a Memory Interface Adapter (MIA) and/or Memory Bus Adapter (MBA) may be employed to provide indirect connection between the SelBUS and an MBC.

MBCs can be locked and unlocked by a CPU. A Memory Lock signal can be sent to the MBC in conjunction with a read transfer, and a Memory Unlock signal can be sent during a write transfer. The Read and Lock transfer is used to access a word instruction in memory and to lock out all other processors from the MBC. A Write and Unlock transfer causes information to be written into memory and enables access to the MBC by other SelBUS devices. Only CPUs can use the Lock and Unlock feature.

When a Read and Lock transfer is received, the MBC involved is temporarily inhibited from accepting any additional transfer requests. However, all transfer requests already accepted by the MBC, but not yet completed, will be processed normally.

In a 32/70 Series multiprocessing system, all processors address memory in the same manner. The CPUs do not share the same interrupt or trap systems. Thus, it is necessary to provide private storage for each CPU to contain its trap and interrupt locations, I/O communication locations, and scratchpad locations. This private memory must begin at 0 and extend at least to 2,048 memory locations (bytes).

The Input/Output Microprogrammable Processor is the basic hardware structure of the I/O processor and consists of a SelBUS interface, a microprocessor, and interface logic for an external device.

The SelBUS interface provides for communication between the IOM and the CPU, or between the IOM and memory. The microprocessor has a Control Read-Only Memory (CROM) that contains the microprogram (firmware) for controlling the SelBUS interface, microprocessor, and device interface logic. The device interface logic may consist of some control logic for operating the I/O interface and the receivers/drivers necessary to communicate with the I/O device or external interface.

There are three classes of I/O processors in a 32/70 Series system: the IOM, the RPU, and the General Purpose Multiplexer I/O processor. The I/O processor can also be used to provide a General Purpose Input/Output interface (GPIO). The customer must design the device interface logic and supporting firmware to make the I/O processor and device dependent interface operate as an I/O processor for some specific type of I/O device(s).

The IOM is the basic I/O processor which contains the microprogrammable processor, the SelBUS interface, and the device interface on a single logic card.

The Regional Processing Unit (RPU) serves as a General Purpose Input/Output interface (GPIO) for the peripheral device(s). The RPU connects directly to the SelBUS, the major artery for transmitting information. The RPU consists of three individual elements which are self-contained on separate modules: the regional processor, the device interface, and optional high-speed Random Access Memory (RAM). The major characteristic of the RPU is that it supports Random Access Memory or Writable Control Storage that can be programmed to suit the user's requirements.
A third type of I/O processor is the General Purpose Multiplexer Controller (GPMC) which controls a number of individual controllers that are located at various distances from the processor. The GPMC can schedule requests for main memory between several controllers. The GPMC also connects each dependent controller to the CPU for initiation or termination of an I/O operation.
SECTION II

CENTRAL PROCESSOR

INTRODUCTION

This section of the manual describes the 32/70 Series Central Processor Unit (CPU). Included are an introduction to the instruction repertoire and descriptions of the modes of operation, their format, and the major functional elements of the CPU.

INSTRUCTION REPertoire

The functional classifications and corresponding number of instructions for the 32/70 Series computer are as follows:

<table>
<thead>
<tr>
<th>Classifications</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed-Point Arithmetic</td>
<td>30</td>
</tr>
<tr>
<td>Floating-Point Arithmetic</td>
<td>8</td>
</tr>
<tr>
<td>Boolean</td>
<td>17</td>
</tr>
<tr>
<td>Load/Store</td>
<td>29</td>
</tr>
<tr>
<td>Bit Manipulation</td>
<td>8</td>
</tr>
<tr>
<td>Zero</td>
<td>5</td>
</tr>
<tr>
<td>Shift</td>
<td>13</td>
</tr>
<tr>
<td>Interrupt</td>
<td>13</td>
</tr>
<tr>
<td>Compare</td>
<td>11</td>
</tr>
<tr>
<td>Branch</td>
<td>9</td>
</tr>
<tr>
<td>Register Transfer</td>
<td>13</td>
</tr>
<tr>
<td>Input/Output</td>
<td>10</td>
</tr>
<tr>
<td>Control</td>
<td>16</td>
</tr>
<tr>
<td>Hardware Memory Management</td>
<td>4</td>
</tr>
<tr>
<td>Writable Control Storage</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>189</strong></td>
</tr>
</tbody>
</table>

Of particular significance are the bit manipulation and floating-point instructions. The eight bit manipulation instructions provide the capability to selectively set, zero, add, or test any bit in memory or register.

The eight floating-point instructions are unique because they can either be executed by the firmware in the CPU, or by the optional High-Speed Floating-Point Arithmetic Unit. Except for the execution speed, the presence or absence of the optional Floating-Point Arithmetic Unit is transparent to the user.

All of the instructions in the repertoire are classified as either being halfword instructions (16 bits) or word instructions (32 bits). The word instructions primarily reference memory locations; the halfword instructions primarily deal with register operands. Because approximately one-third of the instructions are halfword instructions, program core space can be conserved by packing two consecutive instructions into one memory location.

The 32/70's use instruction lookahead for fast instruction execution. Instruction fetches are made concurrently with instruction execution and with decoding a previously fetched instruction.
The 32/70 Series CPU has a set of eight high-speed, general purpose registers for use by the programmer for arithmetic, logical, and shift operations. Three general purpose registers - R1, R2, and R3 - can also be used for indexing operations. Register RO can also be used as a link register. Register R4 can be used as a mask register.

The CPU operates in either of two basic control modes: the PSW mode or the PSD mode. The PSW mode provides an environment to run the Real-Time Monitor (RTM) Operating System. The PSD mode provides an environment to run the optional Mapped Programming Executive (MPX-32) Operating System. The functional difference between the PSW and PSD modes are outlined in Table 2-1.

A Program Status Word (PSW) is used to record all machine conditions that must be preserved prior to context switching when in the PSW mode of operation. The PSW supports only the Class 0,1,2,3, and E I/O devices using the Command Device (CD) and Test Device (TD) instructions. The format of the PSW is shown in Figure 2-1.

A Program Status Doubleword (PSD) is used to record all machine conditions that must be preserved prior to context switching when in the PSD mode of operation. The format of the PSD is shown in Figure 2-2. Execution of any Branch-and-Link instruction replaces the contents of bits 13-30 of the PSD with the effective address specified by the instruction. In addition, if the Branch instruction specifies an Indirect Branch operation, the contents of bits 1-4 of the PSD are replaced by the contents of the corresponding bit positions in the indirect address location.

A 4-bit Condition Code is stored in the PSW or PSD upon completion of the execution of most instructions. These conditions may be tested to determine the status of the results obtained.

CC1 is set if an Arithmetic Exception occurs
CC2 is set if the result is greater than Zero
CC3 is set if the result is less than Zero
CC4 is set if the result is equal to Zero

The Branch Condition True (BCT), Branch Condition False (BCF), and the Branch Function True (BFT) instructions allow testing and branching on the condition codes.

The CPU is capable of either privileged or unprivileged operation in both the PSW and PSD modes. Privileged operation allows the CPU to perform all of its control functions and to modify any part of the system. Privileged operation relates to input/output and to changes in the basic control state of the computer. Unprivileged operation is the problem-solving mode of the CPU. In this mode, memory protection is in effect and all privileged operations are prohibited.

One bit in the Program Status Doubleword (PSD) or Program Status Word (PSW) is designated as the Privileged State bit. If the Privileged State bit is set, privileged instructions can be executed. If the Privileged State bit is reset, any attempt to execute a privileged instruction will cause a Privileged Violation trap.
### Table 2-1. PSW and PSD Modes: Functional Differences

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>PSW Mode*</th>
<th>PSD Mode**</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Status</td>
<td>Word</td>
<td>Doubleword</td>
</tr>
<tr>
<td>Number of Instructions</td>
<td>160</td>
<td>189</td>
</tr>
<tr>
<td>Integrity Features</td>
<td>Interrupts on first RTOM</td>
<td>Traps</td>
</tr>
<tr>
<td>Memory Addressing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nonmapped</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nonextended</td>
<td>512 KB</td>
<td>512 KB+</td>
</tr>
<tr>
<td>Extended</td>
<td>16 MB</td>
<td>16 MB+</td>
</tr>
<tr>
<td>Mapped</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nonextended</td>
<td>None</td>
<td>512 KB per user</td>
</tr>
<tr>
<td>Extended</td>
<td>None</td>
<td>1 MB per user</td>
</tr>
<tr>
<td>CD I/O</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Addressing</td>
<td>512 KB</td>
<td>512 KB</td>
</tr>
<tr>
<td>Extended I/O</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Addressing</td>
<td>None</td>
<td>16 MB</td>
</tr>
</tbody>
</table>

* RTM supported  
** MPX supported  
† No software support
|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| P | C_1 | C_2 | C_3 | C_4 | H | S | T | 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 |

**Figure 2-1. Program Status Word (PSW) Format**

- **BIT 0**: DESIGNATES THE PRIVILEGED STATE BIT
- **BIT 1-4**: DESIGNATE THE CURRENT CONDITION CODE
- **BIT 5**: DEFINES THE EXTENDED ADDRESSING MODE (ABOVE 128K)
  - BIT 5 = 0: NONEXTENDED ADDRESSING
  - BIT 5 = 1: EXTENDED ADDRESSING
- **BITS 6**: DEFINES THE POSITION OF THE LAST INSTRUCTION EXECUTED
  - BIT 6 = 0: LEFT HALFWORD OR FULLWORD
  - BIT 6 = 1: RIGHT HALFWORD
- **BITS 7-12**: UNASSIGNED, MUST BE ZERO
- **BITS 13-29**: CONTAIN THE WORD ADDRESS (PC) COUNT OF THE NEXT INSTRUCTION TO BE EXECUTED
- **BIT 30**: DEFINES THE POSITION OF THE NEXT INSTRUCTION (LEFT OR RIGHT INSTRUCTION)
  - BIT 30 = 0: LEFT HALFWORD
  - BIT 30 = 1: RIGHT HALFWORD
BIT 0 = 0 UNPRIVILEGED MODE
    = 1 PRIVILEGED MODE
BIT 5 = 0 EXTENDED MODE (OFF) CEA
    = 1 EXTENDED MODE (ON) SEA
BIT 6 = 0 LAST INSTRUCTION EXECUTED WAS NOT A RIGHT HALFWORD
    = 1 LAST INSTRUCTION EXECUTED WAS A RIGHT HALFWORD
BIT 7 = 0 ARITHMETIC EXCEPTION TRAP MASK (OFF)
    = 1 ARITHMETIC EXCEPTION TRAP MASK (ON)
*BIT 8 = 0 COMPUTER IS IN PSW MODE (DISPLAYED PSD ONLY) *
    = 1 COMPUTER IS IN PSD MODE (DISPLAYED PSD ONLY) *
*BIT 9 = 0 UNMAPPED (DISPLAYED PSD ONLY) *
    = 1 MAPPED (DISPLAYED PSD ONLY) *
BIT 30 NEXT INSTRUCTION IS A RIGHT HALFWORD
*BIT 31 BLOCKED (DISPLAYED PSD ONLY) *
BIT 32-33 INDICATE MAP GRANULARITY, 00=UNMAPPED AND ALL OTHERS =8K MAP GRANULARITY
BIT 34-45 PROVIDE A WORD INDEX INTO THE MASTER PROCESS LIST (MPL) FOR THE BASE PROCESS
BIT 46 NOT USED
BIT 47 RETAIN CURRENT MAP CONTENTS
BIT 48-49 INTERRUPT CONTROL FLAGS

Figure 2-2. Program Status Doubleword (PSD) Format
The following instructions are privileged:

1. All interrupt related instructions such as Enable Interrupt or Request Interrupt.
2. All instructions that can modify the memory mapping registers.
3. All Input/Output instructions.
4. All instructions that can place the machine in a state that requires operator intervention to continue processing, such as Halt.
5. All instructions that modify Writable Control Storage.

User programs operating in the unprivileged state should use the Call Monitor (CALM) or Supervisor Call (SVC) instruction with the appropriate program flags to use the system features guarded by the privileged/unprivileged system.

Certain events can change the processor from the unprivileged to the privileged state by loading a new Program Status Word or Doubleword. These are:

1. An interrupt from an external event or the I/O system.
2. A hardware trap caused by addressing nonpresent memory, executing undefined instruction, executing privileged instruction by nonprivileged program, or writing to protected memory.
3. A hardware trap caused by a nonrecoverable condition such as an uncorrectable error on a memory read, or an arithmetic exception.
4. The execution of the Call Monitor or Supervisor Call instruction by a user requesting monitor services.

In all cases, traps or interrupts are vectored to monitor routines for proper handling. Both the interrupt/trap vectors and the monitor service routines are in protected memory. This insures that an unprivileged user has no way to become privileged or to alter protected state.

The execution of the Branch and Reset Interrupt (SRI) or the Load Program Status Doubleword (LPSD) instruction can cause the system to change from the privileged to the unprivileged state.

The operator can push the SYSTEM RESET button to initialize a 32/70 Series computer. SYSTEM RESET clears the eight general purpose registers, resets all memory protection, and sets the Privileged State bit.

**CPU ADDRESSING MODES**

The 32/70 Series CPU has four modes for accessing memory:

1. 512 KB mode
2. 512 KB Extended Mode
3. 512 KB Mapped mode
4. Mapped, Extended mode
The 512 KB addressing mode allows the 32/70 Series CPU to access instructions or operands (bit, byte, halfword, word, or doubleword) in the first 512K bytes of memory directly without mapping, indexing, or address modification. A 19-bit Address field is provided in memory referencing instructions for that purpose.

Bit addressing is accomplished by using the Register (R) field in the instruction word to select a bit in the byte specified by the 19-bit address. Therefore, any bit in the first 512K bytes of memory can be directly addressed by the Bit Manipulation instructions.

The 512 KB Extended mode provides the same capabilities as the 512 KB mode described above, and, in addition, it permits operand addressing beyond the first 512K bytes of memory. The effective address can reference any bit, byte, halfword, word, or doubleword residing anywhere within 16 megabytes of physical memory.

The 512 KB Mapped mode allows a 32/70 Series CPU to access any instruction or operand (bit, byte, halfword, word, or doubleword) within a logical primary address space. This space consists of 512K bytes of logical memory, distributed within 16 megabytes of physical memory.

The 32/70 Series CPU allows multiple primary address spaces. A user can access instructions and operands within the logical primary address space in which his program resides. Physical blocks of memory can be common to many logical primary address spaces; thus, users in different spaces can share common blocks of memory.

The 512 KB Mapped addressing mode can be used only when the CPU is in the PSD control mode.

The Mapped Extended mode provides all the capabilities of the 512 KB Mapped mode, plus access to a logical extended operand address space. This space consists of 512K bytes of memory beyond the logical primary address space and allows users additional memory space to store data (operands). Each logical extended operand address space can be 512K bytes long, dispersed anywhere within 16 megabytes of physical memory. The combination of logical primary address space and the logical extended operand address space supports programs up to one megabyte long. The executable code must lie within the logical primary address space, but operands can be in either the logical primary or extended operand address space.

The Mapped Extended addressing mode can be used only when the CPU is in the PSD control mode.

A brief description of some major elements of the CPU are provided in the paragraphs that follow. They include: the data structure, a micro-programmable processor, the implementation logic, and the SelBUS interface. A simplified block diagram of the CPU is shown in Figure 2-3. For a more comprehensive discussion of the CPU, refer to the 32/70 Series Computer Technical Manual.

The data structure contains the eight general purpose file registers and 10 hardware registers organized around an Arithmetic Logic Unit (ALU). Key circuits in the data structure include the following:
Figure 2-3. CPU Simplified Block Diagram
1. Arithmetic Logic Unit (ALU)
2. A-Multiplexer
3. B-Multiplexer
4. Literal Multiplexer
5. General File Register
6. Memory Address Register
7. Program Counter Register
8. N-Counter Register
9. Shift Register
10. Temporary Register/Data Output Register
11. Data Input Register
12. Instruction Register 0
13. Instruction Register 1

The Microprogrammable Processor of the CPU is on board C of the three CPU circuit boards. The logic circuit board which contains the Microprogrammable Processor is commonly referred to as the personality board.

The Microprogrammable Processor utilizes Read-Only Memory (ROM) integrated circuits which house the CPU's Elementary Operations (EO). The EOs, with the associated circuitry, control the CPU operations by testing, controlling, and directing the various functions to be performed. The format for the EOs (also referred to as microinstruction) is shown in Figure 2-4.

The Implementation Logic includes the ALU Decode PROM, a Scale circuit, the Floating-Point Assist PROMs, and a Multiply Assist PROM, all of which serve to implement CPU functions.

The SelBUS interface logic is implemented on all three of the CPU circuit boards and provides control and temporary storage for information being output to and input from the SelBUS. Since the SelBUS is the high-speed communication link between system modules external to the CPU, the SelBUS interface logic plays a vital role in CPU operation.

Writable Control Storage is an option which may be used with the 32/70 Series CPU to expand the instruction set, to enhance the performance of user programs, or to tailor the computer to specific user needs.

Up to 4,096 64-bit words of Writable Control Storage (WCS) can be added to a 32/70 Series computer in increments of 2,048 64-bit words. Each increment plugs into the SelBUS for power and clock. However, communication with the CPU is independent of SelBUS operation.

The block diagram in Figure 2-5 shows two optional WCS units as they could be implemented in conjunction with a 32/70 Series CPU and the optional High-Speed Floating-Point Unit.
### Table 2-4. Microinstruction Format

<table>
<thead>
<tr>
<th>T</th>
<th>B</th>
<th>M</th>
<th>A</th>
<th>B</th>
<th>*</th>
<th>D</th>
<th>R</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>CROM00-03</td>
<td>CROM04-06</td>
<td>CROM07-09</td>
<td>CREG10-12</td>
<td>CREG13-15</td>
<td>CREG16-19</td>
<td>CROM20-23</td>
<td>CREG24-26</td>
<td>CREG27-31</td>
</tr>
<tr>
<td>TEST</td>
<td>SEQUENCE CONTROL</td>
<td>CONTROL EXTENDED</td>
<td>A MUX</td>
<td>B MUX</td>
<td>ALU</td>
<td>DESTINATION</td>
<td>FILE READ</td>
<td>Y-ORDER</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

**Notes:**

1. Bits 0-47 of each microinstruction are in the CPU's control ROM.
2. Portions of the format designated for hardware floating-point apply to the optional high-speed floating-point unit (FPU).
3. Bits 48-63 are physically part of a control ROM in the optional high-speed FPU.

**Figure 2-4. Microinstruction Format**
Figure 2-5. Functional Interrelationship of the CPU, WCS, and High-Speed Floating-Point Unit
The High-Speed Floating-Point Unit (FPU) is an option that may be used with a 32/70 Series CPU to increase the speed of floating-point arithmetic operations. The unit consists of two circuit boards which may be plugged in adjacent to the CPU. No alternations in the software are required.

If the High-Speed Floating-Point Unit (FPU) is installed, addition, subtraction, multiplication, and division of single-precision (32-bit) or double-precision (64-bit) operands can be executed much faster than with the CPU's standard floating-point feature.

An operand in floating-point format has three parts: a sign bit, a fraction, and an exponent. The sign bit indicates whether the fraction is a positive or negative value. The fraction is a binary number with an assumed radix point immediately to the left of its most significant bit. The exponent is a 7-bit binary power to which the base 16 is raised. The quantity that the floating-point number represents can be determined by multiplying the fraction by the number 16 raised to the power represented by the exponent.

Two operands of the same format and length are received by the FPU for each arithmetic operation. One operand is input from a CPU general purpose register (GPR), whereas the other operand is input from memory. The precise GPR and memory location are specified in the floating-point instruction. Upon completion of an operation, the result is returned to the CPU general purpose register.

Figure 2-6 illustrates the major functional elements of the FPU, the general routing of operands, and the relationships between the FPU, the CPU, and the SelBUS.

The Model 2005 Processing Unit is a high-performance processor which has been added to the SYSTEMS 32/70 Series Computer line. The Model 2005 processor's role as a Central Processing Unit (CPU) or Internal Processing Unit (IPU) is selected by a jumper on the C board of the processor. Both CPU and IPU on the same SelBUS must be Model 2005 processors. The IPU is designed for a computer configuration in which a large amount of arithmetic calculation is anticipated and is ideal for compute-bound number processing tasks and subroutines. The IPU, a three-board plug-in module, operates on the same SelBUS with a CPU and shares all of memory (including the resident operating system area) with the CPU.

The IPU and CPU operate in parallel, with the IPU executing task level, SYSTEMS 32 code at the same time the CPU is executing. The capability of paralleled processing of instructions allows for faster completion of code which would normally be processed in a serial manner by the CPU. The CPU is responsible for all task scheduling I/O and system services as well as for the execution of its own scheduled tasks.

Options available with the IPU are the Model 2341 High-Speed Floating Point and the Model 2343 and 2347 Scientific Accelerator.

The IPU is similar, in many instances, to the CPU. Because of this similarity, the IPU information presented in this section will emphasize only the differences and the unique aspects compared to the CPU as presented throughout this manual.
Figure 2-6. Optional High-Speed Floating-Point Unit
GENERAL CHARACTERISTICS

- INSTRUCTIONS
  - New Instruction - SIGNAL IPU (SIPU)
  - Instructions not used by IPU
    - Control Instructions
      - Branch and Reset Interrupt (BRI)
    - Interrupt Instructions
      - All Interrupt Instructions except UEI
    - Input/output instructions
      - All instructions

- TRAPS - Six new traps for IPU/CPU Operation
  - End IPU Processing
  - Start IPU Processing
  - IPU Supervisor CALL
  - IPU Errors
  - IPU Call Monitor
  - Stop IPU Processing

- Software
  - Under MPX-32 the IPU can be transparent to the user, or the user can designate which programs run on the IPU and which run on the CPU.
  - Two programs can run simultaneously because of the parallel operation of the IPU and CPU on the SelBUS.
The functional classifications and corresponding number of instructions of the Internal Processing Unit are as follows:

<table>
<thead>
<tr>
<th>Classification</th>
<th>Number of Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed-Point Arithmetic</td>
<td>30</td>
</tr>
<tr>
<td>Floating-Point Arithmetic</td>
<td>8</td>
</tr>
<tr>
<td>Boolean</td>
<td>17</td>
</tr>
<tr>
<td>Load/Store</td>
<td>26</td>
</tr>
<tr>
<td>Bit Manipulation</td>
<td>8</td>
</tr>
<tr>
<td>Zero</td>
<td>5</td>
</tr>
<tr>
<td>Shift</td>
<td>13</td>
</tr>
<tr>
<td>Interrupt</td>
<td>1 UEI</td>
</tr>
<tr>
<td>Compare</td>
<td>11</td>
</tr>
<tr>
<td>Branch</td>
<td>9</td>
</tr>
<tr>
<td>Register Transfer</td>
<td>13</td>
</tr>
<tr>
<td>Input/Output</td>
<td>0 Unimplemented in IPU</td>
</tr>
<tr>
<td>Control</td>
<td>15 BRI unimplemented in IPU</td>
</tr>
<tr>
<td>Hardware Memory Management</td>
<td>4</td>
</tr>
<tr>
<td>Writable Control Storage</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>163</strong></td>
</tr>
</tbody>
</table>

Of particular significance are the bit manipulation and floating-point instructions. The eight bit manipulation instructions provide the capability to selectively set, zero, add, or test any bit in memory or register.

The eight floating-point instructions are unique because they can either be executed by the firmware in the IPU, or by the optional High-Speed Floating-Point Arithmetic Unit. Except for the execution speed, the presence or absence of the optional Floating-Point Arithmetic Unit is transparent to the user.

All of the instructions in the repertoire are classified as either being halfword instructions (16 bits) or word instructions (32 bits). The word instructions primarily reference memory locations; the halfword instructions primarily deal with register operands. Because approximately one-third of the instructions are halfword instructions, program core space can be conserved by packing two consecutive instructions into one memory location.
The IPU uses instruction lookahead for fast instruction execution. Instruction fetches are made concurrently with instruction execution and with decoding a previously fetched instruction.

**GENERAL PURPOSE REGISTERS**

The IPU includes a set of eight high-speed, general purpose registers for programmer use for arithmetic, logical, and shift operations. Three general purpose registers (R1, R2, and R3) can also be used for indexing operations. Register RO can also be used as a link register. Register R4 can be used as a mask register. These registers are distinctly separate from the registers used in the controlling CPU.

**IPU CONTROL MODE**

The IPU operates in the PSD mode. The PSD mode provides an environment to run the Mapped Programming Executive (MPX-32) Operating System. The PSD mode is outlined in Table 2-2.

**PROGRAM STATUS DOUBLEWORD**

A Program Status Doubleword (PSD) is used to record all machine conditions that must be preserved prior to context switching when in the PSD mode of operation. The format of the PSD is shown in Figure 2-7. Execution of any Branch-and-Link instruction replaces the contents of bits 13 through 30 of the PSD with the effective address specified by the instruction. In addition, if the Branch instruction specifies an Indirect Branch operation, the contents of bits 1 through 4 of the PSD are replaced by the contents of the corresponding bit positions in the indirect address location.

**CONDITION CODES**

A four-bit Condition Code is stored in the PSD upon completion of the execution of most instructions. These conditions may be tested to determine the status of results obtained.

- CC1 is set if an Arithmetic Exception occurs
- CC2 is set if the result is greater than zero
- CC3 is set if the result is less than zero
- CC4 is set if the result is equal to zero

The Branch Condition True (BCT), Branch Condition False (BCF), and the Branch Function True (BFT) instructions allow testing and branching of the condition codes.

**PRIVILEGED AND UNPRIVILEGED OPERATION**

The IPU is capable of either privileged or unprivileged operation in the PSD mode. Privileged operation allows the IPU to perform all of its control functions and to modify any part of the system; it relates to changes in the basic control state of the computer. Unprivileged operation is the problem-solving mode of the IPU. In this mode, memory protection is in effect and all privileged operations are prohibited.

One bit in the Program Status Doubleword (PSD) is designated as the Privileged State bit. If the Privileged State bit is set, privileged instructions can be executed. If the Privileged State bit is reset, any attempt to execute a privileged instruction will cause a Privilege Violation trap.
Figure 2-7. Program Status Doubleword (PSD) Format
Table 2-2. PSD Mode (IPU)

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>PSD Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Status</td>
<td>Doubleword</td>
</tr>
<tr>
<td>Number of instructions</td>
<td>163</td>
</tr>
<tr>
<td>Integrity Features</td>
<td>Traps</td>
</tr>
<tr>
<td>Memory Addressing</td>
<td></td>
</tr>
<tr>
<td>Nonmapped</td>
<td></td>
</tr>
<tr>
<td>Nonextended</td>
<td>512 KB+</td>
</tr>
<tr>
<td>Extended</td>
<td>16 MB+</td>
</tr>
<tr>
<td>Mapped</td>
<td></td>
</tr>
<tr>
<td>Nonextended</td>
<td>512 KB per user</td>
</tr>
<tr>
<td>Extended</td>
<td>16 MB per user</td>
</tr>
<tr>
<td>+ No software support</td>
<td></td>
</tr>
</tbody>
</table>

The following IPU instructions are privileged:

1. All instructions that can modify the memory mapping registers.
2. All instructions that can place the machine in a state that requires CPU intervention to continue processing, such as Halt.
3. All instructions that modify Writable Control Storage.

Certain events can change the processor from the unprivileged to the privileged state by loading a new Program Doubleword. These are:

- A hardware trap caused by addressing nonpresent memory, executing undefined instruction, executing a privileged instruction by a nonprivileged program, or writing to protected memory.
- A hardware trap caused by a nonrecoverable condition such as an uncorrectable error on a memory read, or an arithmetic exception.
- The execution of the Call Monitor or Supervisor Call instruction by a user requesting monitor services.

As long as traps are set they are vectored to monitor routines for proper handling. The trap vectors and the monitor service routines are in protected memory. This insures that an unprivileged user has no way to become privileged or to alter protected state.

The execution of the Load Program Status Doubleword (LPSD) instruction can cause the system to change from the privileged to the unprivileged state.

The operator can depress the SYSTEM RESET pushbutton to initialize a 32/70 SERIES computer and IPU. SYSTEM RESET clears the eight general purpose registers, resets all memory protection, and sets the Privileged State bit.
1. 512-KB mode

2. 512-KB Extended mode

3. 512-KB Mapped mode

4. Mapped, Extended mode

**512-KB MODE**

The 512-KB addressing mode allows the IPU to access instructions or operands (bit, byte, halfword, word, or doubleword) in the first 512K bytes of memory directly without mapping, indexing, or address modification. A 19-bit address field is provided in memory referencing instructions for that purpose.

Bit addressing is accomplished by using the register (R) field in the instruction word to select a bit in the byte specified by the 19-bit address. Therefore, any bit in the first 512K bytes of memory can be directly manipulated by the Bit Manipulation instructions.

**512-KB EXTENDED MODE**

The 512-KB Extended mode provides the same capabilities as the 512-KB mode described above, and, in addition, it permits operand addressing only beyond the first 512K bytes of memory. The effective address can reference any bit, byte, halfword, word, or doubleword residing anywhere within 16 megabytes of physical memory.

**512-KB MAPPED MODE**

The 512-KB Mapped mode allows the IPU to access any instruction or operand (bit, byte, halfword, word, or doubleword) within a logical primary address space. This space consists of 512K bytes of logical memory map, distributed within 16 megabytes of physical memory.

The IPU allows multiple primary address spaces. A user can access instructions and operands within the logical primary address space in which his program resides. Physical blocks of memory can be common to many logical primary address spaces; thus, users in different spaces can share common blocks of memory.

**MAPPED EXTENDED MODE**

The Mapped Extended mode provides all the capabilities of the 512 KB Mapped mode, plus access to a logical extended operand address space. This space consists of 512K bytes of mapped memory beyond the logical primary address space and allows users additional memory space to store data (operands). Each logical extended operand address space can be 512K bytes long, dispersed anywhere within 16 megabytes of physical memory. The combination of logical primary address space and the logical extended operand address space supports programs up to one megabyte long. The executable code must lie within the logical primary address space, but operands can be in either the logical primary or extended operand address space.

**FUNCTIONAL DESCRIPTION**

The major elements of a typical SYSTEMS 32/70 SERIES Computer System with an IPU include: 32/70 CPU, the SelBUS, Real-Time Option Module, main memory, input/output system (not supported by the IPU), Serial Control Panel, optional High-Speed Floating Point, and Scientific Accelerator modules. (See Figure 2-8 for a system block diagram.) The performance gains of a CPU and IPU system over a CPU alone system is application dependent. The IPU allows the user to run two tasks simultaneously in the computer system.
Figure 2-8. System Block Diagram

- MEMORY BUS CONTROLLER
- SCIENTIFIC ACCELERATOR
- PCS OR WCS
- CONTROL PANEL W/ HEX DISP
- MOS MEM MOD 256 KB
- FIRST REAL TIME OPTION MODULE
- MAGNETIC TAPE CONTROLLER
- 75 IPS MTU (1)
- DISC CONTROLLER
- 80 MB DISC (1)
- USER FURNISHED DEVICE
- LINE PRINTER (600 LPM)
- CARD READER (300 CPM)
- ADDITIONAL REAL TIME OPTION MODULE
- 16 EXT INTERRUPTS
- 6 EXT INTERRUPTS
- INTERNAL PROCESSOR UNIT
- HSFPU
- REGIONAL PROCESSING UNIT
- RANDOM ACCESS MEMORY
- REGIONAL PROCESSING UNIT
- USER FURNISHED DEVICE
- LINE PRINTER (600 LPM)
- CARD READER (300 CPM)
- OPTION MDDULE
- 16 EXT INTERRUPTS
- 6 EXT INTERRUPTS
- 6 EXT INTERRUPTS
- 6 EXT INTERRUPTS
- 75 IPS MTU (1)
- 80 MB DISC (1)
- USER FURNISHED DEVICE
- LINE PRINTER (600 LPM)
- CARD READER (300 CPM)
- option equipment
- SELBUS (26.67 MB/SEC)
- MEMORY BUS
- 2048 KB MAX
- HIGH SPEED FLOATING POINT
- CENTRAL PROCESSOR UNIT
- MEMORY BUS
Central Processing Unit

The CPU in the system plays the dominant role in its relationship with the IPU. The CPU is responsible for all task scheduling I/O and system services as well as for the execution of its own scheduled tasks.

IPU Major Elements

A brief description of some major elements of the IPU are provided in the paragraphs that follow. They include: the data structure, a microprogrammable processor, the implementation logic, and the SelBUS interface. A simplified block diagram of the IPU is shown in Figure 2-9.

IPU Data Structure

The data structure contains the eight general purpose file registers and ten hardware registers organized around an Arithmetic Logic Unit (ALU). Key circuits in the data structure include the following:

1. Arithmetic Logic Unit (ALU)
2. A Multiplexer
3. B Multiplexer
4. Literal Multiplexer
5. General File Register
6. Memory Address Register
7. Program Counter Register
8. N Counter Register
9. Shift Register
10. Temporary Register/Data Output Register
11. Data Input Register
12. Instruction Register 0
13. Instruction Register 1

IPU Microprogrammable Processor

The Microprogrammable Processor of the IPU is on board C of the three IPU circuit boards. The logic circuit board, which contains the Microprogrammable Processor, is commonly referred to as the personality board.

The Microprogrammable Processor utilizes Read-Only Memory (ROM) integrated circuits which house the IPU's Elementary Operations (EO). The EOs, with the associated circuitry, control the IPU operations by testing, controlling, and directing the various functions to be performed. The format for the EOs (also referred to as micro-instruction) is shown in Figure 2-10.

Implementation Logic

The Implementation Logic includes the ALU Decode PROM, a Scale circuit, the Floating-Point Assist PROMs, and a Multiply Assist PROM, all of which serve to implement IPU functions.

SelBUS Interface

The SelBUS interface logic is implemented on all three of the IPU circuit boards and provides control and temporary storage for information being output to and input from the SelBUS. Since the SelBUS is the high-speed communication link between system modules external to the IPU, the SelBUS interface logic plays a vital role in IPU operation.
The High-Speed Floating-Point Unit (FPU) is an option that may be used with an IPU and CPU to increase the speed of floating-point arithmetic operations. The unit consists of two circuit boards which may be plugged in adjacent to the IPU. No alterations in the software are required.

If the High-Speed Floating-Point Unit (FPU) is installed, addition, subtraction, multiplication, and division of single-precision (32-bit) or double-precision (64-bit) operands can be executed much faster than the IPU's standard floating-point feature.

An operand in floating-point format has three parts: a sign bit, a fraction, and an exponent. The sign bit indicates whether the fraction is a positive or negative value. The fraction is a binary number with an assumed radix point immediately to the left of its most-significant bit. The exponent is a 7-bit binary power to which the base 16 is raised. The quantity that the floating-point number represents can be determined by multiplying the fraction by the number 16 raised to the power represented by the exponent.

Two operands of the same format and length are received by the FPU for each arithmetic operation. One operand is input from an IPU general purpose register (GPR), whereas the other operand is input from memory. The precise GPR and memory location are specified in the floating-point instruction. Upon completion of an operation, the result is returned to the CPU general purpose register.

It is recommended that any option which is added to the system be for both the IPU and CPU. This will allow the same runtime library to be utilized for the respective software programs.
<table>
<thead>
<tr>
<th>T</th>
<th>B</th>
<th>M</th>
<th>A</th>
<th>B</th>
<th>*</th>
<th>D</th>
<th>R</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST</td>
<td>SEQUENCE</td>
<td>CONTROL</td>
<td>EXTENDED</td>
<td>A</td>
<td>MUX</td>
<td>B</td>
<td>MUX</td>
<td>ALU</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

**HARDWARE FLOATING-PT. CONTROL**

<table>
<thead>
<tr>
<th>X</th>
<th>P</th>
<th>C</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREG32:36</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**X-ORDER**

<table>
<thead>
<tr>
<th>12-BIT BRANCH ADDRESS</th>
<th>HARDWARE FLOATING-POINT CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>33</td>
</tr>
</tbody>
</table>

**W-TEST**

<table>
<thead>
<tr>
<th>CREG</th>
</tr>
</thead>
</table>

**S-TEST**

<table>
<thead>
<tr>
<th>EXTENDED TEST</th>
<th>CROM</th>
</tr>
</thead>
</table>

**U-ORDER**

<table>
<thead>
<tr>
<th>CC'S</th>
<th>CREG</th>
</tr>
</thead>
</table>

**S-ORDER**

<table>
<thead>
<tr>
<th>FILE NUMBER</th>
<th>ROM PAGE</th>
<th>HDWR F-PT ORDER</th>
</tr>
</thead>
</table>

**NOTES:**

1. BITS 0-47 OF EACH MICROINSTRUCTION ARE IN THE IPU'S CONTROL ROM.

2. PORTIONS OF THE FORMAT DESIGNATED FOR HARDWARE FLOATING-POINT APPLY TO THE OPTIONAL HIGH-SPEED FLOATING-POINT UNIT (FPU).

3. BITS 48-63 ARE PHYSICALLY PART OF A CONTROL ROM IN THE OPTIONAL HIGH-SPEED FPU.

**Figure 2-10. Microinstruction Format**
Figure 2-11 illustrates the major functional elements of the FPU, the general routing of operands, and the relationship between the FPU, the IPU, and the SelBUS.

The optional Model 2343 Scientific Accelerator (with PROM control store PCS), or Model 2347 with Writable Control (WCS), may be used in the IPU system. The Scientific Accelerator provides fully-integrated hardware, software, and firmware to improve user program execution speeds.

It is recommended that any option which is added to the system be for both the IPU and CPU. This will allow the same runtime library to be utilized for the respective software programs.

Writable Control Storage is an option which may be used with the IPU to expand the instruction set, to enhance the performance of user programs, or to tailor the computer to specific user needs.

Up to 4,096 64-bit words of Writable Control Storage (WCS) can be added to an IPU in increments of 2,048 64-bit words. Each increment plugs into the SelBUS for power and clock. However, communication with the IPU is independent of SelBUS operation.

The block diagram in Figure 2-12 shows two optional WCS units as they could be implemented in conjunction with an IPU and the optional High-Speed Floating-Point Unit.

For synchronization and communication between the IPU and CPU, six new traps are dedicated in low memory. These traps are listed in Table 2-3. The trap vector location 2E0 is used by the CPU when the IPU has executed the SIPU (X'000A') instruction. The CPU handles this trap in the same manner as any other CPU trap. The trap vectors found at locations 2E4, 2E8, 2EC, 2F0, and 2F4 are traps used by the IPU during IPU processing. A brief description of the communications between the IPU and CPU follows later in this section and involves primarily the use of new traps.

The IPU uses the Program Status Doubleword Mode of operation to run the Mapped Programming Executive, MPX-32. This mode identifies the firmware routing and method of handling traps.

In the IPU mode, the firmware will not execute control instruction BRI, any I/O instructions, nor interrupt control instructions except UEI.

Trap Context Switching in the IPU is accomplished through the use of the Program Status Doubleword Mode using the Trap Context Block (TCB) format. Trap context switching by the IPU is functionally identical to the CPU, except that the trap entry by the IPU is not associated with a service interrupt.

The Trap Context Block (TCB) format type (see Figure 2-13) is used for the PSD mode traps. Words one through four of the TCB contain the IPU Ending and Starting PSDs. Word five of the TCB contains the IPU Hardware Status Word. This status word is assembled by firmware at the time the trap occurs, and is stored in the TCB. The IPU Hardware Status Word is defined later in this section. Word 6 of the TCB is not used.
Figure 2-11. Optional high-speed Floating-point Unit
Figure 2-12. Functional Interrelationship of the IPU, WCS, and High-Speed Floating-Point Unit
Table 2-3. CPU/IPU Communication Traps

<table>
<thead>
<tr>
<th>Trap Relative Priority</th>
<th>Trap Vector Location TVL</th>
<th>Description</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>78</td>
<td>2E0</td>
<td>Ending of IPU Processing</td>
<td>CPU</td>
</tr>
<tr>
<td>79</td>
<td>2E4</td>
<td>Start IPU Processing</td>
<td>IPU</td>
</tr>
<tr>
<td>7A</td>
<td>2E8</td>
<td>Supervisor Call</td>
<td>IPU</td>
</tr>
<tr>
<td>7B</td>
<td>2EC</td>
<td>Error Trap</td>
<td>IPU</td>
</tr>
<tr>
<td>7C</td>
<td>2F0</td>
<td>Call Monitor</td>
<td>IPU</td>
</tr>
<tr>
<td>7D</td>
<td>2F4</td>
<td>Stop IPU Processing</td>
<td>IPU</td>
</tr>
</tbody>
</table>

**IPU STATUS WORD**

The IPU status word is a 32-bit word that is used by IPU firmware to track trap error processing and internal operating modes. The status word is available to software in either of two methods as follows:

1. The Read Status (RDSTS) instruction (when executed by the IPU) causes the status word to be loaded into the general purpose register specified by the instruction.

2. Automatically, upon occurrence of an error trap which causes the status word to be stored in the fifth word of the trap context block.

The status word can be divided into a 24-bit field and an 8-bit field. The 24-bit field is used for error flag reporting and is cleared to zeros after the status word has been reported to software. The 8-bit field of the status word is used for IPU mode control and will always reflect the current operating mode of the IPU. Table 2-4 lists the bits of the status word and their definitions.

The following discussion provides information pertaining to the CPU and IPU interface operation. This discussion is centered primarily around the use of the six new traps, which are used to control the synchronization and communication between the CPU and IPU. The basic interface operational flow between the CPU and IPU is shown in Figure 2-14.

**START IPU TRAP (VECTOR ADDRESS 2E4)**

To start IPU processing, the CPU stores the new Program Status Double-word (PSD) into words 3 and 4 of the Start IPU trap context block which was pointed to from the address contained in the Start IPU trap vector location 2E4. The CPU then executes the SIPU X'000A' instruction which sends a start signal to the IPU and informs the IPU that a new PSD is available for execution. The IPU then fetches the Start IPU trap Context Block pointer at the 2E4 trap location, stores the old PSD into words 1 and 2 of the Start IPU Trap Context block and the IPU Status into word 5. The IPU then reads the new PSD words 3 and 4 from the context block and begins to execute the instructions in memory as directed by the new PSD.
RESTART IPU

If the Signal IPU instruction (SIPU) is issued by the CPU to an active IPU, the second SIPU will cause the following events in the IPU to occur:

1. The IPU will terminate present active execution, and vector to the start IPU Trap Vector Address (TVA) 2E4. The old PSD is stored into Words 1 and 2 of the Context Block as was pointed by the contents of the TVA.

2. The old IPU status word is stored into the context block and the new PSD is used to begin execution of another group of 32/70 macro-assembler instructions.

The End of IPU trap is not generated until the IPU has completed execution as directed by the interrupting SIPU instruction.

IPU ERROR CONDITION TRAP (VECTOR ADDRESS 2EC)

The vector address found at memory location 2EC points to the TCB which is used upon the occurrence of an error condition within the IPU. The error conditions include non-present memory, undefined instruction, parity error, arithmetic exception, and privilege violation. The undefined instruction error is caused by the execution of any I/O instruction (e.g., CD, TD), any interrupt instruction (BRI, AI), or any instruction not defined in the PSD mode 32/75 instruction set.
Figure 2-14. CPU/IPU Interface Operational Flow
Table 2-4. IPU Status Word Bit Definitions

<table>
<thead>
<tr>
<th>Bit</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>=0, Class 0, 1, 2, or E Error*</td>
</tr>
<tr>
<td></td>
<td>=1, Class F (Extended I/O) Error*</td>
</tr>
<tr>
<td>1</td>
<td>=0, I/O Processing Error*</td>
</tr>
<tr>
<td></td>
<td>=1, Interrupt Processing Error*</td>
</tr>
<tr>
<td>2</td>
<td>Final Bus Transfer Error</td>
</tr>
<tr>
<td>3</td>
<td>Bus No Transfer Error</td>
</tr>
<tr>
<td>4</td>
<td>I/O Channel Busy or Busy Status Bit Error*</td>
</tr>
<tr>
<td>5</td>
<td>Ready Timeout Error*</td>
</tr>
<tr>
<td>6</td>
<td>I/O DRT Timeout Error*</td>
</tr>
<tr>
<td>7</td>
<td>Retry Count Exhausted Error*</td>
</tr>
<tr>
<td>8</td>
<td>Operand Fetch Parity Error</td>
</tr>
<tr>
<td>9</td>
<td>Instruction Fetch Parity Error</td>
</tr>
<tr>
<td>10</td>
<td>Operand Nonpresent Error</td>
</tr>
<tr>
<td>11</td>
<td>Instruction Nonpresent Error</td>
</tr>
<tr>
<td>12</td>
<td>Undefined PSD Mode Instruction Error</td>
</tr>
<tr>
<td>13</td>
<td>Memory Fetch DRT Timeout Error</td>
</tr>
<tr>
<td>14</td>
<td>Reset Channel Error*</td>
</tr>
<tr>
<td>15</td>
<td>Channel WCS not Enabled Error</td>
</tr>
<tr>
<td>16</td>
<td>Map Register Address Overflow</td>
</tr>
<tr>
<td></td>
<td>(Map Context Switch)</td>
</tr>
<tr>
<td>17</td>
<td>Unexplained Memory Error</td>
</tr>
<tr>
<td>18</td>
<td>BRI I/O Error*</td>
</tr>
<tr>
<td>19</td>
<td>Undefined Instruction PSW Mode Only*</td>
</tr>
<tr>
<td>20</td>
<td>Map Invalid Access or Map Mode Restrict Register</td>
</tr>
<tr>
<td>21</td>
<td>IPU Privileged Violation</td>
</tr>
<tr>
<td>22</td>
<td>Not Used</td>
</tr>
<tr>
<td>23</td>
<td>IPU Arithmetic Exception</td>
</tr>
<tr>
<td>24</td>
<td>Enable Arithmetic Exception Trap</td>
</tr>
<tr>
<td>25</td>
<td>Disable PSD Mode Traps</td>
</tr>
<tr>
<td>26</td>
<td>Block Mode is Active</td>
</tr>
<tr>
<td>27</td>
<td>IPU Status</td>
</tr>
<tr>
<td>28</td>
<td>Not Used</td>
</tr>
<tr>
<td>29</td>
<td>CPU ELSA Mode*</td>
</tr>
<tr>
<td>30</td>
<td>Not Used</td>
</tr>
<tr>
<td>31</td>
<td>=1, IPU Mode PSD</td>
</tr>
</tbody>
</table>

*Not Applicable to IPU.
The privilege violation error is generated by the IPU attempting to execute an instruction which is defined as privileged but does not have the privileged bit set in the PSD. HALT and WAIT in the IPU must be privileged.

The error status is reflected in the IPU status word as stored into the fifth word of the IPU error TCB (vector location 2EC). The PSD at the time the error occurs is stored into words 1 and 2 of that TCB. The next executed instruction is dictated by the new PSD found in words 3 and 4 of the error TCB.

When the IPU executes a Call Monitor (CALM) instruction, control is transferred to the IPU call monitor trap located at memory address 2FO. The execution which follows the call monitor instruction, as well as any other trap within the IPU, is directed by the contents of the context block related to that specific trap. Execution is directed to the code as defined by the new PSD within the IPU CALM Trap Context Block.

When the IPU executes a Supervisor Call (SVC) instruction, control is transferred to the SVC trap. The address of the context block for the IPU service of a SVC instruction is located at trap address 2EB. This address is the beginning of the 16-entry secondary vector address table. Bits 16 through 19 of the SVC instruction direct the IPU to one of the 16 secondary vector addresses. The secondary vector address selected points the IPU to a TCB for that SVC.

Once the IPU has the address of the TCB, trap processing is handled as a normal trap. The IPU stores the present PSD into words 1 and 2 of the TCB and the status into word 5. Then the IPU uses the new PSD from words 3 and 4 to continue execution.

To stop the IPU processing, the CPU stores a new PSD in words 3 and 4 of the STOP IPU Trap Context Block (TCB) which is pointed to by the address contained in the stop IPU trap vector location 2F4. The STOP IPU TCB is used when the IPU executes an SIPU (X'000A') instruction which is imbedded in the IPU software code. The IPU stores the old PSD into words 1 and 2 of the context block and the IPU Status into word 5 of the context block. The IPU then traps the CPU at location 2EO which indicates that the IPU execution of the SIPU instruction has taken place. The IPU then fetches the new PSD from words 3 and 4 of the context block which can point to a privileged HALT or WAIT instruction to stop the IPU.

The new PSD in the STOP IPU context block may direct the IPU to execute code other than a HALT or WAIT instruction. This utilization of the STOP Trap allows the IPU to signal the CPU at milestones without stopping IPU execution. In either use of this stop IPU trap, the present PSD is stored into words 1 and 2 of TCB and the present IPU status into word 5. The IPU done signal is sent to the CPU after storage of the present PSD and IPU status word and before vectoring to the new PSD address.

The End IPU Processing trap address 2EO is used by the CPU when the IPU generates the IPU done signal. The CPU handles this trap in the same manner as any other CPU Trap, except that this trap can be blocked at the CPU by setting the block mode.

All information as presented in Section IV for the Memory Management is valid for the IPU.
The Internal Processing Unit does not perform I/O operations. All I/O operations are performed by the CPU in the system.

Except for the Scratchpad locations related to I/O and interrupts, the IPU utilizes the internal scratchpad in the same manner as the CPU uses its internal scratchpad.

The scratchpad locations loaded by the IPL are not used by the IPU. Thus, no loading procedure is necessary. The IPU can execute the TRSC and TSCR instructions if the user deems it necessary to load or read scratchpad locations.

The Internal Processing Unit is initialized by a system reset and remains quiescent until a Signal IPU (SIPU) instruction occurs.

The IPU does not perform an IPL. This procedure is controlled by the CPU and the I/O device. Refer to Section VIII for CPU-IPL operation.

The Power Fail Safe feature as implemented in the CPU is not applicable to the IPU operation. The saving of the IPU scratchpad information is not necessary by the IPU since the CPU must re-initialize any IPU operation when the CPU is restarted.
SECTION III
TRAPS AND INTERRUPTS

INTRODUCTION
Traps and interrupts report asynchronous or synchronous events to the software. Traps are error conditions that are generated internally and interrupts are requests that are generated externally. The events that caused the trap or interrupt can be generated asynchronously by hardware or synchronously scheduled by software when an interrupt control instruction is executed. The trap or interrupt causes a transfer of control to unique vector locations in main memory (see Table 3-1).

TRAPS
The traps for the PSW mode (in order of priority) are:
1. Power Fail
2. Memory Parity
3. Nonpresent Memory
4. Undefined Instruction
5. Privileged Violation
6. System Override

Six additional traps are present in the PSD mode. They are:
1. Supervisor Call Trap (software generated)
2. Machine Check Trap
3. System Check Trap
4. MAP Fault Trap
5. Block Mode Timeout (Watchdog) Trap
6. Arithmetic Exception Trap
7. End of IPU processing

INTERRUPTS
Interrupts consist of the following:
1. Any external event scheduled through the Real-Time Option Module (RTOM)
2. Input/Output (I/O) termination interrupts
3. Software request interrupt control instruction

OPERATING MODES
The 32/70 Series CPU is capable of operating in two modes: the PSW mode and the PSD mode. The two modes identify the firmware routing required to operate with a FSW, thereby allowing existing 32/55 software to operate on a 32/70 Series CPU without modifications. The PSD mode is the default at system reset and remains in effect until a Set CPU Mode macro instruction is executed or an Initial Program Load (IPL) sequence is set up to force the CPU into PSW mode of operation.
### Table 3-1. PSW/PSD Mode Relative Trap/Interrupt Priorities

<table>
<thead>
<tr>
<th>INTERRUPT AND TRAP RELATIVE PRIORITY</th>
<th>INTERRUPT LOGICAL PRIORITY</th>
<th>INTERRUPT VECTOR LOCATION (IVL)</th>
<th>TCW ADDRESS **</th>
<th>IOCD ADDRESS **</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td>0F4</td>
<td></td>
<td></td>
<td>Power Fail Safe Trap</td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>0FC</td>
<td></td>
<td></td>
<td>System Override Trap (Not used)</td>
</tr>
<tr>
<td>02</td>
<td></td>
<td>0E8*</td>
<td></td>
<td></td>
<td>Memory Parity Trap</td>
</tr>
<tr>
<td>03</td>
<td></td>
<td>190</td>
<td></td>
<td></td>
<td>Nonpresent Memory Trap</td>
</tr>
<tr>
<td>04</td>
<td></td>
<td>194</td>
<td></td>
<td></td>
<td>Undefined Instruction Trap</td>
</tr>
<tr>
<td>05</td>
<td></td>
<td>198</td>
<td></td>
<td></td>
<td>Privilege Violation Trap</td>
</tr>
<tr>
<td>06</td>
<td></td>
<td>180</td>
<td></td>
<td></td>
<td>Supervisor Call Trap</td>
</tr>
<tr>
<td>07</td>
<td></td>
<td>184</td>
<td></td>
<td></td>
<td>Machine Check Trap</td>
</tr>
<tr>
<td>08</td>
<td></td>
<td>188</td>
<td></td>
<td></td>
<td>System Check Trap</td>
</tr>
<tr>
<td>09</td>
<td></td>
<td>18C</td>
<td></td>
<td></td>
<td>MAP Fault Trap</td>
</tr>
<tr>
<td>0A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not Used</td>
</tr>
<tr>
<td>0B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not Used</td>
</tr>
<tr>
<td>0C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not Used</td>
</tr>
<tr>
<td>0D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not Used</td>
</tr>
<tr>
<td>0E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Block Mode Timeout (Watchdog) Trap</td>
</tr>
<tr>
<td>0F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Arithmetic Exception Trap</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>0F0</td>
<td></td>
<td></td>
<td>Power Fail Safe Interrupt</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>0F8</td>
<td></td>
<td></td>
<td>System Override Interrupt</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>0E8*</td>
<td></td>
<td></td>
<td>Memory Parity Trap</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>1A4*</td>
<td></td>
<td></td>
<td>Nonpresent Memory Trap</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>194*</td>
<td></td>
<td></td>
<td>Undefined Instruction Trap</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Privilege Violation Trap</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Call Monitor Interrupt</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Real-Time Clock Interrupt</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Arithmetic Exception Interrupt</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>THRU</td>
<td></td>
<td>THRU</td>
<td></td>
<td></td>
<td>THRU</td>
</tr>
<tr>
<td>77</td>
<td></td>
<td>2DC</td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
</tbody>
</table>

* Vector Locations Shared With Traps
** For Nonextended I/O Devices
*** PSW Function-Now External/Software Interrupts-For PSD Mode
**** IPU Related Traps
All Interrupts Are Externally Generated
Table 3-1. PSW/PSD Mode Relative Trap/Interrupt Priorities (Cont'd)

<table>
<thead>
<tr>
<th>INTERRUPT AND TRAP RELATIVE PRIORITY</th>
<th>INTERRUPT LOGICAL PRIORITY</th>
<th>INTERRUPT VECTOR LOCATION (IVL)</th>
<th>TCW ADDRESS **</th>
<th>IOC ADDRESS **</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>78</td>
<td></td>
<td>2EO****</td>
<td></td>
<td></td>
<td>Ending of IPU Processing Trap (Used by CPU)</td>
</tr>
<tr>
<td>79</td>
<td></td>
<td>2E4****</td>
<td></td>
<td></td>
<td>Start IPU Processing Trap (Used by IPU)</td>
</tr>
<tr>
<td>7A</td>
<td></td>
<td>2E8****</td>
<td></td>
<td></td>
<td>Supervisor Call Trap (Used by IPU)</td>
</tr>
<tr>
<td>7B</td>
<td></td>
<td>2EC****</td>
<td></td>
<td></td>
<td>Error Trap (Used by IPU)</td>
</tr>
<tr>
<td>7C</td>
<td></td>
<td>2F0****</td>
<td></td>
<td></td>
<td>Call Monitor Trap (Used by IPU)</td>
</tr>
<tr>
<td>7D</td>
<td>7D</td>
<td>2F4****</td>
<td></td>
<td></td>
<td>Stop IPU Processing Trap (Used by IPU)</td>
</tr>
<tr>
<td>7E</td>
<td>7E</td>
<td>2F8</td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>7F</td>
<td>7F</td>
<td>2FC</td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
</tbody>
</table>

** For Nonextended I/O Devices
**** IPU Related Traps (See Section II)

All Interrupts Are Externally Generated
PSW MODE

The PSW mode identifies traps and interrupts on a prioritized, scheduled basis. No distinction is made between traps and interrupts, and both are scheduled by some mechanism external to the CPU (i.e., IOM or RTOM). The trap conditions that are created internally within the CPU are scheduled by the firmware on an RTOM board if the following requirements are met:

1. Trap level is enabled.
2. Trap level is not active.
3. Any other higher priority level is not active or requesting.

If any of the above requirements are not met, the firmware will reset the condition that caused the trap and continue to the next sequential instruction as if the trap never occurred.

Traps and interrupts in the PSW mode require the participation of three component levels in order to function properly. The three component levels are the IOM or RTOM, the CPU, and the software.

The IOM or RTOM schedules a hardware- or software-initiated interrupt service request. When the requesting level becomes the highest contending level, the CPU acknowledges the interrupt request. In order to enqueue the associated software processing, the IOM or RTOM advances from requesting to active, blocking interrupt requests from lower priority levels. When the software interrupt handler completes its processing, the software dequeues itself by executing a Deactivate Interrupt (DAI) or Branch and Reset Interrupt (BRI) instruction which allows the currently active level and all other lower priority levels to resume requesting for interrupts. This operating mode is also referred to as Block with Activate. In summary, the six steps shown below are required to enqueue or dequeue an interrupt process:

1. The IOM, RPU, or RTOM requests an interrupt.
2. The CPU acknowledges the interrupt.
3. The IOM or RTOM goes active, blocking lower priority interrupts.
4. The software handler is given control. (First instruction is noninterruptible)
5. The software executes a Deactivate Interrupt (DAI) or Branch and Reset Interrupt (BRI).
6. The IOM or RTOM deactivates, allowing lower priority levels to resume requesting.
Two types of software trap and interrupt queueing methods exist in the PSD mode. The first method is identical to the queueing described as the PSW mode, where the requesting level advances to active state, blocking all lower priority levels to insure that software is not interruptible by its level or any lower priority levels during the interrupt processing. This method applies to all classes of I/O interrupts and external (RTOM) interrupts.

The second method applies to traps, I/O interrupts and external interrupts. The enqueueing of the software interrupt and trap handlers does not rely on the active state of the applicable channel or RTOM to prevent interrupts or traps for the specific or lower priority levels. The enqueueing function blocks externally generated interrupt requests (channel or RTOM) from being sensed by the CPU firmware. Software must now explicitly dequeue its process with an Unblock External Interrupts (UEI) or a Load PSD (LPSD) macro instruction. The general sequence is:

1. The IOM, RPU, or RTOM requests an I/O interrupt.

2. When the requesting level becomes the highest contending level, the CPU acknowledges the interrupt request and blocks all interrupts until the UNBLOCK command is received (if bits 48 and 49 of the PSD are 0 and 1, respectively).

3. The channel does not go active and is now free to continue I/O related processing.

4. The software is given control with all interrupts blocked.

5. When the software interrupt handler completes its enqueued processing, it will execute an Unblock External Interrupt (UEI) or a Load Program Status Doubleword (LPSD) macro instruction which will allow externally generated interrupts to be sensed by the CPU firmware. This operating mode is also referred to as Block without Activate.
Each trap or interrupt that may occur in the PSD mode has an associated Interrupt Vector Location (IVL) and an Interrupt Context Block (ICB). The IVL contains a 24-bit real address that points to the starting memory address of the ICB. Table 3-1 includes a list of the memory locations dedicated for IVLs.

Generally speaking, an ICB consists of six consecutive memory words. However, for some types of ICBs only four or five words are required. The four different ICB formats are listed as follows:

1. External and Non-Class F I/O Format
2. Trap Format
3. Class F I/O Format
4. Supervisor Call Format

Figures 3-1 through 3-4 illustrate the four ICB formats.

The first four words of all ICB formats are identical in that they contain the old PSD followed by the new PSD.

The old PSD is stored in the ICB whenever a trap or interrupt occurs and is acknowledged. The old PSD locations provide storage for hardware and software CPU context information current at the time a particular trap or interrupt occurs. Normally, when the software interrupt processing is completed, a BRI, LPSD or LPSDCM instruction will be used to restore the old PSD context information.

The new PSD information must be loaded in the ICB by software before a trap or interrupt occurs. The new PSD must contain the necessary information to set up the hardware and software in the appropriate context for servicing the interrupt.

The External and Non-Class F ICB format type (see Figure 3-1) is used with all RTOM interrupts and all CD and TD I/O interrupts. RTOM interrupts include: Console Interrupt (Panel Attention), Call Monitor Interrupt, and Real-Time Clock-Interrupt.

Words 1 through 4 contain the old and new PSDs.

Words 5 and 6 of this ICB format type are optional and may be omitted.

The Trap ICB format type (see Figure 3-2) is used for PSD mode traps.

Words 1 through 4 of the Trap ICB contain the old and new PSDs.

Word 5 of the Trap ICB contains the CPU hardware status word. This is stored in the ICB at the time a trap occurs. The CPU status word may provide additional descriptor bits for defining the error condition. For a detailed description of the CPU status word, refer to the 32/70 Series Technical Manual.

Word 6 of the Trap ICB is optional.
Figure 3-1. Interrupt Context Block Format - External Interrupts and Non-Class F I/O Interrupts

Figure 3-2. Trap Context Block Format
The Class F I/O format type (see Figure 3-3) requires the use of all six ICB words.

Words 1 through 4 contain the old and new PSDs.

Word 5 of the Class F I/O ICB provides the Input/Output Command List (IOCL) address for the associated Class F I/O channel. This word must be set up in the ICB by software prior to the execution of either a Start I/O or Write Channel WCS instruction. The ICL address is transmitted to the I/O channel by the CPU during the Start I/O or Write Channel WCS SelBUS sequences. The ICL address must be in a 24-bit real address format.

Word 6 of the Class F I/O ICB contains the 24-bit real address of the channel status word. Whenever the channel reports status to the CPU (and software), the channel stores the channel status word in memory. The CPU then stores the memory address of the channel status word into word 6 of the ICB.

The channel may report status when any one of the following events occur:

1. An interrupt is acknowledged (a hardware event).
2. A Start I/O instruction is executed.
3. A Test I/O instruction is executed.
4. A Halt I/O instruction is executed.

When status is stored during a Start I/O, Test I/O, or Halt I/O instruction, the channel rejects the instruction, and the CPU Condition Codes are set to reflect the Status Stored condition. Under the Status Stored condition, the channel clears its status pending flags, as well as any interrupt pending flags that are relative to the status just reported.

The Supervisor Call (SVC) instruction is provided with up to 16 different ICBs. These multiple ICBs are provided to reduce the amount of time required for a user program to request service from the operating system program. The address of a specific ICB is obtained by adding a 4-bit word index value from bits 16-19 of the SVC instruction to the 24-bit address that is in the SVC Interrupt Vector Location (IVL). The sum of these values provides a 24-bit real address of a Secondary Vector Location. The contents of the Secondary Vector Location is the 24-bit real address of the appropriate Supervisor Call ICB. Reference Figure 3-4.

Words 1 through 4 of the Supervisor Call ICB contain the Old and New PSD.

Word 5 of the ICB is available for use by the software SVC Trap processor as an index (call number) for the requested operating system service. Bits 20 through 31 of the SVC instruction are used by the CPU to format word 5 of the Supervisor Call ICB.

Word 6 of the Supervisor Call ICB is optional.
Figure 3-3. Interrupt Context Block Format - Class F I/O Interrupts

Figure 3-4. Supervisor Call (SVC) Trap Context Block Format
The eight PSD interrupt and trap related macro instructions are:

1. Block External Interrupts (BEI)
2. Unblock External Interrupts (UEI)
3. Load Program Status Doubleword (LPSD)
4. Load Program Status Doubleword Change Map (LPSDCM)
5. Set CPU Mode (SETCPU)
6. Supervisor Call (SVC)
7. Enable Arithmetic Exception Trap (EAE)
8. Disable Arithmetic Exception Trap (DAE)

All of the above macro instructions, except SVC, can be executed only in the privileged state and BEI, UEI, LPSD, EAE, DAE, and SVC will be valid instructions only if the CPU mode is set to other than the PSW mode. If the PSW mode is set, an undefined instruction trap will occur.

In the PSD mode, traps cannot be inhibited by the Blocked mode or by the activation of any high level interrupt.

A list of the traps, interrupts, and vector addresses is presented in Table 3-1.

The 32/70 Series CPU provides for automatic trap halts in both the PSW and PSD modes of operation.

A PSW mode trap halt* can occur under any of the following conditions:

1. A Memory Parity Error or Nonpresent Memory Error, while handling the dedicated memory locations associated with an interrupt level. This error must occur during the firmware interrupt Store, Place, and Branch sequence or the Branch and Reset Interrupt (BRI) sequence.

2. An I/O communication protocol violation during the interrupt or BRI communication sequence.

*Implementation of the PSW trap halt is the same as described in the PSD trap halt discussion.

A PSD mode trap halt only occurs if the software has not enabled the PSD mode traps by the SETCPU Enable Trap instruction. The PSD mode traps that arm the Trap Halt logic are:

1. Memory Parity Error
2. Nonpresent Memory
3. Undefined Instruction
4. Privileged Violation Trap
5. Machine Check Trap
6. System Check Trap
7. MAP Fault Trap
The PSD mode traps that do not arm the Trap Halt logic are:

1. Supervisor Call Trap
2. Arithmetic Exception Trap
3. Call Monitor Interrupt Trap

**MACHINE CHECK TRAP**

A Machine Check trap is a hardware/firmware failure that has occurred during an interrupt or context switch. These failures include Memory Parity error, Nonpresent Memory error, or I/O and Interrupt SelBUS protocol violations. The specific type of error that causes the trap is described by the CPU Status Word that is stored in the interrupt (trap) context block.

**SYSTEM CHECK TRAP**

A System Check trap is primarily a software failure that attempted to force the CPU into an illogical sequence. The specific type of error that caused the trap is described by the CPU status word stored in the interrupt (trap) context block.

**BLOCK MODE TIME-OUT TRAP**

The Block Mode Time-Out (watchdog) trap occurs under the following conditions:

1. If a Wait instruction is executed with interrupts blocked.
2. If the Block Mode Time-Out trap has been enabled by a SETCPU instruction and more than 128 instructions have been executed with interrupts blocked.

**PSD TRAP HALT IMPLEMENTATION**

The detection of a PSD trap condition causes the following events to occur if traps are not enabled:

1. The CPU is halted.
2. The Interrupt Active light on the Serial Control Panel is turned on.
3. The PC portion of the PSW (PSD1) contains the dedicated memory address for the trap causing the halt.
4. The CPU halfword indicator (PSD1, bit 5) may or may not be on.
5. Starting at memory location 530₁₆, the following error information is stored:

<table>
<thead>
<tr>
<th>Location</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>530</td>
<td>Error PSW (PSD1)</td>
</tr>
<tr>
<td>534</td>
<td>Error PSD2 (PSD mode only)</td>
</tr>
<tr>
<td>538</td>
<td>CPU Status Word</td>
</tr>
<tr>
<td>53C</td>
<td>R(RDEV) Device Table Entry</td>
</tr>
<tr>
<td>540</td>
<td>R(INTRTAB) Device Interrupt Entry</td>
</tr>
</tbody>
</table>
SECTION IV
MEMORY MANAGEMENT

INTRODUCTION
This section provides information that includes the rules for configuring MOS and core memory, as well as memory management programming methods and formats. For a functional description of the major elements in a 32/70 Series Memory Subsystem, the reader should refer to Section I of this manual.

OVERVIEW
All memory subsystems in the 32/70 Series are configured with a Memory Bus Controller (MBC) that communicates with the SelBUS and controls the memory bus to which the memory modules are attached. The MBC and CPU provide for byte, halfword, or word accesses of memory. The Memory Bus Controller is capable of managing up to 16 overlapped memory modules which operate asynchronously on their bus. Computer memory requests can be initiated every 150 nanoseconds due to the overlapped memory design. All modules under one Memory Bus Controller have the same cycle and access time; however, other MBCs may manage up to 16 fully overlapped modules.

MOS AND CORE MEMORY
Depending on the model, 32/70 Series systems can have either core or MOS memory. Core memory systems are organized into 36-bit words: 32 data bits plus 4 parity bits. MOS memory systems are organized with 39-bit words: 32 data bits plus 7 error checking correcting (ECC) bits. The MOS memory module corrects single-bit errors and has the capability of detecting and reporting double-bit errors.

Core memory packages include the following components:
1. Core memory modules
2. Memory chassis
3. Power supply
4. Memory Bus Controller

Core memory for 32/70 Series computers is available in the following forms:
1. The basic 32,768-byte core memory modules with a full memory cycle time of 600 nanoseconds
2. 65,536-byte core memory packages of 600-nanosecond memory
3. 131,072-byte core memory packages of 600-nanosecond memory
4. 65,536-byte core memory modules with a full memory cycle time of 900 nanoseconds
5. 131,072 core memory packages of 900-nanosecond memory
MOS memory packages include the following components:

1. 128 KB or 256 KB 900-nanosecond MOS memory modules(s)
2. Memory chassis
3. Power supply
4. Refresh board
5. Memory Bus Controller (MBC)

The 32/70 Series computers will support both 600- and 900-nanosecond core memory modules if they are not intermixed with one memory interface. Since the individual memory modules connected to the memory interface have a full cycle time of 600 or 900 nanoseconds, and the SelBUS operates synchronously with full 32-bit word transfers occurring every 150 nanoseconds, the memory chassis handles the following combinations of overlapped memory operations:

1. a. Four memory write operations (26.67M bytes/second) (for 600 ns memory)
   b. Six memory write operations (26.67M bytes/second) (for 900 ns memory)
2. a. One memory read and two memory write operations (19.99M bytes/second) (for 600 ns memory)
   b. One memory read and two memory write operations (22.22M bytes/second) (for 900 ns memory)
3. a. Two memory read operations (13.33M bytes/second) (for 600 ns memory)
   b. Three memory read operations (10.00M bytes/second) (for 900 ns memory)

MOS and core memory may be mixed on 32/70 Series systems. However, it should be done only in accordance with the rules listed below:

1. Mixed memory can be accomplished on 32/70 Series systems only.
2. The higher speed memory must be the low order address space.
3. Separate MBCs, chassis, and power supplies must be used for the different memory types.
4. The core memory should occupy the low order address space.
5. The total amount of core memory in the low order address range must be equal to or a multiple of the MOS memory module size.
An amplification of the preceding rules is provided in the paragraphs that follow.

Mixing MOS and core memory should not be attempted on systems other than the 32/70 Series. For example, the 32/35 and 32/55 cannot support MOS memory. The 32/30 and 32/57 cannot have mixed memory because they use a split backplane.

Separate MBCs, chassis, and power supplies are necessary because MOS and core memory units have different requirements in this regard. When adding core memory to a Model 32/77 processor, it is necessary to add Model 2332 Memory Carriage for 900 ns core memory. The Memory Carriage includes the chassis, power supply, and MBC required to support the core memory. This MBC will not support MOS memory. To add MOS memory to a Model 32/75 processor, a Model 2375 or 2380 Memory Package is required and provides the chassis, power supply, MBC, and memory.

Core memory should occupy the low order address space. This is to ensure that register save areas are in nonvolatile memory locations. If a customer is unconcerned about the state of the processor at the time of a power failure, then the core memory could be high address locations.

Assuming the core memory is in the low order address space, it is necessary to protect the memory from unwanted discontiguous memory locations (holes). The amount of memory on the first MBC will be dictated by the incremental granularity of the MOS memory modules on successive MBCs. Since the smallest granularity of the MOS memory boards is 32 KW, there would have to be at least 32 KW of core on the first MBC. If the MOS memory module used contained 64 KW, the amount of core on the first MBC would have to be 64 KW. After the first MOS memory board size is established, any additional boards must be of the same size. An example would be a Model 32/75 CPU with four 8 KW, 600 ns core memory modules (Model 2152). If a customer wished to add the 64 KW MOS Memory Package (Model 2380) to the CPU, a prerequisite would be to add four additional 8 KW, 600 ns core memory modules (2152) to the first MBC. This establishes the memory on the first MBC (64 KW) and is equal to the granularity of the MOS Memory Package of 64 KW. Additional 64 KW memory modules (Model 2381) can then be added to the MOS Memory Package.

Bits 9-31 have the same format in every memory reference instruction whether the effective address is used for storage or retrieval of an operand, as an indirect address operand, or to alter program flow. The Memory Reference instruction format is shown below:

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>F</th>
<th>WORD ADDRESS</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bits 9 and 10 specify the general purpose register (GPR) to be used as an index register, bit 11 is the indirect bit, and bits 12-31 define the word address and data type. The effective address of the instruction depends on the values of I, X, and bits 12-31. If I and X are both zero, bits 12-31 address the data type defined by bits 13-29.

---

**F- AND C-BITS**

The format of the F- and C-bits have been selected so that any selected data type (byte, 16-bit halfword, 32-bit word, or 64-bit doubleword) can be convieniently indexed by that data type. The possible combinations of F- and C-bits are as follows:

<table>
<thead>
<tr>
<th>F</th>
<th>C</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>32-bit word</td>
</tr>
<tr>
<td>0</td>
<td>01</td>
<td>16-bit left halfword (bits 0-15)</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>64-bit doubleword</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>16-bit right halfword (bits 16-32)</td>
</tr>
<tr>
<td>1</td>
<td>00</td>
<td>Byte 0 (bits 0-7)</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>Byte 1 (bits 8-15)</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>Byte 2 (bits 16-23)</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>Byte 3 (bits 24-31)</td>
</tr>
</tbody>
</table>

**DIRECT ADDRESSING**

When an X is equal to Zero (no indexing), and I is equal to Zero (no indirect), the effective memory address is taken directly from bits 13-29 of the Memory Reference instruction.

The Store Word instruction is coded:

```
STW 0,0
```

and is assembled as hexadecimal D4000000. When executed, this instruction stores the contents of General Purpose Register 0 directly into memory byte location 0.

The Store Byte instruction is coded:

```
STB 0,1
```

and is assembled as hexadecimal D4080001. Note that the F- and C-fields of the instruction have been altered. When executed, this instruction stores the least significant byte of General Purpose Register 0 directly into memory byte location 1.
Indirect addressing can be combined with indexing at any indirect level. An example of indirect addressing with indexing follows:

<table>
<thead>
<tr>
<th>Location Counter</th>
<th>Machine Instruction</th>
<th>Byte Address</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>P00000</td>
<td>C9800004</td>
<td>P000C</td>
<td>STRT</td>
<td>PROGRAM</td>
<td>3,4</td>
</tr>
<tr>
<td>P00000</td>
<td>REL</td>
<td></td>
<td></td>
<td>REL</td>
<td></td>
</tr>
<tr>
<td>P00004</td>
<td>AC90000C</td>
<td>P000C</td>
<td>LW</td>
<td>1,*LOC1</td>
<td></td>
</tr>
<tr>
<td>P00008</td>
<td>3055</td>
<td></td>
<td>CALM</td>
<td></td>
<td>X'55'</td>
</tr>
<tr>
<td>P0000A</td>
<td>0002</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P0000C</td>
<td>001000010</td>
<td>P00010</td>
<td>LOC1</td>
<td>ACW</td>
<td>*LOC2</td>
</tr>
<tr>
<td>P00010</td>
<td>00700014</td>
<td>P00014</td>
<td>LOC2</td>
<td>ACW</td>
<td>*LOC3,3</td>
</tr>
<tr>
<td>P00014</td>
<td>00000000</td>
<td>P0001C</td>
<td>LOC3</td>
<td>DATAW</td>
<td>0</td>
</tr>
<tr>
<td>P00018</td>
<td>0000000C</td>
<td>P0001C</td>
<td>LOC4</td>
<td>ACW</td>
<td>LOC4</td>
</tr>
<tr>
<td>P0001C</td>
<td>0000FFFF</td>
<td>P00000</td>
<td>CALM</td>
<td>DATAW</td>
<td>X'0000FFFF'</td>
</tr>
<tr>
<td>P00020</td>
<td>END</td>
<td></td>
<td></td>
<td>END</td>
<td>STRT</td>
</tr>
</tbody>
</table>

The first executable instruction is a Load Immediate (LI) to load a value of 4 into GPR3 (index register). The next instruction to be executed is the Load Word (LW). This instruction directs the machine to load GPR1, indirectly using the contents of LOC1 as the operand address. The address in LOC1, however, has the indirect bit on; the machine uses this address to fetch the contents of LOC2. The contents of LOC2 has an indirect bit on, but it also points to GPR3 for indexing. The machine then takes the address contents of LOC2 and adds to it the contents of GPR3 (which increases the address by four bytes). The resulting address points to LOC4. The address stored in LOC4 has the indirect bit off. The machine then uses the address P0001C stored in LOC4 as the final operand logical address and loads GPR1 with the hexadecimal value 0000FFFF. The ACW statement is a Macro Assembler directive used to generate an address constant. The DATAW is also a Macro Assembler directive.

Any data type may be indexed by adding a bit at the bit position corresponding to the displacement value for each data type. These are as follows:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Bit Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>31</td>
</tr>
<tr>
<td>Halfword</td>
<td>30</td>
</tr>
<tr>
<td>Word</td>
<td>29</td>
</tr>
<tr>
<td>Doubleword</td>
<td>28</td>
</tr>
</tbody>
</table>

If X is nonzero (specifying indexing), bits 13-31 are used to produce a memory address by adding it to the contents of the general purpose register specified by X. Only General Purpose Registers 1, 2, and 3 function as index registers.

For selective or indexed addressing, the displacement is a two complement integer within one of the general purpose registers used for indexing. For word indexing, bit 29 of the index register is the least significant bit of the address. If bit 29 of GPR3 is set to One to provide a displacement of one word, the indexed Store Word instruction is coded:

```
STW 0,0,3
```

This now stores the contents of GPRO in memory indexed by the contents of GPR3. The instruction would assemble as D4600000. The calculated logical effective word operand address (after indexing) would be 00004. Therefore, the contents of GPR will be stored in memory location 00004.
If I is equal to Zero, addressing is direct, and the address already determined from X and bits 12-31 is the effective address used in the execution of the instruction.

If I is equal to One, addressing is indirect, and the processor retrieves another address specified by the operand address. In this new address, bits 9 and 10 select the index register and bit 11 is the indirect bit; bits 12-31 specify the effective address as in the memory reference instructions. To use the indirect addressing capability the instruction would be coded:

```
STW 0,*0
```

which causes bit 11, the indirect bit, to be set to One. When executed, this instruction stores the contents of GPRO in the memory location whose address is stored in memory location 0.

Multilevel indirect addressing can be performed when each new address taken from memory has the indirect bit (bit 11) set to One. The process of fetching indirect addresses continues until an address has bit 11 equal to Zero. This address is the logical effective operand address.

Each fullword instruction (32 bits) must be stored in memory on a word boundary (bits 30 and 31 equal to Zero). Memory information boundaries are illustrated in Figure 4-1.

Halfword instructions are stored two per word. When a halfword is followed by a word instruction, the Assembler positions the instruction in the left half of the word and stores a No Operation (NOP) instruction in the right half of the word. This maintains the word boundary discipline.

Memory Reference instructions which address a byte in memory do not alter the other three bytes in the memory word containing the specified byte. Memory instructions which address a halfword do not alter the other halfword of the memory location. The exception to the preceding is that the Add Bit in Memory instruction may propagate a carry to the most significant bit of the word containing the specified bit.

Word operands must be stored in memory on a word boundary. The most significant word of a doubleword operand must be stored in a memory location having an even word address with the least significant word stored in the next sequentially higher (i.e., odd word) location. Some examples of memory addressing follow:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Halfword</th>
<th>Word</th>
<th>Doubleword</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>0000</td>
<td>00000</td>
<td>00000</td>
</tr>
<tr>
<td>00001</td>
<td>00002</td>
<td>00004</td>
<td></td>
</tr>
<tr>
<td>00006</td>
<td>00006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00008</td>
<td>00008</td>
<td>00008</td>
<td>00008</td>
</tr>
<tr>
<td>00009</td>
<td>0000A</td>
<td>0000A</td>
<td></td>
</tr>
<tr>
<td>0000B</td>
<td>0000C</td>
<td>0000C</td>
<td></td>
</tr>
<tr>
<td>0000D</td>
<td>0000E</td>
<td>0000E</td>
<td></td>
</tr>
<tr>
<td>0000F</td>
<td>00010</td>
<td>00010</td>
<td>00010</td>
</tr>
<tr>
<td>00010</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4-6
Figure 4-1. Information Boundaries in Memory
HARDWARE
MEMORY
MANAGEMENT

The 32/70 Series computer features Hardware Management that provides full utilization of all available memory. The memory management hardware includes: hardware Memory Allocation and Protection (MAP), extensions to the interrupt, I/O, and memory subsystems. This feature also allows programs to be loaded in one area of physical memory, rolled out to disc, rolled back into another area of memory, and to continue execution without requiring time-consuming software relocation biasing.

In addition, these programs may be distributed throughout physical memory in 32K-byte blocks to take complete advantage of available memory. Hardware Memory Management, including automatic context switching, is accomplished through the processing and control of the MAP. The MAP consists of up to thirty-two 16-bit halfwords. The first 16 halfwords (the Primary MAP) are used to define a 512K-byte logical primary address space into which may be loaded either data or executable programs. The second 16 halfwords (the Extended Operand MAP) are used to define a 512K-byte logical extended operating address space into which only data may be loaded.

By using the MAP, a 512K-byte logical primary address space may be distributed in 32K-byte blocks throughout the 16,777,216 bytes of physical memory and may contain data or instructions. The 32/70 Series computer can access and execute programs up to 512K bytes in size, located anywhere within physical memory (16M bytes). The user can also use an additional 512-K byte logical extended operand address space for data storage. The combination of the logical primary address space and the additional extended operand address space provides support throughout physical memory, provided that the executable code lies entirely within the logical primary address space.

ADDRESSING
MODES

The 32/70 Series computer provides the capability of accessing memory in any of the following modes:

1. 512 KB mode
2. 512 KB Extended mode
3. 512 KB Mapped mode
4. Mapped, Extended mode

512 KB MODE

The 512 KB mode of memory address allows the 32/70 Series Central Processor Unit to directly access any byte, halfword, word, or doubleword in the first 512K bytes of memory without mapping, indexing, or address modification. A 19-bit address field is provided in all Memory Reference instructions for this purpose.

Bits are addressed by using the R (register) field of the instruction word to designate a bit in the byte specified by the 19-bit address. Therefore, any bit in 512K bytes of memory can be directly addressed by the Bit Manipulation instructions.

512 KB EXTENDED MODE

The 512 KB Extended mode of memory addressing provides the same capabilities as the 512 KB mode plus operand addressing beyond the first 512K bytes of memory to reference all bits, bytes, halfwords, words, and doublewords residing anywhere within 16 megabytes of physical memory. This mode of addressing combines the contents of an index register with the 19 bits of logical address in the Memory Reference Instruction to produce a 24-bit physical memory address anywhere in the 16 megabytes of memory. All memory above the first 512K bytes is usable only for data storage and retrieval and not for executable instructions. This mode of memory addressing is applicable to both the PSW and the PSD modes of operation.
The 512 KB Mapped mode of memory addressing allows a 32/70 Central Processor Unit to access any byte, halfword, word, or doubleword within 16 megabytes of memory through memory mapping. In this mode, the memory management hardware supports up to 16 logical address pages (a page is 32K bytes) distributed throughout 16 megabytes of physical memory by providing mapping and automatic context, MAP, and protection switching. All 16 pages of logical address pages may be used for executable code instructions or for data storage and retrieval. Physical blocks of memory may be common to multiple address spaces, providing a way for users in different address spaces to share common blocks of memory.

The Mapped/Extended mode of memory addressing allows a 32/70 Series Central Processor Unit to access any byte, halfword, word, or doubleword within 16 megabytes of memory through memory mapping. In this mode, the memory management hardware supports up to 32 logical address pages (a page is 32K bytes) distributed throughout 16 megabytes of physical memory by providing mapping and automatic context, MAP, and protection switching. The first 16 pages of logical address pages may be used for executable code or data, and the last 16 pages of logical address pages must be used for data storage and retrieval only. Multiple-user programs may be loaded into any or all of the first 16 pages of logical address pages. A 32/70 Series Computer allows each of these users to directly address any bit, byte, halfword, word or doubleword within the address space in which it resides. Physical blocks of memory may be common to multiple address spaces, providing a way for users in different address spaces to share common blocks of memory.

The 32/70 Series computer includes thirty-two 16-bit (halfword) locations, the Primary MAP, and the Extended Operand MAP. The Primary MAP and the Extended Operand MAP are used to map the 512K-byte logical primary address space and the 512K-byte logical extended operand address space, respectively, onto physical memory addresses. Each of the 16-bit MAP locations associates 32K bytes of the logical primary address space or logical extended operand address space with 32K bytes (8K words) of physical memory. Logical address spaces are defined by building MAP Image Descriptor Lists (MIDL) as shown in Figure 4-2.

Each MIDL contains up to 32 halfword page entries (a page is 32K bytes or 8K words), which contains a 12-bit Page Entry, a Page Valid or Nonvalid bit, and a Write Protect/Unprotect bit. Any or all of the 32 pages may be designated as Write Protected. The first 16 page entries (logical primary address space) may be used for executable instructions or for data storage and retrieval. The second 16 page entries (Extended Operand MAP Image) may only be used for data storage and retrieval purposes. For a complete description of the Memory Mapping, refer to the Memory Addressing section of the Instruction Repertoire.

A logical representation of the components involved in the memory management process of a 32/70 Series system are depicted in Figure 4-3.
<table>
<thead>
<tr>
<th>BIT</th>
<th>VALID</th>
<th>DETECT</th>
<th>EVEN HALFWORDS</th>
<th>ODD HALFWORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD 16</td>
<td>0 1 2</td>
<td>3 4 5 6 7 8 9 10 11 12 13 14 15</td>
<td>16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>PRIMARY MAP PAGE 0</td>
<td>PRIMARY MAP PAGE 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PRIMARY MAP PAGE 2</td>
<td>PRIMARY MAP PAGE 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PRIMARY MAP PAGE 4</td>
<td>PRIMARY MAP PAGE 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PRIMARY MAP PAGE 6</td>
<td>PRIMARY MAP PAGE 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>PRIMARY MAP PAGE 8</td>
<td>PRIMARY MAP PAGE 9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PRIMARY MAP PAGE 10</td>
<td>PRIMARY MAP PAGE 11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>PRIMARY MAP PAGE 12</td>
<td>PRIMARY MAP PAGE 13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>PRIMARY MAP PAGE 14</td>
<td>PRIMARY MAP PAGE 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>EXTENDED OPERAND MAP PAGE 0</td>
<td>EXTENDED OPERAND MAP PAGE 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>EXTENDED OPERAND MAP PAGE 2</td>
<td>EXTENDED OPERAND MAP PAGE 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>EXTENDED OPERAND MAP PAGE 4</td>
<td>EXTENDED OPERAND MAP PAGE 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>EXTENDED OPERAND MAP PAGE 6</td>
<td>EXTENDED OPERAND MAP PAGE 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>EXTENDED OPERAND MAP PAGE 8</td>
<td>EXTENDED OPERAND MAP PAGE 9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>EXTENDED OPERAND MAP PAGE 10</td>
<td>EXTENDED OPERAND MAP PAGE 11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>EXTENDED OPERAND MAP PAGE 12</td>
<td>EXTENDED OPERAND MAP PAGE 13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>EXTENDED OPERAND MAP PAGE 14</td>
<td>EXTENDED OPERAND MAP PAGE 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 4-3. Memory Management Components

- Program Status Doubleword (PSD)
  - CC's
  - Program Counter
  - Granularity
  - BPI
  - CPI
  - ADD
  - ADD

- Scratchpad
  - X'83'
  - MPL Base Address

- Segment Control Descriptors
  - MAP Segment Descriptor List (MSDL)
  - SPC
  - MIDL Pointer
  - SPC
  - MIDL Pointer
  - SDC
  - MSDL Pointer
  - SDC = Segment Descriptor Count
  - SPC = Segment Page Count

- Master Process List (MPL)
  - MAP Segment Descriptors
  - Segment Descriptor Count (SDC)
  - 0
  - 6 Bits SDC
  - MAP Segment Descriptors
  - MAP IMAGE DESCRIPTORS
  - PAGE ENTRY 0
  - PAGE ENTRY 1
  - PAGE ENTRY 2
  - PAGE ENTRY 3

- MAP IMAGE DESCRIPTOR LIST (MIDL)
  - MAP IMAGE DESCRIPTORS
  - PAGE ENTRY
  - PAGE ENTRY
  - PAGE ENTRY

- MAP IMAGE DESCRIPTORS
  - PAGE ENTRY
  - PAGE ENTRY
The memory protection system provides write protection for individual memory pages. When the CPU is in the Mapped mode (either 512 KB or Extended), each 32 KB memory block of logical program address space may be write protected. Write protection for a 32 KB memory block is selected by setting the protect/unprotect bit that is stored, along with the block address, in the MAP register of the CPU.

When the CPU is in the Unmapped mode (either 512 KB or Extended), 512-word memory pages may be write protected. Up to 256 pages (128K words) can be protected at a time. The sixteen 16-bit Page Protect registers are provided in the Unmapped mode.

Write Protection may be overridden by a CPU operating in the Privileged mode.

The Program Status Doubleword (PSD) provides information relating to the operation that was interrupted or trapped (Old PSD), and the mode and instruction address that is to be given control during context switching (New PSD). The format of the PSD is shown in Figure 4-4.

Execution of any Branch or Branch-and-Link instruction replaces the contents of bits 13-30 of the PSD with the effective address specified by the instruction. In addition, if the Branch instruction specifies an Indirect Branch operation, the contents of bits 1-4 of the PSD are replaced by the contents of the corresponding bit positions in the indirect address location.

The PSD fields are coded as follows:

1. **PRIV** (bit 0) indicates the Privileged mode.
   - 0 = Nonprivileged
   - 1 = Privileged

2. **CCs** (bits 1-4) indicate the condition codes.
   - Bit 1 = CC1
   - Bit 2 = CC2
   - Bit 3 = CC3
   - Bit 4 = CC4

3. **EXT** (bit 5) indicates Indexing mode.
   - 0 = Off
   - 1 = On

4. **HIST** (Bit 6) indicates last instruction was a right halfword (Old PSD only).

5. **AEXP** (Bit 7) indicates Arithmetic Exception Trap Mask.
   - 0 = OFF (Do not generate Arithmetic Exception Trap)
   - 1 = ON (Generates Arithmetic Exception Trap)

6. **PSD** (Bit 8) indicates PSD mode.
   - 0 = PSD mode off (Displayed PSD only)
   - 1 = PSD mode on (Displayed PSD only)
Figure 4-4. Formats for PSD1 and PSD2
7. MAP (Bit 9) indicates Mapped mode
   0 = Unmapped mode (Displayed PSD only)
   1 = Mapped mode (Display PSD only)

8. PROGRAM COUNTER (Bits 10-29) indicate the logical program counter (Word Address).
   Bits 10-12 are reserved for possible later use. (They must be zero)
   Bits 13-29 are the logical address.

9. NR (Bit 30) indicates next instruction is a right halfword.

10. Blocked (Bit 31) indicates Blocked mode (Displayed PSD only).

11. MAP MODE (Bits 32-33) indicate the Granularity as:
    00 = Unmapped
    01 = Mapped 8K Granularity
    10 = Mapped 8K Granularity
    11 = Mapped 8K Granularity

12. BPIX (Bits 34-46) provide a word index into the Master Process List (MPL) for the base process. (Bit 46 is ignored.)

13. Bit 47 = Retain current MAP contents. (New PSD only)

14. EXT INT FLAG (Bits 48 and 49) indicate external interrupt state.

<table>
<thead>
<tr>
<th>Bits</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>= Operate with Unblocked interrupts (interrupt level active)</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>= Operate with Blocked interrupts (interrupt level not active)</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>= Retain Current Blocking Mode (New PSD only)</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>= Retain Current Blocking Mode (New PSD only)</td>
<td></td>
</tr>
</tbody>
</table>

15. CPIX (Bits 50-63) provide a word index into the Master Process List (MPL) for the current process. Bits 62 and 63 are ignored.

**CONDITION CODES**

A 4-bit Condition Code is stored in the PSD on completion of the execution of most instructions. These conditions may be tested to determine the status of the results obtained.

- CC1 is set if an Arithmetic Exception occurs
- CC2 is set if the result is greater than zero
- CC3 is set if the result is less than zero
- CC4 is set if the result is equal to zero

The Branch Condition True (BCT), Branch Condition False (BCF), and the Branch Function True (BFT) instructions allow testing and branching on the Condition Codes.
The second word of the PSD contains two 12-bit fields whose primary purpose is to provide the linkage from that PSD to the correct map entries for execution of the process associated with that PSD. The CPU MAP consists of a RAM with 32 locations, and the firmware will locate the appropriate entries for this RAM in main memory through a set of software-maintained tables which are interpreted by firmware on these two values from the PSD.

The 12-bit fields are named as follows:

1. BPIX – Base Process Index
2. CPIX – Current Process Index

The software maintains a Master Process List in memory. The base address is kept in a known (scratchpad) location. It contains one entry for every value which can appear in either the BPIX or CPIX fields, and it is quite reasonable for PSDs to exist where the CPIX and BPIX are identically equal. This Master Process List is maintained by the most privileged code of the system, and destruction of its contents will surely lead to immediate disaster.

The address of the MPL is set by the CPU firmware at System Reset time by the loading of a predetermined scratchpad cell with the 24-bit physical MPL address. The MPL entries contain the physical address of the MAP Segment Descriptor List (MSDL) and a 6-bit count of the number of Map Segments which concatenate to form the appropriate map contents.

When a PSD is being entered into the CPU, the firmware is faced with one of three possible actions relating to the map:

1. The PSD being loaded has its mode set to Unmapped, which basically means that it is going to operate with physical rather than logical memory addresses. Firmware action when loading this type of PSD is simply to leave the map contents as they are, and cause them to become inactive for the duration of this PSD execution.

   The Unmapped indication in the PSD overrides the Load Program Status Doubleword And Change Map (LPSDCM) instruction.

2. The PSD is being loaded as a result of the software instruction LPSD. In this event, firmware is being assured by the software that the map contains the appropriate contents and the only firmware action necessary is to reactivate the map circuitry. The basic function of this is to avoid the cost of reloading the map when returning from an excursion into an unmapped function, and software will insure that no other mapped process has intervened.

3. With the exception of the two preceding cases, the entry of a new PSD into the CPU always results in a total initialization of the map circuit.

   The MAP RAM will be loaded from page 0 up with values obtained from main memory.

The PSD being loaded contains sufficient information for the firmware to make its way through the series of software-maintained tables in main memory to assemble the information necessary to initialize the map circuit. The objective of the table design is to provide for the assembling of an addressability for that PSD from three distinct types of elements:
1. Private data which is unique to that process.

2. Statically shared data which is shared between several processes. This sharing is known at load (map creation) time. Since there exists in reality only a single copy of the data, it is important to software that a single physical copy of its logical/physical map exists, and that all PSDs using this shared data are funneled through that copy for both software sanity and usage statistics.

3. Data that is shared by means of dynamic invocation. This data (like a Task Service Area (TSA)) is logically "owned" by a particular process, but needed by a variety of other processes which are invoked by the original process in the course of its execution. This data is generally of the type that it is a "per process global" set of data where any number of Operating System (OS) services need a random subset of the information which defies the organization as a reasonable parameter package, and is likely unalterable directly by the "owning" process. The OS services which need this data essentially have a partial map in memory covering their private code and data, which must be completed by adding this invocation page for them to correctly perform their functions.

It would be possible to accomplish this dynamic completion of the OS service map by moving into the service map image in memory, but the complexity of maintaining a stack of these invocations and returns (which are totally unsequenced due to the dispatching strategy) is large, and a dynamic link through the PSD relieves both complexity and overhead in this area.

The key elements of the PSD which provide firmware with the ability to satisfy these requirements are two 12-bit fields in the second word of the PSD, the CPIX (Current Process Index), and the BPIX (Base Process Index).

These two fields are both direct word indices into a software-maintained Master Process List (MPL) which is located in physical memory. It is both reasonable and frequent that the BPIX and CPIX fields of a PSD contain the identical number. The MPL is maintained by the most privileged OS code and any destruction will result in immediate disaster.

When the firmware must initialize the map circuit during the loading of a PSD, the following procedure is followed:

1. Using CPIX, locate the MAP Segment Control Descriptor (MSCD) in the MPL. This word is the controlling factor in map initialization. This word consists of three fields (see Figure 4-5):
   a. Borrowed Bit (Bit 0) - Tells the firmware (1) that the first set of map entries are to be obtained from the BPIX MSCD to satisfy the invocation sharing time of creation of this entry, and (2) the numeric value of the BPIX was unknown (and there exists a multiplicity of BPIXs).
   b. Segment Descriptor Count (SDC) - The count of the number of Segment Descriptors which are required to describe the addressability of the PSD.
   c. MAP Segment Descriptor List (MSDL) Pointer - The physical address in main memory of the first (or second if the borrowed bit was set) CPIX Segment Descriptor.
A MAP Segment Descriptor (MSD) is a single word entry which has two fields (see Figure 4-6):

1. Segment Page Count (SPC) - A count of the number of pages (map locations) which this Segment Descriptor covers.

2. Map Image Descriptor List (MIDL) Pointer - The starting physical address of the map cell block which contains the MAP Image Descriptors (MID). A MAP Image Descriptor is a single word with one or two halfword page entries (see Figure 4-7).

If the borrowed bit is set when the firmware locates the MSCD, the first segment descriptor is taken from the segment list which is described by the BPIX, and the second and subsequent segment descriptors are taken from the list described by this MSCD. When this indirection has been completed, the only noticeable impact on further processing is that the first map cell to be loaded from this list is "n" rather than "0" (if the borrow bit had not been set).

The variable length of pages described by each segment descriptor word are concatenated into the map until the segment count from the MPL is exhausted. The initialization is complete.

Address generation is accomplished by adding the contents of the instruction to the contents of the index register to form a logical address. In the Unmapped modes, the logical address is the same as the physical address. In Mapped modes, a portion of the logical address is used to address the MAP, while the remaining portion is used in the physical address. A graphical representation of the address generation process for each of the four modes is presented in Figures 4-8 to 4-11.
Figure 4-5. MAP Segment Control Descriptor (MSCD)

Figure 4-6. MAP Segment Descriptor (MSD)

Figure 4-7. MAP Image Descriptor (MID)
NOTE: THIS METHOD MAY ADD OR SUBTRACT INDEXED ADDRESSES DEPENDING ON THE SIGN OF THE INSTRUCTION.

Figure 4-8. Address Generation (128 KW)
INSTRUCTION

<table>
<thead>
<tr>
<th>ZERO EXTENDED</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>32</td>
</tr>
</tbody>
</table>

(X)

<table>
<thead>
<tr>
<th>INDEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

LOGICAL ADDRESS

| 8     | 31     |

PHYSICAL ADDRESS

| 8     | 31     |

NOTE: THE INSTRUCTION BEING ZERO EXTENDED DOES NOT ALLOW SUBTRACTION OF INDEXED ADDRESSES.

Figure 4-9. Address Generation (512 KB Extended Mode)
Figure 4-10. Address Generation (512 KB Mapped Mode) (Non-Extended)
Figure 4-11. Address Generation (Mapped, Extended Mode)
INTRODUCTION

Input/Output (I/O) operations consist of transferring blocks of bytes, halfwords, or words between core memory and peripheral devices. Transfers are performed automatically, requiring minimal CPU involvement.

All system components which participate in the execution of an I/O operation are illustrated in Figure 5-1. The peripheral device(s) shown may be either data processing devices such as disc files, magnetic tape units, line printers, card readers, and card punches; or they may be real-time system devices such as data acquisition subsystems, communications control units, or system control units.

There are two modes of I/O operation possible, the first being the Program Status Word (PSW) mode which responds only to Class 0, 1, 2, 3, and E I/O processors. The second is the Program Status Doubleword (PSD) mode, which will respond to all of the preceding I/O processors as well as Class F I/O processors.

The I/O processors used in a 32/70 Series computer are available in three types. The first type is the standard Input/Output Microprogrammable Processor (IOM) containing a SelBUS interface, Microprogrammable Processor, and Device Dependent logic. The second type of I/O processor is the Integrated Channel Controller, also known as the Regional Processing Unit (RPU) (Figure 5-2) which combines the functions of a channel and a controller into one unit. The function of a channel is to schedule the requests for main memory between a number of controllers. The channel also interfaces the controller with the CPU to initiate or terminate an I/O operation. The third type of I/O processor is the General Purpose Multiplexer Controller (GPMC) and General Purpose Device Controller (GPDC) combination. The GPMC functions as the SelBUS interface, and as the decode and control logic for up to 16 device addresses. The GPMC also controls a number of independent device controllers that are located some distance from itself. The independent device controllers (GPDCs) function as device interface logic for one or more devices per GPDC.

DEFINITIONS

The following definitions are presented to aid in understanding the Input/output operations.

1. **I/O Processor**—The entire subsystem that interfaces the SelBUS and provides I/O ports to the devices.

2. **External Media**—A general term for punched cards, printed forms, magnetic tape, or discs.

3. **Input/Output Devices**—The peripheral devices interfaced to a 32/70 Series computer, e.g., card reader, card punch, paper tape reader, paper tape punch, line printer, and magnetic tape drives.
Figure 5-1. 32/70 Series Input/Output Organization
Figure 5-2. Block Diagram - Regional Processing Unit (RPU)
4. **Direct Access Devices**—A type of storage device wherein access to the next position from which information is to be obtained is in no way dependent on the position from which information was previously obtained. Magnetic disc drives and magnetic drums are examples of direct access devices.

5. **Communications Devices**—Real-time devices, such as teletypewriters and process control devices, that interface to a 32/70 Series computer.

6. **Controllers**—A general term used to describe the peripheral device interface logic. One controller may handle several devices.

7. **Channel**—That portion of an I/O processor containing the logic to interface the SELBUS and to control the device interface logic. One channel may handle one or more controllers.

8. **Commands**—Commands are directives that are decoded and executed by the channel, controller, and I/O device to initiate the I/O operation.

9. **Instructions**—Directives to the CPU that are decoded and executed by the CPU. Instructions are a part of the CPU program.

10. **Command List**—One or more commands arranged for sequential execution.

11. **Data Chaining**—Data Chaining is specified by a flag in the IOCD and causes a channel to fetch the next IOCD when the byte count in the current IOCD reaches zero.

12. **Local Store**—Another name for the CPU scratchpad memory.

13. **Channel End**—A termination condition that indicates all information associated with the operation has been received or provided, and that the channel and controller are no longer needed. This condition resets all conditions in the CPU scratchpad pertaining to the specific channel and controller.

14. **Device End**—An indication from the controller to the channel that an I/O device has terminated execution of its operation.

15. **Controller End**—Operations that keep the controller busy after reporting a Channel End cause Controller End reporting (at the end of its operation) indicating that the controller is available for initiation of another operation.

---

**I/O Processor Classification**

I/O processors are classified as types 0, 1, 2, 3, E, and F. The type 0, 1, and 2 I/O processors are associated with the teletype, line printer, and card reader respectively, and are contained on a single IOM. The type 3 I/O Processor is the RTOM Interval Timer. A type E I/O processor is one which is controlled by the use of the Command Device (CD) and Test Device (TD) instructions and has the capability of only addressing 512 KB of memory. The type F I/O processor responds to the 32/70 Series I/O instructions, has the capability of addressing memory throughout a 16 MB range, and in some cases supports an optional Writable Control Storage (WCS) unit.

**Operation with Class 0, 1, 2, and E I/O Processors**

Input/Output (I/O) operations with the Class 0, 1, 2, and E I/O processors consist of transferring blocks of bytes, halfwords, or words between core memory and peripheral devices. Core memory locations addressed by these I/O processors are limited to the first 128K words (512K bytes) of contiguous memory. Transfers are possible at rates up to 1.2 million bytes per second. The system components which participate in the execution of an I/O operation are illustrated in Figure 5-3.
A 32/70 Series system will support a total of 16 I/O processors. Each I/O processor may in turn support as many as 16 device addresses, allowing as many as 128 separate addressed devices to be connected to a 32/70 Series computer at one time.

Two types of I/O instructions, Command Device (CD) and Test Device (TD), are executable by Class 0, 1, 2, and E I/O processors.

Transfer of a block of information is initiated by execution of a Command Device instruction in the CPU. This instruction, illustrated in Figure 5-4, specifies the device, the direction of transfer, and other control parameters required to condition the device to generate or accept data. The control parameters are defined in Figure 5-5. The I/O processor, consisting of an IOM and Device Dependent logic, accepts the Command Device from the CPU, routes the device control parameters to the device specified in the instruction, and initializes the transfer of a block of data. A Transfer Control Word contains the starting memory address and the number of transfers to be made, and is contained in a memory location dedicated to each device address.

The Transfer Control Word (TCW) contains a 20-bit address which defines the memory location for each transfer. It also contains a positive 12-bit binary Transfer Count (TC). The Transfer Count plus the Format Code (FC) permits transfers of blocks of information having any number of bytes, halfwords, or words up to 4,096. The format of the Transfer Control Word (TCW) is shown in Figure 5-6.

The presence of the Format Code in the TCW permits transfers of bytes, halfwords, or words. The Format Code is designed such that when F is equal to One in a given TCW, the address is incremented in bit position 31 each time a transfer occurs. Therefore, each transfer is stored in or read from a consecutive byte in memory in this order:

Word N          Word N+1
---Byte 0,Byte 1,Byte 2,Byte 3     Byte 0,Byte 1,Byte 2,Byte 3---

The proper binary value of Format Code for accessing consecutive halfwords in memory is F equal to 0, C equal to Y1, where Y equal to Zero designates left halfword and Y equal to One designates right halfword. With this value of Format Code, the address is incremented in bit position 30 each time a transfer is made. This results in the desired accessing of consecutive halfwords.

The proper value of Format Code for consecutive word accessing is TCW equal to 000. When this value is present in a given TCW, the I/O processor increments the TCW in bit position 29 each time a transfer occurs.

The Format Code values discussed above are summarized in Table 5-1.

Each time the address is incremented, the Transfer Count is decremented. Therefore, the block length is always defined by the number of memory accesses and not by the number of words transferred. For specific I/O processors (i.e., GPMC, HSD, ADI, and FMS), the TCW address field is used to supply an Input/Output Command Doubleword (IOCD) address.

The dedicated memory addresses used with the 16 I/O Processors are included in the list of Relative Trap/Interrupt Priorities (reference Table 3-1).
Figure 5-3. Class 0, 1, 2, and E I/O Organization

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

Refer to Figure 5-5

BIT 16 = 0

BIT POSITIONS 20 THROUGH 31 OF THE FUNCTION CODE ARE UNIQUE TO THE DEVICE
BIT POSITIONS 18 AND 19 PROVIDE THE FOLLOWING INFORMATION:

BIT 18 = 1 TRANSFER CURRENT WORD ADDRESS
BIT 19 = 1 TERMINATE (RESET I/O CONTROLLER)

BIT 16 = 1

A TRANSFER IS TO BE INITIALIZED AND BITS 18 AND 19 OF THE FUNCTION CODE WILL PROVIDE THE FOLLOWING INFORMATION:

BIT 19 = 0 OUTPUT TRANSFER (WRITE)
BIT 19 = 1 INPUT TRANSFER (READ)

Figure 5-4. Command Device Instruction Format
<table>
<thead>
<tr>
<th>BIT DEVICE</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARD READER</td>
<td>N.U</td>
<td>N.U</td>
<td>0</td>
<td>0</td>
<td>TERMINATE</td>
<td>0</td>
<td>PROGRAM VIOl - 9</td>
<td>INPUT - 1</td>
<td>BINARY MODE</td>
<td>AUTO MODE</td>
<td>* IF ZEROS - TRANSLATE MODE + 1/2 ASCII = FUNNY CODE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINE PRINTER</td>
<td>N.U</td>
<td>N.U</td>
<td>0</td>
<td>0</td>
<td>TERMINATE</td>
<td>0</td>
<td>OUTPUT - 9</td>
<td>ADJUST FORMAT</td>
<td>FORMAT</td>
<td>ADV LOINE</td>
<td>* FORMAT MEANS USE PAPER ADVANCE BY VERT FORMAT LOOP CHAR 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TELEX OR CRT CONSOLE</td>
<td>N.U</td>
<td>N.U</td>
<td>0</td>
<td>0</td>
<td>TERMINATE</td>
<td>0</td>
<td>KEYBOARD ECHO</td>
<td>OUTPUT - 9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAGNETIC TAPE (B TK)</td>
<td>N.U</td>
<td>N.U</td>
<td>TRANSFER CURRENT ADDR + 1</td>
<td>TERMINATE</td>
<td>0</td>
<td>BACKSPACE</td>
<td>ERASE 2.5&quot; TAPE</td>
<td>ADJUST TO EOF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAGNETIC TAPE (T TK)</td>
<td>N.U</td>
<td>N.U</td>
<td>TRANSFER CURRENT ADDR + 1</td>
<td>TERMINATE</td>
<td>0</td>
<td>BACKSPACE</td>
<td>ERASE 2.5&quot; TAPE</td>
<td>ADJUST TO EOF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CARTRIDGE DISC # 9003</td>
<td>N.U</td>
<td>N.U</td>
<td>TRANSFER CURRENT ADDR + 1</td>
<td>TERMINATE</td>
<td>0</td>
<td>RECAL</td>
<td>SEEK</td>
<td>TRACK</td>
<td>TK 128</td>
<td>TK 64</td>
<td>TK 32</td>
<td>TK 16</td>
<td>TK 8</td>
<td>TK 4</td>
<td>TK 2</td>
<td>TK 1</td>
</tr>
<tr>
<td>MOVING-READ DISC # 9019</td>
<td>N.U</td>
<td>N.U</td>
<td>TRANSFER CURRENT ADDR + 1</td>
<td>TERMINATE</td>
<td>0</td>
<td>RECAL</td>
<td>SEEK</td>
<td>TRACK</td>
<td>TK 128</td>
<td>TK 64</td>
<td>TK 32</td>
<td>TK 16</td>
<td>TK 8</td>
<td>TK 4</td>
<td>TK 2</td>
<td>TK 1</td>
</tr>
<tr>
<td>FIXED-READ DISC # 9014</td>
<td>N.U</td>
<td>N.U</td>
<td>TRANSFER CURRENT ADDR + 1</td>
<td>TERMINATE</td>
<td>0</td>
<td>RELEASE</td>
<td>SEEK</td>
<td>TRACK</td>
<td>TK 16</td>
<td>TK 18</td>
<td>0</td>
<td>AND TRACK ADDRESS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.5. Command Device Function Bit Format For Peripheral Devices
BITS 0-11 DESIGNS THE NUMBER OF TRANSFERS TO BE MADE BETWEEN MEMORY AND THE DEVICE CONTROLLER CHANNEL.

BITS 12,30,31 SPECIFY THE FORMAT CODE FOR EACH TRANSFER (SEE TABLE 5-1).

BITS 13-29 DESIGNATE THE MEMORY LOCATION FOR EACH TRANSFER.

NOTE

THE WA FIELD IS INTERPRETED AS A 24-BIT REAL ADDRESS BY THE I/O PROCESS. THEREFORE, THE ADDRESS RANGE IS LIMITED TO THE FIRST 512 KB OF MEMORY.

Figure 5-6. Transfer Control Word Format

### Table 5-1. Transfer Control Word Format Code

<table>
<thead>
<tr>
<th>Information Format</th>
<th>FC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>1XX</td>
</tr>
<tr>
<td>Halfword</td>
<td>0Y1</td>
</tr>
<tr>
<td>Word</td>
<td>000</td>
</tr>
</tbody>
</table>

XX = Byte number
Y = 0 designates left halfword
Y = 1 designates right halfword
Figure 5-7. Test Device Instruction Format

Figure 5-8. Test Device 2000 Status Information
The Test Device (TD) instruction is used to acquire status information from the Input/Output processor and the associated device(s). Three levels of the TD instruction (8000, 4000 and 2000) may be used to acquire this information. The status information is in the form of four condition code bits for each level of test. The TD instruction does not initiate any action in the device. The TD 8000 instruction is used by the CPU to test the general status of the addressed device and associated I/O processor. The TD 4000 instruction is used by the CPU to obtain 16 bits of status information from the device/processor. This instruction causes the addressed I/O processor to transfer a 16-bit status word to the memory address specified by the TCW. The 16-bit status word may be placed in memory in either the right or left halfword position, depending on bits 30 and 31 of the TCW address. A TCW used with a TD 2000 should always specify halfword memory addressing. Figure 5-7 provides a breakdown of the Test Device instruction format. Figure 5-8 provides the status information returned from standard peripheral devices upon execution of TD 2000 instructions.

Each Input/Output processor consists of an Input/Output Microprogrammable Processor (IOM) and Device Dependent Interface logic. The Microprogrammable Processor (MP) and the Device Dependent Interface logic are customized for each device. The firmware for a given Input/Output processor is contained in a set of PROMs that plug into the processor board. The information contained within the PROMs is device dependent.

This design technique provides extreme flexibility for custom designed interfaces since the basic MP and SelBUS interface are also available as a General Purpose I/O Processor (GPIO). All that is needed to convert the GPIO processor into a special purpose I/O processor is the Device Dependent Interface logic and the firmware microprogram.

The maximum throughput of an Input/Output processor is 1.2 million bytes per second.

There are two types of Input/Output processors:
1. Multiple Device Controller (MDC)
2. Multiple Controller Controller (MCC)

The MDC controls like devices, such as four magnetic tapes. The MCC emulates multiple controllers such as the TLC Input/Output processor that controls a teletype, a card reader, and a printer. MCC Input/Output processors are multiplexed processors handling more than one device simultaneously accessing memory. The Asynchronous Data Set Interface (ADS) is an example of a multiplexed processor. The ADS handles four half- or full-duplex lines directly to memory on a message basis. Four memory input buffers and four output buffers can be active at one time.

The Input/Output SelBUS interface contains the registers and SelBUS drivers for a full 32-bit data transfer. The main function of this logic is to receive and drive communications on the SelBUS. All the interface control logic, including processor address recognition, interrupt polling, and data transfer to and from the SelBUS, are included in the interface.

The bus priority logic is controlled by the interface control logic. It polls for the SelBUS, determines when it wins the poll, and then drives the transfer on the bus. Priorities are set through physical switches in the Input/Output processor.
An Input/Output processor will respond to all bus transfers that it receives. It has three immediate responses:

1. Retry
2. Busy
3. Transfer Acknowledge

The sending bus device can determine the status of its transfer to the Input/Output processor by monitoring these lines. A Retry answer means that the Input/Output processor of the MCC type is temporarily busy. A Busy means to set the busy condition code bit in the software instruction and proceed with the next instruction. An Input/Output processor of the MDC type would generate such a return. A Transfer Acknowledge indicates that the transfer was accepted and is being processed. If no answer is present in the bus cycle following the transfer, a non-present Input/Output processor was addressed.

The 10M data structure provides for the transfer of data, arithmetic and logical manipulation of data, storing of device and processor status, decoding of commands, and data buffering. Figure 5-9 provides a block diagram of the 10M.

Two 16- by 16-bit word register groups, RA and RB, are available as working read/write memory. The output for each register pair is the input to the Arithmetic/Logic Unit.

The destination address and the most significant 16 bits of the data bus are directed to the RA register group. The program counter and the ALU output are also directed to the RA register group. The least significant 16 bits of the data bus and 16 bits of data from the peripheral devices are directed to the RB register group. The ALU output and a 16-bit literal from the control register are also input to the RB register group.

The data structure includes a full 16-bit Arithmetic/Logic Unit which inputs from RA and RB. The ALU is equipped with a 3-bit status register which contains previous carry, all zeros condition, and the most significant bit.

A 32-bit by 1,024-word microprogrammed control memory and a 48-bit test structure (32 implemented) control the flow of data and commands between the SelBUS and peripheral devices.

The IOM test structure is used with the Wait and Conditional Branch operations to control the sequencing and timing of instructions.

The IOM has a single Master Interrupt line. For device controllers requiring more interrupts, the necessary mask register and Priority Decode logic is included in the Device Interface logic.

The following discussions refer to the organization and operation of Series Class F I/O processors.

Class F Input/Output operations consist of transferring blocks of bytes, halfwords, or words between core memory and the peripheral devices. Transfers are performed automatically requiring a minimum of CPU involvement.

A typical configuration for Class F I/O operation is illustrated in Figure 5-10. The I/O devices include card readers, line printers, discs, magnetic tapes, and telecommunications equipment. The controller provides the logical and buffering capabilities necessary to operate an I/O device. The controller is attached to a channel. The channel's function is to schedule the requests for main memory between a number of controllers. The channel also connects the controller to the CPU to initiate or terminate an I/O operation.
Figure 5-9. Block Diagram - I/O Microprogrammable Processor
The integrated channel controller, also known as the RPU, combines the functions of a channel and a controller into an indistinguishable unit.

An I/O processor consists of two or more distinct logic subassemblies which are:

1. The Channel— which interfaces with the SelBUS to send and receive information between the channel, the CPU, and/or memory. The other side of the channel interfaces with one or more controllers to provide control signal and data paths to/from the controllers.

2. The Controller—which interfaces between the channel and the device itself. The purpose of the controller is to provide the proper protocol for the device and to convert that protocol to a standard protocol for use by the channel.

3. Writable Control Storage—which interfaces the channel, provides a source of Read/Write memory for the channel. The use of the Writable Control Storage is to customize an I/O processor for specific uses. The Writable Control Storage is loaded by special software instructions and may contain any program the user requires.

The main subassemblies common to all Class F I/O processors are the controller and channel, with the Writable Control Storage being an option.

Dedicated memory locations are associated with each I/O processor and provide main memory locations to transmit or receive control information required to initiate or terminate an I/O operation. The control information consists of:

1. Service Interrupt Vector Address
2. Input/Output Command Doubleword (IOCD) Address
3. Status Address
4. New Program Status Doubleword (PSD)
5. Old Program Status Doubleword (PSD)

A graphic representation of the I/O control words is shown in Figure 5-11.

Memory addresses are transferred to the channel when a Start I/O (SIO) or Write Channel Write Control Storage (WCWCS) instruction is executed by the CPU. Prior to the execution of the I/O instruction, the software stores the address of the first Input/Output Command Doubleword (IOCD) to be executed into the word indicated by adding 20 (decimal) to the contents of the Service Interrupt Vector (SIV). The word indicated is referred to as the Input/Output Command List Address (IOCLA).

The memory addressing method used for Class F I/O is real addressing. Real addressing is the capability to directly address any memory location within the 16 MB maximum capacity of the system without any address translation. This method of addressing differs from the method normally used by the software programmer, who relies on a hardware address conversion to transform the logical address to a real address in order to address memory locations greater than 512K bytes.
Figure 5-10. System Configuration with Class F I/O Processor
Figure 5-11. I/O Control Words (Class F)
When operating in the PSD mode, a set of special instructions augment or replace those used for the PSW mode of operation. The PSD I/O instructions include the following:

1. Start I/O (SIO)
2. Test I/O (TIO)
3. Halt I/O (HIO)
4. Stop I/O (STPIO)
5. Grab Controller (GRI0)
6. Reset Controller (RSCTL)
7. Reset Channel (RSCHNL)
8. Enable Channel WCS Load (ECWCS)
9. Write Channel WCS (WCWCS)
10. Enable Channel Interrupt (ECI)
11. Disable Channel Interrupt (DCI)
12. Activate Channel Interrupt (ACI)
13. Deactivate Channel Interrupt (DACI)

For all Class F I/O instructions, the logical channel and device addresses are specified by bits 16-31 of the instruction plus the contents of the General Purpose Register (GPR) specified by the instruction (if the GPR specified is nonzero). The channel will ignore the subaddress for operations that pertain only to the channel.

The Class F I/O instructions can be executed only when the CPU is in privileged mode and operating in the PSD mode.

**START I/O (SIO)**
The Start I/O initiates an I/O operation. If the necessary channel, subchannel or controller is available, the SIO is accepted and the CPU continues to the next sequential instruction. The channel/controller independently governs the I/O device specified by the instruction.

**TEST I/O (TIO)**
The Test I/O interrogates the current state of the channel, subchannel, controller and device and may be used to clear pending interrupt conditions.

**HALT I/O (HIO)**
The Halt I/O terminates a channel, controller, and/or device operation.

**ENABLE CHANNEL WCS LOAD (ECWCS)**
The Enable Channel WCS Load conditions the channel to have its WCS loaded.

**WRITE CHANNEL WCS (WCWCS)**
The Write Channel WCS is the second part of a two-instruction sequence and causes the specified channel's WCS to be loaded.

**ENABLE CHANNEL INTERRUPT (ECI)**
The Enable Channel Interrupt allows the channel to request interrupts from the CPU.
DISABLE CHANNEL INTERRUPT (DCI)

The Disable Channel Interrupt prohibits the channel from requesting an interrupt. Pending status conditions can only be cleared by the execution of a Start I/O, Test I/O, or Halt I/O if the channel is disabled.

ACTIVATE CHANNEL INTERRUPT (ACI)

The Activate Channel Interrupt causes the channel to actively contend for interrupt priority except that the channel never requests an interrupt. The instruction has no effect on pending status conditions except that it can be cleared by a Start I/O, Test I/O, or Halt I/O.

DEACTIVATE CHANNEL INTERRUPT (DACI)

The Deactivate Channel Interrupt causes the channel to suspend contention for interrupt priority. If an interrupt request is queued, the channel may then request interrupt. All DACI instruction abnormalities or I/O protocol violations are connected to the System Check Trap unless an initial channel nonpresent or inoperable condition is found.

RESET CHANNEL (RSCHNL)

The Reset Channel resets all activity in the channel. All requesting and pending conditions are cleared.

STOP I/O (STPIO)

The Stop I/O terminates the operation in the controller after the completion of the current IOCD. The termination will be orderly. The channel will suppress command and data chaining.

RESET CONTROLLER (RSCTL)

The Reset Controller resets a specific controller if the resetting channel maintains ownership. The reset is immediate.

GRAB CONTROLLER (GRIO)

The Grab Controller takes away control of a controller which is reserved to another channel. The grabbing channel is assigned as the reserving channel.

INPUT/OUTPUT COMMAND LIST ADDRESS

Successful execution of the SIO and WCWCS causes the CPU to transmit the Input/Output Command List Address (IOCLA) to the channel/controller. The IOCLA is located in main memory at locations specified by the service interrupt vector plus 16 (decimal). Each of the 16 channels has a corresponding service interrupt vector. The format for the IOCLA indicated by the contents of the service interrupt vector 11 is:

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

The real IOCLA is passed to the channel/controller on the data bus.

The address indicated in the IOCLA specifies the word address of the first IOCD to be executed. The IOCD format is shown in Figure 5-12.

The SIO is the only instruction that is able to cause the Channel/Controller to fetch an IOCD. One or more IOCDs create an Input/Output Command List (IOCL).

The command field specifies one of the following seven commands:

- Write
- Read
- Read Backward
- Control
- Sense
- Transfer in Channel
- Channel Control
If more than one IOCD is specified, the IOCDs are fetched sequentially except when Transfer in Channel (TIC) is specified. Search (compare) commands can cause the skipping of the next sequential IOCD if the condition becomes true (i.e., Search Equal, Search Low, or Search High). The channel or controller will then increment by 16 rather than 8.

The real data address specifies the starting address of the data area. The data address will be a byte address and the channel will internally align the information transferred to or from main memory. Exclusions to the byte alignment may be required by the lower priced channel(s) operating in Burst mode in high performance controllers.

The byte count specifies the number of bytes that are to be transferred to or from main memory. The actual number of memory transfers performed by the channel will be dependent upon the channel implementation.

**INPUT/OUTPUT COMMANDS**

**WRITE**

The Write command causes a Write (output) operation to the selected I/O device from the specified main memory address.

**READ**

The Read command causes a Read (input) operation from the selected I/O device to the specified main memory address.

**READ BACKWARD**

The Read Backward command causes a Read (input) operation from the selected I/O device to the specified main memory address in descending order.

**CONTROL**

The Control command causes control information to be passed to the selected device. A Control command may provide a data address and byte count for additional control information that may be stored in main memory.

Control information is device dependent and may instruct a magnetic tape to rewind or a printer to space a certain number of lines.

**SENSE**

The Sense command causes the storing of controller/device information in the specified location of main memory. One or more bytes of information will be transferred depending upon the device. The sense information provides additional device dependent information not provided in the status flags.

**TRANSFER IN CHANNEL**

The Transfer in Channel (TIC) command specifies the address of the next IOCD to be executed. The TIC command allows the programmer to change the sequence of the IOCDs executed. The IOCLA cannot specify a TIC as the first IOCD in a command list nor can a TIC specify another TIC command.

**CHANNEL CONTROL**

The Channel Control command causes the transfer of information to or from a specific location in main memory. One or more bytes of information will be transmitted or received from the channel. The channel control provides for the passing of information required to initialize all channels.

**INPUT/OUTPUT TERMINATION**

An I/O operation terminates when the channel, controller, and/or device indicates the end of an operation. All I/O operations accepted by the channel will always terminate with at least one termination status being presented to software.

An I/O operation can also fail to be accepted by the channel during I/O initiation. Conditions that prevent I/O initiation are: (1) channel or subchannel busy, (2) channel not operational or nonexistent, or (3) pending termination status from a previously initiated I/O operation.
BIT ASSIGNMENTS IN THE COMMAND ARE:

X X X X 0 0 0 0 CHANNEL CONTROL
M M M M 0 1 0 0 SENSE
X X X X 1 0 0 0 TRANSFER IN CHANNEL
M M M M 1 1 0 0 READ BACKWARD
M M M M M 0 1 WRITE
M M M M M 1 0 READ
M M M M M 1 1 CONTROL

FLAG BIT ASSIGNMENTS ARE:

1 0 0 0 0 0 DATA CHAIN (HOLDS OFF TERMINATION WHEN XFER CT = 0)
0 1 0 0 0 0 CMD CHAIN
0 0 1 0 0 0 SUPPRESS INCORRECT LENGTH
0 0 0 1 0 0 SKIP
0 0 0 0 1 0 POST PROGRAM CONTROLLED INTERRUPT

C - BIT ASSIGNMENTS ARE:

BIT 30 BIT 31

0 0 BYTE 0 OR FULLWORD
0 1 BYTE 1 OR FIRST HALFWORD
1 0 BYTE 2 OR DOUBLEWORD*
1 1 BYTE 3 OR SECOND HALFWORD

*IF DOUBLEWORD IS INDICATED TO A CHANNEL, AMBIGUOUS RESULTS MAY OCCUR.

Figure 5-12. Input/Output Command Doubleword (IOCD)
I/O initiation failures are reported to software by the setting of condition codes and, where applicable, the storing of status.

The status words are maintained and stored by the channel. The address of the status words is transmitted to the CPU when an interrupt is acknowledged or when another I/O instruction is executed. The status words contain information relating to the execution of the last IOCD or from an asynchronous condition requiring software notification (i.e., tape loaded, disc pack mounted). The status words are in the following format:

**STATUS WORD 1**

<table>
<thead>
<tr>
<th>SUBADDRESS</th>
<th>REAL IOCD ADDRESS</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**STATUS WORD 2**

<table>
<thead>
<tr>
<th>STATUS FLAGS</th>
<th>RESIDUAL BYTE COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
</tr>
</tbody>
</table>

The status flags contain termination information pertaining to both the channel and controller. IOMs that function as integrated channel controllers will maintain both sections.

The address of the status is stored in main memory and can be located by adding 2010 = 1416 to the contents of the service interrupt vector.

Input/Output interrupts can be caused by a response to a probe instruction (i.e., TIO) by the termination of an I/O operation, by operator intervention at the I/O device, or when a post program controlled interrupt is requested by an IOCD. The associated I/O interrupt causes the status address, and the current PSD to be stored in the memory location specified by the service interrupt address. The new PSD (specified by the contents of the service interrupt vector +8) is then loaded.

An I/O interrupt can be caused by the device, controller, or channel. If a channel or controller has multiple I/O interrupt requests pending, it establishes a priority sequence for them before initiating an I/O interrupt request to the CPU. This priority sequence is maintained when the channel stores the status and reports the status address to the CPU.

The mode in which the channel operates during the software interrupt processing is determined by the mode setting of the channel and the implementation of the channel. The software may use bits 48 and 49 of the new PSD to select one of two options: Unblocked or Blocked operation.

Unblocked operation specifies that the CPU, upon receipt of an interrupt, causes the channel to go active and block all interrupts of a lower priority. The channel services the interrupt, and the software in turn issues a DACI or BRI command to restore the interrupt processing.
Blocking specifies that the CPU, upon receipt of an interrupt, causes the channel to deactivate. The CPU blocks all incoming interrupts and services the pending interrupt. The software in turn issues an UEI command or a BRI, LPSD, or LPSDCM to the CPU, thereby restoring interrupt processing. The target PSD of the BRI, LPSD, or LPSDCM instruction should specify Unblocked operation in bits 48 and 49.
SECTION VI
INSTRUCTION REPertoire

| INTRODUCTION | This section contains the description of each computer instruction. The following paragraphs list the standard information given with each instruction. |
| MNEMONIC | A two- to six-letter symbolic representation of the instruction name accepted by the assembler program. |
| INSTRUCTION NAME | A title that indicates the function performed by the instruction. |
| OPERATION CODE | The Operation Code for each instruction is given in left-justified hexadecimal format. This format is presented in a 16-bit skeleton form and takes into consideration the Augmenting Code and the format bit used with byte-oriented instructions. |
| FORMAT | A 16- or 32-bit machine language representation of the instruction. The operation code and all other fixed bits are given in their binary value. |
| DEFINITION | The function performed by the instruction is described following the instruction format. All registers or memory locations which are modified are defined. Special considerations are given in notes following the basic functional description. |
| SUMMARY EXPRESSION | This expression supplements the verbal description of most instructions by symbolically showing the function performed by execution of the instruction. The symbols are defined in Table 6-1. The abbreviations are listed in Table 6-2. Summary expression examples are given below: |

(s_{24-31}) \rightarrow (d_{24-31})

The contents of bits 24-31 of GPR s is replaced with the contents of bits 24-31 of GPR d.

[zeros_{0-23}, \text{byte operand}] \rightarrow (d)

The byte operand is appended with zeros in positions 0-23 and the resulting word replaces the contents of GPR d.

(m), (m+1) is a doubleword effective memory address. (d), (d+1) is a doubleword even/odd GPR pair. |
<p>| ASSEMBLY CODING CONVENTIONS | A symbolic representation of the assembler coding format. Table 6-2 lists all abbreviations and symbols used in the operand coding format. |</p>
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>−</td>
<td>Logical NOT function, for example (☉) is the ones complement of the GPR number ( s ). Replaces; the data to the left of the symbol replaces data to the right. For example, (( s ) \rightarrow ( d )) means the contents of GPR number ( s ) replaces the contents of GPR number ( d ).</td>
</tr>
<tr>
<td>→</td>
<td>The register number or memory address is incremented by one register number or one memory word.</td>
</tr>
<tr>
<td>+1</td>
<td>Greater Than.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Lesser Than.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Algebraic Addition.</td>
</tr>
<tr>
<td>+</td>
<td>Algebraic Subtraction.</td>
</tr>
<tr>
<td>−</td>
<td>Algebraic Multiplication.</td>
</tr>
<tr>
<td>/</td>
<td>Algebraic Division.</td>
</tr>
<tr>
<td>&amp;</td>
<td>Logical AND.</td>
</tr>
<tr>
<td>( B_{m-n} )</td>
<td>Bits ( m ) through ( n ) of a computer word.</td>
</tr>
<tr>
<td>( B_n )</td>
<td>Bit ( n ) of a computer word where ( B_0 ) always refers to the most significant bit of a computer word (the letter ( n ) is also used to indicate scaling; e.g., ( 1_{15} ) indicates a 1 scaled at bit position 15).</td>
</tr>
<tr>
<td>( C_{C_n} )</td>
<td>Comparison Symbol.</td>
</tr>
<tr>
<td>:</td>
<td>Concatenation Sign (e.g., ( R, R+1 ) indicates a doubleword consisting of ( R ) and ( R+1 ), where ( R ) must be an even numbered register).</td>
</tr>
<tr>
<td>EA</td>
<td>Effective Address of an operand or instruction stored in memory.</td>
</tr>
<tr>
<td>EBA</td>
<td>Effective Byte Address.</td>
</tr>
<tr>
<td>EBL</td>
<td>Eight-Bit Location in memory specified by the EBA.</td>
</tr>
<tr>
<td>EDA</td>
<td>Effective Doubleword Address.</td>
</tr>
<tr>
<td>EDL</td>
<td>Sixty-four bit location in memory consisting of an even numbered word location and the next higher word location, specified by the EDA.</td>
</tr>
<tr>
<td>EHA</td>
<td>Effective Halfword Address.</td>
</tr>
<tr>
<td>EHL</td>
<td>Sixteen-bit location in memory specified by the EHA.</td>
</tr>
<tr>
<td>EWA</td>
<td>Effective Word Address.</td>
</tr>
</tbody>
</table>
Table 6-1. Symbol Definitions (Cont'd)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>EWL</td>
<td>Thirty-two bit location in memory specified by the EWA.</td>
</tr>
<tr>
<td>I</td>
<td>Indirect Address bit.</td>
</tr>
<tr>
<td>ISI</td>
<td>Is Set If, used to indicate conditions which set referenced bit locations.</td>
</tr>
<tr>
<td>IW</td>
<td>Instruction Word.</td>
</tr>
<tr>
<td>( )</td>
<td>Contents of.</td>
</tr>
<tr>
<td>+</td>
<td>Exclusive OR.</td>
</tr>
<tr>
<td>MIDL</td>
<td>Memory Image Descriptor List.</td>
</tr>
<tr>
<td>PSDR</td>
<td>Program Status Doubleword Registers.</td>
</tr>
<tr>
<td>PSWR</td>
<td>Program Status Word Register.</td>
</tr>
<tr>
<td>R</td>
<td>General Register 0-7 (R0-R7).</td>
</tr>
<tr>
<td>R&lt;sub&gt;m-n&lt;/sub&gt;</td>
<td>Bits m through n of General Register R.</td>
</tr>
<tr>
<td>R&lt;sub&gt;n&lt;/sub&gt;</td>
<td>Bit n of General Register R.</td>
</tr>
<tr>
<td>SBL</td>
<td>Specified Bit Location with a byte (used as a subscript to designate that the bit location is specified in the Instruction Word).</td>
</tr>
<tr>
<td>SCC</td>
<td>Sets Condition Code bits.</td>
</tr>
<tr>
<td>SE</td>
<td>Used as a subscript to denote a sign extended halfword.</td>
</tr>
<tr>
<td>v</td>
<td>Logical OR.</td>
</tr>
<tr>
<td>X</td>
<td>Index Register:</td>
</tr>
<tr>
<td></td>
<td>X Value</td>
</tr>
<tr>
<td></td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>11</td>
</tr>
<tr>
<td>-Y</td>
<td>Twos complement of Y.</td>
</tr>
<tr>
<td>Y</td>
<td>Ones completion of Y, logical NOT function.</td>
</tr>
</tbody>
</table>
An interpretation of the resulting 4-bit Condition Code in the Program Status Doubleword register. This code defines the result of the operation. The circumstances in which these Condition Codes are set (i.e., equal to One) are noted with each instruction.

Included in the examples with many of the instructions are memory and register contents before and after execution.

The 32/70 Series instruction mnemonics follow a very simple format. The basic types are:

- **L**: load or LM: load masked
- **ST**: store or STM: store masked
- **AD**: add
- **ADM**: add memory to register
- **ARM**: add register to memory
- **SU**: subtract
- **SUM**: subtract memory from register
- **MP**: multiply
- **DV**: divide
- **ADF**: floating-point arithmetic
- **SUF**:
- **MPF**:
- **DVF**:
- **B**: branch
- **AN**: AND
- **OR**: logical OR
- **EO**: exclusive OR
- **C**: compare

These basic mnemonics are then augmented to define the operand data type. (A special set of instructions are provided for bit manipulation.) The five basic data types are:

- **B**: Byte (8 bits)
- **H**: Halfword (16 bits)
- **W**: Word (32 bits)
- **D**: Doubleword (64 bits)
- **I**: Immediate (16 bits)

Therefore, the resulting instruction mnemonics have the form:

- **LB**: Load Byte
- **LMH**: Load Masked Halfword
- **STMW**: Store Masked Word
- **ADI**: Add Immediate to Register
- **SUMD**: Subtract Memory Doubleword

A complete summary of the 32/70 Series instructions is presented in the Appendix of this manual.
The basic assembler coding format for memory reference instructions is:

\[
\text{XXXXXX} \quad |s| \quad , \quad *m, \quad x
\]

which translates to

<table>
<thead>
<tr>
<th>XXXXXX</th>
<th>Instruction mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>s</td>
</tr>
<tr>
<td>(</td>
<td>d</td>
</tr>
<tr>
<td>m</td>
<td>Memory operand</td>
</tr>
<tr>
<td>x</td>
<td>Indexed by register number x</td>
</tr>
</tbody>
</table>

Nonmemory reference instruction coding is similar to the memory reference format. Table 6-2 lists all codes used in defining the Assembler coding formats.

Each instruction definition includes the following information:

- **Instruction Name**: The full name of the instruction.
- **Op Code**: The four most significant hexadecimal digits of the instruction word are listed. Additional bits in the op code are set when the instruction is coded to address a General Purpose Register (GPR), for indirect addressing, or for byte addressing.
- **Assembler Coding Format**: The coding format used by the 32 Macro Assembler. Table 6-2 includes all the abbreviations and symbols used in the operand coding format.
- **Instruction Definition**: A definition of the operation performed by executing the instruction.
- **Summary Expression**: A symbolic or graphic description of the operation performed by the instruction. Summary expressions use the same abbreviations used in the assembler coding format, Table 6-2. In addition, Table 6-1 lists the codes and symbols used in the summary expressions.
- **Condition Codes**: The Condition Codes are set based on the results obtained by executing an instruction. The circumstances in which these condition codes are set (i.e., equal to one) are noted with each instruction.
Table 6-2. Assembler Coding Symbols

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capital Letters</td>
<td>Instruction Mnemonic</td>
</tr>
<tr>
<td>b</td>
<td>Bit number (0-31) in a General Purpose Register</td>
</tr>
<tr>
<td>c</td>
<td>Bit number (0-7) within a byte</td>
</tr>
<tr>
<td>d</td>
<td>Destination General Purpose Register number (0-7)</td>
</tr>
<tr>
<td>f</td>
<td>Function</td>
</tr>
<tr>
<td>m</td>
<td>Operand Memory Address</td>
</tr>
<tr>
<td>n</td>
<td>Device Address</td>
</tr>
<tr>
<td>s</td>
<td>Source General Purpose Register number (0-7)</td>
</tr>
<tr>
<td>v</td>
<td>Value for Immediate Operands, number of shifts, etc.</td>
</tr>
<tr>
<td>x</td>
<td>Index register number 1, 2, or 3. Optional</td>
</tr>
<tr>
<td>*</td>
<td>Indirect Addressing. Optional</td>
</tr>
<tr>
<td>,</td>
<td>Assembler Syntax</td>
</tr>
<tr>
<td>z</td>
<td>Special register field for instructions requiring three register fields</td>
</tr>
</tbody>
</table>
LOAD/STORE INSTRUCTIONS

GENERAL DESCRIPTION

The Load/Store instruction group is used to manipulate data between memory and General Purpose Registers. In general, Load instructions transfer operands from specified memory locations to General Purpose Registers; Store instructions transfer data from General Purpose Registers to specified memory locations. Provisions have also been made to Mask or Clear the contents of General Purpose Registers, memory bytes, halfwords, words, or doublewords during instruction execution.

INSTRUCTION FORMATS

MEMORY REFERENCE

The Load/Store instructions use the following three formats:

The format for most memory reference instructions is defined below. These instructions contain two addresses: a register number R and a memory address with a 20-bit format.

<table>
<thead>
<tr>
<th>OPCODE</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>F</th>
<th>WA</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 0-5 define the Operation Code.

Bits 6-8 designate a General Purpose Register address (0-7).

Bits 9-10 designate one of three General Purpose Registers to be used as an index register.

X = 00 designates that no indexing operation is to be performed.

X = 01 designates the use of R1 for indexing.

X = 10 designates the use of R2 for indexing.

X = 11 designates the use of R3 for indexing.

Bit 11 designates whether an indirect addressing operation is to be performed.

I = 0 designates that no indirect addressing operation is to be performed.

I = 1 designates that an indirect addressing operation is to be performed.

Bits 12-31 specify the address of the operand when the X and I fields are equal to zero.
In immediate operand instructions, the right halfword of the instruction contains the 16-bit operand value. The format for these instructions is given below.

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>AUG CODE</th>
<th>OPERAND VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Bits 0-5 define the Operation Code.

Bits 6-8 designate a General Purpose Register address (0-7).

Bits 9-12 unassigned.


Bits 16-31 contain the 16-bit operand value.

Arithmetic operands are assumed to be represented in two's complement with the sign in bit 16.

Interregister instructions are halfword instructions and as such may be stored in either the left or right half of a memory word. The format for interregister instructions is given below.

<table>
<thead>
<tr>
<th>LEFT HALFWORD</th>
<th>RIGHT HALFWORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP CODE</td>
<td>R</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Left Halfword

Bits 0-5 16-21 define the Operation Code.

Bits 6-8 22-24 designate the register to contain the result of the operation.

Bits 9-11 25-27 designate the register which contains the source operand.

A Condition Code is set during most Load instructions to indicate if the operand being transferred is greater than, less than, or equal to zero. Arithmetic exceptions are also reflected by the Condition Code results. All Store instructions leave the Condition Code unchanged.

Figure 6-1 depicts the positioning of information for transfer from memory to any General Purpose Register.

Figure 6-1. Positioning of Information Transferred Between Memory and Registers
**LOAD BYTE**

**AC08**

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>T</th>
<th><strong>BYTE OPERAND ADDRESS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

**DEFINITION**

The byte in memory specified by the Effective Byte Address (EBA) is accessed and transferred to bit positions 24-31 of the General Purpose Register (GPR) specified by R. Bit positions 0-23 of the GPR specified by R are cleared to zeros.

**SUMMARY**

\[ (E_{BL}) \rightarrow R_{24-31} \]

\[ 0 \rightarrow R_{0-23} \]

**CONDITION CODE**

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always zero</td>
<td>ISI ( R_{0-31} ) is greater than zero</td>
<td>Always zero</td>
<td>ISI ( R_{0-31} ) is equal to zero</td>
</tr>
</tbody>
</table>

**RESULTS**

**EXAMPLE 1**

Memory Location: \[ 01000 \]

Hex Instruction: \[ AC \ 88 \ 11 \ 01 \ (R=1, \ X=0, \ I=0) \]

Assembly Language Coding: \[ LB \ 1,X'1101' \]

Before Execution:

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR1</th>
<th>Memory Byte 01101</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001000</td>
<td>517CD092</td>
<td>B6</td>
</tr>
</tbody>
</table>

After Execution:

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR1</th>
<th>Memory Byte 01101</th>
</tr>
</thead>
<tbody>
<tr>
<td>20001004</td>
<td>000000B6</td>
<td>B6</td>
</tr>
</tbody>
</table>

Note: The contents of memory byte 01101 are transferred to bits 24-31 of GPR1, bits 0-23 of GPR1 are cleared. CC2 is set because the contents of GPR1 are greater than zero.

**EXAMPLE 2**

Memory Location: \[ 01000 \]

Hex Instruction: \[ AD \ 28 \ 14 \ 00 \ (R=2, \ X=1, \ I=0) \]

Assembly Language Coding: \[ LB \ 2,X'1400',1 \]

Before Execution:

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR1</th>
<th>GPR2</th>
<th>Memory Byte 01603</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001000</td>
<td>00000203</td>
<td>12345678</td>
<td>A1</td>
</tr>
</tbody>
</table>

After Execution:

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR1</th>
<th>GPR2</th>
<th>Memory Byte 01603</th>
</tr>
</thead>
<tbody>
<tr>
<td>20001004</td>
<td>00000203</td>
<td>000000A1</td>
<td>A1</td>
</tr>
</tbody>
</table>

Note: The contents of memory byte 01603 are transferred to bits 24-31 of GPR2. Bits 0-23 are cleared, and CC2 is set.
LOAD HALFWORD

ACOO

---

**DEFINITION**
The halfword in memory specified by the Effective Halfword Address (EHA) is accessed and the sign bit (bit 16) is extended left 16 bit positions to form a word. This word is transferred to the GPR specified by R.

**SUMMARY**
(EHL)SE → R

**CONDITION CODE RESULTS**
- **CC1:** Always zero
- **CC2:** ISI $R_{0-31}$ is greater than zero
- **CC3:** ISI $R_{0-31}$ is less than zero
- **CC4:** ISI $R_{0-31}$ is equal to zero

**EXAMPLE**
- **Memory Location:** 00408
- **Hex Instruction:** AE 00 05 03 ($R=4$, $X=0$, $I=0$)
- **Assembly Language Coding:** LH 4,'502'

<table>
<thead>
<tr>
<th>Before Execution</th>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR</td>
<td>PSWR</td>
</tr>
<tr>
<td>GPR4</td>
<td>GPR4</td>
</tr>
<tr>
<td>10000408</td>
<td>1000040C</td>
</tr>
<tr>
<td>5C00D34A</td>
<td>FFFF930C</td>
</tr>
<tr>
<td>Memory Halfword 00502</td>
<td>Memory Halfword 00502</td>
</tr>
<tr>
<td>930C</td>
<td>930C</td>
</tr>
</tbody>
</table>

**Note**
The contents of memory halfword 00502 are transferred to bits 16-31 of GPR4. Bits 0-15 of GPR4 are set by the sign extension, and CC3 is set.
**LOAD WORD**

**ACO0**

<table>
<thead>
<tr>
<th>101011</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>0</th>
<th>WORD OPERAND ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**DEFINITION**
The word in memory specified by the Effective Word Address (EWA) is accessed and transferred to the GPR specified by R.

**SUMMARY**

\[(EWL) \rightarrow R\]

**CONDITION CODE RESULTS**
- CC1: Always zero
- CC2: ISI R0-31 is greater than zero
- CC3: ISI R0-31 is less than zero
- CC4: ISI R0-31 is equal to zero

**EXAMPLE**
- **Memory Location:** 02390
- **Hex Instruction:** AF 80 27 A4 (R=7, X=0, I=0)
- **Assembly Language Coding:** LW 7,X'27A4'

<table>
<thead>
<tr>
<th>Before Execution</th>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR 00002390</td>
<td>PSWR 20002394</td>
</tr>
<tr>
<td>GPR7 0056876A</td>
<td>GPR7 4D61A28C</td>
</tr>
<tr>
<td>Memory Word 027A4</td>
<td>Memory Word 027A4</td>
</tr>
</tbody>
</table>

**Note**
The contents from memory word 027A4 are transferred to GPR7, and CC2 is set.
LOAD DOUBLEWORD
ACOO

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>0</th>
<th>WORD OPERAND ADDRESS</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>0 0</td>
</tr>
</tbody>
</table>

DEFINITION
The doubleword in memory specified by the Effective Doubleword Address (EDA) is accessed and transferred to the GPR specified by R and R+1. R+1 is the GPR one greater than specified by R. The least significant memory word is accessed first and transferred to the GPR specified by R+1. The most significant memory word is accessed last and transferred to the GPR specified by R.

NOTE
The GPR specified by R must have an even address.

SUMMARY
\[(EWL+1) \rightarrow R+1\]

\[(EWL) \rightarrow R\]

EXPRESSION

CONDITION CODE

RESULTS

CC1: Always zero
CC2: ISI (R,R+1) is greater than zero
CC3: ISI (R,R+1) is less than zero
CC4: ISI (R,R+1) is equal to zero

EXAMPLE
Memory Location: 281C4
Hex Instruction: AF 02 8B 7A (R=6, X=0, I=0)
Assembly Language Coding: LD 6,X'28B78'

Before Execution
PSWR 400281C4
GPR6 03F609C3
GPR7 39B8510E
Memory Word 28B78 03F609C3
137F8CA2
Memory Word 28B7C

After Execution
PSWR 100281C8
GPR6 F05B169A
GPR7 137F8CA2
Memory Word 28B78 F05B169A
137F8CA2
Memory Word 28B7C

Note
The contents of memory word 28B78 are transferred to GPR6 and the contents of memory word 28B7C are transferred to GPR7. CC3 is set.
LOAD MASKED BYTE

101100 R X I I
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

DEFINITION
The byte in memory specified by the Effective Byte Address (EBA) is accessed and masked (Logical AND function) with the least significant byte (bits 24-31) of the Mask register (R4). The result of the mask operation is transferred to bit positions 24-31 of the GPR specified by R. Bit positions 0-23 of the GPR specified by R are cleared to zeros.

SUMMARY
\[(EBA) \& (R4_{24-31}) \rightarrow (R_{24-31})\]
\[0 \rightarrow R_{0-23}\]

CONDITION CODE RESULTS
CC1: Always zero
CC2: ISI R0-31 is greater than zero
CC3: Always zero
CC4: ISI R0-31 is equal to zero

EXAMPLE
Memory Location: 00900
Hex Instruction: B0 88 00 A3 (R=1, X=0, I=0)
Assembly Language Coding: LMB 1,X'A3'

Before Execution
PSWR 00000900
GPR1 AA368980
GPR4 000000F0
Memory Byte 000A3

After Execution
PSWR 20000904
GPR1 00000020
GPR4 000000F0
Memory Byte 000A3

Note
The contents of memory byte 000A3 are logically ANDed with the rightmost byte of GPR4, and the result is transferred to bits 24-31 of GPR1. Bits 0-23 of GPR1 are cleared, and CC2 is set.
LOAD MASKED HALFWORD

BO00

<table>
<thead>
<tr>
<th>1 0 1</th>
<th>1 0</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>0</th>
<th>HALFWORD OPERAND ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

DEFINITION
The halfword in memory specified by the Effective Halfword Address (EHA) is accessed, and the sign bit (bit 16) is extended 16 bit positions to the left to form a word. This word is then masked (Logical AND Function) with the contents of the Mask register (R4). The resulting word is transferred to the GPR specified by R.

SUMMARY

\[(EHL)_{SE} \& (R4) \rightarrow R\]

CONDITION CODE

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always zero</td>
<td>ISI $R_{0-31}$ is greater than zero</td>
<td>ISI $R_{0-31}$ is less than zero</td>
<td>ISI $R_{0-31}$ is equal to zero</td>
</tr>
</tbody>
</table>

RESULTS

| Memory Location: | 00300 |
| Hex Instruction: | B2 80 03 A1 (R=5, X=0, I=0) |
| Assembly Language Coding: | LMH $S_{X'}3A0$ |

EXAMPLE

<table>
<thead>
<tr>
<th>Before Execution</th>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR</td>
<td>GPR4</td>
</tr>
<tr>
<td>08000300</td>
<td>OFF00FF0</td>
</tr>
<tr>
<td>PSWR</td>
<td>GPR4</td>
</tr>
<tr>
<td>20000304</td>
<td>OFF00FF0</td>
</tr>
</tbody>
</table>

Note
The contents of memory halfword 003A0 are accessed, the sign is extended 16 bit positions, the result is logically ANDed with the contents of GPR4, and the final result is transferred to GPR5. CC2 is set.
LOAD MASKED WORD

BO00

<table>
<thead>
<tr>
<th>101100</th>
<th>R</th>
<th>X</th>
<th>I0</th>
<th>WORD OPERAND ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

DEFINITION
The word in memory specified by the Effective Word Address (EWA) is accessed and masked (Logical AND Function) with the contents of the Mask register (R4). The resulting word is transferred to the GPR specified by R.

SUMMARY
(EWL)&(R4) → R

EXPRESSION

CONDITION CODE
CC1: Always zero
CC2: ISI R0-31 is greater than zero
CC3: ISI R0-31 is less than zero
CC4: ISI R0-31 is equal to zero

RESULTS

EXAMPLE
Memory Location: 00F00
Hex Instruction: B3 80 0F FC (R=7, X=0, I=0)
Assembly Language Coding: LMW 7,X'FFC'

Before Execution
PSWR: 00000000
GPR4: FF000000
GPR7: 12345678
Memory Word 00FFC

After Execution
PSWR: 80000004
GPR4: FF000007C
GPR7: 890000068
Memory Word 00FFC

Note
The contents of memory word 00FFC are ANDed with the contents of GPR4. The result is transferred to GPR7, and CC3 is set.
LOAD MASKED DOUBLEWORD

Bo00

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

**DEFINITION**
The doubleword in memory specified by the Effective Doubleword Address (EDA) is accessed, and the contents of each word are masked (Logical AND Function) with the contents of the Mask register (R4). The least significant memory word is masked first. The resulting masked doubleword is transferred to the GPR specified by R and R+1. R+1 is the GPR one greater than specified by R.

**SUMMARY**

- **CONDITION CODE**
  - CC1: Always zero
  - CC2: ISI (R,R+1) is greater than zero
  - CC3: ISI (R,R+1) is less than zero
  - CC4: ISI (R,R+1) is equal to zero

**RESULTS**

- (EWL+1) & (R4) → R+1
- (EWL) & (R4) → R

**EXAMPLE**

- **Hex Instruction:** B3 00 02 F2 (R=6, X=0, I=0)
- **Assembly Language Coding:** LMD 6,'2FO'
- **Before Execution**
  - PSWR: 00000200
  - GPR4: 3F3F3F3F
  - Memory Word: 002FO
  - AE69D10C

- **After Execution**
  - PSWR: 20000204
  - GPR4: 3F3F3F3F
  - Memory Word: 002FO
  - AE69D10C

**Note**
The contents of memory word 002F4 are ANDed with the contents of GPR4, and the result is transferred to GPR6. CC2 is set.
LOAD NEGATIVE BYTE
B408

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

DEFINITION
The byte in memory specified by the Effective Byte Address (EBA) is accessed, and 24 zeros are appended to the most significant end to form a word. The two's complement of this word is then taken and transferred to the GPR specified by R.

SUMMARY
- \([00-23, \text{EBL}] \rightarrow R\)

CONDITION CODE
- CC1: Always zero
- CC2: Always zero
- CC3: \(\text{ISI} R_{0-31}\) is less than zero
- CC4: \(\text{ISI} R_{0-31}\) is equal to zero

EXAMPLE
Memory Location: \(\text{OD000}\)
Hex Instruction: \(\text{B4 B8 D1 02} \ (R=1, X=1, I=0)\)
Assembly Language Coding: \(\text{LNB 1,X'D102}\)

| Before Execution | | After Execution |
|------------------|------------------|
| PSWR             | 00000000          | 10000004          |
| GPR1             | 00000000          | FFFFFFFC6         |
| Memory Byte      | OD102             | OD102             |
|                  | 3A                | 3A                |

Note
The contents of memory byte OD102 are prefixed with 24 zeros to form a word; the result is negated and transferred to GPR1. CC3 is set.
LOAD NEGATIVE HALFWORD

B400

<table>
<thead>
<tr>
<th>1 0 1 1 0 R</th>
<th>X</th>
<th>I</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DEFINITION**
The halfword in memory specified by the Effective Halfword Address (EHA) is accessed, and the sign bit (bit 16) is extended 16 bit positions to the left to form a word. The two's complement of this word is then transferred to the GPR specified by R.

**SUMMARY**
- \([- [(EHL)_{SE}] \rightarrow R\]

**CONDITION CODE RESULTS**
- CC1: Always zero
- CC2: ISI R0-31 is greater than zero
- CC3: ISI R0-31 is less than zero
- CC4: ISI R0-31 is equal to zero

**EXAMPLE**
- Memory Location: 08000
- Hex instruction: B6 00 84 03 (R=4, X=0, I=0)
- Assembly Language Coding: LNH 4,X'8402'

**Before Execution**
- PSWR 400008000
- GPR4 12345678
- Memory Halfword 08402

**After Execution**
- PSWR 200080004
- GPR4 000069F4
- Memory Halfword 08402

**Note**
The contents of memory halfword 08402 are sign extended and negated. The result is transferred to GPR4, and CC2 is set.
**LOAD NEGATIVE WORD**

**B400**

**DEFINITION**
The word in memory specified by the Effective Word Address (EWA) is accessed, and its two's complement is transferred to the GPR specified by R.

**SUMMARY**

\[-(EWA) \rightarrow R\]

**CONDITION CODE RESULTS**

CC1: ISI Arithmetic Exception
CC2: ISI RO-31 is greater than zero
CC3: ISI RO-31 is less than zero
CC4: ISI RO-31 is equal to zero

**EXAMPLE**

<table>
<thead>
<tr>
<th>Before Execution</th>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR</td>
<td>10000504</td>
</tr>
<tr>
<td>GPR5</td>
<td>E7A1F28A</td>
</tr>
<tr>
<td>Memory Word</td>
<td>185E0076</td>
</tr>
</tbody>
</table>

**Notes:** The contents of memory word 006C8 are negated and transferred to GPR5, and CC3 is set.
LOAD NEGATIVE DOUBLEWORD

B400

<table>
<thead>
<tr>
<th>1 0 1 1 0 1</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>O</th>
<th>DOUBLEWORD OPERAND ADDRESS</th>
<th>0 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

DEFINITION
The doubleword in memory specified by the Effective Doubleword Address (EDA) is accessed and its two's complement is formed. The least significant memory word is complemented first and the result is transferred to the GPR specified by R+1. R+1 is the GPR one greater than specified by R. The most significant memory word is complemented, and the result is transferred to the GPR specified by R1.

SUMMARY
-(EDA)→R,R+1

CONDITION CODE
CC1: ISI Arithmetic Exception
CC2: ISI (R,R+1) is greater than zero
CC3: ISI (R,R+1) is less than zero
CC4: ISI (R,R+1) is equal to zero

RESULTS

EXAMPLE
Memory Location: 02344
Hex Instruction: B5 00 24 A2 (R=2, X=0, I=0)
Assembly Language Coding: LND 2,X'24A0'

Before Execution
PSWR: 00002344
GPR2: 01234567
GPR3: 89ABCDEF
Memory Word 024A0: 00000000

After Execution
PSWR: 10002348
GPR2: FFFFFFFF
GPR3: FFFFFFFF
Memory Word 024A0: 00000000
Memory Word 024A4: 00000001

Note
The doubleword obtained from the contents of memory words 024A0 and 024A4 is negated, and the result is transferred to GPR2 and GPR3. CC3 is set.
**LOAD IMMEDIATE**

*d*,v

C800

---

**DEFINITION**
The halfword immediate operand in the Instruction Word (IW) is sign-extended (bit 16 extended 16 positions to the left) to form a word. This word is transferred to the GPR specified by R.

**SUMMARY**

(IW,16-31)SE → R

**EXPRESSION**

**CONDITION CODE RESULTS**

CC1: Always zero
CC2: ISI (R0-31) is greater than zero
CC3: ISI (R0-31) is less than zero
CC4: ISI (R0-31) is equal to zero

**EXAMPLE**

Memory Location: 0630C
Hex Instruction: C8 80 FF FB (R=1)
Assembly Language Coding: LI 1,-5

Before Execution

PSWR: GPR1
0000630C 12345678

After Execution

PSWR: GPR1
10006310 FFFFFFFFB

**Note**
The halfword operand is sign-extended and the result is transferred to GPR1. CC3 is set.
LOAD EFFECTIVE ADDRESS

\[ d,*m,x \]

The effective address (bit 12-31) of the LEA instruction is generated in the same manner as in all other memory reference instructions and then is transferred to bit positions 12-31 of the GPR specified by R.

In PSD mode or PSW mode extended, bits 2-7 are cleared and bits 8-31 indicate results of EA.

Notes

1. If \(1=X=0\), the entire 32-bit Instruction Word is transferred to the GPR specified by R. (512 KB mode only)

2. If \(1=0\) and \(X=0\), bit positions 0-11 of the GPR specified by R will contain the sum of bit positions 0-11 of the Instruction Word and bit positions 0-11 of the index register specified by X. (512 KB mode only)

3. If \(1=1\), bit positions 0-11 of the GPR specified by R will contain the sum of bit positions 0-11 of the last word of the indirect chain and bit positions 0-11 of the index register specified (if any) in the last word of the indirect chain. (512 KB mode only)

4. In cases 2 and 3 above, an additional bit may be added to bit position 11 of the GPR specified by R as a result of overflow in the sum of the address and the index values. (512 KB mode only)

SUMMARY

\[ EA \rightarrow R_{12-31} \]

CONDITION CODE RESULTS

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>No change</td>
<td>No change</td>
<td>No change</td>
<td>No change</td>
</tr>
</tbody>
</table>

EXAMPLE

<table>
<thead>
<tr>
<th>Memory Location:</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex Instruction:</td>
<td>DO 804000 (R=1, X=I=0)</td>
</tr>
<tr>
<td>Assembly Language Codings:</td>
<td>LEA 1,X'4000'</td>
</tr>
</tbody>
</table>

Before Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR1</th>
</tr>
</thead>
<tbody>
<tr>
<td>08001000</td>
<td>00000000</td>
</tr>
</tbody>
</table>

Memory Word 4000

AC881203

After Execution (PSD Mode)

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR1</th>
</tr>
</thead>
<tbody>
<tr>
<td>08001004</td>
<td>D0804000</td>
</tr>
</tbody>
</table>

Memory Word 4000

AC881203

AC881203

AC881203
LOAD EFFECTIVE ADDRESS REAL

8000

<table>
<thead>
<tr>
<th>ZERO</th>
<th>F</th>
<th>EFFECTIVE ADDRESS</th>
<th>C</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td></td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td></td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td></td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td></td>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td></td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>28</td>
<td>29</td>
<td></td>
<td>30</td>
<td>31</td>
</tr>
</tbody>
</table>

DEFINITION
This instruction causes the Effective Real (nonmapped) Address of the referenced operand to be transferred to bit positions 7-31 of the GPR specified by R.

NOTE
The format of the 25-bit Effective Real Address transferred to the GPR is as follows:

SUMMARY
ERA → R7-31

0 → R0-6

CONDITION CODE
CC1: No change
CC2: No change
CC3: No change
CC4: No change

RESULTS
Assembly Language Coding: LEAR d,*m,x

NOTES
1. Privileged Instruction
2. Attempt to execute in PSW mode will result in an undefined instruction trap.
3. This instruction may not be the target of an execute instruction.
LOAD ADDRESS

3400

LA d,*m,x

DEFINITION

Loads the Effective Address (EA) into R_d. Bits 0-7 are cleared in R_d. Bits 8-11 receive the results of Extended Indexing (if active). Bit 12 is the F-bit if 512 KB mode and is an Effective Address (EA) bit if in 512 KB Extended mode.

CONDITION CODE

CC1: No change
CC2: No change
CC3: No change
CC4: No change

RESULTS

Assembly Language Coding: LA d,*m,x
LOAD FILE
CC00

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>OPERAND ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>O</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>30</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DEFINITION**
This instruction is used to load from one to eight GPR's. The word in memory specified by the Effective Word Address (EWA) in the Instruction Word is accessed and transferred to the GPR specified by R. Next, the EWA and the GPR address are incremented. The next sequential memory word is then transferred to the next sequential GPR. Successive transfers continue until GPR7 is loaded from memory.

**NOTE**
The EWA must be specified such that, when incremented, no carry will be propagated from bit position 27. Therefore, if all eight registers are to be loaded, bit positions 27-29 must initially be equal to zero.

**SUMMARY**
(EXWL) → R

(EXWL)+1 → R+1

... ...

(EXWL+N) → R7

**CONDITION CODE**
CC1: No change
CC2: No change
CC3: No change
CC4: No change

**RESULTS**
Memory Location: 00300
Hex Instruction: CE 00 02 00 (R=4, X=0, I=0)
Assembly Language Coding: LF 4,X'200'

**EXAMPLE**
Before Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
<th>GPR5</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>08000300</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
</tbody>
</table>

Memory Word 00200
000000001

Memory Word 00204
00000002

Memory Word 00208
00000003

Memory Word 0020C
00000004

After Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
<th>GPR5</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>08000304</td>
<td>00000001</td>
<td>00000002</td>
<td>00000003</td>
<td>00000004</td>
</tr>
</tbody>
</table>

Memory Word 00200
00000001

Memory Word 00204
00000002

Memory Word 00208
00000003

Memory Word 0020C
00000004

**Note**
The contents of memory word 00200 are transferred to GPR4, of memory word 00204 to GPR5, of memory word 00208 to GPR6, and of memory word 0020C to GPR7.
STORE BYTE

STB

s,*m,x

The least significant byte (bits 24-31) of the GPR specified by R is transferred to the memory byte location specified by the Effective Byte Address (EBA) in the Instruction Word. The other three bytes of the memory word containing the byte specified by the EBA remain unchanged.

CONDITION CODE

CC1: No change
CC2: No change
CC3: No change
CC4: No change

EXAMPLE

Memory Location: 03708
Hex Instruction: D4 88 3A 13 (R=1, X=0, I=0)
Assembly Language Coding: STB 1,X'3A13'

Before Execution

PSWR 10003708
GPR1 01020304

After Execution

PSWR 1000370C
GPR1 01020304

Memory Byte 03A13

Memory Byte 03A13

Note

The contents of bits 24-31 of GPR1 are transferred to memory byte 03A13.
**STH**

s,*m,x

---

**DEFINITION**

The least significant halfword (bit 16-31) of the GPR specified by R is transferred to the memory halfword location specified by the Effective Halfword Address (EHA) in the Instruction Word. The other halfword of the memory word containing the halfword specified by the EHA remains unchanged.

---

**SUMMARY**

\[(R_{16-31}) \rightarrow \text{EHL}\]

---

**CONDITION CODES**

- **CC1**: No change
- **CC2**: No change
- **CC3**: No change
- **CC4**: No change

---

**RESULTS**

---

**EXAMPLE**

- **Memory Location**: 082A400
- **Hex Instruction**: D6 00 83 13 (R=4, X=0, I=0)
- **Assembly Language Coding**: STH 4,X'8312'

---

Before Execution

- **PSWR**: 000082A4
- **GPR4**: 01020304
- **Memory Halfword**: 08312
- **PSWR**: 000082A8
- **GPR4**: 01020304
- **Memory Halfword**: 08312

After Execution

- **PSWR**: 000082A8
- **GPR4**: 01020304
- **Memory Halfword**: 08312
- **PSWR**: 000082A8
- **GPR4**: 01020304
- **Memory Halfword**: 0304

---

**Note**

The contents of the right halfword of GPR4 are transferred to memory halfword 08312.
DEFINITION
The word in the GPR specified by R is transferred to the memory word location specified by the Effective Word Address in the Instruction Word.

SUMMARY
(R) → EWL

CONDITION CODE
CC1: No change
CC2: No change
CC3: No change
CC4: No change

RESULTS

EXAMPLE
Memory Location: 03904
Hex Instruction: D7 00 3B 3C (R=6, X=0, I=0)
Assembly Language Coding: STW 6, X'3B3C'

Before Execution
PSWR 10003904
GPR6 0485A276
Memory Word O3B3C

After Execution
PSWR 10003908
GPR6 0485A276
Memory Word O3B3C

Note
The contents of GPR6 are transferred to memory word O3B3C.
**STD s,*m,X**

STORE DOUBLEWORD

D400

<table>
<thead>
<tr>
<th>1 1 0 1 0 1</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>0</th>
<th>DOUBLEWORD OPERAND ADDRESS</th>
<th>0 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DEFINITION**

The doubleword in the GPR specified by R and R+1 (R+1 is the GPR one greater than specified by R) is transferred to the memory doubleword location specified by the Effective Doubleword Address (EDA). The word in the GPR specified by R+1 is transferred to the least significant word of the doubleword memory location first.

**SUMMARY**

\[(R+1) \rightarrow \text{EWL}+1\]

**EXPRESSION**

\[(R) \rightarrow \text{EWL}\]

**CONDITION CODE**

CC1: No change

CC2: No change

CC3: No change

CC4: No change

**RESULTS**

**EXAMPLE**

Memory Location: 0596C
Hex Instruction: D7 00 5C 4A (R=6, X=0, I=0)
Assembly Language Coding: STD 6,X'5C48'

Before Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000596C</td>
<td>E24675C2</td>
<td>5923F8E8</td>
</tr>
</tbody>
</table>

Memory Word 05C48: 0A400729
Memory Word 05C4C: 8104A253

After Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>20005970</td>
<td>E24675C2</td>
<td>5923F8E8</td>
</tr>
</tbody>
</table>

Memory Word 05C48: E24675C2
Memory Word 05C4C: 5923F8E8

**Note**

The contents of GPR6 are transferred to memory word 05C48, and the contents from GPR7 are transferred to memory word 05C4C.
DEFINITION
The least significant byte (bits 24-31) of the GPR specified by R is masked (Logical AND Function) with the least significant byte of the Mask register (R4). The resulting byte is transferred to the memory byte location specified by the Effective Byte Address (EBA) in the Instruction Word. The other three bytes of the memory word containing the byte specified by the EBA remain unchanged.

SUMMARY
\((R_{24-31}) \& (R4_{24-31}) \rightarrow \text{EBL}\)

CONDITION CODE
CC1: No change
CC2: No change
CC3: No change
CC4: No change

RESULTS

EXAMPLE
Memory Location: 01DB0
Hex Instruction: DB 08 1E 91 (R=0, X=0, I=0)
Assembly Language Coding: STMB 0,X'1E91'

Before Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPRO</th>
<th>GPR4</th>
<th>Memory Byte 01E91</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001D80</td>
<td>AC089417</td>
<td>0000FFFFC</td>
<td>94</td>
</tr>
</tbody>
</table>

After Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPRO</th>
<th>GPR4</th>
<th>Memory Byte 01E91</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001D84</td>
<td>AC089417</td>
<td>0000FFFFC</td>
<td>14</td>
</tr>
</tbody>
</table>

Note
The right-hand byte of GPRO is ANDed with the right-hand byte of GPR4. The result is transferred to memory byte 01E91.
**DEFINITION**
The least significant halfword (bits 16-31) of the GPR specified by R is masked (Logical AND Function) with the least significant halfword of the Mask register (R4). The resulting halfword is transferred to the memory halfword location specified by the Effective Halfword Address (EHA) in the Instruction Word. The other halfword of the memory word containing the halfword specified by the EHA remains unchanged.

**SUMMARY**
$\left( R_{16-31} \right) \& \left( R_{416-31} \right) \rightarrow \text{EHL}$

**CONDITION CODE**
- CC1: No change
- CC2: No change
- CC3: No change
- CC4: No change

**RESULTS**

**EXAMPLE**

Memory Location:
01000

Hex Instruction:
DA 80 11 AF (R=5, X=0, I=0)

Assembly Language Coding:
STMH 5,X'11AE'

Before Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
<th>GPR5</th>
<th>Memory Halfword 011AD</th>
</tr>
</thead>
<tbody>
<tr>
<td>20001000</td>
<td>00003FFC</td>
<td>716A58AB</td>
<td>0000</td>
</tr>
</tbody>
</table>

After Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
<th>GPR5</th>
<th>Memory Halfword 011AD</th>
</tr>
</thead>
<tbody>
<tr>
<td>20001004</td>
<td>00003FFC</td>
<td>716A58AB</td>
<td>18AB</td>
</tr>
</tbody>
</table>

**Note**
The right-hand halfword of GPR5 is ANDed with the right-hand halfword of GPR4, and the result is transferred to memory halfword 011AD.
STORE MASKED WORD

The word in the GPR specified by R is masked (Logical AND Function) with the contents of the Mask register (R4). The resulting word is transferred to the memory word location specified by the Effective Word Address.

\[(R) \& (R4) \rightarrow \text{EWL}\]

**CONDITION CODE**

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>No change</td>
<td>No change</td>
<td>No change</td>
<td>No change</td>
</tr>
</tbody>
</table>

**RESULTS**

**EXAMPLE**

- **Memory Location:** 04000
- **Hex Instruction:** DB 00 43 7C (R=6, X=0, I=0)
- **Assembly Language Coding:** STM W 6,X'4376'

<table>
<thead>
<tr>
<th>Before Execution</th>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR</td>
<td>08004000</td>
</tr>
<tr>
<td>GPR4</td>
<td>00FF00FF</td>
</tr>
<tr>
<td>GPR6</td>
<td>718C3594</td>
</tr>
<tr>
<td>Memory Word</td>
<td>0437C</td>
</tr>
<tr>
<td></td>
<td>12345678</td>
</tr>
</tbody>
</table>

**Note**

The contents of GPR6 are ANDed with the contents of GPR4. The result is transferred to memory word 0437C.
STORE MASKED DOUBLEWORD

D800

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>0</th>
<th>DOUBLEWORD OPERAND ADDRESS</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td>25</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td></td>
<td>28</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td></td>
<td>31</td>
<td>32</td>
<td>33</td>
</tr>
</tbody>
</table>

DEFINITION
Each word of the doubleword in the GPR specified by R and R+1 is masked (Logical AND Function) with the contents of the Mask register (R4). R+1 is GPR one greater than specified by R. The resulting doubleword is transferred to the memory doubleword location specified by the Effective Doubleword Address (EDA) in the Instruction Word.

EXPRESSION
(R+1)&(R4) → EWL+1
(R)&(R4) → EWL

CONDITION CODE
CC1: No change
CC2: No change
CC3: No change
CC4: No change

RESULTS

EXAMPLE
Memory Location: OA498
Hex Instruction: DB 00 A6 52 (R=6, X=0, I=0)
Assembly Language Coding: STMD 6, X'A650'

Before Execution
PSWR GPR4 GPR6 GPR7
1000A498 0007FFFC AC88A819 98881407

After Execution
PSWR GPR4 GPR6 GPR7
1000A49C 0007FFFC AC88A819 98881407

Note
The contents of GPR6 are ANDed with the contents of GPR4, and the result is transferred to memory word OA650. The contents of GPR7 are ANDed with the contents of GPR4, and the result transferred to memory word OA654.
DEFINITION
This instruction is used to transfer the contents from one to eight GPR's to
the specified memory locations. The contents of the GPR specified by R are
transferred to the memory location specified by the Effective Word Address
(EWA). The next sequential GPR is then transferred to the next sequential
memory location. Successive transfers continue until GPR7 is loaded into
memory.

NOTE
The EWA must be specified such that, when incremented, no carry will be
propagated from bit position 27. Therefore, if all eight General Purpose
Registers are transferred, bit positions 27-29 must initially be equal to
zero.

SUMMARY
(R) → EWL
(R+1) → EWL+1
:
:
(R7) → EWL+N

CONDITION CODE
CC1: No change
CC2: No change
CC3: No change
CC4: No change

RESULTS

EXAMPLE
Memory Location: 02000
Hex Instruction: DE 00 21 00 (R=4, X=0, I=0)
Assembly Language Coding: STF 4,X'2100'

Before Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
<th>GPR5</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>40002000</td>
<td>11111111</td>
<td>22222222</td>
<td>33333333</td>
<td>44444444</td>
</tr>
</tbody>
</table>

Memory Word 02100
00210000

Memory Word 02108
00210800

After Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
<th>GPR5</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>40002004</td>
<td>11111111</td>
<td>22222222</td>
<td>33333333</td>
<td>44444444</td>
</tr>
</tbody>
</table>

Memory Word 02100
11111111

Memory Word 02108
33333333

Note
The contents of GPR4 are transferred to memory word 02100, of GPR5 to 02104,
of GPR6 to 02108, and of GPR7 to 0210C.
ZERO MEMORY BYTE

F808

**DEFINITION**
The byte in memory specified by the Effective Byte Address (EBA) is cleared to zero. The other three bytes of the memory word containing the byte specified by the EBA remain unchanged.

**SUMMARY EXPRESSION**
0 → EBL

**CONDITION CODE RESULTS**
CC1: No change
CC2: No change
CC3: No change
CC4: No change

**EXAMPLE**
Memory Location: 00308
Hex Instruction: F8 08 04 9F
Assembly Language Coding: ZMB X'49F'

Before Execution
PSWR 100030B 6C
Memory Byte 0049F

After Execution
PSWR 100030C 00
Memory Byte 0049F

Note The contents of memory byte 0049F are cleared to zero.
**DEFINITION**

The halfword in memory specified by the Effective Halfword Address (EHA) is cleared to zero. The remaining halfword containing the 16-bit location in memory specified by EHA remains unchanged.

**SUMMARY**

0 → EHL

**CONDITION CODE**

<table>
<thead>
<tr>
<th>CC1:</th>
<th>CC2:</th>
<th>CC3:</th>
<th>CC4:</th>
</tr>
</thead>
<tbody>
<tr>
<td>No change</td>
<td>No change</td>
<td>No change</td>
<td>No change</td>
</tr>
</tbody>
</table>

**RESULTS**

**EXAMPLE**

Memory Location: 2895C
Hex Instruction: F8 00 2A 42 7 (X=0, I=0)
Assembly Language Coding: ZMH X'2A426'

**Before Execution**

PSWR 0802895C
Memory Halfword 2A426 9AE3

**After Execution**

PSWR 08028960
Memory Halfword 2A426 0000

**Note**

The contents of memory halfword 2A426 are cleared to zero.
DEFINITION
The word in memory specified by the Effective Word Address (EWA) is cleared to zero.

SUMMARY
0 − EWL

CONDITION CODE
CC1: No change
CC2: No change
CC3: No change
CC4: No change

RESULTS

EXAMPLE
Memory Location: 05A14
Hex Instruction: F8 00 5F 90 (X=0, I=0)
Assembly Language Coding: ZMW X'5F90'

Before Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>Memory Word</th>
<th>05F90</th>
</tr>
</thead>
<tbody>
<tr>
<td>00005A14</td>
<td>12345678</td>
<td></td>
</tr>
</tbody>
</table>

After Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>Memory Word</th>
<th>05F90</th>
</tr>
</thead>
<tbody>
<tr>
<td>00005A18</td>
<td>00000000</td>
<td></td>
</tr>
</tbody>
</table>

Note
The contents of memory word 05F90 are cleared to zero.
ZERO MEMORY DOUBLEWORD

**ZMD**

*m, X

---

**DEFINITION**
The doubleword in memory specified by the Effective Doubleword Address (EDA) is cleared to zero.

**SUMMARY**

<table>
<thead>
<tr>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 → EWL</td>
</tr>
<tr>
<td>0 → EWL+1</td>
</tr>
</tbody>
</table>

**CONDITION CODE**

| CC1: | No change |
| CC2: | No change |
| CC3: | No change |
| CC4: | No change |

**RESULTS**

**EXAMPLE**

| Memory Location: | 15B3C |
| Hex Instruction: | F8 01 5D 6A (X=0, I=0) |
| Assembly Language Coding: | ZMD X'15D68' |

**Before Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>Memory Word 15D68</th>
<th>Memory Word 15D6C</th>
</tr>
</thead>
<tbody>
<tr>
<td>10015B3C</td>
<td>617E853C</td>
<td>A2976283</td>
</tr>
</tbody>
</table>

**After Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>Memory Word 15D68</th>
<th>Memory Word 15D6C</th>
</tr>
</thead>
<tbody>
<tr>
<td>10015B40</td>
<td>00000000</td>
<td>00000000</td>
</tr>
</tbody>
</table>

**Note**
The contents of memory words 15D68 and 15D6C are cleared to zero.
ZERO REGISTER

OC00

The word in the GPR specified by R (bits 6-8) is logically Exclusive ORed with the word in the GPR specified by R (bits 9-11) resulting in zero. This result is then transferred to the GPR specified by R. The contents of the two R fields must specify the same GPR.

CONDITION CODE

CC1: Always zero
CC2: Always zero
CC3: Always zero
CC4: Always one

RESULTS

MEMORY LOCATION:

Hex Instruction:

Assembly Language Coding:

Before Execution

After Execution

Note The contents of GPR1 are cleared to zero, and CC4 is set.
The Register Transfer instruction group provides the capability to perform a transfer or exchange of information between registers. Provisions have also been made in some instructions to allow two's complement, one's complement, and Mask operations to be performed during execution.

The following basic instruction format is used by the Register Transfer instruction group.

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R_D</th>
<th>R_S</th>
<th>AUG CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 0-5 define the Operation Code.

Bits 6-8 designate the register to contain the result of the operation.

Bits 9-11 designate the register which contains the source operand.

Bits 12-15 define the Augmenting Operation Code.

A Condition Code is set during execution of most Register Transfer instructions to indicate whether the contents of the Destination register (R_D) are greater than, less than, or equal to zero.
TRANSFER SCRATCHPAD TO REGISTER

**DEFINITION**
The word in the Scratchpad specified by RS, bits 8-15, is transferred to the GPR specified by RD. The contents of RS is not modified and only bits 8-15 are used by the instruction.

**SUMMARY**
Scratchpad addressed by RS → RD

**CONDITION CODE RESULTS**
CC1: No change
CC2: No change
CC3: No change
CC4: No change

**NOTES**
1. TSCR is a halfword privileged instruction.
2. The valid address range for RS to address the 256 Scratchpad locations is XX00XXXXH to XXFFXXXXH.

**Assembly Language Coding:** TSCR RS, RD
TRSC  
$s, d$

TRANSFER REGISTER TO SCRATCHPAD

2COE

<table>
<thead>
<tr>
<th>0 0 1 0 1 1</th>
<th>$RD$</th>
<th>$RS$</th>
<th>1 1 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DEFINITION**
The word located in the General Purpose Register (GPR) specified by $RS$ is transferred to the Scratchpad location specified by $RD$ bits 8-15. The contents of $RD$ is not modified by the instruction and only bits 8-15 are used by the instruction.

**SUMMARY**

$(RS) \rightarrow$ Scratchpad addressed by $RD$ 8-15

**CONDITION CODE RESULTS**

CC1: No change  
CC2: No change  
CC3: No change  
CC4: No change

**Assembly Language Coding:** TRS $RS, RD$

**NOTES**

1. TRSC is a halfword privileged instruction.
2. The valid address range for $RD$ to address the 256 Scratchpad locations is $XX000000_H$ to $XXFFXXXX_H$. 
TRANSFER REGISTER TO REGISTER

2C00

<table>
<thead>
<tr>
<th>0 0 1 0 1 1</th>
<th>R_S</th>
<th>R_D</th>
<th>0 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DEFINITION
The word in the GPR specified by R_S is transferred to the GPR specified by R_D.

SUMMARY
(R_S) → R_D

CONDITION CODE
CC1: Always zero
CC2: ISI (R_D) is greater than zero
CC3: ISI (R_D) is less than zero
CC4: ISI (R_D) is equal to zero

RESULTS

EXAMPLE
Memory Location: 00206
Hex Instruction: 2C A0 (R_D=1, R_S=2)
Assembly Language Coding: TRR 2, 1

Before Execution
PSWR: 00000206
GPR1: 00000000
GPR2: 000803AB

After Execution
PSWR: 20000208
GPR1: 000803AB
GPR2: 000803AB

Note
The contents of GPR2 are transferred to GPR1 and CC2 is set.
**TRRM**  
**TRANSFER REGISTER TO REGISTER MASKED**  
2C08

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>R_\text{D}</th>
<th>R_\text{S}</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**DEFINITION**  
The word in the GPR specified by $R_\text{S}$ is masked (Logical AND Function) with the contents of the Mask register (R4). The resulting word is transferred to the GPR specified by $R_\text{D}$.

**SUMMARY**  
$(R_\text{S}) \& (R4) \rightarrow R_\text{D}$

**CONDITION CODE**  
- **CC1**: Always zero  
- **CC2**: ISI ($R_\text{D}$) is greater than zero  
- **CC3**: ISI ($R_\text{D}$) is less than zero  
- **CC4**: ISI ($R_\text{D}$) is equal to zero

**RESULTS**

**EXAMPLE**  
**Memory Location:** 00206  
**Hex Instruction:** 2C A8 ($R_\text{D}$=1, $R_\text{S}$=2)  
**Assembly Language Coding:** TRRM 2,1

**Before Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR1</th>
<th>GPR2</th>
<th>GPR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000206</td>
<td>00000000</td>
<td>000803AB</td>
<td>0007FFFF</td>
</tr>
</tbody>
</table>

**After Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR1</th>
<th>GPR2</th>
<th>GPR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000208</td>
<td>000003A9</td>
<td>000803AB</td>
<td>0007FFFF</td>
</tr>
</tbody>
</table>

**Note**  
The contents of GPR2 are ANDed with the contents of GPR4, and the result is transferred to GPR1. CC2 is set.
TRANSFER REGISTER TO PROTECT REGISTER

FB00

<table>
<thead>
<tr>
<th>1 1 1 1 1 0 1 1 0 PROT.REG.</th>
<th>R</th>
<th>UNASSIGNED</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DEFINITION
The word in the GPR specified by R is transferred to the Protect register specified by the Protect register field (bits 9-12) in the Instruction Word. The Protect register address is the same as the four high order memory address bits used to specify all memory locations within a given module.

SUMMARY
(R) → PR

EXPRESSION

CONDITION CODE
CC1: No change
CC2: No change
CC3: No change
CC4: No change

RESULTS

EXAMPLE
Memory Location: 0050C
Hex Instruction: FBOF (R=7, Protect Register=1)
Assembly Language Coding: TRP 7,1

Before Execution
PSWR 800005CO GPR7 0000FFFE Protect Register 1
After Execution
PSWR 80000510 GPR7 0000FFFE FFFE

Note
The contents of bits 16-31 of GPR7 are transferred to Protect Register 1. The protection status of Memory Module 1 is established such that a program operating in the unprivileged state can store information only in locations 8000 through 87FF without generating a Privilege Violation trap.
TRANSFER PROTECT REGISTER TO REGISTER

FB80

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>PROT. REG.</th>
<th>R</th>
<th>UNASSIGNED</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

DEFINITION
The word in the Protect register specified by the Protect register field (bits 9-12) is transferred to the GPR specified by R. The Protect register address is the same as the four high order memory address bits used to specify all memory locations within a given module.

SUMMARY
(PR) → R

CONDITION CODE
CC1: No change
CC2: No change
CC3: No change
CC4: No change

RESULTS

EXAMPLE
Memory Location: 0050C
Hex Instruction: FB8F (R=7, Protect Register=1)
Assembly Language Coding: TPR 1,7

Before Execution
PSWR: 0000050C
GPR7: 00000000
Protect Register 1: FFFE

After Execution
PSWR: 00000510
GPR7: 0000FFFF
Protect Register 1: FFFE

Note
The contents of Protect Register 1 are transferred to bits 16-31 of GPR7. This value defines the protection status of Memory Module 1.
**TRANSFER REGISTER NEGATIVE**

**2C04**

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>R&lt;sub&gt;D&lt;/sub&gt;</th>
<th>R&lt;sub&gt;S&lt;/sub&gt;</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DEFINITION**
The word in the GPR specified by R<sub>S</sub> is two's complemented and transferred to the GPR specified by R<sub>D</sub>.

**SUMMARY**

**EXPRESSION**

\[-(R_S) \rightarrow R_D\]

**CONDITION CODE RESULTS**

CC1: ISI Arithmetic exception
CC2: ISI (R<sub>D</sub>) is greater than zero
CC3: ISI (R<sub>D</sub>) is less than zero
CC4: ISI (R<sub>D</sub>) is equal to zero

**EXAMPLE**

Memory Location: 00AAE
Hex Instruction: 2F E4 (R<sub>D</sub>=7, R<sub>S</sub>=6)
Assembly Language Coding: TRN 6,7

<table>
<thead>
<tr>
<th>Before Execution</th>
<th>PSWR</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00000AAE</td>
<td>00000FFF</td>
<td>12345678</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>After Execution</th>
<th>PSWR</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10000AB0</td>
<td>00000FFF</td>
<td>FFFFF001</td>
</tr>
</tbody>
</table>

**Note**
The contents of GPR6 are negated and transferred to GPR7. CC3 is set.
TRANSFER REGISTER NEGATIVE MASKED

2COC

DEFINITION
The word in the GPR specified by Rs is two's complemented and masked
(Logical AND Function) with the contents of the Mask register (R4). The
resulting word is transferred to the GPR specified by Rd.

SUMMARY
-(Rs)&(R4) → Rd

CONDITION CODE
CC1: ISI Arithmetic exception
CC2: ISI (Rd) is greater than zero
CC3: ISI (Rd) is less than zero
CC4: ISI (Rd) is equal to zero

RESULTS

EXAMPLE
Memory Location: O00AAE
Hex Instruction: 2F EC (Rd=7, Rs=6)
Assembly Language Coding: TRNM 6,7

Before Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000AAE</td>
<td>7FFFFFF</td>
<td>00000FFF</td>
<td>12345678</td>
</tr>
</tbody>
</table>

After Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>20000AB0</td>
<td>7FFFFFF</td>
<td>00000FFF</td>
<td>7FFFF001</td>
</tr>
</tbody>
</table>

Note
The contents of GPR6 are negated; the result is ANDed with the content
of GPR4 and transferred to GPR7. CC2 is set.
TRANSFER REGISTER COMPLEMENT

2C03

The word in the GPR specified by RS is one's complemented and transferred to the GPR specified by RD.

\[(RS) \rightarrow RD\]

CONDITION CODE

CC1: Always zero
CC2: ISI (RD) is greater than zero
CC3: ISI (RD) is less than zero
CC4: ISI (RD) is equal to zero

EXAMPLE

Memory Location: 01001
Hex Instruction: 2F E3 (RO=7, RS=6)
Assembly Language Coding: TRC 6,7

Before Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0800100A</td>
<td>55555555</td>
<td>00000000</td>
</tr>
</tbody>
</table>

After Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000100C</td>
<td>55555555</td>
<td>AAAAAAAA</td>
</tr>
</tbody>
</table>

Note: The contents of GPR6 are complemented and transferred to GPR7. CC3 is set.
The word in the GPR specified by Rs is one's complemented and masked (Logical AND Function) with the contents of the Mask register (R4). The result is transferred to the GPR specified by Rd.

\[(R_s) \& (R4) \rightarrow R_d\]

**Condition Code Results**
- **CC1**: Always zero
- **CC2**: ISI \((R_D)\) is greater than zero
- **CC3**: ISI \((R_D)\) is less than zero
- **CC4**: ISI \((R_D)\) is equal to zero

**Example**
- **Memory Location**: 0100A
- **Hex Instruction**: 2F EB \((R_0=7, R_s=6)\)
- **Assembly Language Coding**: TRCM 6,7

**Before Execution**
- **PSWR**: 0800100A
- **GPR4**: OOF0FF00
- **GPR6**: 55555555
- **GPR7**: 00000000

**After Execution**
- **PSWR**: 2000100C
- **GPR4**: OOF0FF00
- **GPR6**: 55555555
- **GPR7**: 00AAAA00

**Note**
The content of GPR6 are complemented and ANDed with the contents of GPR4. The result is transferred to GPR4. The result is transferred to GPR7. CC2 is set.
EXCHANGE REGISTERS

2C05

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>R_D</th>
<th>R_S</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

DEFINITION
The word in the GPR specified by R_S is exchanged with the word in the GPR specified by R_D.

SUMMARY
(R_S) → R_D

EXPRESSION
(R_D) → R_S

CONDITION CODE
CC1: Always zero
CC2: ISI Original (R_D) is greater than zero
CC3: ISI Original (R_D) is less than zero
CC4: ISI Original (R_D) is equal to zero

RESULTS

EXAMPLE
Memory Location: 02002
Hex Instruction: 2C A5 (R_D=1, R_S=2)
Assembly Language Coding: XCR 2,1

Before Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR1</th>
<th>GPR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>40002002</td>
<td>00000000</td>
<td>AC8823C1</td>
</tr>
</tbody>
</table>

After Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR1</th>
<th>GPR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>08002004</td>
<td>AC8823C1</td>
<td>00000000</td>
</tr>
</tbody>
</table>

Note
The contents of GPR1 and GPR2 are exchanged. CC4 is set.
**Definition**
The contents of the GPR specified by \( R_S \) and \( R_D \) are each masked (Logical AND Function) with the contents of the Mask register (R4). The results of both masked operations are exchanged.

**Summary Expression**
\((R_S) \& (R4) \rightarrow R_D\)

\((R_D) \& (R4) \rightarrow R_S\)

**Condition Code Results**
- **CC1**: Always zero
- **CC2**: ISI original \((R_D)\) and \((R4)\) is greater than zero
- **CC3**: ISI original \((R_D)\) and \((R4)\) is less than zero
- **CC4**: ISI original \((R_D)\) and \((R4)\) is equal to zero

**Example**

Memory Location: 02002
Hex Instruction: 2C AD \((R_D=1, R_S=2)\)
Assembly Language Coding: XCRM 2,1

<table>
<thead>
<tr>
<th>Before Execution</th>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR</td>
<td>PSWR</td>
</tr>
<tr>
<td>40002002</td>
<td>08002004</td>
</tr>
<tr>
<td>GPR1</td>
<td>GPR1</td>
</tr>
<tr>
<td>6B000000</td>
<td>000823C1</td>
</tr>
<tr>
<td>GPR2</td>
<td>GPR2</td>
</tr>
<tr>
<td>AC8823C1</td>
<td>00000000</td>
</tr>
<tr>
<td>GPR4</td>
<td>GPR4</td>
</tr>
<tr>
<td>FF00FF00</td>
<td>00000000</td>
</tr>
</tbody>
</table>

**Note**
The contents of GPR1 and GPR2 are each ANDed with the contents of GPR4. The results of the masking operation are exchanged and transferred to GPR2 and GPR1, respectively. CC4 is set.
TRANSFER REGISTER TO PSWR

2800

0 0 1 0 0 0 0 0 0 0

Before Execution

PSWR  6000069E
GPRO  20000B4C

After Execution

PSWR  6000069E
GPRO  20000B4C

DEFINITION

Bit positions 1-4 and 13-30 of the General Purpose Register (GPR) specified by R are transferred to the corresponding bit positions of the Program Status Word Register (PSWR).

SUMMARY

R1-4, 13-30 → PSWR1-4, 13-30

CONDITION CODE

CC1: ISI (R1) is equal to one
CC2: ISI (R2) is equal to one
CC3: ISI (R3) is equal to one
CC4: ISI (R4) is equal to one

RESULTS

Memory Location: 0069E
Hex Instruction: 28 00 (R=0)
Assembly Language Coding: TRSW 0

EXAMPLE

Before Execution

PSWR  6000069E
GPRO  A0000B4C

After Execution

PSWR  6000069E
GPRO  A0000B4C

Note

1. The contents of GPRO, bits 1-4 and 13-30 are transferred to the PSWR. PSWR bits 0, 5-12, and 31 are unchanged.

2. This instruction can be used in PSD mode to modify CC and PC portions of PSW1.
The 32/70 Series Computer provides the capability of accessing memory in any of the following four modes:

1. 512 KB Mode
2. 512 KB Extended Mode
3. 512 KB Mapped Mode
4. Mapped, Extended Mode

The format for the Memory Management instructions vary to the extent that no single format can represent them. The instructions are presented on the following pages.
DEFINITION
The CPU enters the Extended Addressing mode.

CONDITION CODE
CC1: No change
CC2: No change
CC3: No change
CC4: No change

RESULTS
Assembly Language Coding: SEA

NOTES
1. This is a nonprivileged instruction.
2. Sets bit 5 in PSD, word 1.
CLEAR EXTENDED ADDRESSING

OOOF

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

**DEFINITION**
The CPU enters the Normal (Nonextended) Addressing mode.

**CONDITION CODE RESULTS**
CC1: No change
CC2: No change
CC3: No change
CC4: No change

Assembly Language Coding: CEA

**NOTES**
1. This is a nonprivileged instruction.
2. Clears bit 5 in PSD, word 1.
**LOAD MAP**

2C07

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

**DEFINITION**

Loads the MAP Image Descriptor List (MIDL) from main memory into the CPU MAP Registers. \( R_D \) contains the Real Address of a PSD to be used in the MAP loading process.

**SUMMARY**

(MIDL) — MAP Registers

**CONDITION CODE**

CC1: No change
CC2: No change
CC3: No change
CC4: No change

**RESULTS**

Assembly Language Coding: LMAP \( R_D \)

**NOTES**

1. This instruction primarily used for diagnostic purposes.
2. The CPU must be unmapped.
3. Only MAP Load functions are performed, with no context switching.
4. Attempts to execute this instruction in PSW mode will result in an undefined instruction trap.
5. This is a privileged instruction.
6. This is a fullword instruction.
TMAPR
s,d
TRANSFER MAP TO REGISTER

2COA

DEFINITION
This instruction causes the even and odd map entries, specified by Rs
bits 27-31 to be transferred to the GPR specified by Rd. The least
significant map address bit (Rs bit 31) is ignored by the instruction.

SUMMARY
MAP addressed by Rs 27-31 → Rd

EXPRESSION

CONDITION CODE RESULTS
CC1: No change
CC2: No change
CC3: No change
CC4: No change

Assembly Language Coding: TMAPR Rs,Rd

NOTES
1. If this instruction is executed in the PSW mode, an undefined
   instruction trap will occur.
2. This is a halfword privileged instruction.
3. The format for Rs is as follows:

4. The CPU must be Unmapped.
**WRITABLE CONTROL STORAGE (WCS) INSTRUCTIONS**

**GENERAL DESCRIPTION**

Writable Control Storage (WCS) is an option available for use with the CPU or Class F I/O controller. The WCS consists of one or two Random Access Memory (RAM) logic boards, each containing 2K- x 64-bits of RAM memory. The WCS is used to supplement the firmware in the CPU or the Class F I/O controller.

**INSTRUCTION FORMAT**

There are two instruction formats used for WCS instructions, one for the CPU associated WCS, and one for the Class F I/O Controller associated WCS. The formats are as follows:

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R_D</th>
<th>R_S</th>
<th>AUG CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3</td>
<td>4 5</td>
<td>6 7</td>
<td>8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
</tr>
</tbody>
</table>

- **Bits 0-5** Define the Operation Code.
- **Bits 6-8** Varies in usage as follows:
  - Instruction: WWCS, Usage: Specifies the register containing the WCS address.
  - Instruction: RWCS, Usage: Specifies the register containing the Logical Address in main memory that is to receive the WCS contents.
- **Bits 9-11** Varies in usage as follows:
  - Instruction: WWCS, Usage: Specifies the register containing the Logical Address in main memory containing the information to be loaded into WCS.
  - Instruction: RWCS, Usage: Specifies the register containing the WCS address.
- **Bits 12-15** Define the Augmenting Operating Code.
- **Bits 16-31** Not used. This is a halfword instruction.
Bits 6-8 Specify the GPR, when nonzero, whose contents will be added to the constant to form the logical channel and subaddress.

Bits 9-12 Specifies the Channel WCS Operation Code.


Bits 16-31 Specifies a constant that will be added to the contents of R to form the logical Channel and subaddress. If R is zero, only the constant will be used to specify the logical Channel and subaddress.

The Condition Codes remain unchanged when using the CPU associated WCS. When using the class F I/O controller associated WCS, the Condition Codes are changed in accordance with the WCS instructions. Refer to the individual Class F I/O controller WCS instructions for details.

Programming the CPU associated WCS is accomplished by the use of the Write WCS (WWCS) instruction. The contents of the WCS are in the form of microinstructions, which are used to augment the firmware in the CPU. It is beyond the scope of this publication to provide the microinstruction techniques used in the implementation of WCS.

The WCS is organized in 64 bits by 2K modules, allowing up to two modules to be used (4K x 64 bits). Reading or writing WCS is accomplished by alternately placing the first 32-bit word in the first 32 bits and then the second 32-bit word in the second 32 bits. A graphic representation of the Read/Write sequence is shown as follows:

Accessing the CPU associated WCS is accomplished through the use of the Jump to WCS (JWCS) instruction. More complete information of the programming of the WCS is contained in the Writable Control Storage Technical Manual.

Programming of the Class F I/O controller associated WCS is presented in the individual I/O Processor publications.
WRITE WRITABLE CONTROL STORAGE

DEFINITION
This privileged instruction causes the WCS to be written with a single 64-bit word at the location specified by the contents of RD, with two words in main memory specified by the logical addresses contained in RS.

The contents of RS must contain a logical word address that specifies the first word of a two-word pair. F- and C-bits, if specified, are ignored and the address will be interpreted as a word address.

The contents of RD must contain a right-justified, zero-filled address of the WCS location that is to be written.

If the WCS option is not present or if the WCS address is greater than 4095: CC1 will be set, an Undefined Instruction Trap will occur, and no writing into WCS will take place.

CONDITION
CC1: WCS option not present or address out of range
CC2: Zero
CC3: Zero
CC4: Zero

RESULTS
Assembly Language Coding: WWCS RS, RD
**RWCS s,d**

**READ WRITABLE CONTROL STORAGE**

<table>
<thead>
<tr>
<th>0 0 0 0 0 0</th>
<th>( R_D )</th>
<th>( R_S )</th>
<th>1 0 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5</td>
<td>6 7 8</td>
<td>9 10 11</td>
<td>12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
</tr>
</tbody>
</table>

**DEFINITION**

This privileged instruction causes the contents of a single 64-bit location of WCS specified by the contents of \( R_S \) to be written into main memory at the location specified by the logical address contained in \( R_D \).

The contents of \( R_D \) must contain a logical word address that specifies the first word pair. F- and C-bits, if specified, are ignored and the address will be interpreted as a word address.

The contents of \( R_S \) must contain a right-justified, zero-filled address of the WCS location that is to be read.

If the WCS option is not present or if the WCS address is greater than 4095: CC1 will be set, an Undefined Instruction Trap will occur, and no information will be stored into main memory.

**CONDITION CODE RESULTS**

CC1: WCS option not present or address out of range
CC2: Zero
CC3: Zero
CC4: Zero

Assembly Language Coding: RWCS \( R_S, R_D \)
DEFINITION

This instruction causes an Unconditional Branch to the location specified by the resolved Effective Address. The rules for the Effective Address are as follows:

- Nonindirect - the least significant 6 bits of the Effective Address (index and address) will be used as the WCS entry point address.
- Indirect - the least significant 6 bits of the final resolved Effective Address after the resolution of all indirect addresses will be used as the WCS entry point address.

Only the least significant 6 bits of the Effective Address are used and all other bits will be ignored.

When execution in WCS is complete, control will be returned to the next sequential instruction after this instruction.

NOTES

1. Since no registers can be specified by this instruction, the authors of the WCS instructions and the software instructions must mutually agree upon the parameter registers. In general cases, registers 0 and 1 can be used. If the WCS facility is not supported, an Undefined Instruction Trap will occur.

2. If indirect accesses are used, the F-bit must be present in each indirect word.

CONDITION CODE

RESULTS

Assembly Language Coding: JWCS X'WCS Branch Addr'
Branch instructions provide the capability of testing for certain conditions and branching to another address if the conditions specified by the instruction are satisfied. Branch instructions permit referencing subroutines, repeating segments of programs, or returning to the next instruction within a sequence.

The Branch instruction group uses the following instruction format:

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R/D</th>
<th>X</th>
<th>I</th>
<th>F</th>
<th>BRANCH ADDRESS</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 0-5 define the Operation Code.
Bits 6-8 vary in usage as follows:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Contents/Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BU, BFT</td>
<td>000</td>
</tr>
<tr>
<td>BCT, BCF</td>
<td>D field</td>
</tr>
<tr>
<td>BIB, BIH,</td>
<td>Register Number</td>
</tr>
<tr>
<td>BIW, BID</td>
<td></td>
</tr>
<tr>
<td>BL</td>
<td>001</td>
</tr>
<tr>
<td>BRI</td>
<td>010</td>
</tr>
</tbody>
</table>

Bits 9-10 designate one of three index registers.
Bit 11 indicates whether an indirect addressing operation is to be performed.
Bit 12 is zero.
Bits 13-30 specifies the branch address when X and I fields are zero.
Bit 31 is zero.

Condition Code results during branching operations are unique because they reflect the state of the indirect bit of the instruction and the state of bits 1, 2, 3, and 4 of the indirect address obtained from the specified memory location.
The usual procedure for calling a subroutine is to execute a Branch and Link (BL) whose effective address is the starting location of the routine. Since PC+1 is saved in GPRO, a subsequent return can be made to the location following the BL by executing a TRSW 0. The PSW including the PC+1 word is saved in GPRO. Hence, the subroutine can be reentrant (pure); i.e., memory is not modified by calling it. If we wish to use GPRO in the subroutine, we can store the return address in a convenient location in memory, location B, with an STW 0, B, and then return with a BU *B.

Consider a move subroutine to move 50 words beginning at TAB. The routine begins at MOVE, whose address is stored in C.MOVE. The main program would contain:

```
BL *C.MOVE
...
; Return here
```

GPR1 is used as an Index register for counting through the table and GPR5 is used to output the data. The starting address of the table is in TAB 1. The subroutine is as follows:

```
COUNT EQU 50

MOVE LI 1, -COUNT
LOOP LW 5, TAB+COUNT,1
STW 5, TAB+COUNT,1
BIW 1, LOOP
TRSW 0
```

Argument Passing

Given an arithmetic subroutine that operates on arguments in GPR5 and GPR6, leaving the result in GPR6, the subroutine call is as follows:

```
BL SQRT Call with arguments in GPR5 and GPR6
...
```

The subroutine is as follows:

```
SQRT Arithmetic operations
...
TRSW 0 Return to Call + 1 word
```

In the preceding example, the calling program must load the General Purpose Registers before calling the subroutine. It is often convenient for the program to supply the arguments (or the addresses of the locations that contain them) with the call, and for the subroutine to handle the data transfers. With this method, the program gives the arguments in the two memory locations following the BL.

```
BL SQRT
...
Argument 1
...
Argument 2
...
Return here with result in GPR6
```
The return is made to the location following the second argument with the result in GPR6.

\[
\begin{align*}
\text{SQRT} & \quad \text{TRR} & \quad 0,1 \\
\text{LD} & \quad 6,0,1 & \quad \text{Pick up Arguments 1 and 2} \\
\ldots & \\
\text{ADI} & \quad 0,8 & \quad \text{Increment return address by 2 words} \\
\text{TRSW} & \quad 0 & \quad \text{Return to Call + 3 words}
\end{align*}
\]

An alternate method which allows up to six arguments to be passed per instruction utilizes the Load File instruction as follows:

\[
\begin{align*}
\text{SQRT} & \quad \text{TRR} & \quad 0,1 \\
\text{LF} & \quad 2,0,1 & \quad \text{Pick up Arguments 1-6} \\
\ldots & \\
\text{ADI} & \quad 0,24 & \quad \text{Increment return address by 6 words} \\
\text{TRSW} & \quad 0 & \quad \text{Return to Call + 7 words}
\end{align*}
\]

The next method passes an address list instead of arguments following the BL; otherwise, it is identical to the method described above.

\[
\begin{align*}
\text{BL} & \quad \text{SQRT} & \quad \text{Address of Argument 1} \\
\ldots & \quad \text{Address of Argument 2} \\
\ldots & \\
\ldots & \\
\text{SQRT} & \quad \text{TRR} & \quad 0,1 \\
\text{LW} & \quad 6,*0,1 & \quad \text{Pick up Argument 1} \\
\text{ADI} & \quad 1,4 & \quad \text{Pick up Argument 2} \\
\text{LW} & \quad 7,*0,1 & \quad \text{Pick up Argument 2} \\
\ldots & \\
\text{ADI} & \quad 0,8 & \quad \text{Increment return address by 2 words} \\
\text{TRSW} & \quad 0 & \quad \text{Return to Call + 3 words}
\end{align*}
\]

The next method is the same as the previous example except that argument 1 is a table, and the result replaces the second argument in memory:

\[
\begin{align*}
\text{BL} & \quad \text{SQRT} & \quad \text{Address of Argument 1} \\
\ldots & \quad \text{Address of Argument 2 and result} \\
\ldots & \\
\ldots & \\
\text{SQRT} & \quad \text{TRR} & \quad 0,3 & \quad \text{Pick up base address of table, Argument 1} \\
\text{TRR} & \quad 0,1 \\
\text{ABR} & \quad 29,1 & \quad \text{Increment return address by 4 words} \\
\text{LW} & \quad 6,*0,1 & \quad \text{Pick up Argument 2} \\
\ldots & \\
\ldots & \\
\end{align*}
\]
The final method is similar to the previous versions except that GPR1-GPR7 are not disturbed:

<table>
<thead>
<tr>
<th>SQRT</th>
<th>STF</th>
<th>0, SAVE</th>
<th>Save General Purpose Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRR</td>
<td>0,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LW</td>
<td>6, *0,1</td>
<td>Pick up Arguments</td>
<td></td>
</tr>
<tr>
<td>ADI</td>
<td>1,4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LW</td>
<td>7, *0,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>6, *0,1</td>
<td>Store result</td>
<td></td>
</tr>
<tr>
<td>LF</td>
<td>0, SAVE</td>
<td>Restore General Purpose Registers</td>
<td></td>
</tr>
<tr>
<td>ADI</td>
<td>0,8</td>
<td>Increment return address by 2 words</td>
<td></td>
</tr>
<tr>
<td>TRSW</td>
<td>0</td>
<td>Return to Call + 3 words</td>
<td></td>
</tr>
<tr>
<td>SAVE</td>
<td>REZ</td>
<td>1F</td>
<td>Eight zero-filled words on a file boundary</td>
</tr>
</tbody>
</table>
The Effective Address (bits 13-30) in the instruction is transferred to the corresponding bit positions in the Program Status Word Register (PSWR). This causes program control to be transferred to any word or halfword location in memory. Bit positions 1-12 of the PSWR remain unchanged if the indirect bit is equal to zero. If the indirect bit of the Instruction Word is equal to one, bit positions 1-4 of the last memory word in the indirect chain are transferred to the corresponding bit positions of the PSWR. Bit 0 (privileged state bit) of the PSWR remains unchanged. The Extended mode bit remains unchanged. Bits 6 and 5 are changed only by a BRI indirect.

**Definition**

The Effective Address (bits 13-30) in the instruction is transferred to the corresponding bit positions in the Program Status Word Register (PSWR). This causes program control to be transferred to any word or halfword location in memory. Bit positions 1-12 of the PSWR remain unchanged if the indirect bit is equal to zero. If the indirect bit of the Instruction Word is equal to one, bit positions 1-4 of the last memory word in the indirect chain are transferred to the corresponding bit positions of the PSWR. Bit 0 (privileged state bit) of the PSWR remains unchanged. The Extended mode bit remains unchanged. Bits 6 and 5 are changed only by a BRI indirect.

**Summary**

\[ EA_{13-30} \rightarrow PSWR_{13-30}, \text{ IF } I=0 \]

\[ (EWL_{1-4} \text{ and } 13-30) \rightarrow PSWR_{1-4} \text{ and } 13-31, \text{ IF } I=1 \]

**Condition Code**

If the indirect bit is equal to zero, the Condition Code remains unchanged.

**Results**

CC1: ISI (I) is equal to one and (EWL1) is equal to one
CC2: ISI (I) is equal to one and (EWL2) is equal to one
CC3: ISI (I) is equal to one and (EWL3) is equal to one
CC4: ISI (I) is equal to one and (EWL4) is equal to one

**Example 1**

Memory Location: 01000
Hex Instruction: EC 00 14 14 (X=0, I=0)
Assembly Language Coding: BU X'1414'

Before Execution
PSWR 20001000

After Execution
PSWR 20001414

Note
The contents of bits 13-30 of the instruction replace the corresponding portion of the PSWR. The Condition Code remains unchanged.

**Example 2**

Memory Location: 01000
Hex Instruction: EC 10 14 14 (X=0, I=1)
Assembly Language Coding: BU *X'1414'

Before Execution
PSWR 80001000
Memory Word 01414

After Execution
PSWR 800015AC
Memory Word 01414

Note
The contents of bits 1-30 of memory word 01414 replace the previous contents of bits 1-4 and 13-31 of the PSWR.
DEFINITION

The Effective Address (bits 13-30) in the instruction is transferred to the corresponding bit positions in the Program Status Word Register (PSWR), if the condition specified by the D field (bits 6-8 of the instruction) is present. The seven specifiable conditions are tabulated below. If the condition is not as specified, the next instruction in sequence is executed. If the indirect bit of the Instruction Word is equal to one, and the branch occurs, bit positions 1-4 of the last memory word in the indirect chain are transferred to the corresponding bit positions of the PSWR. Bits 0, and 5-15 are unchanged.

<table>
<thead>
<tr>
<th>D Field (Hex)</th>
<th>Branch Condition (Branch if):</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CC1 = zero</td>
</tr>
<tr>
<td>2</td>
<td>CC2 = zero</td>
</tr>
<tr>
<td>3</td>
<td>CC3 = zero</td>
</tr>
<tr>
<td>4</td>
<td>CC4 = zero</td>
</tr>
<tr>
<td>5</td>
<td>CC2 and CC4 both = zero</td>
</tr>
<tr>
<td>6</td>
<td>CC3 and CC4 both = zero</td>
</tr>
<tr>
<td>7</td>
<td>CC1, CC2, CC3, and CC4 all = zero</td>
</tr>
</tbody>
</table>

CONDITION CODE

RESULTS

The resulting Condition Code remains unchanged if the indirect bit (bit 11) is equal to zero.

CC1: ISI (I) is equal to one and (EWL1) is equal to one
CC2: ISI (I) is equal to one and (EWL2) is equal to one
CC3: ISI (I) is equal to one and (EWL3) is equal to one
CC4: ISI (I) is equal to one and (EWL4) is equal to one

EXAMPLE

Memory Location: 02094
Hex Instruction: F1 00 21 4C (C1C2C3 = 2, X = 0, I = 0)
Assembly Language Coding: BCF 2,X'214C'

Before Execution
PSWR
10002094

After Execution
PSWR
1000214C

Note
Condition Code bit 2 is not set. The Effective Address (in this case bit 13-30 of the instruction) is transferred to the PSWR.
BCT

v,n,x

BRANCH CONDITION TRUE

ECOO

<table>
<thead>
<tr>
<th>D Field (Hex)</th>
<th>Branch Condition (Branch if):</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CC1=one</td>
</tr>
<tr>
<td>2</td>
<td>CC2=one</td>
</tr>
<tr>
<td>3</td>
<td>CC3=one</td>
</tr>
<tr>
<td>4</td>
<td>CC4=one</td>
</tr>
<tr>
<td>5</td>
<td>CC2 v CC4=one</td>
</tr>
<tr>
<td>6</td>
<td>CC3 v CC4=one</td>
</tr>
<tr>
<td>7</td>
<td>CC1 v CC2 v CC4=one</td>
</tr>
</tbody>
</table>

The Effective Address (bits 13-30) in the instruction is transferred to the corresponding bit positions in the Program Status Word Register (PSWR), if the condition specified by the D field (bits 6-8) is present. The seven specifiable conditions are tabulated below. If the indirect bit of the Instruction Word is equal to one, bit positions 1-4 of the last memory word in the indirect chain are transferred to the corresponding bit positions of the PSWR. Bits 0 and 5-12 are unchanged.

DEFINITION

RESULT

The resulting Condition Code remains unchanged if the indirect bit (bit 11) is equal to zero.

CC1: ISI (I) is equal to one and (EWL_1) is equal to one
CC2: ISI (I) is equal to one and (EWL_2) is equal to one
CC3: ISI (I) is equal to one and (EWL_3) is equal to one
CC4: ISI (I) is equal to one and (EWL_4) is equal to one

EXAMPLE

Memory Location: 01000
Hex Instruction: EC 80 14 14 (Condition=1, X=0, I=0)
Assembly Language Coding: BCT, 1,X'1414'

Before Execution
PSWR
50001000

After Execution
PSWR
50001414

Note
The contents of bits 13-30 of the instruction are transferred to bits 13-30 of the PSWR.
DEFINITION

The Effective Address (bits 13-30) in the instruction is transferred to the corresponding bit positions in the Program Status Word Register (PSWR) if the function bit in the mask register (R4) for the Condition Code, 1 of the 16 possible combinations of the 4 Condition Code bits which corresponds to the current condition code, is equal to one. The function \( F \) is defined by the 16 least significant bits of the mask register. All 16 Condition Codes of the 4 variables \( A = CC1 \), \( B = CC2 \), \( C = CC3 \), \( D = CC4 \) are defined below.

\[
F = \overline{ABC}D R4_{16} \overline{ABC}D R4_{17} \overline{ABC}D R4_{18} \overline{ABC}D R4_{19} \\
\overline{ABC}D R4_{20} \overline{ABC}D R4_{21} \overline{ABC}D R4_{22} \overline{ABC}D R4_{23} \\
\overline{ABC}D R4_{24} \overline{ABC}D R4_{25} \overline{ABC}D R4_{26} \overline{ABC}D R4_{27} \\
\overline{ABC}D R4_{28} \overline{ABC}D R4_{29} \overline{ABC}D R4_{30} \overline{ABC}D R4_{31}
\]

Therefore, any logical function of the four variables stored in the Condition Code register can be evaluated by storing the proper 16-bit function code in the mask register. The next instruction in sequence is executed if the function is equal to zero. If the Indirect bit of the instruction word is equal to one, bit positions 1-12 of the last memory word in the indirect chain are transferred to the corresponding bit positions of the PSWR. Bits 0 and 5 are unchanged.

SUMMARY

If \( F = 1 \) & \( I = 0 \), \( EA_{13-30} \rightarrow PSWR_{13-30} \)

If \( F = 1 \) & \( I = 1 \), \( EA_{1-30} \rightarrow PSWR_{1-30} \)

If \( F = 0 \) \( PSWR_{13-30} + 129 \rightarrow PSWR_{13-30} \)

CONDITION CODE

The resulting condition code remains unchanged if the indirect bit (bit 11) is equal to zero.

\( CC1 \): ISI (I) is equal to one and \( EA_1 \) is equal to one
\( CC2 \): ISI (I) is equal to one and \( EA_2 \) is equal to one
\( CC3 \): ISI (I) is equal to one and \( EA_3 \) is equal to one
\( CC4 \): ISI (I) is equal to one and \( EA_4 \) is equal to one

EXAMPLE

Memory Location: 01000
Hex Instruction: FO 00 20 00 (X=0, I=0)
Assembly Language Coding: BFT '2000'

Before Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>70001000</td>
<td>00000002</td>
</tr>
</tbody>
</table>

After Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>70002000</td>
<td>00000002</td>
</tr>
</tbody>
</table>

Note

Bit 30 of GPR4 defines a function for which \( CC1=CC2=CC3=1,CC4=0 \). This function is true, so a branch is effected.
BL
*m,x

BRANCH AND LINK

F880

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

DEFINITION
The contents of the Program Status Word Register (PSWR) are incremented by one word and transferred to General Purpose Register 0. If the indirect bit of the Instruction Word is equal to zero, the Effective Address (bit 13-30) is transferred to the corresponding bit positions of the PSWR. Bit positions 1-12 of the PSWR remain unchanged. If the indirect bit of the Instruction Word is equal to one, the Effective Address is also transferred to the corresponding bit positions of the PSWR.

SUMMARY

EXPRESSION

\[(PSWR) \rightarrow RO\]

CONDITION CODE

RESULTS

If the indirect bit is equal to zero, the Condition Code remains unchanged.

CC1: \((ISI)(I)\) is equal to one and \((EWL1)\) is equal to one
CC2: \((ISI)(I)\) is equal to one and \((EWL2)\) is equal to one
CC3: \((ISI)(I)\) is equal to one and \((EWL3)\) is equal to one
CC4: \((ISI)(I)\) is equal to one and \((EWL4)\) is equal to one

EXAMPLE

Memory Location: 0894C
Hex Instruction: F8 80 A3 78 \((X=0, I=0)\)
Assembly Language Coding: BL X'A378'

Before Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPRO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000894C</td>
<td>12345678</td>
</tr>
</tbody>
</table>

After Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPRO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000A378</td>
<td>10008950</td>
</tr>
</tbody>
</table>

Note
The contents of the PSWR are transferred to GPRO. The contents of bits 13-30 of the instruction are transferred to bits 13-30 of the PSWR.
**BRANCH AFTER INCREMENTING BYTE**

F400

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>R</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

**DEFINITION**
The contents of the GPR specified by R are incremented in bit position 31. If the result is nonzero the Effective Address (EA) is transferred to the Program Status Word Register (PSWR) bit positions 13-30 and bit positions 1-4 of the PSWR remain unchanged. If the result is equal to zero after incrementing, the next instruction is executed. Bits 0 and 5 are unchanged.

**SUMMARY**

\[(R) + 1_{31} \rightarrow R\]

**CONDITION CODE RESULTS**
- CC1: No change
- CC2: No change
- CC3: No change
- CC4: No change

**EXAMPLE**
- Memory Location: 1B204
- Hex Instruction: F4 01 B1 A8 (R=0, I=0)
- Assembly Language Coding: BIB 0,X'1B1AB'

**Before Execution**
- PSWR: 2001B204
- GPRO: FFFFFFFF

**After Execution**
- PSWR: 2001B208
- GPRO: 00000000

**Notes**
1. The contents of the GPRO are incremented by one at bit position 31. Since the result is zero, no branch occurs.
2. Indexing is not allowed.
3. If the indirect bit of the Instruction Word is equal to one, and the branch occurs, bit positions 1-4 of the last memory word in the indirect chain are transferred to the corresponding bit positions of the PSWR. Bits 0 and 5-12 are unchanged.
4. The instruction following may not be the target of the System Control Panel or Serial Control Panel Halt.
**DEFINITION**

The contents of the GPR specified by R are incremented in bit position 30. If the result is nonzero the Effective Address (EA) is transferred to the Program Status Word Register (PSWR) bit positions 13-30 and bit positions 1-4 of the PSWR remain unchanged. If the result is equal to zero after incrementing, the next instruction is executed.

**SUMMARY**

\[(R) + 130 \rightarrow R\]

**CONDITION CODE RESULTS**

- CC1: No change
- CC2: No change
- CC3: No change
- CC4: No change

**EXAMPLE**

<table>
<thead>
<tr>
<th>Memory Location:</th>
<th>039A0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex Instruction:</td>
<td>F5 20 39 48 (R=2, I=0)</td>
</tr>
<tr>
<td>Assembly Language Coding:</td>
<td>BIH 2,X'3948'</td>
</tr>
</tbody>
</table>

**Before Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>100039A0</td>
<td>FFFFD72A</td>
</tr>
</tbody>
</table>

**After Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>10003948</td>
<td>FFFFD72C</td>
</tr>
</tbody>
</table>

**Notes**

1. The contents of GPR2 are incremented by one in bit position 30. The result is replaced in GPR2 and a branch occurs to address 03948.
2. Indexing is not allowed.
3. If the indirect bit of the Instruction Word is equal to one, and the branch occurs, bit positions 1-4 of the last memory word in the indirect chain are transferred to the corresponding bit positions of the PSWR. Bits 0 and 5-12 are unchanged.
4. The instruction following may not be the target of the System Control Panel or Serial Control Panel Halt.
The contents of the GPR specified by R are incremented in bit position 29. If the result is nonzero, the Effective Address (EA) is transferred to the Program Status Word Register (PSWR) bit positions 13-30 and bit positions 1-4 of the PSWR remain unchanged. If the result is equal to zero after incrementing, the next instruction is executed.

\[
(R) + 1_{29} \rightarrow R
\]

EA \rightarrow PSWR_{13-30}, \text{ if result} \neq 0

**Notes**
1. The content of GPR6 is incremented by one at bit position 29, and the result is transferred to GPR6. The Effective Address of the BIW instruction, \(04\text{B}2\text{C}\), replaces the previous contents of the PSWR, bits 12-30.

2. Indexing is not allowed.

3. If the indirect bit of the Instruction Word is equal to one, and the branch occurs, bit positions 1-4 of the last memory word in the direct chain are transferred to the corresponding bit positions of the PSWR. Bits 0 and 5-12 are unchanged.

4. The instruction following may not be the target of the System Control Panel or Serial Control Panel Halt.
BRANCH AFTER INCREMENTING DOUBLEWORD

F460

<table>
<thead>
<tr>
<th>1 1 1 1 0 1</th>
<th>R</th>
<th>1 1 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DEFINITION
The contents of the GPR specified by R are incremented in bit position 28. If the result is nonzero the Effective Address (EA) is transferred to the Program Status Word Register (PSWR) bit positions 13-30 and bit positions 1-4 of the PSWR remain unchanged. If the result is equal to zero after incrementing, the next instruction is executed.

SUMMARY
(R) + 128 → R

EXPRESSION
EA → PSWR13-30, if result ≠ 0

CONDITION CODE
CC1: No change
CC2: No change
CC3: No change
CC4: No change

RESULTS

EXAMPLE
Memory Location: 0930C
Hex Instruction: F5 EO 91 A6 (R=3, I=0)
Assembly Language Coding: BID 3,X'91A6'

Before Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0800930C</td>
<td>FFFFFFF8</td>
</tr>
</tbody>
</table>

After Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>08009310</td>
<td>00000000</td>
</tr>
</tbody>
</table>

Notes
1. The content of GPR3 is incremented by one at bit position 28 and replaced. Since the result is zero, no branch occurs.
2. Indexing is not allowed.
3. If the indirect bit of the Instruction Word is equal to one, and the branch occurs, bit positions 1-4 of the last memory word in the direct chain are transferred to the corresponding bit positions of the PSWR. Bits 0 and 5-12 are unchanged.
4. The instruction following may not be the target of the System Control Panel or Serial Control Panel Halt.
Compare instructions provide the capability of comparing data in memory and General Purpose Registers. These operations can be performed on bytes, halfwords, words, or doublewords. Provisions have also been made to allow the result of compare operations to be masked with the contents of the Mask register before final testing.

The Compare instruction group uses three instruction formats.

### Memory Reference

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>F</th>
<th>WORD ADDRESS</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5</td>
<td>6 7 8</td>
<td>9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29</td>
<td>30 31</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Bits 0-5 define the Operation Code.
- Bits 6-8 designate a General Purpose Register address (0-7).
- Bits 9-10 designate one of three index registers.
- Bit 11 indicates whether an indirect addressing operation is to be performed.
- Bit 12-31 specify the address of the operand when the X and I fields equal to zero.

Note: Additional information on the Memory Reference instruction format is included with the Load/Store instruction formats.

### Immediate

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>AUG CODE</th>
<th>OPERAND VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5</td>
<td>6 7 8</td>
<td>9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29</td>
<td>30 31</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Bits 0-5 define the Operation Code.
- Bits 6-8 designate a General Purpose Register address (0-7).
- Bits 9-12 unassigned.
- Bits 16-31 contain the 16-bit operand value.
Bits 0-5 define the Operation Code.

Bits 6-8 designate the register to contain the result of the operation.

Bits 9-11 designate the register which contains the source operand.

Bits 12-15 define the Augmenting Operation Code.

A Condition Code is set during most Compare instructions to indicate whether the operation produced a result greater than, less than, or equal to zero.
COMPARE ARITHMETIC WITH MEMORY BYTE

9008

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

DEFINITION
The byte in memory specified by the Effective Byte Address (EBA) is accessed, right justified, and subtracted algebraically from the word in the GPR specified by R. The result of the subtraction causes one of the Condition Code bits (2-4) to be set. The contents of the GPR specified by R and the byte specified by the EBA remain unchanged.

SUMMARY EXPRESSION
(R) - (EBL) → SCC2,4

CONDITION CODE
CC1: Always zero
CC2: ISI (R) is greater than (EBL)
CC3: ISI (R) is less than (EBL)
CC4: ISI (R) is equal to (EBL)

RESULTS

EXAMPLE
Memory Location: 01000
Hex Instruction: 90 88 10 B5 (R=1, X=0, i=0)
Assembly Language Coding: CAMB 1,X'10B5'

Before Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR1</th>
<th>Memory Byte 010B5</th>
</tr>
</thead>
<tbody>
<tr>
<td>08001000</td>
<td>000000B6</td>
<td>C7</td>
</tr>
</tbody>
</table>

After Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR1</th>
<th>Memory Byte 010B5</th>
</tr>
</thead>
<tbody>
<tr>
<td>10010004</td>
<td>000000B6</td>
<td>C7</td>
</tr>
</tbody>
</table>

Note
CC3 is set, indicating that the contents of GPR1 are less than the contents of memory byte 010B5.
COMPARE ARITHMETIC WITH MEMORY HALFWORD

9000

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

### DEFINITION
The halfword in memory specified by the Effective Halfword Address (EHA) is accessed, and the sign bit is extended 16 bits to the left to form a word. The resulting word is subtracted algebraically from the word in the GPR specified by R. The result of the subtraction causes one of the Condition Code bits (2-4) to be set. The word in the GPR specified by R and the halfword specified by the EHA remain unchanged.

### SUMMARY
\[(R) - (EHL)_{SE} \rightarrow SCC_{2-4}\]

### CONDITION CODE RESULTS
- **CC1**: Always zero
- **CC2**: ISI \((R)\) is greater than \((EHL)_{SE}\)
- **CC3**: ISI \((R)\) is less than \((EHL)_{SE}\)
- **CC4**: ISI \((R)\) is equal to \((EHL)_{SE}\)

### EXAMPLE
- **Memory Location**: 0379C
- **Hex Instruction**: 92 00 39 77 \((R=4, X=0, I=0)\)
- **Assembly Language Coding**: CAMH 4,X'3976'

Before Execution
- PSWR: 0800379C
- GPR4: 00008540
- Memory Halfword: 03976
- 8640

After Execution
- PSWR: 200037AO
- GPR4: 00008540
- Memory Halfword: 03976
- 8640

**Note**: CC2 is set indicating that the contents of GPR4 are greater than the contents of memory halfword 03976 (a negative value).
COMPARE ARITHMETIC WITH MEMORY WORD

\[
\text{CAMW} \quad d,m,x
\]

9000

\[
\begin{array}{cccccccccccccccccccccccccccccc}
1 & 0 & 1 & 0 & 0 & R & X & 1 & 0 & \text{WORD OPERAND ADDRESS} & 0 & 0
\end{array}
\]

**DEFINITION**
The word in memory specified by the Effective Word Address (EWA) is accessed and subtracted algebraically from the word in the GPR specified by R. The result of the subtraction causes one of the Condition Code bits (2-4) to be set. The word in the GPR specified by R and the word specified by the EWA remain unchanged.

**SUMMARY**
\[(R) - (EWL) \rightarrow SCC2-4\]

**CONDITION CODE**

- **CC1**: Always zero
- **CC2**: ISI (R) is greater than (EWL)
- **CC3**: ISI (R) is less than (EWL)
- **CC4**: ISI (R) is equal to (EWL)

**RESULTS**

- **CC1**: Always zero
- **CC2**: ISI (R) is greater than (EWL)
- **CC3**: ISI (R) is less than (EWL)
- **CC4**: ISI (R) is equal to (EWL)

**MEMORY LOCATION**: 05B20

**Hex Instruction**: 93 00 5C 78 (R=6, X=0, I=0)

**Assembly Language Coding**: CAMW 6,'5C78'

**EXAMPLE**

**Before Execution**

- **PSWR**: 40005B20
- **GPR6**: 9E03B651
- **Memory Word**: 05C78

**After Execution**

- **PSWR**: 9E03B651
- **GPR6**: 9E03B651
- **Memory Word**: 05C78

**Note**: CC3 is set indicating that the contents of the GPR6 are less than the contents of memory word 05C78.
COMPARE ARITHMETIC WITH MEMORY DOUBLEWORD

The doubleword in memory specified by the Effective Doubleword Address (EDA) is accessed and subtracted algebraically from the doubleword in the GPR specified by R and R+1. R+1 is the GPR one greater than specified by R. The result of the subtraction causes one of the Condition Code bits (2-4) to be set. The doubleword in the GPR specified by R and R+1, and the doubleword specified by the EDA remain unchanged.

CONDITION CODE RESULTS

CC1: Always zero
CC2: ISI (R, R+1) is greater than (EDL)
CC3: ISI (R, R+1) is less than (EDL)
CC4: ISI (R, R+1) is equal to (EDL)

EXAMPLE

Memory Location: 27C14
Hex Instruction: 92 02 7F 52 (R=4, X=0, I=0)
Assembly Language Coding: CAMD 4,X'27F50'
COMPARE ARITHMETIC WITH REGISTER

1000

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
</tr>
</tbody>
</table>

**DEFINITION**
The word in the GPR specified by Rs is subtracted algebraically from the word in the GPR specified by Rd. The result of the subtraction causes one of the Condition Code bits (2-4) to be set. The words specified by Rs and Rd remain unchanged.

**SUMMARY**
(Rd) - (Rs) → SCC2-4

**CONDITION CODE RESULTS**
CC1: Always zero
CC2: ISI (Rd) is greater than (Rs)
CC3: ISI (Rd) is less than (Rs)
CC4: ISI (Rd) is equal to (Rs)

**EXAMPLE**
Memory Location: OB3C2
Hex Instruction: 10 10 (Rd=0, Rs=1)
Assembly Language Coding: CAR 1,0

Before Execution:
- PSWR: 0800B3C2
- GPRO: 58DF620A
- GPR1: 6A92B730

After Execution:
- PSWR: 1000B3C4
- GPRO: 58DF620A
- GPR1: 6A92B730

**Note**
CC3 is set indicating that the contents of GPRO are less than the contents of GPR1.
The sign bit (bit 16) of the immediate operand is extended 16 bit positions to the left to form a word. This word is subtracted from the word in the GPR specified by R. The result of the subtraction causes one of the Condition Code bits (2-4), to be set. The word in the GPR specified by R and the immediate operand (bit 16-31) remain unchanged.

\[(R) - (IW_{16-31})_{SE} \rightarrow SCC_{2-4}\]

**Condition Code Results**
- CC1: Always zero
- CC2: ISI (R) is greater than \((IW_{16-31})_{SE}\)
- CC3: ISI (R) is less than \((IW_{16-31})_{SE}\)
- CC4: ISI (R) is equal to \((IW_{16-31})_{SE}\)

**Example**
- Memory Location: 0A794
- Hex Instruction: C8 85 71 A2 (R=1)
- Assembly Language Coding: CI 1,X'71A2'

**Before Execution**
- PSWR: GPR1
- 4000A794: 00005719

**After Execution**
- PSWR: GPR1
- 1000A798: 00005719

**Note**
CC3 is set, indicating that the contents of GPR1 are less than the immediate operand.
### DEFINITION

The byte in memory specified by the Effective Byte Address (EBA) is accessed, and 24 zeros are appended to the most significant end to form a word. This word is logically compared (Exclusive OR Function) with the word in the GPR specified by R. The resulting word is then masked (Logical AND Function) with the contents of the Mask register (R4). The masked result is tested and Condition Code bit 4 is set if all 32 bits equal zero. The word in the GPR specified by R and the byte specified by the EBA remain unchanged.

### SUMMARY

\[ (R) \oplus 0_{0-23}, (EBA) \] \& (R4) \rightarrow SCC4

### CONDITION CODE

- **CC1:** Always zero
- **CC2:** Always zero
- **CC3:** Always zero
- **CC4:** ISI Result is equal to zero

### RESULTS

### EXAMPLE

- **Memory Location:** 00800
- **Hex Instruction:** 94 08 09 17 \( (R=0, X=0, I=0) \)
- **Assembly Language Coding:** CMMB 0, R'917'

<table>
<thead>
<tr>
<th>Before Execution</th>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR 10000800</td>
<td>PSWR 08000804</td>
</tr>
<tr>
<td>GPRO 000000A1</td>
<td>GPRO 000000A1</td>
</tr>
<tr>
<td>GPR4 000000F0</td>
<td>GPR4 000000F0</td>
</tr>
<tr>
<td>Memory Byte 00917</td>
<td>Memory Byte 00917</td>
</tr>
</tbody>
</table>

**Note**

The contents of GPRO and memory byte 00917 are identical in those bit positions specified by the contents of GPR4. CC4 is set.
COMPARE MASKED WITH MEMORY HALFWORD

9400

<table>
<thead>
<tr>
<th>1 0 0 1 0 1</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DEFINITION
The halfword in memory specified by the Effective Halfword Address (EHA) is accessed, and the sign (bit 16) is extended 16 bits to the left to form a word. The resulting word is logically compared (Exclusive OR Function) with the word in the GPR specified by R. The resulting word is then masked (Logical AND Function) with the contents of the Mask register (R4). The masked result is tested and Condition Code bit 4 is set if all 32 bits equal zero. The word in the GPR specified by R and the halfword specified by the EHA remain unchanged.

SUMMARY

EXPRESSION

\[(R) \oplus (EHL)_{SE} \& (R4) \rightarrow SCC_4\]

CONDITION CODE

RESULTS
CC1: Always zero
CC2: Always zero
CC3: Always zero
CC4: ISI result is equal to zero

EXAMPLE
Memory Location: 061B8
Hex Instruction: 95 00 62 93 (R=2, X=0, I=0)
Assembly Language Coding: CMMH 2,X '6293'

Before Execution
PSWR 100061B8
GPR2 09A043B6
GPR4 00004284
Memory Halfword 06292

After Execution
PSWR 080061BC
GPR2 09A043B6
GPR4 00004284
Memory Halfword 06292

Note
The contents of GPR2 and memory halfword 06292 are identical in those bit positions specified by the contents of GPR4. CC4 is set.
COMPARE MASKED WITH MEMORY WORD

9400

<table>
<thead>
<tr>
<th>1 0 0 1 0 1</th>
<th>R</th>
<th>X</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

**DEFINITION**

The word in memory specified by the Effective Word Address (EWA) is accessed and logically compared (Exclusive OR Function) with the word in the GPR specified by R. The result of the comparison is then masked (Logical AND Function) with the contents of the Mask register (R4). The masked result is tested and Condition Code bit 4 is set if all 32 bits equal zero. The word in the GPR specified by R and the word specified by the EWA remain unchanged.

**SUMMARY**

[(R) $\oplus$ (EWA)] & (R4) $\rightarrow$ SCC$_4$

**CONDITION CODE RESULTS**

- CC1: Always zero
- CC2: Always zero
- CC3: Always zero
- CC4: ISI result is equal to zero

**EXAMPLE**

Memory Location: 13A74
Hex Instruction: 97 01 3C 94 (R=6, X=0, I=0)
Assembly Language Coding: CMMW 6, X'3C94'

**Before Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
<th>GPR6</th>
<th>Memory Word 13C94</th>
</tr>
</thead>
<tbody>
<tr>
<td>08013A74</td>
<td>00FFFF00</td>
<td>132A1C04</td>
<td>472A3D04</td>
</tr>
</tbody>
</table>

**After Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
<th>GPR6</th>
<th>Memory Word 13C94</th>
</tr>
</thead>
<tbody>
<tr>
<td>00013A78</td>
<td>00FFFF00</td>
<td>132A1C04</td>
<td>472A3D04</td>
</tr>
</tbody>
</table>

**Note**

The contents of GPR6 and memory word 13C94 are not equal within the bit positions specified by the contents of GPR4.
COMPARE MASKED WITH MEMORY DOUBLEWORD

9400

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>0</th>
<th>DOUBLEWORD OPERAND ADDRESS</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

DEFINITION
The doubleword in memory specified by the Effective Doubleword Address (EDA) is accessed and compared (Exclusive OR Function) with the doubleword in the GPR specified by R and R+1. R+1 is the GPR one greater than specified by R. Each result from the comparison is then masked (Logical AND Function) with the contents of the Mask register (R4). The doubleword masked result is tested and Condition Code bit 4 is set if all 64 bits equal zero. The doubleword in the GPR specified by R and R+1 and the doubleword specified by the EDA remain unchanged.

CONDITION CODE
CC1: Always zero
CC2: Always zero
CC3: Always zero
CC4: ISI result is equal to zero

SUMMARY
EXPRESSION

\[ [R] \oplus (\text{EWL}) \oplus (R4), [(R+1) \oplus (\text{EWL+1}) \oplus (R4) \rightarrow \text{SCC}_4 \]

RESULT

EXAMPLE
Memory Location: 03000
Hex Instruction: 97 00 31 BA (R=6, X=0, I=0)
Assembly Language Coding: CMD 6,X'31BB'

Before Execution
PSWR 10003000
GPR4 000FFFFF
GPR6 FFF3791B
GPR7 890A45D6
Memory Word 031B8
0003791B

After Execution
PSWR 00003004
GPR4 000FFFFF
GPR6 FFF3791B
GPR7 890A45D6
Memory Word 031B8
0003791B

Note
The contents of GPR7 and memory word 031BC differ within the bit positions specified by the contents of GPR4.
COMPARE MASKED WITH REGISTER

1400

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>R₀</th>
<th>R₅</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
</tr>
</tbody>
</table>

**DEFINITION**

The word in the GPR specified by R₀ is logically compared (Exclusive OR function) with the word in the GPR specified by R₅. The result of the comparison is then masked (Logical AND function) with the contents of the Mask register (R₄). The result is tested and Condition Code bit 4 is set if all 32 bits equal zero. The words specified by R₅ and R₀ remain unchanged.

**SUMMARY**

\[ (R₀) \oplus (R₅) \] & (R₄) → SCC₄

**EXPRESSION**

**CONDITION CODE RESULTS**

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always zero</td>
<td>Always zero</td>
<td>Always zero</td>
<td>ISI result is equal to zero</td>
</tr>
</tbody>
</table>

**EXAMPLE**

Memory Location: 05002
Hex Instruction: XXXX14 A0 (R₀=1, R₅=2)
Assembly Language Coding: CMR 2,1

<table>
<thead>
<tr>
<th>Before Execution</th>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR</td>
<td>PSWR</td>
</tr>
<tr>
<td>GPR1</td>
<td>GPR1</td>
</tr>
<tr>
<td>GPR2</td>
<td>GPR2</td>
</tr>
<tr>
<td>GPR4</td>
<td>GPR4</td>
</tr>
<tr>
<td>100050D2</td>
<td>080050D4</td>
</tr>
<tr>
<td>583C94A2</td>
<td>583C94A2</td>
</tr>
<tr>
<td>0C68C5F6</td>
<td>0C68C5F6</td>
</tr>
<tr>
<td>AAAAAAAAAA</td>
<td>AAAAAAAAAA</td>
</tr>
</tbody>
</table>

**Note**

The contents of GPR1 and GPR2 are identical within the bit positions specified by the contents of GPR4. CC4 is set.
The Logical instruction group provides the capability of performing AND, OR, and Exclusive OR operations on bytes, halfwords, and doublewords in memory and General Purpose Registers. Provisions have also been made to allow the result of Register-to-Register OR and Exclusive OR operations to be masked with the contents of Mask register (R4) before final storage.

The Logical instruction group uses the following two instruction formats:

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>F</th>
<th>WA</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 0-5 define the Operation Code.
Bits 6-8 designate a General Purpose Register address (0-7).
Bits 9-10 designate one of three index registers.
Bit 11 indicates whether an indirect addressing operation is to be performed.
Bits 12-31 specify the address of the operand when the X and I fields are equal to zero.

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R_D</th>
<th>R_S</th>
<th>AUG CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 0-5 define the Operation Code.
Bits 6-8 designate the register to contain the result of the operation.
Bits 9-11 designate the register which contains the source operand.
Bits 12-15 define the Augmenting Operation Code.

A Condition Code is set during execution of most Logical instructions to indicate whether the result of that operation was greater than, less than, or equal to zero.
AND MEMORY BYTE

8408

<table>
<thead>
<tr>
<th>100001</th>
<th>R</th>
<th>X</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

DEFINITION
The byte in memory specified by the Effective Byte Address (EBA) is accessed and logically ANDed with the least significant byte (bits 24-31) of the GPR specified by R. The result is transferred to bit positions 24-31 of the GPR specified by R. Bit positions 0-23 of the GPR specified by R remain unchanged.

SUMMARY
(EBL) & (R_{24-31}) \rightarrow R_{24-31}
R_{0-23} Unchanged

CONDITION CODE
CC1: Always zero
CC2: ISI R_{24-31} is greater than zero
CC3: Always zero
CC4: ISI R_{24-31} is equal to zero

RESULTS

EXAMPLE
Memory Location: 00200
Hex Instruction: 84 88 03 73 (R=1, X=0, I=0)
Assembly Language Coding: ANMB 1,X'373'

Before Execution
PSWR 00000200 GPRI 36AC718F Memory Byte 00373 C7

After Execution
PSWR 20000204 GPRI 36AC7187 Memory Byte 00373 C7

Note
The contents of memory byte 00373 are ANDed with the right-hand byte of GPRI, and the result replaces the byte in GPRI. CC2 is set.
ANMH d,*m,x

AND MEMORY HALFWORD

8400

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>R</th>
<th>X</th>
<th>1</th>
<th>0</th>
<th>HALFWORD OPERAND ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

**DEFINITION**
The halfword in memory specified by the Effective Halfword Address (EHA) is accessed and logically ANDed with the least significant halfword (bits 16-31) of the GPR specified by R. The result is transferred to bit positions 16-31 of the GPR specified by R. Bit positions 0-15 of the GPR remain unchanged.

**SUMMARY**

\[ (EHL) \& (R_{16-31}) \rightarrow R_{16-31} \]

**CONDITION CODE**

CC1: Always zero
CC2: ISI \( R_{16-31} \) is greater than zero
CC3: Always zero
CC4: ISI \( R_{16-31} \) is equal to zero

**RESULTS**

**EXAMPLE**

Memory Location: \( 0100 \)
Hex Instruction: \( 87 \ 00 \ 12 \ A3 \) (\( R=6, \ X=0, \ I=0 \))
Assembly Language Coding: \( \text{ANMH} \ 6, X'12A2' \)

Before Execution

PSWR 40001000 GPR6 4F638301 Memory Halfword 012A2

After Execution

PSWR 08001004 GPR6 4F630000 Memory Halfword 012A2

Note

The contents of memory halfword 012A2 are ANDed with the right halfword of GPR6, and the result replaces the halfword in GPR6. CC4 is set.
### Definition
The word in memory specified by the Effective Word Address (EWA) is accessed and logically ANDed with the word located in the GPR specified by R.

### Summary
\[(EWA) \& (R) \rightarrow R\]

### Condition Code
- **CC1:** Always zero
- **CC2:** ISI R\(_{0-31}\) is greater than zero
- **CC3:** ISI R\(_{0-31}\) is less than zero
- **CC4:** ISI R\(_{0-31}\) is equal to zero

### Example

#### Memory Location:
- OOF1C

#### Hex Instruction:
- 87 80 0F DO (R=7, X=0, I=0)

#### Assembly Language Coding:
- ANMW 7,'FDO'

<table>
<thead>
<tr>
<th>Before Execution</th>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR</td>
<td>PSWR</td>
</tr>
<tr>
<td>GPR7</td>
<td>GPR7</td>
</tr>
<tr>
<td>Memory Word OOFDO</td>
<td>Memory Word OOFDO</td>
</tr>
<tr>
<td>08000F1C</td>
<td>10000F20</td>
</tr>
<tr>
<td>FOFOFOFO</td>
<td>90D03050</td>
</tr>
<tr>
<td>9ED13854</td>
<td>9ED13854</td>
</tr>
</tbody>
</table>

#### Memory Word
- OOFDO 9ED13854

### Note
The contents of memory word OOFDO are ANDed with the contents of GPR7, and the result replaces the contents of that register. CC3 is set.
The doubleword in memory specified by the Effective Doubleword Address (EDA) is accessed and logically ANDed with the doubleword in the GPR specified by R and R+1. R+1 is the GPR one greater than specified by R. The resulting doubleword is transferred to the GPR specified by R and R+1.

\[(EWL+1)\&(R+1) \rightarrow R+1\]

\[(EWL)\&(R) \rightarrow R'\]

**CONDITION CODE RESULTS**

- **CC1:** Always zero
- **CC2:** ISI \((R,R+1)\) is greater than zero
- **CC3:** ISI \((R,R+1)\) is less than zero
- **CC4:** ISI \((R,R+1)\) is equal to zero

**EXAMPLE**

- **Memory Location:** 00674
- **Hex Instruction:** 86 00 08 1A \((R=4, X=0, I=0)\)
- **Assembly Language Coding:** \texttt{ANMD 4,'X'818'}

**Before Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
<th>GPR5</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000674</td>
<td>9045C64A</td>
<td>32B08F00</td>
</tr>
</tbody>
</table>

**Memory Word 00818**

| 684A711C | Memory Word 0081C |
| 8104A2BC |

**After Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
<th>GPR5</th>
</tr>
</thead>
<tbody>
<tr>
<td>20000678</td>
<td>00404008</td>
<td>00008200</td>
</tr>
</tbody>
</table>

**Memory Word 00818**

| 684A711C | Memory Word 0081C |
| 8104A2BC |

**Note**

The contents of memory word 00818 are ANDed with the contents of GPR4, and the result replaces the contents of GPR4. The contents of memory word 0081C are ANDed with the contents of GPR5, and the result replaces the contents of GPR5. CC2 is set.
AND REGISTER AND REGISTER

0400

**DEFINITION**
The word in the GPR specified by \( R_D \) is logically ANDed with the word in the GPR specified by \( R_S \). The resulting word is transferred to the GPR specified by \( R_D \).

**SUMMARY**
\( (R_S) \& (R_D) \rightarrow R_D \)

**CONDITION CODE**
- **CC1**: Always zero
- **CC2**: ISI \( (R_D) \) is greater than zero
- **CC3**: ISI \( (R_D) \) is less than zero
- **CC4**: ISI \( (R_D) \) is equal to zero

**RESULTS**

**EXAMPLE**

Memory Location: O3812
Hex Instruction: 04 F0 \( (R_D=1, R_S=7) \)
Assembly Language Coding: ANR 7,1

Before Execution
- PSWR: 40003812
- GPR1: AC881101
- GPR7: 000FFFFF

After Execution
- PSWR: 20003814
- GPR1: 00081101
- GPR7: 000FFFFF

Note
The contents of GPR1 and GPR7 are ANDed, and the result is transferred to GPR1. CC2 is set.
ORMB

**DEFINITION**
The byte in memory specified by the Effective Byte Address (EBA) is accessed and logically ORed with the least significant byte (bits 24-31) of the GPR specified by R. The resulting byte is transferred to bit positions 24-31 of the GPR specified by R. Bit positions 0-23 of the GPR specified by R remain unchanged.

**SUMMARY EXPRESSION**
\((\text{EBA}) \lor (R_{24-31}) \rightarrow R_{24-31}\)

**CONDITION CODE RESULTS**
- **CC1**: Always zero
- **CC2**: ISI R0-31 is greater than zero
- **CC3**: ISI R0-31 is less than zero
- **CC4**: ISI R0-31 is equal to zero

**EXAMPLE**
- **Memory Location**: 00600
- **Hex Instruction**: 88 88 08 A3 \((R=1, X=0, I=0)\)
- **Assembly Language Coding**: ORMB 1, '8A3'

<table>
<thead>
<tr>
<th>Before Execution</th>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR</td>
<td>000000600</td>
</tr>
<tr>
<td>GPR1</td>
<td>40404040</td>
</tr>
<tr>
<td>Memory Byte 8A3</td>
<td>Memory Byte 8A3</td>
</tr>
</tbody>
</table>

**Note**
The contents of memory byte 8A3 are logically ORed with the right-hand byte of GPR1, and the result replaces that byte in GPR2. CC2 is set.
DEFINITION
The halfword in memory specified by the Effective Halfword Address (EHA) is accessed and logically ORed with the least significant halfword (bits 16-31) of the GPR specified by R. The resulting halfword is transferred to bit positions 16-31 of the GPR specified by R. Bit positions 0-15 of the GPR specified by R remain unchanged.

SUMMARY
\[(EHL)v(R_{16-31}) \rightarrow R_{16-31}\]

CONDITION CODE
CC1: Always zero
CC2: ISI $R_{0-31}$ is greater than zero
CC3: ISI $R_{0-31}$ is less than zero
CC4: ISI $R_{0-31}$ is equal to zero

EXAMPLE
Memory Location: 018AC
Hex Instruction: 8B 00 19 45 (R=6, X=0, I=0)
Assembly Language Coding: ORMH 6,X'1944'

Before Execution
- PSWR: 000018AC
- GPR6: BD71A4C6
- Memory Halfword 01944

After Execution
- PSWR: 100018B0
- GPR6: BD71E5F7
- Memory Halfword 01944

Note
The contents of memory halfword 01944 are ORed with the right halfword from GPR6, and the result replaces that halfword in GPR6. CC3 is set.
OR MEMORY WORD

8800

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

DEFINITION

The word in memory specified by the Effective Word Address (EWA) is accessed and logically ORed with the word in the GPR specified by R. The result is transferred to the GPR specified by R.

SUMMARY EXPRESSION

\((EWA)_v(R) → R\)

CONDITION CODE

<table>
<thead>
<tr>
<th>RESULTS</th>
<th>CONDITION CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC1: Always zero</td>
<td></td>
</tr>
<tr>
<td>CC2: ISI (R_{0-31}) is greater than zero</td>
<td></td>
</tr>
<tr>
<td>CC3: ISI (R_{0-31}) is less than zero</td>
<td></td>
</tr>
<tr>
<td>CC4: ISI (R_{0-31}) is equal to zero</td>
<td></td>
</tr>
</tbody>
</table>

EXAMPLE

Memory Location: 05000
Hex Instruction: 89 80 52 0C (\(R=3, X=0, I=0\))
Assembly Language Coding: ORMW 3,X'52OC'

Before Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR</th>
<th>Memory Word 0520C</th>
</tr>
</thead>
<tbody>
<tr>
<td>40005000</td>
<td>88888888</td>
<td>OEDC4657</td>
</tr>
</tbody>
</table>

After Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR3</th>
<th>Memory Word 0520C</th>
</tr>
</thead>
<tbody>
<tr>
<td>10005004</td>
<td>BDCEDF</td>
<td>OEDC4657</td>
</tr>
</tbody>
</table>

Note: The contents of memory word 0520C are ORed with the contents of GPR3, and the result is transferred to GPR3. CC3 is set.
OR MEMORY DOUBLEWORD

8800

DEFINITION
The doubleword in memory specified by the Effective Doubleword Address (EDA) is accessed and logically ORed with the doubleword in the GPR specified by R and R+1. R+1 is the GPR one greater than specified by R. The result is transferred to the GPR specified by R and R+1.

SUMMARY
(EXL+1)v(R+1) → R+1
(EXL)v(R) → R

CONDITION CODE
CC1: Always zero
CC2: ISI (R,R+1) is greater than zero
CC3: ISI (R,R+1) is less than zero
CC4: ISI (R,R+1) is equal to zero

RESULTS
(EXL+1)v(R+1) → R+1
(EXL)v(R) → R

EXAMPLE
Memory Location:
PSWR 00B68
GPR6 002A0031
GPR7 00100039
Memory Word 00C30 09002400
Memory Word 00C34 09102439

Before Execution
Assembly Language Coding: ORMD 6,X'C30'
Hex Instruction: 8B 00 0C 32 (R=6, X=0, I=0)

After Execution

Note
The contents of memory word 00C30 are ORed with the contents of GPR6, and the result is transferred to GPR7. The contents of memory word 00C34 are ORed with the contents of GPR7, and the result is transferred to GPR7. CC2 is set.
The word in the GPR specified by \( R_D \) is logically ORed with the word in the GPR specified by \( R_S \). The result is transferred to the GPR specified by \( R_D \).

\[(R_S) \lor (R_D) \rightarrow R_D\]

**CONDITION CODE RESULTS**

- CC1: Always zero
- CC2: \( \text{ISI} (R_D) \) is greater than zero
- CC3: \( \text{ISI} (R_D) \) is less than zero
- CC4: \( \text{ISI} (R_D) \) is equal to zero

**EXAMPLE**

Memory Location: 00F8A
Hex Instruction: 08 A0 (\( R_D=1 \), \( R_S=2 \))
Assembly Language Coding: ORR 2,1

Before Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR1</th>
<th>GPR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>0001D63F</td>
<td>88880000</td>
</tr>
</tbody>
</table>

After Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR1</th>
<th>GPR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>8889D635</td>
<td>88880000</td>
</tr>
</tbody>
</table>

**Note** The contents of GPR1 and GPR2 are ORed, and the result is transferred to GPR1. CC3 is set.
**OR REGISTER AND REGISTER MASKED**

**DEFINITION**

The word in the GPR specified by \( R_D \) is logically ORed with the word in the GPR specified by \( R_S \). The resulting word is then masked (Logical AND Function) with the contents of the Mask register (R4). The result is then transferred to the GPR specified by \( R_D \).

**SUMMARY**

\( (R_S)v(R_D)&(R4) \rightarrow R_D \)

**CONDITION CODE**

- **CC1**: Always zero
- **CC2**: ISI \( R_D \) is greater than zero
- **CC3**: ISI \( R_D \) is less than zero
- **CC4**: ISI \( R_D \) is equal to zero

**RESULTS**

**EXAMPLE**

- **Memory Location**: 03956
- **Hex Instruction**: OB 58 (\( R_D=6, R_S=5 \))
- **Assembly Language Coding**: ORRM 5,6

<table>
<thead>
<tr>
<th>Before Execution</th>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR</td>
<td>GPR4</td>
</tr>
<tr>
<td>08003956</td>
<td>EEEEEEE</td>
</tr>
<tr>
<td>PSWR</td>
<td>GPR4</td>
</tr>
<tr>
<td>10003958</td>
<td>EEEEEEE</td>
</tr>
</tbody>
</table>

**Note**

The contents of GPR5 and GPR6 are ORed; the result is ANDed with the contents of GPR4 and transferred to GPR6. CC3 is set.
EXCLUSIVE OR MEMORY BYTE

8C08

**DEFINITION**
The byte in memory specified by the Effective Byte Address (EBA) is accessed and logically Exclusive ORed with the least significant byte (bits 24-31) of the GPR specified by R. The result is transferred to bit positions 24-31 of the GPR specified by R. Bits 0-23 of the GPR specified by R remain unchanged.

**SUMMARY**
\[(EBA) \oplus (R_{24-31}) = R_{24-31}\]

**CONDITION CODE RESULTS**
- CC1: Always zero
- CC2: ISI \(R_0 - R_{31}\) is greater than zero
- CC3: ISI \(R_0 - R_{31}\) is less than zero
- CC4: ISI \(R_0 - R_{31}\) is equal to zero

**EXAMPLE**
- Memory Location: 012F8
- Hex Instruction: 8C 08 13 A1 (R=0, X=0, I=0)
- Assembly Language Coding: EOMB 0,X'13A1'

Before Execution
- PSWR: 000012F8
- GPRO: D396F458

After Execution
- PSWR: 100012FC
- GPRO: D396F4F1

**Note**
The contents of memory byte 013A1 are Exclusive ORed with the right-hand byte of GPRO; the result replaces that byte in GPRO. CC3 is set.
**EXCLUSIVE OR MEMORY HALFWORD**

8C00

<table>
<thead>
<tr>
<th>1 0 0 0 1 1</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>O</th>
<th>HALFWORD OPERAND ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
</tr>
</tbody>
</table>

**DEFINITION**
The halfword in memory specified by the Effective Halfword Address (EHA) is accessed and logically Exclusive ORed with the least significant halfword (bits 16-31) of the GPR specified by R. The result is transferred to bit positions 16-31 of the GPR specified by R. Bit positions 0-15 of the GPR specified by R remain unchanged.

**SUMMARY**
\[(EHL) \oplus (R_{16-31}) \rightarrow R_{16-31}\]

**EXPRESSION**
\[R_{0-15} \text{ Unchanged}\]

**CONDITION CODE**
- CC1: Always zero
- CC2: ISI R0-31 is greater than zero
- CC3: ISI R0-31 is less than zero
- CC4: ISI R0-31 is equal to zero

**RESULTS**
Memory Location: 00958
Hex Instruction: 8E 80 0A 41 (R=5, X=0, I=0)
Assembly Language Coding: EOMH 5,'A40'

**EXAMPLE**
Before Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR5</th>
</tr>
</thead>
<tbody>
<tr>
<td>40000958</td>
<td>96969696</td>
</tr>
<tr>
<td>Memory Halfword 00A40</td>
<td></td>
</tr>
</tbody>
</table>
After Execution
<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000095C</td>
<td>9696CA3D</td>
</tr>
<tr>
<td>Memory Halfword 00A40</td>
<td></td>
</tr>
</tbody>
</table>

**Note**
The contents of memory halfword 00A40 are Exclusive ORed with the right halfword of GPR5, and the result replaces that halfword in GPR5. CC3 is set.
**EXCLUSIVE OR MEMORY WORD**

**EOMW**

d,*m,x

**8C00**

<table>
<thead>
<tr>
<th>100011</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>0</th>
<th>WORD OPERAND ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DEFINITION**
The word in memory specified by the Effective Word Address (EWA) is accessed and logically Exclusive ORed with the word in the GPR specified by R. The result is transferred to the GPR specified by R.

**SUMMARY**

\[(EWA) \oplus (R) \rightarrow R\]

**CONDITION CODE**

- **CC1**: Always zero
- **CC2**: ISI \(R_{0-31}\) is greater than zero
- **CC3**: ISI \(R_{0-31}\) is less than zero
- **CC4**: ISI \(R_{0-31}\) is equal to zero

**RESULTS**

**EXAMPLE**

- **Memory Location**: 185BC
- **Hex Instruction**: 8F 81 86 94 (\(R=7, X=0, I=0\))
- **Assembly Language Coding**: EDM 7,X'18694'

**Before Execution**

- **PSWR**: 010185BC
- **GPR7**: 13579BDF
- **Memory Word 18694**: 22222222

**After Execution**

- **PSWR**: 200185C0
- **GPR7**: 3175B9FD
- **Memory Word 18694**: 22222222

**Note**
The contents of memory word 18694 are Exclusive ORed with the contents of GPR7. The result replaces the contents of GPR7. CC2 is set.
**EXCLUSIVE OR MEMORY DOUBLEWORD**

**DEFINITION**
The doubleword in memory specified by the Effective Doubleword Address (EDA) is accessed and logically Exclusive ORed with the doubleword in the GPR specified by R and R+1. R+1 is the GPR one greater than specified by R. The result is transferred to the GPR specified by R and R+1.

**SUMMARY**

\[(EWL+1) \oplus (R+1) \rightarrow R+1\]

**CONDITION CODE**
- CC1: Always zero
- CC2: ISI (R, R+1) is greater than zero
- CC3: ISI (R, R+1) is less than zero
- CC4: ISI (R, R+1) is equal to zero

**RESULTS**
- Memory Location:
  - PSWR
  - GPR6
  - GPR7
  - GPR7

- Assembly Language Coding:
  - EOMD 6,X'538'

**EXAMPLE**

Before Execution
- PSWR: 00000448
- GPR6: 00FFFO00
- Memory Word 00538
- 482144C0

After Execution
- PSWR: 00000448
- GPR6: 8F 00 05 3A (R=6, X=0, I=0)
- Memory Word 00538
- 2881433A

**Note**
The contents of memory word 00538 and GPR6 are Exclusive ORed and the result is transferred to GPR6. The contents of memory word 0053C and GPR7 are Exclusive ORed and the result is transferred to GPR7. CC2 is set.
EXCLUSIVE OR REGISTER AND REGISTER

OC00

<table>
<thead>
<tr>
<th>00011Rd</th>
<th>Rs</th>
<th>0000</th>
</tr>
</thead>
</table>

DEFINITION
The word in the GPR specified by Rd is logically Exclusive ORed with the word in the GPR specified by Rs. The result is transferred to the GPR specified by Rd.

SUMMARY
(Rs) \( \oplus \) (Rd) \rightarrow Rd

CONDITION CODE
CC1: Always zero
CC2: ISI (RD) is greater than zero
CC3: ISI (RD) is less than zero
CC4: ISI (RD) is equal to zero

RESULTS

EXAMPLE
Memory Location: 0139E
Hex Instruction: OF E0 (Rd=7, Rs=6)
Assembly Language Coding: EOR 6,7

Before Execution
PSWR: 0100139E
GPR6: 33333333
GPR7: 55555555

After Execution
PSWR: 200013A0
GPR6: 33333333
GPR7: 66666666

Note
The contents of GPR6 and GPR7 are Exclusive ORed, and the result is transferred to GPR7. CC2 is set.
EXCLUSIVE OR REGISTER AND REGISTER MASKED

DEFINITION
The word in the GPR specified by \( R_D \) is logically Exclusive ORed with the word in the GPR specified by \( R_S \). The resulting word is then masked (Logical AND Function) with the contents of the Mask register (R4). The result is transferred to the GPR specified by \( R_D \).

SUMMARY
\((R_S) \oplus (R_D) \land (R4) \rightarrow R_D\)

CONDITION CODE
CC1: Always zero
CC2: ISI \((R_D)\) is greater than zero
CC3: ISI \((R_D)^{\sim}\) is less than zero
CC4: ISI \((R_D)\) is equal to zero

RESULTS

EXAMPLE
Memory Location:
Hex Instruction:
Assembly Language Coding:

Before Execution
PSWR: 00025A32
GPR4: 00FEDFOO
GPR6: 9725A2C8
GPR7: 6C248237

After Execution
PSWR: 08025A34
GPR4: 00FEDFOO
GPR6: 9725A2C8
GPR7: 00000000

Note
The contents of GPR6 and GPR7 are Exclusive ORed. The result is ANDed with the contents of GPR4 and transferred to GPR7. CC4 is set.
This group of instructions provides the capability to perform Arithmetic, Logical, and Circular Left or Right shift operations on the contents of words or doublewords in General Purpose Registers. Provisions have also been made to allow Normalize operations to be performed on the contents of words or doublewords in General Purpose Registers.

The following two instruction formats are used by the Shift instruction group:

**SHIFT INFORMATION**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-5</td>
<td>define the Operation Code.</td>
</tr>
<tr>
<td>6-8</td>
<td>designate a General Purpose Register address (0-7).</td>
</tr>
<tr>
<td>9</td>
<td>designates direction.</td>
</tr>
<tr>
<td></td>
<td>( D=1 ) designates shift left</td>
</tr>
<tr>
<td></td>
<td>( D=0 ) designates shift right</td>
</tr>
<tr>
<td>10</td>
<td>unassigned.</td>
</tr>
<tr>
<td>11-15</td>
<td>define the number of shifts to be made.</td>
</tr>
</tbody>
</table>

**INTERREGISTER**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-5</td>
<td>define the Operation Code.</td>
</tr>
<tr>
<td>6-8</td>
<td>designate the register to contain the result of the operation.</td>
</tr>
<tr>
<td>9-11</td>
<td>designate the register which contains the source operand.</td>
</tr>
<tr>
<td>12-15</td>
<td>define the Augmenting Operation Code.</td>
</tr>
</tbody>
</table>

Most Shift instructions leave the Condition Code unchanged.
DEFINITION

The word in the GPR specified by RS is shifted left, 4 bit positions at a time, until the contents are normalized for the base 16 exponent.

The contents of RS are less than one or equal to or greater than 1/16 (1 > (RS ≥ 1/16).) The exponent is set to 40₁₆ and is decremented once for each group of 4 shifts performed. When normalization is complete, the exponent is stored in bit positions 25-31 of the GPR specified by RD.

Bit positions 0-24 of the GPR specified by RS are cleared to zeros. If the contents of the GPR specified by RS are equal to zero, the exponent stored in bit positions 25-31 of the GPR specified by RD will equal zero and no shifting will be performed.

Note

The normalized result must be converted to the format defined on page 6-171 prior to use by the floating-point arithmetic unit or standard FORTRAN floating-point subroutines. In addition, a test must be made for minus full scale (lXXX XXXX 0000 0000 --- 0000) and a conversion made to (lYYY YYYY 1111 0000 --- 0000), where YYYY is one less than XXX XXXX.

CONDITION CODE

RESULTS

CC1: No change
CC2: No change
CC3: No change
CC4: No change

EXAMPLE

Memory Location:
Hex Instruction:
Assembly Language Coding:

Before Execution
PSWR GPR1 GPR6
20000032 12345678 0002E915

After Execution
PSWR GPR1 GPR6
20000034 0000003D 2E915000

Note

The content of GPR6 is normalized by three left shifts of four bits each. The exponent is determined by decrementing 40₁₆ once for each shift and transferred to GPR1.
NORMALIZE DOUBLE

6400

<table>
<thead>
<tr>
<th>RS</th>
<th>RD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

DEFINITION

The doubleword in the GPR specified by Rs and Rs+1 is shifted left, 4 bit positions at a time, until the contents are normalized for the base 16 exponent \((1 > (\text{Rs}, \text{Rs}+1) \geq 1/16)\). The contents of Rs and Rs+1 are less than or equal to or greater than 1/16. Rs+1 is the GPR one greater than specified by Rs. The exponent of the doubleword is set to 40\text{H} and is decremented once for each group of four shifts performed. When normalization is complete, the exponent is stored in bit positions 25-31 of the GPR specified by Rd. Bit positions 0-24 of the GPR specified by Rd are cleared to zeros. If the contents of the doubleword specified by Rs and Rs+1 are equal to zero, the exponent stored in bit positions 25-31 of the GPR specified by Rd will equal zero, and no shifting will be performed.

Note

The normalized result must be converted to the format defined on page 6-171 prior to use by the floating-point arithmetic unit or standard FORTRAN floating-point subroutines. In addition, a test must be made for minus full scale (1XXX XXXX 0000 0000 --- 0000) and a conversion made to (1YYY YYYY 1111 0000 --- 0000), where \(YYY YYYY\) is one less than \(XXX XXXX\).

CONDITION CODE

<table>
<thead>
<tr>
<th>Code</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC1</td>
<td>No change</td>
</tr>
<tr>
<td>CC2</td>
<td>No change</td>
</tr>
<tr>
<td>CC3</td>
<td>No change</td>
</tr>
<tr>
<td>CC4</td>
<td>No change</td>
</tr>
</tbody>
</table>

RESULTS

Memory Location: 0046E
Hex Instruction: 67 10 (Rs=6, Rd=1)
Assembly Language Coding: NORD 6,1

EXAMPLE

Before Execution
- PSWR: 1000046E
- GPR1: 9ABCDEFO
- GPR6: FFFFFFFF
- GPR7: FF3AD915

After Execution
- PSWR: 10000470
- GPR1: 00000037
- GPR6: F3AD9150
- GPR7: 00000000

Note

The doubleword obtained from the contents of GPR6 and GPR7 is normalized by nine left shifts of four bit positions each. The result is returned to GPR6 and GPR7, and the exponent (40\text{H}-9) is transferred to GPR1.
SHIFT AND COUNT ZEROS

6800

**DEFINITION**

The word in the GPR specified by \( R_S \) is shifted left, one bit position at a time, until the sign (bit 0) changes from zero to one. The contents are then shifted left one more bit position, and the total number of shifts minus one is placed in bit positions 27-31 of the GPR specified by \( R_D \). Bit positions 0-26 of the GPR specified by \( R_D \) are set to zeros. The shift count specifies the most significant bit position (0-31) of \( R_S \) that was equal to one.

**NOTES**

1. If the contents of the GPR specified by \( R_S \) are equal to zero, the shift count placed in bit positions 27-31 of the GPR specified by \( R_D \) is zero, and Condition Code bit 4 is set to one.

2. If the sign (bit 0) of the GPR specified by \( R_S \) is equal to one, the shift count placed in bit positions 27-31 of the GPR specified by \( R_D \) is zero, and Condition Code bit 4 is set to zero.

**CONDITION CODE RESULTS**

- **CC1**: Always zero
- **CC2**: Always zero
- **CC3**: Always zero
- **CC4**: \( ISI \) if \( R_S \) 0-31 is equal to zero

**EXAMPLE**

<table>
<thead>
<tr>
<th>Memory Location:</th>
<th>0399E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex Instruction:</td>
<td>6A 20 (( R_S = 4 ), ( R_D = 2 ))</td>
</tr>
<tr>
<td>Assembly Language Coding:</td>
<td>SCZ 2,N</td>
</tr>
</tbody>
</table>

**Before Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR2</th>
<th>GPR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000399E</td>
<td>12345678</td>
<td>00000611</td>
</tr>
</tbody>
</table>

**After Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR2</th>
<th>GPR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>000039A0</td>
<td>0000000A</td>
<td>80308800</td>
</tr>
</tbody>
</table>

**Note**

The content of GPR4 are left shifted 10 bits when bit 0 is equal to one. The contents are then shifted one more bit position, and the zero count of 10 (\( A_H \)) is transferred to GPR2.
**SHIFT LEFT ARITHMETIC**

6C40

**DEFINITION**

Bit positions 1-31 of the GPR specified by R are shifted left the number of bit positions specified by the shift field (bits 11-15) in the Instruction Word. Bit position 0 (sign bit) of the GPR specified by R remains unchanged. Condition Code bit 1 is set to one if any bit shifted out of position 1 differs from the sign bit.

**CONDITION CODE**

- **CC1:** ISI arithmetic exception
- **CC2:** Always zero
- **CC3:** Always zero
- **CC4:** Always zero

**RESULTS**

**EXAMPLE**

| Memory Location: | 00106 |
| Hex Instruction: | 6F 4C (R=6, Shift Count=1210) |
| Assembly Language Coding: | SLA 6,12 |

**Before Execution**

| PSWR | GPR6 |
| 10000106 | 000013AD |

**After Execution**

| PSWR | GPR6 |
| 00000108 | 013AD000 |

**Note**

The contents of GPR6 are left shifted 12 bit positions and then zero-filled from the right. The result is transferred to GPR6.

**EXAMPLE 2**

| Memory Location: | 00106 |
| Hex Instruction: | 6F 4C (R=6, Shift Count=1210) |
| Assembly Language Coding: | SLA 6,12 |

**Before Execution**

| PSWR | GPR6 |
| 10000106 | 001FA58 |

**After Execution**

| PSWR | GPR6 |
| 40000108 | 7A58000 |

**Note**

Overflow occurs and is indicated by CC1.
SHIFT LEFT LOGICAL

The word in the GPR specified by R is shifted left the number of bit positions specified by the shift field (bits 11-15) in the Instruction Word.

CONDITION CODE

CC1: No change
CC2: No change
CC3: No change
CC4: No change

RESULTS

EXAMPLE

Memory Location: 00812
Hex Instruction: 73D4 (R=7, Shift Count=20_10)
Assembly Language Coding: SLL 7,20

Before Execution
PSWR GPR7
A0000812 12345678

After Execution
PSWR GPR7
A0000814 67800000

Note The contents of GPR7 are left-shifted 20 bits and replaced.
SLC

SHIFT LEFT CIRCULAR

7440

The word in the GPR specified by R is shifted left the number of bit positions specified by the shift field (bits 11-15) in the Instruction Word. Bits shifted out of bit position 0 are shifted into bit position 31.

CONDITION CODE
CC1: No change
CC2: No change
CC3: No change
CC4: No change

RESULTS

EXAMPLE
Memory Location: 001FA
Hex Instruction: 77 CF (R=7, Shift Field=1610)
Assembly Language Coding: SLC 7,16

Before Execution
PSWR 000001FA
GPR7 12345678

After Execution
PSWR 000001FC
GPR7 56781234

Note The contents of GPR7 are shifted left circular for 16 bit positions.
SHIFT LEFT ARITHMETIC DOUBLE

The doubleword in the GPR specified by R and R+1 is shifted left the number of bit positions specified by the shift field (bits 11-15) in the Instruction Word. R+1 is the GPR one greater than specified by R. The sign (bit 0) of the GPR specified by R remains unchanged. Condition Code bit 1 is set to One if any bit shifted out of position 1 differs from the sign bit, position 0.

CONDITION CODE
CC1: ISI arithmetic exception
CC2: Always zero
CC3: Always zero
CC4: Always zero

EXAMPLE
Memory Location: 02DF6
Hex Instruction: 7A 58 (R=4, Shift Field=2410)
Assembly Language Coding: SLAD 4,24

Before Execution
PSWR 80002DF6 GPR4 FFFFFFFA3 GPR5 9A178802

After Execution
PSWR 80002DF8 GPR4 A39A1788 GPR5 02000000

Note The doubleword obtained from the contents of GPR4 and GPR5 is left-shifted 24 bit positions, then zero-filled from the right. The result is returned to GPR4 and GPR5.
SHIFT LEFT LOGICAL DOUBLE

7C40

DEFINITION
The doubleword in the GPR specified by R and R+1 is shifted left the number of bit positions specified by the shift field (bits 11-15) in the Instruction Word. R+1 is the GPR one greater than specified by R.

CONDITION CODE

RESULTS

EXAMPLE

Memory Location: 001FE
Hex Instruction: 7F 58 (R=6, Shift Field=24)
Assembly Language Coding: SLLD 6,24

Before Execution
PSWR GPR6 GPR7
100001FE 01234567 89ABCDEF

After Execution
PSWR GPR6 GPR7
10000200 6789ABCD EF000000

Note: The doubleword obtained from GPR6 and GPR7 is left-shifted 24 bit positions, then zero-filled from the right. The result is returned to GPR6 and GPR7.
SHIFT RIGHT ARITHMETIC

6C00

**DEFINITION**
The word in the GPR specified by R is shifted right the number of bit positions specified by the shift field (bits 11-15) in the Instruction Word. Bit position 0 (sign bit) is shifted into bit position 1 on each shift. The sign bit remains unchanged.

**CONDITION CODE**

<table>
<thead>
<tr>
<th>Condition Code</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC1</td>
<td>No change</td>
</tr>
<tr>
<td>CC2</td>
<td>No change</td>
</tr>
<tr>
<td>CC3</td>
<td>No change</td>
</tr>
<tr>
<td>CC4</td>
<td>No change</td>
</tr>
</tbody>
</table>

**EXAMPLE**

Memory Location: 00372
Hex Instruction: 6D OA (R=4, Shift Field=1010)
Assembly Language Coding: SRA 4,10

**Before Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000372</td>
<td>B69825F1</td>
</tr>
</tbody>
</table>

**After Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000374</td>
<td>FFEDA609</td>
</tr>
</tbody>
</table>

**Note**
The contents of GPR4 are shifted right 10 bit positions. Since that value is negative, a one is entered into bit position 1 with each shift.
SHIFT RIGHT LOGICAL
7000

DEFINITION
The word in the GPR specified by R is shifted right the number of bit positions specified by the shift field (bits 11-15) in the Instruction Word.

CONDITION CODE
CC1: No change
CC2: No change
CC3: No change
CC4: No change

EXAMPLE
Memory Location: 00372
Hex Instruction: 72 OA (R=4, Shift Field=1010)
Assembly Language Coding: SRL 4,10

Before Execution
PSWR 10000372
GPR4 B69825F1

After Execution
PSWR 10000374
GPR4 002DA609

Note
The content of GPR4 is shifted right 10 bit positions, then zero-filled from the left.
SHIFT RIGHT CIRCULAR

7400

DEFINITION
The word in the GPR specified by R is shifted right the number of bit positions specified by the shift field (bits 11-15) in the Instruction Word. Bits shifted out of bit position 31 are shifted into bit position 0.

CONDITION CODE

CC1: No change
CC2: No change
CC3: No change
CC4: No change

EXAMPLE
Memory Location: 00372
Hex Instruction: 76 0C (R=4, Shift Field=1210)
Assembly Language Coding: SRC 4,12

Before Execution
PSWR 20000372
GPR4 01234567

After Execution
PSWR 20000374
GPR4 56701234

Note
The contents of GPR4 are shifted right circular 12 bit positions and replaced in GPR4.
SHIFT RIGHT ARITHMETIC DOUBLE

7800

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>R</th>
<th>0</th>
<th>0</th>
<th>SHIFTFIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

DEFINITION

The doubleword in the GPR specified by R and R+1 is shifted right the number of bit positions specified by the shift field (bits 11-15) in the Instruction Word. R+1 is the GPR one greater than specified by R. The sign (bit 0) of the GPR specified by R remains unchanged. Bit position 0 (sign bit) is shifted into bit position 1 with each shift.

CONDITION CODE

<table>
<thead>
<tr>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Location:</td>
</tr>
<tr>
<td>Hex Instruction:</td>
</tr>
<tr>
<td>Assembly Language Coding:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Before Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR</td>
</tr>
<tr>
<td>20002B46</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR</td>
</tr>
<tr>
<td>20002B48</td>
</tr>
</tbody>
</table>

Note: The doubleword obtained from the contents of GPR6 and GPR7 is shifted right 24 bit positions, with the sign extended 24 bits from the left. The result is transferred to GPR6 and GPR7.
DEFINITION
The doubleword in the GPR specified by R and R+1 is shifted right the number of bit positions specified by the shift field (bits 11-15) in the Instruction Word. R+1 is the GPR one greater than specified by R.

CONDITION CODE
CC1: No change
CC2: No change
CC3: No change
CC4: No change

RESULTS

EXAMPLE
Memory Location: 02B46
Hex Instruction: 7F 18 (R=6, Shift Field=2410)
Assembly Language Coding: SRLD 6,24

Before Execution
PSWR  GPR6  GPR7
20002B46  8E2A379B  58C19640

After Execution
PSWR  GPR6  GPR7
20002B48  0000008E  2A379B58

Note
The doubleword obtained from the contents of GPR6 and GPR7 is shifted right 24 bit positions, then zero-filled from the left. The result is transferred to GPR6 and GPR7.
The Bit Manipulation instruction group provides the capability to set, read, or add a bit to a specified bit location within a specified byte of a memory location or General Purpose Register. Provisions have also been made to test a bit in memory or a General Purpose Register by transferring the contents of that bit position to the Condition Code register.

The Bit Manipulation instruction group uses the following two instruction formats:

**MEMORY REFERENCE**

<table>
<thead>
<tr>
<th>Bit Field</th>
<th>X</th>
<th>I</th>
<th>F</th>
<th>WA</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPCODE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 0-5 define the Operation Code.

Bits 6-8 specify a bit (0-7).

Bits 9-10 designate one of three index registers.

Bit 11 indicates whether an indirect addressing operation is to be performed.

Bits 12-31 specify the address of the operand when the X and I fields are equal to zero.

**INTERREGISTER**

<table>
<thead>
<tr>
<th>Bit Field</th>
<th>R</th>
<th>O</th>
<th>O</th>
<th>BYTE FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPCODE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 0-5 define the Operation Code.

Bits 6-8 specify a bit (0-7).

Bits 9-11 designate a General Purpose Register address (0-7).

Bits 12-13 unassigned.

Bits 14-15 specify a byte (0-3).
A Condition Code is set during execution of Set Bit, Zero Bit, and Test Bit operations, if the bit on which the operation is being performed is equal to one. During Add Bit operations, a Condition Code is set to indicate whether the execution of the instruction caused a result greater than zero, less than zero, equal to zero, or an arithmetic exception.

When two processors share memory and other resources, a simple positive method must be provided for dynamically reserving/releasing shared memory pages and the other shared resources. The Set Bit in Memory, Zero Bit in Memory, or Add Bit in Memory instructions (SBM, ZBM) are used for this purpose. If both processors attempt to set (or zero) the same semaphore bit at the same time, one processor will actually access the memory location before the other processor by virtue of the shared memory bus design. The first processor to access the bit will copy the previous contents of the bit into its Condition Code register before setting (or clearing) the bit. On the very next memory cycle, the other processor will copy the state of the bit as set by the first processor into its Condition Code register and then set (or clear) the bit again. Both processors then execute Branch on Condition Code instructions to test the status of the bit prior to changing it. The first processor will find the bit previously not set (or set), indicating that it was able to reserve the resource which the user has associated with the bit. The second processor will find the bit already set (or not set), indicating that the resource is currently reserved by the other processor and that subsequent attempts should be made.
SET BIT IN MEMORY

9808

**DEFINITION**
The byte in memory specified by the Effective Byte Address (EBA) is accessed, and the specified bit (bit field) within the byte set to one. All other bits within the byte remain unchanged. The resulting byte is replaced in the location specified by the EBA. Condition Code bit 3 (CC3) is transferred to CC4, CC2 is transferred to CC3, CC1 is transferred to CC2, and the original status of the specified bit of the byte specified by the EBA is transferred to CC1.

**NOTE**
Since the contents of the Condition Code register are shifted to the next highest position before the specified bit is loaded into CC1, any 4 bits in memory or the GPR's can be stored in the Condition Code register for a combined Conditional Branch test.

**SUMMARY**

<table>
<thead>
<tr>
<th>Condition Code</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>(CC3) → CC4</td>
<td>(CC2) → CC3</td>
</tr>
<tr>
<td>(CC1) → CC2</td>
<td>(EBL SBL) → CC1</td>
</tr>
<tr>
<td>EBL SBL</td>
<td>1 → EBL SBL</td>
</tr>
</tbody>
</table>

**CONDITION CODE RESULTS**
- CC1: ISI EBL SBL is equal to one
- CC2: ISI CC1 was one
- CC3: ISI CC2 was one
- CC4: ISI CC3 was one

**EXAMPLE**
- Memory Location: 01000
- Hex Instruction: 98 88 14 03 (bit field = 1)
- Assembly Language Coding: SBM 1,X'1403'

**Before Execution**
- PSWR 20001000
- Memory Byte 01403

**After Execution**
- PSWR 10001004
- Memory Byte 01403

**Note**
Bit 1 of memory byte 01403 is set to one.
**SET BIT IN REGISTER**

1800

The specified bit (bit field) of the specified byte (byte field) in the GPR specified by R is set to one. All other bits within the GPR specified by R remain unchanged. Condition Code bit 3 (CC3) is transferred to CC4, CC2 is transferred to CC3, CC1 is transferred to CC2, and the original status of the specified bit in register R is transferred to CC1.

Since the contents of the Condition Code register are shifted to the next highest position before the specified bit is loaded into CC1, any four bits in memory or the GPR's can be stored in the Condition Code register for a combined Conditional Branch test.

### Definition

The specified bit (bit field) of the specified byte (byte field) in the GPR specified by R is set to one. All other bits within the GPR specified by R remain unchanged. Condition Code bit 3 (CC3) is transferred to CC4, CC2 is transferred to CC3, CC1 is transferred to CC2, and the original status of the specified bit in register R is transferred to CC1.

### Note

Since the contents of the Condition Code register are shifted to the next highest position before the specified bit is loaded into CC1, any four bits in memory or the GPR's can be stored in the Condition Code register for a combined Conditional Branch test.

### Summary

- \( \text{CC3} \rightarrow \text{CC4} \)
- \( \text{CC2} \rightarrow \text{CC3} \)
- \( \text{CC1} \rightarrow \text{CC2} \)
- \( \text{R}_{\text{SBL}} \rightarrow \text{CC1} \)
- \( 1 \rightarrow \text{EBL}_{\text{SBL}} \)

### Condition Code

- CC1: ISI \( R_{\text{SBL}} \) is equal to one
- CC2: ISI CC1 was one
- CC3: ISI CC2 was one
- CC4: ISI CC3 was one

### Example

Memory Location: 01002
Hex Instruction: XXXX1B 82 (bit field=7, R=0, byte field=2)
Assembly Language Coding: SBR 0,2

**Before Execution**

- PSWR: 10001002
- GPRO: 0374B891

**After Execution**

- PSWR: 08001004
- GPRO: 0374B991

**Note**

Bit 23 of GPRO is set to one.
ZERO BIT IN MEMORY

9C08

<table>
<thead>
<tr>
<th>1 0 0 1 1 1</th>
<th>1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT FIELD</td>
<td>BYTE OPERAND ADDRESS</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**DEFINITION**
The byte in memory specified by the Effective Byte Address (EBA) is accessed and the specified bit (bit field) within the byte is set to zero. All other bits within the byte remain unchanged. The resulting byte is replaced in the location specified by the EBA. Condition Code bit 3 (CC3) is transferred to CC4, CC2 is transferred to CC3, CC1 is transferred to CC2 and the original status of the specified bit of the byte specified by the EBA is transferred to CC1.

**NOTE**
Since the contents of the Condition Code register are shifted to the next highest position before the specified bit is loaded into CC1, any four bits in memory or the GPR's can be stored in the Condition Code register for a combined Conditional Branch test.

**SUMMARY**
(CC3) → CC4
(CC2) → CC3
(CC1) → CC2
(EBLSBL) → CC1
0 → EBLsBL

**CONDITION CODE RESULTS**
CC1: ISI EBLsBL is equal to one
CC2: ISI CC1 was one
CC3: ISI CC2 was one
CC4: ISI CC3 was one

**EXAMPLE**
Memory Location:
Hex Instruction:
Assembly Language Coding:
Before Execution
After Execution

<table>
<thead>
<tr>
<th>Before Execution</th>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR 1001F684</td>
<td>Memory Byte 20122</td>
</tr>
<tr>
<td>34</td>
<td>4801F688</td>
</tr>
<tr>
<td>Memory Byte 20122</td>
<td>30</td>
</tr>
</tbody>
</table>

| Memory Byte 20122 | Memory Byte 20122 |
| 1F684             | 9E 8A 01 22 (bit field=5) |
| ZMB 5,X'20122'    | ZMB 5,X'20122' |

6-130
ZERO BIT IN REGISTER

**DEFINITION**
The specified bit (bit field) of the specified byte (byte field) in the GPR specified by R is set to zero. All other bits within the GPR specified by R remain unchanged. Condition Code bit 3 (CC3) is transferred to CC2, and the original status of the specified bit of the specified byte in register R is transferred to CCI.

**NOTE**
Since the contents of the Condition Code register are shifted to the next highest position before the bit is loaded into CCI, any four bits in memory or the GPR's can be stored in the Condition Code register for a combined Conditional Branch test.

**SUMMARY**
- (CC3) → CC4
- (CC2) → CC3
- (CC1) → CC2
- (RSBL) → CC1
- 0 → EBLEBL

**CONDITION CODE**
- CC1: ISI RSBL is equal to one
- CC2: ISI CCI was one
- CC3: ISI CCI2 was one
- CC4: ISI CC3 was one

**EXAMPLE**
Memory Location: 00C56
Hex Instruction: 1C51 (bit field=0, R=5, byte field=1)
Assembly Language Coding: ZBR 5,8

Before Execution:
- PSWR: 10000C56
- GPR5: 76A43B19

After Execution:
- PSWR: 48000C58
- GPR5: 76243B19

**Note**
Bit 8 of GPR5 is cleared to zero. CC4 is set.
**ADD BIT IN MEMORY**

**A008**

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>X</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

**DEFINITION**
The byte in memory specified by the Effective Byte Address (EBA) is accessed and one is added to the bit position specified by the bit field. The addition is performed on the entire memory word containing the byte specified by the EBA. Therefore, a carry may be propagated left to the sign bit. The resulting word is transferred to the memory word location containing the byte specified by the EBA.

**SUMMARY EXPRESSION**

(EBL)+1SBL → EBL

**CONDITION CODE**

- **CC1:** ISI arithmetic exception
- **CC2:** ISI (EWL) is greater than zero
- **CC3:** ISI (EWL) is less than zero
- **CC4:** ISI (EWL) is equal to zero

**RESULTS**

**EXAMPLE**

- **Memory Location:** 03000
- **Hex Instruction:** A2 08 31 92 (bit field=4, X=0, I=0)
- **Assembly Language Coding:** ABM 4,X'3192'

**Before Execution**

- **PSWR:** Memory Word 03190
- **00003000:** 51A3F926

**After Execution**

- **PSWR:** Memory Word 03190
- **20003004:** 51A40126

**Note**

A one is added to bit position 20\text{\textsubscript{10}} of memory word 03190 (byte 2, bit 4) which propagates a carry left to bit position 13\text{\textsubscript{10}}. The result is returned to memory word 03190. CC2 is set.
ADD BIT IN REGISTER

2000

<table>
<thead>
<tr>
<th>0 1 0 0 0</th>
<th>BIT FIELD</th>
<th>R</th>
<th>0</th>
<th>BYTE FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DEFINITION**

A one is added to the specified bit (bit field) of the specified byte (byte field) in the GPR specified by R. The addition is performed on the entire word of the GPR specified by R. Therefore, a carry may be propagated left to the sign bit. The result is then transferred to the GPR specified by R.

**SUMMARY**

\[(R)+1_{SBL} \rightarrow R\]

**CONDITION CODE RESULTS**

CC1: ISI arithmetic exception  
CC2: ISI \(R_0-31\) is greater than zero  
CC3: ISI \(R_0-31\) is less than zero  
CC4: ISI \(R_0-31\) is equal to zero

**EXAMPLE**

Memory Location: \(0184E\)  
Hex Instruction: \(21 61\) (bit field=2, R=6, byte field=1)  
Assembly Language Coding: ABR 6,10

**Before Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0800184E</td>
<td>3BE9AC48</td>
</tr>
</tbody>
</table>

**After Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR6</th>
</tr>
</thead>
<tbody>
<tr>
<td>20001850</td>
<td>3C09AC48</td>
</tr>
</tbody>
</table>

**Note**

A one is added to bit position 10 of GPR6, and the result is replaced in GPR6. CC2 is set.
TEST BIT IN MEMORY

A408

<table>
<thead>
<tr>
<th>BIT FIELD</th>
<th>X</th>
<th>I</th>
<th>BYTE OPERAND ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0 0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### DEFINITION

The specified bit in memory is transferred to the Condition Code register. Condition Code bit 3 (CC3) is transferred to CC4, CC2 is transferred to CC3, CC1 is transferred to CC2, and the specified bit (bit field) of the byte specified by the Effective Byte Address (EBA) is transferred to CC1.

### NOTE

Since the contents of the Condition Code register are shifted to the next highest position before the specified bit is loaded into CC1, any four bits in memory or the GPR's can be stored in the Condition Code register for a combined Conditional Branch test.

### SUMMARY

- (CC3) → CC4
- (CC2) → CC3
- (CC1) → CC2
- (EBL-SBL) → CC1

### CONDITION CODE

- CC1: ISI RSBL is equal to one
- CC2: ISI CC1 was equal to one
- CC3: ISI CC2 was equal to one
- CC4: ISI CC3 was equal to one

### EXAMPLE

Memory Location: 05A38  
Hex Instruction: A6 08 5B 21 (bit field=4, X=0, I=0)  
Assembly Language Coding: TBM 4,X'5B21'

**Before Execution**  
PSWR 10005A38  
Memory Byte 05B21 29

**After Execution**  
PSWR 48005A3C  
Memory Byte 05B21 29

**Note**  
Bit 4 of memory byte 05B21 is transferred to CC1. CC3 is transferred to CC4.
TEST BIT IN REGISTER

2400

<table>
<thead>
<tr>
<th>BIT FIELD</th>
<th>R</th>
<th>BYTE FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 0 0 1</td>
<td>0 0</td>
<td></td>
</tr>
</tbody>
</table>

DEFINITION
The specified bit in the GPR specified by R is transferred to the Condition Code register. Condition Code bit 3 (CC3) is transferred to CC4, CC2 is transferred to CC3, CC1 is transferred to CC2, and the specified bit (bit field) of the specified byte (byte field) in the GPR specified by R is transferred to CC1.

NOTE
Since the contents of the Condition Code register are shifted to the next highest position before the specified bit is loaded into CC1, any four bits in memory or the GPR's can be stored in the Condition Code register for a combined Conditional Branch test.

SUMMARY
(CC3) → CC4
(CC2) → CC3
(CC1) → CC2
(RSBL) → CC1

CONDITION CODE RESULTS
CC1: ISI RSBL was equal to one
CC2: ISI CC1 was equal to one
CC3: ISI CC2 was equal to one
CC4: ISI CC3 was equal to one

EXAMPLE
Memory Location: 01982
Hex Instruction: 25 D3 (bit field=3, R=5, byte field=3)
Assembly Language Coding: TBR 5,27

Before Execution
PSWR 18001982 GPR5 81A2C64D

After Execution
PSWR 08001984 GPR5 81A2C64D

Note
CC2 through CC4 are right-shifted one bit position. CC1 is cleared to zero since bit 2710 of GPR5 is zero.
The Fixed-Point Arithmetic group is used to perform addition, subtraction, multiplication, division, and sign control functions on bytes, halfwords, words, and doublewords in memory and General Purpose Registers. Provisions have also been made to allow the result of a register-to-register addition or subtraction to be masked before final storage.

The Fixed-Point Arithmetic instructions use the following three instruction formats:

### MEMORY REFERENCE

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>F</th>
<th>WA</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 0-5 define the Operation Code.

Bits 6-8 designate a General Purpose Register address (0-7).

Bits 9-10 designate one of three index registers.

Bit 11 designates whether an Indirect Addressing operation is to be performed.

Bits 12-31 specify the address of the operand when the X and I fields are equal to zero.

### IMMEDIATE

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>AUG CODE</th>
<th>OPERAND VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits 0-5 define the Operation Code.

Bits 6-8 designate a General Purpose Register address (0-7).

Bits 9-12 unassigned.


Bits 16-31 contain the 16-bit operand value.
The Fixed-Point Arithmetic instructions use the following data formats:

**Byte**

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

**Halfword (Sign Extended)**

```
S S S S S S S S S S S S S S S S S S S S S S
```

**Word**

```
S
```

**Doubleword**

```
S
```

Bits 0-5 define the Operation Code.

Bits 6-8 designate the register to contain the result of the operation.

Bits 9-11 designate the register which contains the source operand.

Bits 12-15 define the Augmenting Operation Code.
Execution of most Fixed-Point Arithmetic instructions causes a Condition Code to be set to indicate whether the result of the operation was greater than, less than, or equal to zero. Arithmetic exceptions produced by an arithmetic operation are also reflected by the Condition Code results.

To perform logical operations, the hardware interprets operands as logical words. For fixed-point arithmetic operations, operands are treated as unsigned numbers. Logical and arithmetic operations can be performed on any of the data types available in the SEL 32 Series Computer bytes, 16-bit halfwords, 32-bit words, and 64-bit doublewords. A program executing on the SEL 32 Series Computer however, can interpret any of the available data types as a two's complement notation number. It is a property of two's complement arithmetic that operations on signed numbers using two's complement conversions are identical to operations on unsigned numbers; in other words, the hardware treats the sign as the most significant magnitude bit.

Consider a General Purpose Register that contains:

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

As an unsigned number, this would be equivalent to:

\[
82_{16} = 130_{10}
\]

Interpreted as a signed number using two's complement notation, it would be:

\[
7E_{16} = 126_{10}
\]

It makes no difference as to how the programmer interprets data as far as processor operation is concerned. However, the programmer is aided in the use of two's complement notation by the Condition Code (CC) bits of the Program Status Word (PSW), which are generally set based on two's complement notation.
Numbers in two's complement notation are symmetrical in magnitude around a zero representation, so all even numbers, both positive and negative, will end in zero, and all odd numbers will end in one (binary word containing all one's represents minus one).

If one's complement notation was used for negative numbers, a negative number could be read by attaching significance to the zeros instead of the one's.

In two's complement notation, each number is one greater than the complement of the positive number of the same magnitude, so a negative number can be read by attaching significance to the right-hand one and to the zeros to the left of it. (The negative number of the largest magnitude has a one only in the sign position.) Assuming a binary integer, one's may be discarded at the left in a negative integer in the same way that leading zeros may be dropped from a positive integer.

Associated with the Arithmetic/Logic Unit is a 4-bit Condition Code register which forms the CC portion of the PSW. These CC bits are altered during all Arithmetic/Logical operations and data transfers. The CC bits indicate such conditions as arithmetic exception, overflow, zero, and positive or negative magnitude.
ADD MEMORY BYTE

B808

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DEFINITION

The byte in memory specified by the Effective Byte Address (EBA) is accessed and 24 zeros are appended to the most significant end to form a word. This word is algebraically added to the contents of the GPR specified by R. The resulting word is then transferred to the GPR specified by R.

SUMMARY

\( (EBA) + (R) \rightarrow R \)

CONDITION CODE

- CC1: ISI arithmetic exception
- CC2: ISI \( R_0-31 \) is greater than zero
- CC3: ISI \( R_0-31 \) is less than zero
- CC4: ISI \( R_0-31 \) is equal to zero

RESULTS

0-23, \((EBA) + (R) \rightarrow R\)

EXAMPLE

Memory Location: 00800
Hex Instruction: BA 08 09 15 \((R=4, X=0, I=0)\)
Assembly Language Coding: ADMB 4,X'915'

Before Execution
- PSWR: 10000800
- GPR4: 00000099
- Memory Byte: 00915

After Execution
- PSWR: 20000804
- GPR4: 00000123
- Memory Byte: 00915

Note

The contents of memory byte 00915, with zeros prefixed, are added to the contents of GPR4, and the result is transferred to GPR4. CC2 is set.
ADD MEMORY HALFWORD

B800

### DEFINITION
The halfword in memory specified by the Effective Halfword Address (EHA) is accessed and the sign bit (bit 16) is extended 16 bits to the left to form a word. This word is algebraically added to the contents of the GPR specified by R. The resulting word is then transferred to the GPR specified by R.

### SUMMARY
\[(EHL)_{SE}^+(R) \rightarrow R\]

### CONDITION CODE
- **CC1**: ISI arithmetic exception
- **CC2**: ISI \(R_{0-31}\) is greater than zero
- **CC3**: ISI \(R_{0-31}\) is less than zero
- **CC4**: ISI \(R_{0-31}\) is equal to zero

### RESULTS
- **Memory Location**: 40D68
- **Hex Instruction**: BB 84 10 97 (R=7, X=0, I=0)
- **Assembly Language Coding**: ADMH 7,X'41096'

### EXAMPLE

**Before Execution**
- PSWR: 20040D68
- GPR7: 000006C4
- Memory Halfword 41096

**After Execution**
- PSWR: 10040D6C
- GPR7: FFFF9306
- Memory Halfword 41096

**Note**
The contents of memory halfword 41096 with sign extension are added to the contents of GPR7, and the result replaces the contents of GPR7. CC3 is set.
ADMW d,m,x

ADD MEMORY WORD

B800

<table>
<thead>
<tr>
<th>101110</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>0</th>
<th>WORD OPERAND ADDRESS</th>
<th>00</th>
</tr>
</thead>
</table>

**DEFINITION**
The word in memory specified by the Effective Word Address (EWA) is accessed and algebraically added to the contents of the GPR specified by R. The resulting word is then transferred to the GPR specified by R.

**SUMMARY**

\[(EWL)+(R) \rightarrow R\]

**CONDITION CODE RESULTS**

CC1: ISI arithmetic exception
CC2: ISI R0-31 is greater than zero
CC3: ISI R0-31 is less than zero
CC4: ISI R0-31 is equal to zero

**EXAMPLE**

Memory Location: 00D50
Hex Instruction: BB 00 11 AC (R=6, X=0, I=0)
Assembly Language Coding: ADMW 6,X'11AC'

Before Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR6</th>
</tr>
</thead>
<tbody>
<tr>
<td>400000D50</td>
<td>0037C1F3</td>
</tr>
</tbody>
</table>

After Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR6</th>
</tr>
</thead>
<tbody>
<tr>
<td>200000D54</td>
<td>00878469</td>
</tr>
</tbody>
</table>

Memory Word 011AC
004FC276

**Note**
The contents of memory word 011AC are added to the contents of GPR6. The result is transferred to GPR6. CC2 is set.
ADD MEMORY DOUBLEWORD

B800

<table>
<thead>
<tr>
<th>1 0 1 1 1 0</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DEFINITION
The doubleword in memory specified by the Effective Doubleword Address (EDA) is accessed and algebraically added to the contents of the GPR specified by R and R+1. R+1 is the GPR one greater than specified by R. The contents of the GPR specified by R+1 are added to the contents of the least significant word of the doubleword first. The contents of the GPR specified by R are added to the contents of the most significant word of the doubleword last. The resulting doubleword is transferred to the GPR specified by R and R+1.

SUMMARY
(EXWL + 1) + (R+1) \rightarrow R+1 + Carry

CONDITION CODE
CC1: ISI arithmetic exception
CC2: ISI (R, R+1) is greater than zero
CC3: ISI (R, R+1) is less than zero
CC4: ISI (R, R+1) is equal to zero

RESULTS

EXAMPLE
Memory Location: 08E3C
Hex Instruction: BA 00 92 52 (R=4, X=0, I=0)
Assembly Language Coding: ADMD 4,X'9250'

Before Execution
PSWR: 08008E3C
GPR4: 000298A1
GPR5: 815BC63E
Memory Word 09250: 3B69A07E
Memory Word 09254: 7F3579A4

After Execution
PSWR: 20008E40
GPR4: 3B6C3920
GPR5: 00913FE2
Memory Word 09250: 3B69A07E
Memory Word 09254: 7F3579A4

Note
The doubleword obtained from the contents of memory words 09250 and 09254 is added to the doubleword obtained from the contents of GPR4 and GPR5. The result is transferred to GPR4 and GPR5. CC2 is set.
ADD REGISTER TO REGISTER

3800

DEFINITION
The word in the GPR specified by \( R_D \) is algebraically added to the word in the GPR specified by \( R_S \). The resulting word is then transferred to the GPR specified by \( R_D \).

SUMMARY
\((R_S + R_D) \rightarrow R_D\)

CONDITION CODE

CC1: ISI arithmetic exception
CC2: ISI \((R_D)\) is greater than zero
CC3: ISI \((R_D)\) is less than zero
CC4: ISI \((R_D)\) is equal to zero

RESULTS

EXAMPLE
Memory Location: 03FA2
Hex Instruction: 3B 70 \((\text{RS}=6, \text{RS}=7)\)
Assembly Language Coding: ADR 7,6

Before Execution
PSWR GPR6 GPR7
08003FA2 FF03C67D 045C6E3F

After Execution
PSWR GPR6 GPR7
20003FA4 036034BC 045C6E3F

Note
The contents of GPR6 and GPR7 are added and the result is transferred to GPR6. CC2 is set.
ADD REGISTER TO REGISTER MASKED

3808

<table>
<thead>
<tr>
<th>0 0 1 1 1 0</th>
<th>R_D</th>
<th>R_S</th>
<th>1 0 0</th>
</tr>
</thead>
</table>

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

**DEFINITION**
The word in the GPR specified by R_D is algebraically added to the word in the GPR specified by R_S. The sum of this addition is masked (Logical AND Function) with the contents of the Mask register (R4). The resulting word is then transferred to the GPR specified by R_D.

**SUMMARY**

\[(R_S) + (R_D) \& (R4) \rightarrow R_D\]

**CONDITION CODE**

- **CC1:** ISI arithmetic exception
- **CC2:** ISI \((R_D)\) is greater than zero
- **CC3:** ISI \((R_D)\) is less than zero
- **CC4:** ISI \((R_D)\) is equal to zero

**RESULTS**

**EXAMPLE**

Memory Location: 16A9A

Hex Instruction: 3B 78 \((R_D=6, R_S=7)\)

Assembly Language Coding: ADRM 7,6

**Before Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>40016A9A</td>
<td>007FFFFC</td>
<td>004FC276</td>
<td>0037C1F3</td>
</tr>
</tbody>
</table>

**After Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR4</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>20016A9C</td>
<td>007FFFFC</td>
<td>00078468</td>
<td>0037C1F3</td>
</tr>
</tbody>
</table>

**Note**
The contents of GPR6 and GPR7 are added; the result is ANDed with the contents of GPR4 and transferred to GPR6. CC2 is set.

6-145
ADD REGISTER TO MEMORY BYTE
E808

The byte in memory specified by the Effective Byte Address (EBA) is accessed and algebraically added to the contents of the GPR specified by R. Bits 24-31 of the result are then transferred to the memory byte location specified by the EBA. The GPR and the other three bytes in the word which contains the byte specified by the EBA remain unchanged.

(R)+(EBL) → EBL

CC1: Undefined
CC2: Undefined
CC3: Undefined
CC4: ISI the 32-bit sum is equal to zero

Memory Location: 01A64
Hex Instruction: EB 08 1A 97 (R=6, X=0, I=0)
Assembly Language Coding: ARMB 6,X'1A97'

Before Execution
PSWR 00001A64
GPR6 0000004A
Memory Byte 01A97

After Execution
PSWR 00001A68
GPR6 0000004A
Memory Byte 01A97

Note The contents of GPR6 and memory byte 01A97 are added and the result is transferred to memory byte 01A97.
ADD REGISTER TO MEMORY HALFWORD

E800

**DEFINITION**
The halfword in memory specified by the Effective Halfword Address (EHA) is accessed and algebraically added to the least significant halfword (bits 16-31) of the GPR specified by R. The result is then transferred to the memory halfword location specified by the EHA. The other halfword of the word which contains the halfword specified by the EHA remains unchanged.

**SUMMARY**
\[(R_{16-31}) + (EHA) \rightarrow EHL\]

**CONDITION CODE RESULTS**

- CC1: Undefined
- CC2: Undefined
- CC3: Undefined
- CC4: ISI (EHL) is equal to zero

**EXAMPLE**

- **Memory Location:** 20084
- **Hex Instruction:** EA 82 09 19 (R=5, X=0, I=0)
- **Assembly Language Coding:** ARMH 5,"X'20918"

**Before Execution**

- PSWR: 000200B4
- GPR5: FFFFFFC42
- Memory Halfword: 20918

**After Execution**

- PSWR: 000200B8
- GPR5: FFFFFFC42
- Memory Halfword: 20918

**Note**
The contents of bits 16-31 of GPR5 and memory halfword 20918 are added and the result is transferred to memory halfword 20918.
ADD REGISTER TO MEMORY WORD

`ARMW s,*m,x`

`E800`

### Definition
The word in memory specified by the Effective Word Address (EWA) is accessed and algebraically added to the word in the GPR specified by R. The resulting word is then transferred to the memory word location specified by the EWA.

### Summary

\[(E) + (EWL) \rightarrow EWL\]

### Condition Code Results

- **CC1**: ISI arithmetic exception
- **CC2**: ISI (EWL) is greater than zero
- **CC3**: ISI (EWL) is less than zero
- **CC4**: ISI (EWL) is equal to zero

### Example

**Memory Location:** 03000  
**Hex Instruction:** EB 80 31 00 (R=7, X=0, I=0)  
**Assembly Language Coding:** `ARMW 7,X'3100'`

<table>
<thead>
<tr>
<th>Before Execution</th>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR 08003000</td>
<td>PSWR 20003004</td>
</tr>
<tr>
<td>GPR7 245C6E3F</td>
<td>GPR7 245C6E3F</td>
</tr>
<tr>
<td>Memory Word 03100</td>
<td>Memory Word 03100</td>
</tr>
</tbody>
</table>

**Note:** The contents of GPR7 and memory word 03100 are added and the result is transferred to memory word 03100. CC2 is set.
ADD REGISTER TO MEMORY DOUBLEWORD

E800

<table>
<thead>
<tr>
<th>DOUBLEWORD OPERAND ADDRESS</th>
<th>0 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R X I</td>
<td>1 1 0</td>
</tr>
</tbody>
</table>

DEFINITION
The doubleword in memory specified by the Effective Doubleword Address (EDA) is accessed and algebraically added to the doubleword in the GPR specified by R and R+1. R+1 is the GPR one greater than specified by R. The contents of the GPR specified by R+1 are added to the contents of the least significant word of the doubleword first. The resulting doubleword is transferred to the memory doubleword location specified by the EDA.

SUMMARY
(R+1)+(EQL+1) → EWL+1+Carry
(R)+(EWL)+Carry → EWL

CONDITION CODE
CC1: ISI arithmetic exception
CC2: ISI (EDL) is greater than zero
CC3: ISI (EDL) is less than zero
CC4: ISI (EDL) is equal to zero

RESULTS

EXAMPLE
Memory Location: 0819C
Hex Instruction: EB 00 83 AA (R=6, X=0, I=0)
Assembly Language Coding: ARMD 6,X'83A8'

Before Execution
PSWR 0400819C GPR6 01A298A1 GPR7 F15BC63E
Memory Word 083AB 3869A07E

After Execution
PSWR 020081A0 GPR6 01A298A1 GPR7 F15BC63E
Memory Word 083AB 3D0C3920

Note
The doubleword obtained from GPR6 and GPR7 is added to the doubleword from memory words 083AB and 083AC. The result is transferred to memory words 083AB and 083AC. CC2 is set.
DEFINITION The sign of the least significant bit (bit 16) of the Instruction Word is extended 16 bits to the left to form a word. This word is algebraically added to the word in the GPR specified by R. The resulting word is transferred to the GPR specified by R.

SUMMARY 

EXPRESSION 

CONDITION CODE

RESULTS

CC1: ISI arithmetic exception
CC2: ISI R0-31 is greater than zero
CC3: ISI R0-31 is less than zero
CC4: ISI R0-31 is equal to zero

EXAMPLE

Memory Location: 00D88
Hex Instruction: C8 01 86 B2 (R=0)
Assembly Language Coding: ADI 0, '86B2'

Before Execution

After Execution

PSWR 20000088
GPRO 0000794E

PSWR 0800008C
GPRO 00000000

Note The immediate operand, sign extended, is added to the contents of the GPRO and the result replaces the previous contents of GPRO. CC4 is set.
DEFINITION
The byte in memory specified by the Effective Byte Address (EBA) is accessed and 24 zeros are appended to the most significant end to form a word. This word is algebraically subtracted from the word in the GPR specified by R. The resulting word is transferred to the GPR specified by R.

SUMMARY
\[(R) - [0:0-23,(EBL)] \rightarrow R\]

CONDITION CODE
CC1: ISI arithmetic exception
CC2: ISI \(R_{0-31}\) is greater than zero
CC3: ISI \(R_{0-31}\) is less than zero
CC4: ISI \(R_{0-31}\) is equal to zero

EXAMPLE
Memory Location: 01000
Hex Instruction: BC 88 12 01 (R=1, X=0, I=0)
Assembly Language Coding: SUMB 1, X\'1201'

Before Execution
PSWR: 40001000
GPR1: 0194A7F2
Memory Byte 01201

After Execution
PSWR: 20001004
GPR1: 0194A758
Memory Byte 01201

Note
The contents of memory byte 01201, with 24 zeros prefixed, are subtracted from the contents of GPR1. The result is transferred to GPR1. CC2 is set.
SUMH d,*m,x

SUBTRACT MEMORY HALFWORD

BC00

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>R</th>
<th>X</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DEFINITION**
The halfword in memory specified by the Effective Halfword Address is accessed and the sign bit (bit 16) is extended 16 bits to the left to form a word. This word is algebraically subtracted from the word in the GPR specified by R. The resulting word is then transferred to the GPR specified by R.

**SUMMARY**

(R)-(EHL)SE → R

**CONDITION CODE RESULTS**

CC1: ISI arithmetic exception
CC2: ISI R0-31 is greater than zero
CC3: ISI R0-31 is less than zero
CC4: ISI R0-31 is equal to zero

**EXAMPLE**

Memory Location: 01604
Hex Instruction: BF 00 18 77 (R=6, X=0, I=0)
Assembly Language Coding: SUMH 6,'X'1876'

Before Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR6</th>
<th>Memory Halfword 01876</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001604</td>
<td>00024CB3</td>
<td>34C6</td>
</tr>
</tbody>
</table>

After Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR6</th>
<th>Memory Halfword 01876</th>
</tr>
</thead>
<tbody>
<tr>
<td>20001608</td>
<td>000217ED</td>
<td>34C6</td>
</tr>
</tbody>
</table>

**Note**
The contents of memory halfword 01876, sign extended, are subtracted from the contents of GPR6. The result is transferred to GPR6. CC2 is set.
**DEFINITION**

The word in memory specified by the Effective Word Address is accessed and algebraically subtracted from the word in the GPR specified by R. The resulting word is then transferred to the GPR specified by R.

**SUMMARY**

\[(R)-(EWL) \rightarrow R\]

**CONDITION CODE**

<table>
<thead>
<tr>
<th>CC1</th>
<th>ISI arithmetic exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC2</td>
<td>ISI (R_0-31) is greater than zero</td>
</tr>
<tr>
<td>CC3</td>
<td>ISI (R_0-31) is less than zero</td>
</tr>
<tr>
<td>CC4</td>
<td>ISI (R_0-31) is equal to zero</td>
</tr>
</tbody>
</table>

**RESULTS**

- Memory Location: 6C208
- Hex Instruction: BC 86 F9 14 (R=1, X=0, I=0)
- Assembly Language Coding: SUMW 1,X'6F914'

**EXAMPLE**

<table>
<thead>
<tr>
<th>Before Execution</th>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR 0406C208</td>
<td>PSWR 2006C20C</td>
</tr>
<tr>
<td>GPR1 00A6264D</td>
<td>GPR1 009EDA8A</td>
</tr>
<tr>
<td>Memory Word 6F914</td>
<td>Memory Word 6F914</td>
</tr>
</tbody>
</table>

**Note**

The contents of memory word 6F914 are subtracted from the contents of GPR1 and the result is transferred to GPR1. CC2 is set.
SUBTRACT MEMORY DOUBLEWORD

BC00

DEFINITION
The doubleword in memory specified by the Effective Doubleword Address (EDA) is accessed and algebraically subtracted from the doubleword in the GPR specified by R and R+1. R+1 is the GPR one greater than specified by R. The word located in the GPR specified by R+1 is subtracted from the least significant word of the doubleword first. The resulting doubleword is transferred to the GPR specified by R and R+1.

SUMMARY

CONDITION CODE

RESULTS

EXAMPLE

Before Execution

After Execution

Note

The doubleword obtained from memory words 03100 and 03104 is subtracted from the doubleword in GPR6 and GPR7. The result is transferred to GPR6 and GPR7. CC2 is set.
SUBTRACT REGISTER FROM REGISTER

3C00

\[
\begin{array}{ccccccccccccccc}
0 & 0 & 1 & 1 & 1 & R_D & & R_S & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

**DEFINITION**

The word in the GPR specified by \( R_S \) is algebraically subtracted from the word in the GPR specified by \( R_D \). The resulting word is then transferred to the GPR specified by \( R_D \).

\((R_D) - (R_S) \rightarrow R_D\)

**SUMMARY**

**EXPRESSION**

**CONDITION CODE**

CC1: ISI arithmetic exception

CC2: ISI \((R_D)\) is greater than zero

CC3: ISI \((R_D)\) is less than zero

CC4: ISI \((R_D)\) is equal to zero

**RESULTS**

**EXAMPLE**

Memory Location: 106AE
Hex Instruction: 3C A0 \((R_D=1, R_S=2)\)
Assembly Language Coding: SUR 2,1

Before Execution
PSWR: 100106AE
GPR1: 12345678
GPR2: 12345678

After Execution
PSWR: 080106B0
GPR1: 00000000
GPR2: 12345678

**Note**

The contents of GPR2 are subtracted from the contents of GPR1. The result is replaced in GPR1. CC4 is set.
SUBTRACT REGISTER FROM REGISTER MASKED

3C08

**DEFINITION**
The word in the GPR specified by Rs is algebraically subtracted from the word in the GPR specified by Ro. The difference of this subtraction is then masked (Logical AND Function) with the contents of the Mask register (R4). The resulting word is transferred to the GPR specified by RD.

**SUMMARY EXPRESSION**

\[(R_D) - (R_S) & (R4) \rightarrow R_D\]

**CONDITION CODE**
- CC1: ISI arithmetic exception
- CC2: ISI (R_D) is greater than zero
- CC3: ISI (R_D) is less than zero
- CC4: ISI (R_D) is equal to zero

**RESULTS**

**EXAMPLE**

<table>
<thead>
<tr>
<th>Before Execution Memory Location: 00496</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex Instruction: 3F 58 (R_D=6, R_S=5)</td>
</tr>
<tr>
<td>Assembly Language Coding: SURM 5,6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Before Execution PSWR</th>
<th>GPR4</th>
<th>GPR5</th>
<th>GPR6</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000496</td>
<td>00FF00</td>
<td>0074BC3</td>
<td>00A6264D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>After Execution PSWR</th>
<th>GPR4</th>
<th>GPR5</th>
<th>GPR6</th>
</tr>
</thead>
<tbody>
<tr>
<td>20000498</td>
<td>00FF00</td>
<td>0074BC3</td>
<td>009EDA00</td>
</tr>
</tbody>
</table>

**Note**
The contents of GPR5 are subtracted from the contents of GPR6. The result is ANDed with the contents of GPR4 and transferred to GPR6. CC2 is set.
SUBTRACT IMMEDIATE

C802

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>R</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

**DEFINITION**
The sign of the least significant halfword (bits 16-31) of the Instruction Word is extended 16 bits to the left to form a word. This word is algebraically subtracted from the word in the GPR specified by R. The resulting word is transferred to the GPR specified by R.

**SUMMARY**

(R)-(W[16-31]) SE → R

**CONDITION CODE**

CC1: ISI arithmetic exception
CC2: ISI R[0-31] is greater than zero
CC3: ISI R[0-31] is less than zero
CC4: ISI R[0-31] is equal to zero

**EXAMPLE**

Memory Location: 019B8
Hex Instruction: CB 82 83 9A (R=7)
Assembly Language Coding: SUI 7,X'839A'

**Before Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>100019B8</td>
<td>FFFF839A</td>
</tr>
</tbody>
</table>

**After Execution**

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>080019BC</td>
<td>00000000</td>
</tr>
</tbody>
</table>

**Note**
The immediate operand with sign extension is subtracted from the contents of GPR7. The result is transferred to GPR7. CC4 is set.
MULTIPLY BY MEMORY BYTE

CO08

DEFINITION
The byte in memory specified by the Effective Byte Address (EBA) is accessed and 24 zeros are appended to the most significant end to form a word. This word is algebraically multiplied by the word in the GPR specified by R+1. R+1 is the GPR one greater than specified by R. The double-precision result is transferred to the GPR specified by R and R+1.

NOTES
1. An arithmetic exception will never occur since the result of a multiplication can never exceed the length of the doubleword register.
2. GPR specified by R must have an even address.

SUMMARY
$0_{0-23}, (EBA) \times (R+1) \rightarrow R, R+1$

CONDITION CODE
CC1: Always zero
CC2: ISI (R, R+1) is greater than zero
CC3: ISI (R, R+1) is less than zero
CC4: ISI (R, R+1) is equal to zero

EXAMPLE
Memory Location: 2BA28
Hex Instruction: CO OA C3 D9
Assembly Language Coding: MPMB 0,'X'2C3D9'

Before Execution
PSWR GPRO GPR1
0002BA28 12345678 6F90C859
Memory Byte 2C3D9
40

After Execution
PSWR GPRO GPR1
2002BA2C 0000001B E4321640

Note
The contents of memory byte 2C3D9, with zeros prefixed, are multiplied by the contents of GPR1. The result is transferred to GPRO and GPR1. CC2 is set.
MULTIPLY BY MEMORY HALFWORD

CO000

<table>
<thead>
<tr>
<th>1 1 0 0 0 0</th>
<th>R</th>
<th>X</th>
<th>i</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
</tr>
</tbody>
</table>

DEFINITION
The halfword in memory specified by the Effective Halfword Address (EHA) is accessed and the sign bit (bit 16) is extended 16 bits to the left to form a word. This word is algebraically multiplied by the word in the GPR specified by R+1. R+1 is the GPR one greater than specified by R. The double-precision result is transferred to the GPR specified by R and R+1.

NOTES
1. An arithmetic exception will never occur since the result of a multiplication can never exceed the length of the doubleword register.
2. GPR specified by R must have an even address.

SUMMARY
(EHL)SE×(R+1)→ R,R+1

CONDITION CODE
CC1: Always zero
CC2: ISI(R, R+1) is greater than zero
CC3: ISI (R, R+1) is less than zero
CC4: ISI (R, R+1) is equal to zero

EXAMPLE
Memory Location: 096A4
Hex Instruction: CI 00 9B 57 (R=2, X=0, I=0)
Assembly Language Coding: MPMH 2,'9B56'

Before Execution
PSWR GPR2 GPR3 Memory Halfword 09B56
080096A4 12345678 00000003 FFFD

After Execution
PSWR GPR2 GPR3 Memory Halfword 09B56
100096A8 FFFFFFFF FFFFFFF7 FFFD

Note The contents of GPR3 are multiplied by the contents of memory halfword 09B56. The doubleword result is transferred to GPR2 and GPR3. CC3 is set.
MULTIPLY BY MEMORY WORD

CO00

**DEFINITION**

The word in memory specified by the Effective Word Address (EWA) is accessed and algebraically multiplied by the word GPR specified by R+1. R+1 is the GPR one greater than specified by R. The double-precision result is transferred to the GPR specified by R and R+1.

**NOTES**

1. An arithmetic exception will never occur since the result of a multiplication can never exceed the length of the doubleword register.

2. GPR specified by R must have an even address.

**SUMMARY**

(\(EWL\)x(R+1) \(\rightarrow (R,R+1)\))

**CONDITION CODE RESULTS**

- CC1: Always zero
- CC2: ISI (R, R+1) is greater than zero
- CC3: ISI (R, R+1) is less than zero
- CC4: ISI (R, R+1) is equal to zero

**EXAMPLE**

Memory Location: 04AC8
Hex Instruction: C3 00 4B 1C (R=6, X=0, I=0)
Assembly Language Coding: MPMW 6,X'4B1C'

Before Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR6</th>
<th>GPR7</th>
<th>Memory Word 04B1C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>00000000</td>
<td>00000000</td>
<td>80000000</td>
</tr>
</tbody>
</table>

After Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR6</th>
<th>GPR7</th>
<th>Memory Word 04B1C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>00000000</td>
<td>00000000</td>
<td>80000000</td>
</tr>
</tbody>
</table>

Note: The contents of GPR7 and memory word 04B1C are multiplied, and the result is transferred to GPR6 and GPR7. CC2 is set.
MULTIPLY REGISTER BY REGISTER

4000

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
</tr>
</tbody>
</table>

**DEFINITION**
The word GPR specified by \( R_S \) is algebraically multiplied by the word in the GPR specified by \( R_D+1 \). \( R_D+1 \) is the GPR one greater than specified by \( R_D \). The double-precision result is transferred to the GPR specified by \( R_D \) and \( R_D+1 \).

**NOTES**
1. The multiplicand register \( R_S \) can be any register, including register \( R_D+1 \); however, \( R_D \) must be an even-numbered register.
2. An arithmetic exception will never occur since the result of a multiplication can never exceed the length of the doubleword register.

**SUMMARY**

\[(R_S) \times (R_D+1) \rightarrow R_D, R_D+1\]

**CONDITION CODE**

\[
\begin{align*}
CC1: & \text{ Always zero} \\
CC2: & \text{ ISI } (R_D, R_D+1) \text{ is greater than zero} \\
CC3: & \text{ ISI } (R_D, R_D+1) \text{ is less than zero} \\
CC4: & \text{ ISI } (R_D, R_D+1) \text{ is equal to zero}
\end{align*}
\]

**RESULTS**

**EXAMPLE**

Memory Location: 0098E
Hex Instruction: 40 10 \((R_D=0, R_S=1)\)
Assembly Language Coding: MPR 1,0

<table>
<thead>
<tr>
<th>Before Execution</th>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSWR 1000098E</td>
<td>PSWR 20000990</td>
</tr>
<tr>
<td>GPRO 00000000</td>
<td>GPRO 00000000</td>
</tr>
<tr>
<td>GPR1 00000E1</td>
<td>GPR1 00000E1</td>
</tr>
</tbody>
</table>

Note
The content of GPR1 is multiplied by itself, and the doubleword product is transferred to GPRO and GPR1. CC2 is set.
MULTIPLY IMMEDIATE

The sign of the least significant halfword (bits 16-31) of the Instruction Word is extended 16 bits to the left to form a word. This word is algebraically multiplied by the word in the GPR specified by R+1. R+1 is the GPR one greater than specified by R. The result is transferred to the GPR specified by R and R+1.

NOTES
1. An arithmetic exception will never occur since the result of a multiplication can never exceed the length of the doubleword register.
2. The GPR specified by R must have an even address.

SUMMARY

\[(IW_{16-31}) \times (R+1) \rightarrow R,R+1\]

CONDITION CODE

- CC1: Always zero
- CC2: ISI \((R,R+1)\) is greater than zero
- CC3: ISI \((R,R+1)\) is less than zero
- CC4: ISI \((R,R+1)\) is equal to zero

EXAMPLE

| Memory Location: | 00634 |
| Hex Instruction: | CB 03 01 00 \(R=6\) |
| Assembly Language Coding: | MPI 6,'X'0100' |

| Before Execution |
|------------------|------------------|
| PSWR             | GPR6             | GPR7 |
| 20000634         | 12345678         | F37A9B15 |

| After Execution |
|------------------|------------------|
| PSWR             | GPR6             | GPR7 |
| 10000638         | FFFFFFFF3        | 7A9B1500 |

Note: The immediate operand, sign extended, is multiplied by the contents of GPR7. The result is transferred to GPR6 and GPR7. CC3 is set.
DEFINITION
The byte in memory specified by the Effective Byte Address (EBA) is accessed and 24 zeros are appended to the most significant end to form a word. This word is algebraically divided into the doubleword in the GPR specified by R and R+1. R+1 is the GPR one greater than specified by R. The resulting quotient is then transferred to the GPR specified by R+1, and the remainder is transferred to the GPR specified by R. The sign of the GPR specified by R (remainder) is set to the original sign of the dividends. The sign of the GPR specified by R+1 (quotient) will be the algebraic product of the original signs of the dividend and the divisor except when the absolute value of the dividend is less than the absolute value of the divisor. In that case, the resulting quotient (GPR specified by R+1) will be set to zero.

NOTES
1. An arithmetic exception occurs if the value of the quotient exceeds 32 bits. If an arithmetic exception occurs, the original dividend will be restored in the GPR specified by R and R+1.
2. GPR specified by R must have an even address.

SUMMARY
(R,R1) / [00_23*(EBL)] → R+1
Remainder → R

CONDITION CODE
CC1: ISI arithmetic exception
CC2: ISI (R+10-31) is greater than zero
CC3: ISI (R+10-31) is less than zero
CC4: ISI (R+10-31) is equal to zero

EXAMPLE
Memory Location: 03000
Hex Instruction: C4 08 30 BF (R=0, X=0, I=0)
Assembly Language Coding: DVMB 0,X'30BF'
Before Execution
PSWR 00000000
GPRO 00000000
GPR1 00000139
Memory Byte 030BF
After Execution
PSWR 00000000
GPRO 00000000
GPR1 0000004E
Memory Byte 030BF

Note
The doubleword contents of GRO and GPR1 are divided by the content of memory byte 030BF with 24 zeros prefixed. The quotient is transferred to GPR1 and the remainder is transferred to GPR0. CC2 is set.
DIVIDE BY MEMORY HALFWORD

C400

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>R</th>
<th>X</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td>30</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DEFINITION

The halfword in memory specified by the Effective Halfword Address (EHA) is accessed, and the sign is extended 16 bits to the left to form a word. This word is algebraically divided into the doubleword in the GPR specified by R and R+1. R+1 is the GPR one greater than specified by R. The resulting quotient is then transferred to the GPR specified by R+1 and the remainder is transferred to the GPR specified by R. The sign of the GPR specified by R (remainder) is set to the original sign of the dividend. The sign of the GPR specified by R+1 (quotient) will be the algebraic product of the original signs of the dividend and the divisor, except when the absolute value of the dividend is less than the absolute value of the divisor. In that case, the resulting quotient (GPR specified by R+1) will be set to zero.

NOTES

1. An arithmetic exception occurs if the value of the quotient exceeds 32 bits. If an arithmetic exception occurs, the original dividend will be restored in the GPR specified by R and R+1.

2. The GPR specified by R must have an even address.

SUMMARY

Expression:

(EHL)SE → (R, R+1)

CONDITION CODE

RESULTS

CC1: ISI arithmetic exception
CC2: ISI R+1 0_31 is greater than zero
CC3: ISI R+1 0_31 is less than zero
CC4: ISI R+1 0_31 is equal to zero

EXAMPLE

Memory Location: 05A94
Hex Instruction: C7 00 5D 6B (R=6, X=0, I=0)
Assembly Language Coding: DVMH 6, '5D6A'

Before Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>00005A94</td>
<td>00000000</td>
<td>0000003B</td>
</tr>
<tr>
<td>Memory Halfword 05D6A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>10005A98</td>
<td>00000005</td>
<td>FFFFFF9</td>
</tr>
<tr>
<td>Memory Halfword 05D6A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note

The doubleword contents of GPR6 and GPR7 are divided by the contents of memory halfword 05D6A with sign extension. The quotient is transferred to GPR7 and the remainder is transferred to GPR6. CC3 is set.
DIVIDE BY MEMORY WORD

C400

DEFINITION
The word in memory specified by the Effective Word Address (EWA) is accessed and algebraically divided into the doubleword in the GPR specified by R and R+1. R+1 is the GPR one greater than specified by R. The resulting quotient is then transferred to the GPR specified by R+1, and the remainder is transferred to the GPR specified by R. The sign of the GPR specified by R (remainder) is set to the original sign of the dividend. The sign of the GPR specified by R+1 (quotient) will be the algebraic product of the original signs of the dividend and the divisor, except when the absolute value of the dividend is less than the absolute value of the divisor. In that case, the resulting quotient (GPR specified by R+1) will be set to zero.

NOTES
1. An arithmetic exception occurs if the value of the quotient exceeds 32 bits. If an arithmetic exception occurs, the original dividend will be restored in the GPR specified by R and R+1.

2. The GPR specified by R must have an even address.

SUMMARY EXPRESSION
(R,R+1)/(EWA) → R+1

Remainder → R

CONDITION CODE
CC1: ISI arithmetic exception
CC2: ISI R+10-31 is greater than zero
CC3: ISI R+10-31 is less than zero
CC4: ISI R+10-31 is equal to zero

EXAMPLE
Memory Location: 078C0
Hex Instruction: C6 00 7B 5C (R=4, X=0, I=0)
Assembly Language Coding: DVMW 4,'7B5C'

Before Execution
PSWR GPR4 GPR5 Memory Word 07B5C
400078C0 00000000 039A20CF FC000000

After Execution
PSWR GPR4 GPR5 Memory Word 07B5C
080078C4 039A20CF 00000000 FC000000

Note
The doubleword obtained from GPR4 and GPR5 is divided by the contents of memory word 07B5C. The quotient is transferred to GPR5, and the remainder is transferred to GPR4. CC4 is set.
DIVIDE REGISTER BY REGISTER

4400

**DEFINITION**

The word in the GPR specified by Rs is algebraically divided into the
doubleword in the GPR specified by Rd and Rd+1. Rd+1 is the GPR one
greater than specified by Rd. The resulting quotient is then transferred to
the GPR specified by Rd+1, and the remainder is transferred to the GPR
specified by Rd. The sign of the GPR specified by Rd (remainder) is set to
the original sign of the dividend. The sign of the GPR specified by Rd+1
(quotient) will be the algebraic product of the original signs of the
dividend and the divisor, except when the absolute value of the dividend is
less than the absolute value of the divisor. In that case, the resulting
quotient (GPR specified by Rd+1) will be set to zero.

**NOTES**

1. An arithmetic exception occurs if the value of the quotient exceeds 32
   bits. If an arithmetic exception occurs, the original dividend will
   be restored in the GPR specified by R and R+1.

2. The GPR specified by Rd must have an even address.

3. Rs must not equal Rd or Rd+1.

**SUMMARY**

\((R_d,R_{d+1})/R_s \rightarrow R_{d+1}\)

**CONDITION CODE**

CC1: ISI arithmetic exception
CC2: ISI \(R_{d+1}0_{31}\) is greater than zero
CC3: ISI \(R_{d+1}0_{31}\) is less than zero
CC4: ISI \(R_{d+1}0_{31}\) is equal to zero

**RESULTS**

Memory Location: 04136
Hex Instruction: 47 20 \((R_d=6, R_s=2)\)
Assembly Language Coding: DVR 2,6

Before Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR2</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>10004136</td>
<td>0000000A</td>
<td>00000000</td>
<td>0000000F</td>
</tr>
</tbody>
</table>

After Execution

<table>
<thead>
<tr>
<th>PSWR</th>
<th>GPR2</th>
<th>GPR6</th>
<th>GPR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>20004138</td>
<td>0000000A</td>
<td>00000005</td>
<td>00000019</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The doubleword obtained from GPR6 and GPR7 is divided by the contents of
GPR2. The quotient is transferred to GPR7, and the remainder is trans-
ferred to GPR6. CC2 is set.
DEFINITION
The sign of the least significant halfword (bits 16-31) of the Instruction Word is extended 16 bits to the left to form a word. This word is algebraically divided into the doubleword in the GPR specified by R and R+1. R+1 is the GPR one greater than specified by R. The resulting quotient is then transferred to the GPR specified by R+1, and the remainder is transferred to the GPR specified by R. The sign of the GPR specified by R (remainder) is set to the original sign of the dividend. The sign of the GPR specified by R+1 (quotient) will be the algebraic product of the original signs of the dividend and the divisor, except when the absolute value of the dividend is less than the absolute value of the divisor. In that case, the resulting quotient (GPR specified by R+1) will be set to zero.

NOTES
1. An arithmetic exception occurs if the value of the quotient exceeds 32 bits. If an arithmetic exception occurs, the original dividend will be restored in the GPR specified by R and R+1.

2. The GPR specified by R must have an even address.

SUMMARY
(R,R+1)/(IW 16-31)SE → R+1

CONDITION CODE
CC1: ISI arithmetic exception
CC2: ISI R+1 0-31 is greater than zero
CC3: ISI R+1 0-31 is less than zero
CC4: ISI R+1 0-31 is equal to zero

EXAMPLE
Memory Location: 08000
Hex Instruction: C9 04 FF FD (R=2)
Assembly Language Coding: DVI 2,-3

Before Execution
PSWR  GPR2  GPR3
04008000 00000000 000001B7

After Execution
PSWR  GPR2  GPR3
10008004 00000001 FFFFFFF6F

Note
The doubleword obtained from GPR2 and GPR3 is divided by the immediate operand, sign extended. The quotient is transferred to GPR3, and the remainder is transferred to GPR2. CC3 is set.
EXTEND SIGN

0004

The sign (bit 0) of the contents of the GPR specified by R+1 is extended through all 32 bit positions of the GPR specified by R.

(R+10) → R0-31

CC1: Always zero
CC2: ISI R0-31 is greater than zero
CC3: ISI R0-31 is less than zero
CC4: ISI R0-31 is equal to zero

Memory Location: 0083A
Hex Instruction: 00 84 (R=1)
Assembly Language Coding: ES 1

Before Execution
PSWR 0800083A
GPR1 0000B074
GPR2 8000C361

After Execution
PSWR 1000083C
GPR1 FFFFFFFF
GPR2 8000C361

Note Bits 0-31 of GPR1 are set to one's. CC3 is set.
DEFINITION
The contents of the GPR specified by R are incremented by one if bit position 0 of the GPR specified by R+1 is equal to one. R+1 is the GPR one greater than specified by R.

SUMMARY
(R)+1, if(R+1)_0 = 1

CONDITION CODE
CC1: ISI arithmetic exception
CC2: ISI R_0-31 is greater than zero
CC3: ISI R_0-31 is less than zero
CC4: ISI R_0-31 is equal to zero

RESULTS

EXAMPLE

Memory Location:
Hex Instruction:
Assembly Language Coding:

Before Execution
PSWR
GPR6
GPR7
40000FFE
783A05B2
92CD061F

After Execution
PSWR
GPR6
GPR7
20001000
783A05B3
92CD061F

Note
The contents of GPR6 are incremented by one, and the result is returned to GPR6. CC2 is set.
The Floating-Point Arithmetic instructions provide the capability to add, subtract, multiply, or divide operands of large magnitude with precise results. A floating-point number is made up of three parts: a sign, a fraction, and an exponent. The sign applies to the fraction and denotes a positive or negative value. The fraction is a binary number with an assumed radix point between the sign bit and the most significant bit. The exponent is a 7-bit binary power to which the base 16 is raised. The quantity that the floating-point number represents is obtained by multiplying the fraction by the number 16 raised to the power represented by the exponent.

The following instruction format is used for all floating-point operations:

```
<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>F</th>
<th>WORD ADDRESS</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Bits 0-5 define the Operation Code.

Bits 6-8 designate a General Purpose Register address (0-7).

Bits 9-10 designate one of three index registers.

Bit 11 indicates whether an indirect addressing operation is to be performed.

Bits 12-31 directly specifies the address of the operand when the X and I fields are equal to zero. If X is not equal to zero, indirect addressing is specified. Bit 12 (F) is used as an augment bit by the Floating-Point instructions.

Execution of all Floating-Point Arithmetic instructions causes a Condition Code to be set to indicate whether the result of the operation was greater than, less than, or equal to zero. Arithmetic exceptions produced by a Floating-Point operation are also reflected by the Condition Code results.

The meaning of the Condition Codes differ for the execution of the Floating-Point instructions. CC1 is set by an Arithmetic Exception condition (underflow or overflow). To differentiate between these exceptions, CC4 is also set when the overflow condition occurs. In both instances, either CC2 or CC3 is used to indicate the state of what would have been the sign of the resultant fraction had the arithmetic exception not occurred. The following table reflects the possible Condition Code settings:

<table>
<thead>
<tr>
<th>Condition Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC1  CC2 CC3  CC4</td>
<td></td>
</tr>
<tr>
<td>1  0  0  0</td>
<td>Arithmetic exception</td>
</tr>
<tr>
<td>0  1  0  0</td>
<td>Positive fraction</td>
</tr>
<tr>
<td>0  0  1  0</td>
<td>Negative</td>
</tr>
<tr>
<td>0  0  0  1</td>
<td>Zero fraction</td>
</tr>
<tr>
<td>1  1  0  0</td>
<td>Exponent Underflow, positive fraction</td>
</tr>
<tr>
<td>1  0  1  0</td>
<td>Exponent Underflow, negative fraction</td>
</tr>
<tr>
<td>1  1  0  1</td>
<td>Exponent Overflow, positive fraction</td>
</tr>
<tr>
<td>1  0  1  1</td>
<td>Exponent Overflow, negative fraction</td>
</tr>
</tbody>
</table>
A floating-point number can be represented in two different formats: word and doubleword. These two formats are the same except that the doubleword contains eight additional hexadecimal digits of significance in the fraction. These two formats are shown below.

The floating-point number, in either format, is made up of three parts: a sign, a fraction, and an exponent. The sign bit (bit 0) applies to the fraction and denotes a positive or negative value. The fraction is a hexadecimal normalized number with a radix point to the left of the highest order fraction bit (bit 8). The exponent (bits 1-7) is a 7-bit binary number to which the base 16 is raised.

Negative exponents are carried in the two's complement format. To remove the sign and therefore enable exponents to be compared directly, both positive and negative exponents are biased up by $40_{16}$ (excess $64_{10}$ notation). The quantity that a floating-point number represents is obtained by multiplying the fraction by the number $16_{10}$ raised to the power of the exponent minus $40_{16}$.

A positive floating-point number is converted to a negative floating-point number by taking the two's complement of the positive fraction and the one's complement of the biased exponent. If the minus one case is ruled illegitimate, all floating-point numbers can be converted from positive to negative and from negative to positive by taking the two's complement of the number in floating-point format. Signed numbers in the floating-point format can then be compared directly, one with another, by using the Compare Arithmetic class of instructions.

All floating-point operands must be normalized before being operated on by a floating-point instruction. A positive floating-point number is normalized when the value of the fraction is less than one and greater than or equal to one-sixteenth ($1 > F \geq 1/16$). A negative floating-point number is normalized when the value of the fraction is greater than minus one and less than or equal to minus one-sixteenth ($-1 < F \leq -1/16$). All floating-point answers are normalized by the CPU. If a floating-point operation results in a minus one of the form $1 \text{XXX XXXX 0000}...0000$, the CPU will convert that result to a legitimate normalized floating-point number of the form $1 \text{YYY YYY} 1111\text{0000}...0000$, where YYY YYY is one less than XXX XXXX.

A hexadecimal guard digit is appended to the least significant hexadecimal digit of the floating-point word operands by the CPU. This guard digit is carried throughout all floating-point word computations. The most significant bit of the guard digit is used as the basis for rounding by the CPU at the end of every floating-point word computation.
ADD FLOATING-POINT WORD

E008

DEFINITION
The floating-point operand in memory is accessed. If either of the floating-point numbers is negative, the one's complement of the base 16 exponent (bits 1-7) is taken of the negative number. Both exponents are then stripped of their $40_{16}$ bias and algebraically compared. If the two exponents are equal, the signed fractions of the two numbers are algebraically added. If the exponents differ, and the difference is greater than or equal to one, or less than or equal to six (1 exponent difference 6), the fraction of the operand containing the smaller exponent is shifted right one hexadecimal digit. After exponent equalization, the fractions are added algebraically. The normalized and rounded sum of the two fractions is placed in bit positions 0 and 8-31 of GPR d. The resulting exponent is biased up by $40_{16}$, and, if the resulting fraction is negative, the one's complement of the exponent is placed in bit positions 1-7 of GPR d.

NOTES
1. If the resulting fraction equals zero, the exponent and fraction are set to zero in GPR d.
2. Operands are expected to be normalized.
3. If the exponent difference is greater than six, the operand having the larger exponent is normalized and placed in the GPR specified by R.

SUMMARY
(R)+(EWL) → (R)

CONDITION CODE RESULTS
CC1: ISI arithmetic exception
CC2: ISI $R_{0,8-31}$ is greater than zero
CC3: ISI $R_{0,8-31}$ is less than zero
CC4: ISI $R_{0,8-31}$ is equal to zero
ADD FLOATING POINT DOUBLEWORD

E008

1 1 1 0 0 0  R  X  I  1  DOUBLEWORD OPERAND ADDRESS  0 1 0

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

DEFINITION

The floating-point operand in memory is accessed. If either of the floating-point numbers is negative, the one's complement of the base 16 exponent (bits 1-7) is taken of the negative number. Both exponents are then stripped of their $40_{16}$ bias and algebraically compared. If the two exponents are equal, the signed fractions of the two numbers are algebraically added. If the exponents differ, and the difference is greater than or equal to one, or less than or equal to six ($1 \leq \text{exponent difference} \leq 6$), the fraction of the operand containing the smaller exponent is shifted right one hexadecimal digit. After exponent equalization, the fractions are added algebraically. The normalized and rounded sum of the two fractions is placed in bit positions 0 and 8-63 of GPR d+1. The resulting exponent is biased up by $40_{16}$, and, if the resulting fraction is negative, the one's complement of the exponent is placed in bit positions 1-7 of GPR d.

NOTES

1. If the resulting fraction equals zero, the exponent and fraction are set to zero in GPR d+1.

2. Operands are expected to be normalized.

3. If the exponent difference is greater than 13, the operand having the larger exponent is normalized and placed in the GPR specified by R, R+1.

SUMMARY

(R),(R+1)+(EWL),(EWL+1)→ (R),(R+1)

CONDITION CODE

CC1: ISI arithmetic exception

CC2: ISI $R_{0,8-31}$ is greater than zero

CC3: ISI $R_{0,8-31}$ is less than zero

CC4: ISI $R_{0,8-31}$ is equal to zero

RESULTS

Assembly Language Coding: ADFD R,X'(DW Op Addr)
DEFINITION

The floating-point operand in memory is accessed. If either the floating-point number in the GPR or memory is negative, the one’s complement of the base-16 exponent (bits 1-7) is taken. Both exponents are then stripped of their $40_{16}$ bias and algebraically compared. If the two exponents are equal, the 24-bit signed fractions are algebraically subtracted (i.e., the memory operand is subtracted from the GPR or GPR’s). If the exponents differ, and the difference is greater than one, or less than six ($1 \leq$ exponent difference $\leq 6$), the fraction of the operand containing the smaller exponent is shifted right one hexadecimal digit at a time until the exponents are equalized. The exponent of this operand is effectively incremented by one each time the fraction is shifted right one hexadecimal. After exponent equalization, the fractions are subtracted algebraically. The normalized and rounded difference between the two fractions is placed in bit positions 0 and 8-31 of GPR d. The resulting exponent is biased up by $40_{16}$, and, if the resulting fraction is negative, the one’s complement of the exponent is placed in bit positions 1-7 of GPR d.

NOTES

1. If the resulting fraction is equal to zero, the exponent and fraction are set to zero in the GPR or GPR’s.

2. Operands are expected to be normalized.

3. If the exponent difference is greater than six, the operand having the larger exponent is normalized and placed in the GPR specified by R.

SUMMARY

EXPRESSION

(R)-(EWL) → (R)

CONDITION CODE

RESULTS

CC1: ISI arithmetic exception
CC2: ISI R0,8-31 is greater than zero
CC3: ISI R0,8-31 is less than zero
CC4: ISI R0,8-31 is equal to zero

Assembly Language Coding: SUFW R, X'(W Op Addr)
DEFINITION

The floating-point operand in memory is accessed. If either the floating-point number in the GPR or memory is negative, the one's complement of the base 16 exponent (bits 1-7) is taken. Both exponents are then stripped of their 40₆₁ bias and algebraically compared. If the two exponents are equal, the 24-bit signed fractions are algebraically subtracted (i.e., the memory operand is subtracted from the GPR or GPR s). If the exponents differ, and the difference is greater than or equal to one, or less than or equal to six (1 ≤ exponent difference ≤ 6), the fraction of the operand containing the smaller exponent is shifted right one hexadecimal digit at a time until the exponents are equalized. The exponent of this operand is effectively incremented by one each time the fraction is shifted right one hexadecimal digit. After exponent equalization, the fractions are subtracted algebraically. The normalized and rounded difference between the two fractions is placed in bit positions 0 and 8-63 of GPR d+1. The resulting exponent is biased up by 40₆₁, and, if the resulting fraction is negative, the one's complement of the exponent is placed in bit positions 1-7 of GPR d.

NOTES

1. If the resulting fraction is equal to zero, the exponent and fraction are set to zero in the GPR or GPR s.
2. Operands are expected to be normalized.
3. If the exponent difference is greater than 13, the operand having the larger exponent is normalized and placed in the GPR specified by R, R+1.

SUMMARY

(R),(R+1)-(EWL),(EWL+1) → (R),(R+1)

CONDITION CODE RESULTS

CC1: ISI arithmetic exception
CC2: ISI R₀,8-31 is greater than zero
CC3: ISI R₀,8-31 is less than zero
CC4: ISI R₀,8-31 is equal to zero

Assembly Language Coding: SUFD R,X'(DW Op Addr)
MULTIPLY FLOATING-POINT WORD

E408

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>T</th>
<th>WORD OPERAND ADDRESS</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

DEFINITION

The floating-point operand fraction is multiplied by the fraction of GPR d. If either one or both of the floating-point numbers are negative, the exponent of the negative number is changed to its one's complement. Both exponents are then stripped of their 4016 bias and algebraically added. The normalized and rounded product of the multiplication is placed in bits 0 and 8-31 of GPR d. The resulting exponent is biased up by 4016, and, if the resulting fraction is negative, the one's complement of the resulting exponent is placed in bits 1-7 of GPR d.

NOTE

Operands are expected to be normalized.

SUMMARY

\( (EWL_{0,8-31}) \times (R_{0,8-31}) \rightarrow R_{0,8-31} \)

\( (EWL_{1-7}) + (R_{1-7}) \rightarrow R_{1-7} \)

CONDITION CODE RESULTS

CC1: ISI arithmetic exception
CC2: ISI \( R_{0,8-31} \) is greater than zero
CC3: ISI \( R_{0,8-31} \) is less than zero
CC4: ISI \( R_{0,8-31} \) is equal to zero

Assembly Language Coding:

MPFW R,X'(W Op Addr)'

Assembly Language Coding:

MPFW R,X'(W Op Addr)'

6-176
MULTIPLY FLOATING-POINT DOUBLEWORD

MPFD
d,*m,x

E408

DEFINITION
The floating-point operand fraction is multiplied by the fraction of GPR d+1. If either one or both of the floating-point numbers are negative, the exponent of the negative number is changed to its one's complement. Both exponents are then stripped of their 4016 bias and algebraically added. The normalized and rounded product of the multiplication is placed in bits 0 and 8-63 of GPR d+1. The resulting exponent is biased up by 4016, and if the resulting fraction is negative, the one's complement of the resulting exponent is placed in bits 1-7 of GPR d.

NOTE
Operands are expected to be normalized.

SUMMARY

EXPRESSION

\[(EWL_0,8-31 \cdot EWL+10-31) \times (R_0,8-31 \cdot R+10-31)\]

\[\rightarrow R_0,8-31 \cdot R+10-31\]

\[(EWL_{1-7})+(R_{1-7}) \rightarrow R_{1-7}\]

CONDITION CODE

RESULTS

CC1: ISI arithmetic exception
CC2: ISI R0,8-31 is greater than zero
CC3: ISI R0,8-31 is less than zero
CC4: ISI R0,8-31 is equal to zero

Assembly Language Coding: MPFD R,X'(DW Op Addr)'

6-177
DIVIDE FLOATING-POINT WORD

E400

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

**DEFINITION**
The floating-point operand in memory (divisor) is accessed, and the fraction is divided into the fraction of GPR d. If either one or both of the floating-point numbers are negative, the one's complement of the exponent is taken. Both exponents are then stripped of their 40₁₆ bias, and the exponent of the divisor is subtracted algebraically from the exponent of the dividend. The normalized and rounded quotient is placed in bit 0 and bit positions 8-31 of the GPR d. The resulting exponent is biased up by 40₁₆, and, if the resulting fraction is negative, the one's complement of the resulting fraction is placed in bits 1-7 of GPR d.

**NOTE**
Operands are expected to be normalized.

**SUMMARY**

<table>
<thead>
<tr>
<th>EXPRESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R₀,8-31)/(EWL₀,8-31) → R₀,8-31</td>
</tr>
<tr>
<td>(R₁₋7)-(EWL₁₋7) → R₁₋7</td>
</tr>
</tbody>
</table>

**CONDITION CODE**

<table>
<thead>
<tr>
<th>RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC1: ISI arithmetic exception</td>
</tr>
<tr>
<td>CC2: ISI R₀,8-31 is greater than zero</td>
</tr>
<tr>
<td>CC3: ISI R₀,8-31 is less than zero</td>
</tr>
<tr>
<td>CC4: ISI R₀,8-31 is equal to zero</td>
</tr>
</tbody>
</table>

Assembly Language Coding: DVFW R,X'(W Op Addr)'
The floating-point operand in memory (divisor) is accessed and the fraction is divided into the fraction of GPR d+1. If either one or both of the floating-point numbers are negative, the one's complement of the exponent is taken. Both exponents are then stripped of their 40[16] bias, and the exponent of the divisor is subtracted algebraically from the exponent of the dividend. The normalized and rounded quotient is placed in bit 0 and bit positions 8-63 of the GPR d+1. The resulting exponent is biased up by 40[16], and, if the resulting fraction is negative, the one's complement of the resulting fraction is placed in bits 1-7 of GPR d.

Operands are expected to be normalized.

(R0,8-31,R+10-31)/(EWL0,8-31,EWL+10-31)

= R0,8-31,R+10-31

(R1-7)-(EWL1-7) → R1-7

Assembly Language Coding: DVFD R,X'(DW Op Addr)
This group of instructions allows the mainframe to perform Execute, No Op, Halt, and Wait operations.

Control instructions use the Memory Reference and Interregister instruction formats. Several of the Control instructions vary the basic Interregister format in that certain portions are not used and are left blank.

**Memory Reference**

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R</th>
<th>X</th>
<th>I</th>
<th>F</th>
<th>WA</th>
<th>C</th>
</tr>
</thead>
</table>

- Bits 0-5 define the Operation Code.
- Bits 6-8 designate a General Purpose Register address (0-7).
- Bits 9-10 designate one of three index registers.
- Bit 11 indicates whether an indirect addressing operation is to be performed.
- Bits 12-31 specify the address of the operand when the X and I fields are equal to zero.

**Interregister**

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R_D</th>
<th>R_S</th>
<th>AUG CODE</th>
</tr>
</thead>
</table>

- Bits 0-5 define the Operation Code.
- Bits 6-8 designate the register to contain the result of the operation.
- Bits 9-11 designate the register which contains the source operand.
- Bits 12-15 define the Augmenting Operation Code.

**Condition Code Utilization**

Condition Code results for Execute operations will be dependent on the instruction that was performed. All other control operations leave the current Condition Code unchanged.
BRANCH AND RESET INTERRUPT

F900

<table>
<thead>
<tr>
<th>1 1 1 1 1 0</th>
<th>0 1 0</th>
<th>X</th>
<th>1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5</td>
<td>6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DEFINITION
This instruction resets the highest active interrupt level and branches to the address indicated.

When coded indirect, this instruction causes the target PSW or PSD to be loaded into the CPU, resets the highest active interrupt level, and branches to the address in the PSW or PSD.

CONDITION CODE

CC1: ISI if (I) is equal to one and (EWL1) is equal to one.
CC2: ISI if (I) is equal to one and (EWL2) is equal to one.
CC3: ISI if (I) is equal to one and (EWL3) is equal to one.
CC4: ISI if (I) is equal to one and (EWL4) is equal to one.

Assembly Language Coding: BRI X'(Branch Addr)'

NOTES
1. Used only with interrupts operating in Active mode.
2. Privileged instruction.
3. If granularity of PSD is MAP, the contents of the MAP are changed in accord with the instructions in PSD word 2.
4. This instruction cannot be used with Post-indexing.
LOAD PROGRAM STATUS DOUBLEWORD

F980

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

**DEFINITION**

Causes the PSD addressed by the instruction to be loaded into the Program Status Doubleword Registers.

**SUMMARY**

(EDL) \( \rightarrow \) (PSDR)

**CONDITION CODE RESULTS**

CC1: Changed by the PSD being loaded
CC2: Changed by the PSD being loaded
CC3: Changed by the PSD being loaded
CC4: Changed by the PSD being loaded

Assembly Language Coding: LPSD X'(PSD Addr)'

**NOTES**

1. Privileged instruction.
2. Causes system to go Mapped or Unmapped in accordance with codes in PSD that is being loaded.
3. This instruction does not modify contents of the MAP.
4. Attempt to execute this instruction in PSW mode will result in an undefined instruction trap.
5. The Block External Interrupts will be changed in accord with bits 48 and 49 of the PSD.
LOAD PROGRAM STATUS DOUBLEWORD AND CHANGE MAP

LPSDCM d,*m,x

FA80

<table>
<thead>
<tr>
<th>1 1 1 1 0 1 0 1</th>
<th>X</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10 11 12 13 14 15 16 17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>20 21 22 23 24 25 26 27</td>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td>30 31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DEFINITION

Causes the PSD addressed by the instruction to be loaded into the Program Status Doubleword Registers, and the MAP to be loaded in accord with the BPIX and CPIX contents of the PSD. If the PSD defines the mapped condition, this instruction will cause the CPU to go mapped.

SUMMARY

(EDL) → (PSDR)

EXPRESSION

(MIDL) → Map Registers

CONDITION CODE RESULTS

CC1: Changed by the PSD being loaded
CC2: Changed by the PSD being loaded
CC3: Changed by the PSD being loaded
CC4: Changed by the PSD being loaded

Assembly Language Coding: LPSDCM X'(PSD Addr)'

NOTES

1. Privileged instruction.
2. The Block External Interrupts will be changed in accord with bits 48 and 49 of the PSD.
3. Attempt to execute this instruction in PSW mode will result in an undefined instruction trap.
LOAD CONTROL SWITCHES

0003

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

DEFINITION
The contents of Control Switches (CSW) 0-15 are transferred to bit positions 0-15 of the GPR specified by R. Bit positions 16-31 of the GPR specified by R are cleared to zeros.

SUMMARY
(CS\textsubscript{0-15}) \rightarrow R\textsubscript{0-15}

CONDITION CODE RESULTS
CC1: Always zero
CC2: ISI (R\textsubscript{0-31}) is greater than zero
CC3: ISI (R\textsubscript{0-31}) is less than zero
CC4: ISI (R\textsubscript{0-31}) is equal to zero

EXAMPLE

Memory Location: 06002
Hex Instruction: 03 83 (R=7)
Assembly Language Coding: LCS 7

Before Execution
PSWR 00006002
GPR7 FFFFFFFF

After Execution
PSWR 10006004
GPR7 82000000

Note
Bit positions 0 and 6 of GPR7 are set and all other bits are cleared. CC3 is set.
DEFINITION

The word in the GPR specified by R is transferred to the Instruction register to be executed as the next instruction. If this instruction is not a Branch, the next instruction executed (following execution of the instruction in register R) is in the sequential memory location following the EXR instruction. If the GPR specified by R does contain a Branch instruction, the Program Status Word Register (PSWR) is changed accordingly.

NOTES

1. If two halfword instructions are in the GPR specified by R, only the left halfword instruction is executed.

2. An Unimplemented Instruction trap is generated if an EXR instruction attempts to execute an Unimplemented instruction or another Execute instruction.

3. The "PSD mode only" instructions cannot be targets of EXR, EXRR, or EXM.

CONDITION CODE RESULTS

Defined by the executed instruction.

Assembly Language Coding: EXR R
EXRR

EXECUTE REGISTER RIGHT

C807

| 1 | 1 | 0 | 0 | 1 | 0 | R | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

DEFINITION
The contents of the least significant halfword (bits 16-31) of the GPR specified by R are transferred to the most significant halfword position (bits 0-15) of the Instruction register to be executed as the next instruction. If this halfword instruction is not a Branch, the next instruction executed (following execution of the halfword instruction transferred to the Instruction register) is in the sequential memory location following the EXRR instruction. If the instruction transferred to the Instruction register is a Branch instruction, the Program Status Word Register (PSWR) is changed accordingly.

NOTE
1. An unimplemented Instruction trap is generated if an EXRR instruction attempts to execute an Unimplemented instruction or another Execute instruction.
2. The "PSD mode only" instructions cannot be targets of EXR, EXRR, or EXM.

SUMMARY EXPRESSION
(R_{16-31}) \rightarrow I_{0-15}

CONDITION CODE RESULTS
Defined by the executed instruction.

Assembly Language Coding: EXRR R
**EXECUTE MEMORY**

EXM

*E*, X

---

**DEFINITION**

The word in memory specified by the Effective Address (EA) is accessed and executed as the next instruction. If this instruction is not a Branch, the next instruction executed (following execution of the instruction specified by the EA) is in the next sequential memory location following the EXM instruction. If the instruction in memory specified by the EA is a Branch instruction, the Program Status Word Register (PSWR) is changed accordingly.

**NOTES**

1. If two halfword instructions are in the memory location specified by the EA, bit 30 of the EA determines which halfword instruction is executed. When bit 30 equals zero, the left halfword is executed. When bit 30 equals one, right halfword is executed.

2. An Unimplemented Instruction trap is generated if an EXM instruction attempts to execute an Unimplemented instruction or another Execute instruction.

3. The "PSD mode only" instructions cannot be targets of EXR, EXRR, or EXM.

**CONDITION CODE RESULTS**

(EWL0-31) \(\rightarrow I\), if \(E_{A30}=0\)

(EWL16-31) \(\rightarrow I\), if \(E_{A30}=1\)

Defined by the executed instruction.

Assembly Language Coding: EXM X' (Op Addr)'
The execution of this instruction causes computer operation to be stopped. This includes input/output transfers and the servicing of priority interrupts. I/O in progress will be completed, but no interrupts will be serviced. Leaving a HALT condition requires depressing the RUN/HALT switch on the Systems Control Panel.

Assembly Language Coding: HALT

This is a privileged instruction.
DEFINITION
The execution of this instruction causes the CPU to enter the Idle mode and lights the Wait indicator on the System Control Panel. Input/output transfers and priority interrupt servicing continue. If an interrupt occurs during a Wait condition, a return to the Wait occurs after the interrupt is serviced.

CONDITION CODE
CC1: No change
CC2: No change
CC3: No change
CC4: No change

RESULTS

Assembly Language Coding: WAIT

NOTE
If there is an attempt to execute a WAIT with interrupts blocked, a Block Mode Timeout Trap will be generated.
NOP

NO OPERATION

0002

The Assembler uses the No Operation instruction to pad a halfword instruction which forces the next instruction to start on a word boundary, if the next instruction is a word instruction. It is also used whenever there is a need for an executable instruction that does not alter the machine status.

CONDITION CODE

CC1: No change
CC2: No change
CC3: No change
CC4: No change

RESULTS

Assembly Language Coding: NOP
**DEFINITION**

This instruction is a control class unprivileged instruction used to start and stop the operation of the Internal Processing Unit. When the SIPU instruction is executed, this instruction functions as a START IPU instruction in the CPU and a STOP IPU instruction in the IPU.

**INTRODUCTION**

To start IPU processing, the CPU stores the new Program Status Doubleword (PSD) into words 3 and 4 of the Start IPU Trap Context Block which is pointed to by the address contained in the Start IPU trap vector location 2E4. The CPU then executes the SIPU X'000A' instruction which sends a start signal to the IPU and informs the IPU that a new PSD is available for execution. The IPU stores the old PSD into words 1 and 2 of the Start IPU Trap Context Block and IPU Status into word 5. The IPU then fetches the new PSD words 3 and 4 from the context block and begins to execute the instructions in memory as directed by the new PSD.

To stop the IPU processing, the CPU stores a new PSD in words 3 and 4 of the Stop IPU Trap Context Block (TCB) which is pointed to by the address contained in the Stop IPU Trap vector location 2F4. The Stop IPU Trap Context Block (TCB) is used when the IPU then executes an SIPU (X'000A') instruction which is imbedded in the IPU software code. The IPU stores the old PSD into words 1 and 2 of the context block and the IPU Status into word 5 of the context block. The IPU then traps the CPU at location 2EO which indicates that the IPU execution of the SIPU instruction has taken place. The IPU then fetches the new PSD from words 3 and 4 of the context block which can point to a privileged HALT or WAIT instruction to stop the IPU.

The new PSD in the STOP IPU context block may direct the IPU to execute code other than a HALT or WAIT instruction. This utilization of the Stop Trap allows the IPU to signal the CPU at milestones without stopping IPU execution. In either use of this stop IPU trap, the old PSD is stored into words 1 and 2 of TCB and the ending IPU status into word 5. The IPU DONE signal is sent to the CPU after storage of the old PSD and IPU status word and before vectoring to the new PSD address.

**CONDITION CODE RESULTS**

No change.

**Assembly Language Coding:** SIPU
The execution of this instruction causes an interrupt request signal to be applied to interrupt priority \( 27_{16} \). Bit positions 6-15 of the Instruction Word may be used to contain program flags which can be examined by the interrupt service routine.

**CONDITION CODE**

- **CC1:** No change
- **CC2:** No change
- **CC3:** No change
- **CC4:** No change

**Assembly Language Coding:** CALM PROGRAM. FLAGS

**NOTES**

1. Interrupt level 27 must be enabled prior to execution of this instruction.
2. This instruction must not be executed with a higher priority level active.
SUPERVISOR CALL

C806

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>IND</th>
<th>CALL NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

DEFINITION

The execution of this instruction causes a pseudo-trap to the trap/interrupt vector for relative priority level 6. Bits 16-19 may be used to index the interrupt vector (location 180) with up to 16 locations. This index vector address will point to a SVC vector table whose content will point to the trap subroutine.

Bits 20-31 are used for the call number. This call number serves as an identifier parameter for the software use.

![Diagram showing secondary vector table and interrupt context blocks]

CONDITION CODE RESULTS

| CC1: | zero |
| CC2: | zero |
| CC3: | zero |
| CC4: | zero |

Assembly Language Coding: SVC  IND, CALL#

NOTE

The CPU must have previously been set to PSD mode. Otherwise, an Undefined Instruction Trap will occur.
**SET CPU MODE**

2C09

<table>
<thead>
<tr>
<th>bits</th>
<th>reserved</th>
<th>mode</th>
<th>reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-13</td>
<td>0000000000000000</td>
<td>000</td>
<td>0000000000000000</td>
</tr>
</tbody>
</table>

The execution of this instruction causes the operating characteristic of the CPU to change to the mode specified by the contents of R.

The contents of R will be:

<table>
<thead>
<tr>
<th>bits</th>
<th>reserved</th>
<th>mode</th>
<th>reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-13</td>
<td>0000000000000000</td>
<td>000</td>
<td>0000000000000000</td>
</tr>
</tbody>
</table>

- **Bits 0-13** Must be zeros and reserved for future use.
- **Bit 14** Enable Block Mode Timeout Trap.
- **Bit 15** Enable PSD Traps (m/c halts if not enabled)
- **Bit 16-18** Reserved (must be zero).
- **Bit 19** 0=PSW mode 55  
1=PSD mode 75

**CONDITION CODE**

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>No change</td>
<td>No change</td>
<td>No change</td>
<td>No change</td>
</tr>
</tbody>
</table>

**Assembly Language Coding:**

`SETCPU S`

**NOTE**

The PSD mode of operation must be enabled (allowed) by way of a hardware jumper on the C Board, or an Undefined Instruction Trap will occur.
DEFINITION

This instruction places the CPU Status Word into Register \( R_D \). The source of the CPU Status Word is location 91H in the CPU Scratchpad. After reporting status, bits 00-23 of the Status Word (in the Scratchpad) are set to zero. Bits 24-31 of the Scratchpad Status Word remain unchanged. The CPU Status Word in Register \( R_D \) is defined as follows:

Bit

0  =0, CLASS 0,1,2, OR E ERROR
     =1, CLASS F (EXTENDED I/O) ERROR
.1  =0, I/O PROCESSING ERROR
     =1, INTERRUPT PROCESSING ERROR
2  FINAL BUS TRANSFER ERROR
3  BUS NO RESPONSE ERROR
4  I/O CHANNEL BUSY OR BUSY STATUS BIT ERROR
5  READY TIMEOUT ERROR
6  I/O DRT TIMEOUT ERROR
7  RETRY COUNT EXHAUSTED ERROR
8  OPERAND FETCH PARITY ERROR
9  INSTRUCTION FETCH PARITY ERROR
10 OPERAND NONPRESENT ERROR
11 INSTRUCTION NONPRESENT ERROR
12 UNDEFINED PSD MODE INSTRUCTION ERROR
13 MEMORY FETCH DRT TIMEOUT ERROR
14 RESET CHANNEL ERROR
15 CHANNEL WCS NOT ENABLED ERROR
16 MAP NOT FOUND
   MAP REGISTER ADDRESS OVERFLOW (MAP CONTEXT SWITCH)
17 UNEXPLAINED MEMORY ERROR
18 BRI I/O ERROR
19 UNDEFINED INSTRUCTION PSD MODE ONLY
20 MAP INVALID ACCESS OR MAP MODE RESTRICTION ERROR
21 IPL I/O OR MEMORY ERROR FLAG
22 CPU WCS NOT PRESENT ERROR
23 NOT USED
24 ENABLE ARITHMETIC EXCEPTION TRAP
25 DISABLE PSD MODE TRAPS
26 BLOCK MODE IS ACTIVE
27 CPU POWER FAIL UP MEMORY ERROR
28 NOT USED
29 NOT USED
30 NOT USED
31 =0, CPU MODE PSW 55
     =1, CPU MODE PSW 75
CONDITION CODE
CC1: Not used
CC2: ISI PSD mode
CC3: ISI interrupts are blocked
CC4: ISI R₀ bits 0-23 equal zero

Assembly Language Coding:  RDSTS R₀

RESULTS

NOTES
1. This instruction is a Privileged Halfword instruction.
2. This instruction may not be the target of an Execute instruction.
3. The PSD mode of operation must be enabled (allowed) by way of a hardware jumper on the C-board, or an undefined instruction trap will occur.
ENABLE ARITHMETIC EXCEPTION TRAP

0008

Sets bit 7 of PSD to enable Arithmetic Exception Trap.

CC1: No change
CC2: No change
CC3: No change
CC4: No change

Assembly Language Coding: EAE

1. Halfword Instruction.
2. Attempt to execute this instruction in PSW mode will result in an Undefined Instruction Trap.
DAE  DISABLE ARITHMETIC EXCEPTION TRAP

000E

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

DEFINITION
Resets bit 7 of PSD to disable Arithmetic Exception Trap.

CONDITION CODE RESULTS
CC1: No change
CC2: No change
CC3: No change
CC4: No change

Assembly Language Coding: DAE

NOTES
1. Halfword Instruction.
2. Attempt to execute this instruction in 55 mode will result in an Undefined Instruction Trap.
INTERRUPT INSTRUCTIONS

GENERAL DESCRIPTION

The Interrupt Control instruction group provides the availability to permit selective Enable, Disable, Request, Activate, and Deactivate operations to be performed on any addressed interrupt level. These instructions can only be executed when bit 0 of the PSWR equals one (Privileged State).

INSTRUCTION FORMATS

The following instruction format is used for all Interrupt Control operations: (Trap/Interrupt priorities are shown in Table 6-3.)

INTERRUPT CONTROL

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>PRIORITY LEVEL</th>
<th>AUG CODE</th>
<th>UNASSIGNED</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5</td>
<td>6 7 8 9 10 11 12 13 14 15</td>
<td>16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
</tr>
</tbody>
</table>

Bits 0-5 define the Operation Code.

Bits 6-12 define the binary priority level number of the interrupt being commanded.


Bits 16-31 unassigned.

CONDITION CODE UTILIZATION

All Interrupt Control instructions leave the current Condition Code unchanged.
Table 6-3. 32/70 Series Relative Trap/Interrupt Priorities

<table>
<thead>
<tr>
<th>INTERRUPT AND TRAP RELATIVE PRIORITY</th>
<th>INTERRUPT LOGICAL PRIORITY</th>
<th>INTERRUPT VECTOR LOCATION (IVL)</th>
<th>TCW ADDRESS **</th>
<th>IOCQ ADDRESS **</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td>0F4</td>
<td></td>
<td></td>
<td>Power Fail Safe Trap</td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>0FC</td>
<td></td>
<td></td>
<td>System Override Trap (Not Used)</td>
</tr>
<tr>
<td>02</td>
<td></td>
<td>0EB*</td>
<td></td>
<td></td>
<td>Memory Parity Trap</td>
</tr>
<tr>
<td>03</td>
<td></td>
<td>190</td>
<td></td>
<td></td>
<td>Nonpresent Memory Trap</td>
</tr>
<tr>
<td>04</td>
<td></td>
<td>194</td>
<td></td>
<td></td>
<td>Undefined Instruction Trap</td>
</tr>
<tr>
<td>05</td>
<td></td>
<td>198</td>
<td></td>
<td></td>
<td>Privilege Violation Trap</td>
</tr>
<tr>
<td>06</td>
<td></td>
<td>180</td>
<td></td>
<td></td>
<td>Supervisor Call Trap</td>
</tr>
<tr>
<td>07</td>
<td></td>
<td>184</td>
<td></td>
<td></td>
<td>Machine Check Trap</td>
</tr>
<tr>
<td>08</td>
<td></td>
<td>188</td>
<td></td>
<td></td>
<td>System Check Trap</td>
</tr>
<tr>
<td>09</td>
<td></td>
<td>18C</td>
<td></td>
<td></td>
<td>Map Fault Trap</td>
</tr>
<tr>
<td>0A</td>
<td></td>
<td>0E4</td>
<td></td>
<td></td>
<td>Not Used</td>
</tr>
<tr>
<td>0B</td>
<td></td>
<td>0F0</td>
<td></td>
<td></td>
<td>Not Used</td>
</tr>
<tr>
<td>0C</td>
<td></td>
<td>0F8</td>
<td></td>
<td></td>
<td>Not Used</td>
</tr>
<tr>
<td>0D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not Used</td>
</tr>
<tr>
<td>0E</td>
<td></td>
<td>0E4</td>
<td></td>
<td></td>
<td>Block Mode Timeout Trap</td>
</tr>
<tr>
<td>0F</td>
<td></td>
<td>1A4*</td>
<td></td>
<td></td>
<td>Arithmetic Exception Trap</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>0F0</td>
<td></td>
<td></td>
<td>Power Fail Safe Interrupt</td>
</tr>
<tr>
<td>11</td>
<td>01</td>
<td>0F8</td>
<td></td>
<td></td>
<td>System Override Interrupt</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>0EB*</td>
<td></td>
<td></td>
<td>***Memory Parity Trap</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>0EC</td>
<td></td>
<td></td>
<td>Attention Interrupt</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>140</td>
<td>100</td>
<td>700</td>
<td>I/O Channel 0 Interrupt</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>144</td>
<td>104</td>
<td>708</td>
<td>I/O Channel 1 Interrupt</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>148</td>
<td>108</td>
<td>710</td>
<td>I/O Channel 2 Interrupt</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>14C</td>
<td>10C</td>
<td>718</td>
<td>I/O Channel 3 Interrupt</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>150</td>
<td>110</td>
<td>720</td>
<td>I/O Channel 4 Interrupt</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>154</td>
<td>114</td>
<td>728</td>
<td>I/O Channel 5 Interrupt</td>
</tr>
<tr>
<td>1A</td>
<td>1A</td>
<td>158</td>
<td>118</td>
<td>730</td>
<td>I/O Channel 6 Interrupt</td>
</tr>
<tr>
<td>1B</td>
<td>1B</td>
<td>15C</td>
<td>11C</td>
<td>738</td>
<td>I/O Channel 7 Interrupt</td>
</tr>
<tr>
<td>1C</td>
<td></td>
<td>160</td>
<td>120</td>
<td>740</td>
<td>I/O Channel 8 Interrupt</td>
</tr>
<tr>
<td>1D</td>
<td></td>
<td>164</td>
<td>124</td>
<td>748</td>
<td>I/O Channel 9 Interrupt</td>
</tr>
<tr>
<td>1E</td>
<td></td>
<td>168</td>
<td>128</td>
<td>750</td>
<td>I/O Channel A Interrupt</td>
</tr>
<tr>
<td>1F</td>
<td></td>
<td>16C</td>
<td>12C</td>
<td>758</td>
<td>I/O Channel B Interrupt</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>170</td>
<td>130</td>
<td>760</td>
<td>I/O Channel C Interrupt</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>174</td>
<td>134</td>
<td>768</td>
<td>I/O Channel D Interrupt</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>178</td>
<td>138</td>
<td>770</td>
<td>I/O Channel E Interrupt</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>17C</td>
<td>13C</td>
<td>778</td>
<td>I/O Channel F Interrupt</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>190*</td>
<td></td>
<td></td>
<td>***Nonpresent Memory Trap</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>194*</td>
<td></td>
<td></td>
<td>***Undefined Instruction Trap</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>198*</td>
<td></td>
<td></td>
<td>***Privilege Violation Trap</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>19C</td>
<td></td>
<td></td>
<td>Call Monitor Interrupt</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td>1A0</td>
<td></td>
<td></td>
<td>Real-Time Clock Interrupt</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td>1A4*</td>
<td></td>
<td></td>
<td>***Arithmetic Exception Interrupt</td>
</tr>
<tr>
<td>2A</td>
<td></td>
<td>1A8</td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>2B</td>
<td></td>
<td>1AC</td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>2C</td>
<td></td>
<td>180</td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
</tbody>
</table>
### Table 6-3. 32/70 Series Relative Trap/Interrupt Priorities (Cont'd)

<table>
<thead>
<tr>
<th>INTERRUPT AND TRAP RELATIVE PRIORITY</th>
<th>INTERRUPT LOGICAL PRIORITY</th>
<th>INTERRUPT VECTOR LOCATION (IVL)</th>
<th>TCW ADDRESS **</th>
<th>IOC ADDRESS **</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D</td>
<td>2D</td>
<td>1B4</td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>2E</td>
<td>2E</td>
<td>1BB</td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>2F</td>
<td>2F</td>
<td>1BC</td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>1CO</td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>1C4</td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>THROUGH</td>
<td>THROUGH</td>
<td>THROUGH</td>
<td></td>
<td></td>
<td>THROUGH</td>
</tr>
<tr>
<td>77</td>
<td>77</td>
<td>2DC</td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>78</td>
<td>78</td>
<td>2ED****</td>
<td></td>
<td></td>
<td>Ending of IPU Processing Trap (Used by CPU)</td>
</tr>
<tr>
<td>79</td>
<td>79</td>
<td>2E4****</td>
<td></td>
<td></td>
<td>Start IPU Processing Trap (Used by IPU)</td>
</tr>
<tr>
<td>7A</td>
<td>7A</td>
<td>2EB****</td>
<td></td>
<td></td>
<td>Supervisor Call Trap (Used by IPU)</td>
</tr>
<tr>
<td>7B</td>
<td>7B</td>
<td>2EC****</td>
<td></td>
<td></td>
<td>Error Trap (Used by IPU)</td>
</tr>
<tr>
<td>7C</td>
<td>7C</td>
<td>2F0****</td>
<td></td>
<td></td>
<td>Call Monitor Trap (Used by IPU)</td>
</tr>
<tr>
<td>7D</td>
<td>7D</td>
<td>2F4****</td>
<td></td>
<td></td>
<td>Stop IPU Processing Trap (Used by IPU)</td>
</tr>
<tr>
<td>7E</td>
<td>7E</td>
<td>2FB</td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
<tr>
<td>7F</td>
<td>7F</td>
<td>2FC</td>
<td></td>
<td></td>
<td>External/Software Interrupts</td>
</tr>
</tbody>
</table>

* Vector Locations Shared With Traps
** For Nonextended I/O Devices
*** PSW Function - Now External/Software Interrupts - For PSD Mode.
**** IPU Related Traps (See Section II)

All Interrupts Are Externally Generated
**DEFINITION**

If bit position 0 of the PSWR is equal to one (Privileged State), the priority interrupt level specified by the priority level field (bits 6-12) in the Instruction Word (IW) is conditioned to respond to an interrupt signal. If bit position 0 of the PSWR is equal to zero (Unprivileged State), execution of this instruction will generate the Privileged Violation trap.

**NOTES**

1. This instruction does not operate with priority levels $2_{16} - 11_{16}$.
2. Any stored requests for the specified level are eligible to become active.
3. In the PSD mode, traps are always enabled.
4. This instruction has no affect on levels assigned to Class F I/O and is treated as NOP.
5. For levels 0 and 1, the RTOM jumpers provide either constant enable or software enable/disable.

<table>
<thead>
<tr>
<th>INSTRUCTION</th>
<th>Bits 6 through 12</th>
<th>Priority Level (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIORITY</td>
<td>0010010</td>
<td>12</td>
</tr>
<tr>
<td>LEVEL FIELD</td>
<td>0010011</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>0010100</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1111110</td>
<td>7E</td>
</tr>
<tr>
<td></td>
<td>1111111</td>
<td>7F</td>
</tr>
</tbody>
</table>

**CONDITION CODE RESULTS**

- CC1: No change
- CC2: No change
- CC3: No change
- CC4: No change

**ASSEMBLY LANGUAGE CODING**

- EI , LEVEL

**NOTE**

Any stored requests for the specified level are eligible to become active.
REQUEST INTERRUPT

FC02

<table>
<thead>
<tr>
<th>PRIORITY LEVEL</th>
<th>AUG CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 1 1</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

**DEFINITION**

If bit position 0 of the PSWR is equal to one (Privileged State), an interrupt request signal is applied to the interrupt level specified by the priority level field (bits 6-12) in the Instruction Word (IW). This signal simulates the signal generated by the internal or external condition connected to the specified level. If bit position 0 of the PSWR is equal to zero (Unprivileged State), execution of this instruction will generate the Privileged Violation Trap. The interrupt request signal is stored in the specified level whether or not it is enabled and/or active.

**NOTES**

1. This instruction does not operate with priority levels \(2_{16} - 1_{16}\).
2. For RI's on levels 0 or 1, the RTOM jumpers select either that levels 0 and 1 are enabled, or that software enables are required.
3. This instruction has no affect on levels assigned to Class F I/O and is treated as NOP.

**INSTRUCTION PRIORITY LEVEL FIELD**

<table>
<thead>
<tr>
<th>Bits 6 through 12</th>
<th>Priority Level (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
<td>00</td>
</tr>
<tr>
<td>0000001</td>
<td>01</td>
</tr>
<tr>
<td>0010010</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
<tr>
<td>1111110</td>
<td>7E</td>
</tr>
<tr>
<td>1111111</td>
<td>7F</td>
</tr>
</tbody>
</table>

**CONDITION CODE RESULTS**

CC1: No change
CC2: No change
CC3: No change
CC4: No change

**ASSEMBLY LANGUAGE CODING**

RI LEVEL
DEFINITION
If bit position 0 of the PSWR is equal to one (Privileged State), a signal is applied to set the active condition in the priority interrupt level specified by the priority level field (bits 6-12) in the Instruction Word (IW). The active level is set in the specified level whether or not that level is enabled. This condition prohibits this level and any lower levels not already in service from being serviced until this level is deactivated. However, request signals occurring at this or lower levels are stored for subsequent servicing. If bit position 0 of the PSWR is equal to zero (Unprivileged State), execution of this instruction will generate the Privileged Violation Trap.

NOTES
1. This instruction does not operate with priority levels 2₁₆ - 11₁₆.
2. This instruction has no affect on levels assigned to Class F I/O and is treated as NOP.

<table>
<thead>
<tr>
<th>INSTRUCTION PRIOIITY LEVEL FIELD</th>
<th>Bits 6 through 12</th>
<th>Priority Level (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>0000001</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>0010010</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1111110</td>
<td>7E</td>
<td></td>
</tr>
<tr>
<td>1111111</td>
<td>7F</td>
<td></td>
</tr>
</tbody>
</table>

CONDITION CODE RESULTS
CC1: No change
CC2: No change
CC3: No change
CC4: No change

ASSEMBLY LANGUAGE CODING
AI LEVEL
DISABLE INTERRUPT

If bit position 0 of the PSWR is equal to one (Privileged State), the priority interrupt level specified by the priority level field (bits 6-12) in the Instruction Word (IW) is disabled and will not respond to an interrupt signal. If bit position 0 of the PSWR is equal to zero (Unprivileged State), execution of this instruction will generate the Privileged Violation Trap. The active state of the interrupt is not affected.

NOTES

1. Any unserviced request signal at this level is cleared by execution of this instruction.
2. This instruction does not operate with priority levels \( 2_{16} - 11_{16} \).
3. In the PSD mode, traps are always enabled.
4. This instruction has no affect on levels assigned to Class F I/O and is treated as NOP.
5. For levels 0 and 1, the RTOM jumpers provide either constant enable or software enable/disable.

<table>
<thead>
<tr>
<th>INSTRUCTION</th>
<th>Bits 6 through 12</th>
<th>Priority Level (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIORITY LEVEL FIELD</td>
<td>0010010</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>0010011</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>0010100</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1111110</td>
<td>7E</td>
</tr>
<tr>
<td></td>
<td>1111111</td>
<td>7F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONDITION CODE RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC1: No change</td>
</tr>
<tr>
<td>CC2: No change</td>
</tr>
<tr>
<td>CC3: No change</td>
</tr>
<tr>
<td>CC4: No change</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ASSEMBLY LANGUAGE CODING</th>
</tr>
</thead>
<tbody>
<tr>
<td>DI LEVEL</td>
</tr>
</tbody>
</table>
DEACTIVATE INTERRUPT

FC04

If bit position 0 of the PSWR is equal to one (Privileged State), a signal is applied to reset the active condition for the priority interrupt level specified by the priority level field (bits 6-12) in the Instruction Word. The specified level is set inactive whether the level is enabled or disabled. Execution of the Deactivate Interrupt instruction does not clear any request signals on the specified level or any other level. If bit position 0 of the PSWR is equal to zero (Unprivileged State), execution of this instruction will generate the Privileged Violation Trap.

NOTE
1. This instruction does not operate with priority levels \(2^{16} - 11_{16}\).
2. This instruction has no affect on levels assigned to Class F I/O and is treated as a NOP.
3. In PSD mode, DAI and the following instruction are executed as an uninterruptible pair.
4. Using a Deactivate Interrupt and then LPSD (Load Program Status Doubleword) or a Deactivate Interrupt and then LPSCDM, is preferable to using a BRI (faster)

<table>
<thead>
<tr>
<th>Bits 6 through 12</th>
<th>Priority Level (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
<td>00</td>
</tr>
<tr>
<td>0000001</td>
<td>01</td>
</tr>
<tr>
<td>0010010</td>
<td>12</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1111110</td>
<td>7E</td>
</tr>
<tr>
<td>1111111</td>
<td>7F</td>
</tr>
</tbody>
</table>

CONDITION CODE RESULTS
CC1: No change
CC2: No change
CC3: No change
CC4: No change

ASSEMBLY LANGUAGE CODING
DAI LEVEL
ACTIVATE CHANNEL INTERRUPT

FC77

<table>
<thead>
<tr>
<th>OPCODE</th>
<th>R</th>
<th>AUG CODE</th>
<th>CHANNEL</th>
<th>SUBADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 1 1 1 1 1 1 1 0</td>
<td>1 1 0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

DEFINITION

The Activate Channel Interrupt will cause the addressed channel to begin actively contending with other interrupt levels, causing a blocking of its level, and all lower priority levels, from requesting an interrupt. If a request is currently pending in the channel, the request interrupt is removed but the interrupt level remains in contention.

- Bits 0-5 specify the operation code, octal 77.
- Bits 6-8 specify the General Purpose Register, when nonzero, whose contents will be added to the channel and subaddress field to form the logical channel and subaddress.
- Bits 9-12 specify the operation as an ACI, hex E.
- Bits 13-15 specify the augment code, octal 7.
- Bits 16-31 specify a constant that will be added to the contents of R to form the logical channel and subaddress. If R is zero only, constant will be used to specify the logical channel and subaddress.

CONDITION CODE

CCI, 2, 3, and 4 = (0000)₂ or (1000)₂

This indicates that the instruction was accepted. For other Condition Code combinations refer to the Class F Condition Codes on page 6-214 of this manual.

ASSEMBLY LANGUAGE CODING

ACI  R,'(Constant)'

NOTES

1. Condition Codes, after execution of the ACI, will be set and can be tested by a subsequent BCT or BCF to determine if the ACI was accepted by the channel.

2. If this instruction is executed for a Non-Class F channel, an Undefined Instruction Trap will occur.
**ECI**

**ENABLE CHANNEL INTERRUPT**

FC67

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R</th>
<th>ECI</th>
<th>AUG CODE</th>
<th>CHANNEL</th>
<th>SUBADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 1</td>
<td>1 1 0 0</td>
<td>1 1 1 1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**DEFINITION**

The Enable Channel Interrupt causes the addressed channel to be enabled to request interrupts from the CPU.

- **Bits 0-5** specify the operation code, octal 77.
- **Bits 6-8** specify the General Purpose Register, when nonzero, whose contents will be added to the channel and subaddress field to form the logical channel and subaddress.
- **Bits 9-12** specify the operation as ECI, hex C.
- **Bits 13-15** specify the augment code, octal 7.
- **Bits 16-31** specify a constant that will be added to the contents of R to form the logical channel and subaddress. If R is zero only constant will be used to specify the logical channel and subaddress.

**CONDITION CODE RESULTS**

CCI, 2, 3, and 4 = (0000)\(_2\) or (1000)\(_2\)

This indicates that the instruction was accepted. For other Condition Code combinations, refer to the Class F Condition Codes on Page 6-214 of this manual.

**ASSEMBLY LANGUAGE CODING**

ECI R,'(Constant)'

**NOTES**

1. Condition Codes after execution of the ECI will be set and can be tested by a subsequent BCT or BCF to determine if the ECI was accepted by the channel.

2. If this instruction is executed for a Non-Class F channel, an Undefined Instruction Trap will occur.
DISABLE CHANNEL INTERRUPTS

FC6F

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R</th>
<th>DCI</th>
<th>AUG CODE</th>
<th>CHANNEL</th>
<th>SUBADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1,1,1,1</td>
<td>1,1,0,1,1</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

DESCRIPTION
The Disable Channel Interrupt causes the addressed channel to be disabled from requesting interrupts from the CPU.

Bits 0-5 specify the operation code, octal 77.

Bits 6-8 specify the General Purpose Register, when nonzero, whose contents will be added to the channel and subaddress field to form the logical channel and subaddress.

Bits 9-12 specify the operation as DCI, hex D.

Bits 13-15 specify the augment code, octal 7.

Bits 16-31 specify a constant that will be added to the contents of R to form the logical channel and subaddress. If R is zero, only constant will be used to specify the logical channel and subaddress.

CONDITION CODE RESULTS
CCI, 2, 3, and 4 = (0000)2 or (1000)2

This indicates that the instruction was accepted. For other Condition Code combinations refer to the Class F Condition Codes on page 6-214 of this manual.

ASSEMBLY LANGUAGE CODING
DCI  R,'(Constant)'

NOTES
1. Condition Codes after execution of the DCI will be set and can be tested by a subsequent BCT or BCF to determine if the DCI was accepted by the channel.

2. If this instruction is executed for a Non-Class F channel, an Undefined Instruction Trap will occur.
DEACTIVATE CHANNEL INTERRUPT

**FC7F**

**OP CODE**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-5</td>
<td>Specify the operation code, octal 77.</td>
</tr>
<tr>
<td>6-8</td>
<td>Specify the General Purpose Register, when nonzero, whose contents will be added to the channel and subaddress fields to form the logical channel and subaddress.</td>
</tr>
<tr>
<td>9-12</td>
<td>Specify the operation as DACI, hex F.</td>
</tr>
<tr>
<td>13-15</td>
<td>Specify the augment code, octal 7.</td>
</tr>
<tr>
<td>16-31</td>
<td>Specify a constant that will be added to the contents of R to form the logical channel and subaddress. If R is zero, only constant will be used to specify the logical channel and subaddress.</td>
</tr>
</tbody>
</table>

** DEFINITION**
The Deactivate Channel Interrupt will cause the addressed channel to remove its interrupt level from contention. If a request interrupt is currently queued, the deactivate will cause the queued request to actively request if the channel is enabled.

**CONDITION CODE**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1, 2, 3, and 4 = (0000)₂ or (1000)₂</td>
<td></td>
</tr>
</tbody>
</table>

This indicates that the instruction was accepted. For other Condition Code combinations, refer to the Class F Condition Codes on page 6-214 of this manual.

**ASSEMBLY LANGUAGE CODING**

DACI R,'(Constant)'

**NOTES**

1. Condition Codes after execution of the DACI will be set and can be tested by a subsequent BCT or BDF to determine if the DACI was successfully executed.

2. On PSD mode, the DACI and following instructions are executed as an uninterruptible pair.

3. Using Deactivate Channel Interrupt and LPSD or Deactivate Channel Interrupt and LPSDCM is preferable to using a BRI.

4. If this instruction is executed for a Non-Class F channel, an Undefined Instruction Trap will occur.

5. All DACI instruction abnormalities or I/O protocol violations are connected to the System Check Trap unless an initial channel nonpresent or inoperable condition is found.
The execution of this instruction prevents the CPU from sensing all interrupt requests generated by the I/O channel and RTOM.

CC1: No change
CC2: No change
CC3: No change
CC4: No change

The CPU must have previously been set to PSD mode.
**UEI**

**UNBLOCK EXTERNAL INTERRUPTS**

0007

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
</tr>
<tr>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**DEFINITION**

The execution of this instruction causes the CPU to sense all interrupt requests generated by the I/O channel and RTOM.

**CONDITION CODE**

**RESULTS**

CC1: No change  
CC2: No change  
CC3: No change  
CC4: No change

**ASSEMBLY LANGUAGE CODING**

UEI

**NOTE**

The CPU must have previously been set to PSD mode.
The Input/Output instructions provide the capability to perform Command or Test operations to attached peripheral devices. Both the Command Device and the Test Device instructions cause a 16-bit function code to be sent to the device specified by the instruction.

The following instruction format is used by both Input/Output instructions.

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>DEVICE NO</th>
<th>AUG CODE</th>
<th>FUNCTION CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Bits 0-5 define the Operation Code.

Bits 6-12 designate the device number.


Bits 16-31 contain the 16-bit function code.

The Condition Code is set during execution of a Test Device instruction to indicate the result of the test being performed. The Command Device instruction leaves the current Condition Code unchanged.
All Class F I/O instructions will be in the following format:

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R</th>
<th>SUB OP</th>
<th>AUG CODE</th>
<th>CHANNEL</th>
<th>SUBADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Op Code bits 0-5 and Aug Code bits 13-15 must contain ones. The R field (bits 6-8), if nonzero, specifies the general register whose contents will be added to the channel and subaddress field bits 16-31 to form the logical channel and subaddress. If R is specified as zero, only the channel and subaddress fields will be used. The format of the computed logical channel and subaddress is:

<table>
<thead>
<tr>
<th>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0</th>
<th>LOGICAL CHANNEL</th>
<th>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The subaddress will be ignored by the channel if the operation does not apply to a controller or device.

The sub op field bits 09-12 specify the type of operation that is to be performed as described below:

**BITS 09-12**

<table>
<thead>
<tr>
<th>SUB OP</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 - X'0'</td>
<td>Unassigned</td>
</tr>
<tr>
<td>0 0 0 1 - X'1'</td>
<td>Unassigned</td>
</tr>
<tr>
<td>0 0 1 0 - X'2'</td>
<td>START I/O (SIO)</td>
</tr>
<tr>
<td>0 0 1 1 - X'3'</td>
<td>TEST I/O (TIO)</td>
</tr>
<tr>
<td>0 1 0 0 - X'4'</td>
<td>STOP I/O (STPIO)</td>
</tr>
<tr>
<td>0 1 0 1 - X'5'</td>
<td>RESET CHANNEL (RSCHNL)</td>
</tr>
<tr>
<td>0 1 1 0 - X'6'</td>
<td>HALT I/O (HIO)</td>
</tr>
<tr>
<td>0 1 1 1 - X'7'</td>
<td>GRAB CONTROLLER (GRI0)</td>
</tr>
<tr>
<td>1 0 0 0 - X'8'</td>
<td>RESET CONTROLLER (RSCTL)</td>
</tr>
<tr>
<td>1 0 0 1 - X'9'</td>
<td>ENABLE WRITE CHANNEL WCS (ECWCS)</td>
</tr>
<tr>
<td>1 0 1 0 - X'A'</td>
<td>Unassigned</td>
</tr>
<tr>
<td>1 0 1 1 - X'B'</td>
<td>WRITE CHANNEL WCS (WCWCS)</td>
</tr>
<tr>
<td>1 1 0 0 - X'C'</td>
<td>ENABLE CHANNEL INTERRUPT (ECI)</td>
</tr>
<tr>
<td>1 1 0 1 - X'D'</td>
<td>DISABLE CHANNEL INTERRUPT (DCI)</td>
</tr>
<tr>
<td>1 1 1 0 - X'E'</td>
<td>ACTIVATE CHANNEL INTERRUPT (ACI)</td>
</tr>
<tr>
<td>1 1 1 1 - X'F'</td>
<td>DEACTIVATE CHANNEL INTERRUPT (DACI)</td>
</tr>
</tbody>
</table>

**NOTES**

1. Channel must be ICL'd as Class F.
2. EXR, EXRR, and EXM may not be used.
3. Must be in PSD mode.
4. CCs must be tested after each instruction.
5. CD, TD, EI, DI, AI, DAI, and RI cannot be executed to Class F channel.
The condition codes will be set for the execution of all Class F I/O instructions and indicate the successful or unsuccessful initiation of an I/O instruction. The condition codes can be set by the CPU, for channel busy and inoperable or undefined channel, or by the information passed directly from the channel. The assignments for the condition codes are:

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>REQUEST ACTIVATED, WILL ECHO STATUS</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>CHANNEL BUSY</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>CHANNEL INOPERABLE OR UNDEFINED</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>SUBCHANNEL BUSY</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>STATUS STORED</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>UNSUPPORTED TRANSACTION</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>UNASSIGNED</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>UNASSIGNED</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>REQUEST ACCEPTED AND QUEUED, NO ECHO STATUS</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>UNASSIGNED</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>UNASSIGNED</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>UNASSIGNED</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>UNASSIGNED</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>UNASSIGNED</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>UNASSIGNED</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>UNASSIGNED</td>
</tr>
</tbody>
</table>

Although 16 encoded conditions are possible, only the assigned patterns will occur.
CD
n,f

COMMAND DEVICE

FC06

<table>
<thead>
<tr>
<th>1 1 1 1 1</th>
<th>DEVICE ADDRESS</th>
<th>1 1 0</th>
<th>COMMAND CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DEFINITION**

The contents of the Command Code field (bits 16-31) are transferred to the Device Controller Channel specified by the device address contained in bit positions 6-12 of the Instruction Word.

**CONDITION CODE**

<table>
<thead>
<tr>
<th>RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC1: No change</td>
</tr>
<tr>
<td>CC2: No change</td>
</tr>
<tr>
<td>CC3: No change</td>
</tr>
<tr>
<td>CC4: No change</td>
</tr>
</tbody>
</table>

**ASSEMBLY EXAMPLE**

<table>
<thead>
<tr>
<th>Dev</th>
<th>Comm</th>
<th>Add</th>
<th>Code</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD</td>
<td>X'7A', X'8000'</td>
<td>Output data to device 7A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD</td>
<td>X'78', X'9000'</td>
<td>Input data from device 78</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTES**

1. Class 0,1,2,3, and E I/O Processor instruction only.

2. If the CPU is in the PSW mode and a CD instruction to a Class F channel is attempted, a No Operation (NOP) will be executed instead.

3. If the CPU is in the PSD mode and a CD instruction to a Class F channel is attempted, a System Check Trap will occur.
TEST DEVICE

FC05

<table>
<thead>
<tr>
<th>1 1 1 1 1</th>
<th>DEVICE ADDRESS</th>
<th>1 0 1</th>
<th>TEST CODE</th>
<th>0 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4</td>
<td>5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DEFINITION

The contents of the Test Code field (bits 16-27) are transferred to the Device Controller Channel (DCC) specified by the device address contained in bit positions 6-12 of the Instruction Word. The device test defined by the Test Code is performed in the DCC, and the test results are stored in Condition Code bits 1-4 (CC1-4).

NOTE

A TD having a unique Test Code is available with most peripheral devices. Execution of a TD with this code causes a snapshot of all device and DCC status to be stored in memory. The individual peripheral device reference manuals define the operation of this instruction with each device.

CONDITION CODE RESULTS

Test results defined for specific peripheral device.

ASSEMBLY EXAMPLE

Dev  Comm  Add  Code  Command
TD  X'10',X'8000'  Request the Controller Status for unit 10
TD  X'10',X'2000'  Request the Device status for unit 10

NOTES

1. Class 0,1,2,3, and E I/O Processor instruction only.
2. If the CPU is in the PSW mode and a TD instruction to a Class F channel is attempted, the following Condition Codes will be set:
   a. TD 8000 - CC3 (Channel Error)
   b. TD 4000 - CC3 (Program Violation)
   c. TD 2000 - CC2 (Status Transfer Not Performed)
3. If the CPU is in the PSD mode and a TD instruction to a Class F channel is attempted, a System Check Trap will occur.
START I/O

FC17

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R</th>
<th>SIO CODE</th>
<th>AUG CODE</th>
<th>CHANNEL</th>
<th>SUBADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1</td>
<td>0 0 1 0</td>
<td>1 1 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DEFINITION**

Start I/O will be used to begin I/O execution or to return appropriate Condition Codes and status if I/O execution could not be accomplished.

- **Bits 0-5**: specify the operation code, octal 77.
- **Bits 6-8**: specify the General Purpose Register, when nonzero, whose contents will be added to the channel and subaddress fields to form the logical channel and subaddress.
- **Bits 9-12**: specify the operation as an SIO, hex 2.
- **Bits 13-15**: specify the augment code, octal 7.
- **Bits 16-31**: specify a constant that will be added to the contents of R to form the logical channel and subaddress. If R is zero, only bits 16-31 will be used to specify the logical channel and subaddress.

**CONDITION CODE RESULTS**

CC1, 2, 3, and 4 = (0000₂) or (1000₂)

This indicates that the instruction was accepted. For other Condition Code combinations refer to the Class F Condition Codes on page 6-214 of this manual.

**ASSEMBLY LANGUAGE CODING**

SIO R,'(Constant)'

**NOTES**

1. Condition Codes, after execution of an SIO, will be set and can be tested by a subsequent BCT or BCF to ascertain if the I/O was accepted.

2. If this instruction is executed for a Non-Class F channel, an Undefined Instruction Trap will occur.
**TEST I/O**

FCIF

**DEFINITION**

Test I/O will be used to test controller state and to return appropriate Condition Codes and status reflecting the state of the addressed controller and/or device. Channel implementation will dictate the depth that the channel must test to determine current state.

- **Bits 0-5** specify the operation code, octal 77.
- **Bits 6-8** specify the General Purpose Register, when nonzero, whose contents will be added to the channel and subaddress fields to form the logical channel and subaddress.
- **Bits 9-12** specify the operation as a TIO, hex 3.
- **Bits 13-15** specify the augment code, octal 7.
- **Bits 16-31** specify a constant that will be added to the contents of R to form the logical channel and subaddress. If R is zero, only bits 16-31 will be used to specify the logical channel and subaddress.

**CONDITION CODE RESULTS**

CC1, 2, 3, and 4 = (0000)₂ or (1000)₂

This indicates that the instruction was accepted. For other Condition Code combinations refer to the Class F Condition Codes on Page 6-214 of this manual.

**ASSEMBLY LANGUAGE CODING**

TIO R,'(Constant)'

**NOTES**

1. Condition Codes, after execution of the TIO, will be set and can be tested by a subsequent BCT or BCF to ascertain channel/controller/device state.

2. If this instruction is executed for a Non-Class F channel, an Undefined Instruction Trap will occur.
The STOP I/O (STPIO) is used to terminate the current I/O operation after the completion of the current IOCD. The STOP I/O applies only to the addressed subchannel, and the only function is to suppress command and data chain flags in the current IOCD.

- **Bits 0-5**: specify the operation code, octal 77.
- **Bits 6-8**: specify the General Purpose Register, when nonzero, whose contents will be added to the channel and subaddress fields to form the logical channel and subaddress.
- **Bits 9-12**: specify the operation as a STPIO, hex 4.
- **Bits 13-15**: specify the augment code, octal 7.
- **Bits 16-31**: specify a constant that will be added to the contents of R to form the logical channel and subaddress. If R is zero, only bits 16-31 will be used to specify the logical channel and subaddress.

**Condition Code Results**

CC1, 2, 3, and 4 = (0000)₂ or (1000)₂

This indicates that the instruction was accepted. For other Condition Code combinations refer to the Class F Condition Codes on page 6-214 of this manual.

**Assembly Language Coding**

STPIO R,'(Constant)'

**Notes**

1. Condition Codes, after execution of an STPIO, will be set and can be tested by a subsequent BCT or BCF to ascertain the channel/controller/device state.
2. If this instruction is executed for a Non-Class F channel, an Undefined Instruction Trap will occur.
The Reset Channel (RSCHNL) causes the addressed channel to cease and reset all activity and to return to the idle state. The channel will also reset all subchannels. No controller or device will be affected. Any requesting or active interrupt level will be reset.

Bits 0-5 specify the operation code, octal 77.

Bits 6-8 specify the General Purpose Register, when nonzero, whose contents will be added to the channel and subaddress fields to form the logical channel and subaddress.

Bits 9-12 specify the operation as a RSCHNL, hex 5.

Bits 13-15 specify the augment code, octal 7.

Bits 16-31 specify a constant that will be added to the contents of R to form the logical channel and subaddress. If R is zero, only bits 16-31 will be used to specify the logical channel and subaddress.

CC1, 2, 3, and 4 = (0000)2 or (1000)2

This indicates that the instruction was accepted. For other Condition Code combinations refer to the Class F Condition Codes on page 6-214 of this manual.

NOTES

1. Condition Codes, after execution of a RSCHNL, will be set and can be tested by a subsequent BCT or BCF to ascertain the channel/controller/device state.

2. If this instruction is executed for a Non-Class F channel, an Undefined Instruction Trap will occur.
**Halt I/O (HIO)** is used to cause an immediate but orderly termination in the controller. The Device End condition will notify the software of the actual termination in the controller; thus, indicating its availability for new requests. If the Halt I/O caused the generation of status relating to the terminated I/O operation, then the Device End condition for the termination of the I/O operation will be the only Device End condition generated.

**Bits 0-5** specify the operation code, octal 77.

**Bits 6-8** specify the General Purpose Register, when nonzero, whose contents will be added to the channel and subaddress fields to form the logical channel and subaddress.

**Bits 9-12** specify the operation as a HIO, hex 6.

**Bits 13-15** specify the augment code, octal 7.

**Bits 16-31** specify a constant that will be added to the contents of R to form the logical channel and subaddress. If R is zero, only bits 16-31 will be used to specify the logical channel and subaddress.

**Condition Code Results**

CC1, 2, 3, and 4 = \((0000)_2\) or \((1000)_2\)

This indicates that the instruction was accepted. For other Condition Code combinations refer to the Class F Condition Codes on page 6-214 of this manual.

**Assembly Language Coding**

```
HIO R,'(Constant)'
```

1. Condition Codes after execution of the HIO, will be set and be tested by a subsequent BCT or BCF to ascertain if the HIO was successfully executed.

2. If this instruction is executed for a Non-Class F channel, an Undefined Instruction Trap will occur.
The Grab Controller (GRIO) will cause the addressed controller to release itself from the currently assigned channel and to reserve itself for the grabbing channel.

**Notes**

1. Condition Codes, after execution of the GRIO, will be set and can be tested by a subsequent BCT or BCF to determine if the GRIO was successfully executed.

2. If this instruction is executed for a Non-Class F channel, an Undefined Instruction Trap will occur.
## RSCTL

### RESET CONTROLLER

FC47

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>RSCTL</th>
<th>AUG CODE</th>
<th>CHANNEL</th>
<th>SUBADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>01111111</td>
<td>$1$</td>
<td>$0000$</td>
<td>$0000$</td>
<td>$0000$</td>
<td>$0000$</td>
</tr>
</tbody>
</table>

### DEFINITION

This instruction causes the addressed controller to be completely reset. In addition, the subchannel and all pending and generated status conditions are cleared.

- **Bits 0-5**: specify the operation code, octal 77.
- **Bits 6-8**: specify the General Purpose Register (R), when nonzero, whose contents will be added to the channel and subaddress fields to form the logical channel and subaddress.
- **Bits 9-12**: specify the operation as RSCTL, hex 8.
- **Bits 13-15**: specify the augment code, octal 7.
- **Bits 16-31**: specifies a constant that will be added to the contents of R to form the logical channel and subaddress. If R is zero, only bits 16-31 will be used to specify the logical channel and subaddress.

### CONDITION CODE RESULTS

CC1, 2, 3, and 4 = (000)$_2$ or (1000)$_2$

This indicates that the instruction was accepted. For other Condition Code combinations refer to the Class F Condition Codes on page 6-214 of this manual.

### ASSEMBLY LANGUAGE CODING

RSCTL R,'(Constant)'

### NOTE

If this instruction is executed for a Non-Class F channel, an Undefined Instruction Trap will occur.
ENABLE CHANNEL WCS LOAD

The Enable Channel WCS Load (ECWCS) sets an interlock within the CPU to enable the loading of WCS. The ECWCS must be the first of a 2-instruction sequence.

1. Condition Codes after the execution of the ECWCS instruction will be set and can be tested by a subsequent BCT or BCF to ascertain whether the ECWCS instruction was successfully executed.

2. If this instruction is executed for a Non-Class F channel, an Undefined Instruction Trap will occur.
**WRITE CHANNEL WCS**

**FC5F**

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>R</th>
<th>WCWCS</th>
<th>AUG CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1,1,1,1,1</td>
<td>1,0,1,1</td>
<td>1,1,1</td>
<td>0,0,0,0,0,0</td>
</tr>
</tbody>
</table>

**DEFINITION**

The Write Channel WCS (WCWCS) causes the loading of the channel WCS. The WCWCS must be the second instruction executed to the Class F I/O controller, the first being ECWCS, without any intervening I/O instructions to the Class F I/O controller to be loaded.

- **Bits 0-5**: Specify the operation code, octal 77.
- **Bits 6-8**: Specify the general register, when nonzero, whose contents will be added to the channel and subaddress fields to form the logical channel and subaddress.
- **Bits 9-12**: Specify the operation as a WCWCS, hex B.
- **Bits 13-15**: Specify the augment code, octal 7.
- **Bits 16-31**: Specify a constant that will be added to the contents of R to form the logical channel and subaddress. If R is zero, only bits 16-31 will be used to specify the logical channel and subaddress.

**CONDITION CODE RESULTS**

CC1, 2, 3, and 4 = (0000)₂ or (1000)₂

This indicates that the instruction was accepted. For other Condition Code combinations refer to the Class F Condition Codes on page 6-214 of this manual.

**ASSEMBLY LANGUAGE CODING**

WCWCS R,'(Constant)'

**NOTES**

1. The information that is required by the WCS load will be passed to the Class F I/O controller by a parameter list. The IOCD address location specified for this controller will be initialized by software prior to the execution of this instruction. The subaddress field will be ignored.

2. If this instruction is executed for a Non-Class F channel, an Undefined Instruction Trap will occur.

3. If the WCWCS instruction is not preceded by an ECWCS instruction, a System Check Trap will occur.
Real Data Address: Bits 8-31 (MSW) will contain the address of the memory location for the first word to be loaded.

Start WCS Address: Bits 0-15 (LSW) will contain the address of WCS where the first word is to be loaded.

Byte Count: Bits 16-31 (LSW) will contain the number of bytes to be loaded.
SECTION VII
CONTROL PANEL

INTRODUCTION
This section describes the function and operation of the Serial System Control Panel of the 32/70 Series Computer. Figure 7-1 shows the controls, keyboards, and displays of the Serial System Control Panel.

PANEL LOCK
The PANEL LOCK switch is a two-position rotary key switch having an unlocked and locked position. The turnkey can be removed in either position. When the switch is in the unlocked position, all panel keys on the Serial System Control Panel are operational. In the locked position, all panel keys are disabled except for the ATTENTION key and those panel keys for write/read of control switches on the Hexadecimal Keyboard and the Function Keyboard which remain operational at all times.

POWER
The POWER switch is a two-position latching pushbutton which provides the capability to power the system on or off. The state of the power is determined by the RUN and HALT indicators. When the power is on, either the RUN or HALT indicator is on. When the power is off, all indicators on the panel will be off.

RUN/HALT
Depressing the RUN/HALT key while the CPU is in the Halt mode causes the CPU to enter the Run mode and begin executing instructions from the location specified in the Program Status Word.

Depressing the RUN/HALT key while the CPU is in the Run mode causes the CPU to enter the Halt mode. In the Halt mode, the CPU no longer executes instructions from memory; instead, it is placed in a microroutine which monitors selected panel support functions.

SYSTEM RESET
Depressing the SYSTEM RESET key when the system is in the Halt mode initializes all appropriate logic in all SelBUS devices.

ATTENTION
Depressing the ATTENTION key causes an interrupt to occur at the Attention interrupt level, priority level 1316.

INITIAL PROGRAM LOAD
Depressing the INITIAL PROGRAM LOAD key when the CPU is in the Halt mode puts the CPU in the Initial Program Load mode. This initiates the microprogram loading sequence which consists of reading a dedicated device address and then reading from the specified device. The device number is entered through the Serial System Control Panel.

CLOCK OVERRIDE
Depressing the CLK OVRD key activates the override condition; no further interrupts from the Real-Time Clock or the Interval Timer will be permitted. A second depression of this key deactivates the clock override condition.

OPERATION/MODE INDICATORS
The Operation/Mode indicators consist of single-bit, light-emitting diodes. These indicators display either the operational mode of the CPU or a conditioned interruption in computer operation.

PARITY ERROR
The PARITY ERROR display, when lit, indicates that a memory parity error has occurred during a CPU memory access.

INTERRUPT ACTIVE
The INTERRUPT ACTIVE display is on if any interrupt (I/O or external) is in the active state.
The CLOCK OVERRIDE display is on when the clock override condition is active (The CLK OVRD key is depressed.)

The RUN display is on when the CPU is in the Run mode. While in the Run mode, the CPU is executing instructions.

The HALT display is on when the CPU is in the Halt mode. In this mode, no instructions are executed.

The WAIT display is on when the CPU is in the Wait state: that is, no instructions are being executed. However, I/O operations continue to completion.

The Hexadecimal keyboard and the Function keyboard operate in conjunction with the panel displays as a unified Input/Output device to the CPU. Operation of the keyboards provides the capability to selectively store and/or read data in memory or in registers.

The Hexadecimal keyboard, referred to as the "Hex keyboard," is used to either enter data into the B-Display or to enter the source/destination of the panel function to be performed. The dual function of each Hex keyboard key is indicated by the upper and lower case values printed on each key.

The upper case values are used when data is entered into the B-Display. The upper case values are enabled by first depressing the Function keyboard KEYBOARD key. The Function keyboard KEYBOARD key causes the B-Display to be cleared and the KEYBOARD indicator to illuminate. When the KEYBOARD indicator is illuminated, all entries from the Hex keyboard are interpreted as data and are entered into the B-Display by a 4-bit left shift of the contents of the B-Display and insertion of the hex value of the depressed key into the four least significant bit positions (hex digit) of the B-Display. If the 32-bit capacity of the B-Display is exceeded, the most significant four bits of the B-Display are shifted out of the display and lost, and the new digit is loaded into the least significant bit positions.

The lower case values of the Hex keyboard are used to specify the source/destination of a function to be performed by the Serial System Control Panel. The lower case values are enabled by first depressing the Function keyboard WRITE or READ keys, causing the subsequent entry from the Hex keyboard to be interpreted as the source/destination of the Write or Read function. When a source/destination is entered in the Hex keyboard, it causes a corresponding indicator to illuminate on the Serial System Control Panel. The Hex keyboard keys that cause an indicator to illuminate are listed as follows:

1. The keys cause the EVEN register Hex indicator to indicate the hexadecimal value of the even register addressed.

2. The keys cause the ODD REGISTER Hex indicator to indicate the hexadecimal value of the odd register addressed.

3. The key causes the MEMORY ADDRESS indicator to illuminate.
4. The \( q_{PSW} \) key causes the PSW (Program Status Word) indicator to illuminate.

5. The \( A_{PC} \) key causes the PROGRAM COUNTER indicator to illuminate.

6. The \( B_{CSWS} \) key causes the CONTROL SWITCHES indicator to illuminate.

7. The \( C_{MD} \) key causes the MEMORY DATA indicator to illuminate.

8. The \( D_{EA} \) key causes the EFFECTIVE ADDRESS indicator to illuminate.

9. The \( E_{PSD} \) key causes the second word of the PSD to be displayed in the B-Display.

10. The \( F_{CONV} \) key causes a logical address in the A-Display to be converted to a 24-bit physical address and be displayed in the B-Display.

The Function keyboard sets the function to be performed by the Control Panel according to the key that is depressed. The functions that can be selected by the Function keyboard keys are as follows:

**WRITE \( X \) KEY**

Depressing the \( X_{WRITE} \) key causes the operand in the B-Display to be stored in the destination specified by a subsequent depression of a Hex keyboard key. The lower case value of the Hex keyboard key describes the destination of the operand and the function indicator that will illuminate. The use of the Hex keyboard \( D_{EA} \) key is prohibited for the destination of a Write function. If the Hex keyboard \( C_{MD} \) is depressed, the contents of the A-Display (which must contain a valid memory address, PSW, or Program Counter Value) are used to address memory. The operand in the B-Display is stored at that memory address.

**READ \( X \) KEY**

Depressing the \( X_{READ} \) key causes the operand specified by a subsequent depression of a Hex keyboard key to be loaded into either the A- or B-Display. The lower case value of the Hex keyboard key describes the source of the operand and the function indicator that will illuminate. The use of the Hex keyboard \( B_{MA} \) key is prohibited as a source of a Read function.

If the Hex keyboard \( C_{MD} \) key is depressed, the contents of the A-Display (which must contain a valid memory address, PSW, or Program Counter Value) are used to address memory. The contents of the addressed memory location are loaded into the B-Display.

**WRITE & INC 'A' KEY**

Depressing the WRITE & INC 'A' key causes the operand in the B-Display to be stored in the memory location addressed by the A-Display. The A-Display is then incremented by four (one memory word). The A-Display must contain a valid memory address, and the B-Display must contain the operand to be stored in memory. The WRITE & INC 'A' key is used for Write functions to sequential memory locations.

**INC 'A' & READ KEY**

The INC 'A' & READ key causes the address in the A-Display to be incremented by four (one memory word), and the updated address is used to address memory. The contents of the addressed memory location are then loaded into the B-Display. The A-Display must contain a valid memory address. The INC 'A' & READ Key is used for Read functions of sequential memory locations.
The EXT FUNCT key is used for extended functions, such as a lamp test routine.

Depressing the INSTR STOP key causes the Instruction Stop function to become active or inactive. If the Instruction Stop function was active, and the INSTR STOP indicator was illuminated, depressing the Function keyboard INSTR STOP key would deactivate the Instruction Stop function and turn off the indicator. If the Instruction Stop function was inactive, and the INSTR STOP indicator was off, depressing the Function keyboard INSTR STOP key would activate the Instruction Stop function, illuminate the INSTR Stop indicator and load the memory address from the B-Display into the Address Compare register. When the CPU fetches an instruction from the memory location specified by the Address Compare register, the STOP indicator illuminates, and the CPU halts. The B-Display must be loaded with the instruction address by way of the Hex keyboard before depressing the Function keyboard INSTR STOP key.

Depressing the OPRND R STOP key causes the Operand Read Stop function to become active or inactive. If the Operand Read Stop function was active, and the OPERAND READ STOP indicator was illuminated, depressing the Function keyboard OPRND R STOP key would deactivate the Operand Read Stop function and turn off the indicator. If the Operand Read Stop was inactive, depressing the Function keyboard OPRND R STOP key would activate the Operand Read Stop function and load the memory address from the B-Display into the Address Compare register. When the CPU reads an operand from the specified memory location, the STOP indicator illuminates, and the CPU halts. The B-Display must be loaded with the operand memory address by way of the Hex keyboard before depressing the OPRND R STOP key. The address in the B-Display for Compare Halt must be entered in a 24-bit physical address format.

Depressing the OPRND W STOP key causes the Operand Write Stop function to become active or inactive. If the Operand Write Stop function was active, and the OPERAND WRITE STOP indicator was illuminated, depressing the function keyboard OPRND W STOP key would deactivate the Operand Write Stop function and turn off the indicator. If the Operand Write Stop was inactive, depressing the Function keyboard OPRND W STOP key would activate the Operand Write Stop function, illuminate the OPERAND WRITE STOP indicator, and load the memory address from the B-Display into the Address Compare register. When the CPU stores an operand in the specified memory location, the STOP indicator illuminates, and the CPU halts. The B-Display must be loaded with the operand memory address by way of Hex keyboard before depressing the OPRND W STOP key. The address in the B-Display for Compare Halt must be entered in a 24-bit physical address format.

Depressing the INSTR STEP key causes both the A- and B-Displays and all function indicators, except the Instruction and Operand STOP indicators, to be cleared. It then causes the CPU to execute one software instruction that is addressed by the CPU Program Status Word Register. After one instruction has been executed, the CPU halts, the A-Display will indicate the next Program Status Word, and the B-Display will indicate the new Instruction word.

Depressing the KEYBOARD key causes the B-Display to be cleared, the KEYBOARD indicator to illuminate, and any subsequent Hex keyboard entries to be interpreted at their upper case values and inserted into the four rightmost bit positions of the B-Display. The KEYBOARD key is normally used to clear the B-Display before entering an operand into the B-Display from the Hex keyboard.
The A-Display consists of 32 binary indicators that are divided into eight 4-bit fields for easy hexadecimal read-out. When the Hex Display option is included in the Serial Control Panel, a hex display indicator above each 4-bit field provides a direct hexadecimal read-out of the contents of the field.

The contents of the A-Display are described by the function indicators directly to the right of the A-Display or by the EVEN REGISTER hex display indicator to the left of the A-Display. The contents of the A-Display can be any of the following:

1. A memory address in bit positions 8-31.
2. The contents of the CPU Program Status Word Register.
3. The Program Counter bits from the CPU Program Status Word Register in bit positions 8-31.
4. The most significant word of the Program Status Doubleword.
5. The contents of any of four even-numbered CPU general purpose registers.

The A-Display can be loaded in either a Write or a Read function, as specified by the corresponding keys of the Function keyboard. In a Write function, the A-Display is loaded as follows:

1. The B-Display is loaded with an operand or address by way of the Hex keyboard.
2. The Function keyboard WRITE key is depressed to specify the Write function.
3. The Hex keyboard lower case value (operand destination) is specified by depressing one of the even-numbered register keys on the MA, PSW, or PC keys.

In a Read function, the A-Display is loaded as follows:

1. The Function keyboard READ key is depressed to specify the Read function.
2. The Hex keyboard lower case value (operand source) is specified by depressing one of the even-numbered register keys, the PSW or the PC key.

When the Read function is complete, the operand specified by the Hex keyboard will be loaded into the A-Display, and the corresponding function indicator will illuminate to define the contents of the A-Display. The exception being the E key which will load PSD word 2 into the B-Display.

When the A-Display contains a memory address, Program Status Word, or Program Counter, the contents of the A-Display can be used to address memory during memory Read or Write functions. In these types of functions, the WRITE & INC 'A' and the INC 'A' & READ keys of the Function keyboard can be used to access memory and increment the contents of the A-Display to the next sequential memory word address.
The B-Display consists of 32 binary indicators that are divided into eight 4-bit fields for easy hexadecimal read-out. When the Hex Display option is included in the Serial System Control Panel, a hex display indicator above each 4-bit field provides a direct hexadecimal read-out of the contents of the field.

The contents of the B-Display are described by the function indicators to the right of the B-Display or by the ODD REGISTER hex display indicator to the left of the B-Display. The contents of the B-Display can be any of the following:

1. Keyboard data being entered from the Hex keyboard.
2. A memory data word.
3. An Effective Address of the instruction addressed by the PSW or PC in the A-Display.
4. An instruction addressed by the PSW or PC in the A-Display.
5. The contents of the CPU Control Switches in bit positions 0-11.
6. The contents of any of four odd-numbered CPU General Purpose Registers.
7. The least significant word of the Program Status Doubleword (PSD).
8. The physical address in an address conversion operation.

The B-Display can be loaded in either a Write or Read function, as specified by the corresponding keys of the Function keyboard. In a Write function, the B-Display is loaded as follows:

1. An operand is loaded from the Hex keyboard.
2. The Function keyboard \( \text{WRITE} \) key is depressed to specify the Write function.
3. The contents of the B-Display can be transferred to the A-Display by depressing any even-numbered register key, the MA key, the PSW key, or the PC key to specify the operand destination.
4. The contents of the B-Display can be transferred directly to an odd-numbered register, the CPU Control Switch register, or to the memory location addressed by the A-Display by depressing one of the odd-numbered register keys, the CSWS key, or the MD key, respectively, to specify the operand destination.

In a Read function, the B-Display is loaded as follows:

1. The Function keyboard \( \text{READ} \) key is depressed to specify a Read function.
2. The Hex keyboard lower case value (operand source) is specified by depressing an odd-numbered register key, the CSWS key, the MD key, the EA key, or the PSD2 key.

When the Read function is complete, the corresponding indicator will illuminate to define the contents of the B-Display.
The EVEN REGISTER indicator consists of a hexadecimal display (optional) indicator that provides a direct read-out of the even-numbered register being addressed by the Serial System Control Panel. The contents of this register are displayed to the left of the A-Display. The EVEN REGISTER indicator will be illuminated only when the A-Display contains the contents of an even-numbered register.

The four binary indicators directly below the EVEN REGISTER indicator correspond to the even register address.

The ODD REGISTER indicator consists of a hexadecimal display (optional) indicator that provides a direct read-out of the odd-numbered register being addressed by the Serial System Control Panel. The contents of this register are displayed in the B-Display. The ODD REGISTER indicator will be illuminated only when the B-Display contains the contents of an odd-numbered register.

The four binary displays directly below the ODD REGISTER indicator correspond to the odd register address.

The MEMORY ADDRESS indicator is a 1-bit display that defines the contents of the A-Display as a memory address. The memory address can only be loaded into the into the A-Display with a Write function. The memory address is primarily used for memory addressing in subsequent memory read or write operations.

The PSW indicator is a 1-bit display that defines the contents of the A-Display as the CPU Program Status Word Register. The PSW can be used for changing the contents of the CPU PSW and for memory addressing in subsequent memory read or write operations. In PSD mode, the A-Display represents the most significant word of the PSD.

The PROGRAM COUNTER indicator is a 1-bit display that defines the contents of the A-Display as the current value of the CPU Program Counter portion of the Program Status Word Register. The Program Counter can be loaded into the A-Display with either a Write or a Read function. The Program Counter can be used for changing the Program Counter portion of the Program Status Word Register and for memory addressing in subsequent memory read or write operations.

The OPERATOR FAULT indicator is a 1-bit display that indicates that an operator fault has occurred on the Serial System Control Panel. Two types of Operator Faults can normally occur:

1. The function selected by the Function keyboard was illogical with respect to the operand source/destination selected by the Hex keyboard.

2. The function selected by the Function keyboard combined with the operation and source/destination specified by the Hex keyboard cannot be performed because the CPU is in a Run mode and the specified function is not is not allowed.

The specific type of Operator Fault that has occurred must be determined by the Serial System Control Panel operator.
The MEMORY DATA indicator is a 1-bit display that defines the contents of the B-Display as memory data from the memory location addressed by the A-Display. For the MEMORY DATA indicator to be illuminated, the A-Display must contain a memory address and the MEMORY ADDRESS indicator must be illuminated. Memory data can be manually loaded into the B-Display and the addressed memory location in a Write function or read into the B-Display from the addressed memory location in a Read function.

The EFFECTIVE ADDRESS indicator is a 1-bit display that defines the contents of the B-Display as an effective address of a software memory reference instruction that is addressed by the contents of the A-Display. The A-Display must contain either a PSW or Program Counter Value, which is used by the CPU to access the software memory reference instruction. The CPU then computes the instruction's effective address based on any indexed or indirect addressing specified by the instruction. When the addressing is complete, the effective address can only be loaded into the B-Display by a Read function.

The ERROR indicator is a 1-bit display that defines the contents of the B-Display as an internal error code. The internal errors exclude operator errors and include Serial System Control Panel errors, CPU acknowledge errors, SelBUS transmission errors, and memory errors.

The CONTROLSWITCHES indicator is a 1-bit display that defines the contents of the B-Display as the CPU Control Switches. The Control Switches can be loaded into the B-Display in either a Write or a Read function. In a Write function, the B-Display is loaded from the Hex keyboard. The contents of the B-Display (Control Switches) are then loaded into a dedicated memory location. In a Read function, the Serial System Control Panel reads the dedicated memory location and transfers its contents (Control Switches) to the B-Display.

The specific dedicated memory address used for storage of the Control Switches is a function of the computer system configuration and CPU firmware.

The KEYBOARD indicator is a 1-bit display that indicates when the upper case values (hex digits 0 through F) can be loaded into the B-Display from the Hex keyboard. The KEYBOARD indicator illuminates in response to the KEYBOARD switch on the Function keyboard.

The INSTRUCTION indicator is a 1-bit display that defines the contents of the B-Display as an instruction addressed by a PSW or Program Counter Value in the A-Display. An instruction can be manually loaded into the B-Display and addressed memory location in a Write function or read into the B-Display from the addressed memory location in a Read function. The Serial System Control Panel defines the contents of any memory location as an instruction if the A-Display contains a PSW or Program Counter Value. If the A-Display contains a memory address (the MEMORY ADDRESS indicator is illuminated), the contents of the addressed memory location is defined as memory data, which illuminates the MEMORY DATA indicator.

The STOP indicator is a 1-bit display that indicates when the CPU has been halted by the Instruction Stop, Operand Read Stop, or Operand Write Stop logic. In addition to the STOP indicator, one or more of the INSTR STOP, OPERAND READ STOP, or OPERAND WRITE STOP indicators should also be illuminated indicating the type of stop logic that is active. When the STOP indicator illuminates and CPU halts, the A-Display will contain the current contents of the CPU PSW, and the B-Display will contain the instruction addressed by the Program Counter portion of the PSW (A-Display).
The INSTR STOP indicator is a 1-bit display that defines the active condition of the Instruction Stop logic. When the Instruction Stop is active, a memory address is in the Address Compare register. When the CPU fetches an instruction from that memory location, the CPU will halt and the STOP indicator will illuminate.

The OPERAND READ STOP indicator is a 1-bit display that defines the active condition of the Operand Read Stop logic. When Operand Read Stop is active, a memory address is in the Address Compare register. When the CPU performs a memory read from that memory location, the CPU will halt and the STOP indicator will illuminate.

The OPERAND WRITE STOP indicator is a 1-bit display that defines the active condition of the Operand Write Stop logic. When the Operand Write Stop is active, a memory address is in the Address Compare register. When the CPU performs a memory write to that location, the CPU will halt and the STOP indicator will illuminate.

The Serial System Control Panel is equipped with an OPERATOR FAULT indicator that illuminates when the panel detects an operator fault condition. When the OPERATOR FAULT indicator lights, the rightmost digit of the B-Display will indicate the source of the fault as follows:

<table>
<thead>
<tr>
<th>Fault Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Does not Apply to the Serial Panel</td>
</tr>
<tr>
<td>2.</td>
<td>Operation Not Allowed - Run on Lock Restrictions</td>
</tr>
<tr>
<td>3.</td>
<td>Invalid Operand Source or Destination</td>
</tr>
<tr>
<td>4.</td>
<td>A-Display Not Valid for Operation to be Performed</td>
</tr>
<tr>
<td>5.</td>
<td>Invalid Extended Function</td>
</tr>
<tr>
<td>6.</td>
<td>Special Extended Function Not Enabled</td>
</tr>
<tr>
<td>7.</td>
<td>Does not Apply to the Serial Panel</td>
</tr>
</tbody>
</table>

The Serial System Control Panel is equipped with an ERROR indicator that illuminates when a panel error is detected. When the ERROR indicator lights, the rightmost digit of the B-Display will indicate the source of the fault as follows:

<table>
<thead>
<tr>
<th>Fault Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>CPU Uart Error</td>
</tr>
<tr>
<td>2.</td>
<td>Transmission Error other than CPU Uart</td>
</tr>
<tr>
<td>3.</td>
<td>No Response from Memory</td>
</tr>
<tr>
<td>4.</td>
<td>Nonpresent Memory</td>
</tr>
<tr>
<td>5.</td>
<td>Parity Error in Memory</td>
</tr>
<tr>
<td>6.</td>
<td>Write/Read Compare Error in Memory</td>
</tr>
<tr>
<td>7.</td>
<td>Bus Interchange or Memory is Broken</td>
</tr>
</tbody>
</table>
Several indicators are available to the operator when the computer, while in the PSD mode, enters the Halt mode or when the PSW is read by the panel switches. They are as follows:

1. Bit 6 indicates last instruction executed was a right halfword.
2. Bit 7 indicates Arithmetic Exception.
3. Bit 8 indicates PSD mode if set or PSW mode if zero.
4. Bit 9 indicates Mapped if set or Unmapped if zero.
5. Bit 32 indicates Interrupts Blocked if set.

The following discussions provide step-by-step instructions for using the controls and indicators of the Serial System Control Panel. Each heading designates a specific function to be performed and the sequential steps necessary to complete the function. Each discussion includes two significant conditions necessary for each function: Panel Lock position and CPU mode.

Description of the Load B-Display from Hex keyboard and description of the Load A-Display provide the primary functions of the Serial System Control Panel that are necessary for all other functions. After these descriptions are initially presented, they are referred to by title only in subsequent descriptions.

1. The Panel Lock must be in the Unlocked mode.
2. The CPU can be in the Run or Halt mode.
3. Depress the KEYBOARD key on the Function keyboard.
4. Observe that the B-Display clears and the KEYBOARD indicator illuminates.
5. Enter the operand into the B-Display by depressing the correct hex digit key on the Hex keyboard, one digit at a time.
6. Observe that the last digit entered from the Hex keyboard is loaded into the four least significant bit positions of the B-Display and that any previous contents of the B-Display is left-shifted by four bit positions.
7. When the B-Display is full, or the complete operand has been entered into the B-Display, the operation is complete.
8. If the 32-bit capacity of the B-Display is exceeded, the four most significant bit positions of the B-Display will be lost as each new digit is entered into the B-Display.
9. If a mistake is made while entering the operand, depress the KEYBOARD key on the Function keyboard and return to step 4.

The Load A-Display function can be divided into seven subfunctions that described separately in the following descriptions. The seven subfunctions are:

1. Write Memory Address
2. Write PSW (Program Status Word)
3. Read PSW (Program Status Word)
4. Write PSD2
5. Read PSD2
6. Write Program Counter
7. Read Program Counter

WRITE MEMORY ADDRESS

1. The Panel Lock must be in the Unlocked mode.
2. The CPU can be in the Run or Halt mode.
3. Enter the memory address into the B-Display from the Hex keyboard. (See Load B-Display from Hex keyboard.)
4. Depress the WRITE key on the Function keyboard.
5. Depress the key on the Hex keyboard.
6. Observe that the memory address is transferred from the B-Display to the A-Display and that the MEMORY ADDRESS indicator illuminates.
7. The operation is complete. If a mistake was made during the sequence, return to Step 3.

WRITE PSW

1. The Panel Lock must be in the Unlocked mode
2. The CPU must be in the Halt mode.
3. Enter the PSW operand into the B-Display from the Hex keyboard. (See Load B-Display from Hex keyboard.)
4. Depress the WRITE key on the Function keyboard.
5. Depress the key on the Hex keyboard.
6. Observe that the PSW operand is transferred from the B-Display to the A-Display and that PSW indicator illuminates. At this time, the PSW operand has also been loaded into the CPU Program Status Word Register.
7. The operation is complete. If a mistake was made during the sequence, return to Step 3.

READ PSW

1. The Panel Lock must be in the Unlocked mode.
2. The CPU must be in the Halt mode.
3. Depress the key on the Function keyboard.
4. Depress the key on the Hex keyboard.
5. Observe that the Program Status Word is transferred from the CPU Program Status Word Register to the A-Display and that the PSW indicator illuminates.
WRITE PSD2

1. The Panel Lock must be in the Unlocked mode.
2. The CPU must be in the Halt mode.
3. Enter the PSD2 (least significant word of the PSD) operand into
   the B-Display from the Hex keyboard. (See Load B-Display from
   Hex keyboard).
4. Depress the \textit{WRITE} \textsubscript{X} key on the Function keyboard.
5. Depress the \textit{E} \textsubscript{PSD2} key on the Hex keyboard.
6. The operation is complete. If a mistake was made during the se-
   quence, return to Step 3.

READ PSD2

1. The Panel Lock must be in the Unlocked mode.
2. The CPU must be in the Halt mode.
3. Depress the \textit{READ} \textsubscript{X} key on the Function keyboard.
4. Depress the \textit{E} \textsubscript{PSD2} key on the Hex keyboard.
5. The operation is complete. If a mistake was made during the se-
   quence, return to Step 3.

WRITE PROGRAM COUNTER

1. The Panel Lock must be in the Unlocked mode.
2. The CPU must be in the Halt mode.
3. Enter the Program Counter Value into bits 8-31 of the B-Display
   from the Hex keyboard. (See Load B-Display from Hex keyboard.)
4. Depress the \textit{WRITE} \textsubscript{X} key on the Function keyboard.
5. Depress the \textit{A} \textsubscript{PC} key on the Hex keyboard.
6. Observe that bits 13-31 of the B-Display are transferred to the
   A-Display and that the PROGRAM COUNTER indicator illuminates. At
   this time, the Program Counter Value has been loaded into the
   Program Counter portion of the CPU Program Status Word Register.
7. The operation is complete. If a mistake was made during the se-
   quence, return to Step 3.

READ PROGRAM COUNTER

1. The Panel Lock must be in the Unlocked mode.
2. The CPU must be in the Halt mode.
3. Depress the \textit{READ} \textsubscript{X} key on the Function keyboard.
4. Depress the $A_{PC}$ key on the Hex keyboard.

5. Observe that the Program Counter Value is transferred from the CPU Program Status Word Register and transferred to bits 13-31 of the A-Display and that the PROGRAM COUNTER indicator illuminates.

6. The operation is complete. If a mistake was made during the sequence, return to Step 3.

The Write Memory sequence is dependent on a valid address (Memory Address, PSW, or Program Counter Value) in the A-Display. This value can be set in the A-Display by using any of the subfunctions described in the Load A-Display discussion.

1. The Panel Lock must be in the Unlocked mode.

2. Enter a Memory Address, PSW, or Program Counter Value into the A-Display as described in the Load A-Display discussion.

3. Enter the operand to be stored in memory into the B-Display from the Hex keyboard. (See Load B-Display from Hex keyboard.)

4. Depress the WRITE $X$ key on the Function keyboard.

5. Depress the $C_{MD}$ key on the Hex keyboard.

6. Observe that the operand in the B-Display remains unchanged and that either the MEMORY DATA or INSTRUCTION indicator illuminates as follows:
   a. If the A-Display contains a memory address, the MEMORY DATA indicator should illuminate.
   b. If the A-Display contains either a PSW or Program Counter Value, the INSTRUCTION indicator should illuminate.

7. The operation is complete. If a mistake was made during the sequence, return to Step 3.

The Read Memory sequence is dependent on a valid address (Memory Address, PSW, or Program Counter Value) in the A-Display. This value can be set in the A-Display by using any of the subfunctions described in the Load A-Display discussion.

1. The Panel Lock must be in the Unlocked mode.

2. Enter a Memory Address, PSW, or Program Counter Value into the A-Display as described in the Load A-Display discussion.

3. Depress the INC 'A' & READ key on the Function keyboard.

4. Observe that the A-Display is incremented by four to the next sequential memory address.

5. Observe that the MEMORY DATA or INSTRUCTION indicator illuminates as follows:
   a. If the A-Display contains a memory address, the MEMORY DATA indicator should illuminate.
b. If the A-Display contains a PSW or Program Counter Value, the INSTRUCTION indicator should illuminate.

6. The operand in the B-Display should be the contents of the memory location addressed by the A-Display.

7. If no mistakes occurred in the above sequence, return to Step 4 to read the next memory location.

8. If a mistake was made, the same memory address can be reread by performing the Read Memory (Single Address) sequence beginning with Step 4.

When using the Read Memory (Sequential Addresses) sequence, the first address entered into the A-Display will not be read. To read the first address, perform the Read Memory (Single Address) sequence, then enter the Read Memory (Sequential Addresses) sequence beginning with Step 4.

The Instruction Step function causes the CPU to enter the Run mode and execute one software instruction. After the instruction has been executed, the CPU returns to the Halt mode.

The sequence for the Instruction Step function is as follows:

1. The Panel Lock must be in the Unlocked mode.
2. The CPU must be in the Halt mode.
3. If the CPU Program Status Word Register does not point to the instruction to be executed, load a Program Counter or PSW Value into the A-Display and CPU register as described in the Load A-Display description.
4. Depress the INSTR STEP key on the Function keyboard.
5. Observe that the PANEL HALT indicator is illuminated.
6. The system halts with the updated PSW Value in the A-Display and instruction addressed by the A-Display (PSW) in the B-Display.
7. To execute the next instruction, return to Step 4.

The Read Effective Address sequence causes the CPU to fetch the instruction addressed by the Program Counter of PSW Value in the A-Display. The instruction fetched should be a memory reference instruction to generate a valid effective address. After the instruction has been fetched, the CPU calculates the instruction's effective memory address by performing the indexing and indirect addressing specified by the instruction. When the address computations are complete, the CPU transfers the effective address to the Serial System Control Panel's B-Display.

The Read Effective Address sequence is as follows:

1. The Panel Lock must be in the Unlocked mode.
2. The CPU must be in the Halt mode.
3. Enter a PSW or Program Counter Value into the A-Display as described in the Load A-Display discussion.
4. Depress the READ key on the Function keyboard.
5. Depress the $D_{EA}$ key on the Hex keyboard.

6. Observe that the EFFECTIVE ADDRESS indicator illuminates and the effective address is loaded into the B-Display.

7. The operation is complete. If a mistake occurred, return to Step 3.

The Convert Address sequence causes conversion of a logical address in the A-Display to a 24-bit physical address in the B-Display.

The Convert Address sequence is as follows:

1. The Panel Lock must be in the Unlocked mode.

2. The CPU must be in the Halt mode.

3. The CPU must be in the PSD mode.

4. Enter a PSW, Program Counter Value, or memory address in the A-Display as described in the Load A-Display discussion.

5. Depress the $READ_X$ key on the Function keyboard.

6. Depress the $F_{CONV}$ key on the Hex keyboard.

7. The operation is complete. If a mistake occurred, return to Step 4.

The Stop sequence includes the Instruction Stop, Operand Read Stop, and Operand Write Stop functions. Each function has its own key on the Function Keyboard and its own indicator to indicate when that function is active.

The sequence for the Stop functions is as follows:

1. The Panel Lock must be in the Unlocked mode.

2. The CPU must be in the Halt mode.

3. Enter the memory stop address into the B-Display from the Hex keyboard.

4. Depress the INSTR STOP, OPRND R STOP, or OPRND W STOP key on the Function keyboard.

5. Observe that the indicator for the Stop function selected by the Function keyboard illuminates.

6. If the CPU is in the Run mode and the specified memory location is accessed in the correct operating mode (Instruction Fetch, Operand Read, or Operand Write), the following events should occur.

a. The PANEL HALT indicator should illuminate.

b. The STOP indicator should illuminate.

   c. The current contents of the CPU PSWR should appear in the A-Display, and the PSW indicator should illuminate.
d. The instruction addressed by the Program Counter portion of the PSW should appear in the B-Display, and the INSTRUCTION indicator should illuminate.

7. To clear any active Stop function, perform the following steps:
   a. Depress the Function keyboard key that corresponds to the function to be cleared.
   b. Observe that the corresponding Stop function indicator turns.

When using the Stop function, multiple Stop functions can be set by entering the Stop functions sequentially; however, if a different Stop address is entered with each Stop function, the most recently entered Stop address will be used for all active Stop functions.

The Control Switches sequence is used to set or monitor the CPU Control Switches that are stored in a dedicated memory location. The Control Switches sequence is divided into the Write Control Switches function that sets the Control Switches in the dedicated memory location and the Read Control Switches function that reads the contents of the dedicated memory location.

**CONTROL SWITCHES SEQUENCE**

**WRITE CONTROL SWITCHES**

1. The Panel Lock must be in the Unlocked mode.
2. Enter the Control Switch configuration into bit positions 0-12 of the B-Display from the Hex keyboard. (See Load B-Display from Hex keyboard).
3. Depress the \( \text{WRITE} \) key on the Function keyboard.
4. Depress the \( \text{B}_{\text{CSWS}} \) key on the Hex keyboard.
5. Observe that the CONTROL SWITCHES indicator illuminates. At this time, the contents of the B-Display have been transferred to the control switches dedicated memory location.
6. The operation is complete. If a mistake was made, return to Step 3.

**READ CONTROL SWITCHES**

1. The Panel Lock must be in the Unlocked mode.
2. The CPU can be in the Run or Halt mode.
3. Depress the \( \text{READ} \) key on the Function keyboard.
4. Depress the \( \text{B}_{\text{CSWS}} \) key on the Hex keyboard.
5. Observe that the CONTROL SWITCHES indicator illuminates, and the contents of the control switches dedicated memory location are transferred to the B-Display.
6. The operation is complete. If a mistake was made, return to Step 3.
The Initial Program Load (IPL) sequence is a function of the Serial System Control Panel and CPU firmware. The IPL sequence is as follows:

1. The Panel Lock must be in the Unlocked mode.
2. The CPU must be in the Halt mode.
3. Depress the SYSTEM RESET key.
4. Enter the peripheral device address of the IPL device into the B-Display from the Hex keyboard. (See Load B-Display from Hex keyboard.) Note: If an all-zeros device address is entered into the B-Display, the CPU firmware will default to a firmware-specified IPL device address.
5. Depress the INITIAL PROGRAM LOAD key.
6. When the IPL sequence is complete, the CPU will be in the Halt mode. Any changes in the software program can be made at this time.
7. The operation is complete. Refer to the software description of the Bootstrap program for operating instructions of the Bootstrap program.
SECTION VIII
SYSTEM INITIALIZATION

INITIAL PROGRAM LOAD (IPL)

Initialization and configuration of a 32/70 Series System is accomplished through the use of the Initial Program Load (IPL) sequence. This sequence initializes the system, sets up the I/O configuration, and boots in the operating system. The usual method of initializing the system is through the use of the card reader to read in a deck of cards containing the I/O device configuration and assigned interrupt organization. The IPL sequence is initiated by placing the Initial Configuration Load (ICL) deck of cards in the card reader, setting up of the address of the card reader on the system front panel, and depressing the IPL button on the system front panel.

It should be noted that if the mode jumper on the CPU is set up for the PSD mode, the CPU will come up in the PSD mode. If, when placing the address of the IPL device in the B-Display of the front panel, additional information is added, then the CPU can be made to come up in the PSW mode of operation. The procedure for establishing the PSW mode of operation is as follows:

1. If using either the parallel or serial front panel for data entry, add 8000 to the device address (sets bit 16 to One). For example, if the address of the card reader is 7800, then by the setting of bit 16 to One (or adding 8000), the resultant address becomes F800.

2. If using the serial front panel, entering a 55 plus the card reader address results in the CPU coming up in the PSW mode. The resultant address in the B-Display is then 00557800.

After the cards are read into the system, the SYSTEM RESET button is depressed, the address of the device (disc) containing the operating system is entered on the front panel, and the IPL button is again depressed, thereby booting in the operating system.

The Initial Configuration Load (ICL) deck of cards contains three basic record formats. The following sections provide descriptions for each format.

FORMATS OF THE INITIAL CONFIGURATION LOAD (ICL)

Initial Configuration Load (ICL) records are read from a default or selected peripheral device. The ICL records are converted into information that is used to initialize the 256- x 32-bit Configuration RAM (CR) contained in the 32/70 Series Central Processor Unit (CPU). Information contained in the CR is used by the CPU to address and maintain the status of the 128 possible devices and the 112 possible interrupts.

Initial Configuration Load records must be in the following ASCII or Hollerith formats:
FORMAT #1  *DEVXX=FCILCASA (,NN)

where:

*DEV  defines that the record contains a controller definition entry.

XX  is the hexadecimal address that will be used by macro level input/output instructions to address the controller.

=  is a necessary delimiter. Each letter to the right of this delimiter represents one hexadecimal digit (four binary bits).

F  flags used by the CPU for input/output emulation. Presently, this field is always zero.

C  defines the class of controller being emulated. Presently, this field can contain one of the following values:

0 = LINE PRINTER
1 = CARD READER
2 = TELETYPE
3 = INTERVAL TIMER
4 = PANEL
5 to D = Unassigned
E = ALL OTHERS
F = EXTENDED I/O

IL is the hexadecimal interrupt priority level of the Service Interrupt (i.e., priority levels 14_{16} through 23_{16}) for the defined controller.

CA is the hexadecimal controller address as defined by the hardware switches on the IOM.

SA is the lowest hexadecimal device subaddress used by the controller. This field is normally zero when more than one device is configured.

( ) denotes optional parameter.

is a delimiter that must be used when more than one device is configured.

NN is a 2-digit hexadecimal number that specifies the number of devices configured on the controller.

NOTE 1: The subaddress (SA) field must reflect the following for the Teletype, Line Printer, Card Reader (TLC) controller:

1. Card Reader is subaddress 0_{16}.
2. Teletype is subaddress 1_{16}.
3. Line Printer is subaddress 2_{16}. 


FORMAT #2

**INTXX RS**

where:

- **INT** defines that the record contains an interrupt definition entry.
- **XX** is the hexadecimal interrupt priority level that is to be emulated.
- **=** is a necessary delimiter. Each letter to the right of this delimiter represents one hexadecimal digit (four binary bits).
- **R** is the hexadecimal RTOM board number to which the interrupt XX is assigned.
- **S** is the hexadecimal subaddress on the RTOM board to which the interrupt XX is assigned.

**NOTE 1:** RTOM physical controller address 79\textsubscript{16} is RTOM board number 1, address 7A\textsubscript{16} is RTOM board number 2, etc.

**NOTE 2:** Real-Time Clock hardware is connected to subaddress 6\textsubscript{16} on the RTOM board.

**NOTE 3:** Interval Timer hardware is connected to subaddress 4\textsubscript{16} on the RTOM board.

**NOTE 4:** RTOM physical controller addresses must be 79\textsubscript{16} or above. This convention allows a maximum of seven RTOM boards to be defined on a single 32/70 Series system. Seven RTOM boards will support 112\textsubscript{10} interrupt levels.

FORMAT #3

**END**

where:

- **END** is the last record of an Initial Configuration Load (ICL) deck. This record signifies the end of the load process.

**EXAMPLES OF INITIAL CONFIGURATION LOAD (ICL) RECORDS**

A device entry:

- **DEV04=OE140100,04**

The device entry above specifies the following information:

1. The 32/70 series input/output commands will address the controller as 04\textsubscript{16}.
2. The "04" is an optional parameter that specifies that there are 4\textsubscript{16} devices on the controller. There will be four entries defined in the Configuration RAM (CR). The input/output commands (i.e., CD and TD) will address the devices as 4\textsubscript{16}, 5\textsubscript{16}, 6\textsubscript{16}, and 7\textsubscript{16}.
3. The controller is an "E" class controller.
4. The priority of the Service Interrupt (SI) is 14\textsubscript{16}.
Assigning a priority to a controller has the following implications:

a. The Transfer Interrupt location for priority \(14_{16}\) is \(100_{16}\).
b. The Service Interrupt vector location for priority \(14_{16}\) is \(140_{16}\).
c. The emulation IOCD will be stored at location \(700_{16}\).
d. The interrupt control instructions (i.e., DI, EI, RI, AI, DAI) will control the interrupt on the controller by addressing priority \(14_{16}\).

5. The physical address of the controller is \(01_{16}\).

An interrupt entry (RTOM):

\*INT28=16

The interrupt entry above specifies the following information:

1. The 32/70 Series interrupt control instructions (i.e., DI, EI, RI, AI, DAI) will control the interrupt on the RTOM by addressing priority \(28_{16}\).
2. The number of the RTOM board is 1.
3. The subaddress on the RTOM board is \(6_{16}\) (jumpered logic subaddress is 9).

A sample Initial Configuration Load (ICL) Deck is given in Figure 8-1.
**EXAMPLE** | **COMMENTS**
--- | ---
*(SEE NOTE)* | READ ASCII CARD READER IOC D
*DEV04=0E150400,02* | CARTRIDGE DISC WITH TWO PLATTERS
*DEV08=0E160800,04* | MOVING-HEAD DISC
*DEV10=0E181000,04* | 9-TRACK MAG TAPE
*DEV20=0E1A2000,10* | GPMC
*DEV60=0E1E6000,08* | ADS
*DEV78=01207800* | PRIMARY CARD READER
*DEV7A=00217802* | PRIMARY LINE PRINTER
*DEV7E=02237801* | PRIMARY TELETY P
*INT00=1F* | POWER FAIL/AUTO RESTART
*INT01=1E* | SYSTEM OVERRIDE
*INT12=1D* | MEMORY PARITY TRAP
*INT13=1C* | CONSOLE INTERRUPT
*INT24=1B* | NONPRESENT MEMORY
*INT25=1A* | UNDEFINED INSTRUCTION TRAP
*INT26=19* | PRIVILEGE VIOLATION
*INT27=18* | CALL MONITOR
*INT28=16* | REAL-TIME CLOCK
*INT29=17* | ARITHMETIC EXCEPTION
*INT2A=15* | EXTERNAL INTERRUPT
*INT28=14* | EXTERNAL INTERRUPT
*INT2C=13* | EXTERNAL INTERRUPT
*INT2D=12* | EXTERNAL INTERRUPT
*END* | LAST CARD

**NOTE:** THE FIRST RECORD IS DEVICE DEPENDENT AND REPRESENTS TWO 32-BIT WORDS, THE FIRST BEING ALL ZEROS AND THE SECOND A VALID IOC D TO READ THE FOLLOWING RECORDS.

Figure 8-1. System Initial Configuration Load (ICL) Deck
APPENDIX A
INSTRUCTION SET
(FUNCTIONALLY GROUPED)

The 32/70 Series instructions are listed alphabetically by mnemonic code within one of the following functional groupings:

- Load/Store Instructions
- Branch Instructions
- Compare Instructions
- Logical Instructions
- Register Transfer Instructions
- Shift Operation Instructions
- Bit Manipulation Instructions
- Fixed-Point Arithmetic Instructions
- Floating-Point Arithmetic Instructions
- Control Instructions
- Interrupt Instructions
- Input/Output Instructions
- Memory Management
- Writable Control Storage

Each entry includes the following information:

- Instruction Mnemonic
- Operand Format
- Operation Code
- Instruction Function

The following symbols are used to denote required entries for operand formats:

- b - Bit Number In General Register (0-31)
- c - Bit Number In Memory Byte
- d - Destination General Register (0-7)
- f - Function
- m - Memory Address
- n - Channel Or Device Number
- p - Protect Register Number
- s - Source General Register (0-7)
- v - Value of Operand For Immediate, Shift, and Condition Code Instructions
- x - Index Register (1-3)
- * - Indirect Addressing
- z - Register Address Field for Special Instructions

Halfword instructions are denoted by # preceding the instruction mnemonic. The halfword instructions are all interregister (except TRP and TPR) instructions: CALM, WAIT, HALT, and NOP.
# LOAD/STORE INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand Format</th>
<th>Op Code</th>
<th>Page</th>
<th>Instruction Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>LB</td>
<td>d,*m,x</td>
<td>AC08</td>
<td>6-10</td>
<td>Load Byte</td>
</tr>
<tr>
<td>LD</td>
<td>d,*m,x</td>
<td>AC00</td>
<td>6-13</td>
<td>Load Doubleword</td>
</tr>
<tr>
<td>LH</td>
<td>d,*m,x</td>
<td>AC00</td>
<td>6-11</td>
<td>Load Halfword</td>
</tr>
<tr>
<td>LW</td>
<td>d,*m,x</td>
<td>AC00</td>
<td>6-12</td>
<td>Load Word</td>
</tr>
<tr>
<td>LF</td>
<td>d,*m,x</td>
<td>CC00</td>
<td>6-28</td>
<td>Load File</td>
</tr>
<tr>
<td>LEA</td>
<td>d,*m,x</td>
<td>D000</td>
<td>6-23</td>
<td>Load Effective Address</td>
</tr>
<tr>
<td>LEAR</td>
<td>d,*m,x</td>
<td>B000</td>
<td>6-24</td>
<td>Load Effective Address Real</td>
</tr>
<tr>
<td>LA</td>
<td>d,*m,x</td>
<td>3400</td>
<td>6-25</td>
<td>Load Address</td>
</tr>
<tr>
<td>LI</td>
<td>d,v</td>
<td>C800</td>
<td>6-22</td>
<td>Load Immediate</td>
</tr>
<tr>
<td>LMB</td>
<td>d,*m,x</td>
<td>B008</td>
<td>6-14</td>
<td>Load Masked Byte</td>
</tr>
<tr>
<td>LMD</td>
<td>d,*m,x</td>
<td>B000</td>
<td>6-17</td>
<td>Load Masked Doubleword</td>
</tr>
<tr>
<td>LMH</td>
<td>d,*m,x</td>
<td>B000</td>
<td>6-15</td>
<td>Load Masked Halfword</td>
</tr>
<tr>
<td>LMW</td>
<td>d,*m,x</td>
<td>B000</td>
<td>6-16</td>
<td>Load Masked Word</td>
</tr>
<tr>
<td>LNB</td>
<td>d,*m,x</td>
<td>B408</td>
<td>6-18</td>
<td>Load Negative Byte</td>
</tr>
<tr>
<td>LND</td>
<td>d,*m,x</td>
<td>B400</td>
<td>6-21</td>
<td>Load Negative Doubleword</td>
</tr>
<tr>
<td>LNH</td>
<td>d,*m,x</td>
<td>B400</td>
<td>6-19</td>
<td>Load Negative Halfword</td>
</tr>
<tr>
<td>LNW</td>
<td>d,*m,x</td>
<td>B400</td>
<td>6-20</td>
<td>Load Negative Word</td>
</tr>
<tr>
<td>STB</td>
<td>s,*m,x</td>
<td>D408</td>
<td>6-29</td>
<td>Store Byte</td>
</tr>
<tr>
<td>STD</td>
<td>s,*m,x</td>
<td>D400</td>
<td>6-32</td>
<td>Store Doubleword</td>
</tr>
<tr>
<td>STW</td>
<td>s,*m,x</td>
<td>D400</td>
<td>6-30</td>
<td>Store Halfword</td>
</tr>
<tr>
<td>STF</td>
<td>s,*m,x</td>
<td>D400</td>
<td>6-31</td>
<td>Store Word</td>
</tr>
<tr>
<td>STF</td>
<td>s,*m,x</td>
<td>D400</td>
<td>6-37</td>
<td>Store File</td>
</tr>
<tr>
<td>STMB</td>
<td>s,*m,x</td>
<td>D808</td>
<td>6-33</td>
<td>Store Masked Byte</td>
</tr>
<tr>
<td>STMD</td>
<td>s,*m,x</td>
<td>D800</td>
<td>6-36</td>
<td>Store Masked Doubleword</td>
</tr>
<tr>
<td>STMH</td>
<td>s,*m,x</td>
<td>D800</td>
<td>6-34</td>
<td>Store Masked Halfword</td>
</tr>
<tr>
<td>STMW</td>
<td>s,*m,x</td>
<td>D800</td>
<td>6-35</td>
<td>Store Masked Word</td>
</tr>
<tr>
<td>ZMB</td>
<td>*m,x</td>
<td>F808</td>
<td>6-39</td>
<td>Zero Memory Byte</td>
</tr>
<tr>
<td>ZMD</td>
<td>*m,x</td>
<td>F800</td>
<td>6-42</td>
<td>Zero Memory Doubleword</td>
</tr>
<tr>
<td>ZMH</td>
<td>*m,x</td>
<td>F800</td>
<td>6-40</td>
<td>Zero Memory Halfword</td>
</tr>
<tr>
<td>ZMW</td>
<td>*m,x</td>
<td>F800</td>
<td>6-41</td>
<td>Zero Memory Word</td>
</tr>
<tr>
<td>#ZR</td>
<td>d</td>
<td>0C00</td>
<td>6-43</td>
<td>Zero Register</td>
</tr>
</tbody>
</table>

# MEMORY MANAGEMENT INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand Format</th>
<th>Op Code</th>
<th>Page</th>
<th>Instruction Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>#SEA</td>
<td></td>
<td>0000</td>
<td>6-59</td>
<td>Set Extended Addressing</td>
</tr>
<tr>
<td>#CEA</td>
<td></td>
<td>000F</td>
<td>6-60</td>
<td>Clear Extended Addressing</td>
</tr>
<tr>
<td>LMAP</td>
<td>d</td>
<td>2C07</td>
<td>6-61</td>
<td>Load MAP</td>
</tr>
<tr>
<td>#TMAPR</td>
<td>s,d</td>
<td>2C0A</td>
<td>6-62</td>
<td>Transfer MAP to Register</td>
</tr>
</tbody>
</table>

# Indicates Halfword Instruction
* Indicates Indirect Addressing
# BRANCH INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand Format</th>
<th>Op Code</th>
<th>Page</th>
<th>Instruction Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCF</td>
<td>v,*m,x</td>
<td>F000</td>
<td>6-73</td>
<td>Branch Condition False</td>
</tr>
<tr>
<td>BCT</td>
<td>v,*m,x</td>
<td>E000</td>
<td>6-74</td>
<td>Branch Condition True</td>
</tr>
<tr>
<td>BFT</td>
<td>*m,x</td>
<td>F000</td>
<td>6-75</td>
<td>Branch Function True</td>
</tr>
<tr>
<td>BIB</td>
<td>d,m</td>
<td>F400</td>
<td>6-77</td>
<td>Branch After Incrementing Byte</td>
</tr>
<tr>
<td>BID</td>
<td>d,m</td>
<td>F460</td>
<td>6-80</td>
<td>Branch After Incrementing Doubleword</td>
</tr>
<tr>
<td>BIH</td>
<td>d,m</td>
<td>F420</td>
<td>6-78</td>
<td>Branch After Incrementing Halfword</td>
</tr>
<tr>
<td>BIW</td>
<td>d,m</td>
<td>F440</td>
<td>6-79</td>
<td>Branch After Incrementing Word</td>
</tr>
<tr>
<td>BL</td>
<td>*m,x</td>
<td>F880</td>
<td>6-76</td>
<td>Branch and Link</td>
</tr>
<tr>
<td>BU</td>
<td>*m,x</td>
<td>E000</td>
<td>6-72</td>
<td>Branch Unconditionally</td>
</tr>
</tbody>
</table>

# COMPARE INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand Format</th>
<th>Op Code</th>
<th>Page</th>
<th>Instruction Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAMB</td>
<td>d,*m,x</td>
<td>9008</td>
<td>6-83</td>
<td>Compare Arithmetic with Memory Byte</td>
</tr>
<tr>
<td>CAMD</td>
<td>d,*m,x</td>
<td>9000</td>
<td>6-86</td>
<td>Compare Arithmetic with Memory Doubleword</td>
</tr>
<tr>
<td>CAMH</td>
<td>d,*m,x</td>
<td>9000</td>
<td>6-84</td>
<td>Compare Arithmetic with Memory Halfword</td>
</tr>
<tr>
<td>CAMW</td>
<td>d,*m,x</td>
<td>9000</td>
<td>6-85</td>
<td>Compare Arithmetic with Memory Word</td>
</tr>
<tr>
<td>#CAR</td>
<td>s,d</td>
<td>1000</td>
<td>6-87</td>
<td>Compare Arithmetic with Register</td>
</tr>
<tr>
<td>CI</td>
<td>d,v</td>
<td>C005</td>
<td>6-88</td>
<td>Compare Immediate</td>
</tr>
<tr>
<td>CMDB</td>
<td>d,*m,x</td>
<td>9408</td>
<td>6-89</td>
<td>Compare Masked with Memory Byte</td>
</tr>
<tr>
<td>CMMD</td>
<td>d,*m,x</td>
<td>9400</td>
<td>6-92</td>
<td>Compare Masked with Memory Doubleword</td>
</tr>
<tr>
<td>CMMH</td>
<td>d,*m,x</td>
<td>9400</td>
<td>6-90</td>
<td>Compare Masked with Memory Halfword</td>
</tr>
<tr>
<td>CMMW</td>
<td>d,*m,x</td>
<td>9400</td>
<td>6-91</td>
<td>Compare Masked with Memory Word</td>
</tr>
<tr>
<td>#CMR</td>
<td>s,d</td>
<td>1400</td>
<td>6-93</td>
<td>Compare Masked with Register</td>
</tr>
</tbody>
</table>

# LOGICAL INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand Format</th>
<th>Op Code</th>
<th>Page</th>
<th>Instruction Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANMB</td>
<td>d,*m,x</td>
<td>8408</td>
<td>6-95</td>
<td>AND Memory Byte</td>
</tr>
<tr>
<td>ANMD</td>
<td>d,*m,x</td>
<td>8400</td>
<td>6-98</td>
<td>AND Memory Doubleword</td>
</tr>
<tr>
<td>ANMH</td>
<td>d,*m,x</td>
<td>8400</td>
<td>6-96</td>
<td>AND Memory Halfword</td>
</tr>
<tr>
<td>ANMW</td>
<td>d,*m,x</td>
<td>8400</td>
<td>6-97</td>
<td>AND Memory Word</td>
</tr>
<tr>
<td>#ANR</td>
<td>s,d</td>
<td>0400</td>
<td>6-99</td>
<td>AND Register and Register</td>
</tr>
<tr>
<td>EOMB</td>
<td>d,*m,x</td>
<td>8C08</td>
<td>6-106</td>
<td>Exclusive OR Memory Byte</td>
</tr>
<tr>
<td>EOMD</td>
<td>d,*m,x</td>
<td>8C00</td>
<td>6-109</td>
<td>Exclusive OR Memory Doubleword</td>
</tr>
<tr>
<td>EOMH</td>
<td>d,*m,x</td>
<td>8C00</td>
<td>6-107</td>
<td>Exclusive OR Memory Halfword</td>
</tr>
<tr>
<td>EDMW</td>
<td>d,*m,x</td>
<td>8C00</td>
<td>6-108</td>
<td>Exclusive OR Memory Word</td>
</tr>
<tr>
<td>#EOR</td>
<td>s,d</td>
<td>0C00</td>
<td>6-110</td>
<td>Exclusive OR Register and Register</td>
</tr>
<tr>
<td>#EORM</td>
<td>s,d</td>
<td>0C08</td>
<td>6-111</td>
<td>Exclusive OR Register and Register Masked</td>
</tr>
<tr>
<td>ORMB</td>
<td>d,*m,x</td>
<td>8808</td>
<td>6-100</td>
<td>OR Memory Byte</td>
</tr>
<tr>
<td>ORMD</td>
<td>d,*m,x</td>
<td>8800</td>
<td>6-103</td>
<td>OR Memory Doubleword</td>
</tr>
<tr>
<td>ORMH</td>
<td>d,*m,x</td>
<td>8800</td>
<td>6-101</td>
<td>OR Memory Halfword</td>
</tr>
<tr>
<td>ORMW</td>
<td>d,*m,x</td>
<td>8800</td>
<td>6-102</td>
<td>OR Memory Word</td>
</tr>
<tr>
<td>#ORR</td>
<td>s,d</td>
<td>0800</td>
<td>6-104</td>
<td>OR Register and Register</td>
</tr>
<tr>
<td>#ORRM</td>
<td>s,d</td>
<td>0808</td>
<td>6-105</td>
<td>OR Register and Register Masked</td>
</tr>
</tbody>
</table>

# Indicates Halfword Instruction
* Indicates Indirect Addressing
### REGISTER TRANSFER INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand Format</th>
<th>Op Code</th>
<th>Page</th>
<th>Instruction Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>#XCR</td>
<td>s,d</td>
<td>2C05</td>
<td>6-55</td>
<td>Exchange Registers</td>
</tr>
<tr>
<td>#XCRM</td>
<td>s,d</td>
<td>2C0D</td>
<td>6-56</td>
<td>Exchange Registers Masked</td>
</tr>
<tr>
<td>TPR</td>
<td>r,p</td>
<td>FB80</td>
<td>6-50</td>
<td>Transfer Protect Register to Register</td>
</tr>
<tr>
<td>#TRC</td>
<td>s,d</td>
<td>2C03</td>
<td>6-53</td>
<td>Transfer Register Complement</td>
</tr>
<tr>
<td>#TRCM</td>
<td>s,d</td>
<td>2C0B</td>
<td>6-54</td>
<td>Transfer Register Complement Masked</td>
</tr>
<tr>
<td>#TRN</td>
<td>s,d</td>
<td>2C04</td>
<td>6-51</td>
<td>Transfer Register Negative</td>
</tr>
<tr>
<td>#TRNM</td>
<td>s,d</td>
<td>2C0C</td>
<td>6-52</td>
<td>Transfer Register Negative Masked</td>
</tr>
<tr>
<td>TRP</td>
<td>s,p</td>
<td>FB00</td>
<td>6-49</td>
<td>Transfer Register to Protect Register</td>
</tr>
<tr>
<td>#TRR</td>
<td>s,d</td>
<td>2C00</td>
<td>6-47</td>
<td>Transfer Register to Register</td>
</tr>
<tr>
<td>#TRRM</td>
<td>s,d</td>
<td>2C08</td>
<td>6-48</td>
<td>Transfer Register to Register Masked</td>
</tr>
<tr>
<td>#TRSW</td>
<td>s</td>
<td>2800</td>
<td>6-57</td>
<td>Transfer Register to PSWR</td>
</tr>
<tr>
<td>#TRSC</td>
<td>s,d</td>
<td>2C0E</td>
<td>6-46</td>
<td>Transfer Register to Scratchpad</td>
</tr>
<tr>
<td>#TSCR</td>
<td>s,d</td>
<td>2C0F</td>
<td>6-45</td>
<td>Transfer Scratchpad to Register</td>
</tr>
</tbody>
</table>

### SHIFT OPERATION INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand Format</th>
<th>Op Code</th>
<th>Page</th>
<th>Instruction Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>#NOR</td>
<td>d,s</td>
<td>6000</td>
<td>6-113</td>
<td>Normalize</td>
</tr>
<tr>
<td>#NORD</td>
<td>d,s</td>
<td>6400</td>
<td>6-114</td>
<td>Normalize Double</td>
</tr>
<tr>
<td>#SCZ</td>
<td>d,s</td>
<td>6800</td>
<td>6-115</td>
<td>Shift and Count Zeros</td>
</tr>
<tr>
<td>#SLA</td>
<td>d,v</td>
<td>6C40</td>
<td>6-116</td>
<td>Shift Left Arithmetic</td>
</tr>
<tr>
<td>#SLAD</td>
<td>d,v</td>
<td>7840</td>
<td>6-119</td>
<td>Shift Left Arithmetic Double</td>
</tr>
<tr>
<td>#SLC</td>
<td>d,v</td>
<td>7440</td>
<td>6-118</td>
<td>Shift Left Circular</td>
</tr>
<tr>
<td>#SLL</td>
<td>d,v</td>
<td>7040</td>
<td>6-117</td>
<td>Shift Left Logical</td>
</tr>
<tr>
<td>#SLLD</td>
<td>d,v</td>
<td>7C40</td>
<td>6-120</td>
<td>Shift Left Logical Double</td>
</tr>
<tr>
<td>#SRA</td>
<td>d,v</td>
<td>6C00</td>
<td>6-121</td>
<td>Shift Right Arithmetic</td>
</tr>
<tr>
<td>#SRAD</td>
<td>d,v</td>
<td>7800</td>
<td>6-124</td>
<td>Shift Right Arithmetic Double</td>
</tr>
<tr>
<td>#SRC</td>
<td>d,v</td>
<td>7400</td>
<td>6-123</td>
<td>Shift Right Circular</td>
</tr>
<tr>
<td>#SRL</td>
<td>d,v</td>
<td>7000</td>
<td>6-122</td>
<td>Shift Right Logical</td>
</tr>
<tr>
<td>#SRLD</td>
<td>d,v</td>
<td>7C00</td>
<td>6-125</td>
<td>Shift Right Logical Double</td>
</tr>
</tbody>
</table>

### BIT MANIPULATION INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand Format</th>
<th>Op Code</th>
<th>Page</th>
<th>Instruction Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABM</td>
<td>c,*m,x</td>
<td>A008</td>
<td>6-132</td>
<td>Add Bit in Memory</td>
</tr>
<tr>
<td>#ABR</td>
<td>d,b</td>
<td>2000</td>
<td>6-133</td>
<td>Add Bit in Register</td>
</tr>
<tr>
<td>SBM</td>
<td>c,*m,x</td>
<td>9808</td>
<td>6-128</td>
<td>Set Bit in Memory</td>
</tr>
<tr>
<td>#SBR</td>
<td>d,b</td>
<td>1800</td>
<td>6-129</td>
<td>Set Bit in Register</td>
</tr>
<tr>
<td>TBM</td>
<td>c,*m,x</td>
<td>A408</td>
<td>6-134</td>
<td>Test Bit in Memory</td>
</tr>
<tr>
<td>#TBR</td>
<td>d,b</td>
<td>2400</td>
<td>6-135</td>
<td>Test Bit in Register</td>
</tr>
<tr>
<td>ZBM</td>
<td>c,*m,x</td>
<td>9C08</td>
<td>6-130</td>
<td>Zero Bit in Memory</td>
</tr>
<tr>
<td>#ZBR</td>
<td>d,b</td>
<td>1C00</td>
<td>6-131</td>
<td>Zero Bit in Register</td>
</tr>
</tbody>
</table>

# Indicates Halfword Instruction
* Indicates Indirect Addressing
### FIXED-POINT ARITHMETIC INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand Format</th>
<th>Op Code</th>
<th>Page</th>
<th>Instruction Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADI</td>
<td>d,v</td>
<td>C801</td>
<td>6-150</td>
<td>Add Immediate</td>
</tr>
<tr>
<td>ADMB</td>
<td>d,*m,x</td>
<td>B808</td>
<td>6-140</td>
<td>Add Memory Byte</td>
</tr>
<tr>
<td>ADMD</td>
<td>d,*m,x</td>
<td>B800</td>
<td>6-143</td>
<td>Add Memory Doubleword</td>
</tr>
<tr>
<td>ADMH</td>
<td>d,*m,x</td>
<td>B800</td>
<td>6-141</td>
<td>Add Memory Halfword</td>
</tr>
<tr>
<td>ADMW</td>
<td>d,*m,x</td>
<td>B800</td>
<td>6-142</td>
<td>Add Memory Word</td>
</tr>
<tr>
<td>#ADR</td>
<td>s,d</td>
<td>3800</td>
<td>6-144</td>
<td>Add Register to Register</td>
</tr>
<tr>
<td>#ADRM</td>
<td>s,d</td>
<td>3808</td>
<td>6-145</td>
<td>Add Register to Register Masked</td>
</tr>
<tr>
<td>ARMB</td>
<td>s,*m,x</td>
<td>E808</td>
<td>6-146</td>
<td>Add Register to Memory Byte</td>
</tr>
<tr>
<td>ARMD</td>
<td>s,*m,x</td>
<td>E800</td>
<td>6-149</td>
<td>Add Register to Memory Doubleword</td>
</tr>
<tr>
<td>ARMH</td>
<td>s,*m,x</td>
<td>E800</td>
<td>6-147</td>
<td>Add Register to Memory Halfword</td>
</tr>
<tr>
<td>ARMW</td>
<td>s,*m,x</td>
<td>E800</td>
<td>6-148</td>
<td>Add Register to Memory Word</td>
</tr>
<tr>
<td>SUI</td>
<td>s,v</td>
<td>C802</td>
<td>6-157</td>
<td>Subtract Immediate</td>
</tr>
<tr>
<td>SUMB</td>
<td>d,*m,x</td>
<td>BC08</td>
<td>6-151</td>
<td>Subtract Memory Byte</td>
</tr>
<tr>
<td>SUMD</td>
<td>d,*m,x</td>
<td>BC00</td>
<td>6-154</td>
<td>Subtract Memory Doubleword</td>
</tr>
<tr>
<td>SUMH</td>
<td>d,*m,x</td>
<td>BC00</td>
<td>6-152</td>
<td>Subtract Memory Halfword</td>
</tr>
<tr>
<td>SUMW</td>
<td>d,*m,x</td>
<td>BC00</td>
<td>6-153</td>
<td>Subtract Memory Word</td>
</tr>
<tr>
<td>#SUR</td>
<td>s,d</td>
<td>3C00</td>
<td>6-155</td>
<td>Subtract Register from Register</td>
</tr>
<tr>
<td>#SURM</td>
<td>s,d</td>
<td>3C08</td>
<td>6-156</td>
<td>Subtract Register from Register Masked</td>
</tr>
<tr>
<td>MPMH</td>
<td>d,*m,x</td>
<td>C000</td>
<td>6-159</td>
<td>Multiply by Memory Halfword</td>
</tr>
<tr>
<td>MPMW</td>
<td>d,*m,x</td>
<td>C000</td>
<td>6-160</td>
<td>Multiply by Memory Word</td>
</tr>
<tr>
<td>#MPR</td>
<td>s,d</td>
<td>4000</td>
<td>6-161</td>
<td>Multiply Register by Register</td>
</tr>
<tr>
<td>MPI</td>
<td>d,v</td>
<td>C803</td>
<td>6-162</td>
<td>Multiply Immediate</td>
</tr>
<tr>
<td>MPMB</td>
<td>d,*m,x</td>
<td>C008</td>
<td>6-158</td>
<td>Multiply by Memory Byte</td>
</tr>
<tr>
<td>DVI</td>
<td>d,v</td>
<td>C804</td>
<td>6-167</td>
<td>Divide Immediate</td>
</tr>
<tr>
<td>DVMB</td>
<td>d,*m,x</td>
<td>C408</td>
<td>6-163</td>
<td>Divide by Memory Byte</td>
</tr>
<tr>
<td>DVMD</td>
<td>d,*m,x</td>
<td>C400</td>
<td>6-164</td>
<td>Divide by Memory Halfword</td>
</tr>
<tr>
<td>DVMW</td>
<td>d,*m,x</td>
<td>C400</td>
<td>6-165</td>
<td>Divide by Memory Word</td>
</tr>
<tr>
<td>#DVR</td>
<td>s,d</td>
<td>4400</td>
<td>6-166</td>
<td>Divide Register by Register</td>
</tr>
<tr>
<td>#ES</td>
<td>d</td>
<td>0004</td>
<td>6-168</td>
<td>Extend Sign</td>
</tr>
<tr>
<td>#RND</td>
<td>d</td>
<td>0005</td>
<td>6-169</td>
<td>Round Register</td>
</tr>
</tbody>
</table>

### FLOATING-POINT ARITHMETIC INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand Format</th>
<th>Op Code</th>
<th>Page</th>
<th>Instruction Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFD</td>
<td>d,*m,x</td>
<td>E008</td>
<td>6-173</td>
<td>Add Floating-Point Doubleword</td>
</tr>
<tr>
<td>AFDW</td>
<td>d,*m,x</td>
<td>E008</td>
<td>6-172</td>
<td>Add Floating-Point Word</td>
</tr>
<tr>
<td>SUFD</td>
<td>d,*m,x</td>
<td>E008</td>
<td>6-175</td>
<td>Subtract Floating-Point Doubleword</td>
</tr>
<tr>
<td>SUFW</td>
<td>d,*m,x</td>
<td>E000</td>
<td>6-174</td>
<td>Subtract Floating-Point Word</td>
</tr>
<tr>
<td>MPFD</td>
<td>d,*m,x</td>
<td>E408</td>
<td>6-177</td>
<td>Multiply Floating-Point Doubleword</td>
</tr>
<tr>
<td>MPFW</td>
<td>d,*m,x</td>
<td>E408</td>
<td>6-176</td>
<td>Multiply Floating-Point Word</td>
</tr>
<tr>
<td>DFD</td>
<td>d,*m,x</td>
<td>E400</td>
<td>6-179</td>
<td>Divide Floating-Point Doubleword</td>
</tr>
<tr>
<td>DVFW</td>
<td>d,*m,x</td>
<td>E400</td>
<td>6-178</td>
<td>Divide Floating-Point Word</td>
</tr>
</tbody>
</table>

# Indicates Halfword Instruction
* Indicates Indirect Addressing
### CONTROL INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand Format</th>
<th>Op Code</th>
<th>Page</th>
<th>Instruction Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRI</td>
<td>*m,x</td>
<td>F900</td>
<td>6-181</td>
<td>Branch and Reset Interrupt</td>
</tr>
<tr>
<td>LPSD</td>
<td>d,*m,x</td>
<td>F980</td>
<td>6-182</td>
<td>Load Program Status Doubleword</td>
</tr>
<tr>
<td>LPSDCM</td>
<td>d,*m,x</td>
<td>FA80</td>
<td>6-183</td>
<td>Load Program Status Doubleword and Change Map</td>
</tr>
<tr>
<td>#CALM</td>
<td>v</td>
<td>3000</td>
<td>6-192</td>
<td>Call Monitor</td>
</tr>
<tr>
<td>DAE</td>
<td></td>
<td>000E</td>
<td>6-198</td>
<td>Disable Arithmetic Exception Trap</td>
</tr>
<tr>
<td>EAE</td>
<td></td>
<td>0008</td>
<td>6-197</td>
<td>Enable Arithmetic Exception Trap</td>
</tr>
<tr>
<td>EXM</td>
<td>*m,x</td>
<td>A800</td>
<td>6-187</td>
<td>Execute Memory</td>
</tr>
<tr>
<td>EXR</td>
<td>s</td>
<td>C807</td>
<td>6-185</td>
<td>Execute Register</td>
</tr>
<tr>
<td>EXRR</td>
<td>s</td>
<td>C807</td>
<td>6-186</td>
<td>Execute Register Right</td>
</tr>
<tr>
<td>#HALT</td>
<td></td>
<td>0000</td>
<td>6-188</td>
<td>Halt</td>
</tr>
<tr>
<td>#LCS</td>
<td></td>
<td>0003</td>
<td>6-184</td>
<td>Load Control Switches</td>
</tr>
<tr>
<td>#NOP</td>
<td></td>
<td>0002</td>
<td>6-190</td>
<td>No Operation</td>
</tr>
<tr>
<td>RDSTS</td>
<td>d</td>
<td>0009</td>
<td>6-195</td>
<td>Read CPU Status Word</td>
</tr>
<tr>
<td>SVC</td>
<td>IND,CALL#</td>
<td>C806</td>
<td>6-193</td>
<td>Supervisor Call</td>
</tr>
<tr>
<td>#SIPU</td>
<td></td>
<td>000A</td>
<td>6-191</td>
<td>Signal IPU</td>
</tr>
<tr>
<td>#SETCPU</td>
<td>s</td>
<td>2C09</td>
<td>6-194</td>
<td>Set CPU Mode</td>
</tr>
<tr>
<td>#WAIT</td>
<td></td>
<td>0001</td>
<td>6-189</td>
<td>Wait</td>
</tr>
</tbody>
</table>

### INTERRUPT INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand Format</th>
<th>Op Code</th>
<th>Page</th>
<th>Instruction Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACI</td>
<td>s,v</td>
<td>FC77</td>
<td>6-207</td>
<td>Activate Channel Interrupt</td>
</tr>
<tr>
<td>AI</td>
<td>v</td>
<td>FC03</td>
<td>6-204</td>
<td>Activate Interrupt</td>
</tr>
<tr>
<td>#BEI</td>
<td></td>
<td>0006</td>
<td>6-211</td>
<td>Block External Interrupts</td>
</tr>
<tr>
<td>DACI</td>
<td>s,v</td>
<td>FC7F</td>
<td>6-210</td>
<td>Deactivate Channel Interrupt</td>
</tr>
<tr>
<td>DAI</td>
<td>v</td>
<td>FC04</td>
<td>6-206</td>
<td>Deactivate Interrupt</td>
</tr>
<tr>
<td>DCI</td>
<td>s,v</td>
<td>FC6F</td>
<td>6-209</td>
<td>Disable Channel Interrupt</td>
</tr>
<tr>
<td>DI</td>
<td>v</td>
<td>FC01</td>
<td>6-205</td>
<td>Disable Interrupt</td>
</tr>
<tr>
<td>ECI</td>
<td>s,v</td>
<td>FC67</td>
<td>6-208</td>
<td>Enable Channel Interrupt</td>
</tr>
<tr>
<td>EI</td>
<td>v</td>
<td>FC00</td>
<td>6-202</td>
<td>Enable Interrupt</td>
</tr>
<tr>
<td>RI</td>
<td>v</td>
<td>FC02</td>
<td>6-203</td>
<td>Request Interrupt</td>
</tr>
<tr>
<td>#UEI</td>
<td></td>
<td>0007</td>
<td>6-212</td>
<td>Unblock External Interrupts</td>
</tr>
</tbody>
</table>

### INPUT/OUTPUT INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand Format</th>
<th>Op Code</th>
<th>Page</th>
<th>Instruction Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD</td>
<td>n,f</td>
<td>FC06</td>
<td>6-216</td>
<td>Command Device</td>
</tr>
<tr>
<td>TD</td>
<td>n,f</td>
<td>FC05</td>
<td>6-217</td>
<td>Test Device</td>
</tr>
<tr>
<td>SIO</td>
<td>s,v</td>
<td>FC17</td>
<td>6-218</td>
<td>Start I/O</td>
</tr>
<tr>
<td>TIO</td>
<td>s,v</td>
<td>FC1F</td>
<td>6-219</td>
<td>Test I/O</td>
</tr>
<tr>
<td>STPIO</td>
<td>s,v</td>
<td>FC27</td>
<td>6-220</td>
<td>Stop I/O</td>
</tr>
<tr>
<td>RSCHNL</td>
<td>s,v</td>
<td>FC2F</td>
<td>6-221</td>
<td>Reset Channel</td>
</tr>
<tr>
<td>HIO</td>
<td>s,v</td>
<td>FC37</td>
<td>6-222</td>
<td>Halt I/O</td>
</tr>
<tr>
<td>GR10</td>
<td>s,v</td>
<td>FC3F</td>
<td>6-223</td>
<td>Grab Controller</td>
</tr>
<tr>
<td>RSCTL</td>
<td>s,v</td>
<td>FC47</td>
<td>6-224</td>
<td>Reset Controller</td>
</tr>
<tr>
<td>ECWCS</td>
<td>s,v</td>
<td>FC4F</td>
<td>6-225</td>
<td>Enable Channel WCS Load</td>
</tr>
<tr>
<td>WCWCS</td>
<td>s,v</td>
<td>FC5F</td>
<td>6-226</td>
<td>Write Channel WCS</td>
</tr>
</tbody>
</table>

### WRITABLE CONTROL STORAGE INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand Format</th>
<th>Op Code</th>
<th>Page</th>
<th>Instruction Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>#WWCS</td>
<td>s,d</td>
<td>000C</td>
<td>6-65</td>
<td>Write WCS</td>
</tr>
<tr>
<td>#RWCS</td>
<td>s,d</td>
<td>000B</td>
<td>6-66</td>
<td>Read WCS</td>
</tr>
<tr>
<td>#JWCS</td>
<td>*m,x</td>
<td>FA00</td>
<td>6-67</td>
<td>Jump WCS</td>
</tr>
</tbody>
</table>

# Indicates Halfword Instruction
* Indicates Indirect Addressing
The following table contains the necessary information for direct conversion of decimal and hexadecimal numbers in these ranges:

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000 to 01FFF</td>
<td>000000 to 008191</td>
</tr>
</tbody>
</table>

To convert a hexadecimal number to a decimal value, locate all but the last digit of the hexadecimal value in the left-most column of the table, then follow that line of figures to the right to the column under the last digit of the hexadecimal value. At this intersection is the decimal value of the hexadecimal number.

Example: Convert hexadecimal 3EC to decimal.

```
  0 1 2 3 4 5 6 7 8 9 A B C D E F
003E 000932 000963 000994 000966 000987 000988 000989 000990 001000 001001 001002 001003  
```

Answer = 001004 decimal

For decimal to hexadecimal conversion as in the example, first find the decimal value (1004) in the table, then construct the hexadecimal value from the hexadecimal characters above the column and in the left-most column.

For numbers outside the range of the table, add the following values to the table figures:

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000</td>
<td>12288</td>
</tr>
<tr>
<td>4000</td>
<td>16384</td>
</tr>
<tr>
<td>5000</td>
<td>20480</td>
</tr>
<tr>
<td>6000</td>
<td>24576</td>
</tr>
<tr>
<td>7000</td>
<td>28672</td>
</tr>
<tr>
<td>8000</td>
<td>32768</td>
</tr>
<tr>
<td>9000</td>
<td>36864</td>
</tr>
<tr>
<td>A000</td>
<td>40960</td>
</tr>
<tr>
<td>B000</td>
<td>45056</td>
</tr>
<tr>
<td>C000</td>
<td>49152</td>
</tr>
<tr>
<td>D000</td>
<td>52248</td>
</tr>
<tr>
<td>E000</td>
<td>57344</td>
</tr>
<tr>
<td>F000</td>
<td>61440</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>000</td>
<td>0000</td>
</tr>
<tr>
<td>001</td>
<td>0005</td>
</tr>
<tr>
<td>002</td>
<td>000A</td>
</tr>
<tr>
<td>003</td>
<td>000F</td>
</tr>
<tr>
<td>004</td>
<td>0014</td>
</tr>
<tr>
<td>005</td>
<td>0019</td>
</tr>
<tr>
<td>006</td>
<td>001E</td>
</tr>
<tr>
<td>007</td>
<td>0023</td>
</tr>
<tr>
<td>008</td>
<td>0028</td>
</tr>
<tr>
<td>009</td>
<td>002D</td>
</tr>
</tbody>
</table>

HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0002</td>
</tr>
<tr>
<td>3</td>
<td>0003</td>
</tr>
<tr>
<td>4</td>
<td>0004</td>
</tr>
<tr>
<td>5</td>
<td>0005</td>
</tr>
<tr>
<td>6</td>
<td>0006</td>
</tr>
<tr>
<td>7</td>
<td>0007</td>
</tr>
<tr>
<td>8</td>
<td>0008</td>
</tr>
<tr>
<td>9</td>
<td>0009</td>
</tr>
<tr>
<td>10</td>
<td>000A</td>
</tr>
<tr>
<td>11</td>
<td>000B</td>
</tr>
<tr>
<td>12</td>
<td>000C</td>
</tr>
<tr>
<td>13</td>
<td>000D</td>
</tr>
<tr>
<td>14</td>
<td>000E</td>
</tr>
<tr>
<td>15</td>
<td>000F</td>
</tr>
<tr>
<td>16</td>
<td>0010</td>
</tr>
<tr>
<td>17</td>
<td>0011</td>
</tr>
<tr>
<td>18</td>
<td>0012</td>
</tr>
<tr>
<td>19</td>
<td>0013</td>
</tr>
<tr>
<td>20</td>
<td>0014</td>
</tr>
<tr>
<td>21</td>
<td>0015</td>
</tr>
<tr>
<td>22</td>
<td>0016</td>
</tr>
<tr>
<td>23</td>
<td>0017</td>
</tr>
<tr>
<td>24</td>
<td>0018</td>
</tr>
<tr>
<td>25</td>
<td>0019</td>
</tr>
<tr>
<td>26</td>
<td>001A</td>
</tr>
<tr>
<td>27</td>
<td>001B</td>
</tr>
<tr>
<td>28</td>
<td>001C</td>
</tr>
<tr>
<td>29</td>
<td>001D</td>
</tr>
<tr>
<td>30</td>
<td>001E</td>
</tr>
<tr>
<td>31</td>
<td>001F</td>
</tr>
<tr>
<td>32</td>
<td>0020</td>
</tr>
<tr>
<td>33</td>
<td>0021</td>
</tr>
<tr>
<td>34</td>
<td>0022</td>
</tr>
<tr>
<td>35</td>
<td>0023</td>
</tr>
<tr>
<td>36</td>
<td>0024</td>
</tr>
<tr>
<td>37</td>
<td>0025</td>
</tr>
<tr>
<td>38</td>
<td>0026</td>
</tr>
<tr>
<td>39</td>
<td>0027</td>
</tr>
<tr>
<td>40</td>
<td>0028</td>
</tr>
<tr>
<td>41</td>
<td>0029</td>
</tr>
<tr>
<td>42</td>
<td>002A</td>
</tr>
<tr>
<td>43</td>
<td>002B</td>
</tr>
<tr>
<td>44</td>
<td>002C</td>
</tr>
<tr>
<td>45</td>
<td>002D</td>
</tr>
<tr>
<td>46</td>
<td>002E</td>
</tr>
<tr>
<td>47</td>
<td>002F</td>
</tr>
<tr>
<td>48</td>
<td>0030</td>
</tr>
<tr>
<td>49</td>
<td>0031</td>
</tr>
<tr>
<td>50</td>
<td>0032</td>
</tr>
<tr>
<td>51</td>
<td>0033</td>
</tr>
<tr>
<td>52</td>
<td>0034</td>
</tr>
<tr>
<td>53</td>
<td>0035</td>
</tr>
<tr>
<td>54</td>
<td>0036</td>
</tr>
<tr>
<td>55</td>
<td>0037</td>
</tr>
<tr>
<td>56</td>
<td>0038</td>
</tr>
<tr>
<td>57</td>
<td>0039</td>
</tr>
<tr>
<td>58</td>
<td>003A</td>
</tr>
<tr>
<td>59</td>
<td>003B</td>
</tr>
<tr>
<td>60</td>
<td>003C</td>
</tr>
<tr>
<td>61</td>
<td>003D</td>
</tr>
<tr>
<td>62</td>
<td>003E</td>
</tr>
<tr>
<td>63</td>
<td>003F</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>0200</td>
<td>0189</td>
</tr>
<tr>
<td>0210</td>
<td>01E0</td>
</tr>
<tr>
<td>0220</td>
<td>01F1</td>
</tr>
</tbody>
</table>

HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE (Cont'd)
### HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE (Cont'd)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0A</td>
<td>02090</td>
<td>020B</td>
<td>020D</td>
<td>020F</td>
<td>0210</td>
</tr>
<tr>
<td>0B</td>
<td>0211</td>
<td>0214</td>
<td>0217</td>
<td>0219</td>
<td>021C</td>
</tr>
<tr>
<td>0C</td>
<td>021E</td>
<td>0221</td>
<td>0224</td>
<td>0227</td>
<td>0229</td>
</tr>
<tr>
<td>0D</td>
<td>022B</td>
<td>022E</td>
<td>0231</td>
<td>0234</td>
<td>0237</td>
</tr>
<tr>
<td>0E</td>
<td>0239</td>
<td>023B</td>
<td>023E</td>
<td>0241</td>
<td>0244</td>
</tr>
<tr>
<td>0F</td>
<td>0246</td>
<td>0249</td>
<td>024B</td>
<td>024E</td>
<td>0251</td>
</tr>
<tr>
<td>10</td>
<td>0253</td>
<td>0256</td>
<td>0259</td>
<td>025B</td>
<td>025E</td>
</tr>
<tr>
<td>11</td>
<td>0261</td>
<td>0264</td>
<td>0267</td>
<td>0269</td>
<td>026C</td>
</tr>
<tr>
<td>12</td>
<td>026E</td>
<td>0271</td>
<td>0274</td>
<td>0277</td>
<td>0279</td>
</tr>
<tr>
<td>13</td>
<td>027B</td>
<td>027E</td>
<td>0281</td>
<td>0284</td>
<td>0287</td>
</tr>
<tr>
<td>14</td>
<td>0289</td>
<td>028B</td>
<td>028E</td>
<td>0291</td>
<td>0294</td>
</tr>
<tr>
<td>15</td>
<td>0296</td>
<td>0299</td>
<td>029C</td>
<td>029F</td>
<td>02A2</td>
</tr>
<tr>
<td>16</td>
<td>02A4</td>
<td>02A7</td>
<td>02A9</td>
<td>02AB</td>
<td>02AD</td>
</tr>
<tr>
<td>17</td>
<td>02AE</td>
<td>02B1</td>
<td>02B4</td>
<td>02B6</td>
<td>02B9</td>
</tr>
<tr>
<td>18</td>
<td>02BB</td>
<td>02BD</td>
<td>02BF</td>
<td>02C2</td>
<td>02C5</td>
</tr>
<tr>
<td>19</td>
<td>02C7</td>
<td>02CE</td>
<td>02D0</td>
<td>02D3</td>
<td>02D6</td>
</tr>
<tr>
<td>1A</td>
<td>02D8</td>
<td>02DA</td>
<td>02DB</td>
<td>02DE</td>
<td>02E0</td>
</tr>
<tr>
<td>1B</td>
<td>02E2</td>
<td>02E4</td>
<td>02E6</td>
<td>02E8</td>
<td>02E9</td>
</tr>
<tr>
<td>1C</td>
<td>02EA</td>
<td>02EB</td>
<td>02EC</td>
<td>02ED</td>
<td>02EE</td>
</tr>
<tr>
<td>1D</td>
<td>02EF</td>
<td>02F0</td>
<td>02F1</td>
<td>02F2</td>
<td>02F3</td>
</tr>
<tr>
<td>1E</td>
<td>02F4</td>
<td>02F5</td>
<td>02F6</td>
<td>02F7</td>
<td>02F8</td>
</tr>
<tr>
<td>1F</td>
<td>02F9</td>
<td>02FA</td>
<td>02FB</td>
<td>02FC</td>
<td>02FD</td>
</tr>
</tbody>
</table>

**Notes:**
- The table continues in this format, with rows and columns corresponding to hexadecimal (A-F) and decimal (0-15) values.
- Each cell represents a conversion from hexadecimal to its decimal equivalent.
- The table is extended to cover a wide range of hexadecimal values.

**Example Conversion:**
- Converting 0x1A from hexadecimal to decimal:
  - 0x1A in hexadecimal is equivalent to 26 in decimal.

This table is useful for understanding the relationship between hexadecimal and decimal number systems, which are fundamental in computer science and digital electronics.
| G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 000 | 0000 | 0001 | 0002 | 0003 | 0004 | 0005 | 0006 | 0007 | 0008 | 0009 | 000A | 000B | 000C | 000D | 000E | 000F | 0010 | 0011 | 0012 |
| 001 | 0013 | 0014 | 0015 | 0016 | 0017 | 0018 | 0019 | 001A | 001B | 001C | 001D | 001E | 001F | 0020 | 0021 | 0022 | 0023 | 0024 | 0025 |
| 002 | 0026 | 0027 | 0028 | 0029 | 002A | 002B | 002C | 002D | 002E | 002F | 0030 | 0031 | 0032 | 0033 | 0034 | 0035 | 0036 | 0037 | 0038 |
| 003 | 0039 | 003A | 003B | 003C | 003D | 003E | 003F | 0040 | 0041 | 0042 | 0043 | 0044 | 0045 | 0046 | 0047 | 0048 | 0049 | 004A | 004B |
| 004 | 004C | 004D | 004E | 004F | 0050 | 0051 | 0052 | 0053 | 0054 | 0055 | 0056 | 0057 | 0058 | 0059 | 005A | 005B | 005C | 005D | 005E |
| 005 | 005F | 0060 | 0061 | 0062 | 0063 | 0064 | 0065 | 0066 | 0067 | 0068 | 0069 | 006A | 006B | 006C | 006D | 006E | 006F | 0070 | 0071 |
| 007 | 0072 | 0073 | 0074 | 0075 | 0076 | 0077 | 0078 | 0079 | 007A | 007B | 007C | 007D | 007E | 007F | 0080 | 0081 | 0082 | 0083 | 0084 |
| 008 | 0085 | 0086 | 0087 | 0088 | 0089 | 008A | 008B | 008C | 008D | 008E | 008F | 0090 | 0091 | 0092 | 0093 | 0094 | 0095 | 0096 | 0097 |
| 009 | 0098 | 0099 | 009A | 009B | 009C | 009D | 009E | 009F | 00A0 | 00A1 | 00A2 | 00A3 | 00A4 | 00A5 | 00A6 | 00A7 | 00A8 | 00A9 | 00AA |
| 00B | 00AB | 00AC | 00AD | 00AE | 00AF | 00B0 | 00B1 | 00B2 | 00B3 | 00B4 | 00B5 | 00B6 | 00B7 | 00B8 | 00B9 | 00BA | 00BB | 00BC | 00BD |
| 00BE | 00BF | 00C0 | 00C1 | 00C2 | 00C3 | 00C4 | 00C5 | 00C6 | 00C7 | 00C8 | 00C9 | 00CA | 00CB | 00CC | 00CD | 00CE | 00CF | 00D0 |
| 00D1 | 00D2 | 00D3 | 00D4 | 00D5 | 00D6 | 00D7 | 00D8 | 00D9 | 00DA | 00DB | 00DC | 00DD | 00DE | 00DF | 00E0 | 00E1 | 00E2 | 00E3 |
| 00E4 | 00E5 | 00E6 | 00E7 | 00E8 | 00E9 | 00EA | 00EB | 00EC | 00ED | 00EE | 00EF | 00F0 | 00F1 | 00F2 | 00F3 | 00F4 | 00F5 | 00F6 |
| 00F7 | 00F8 | 00F9 | 00FA | 00FB | 00FC | 00FD | 00FE | 00FF | 0100 | 0101 | 0102 | 0103 | 0104 | 0105 | 0106 | 0107 | 0108 | 0109 |
| 010A | 010B | 010C | 010D | 010E | 010F | 0110 | 0111 | 0112 | 0113 | 0114 | 0115 | 0116 | 0117 | 0118 | 0119 | 011A | 011B | 011C |
| 011D | 011E | 011F | 0120 | 0121 | 0122 | 0123 | 0124 | 0125 | 0126 | 0127 | 0128 | 0129 | 012A | 012B | 012C | 012D | 012E | 012F |

**HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE (Cont'd)**
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>B</td>
<td>6</td>
<td>2</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>E</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>20</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>25</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>30</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>35</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>35</td>
<td>40</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>45</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>45</td>
<td>50</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>55</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>55</td>
<td>60</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>65</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>65</td>
<td>70</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>70</td>
<td>75</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>75</td>
<td>80</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>85</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>85</td>
<td>90</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>90</td>
<td>95</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>95</td>
<td>100</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

HEXADECIMAL-DECIMAL NUMBER CONVERSION Table (Cont'd)
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>001001</td>
<td>011001</td>
<td>010100</td>
<td>001001</td>
<td>011001</td>
</tr>
<tr>
<td>0101</td>
<td>001000</td>
<td>011000</td>
<td>010100</td>
<td>001001</td>
<td>011001</td>
</tr>
<tr>
<td>0102</td>
<td>001000</td>
<td>011000</td>
<td>010100</td>
<td>001001</td>
<td>011001</td>
</tr>
<tr>
<td>0103</td>
<td>001000</td>
<td>011000</td>
<td>010100</td>
<td>001001</td>
<td>011001</td>
</tr>
<tr>
<td>0104</td>
<td>001000</td>
<td>011000</td>
<td>010100</td>
<td>001001</td>
<td>011001</td>
</tr>
<tr>
<td>0105</td>
<td>001000</td>
<td>011000</td>
<td>010100</td>
<td>001001</td>
<td>011001</td>
</tr>
<tr>
<td>0106</td>
<td>001000</td>
<td>011000</td>
<td>010100</td>
<td>001001</td>
<td>011001</td>
</tr>
<tr>
<td>0107</td>
<td>001000</td>
<td>011000</td>
<td>010100</td>
<td>001001</td>
<td>011001</td>
</tr>
<tr>
<td>0108</td>
<td>001000</td>
<td>011000</td>
<td>010100</td>
<td>001001</td>
<td>011001</td>
</tr>
<tr>
<td>0109</td>
<td>001000</td>
<td>011000</td>
<td>010100</td>
<td>001001</td>
<td>011001</td>
</tr>
<tr>
<td>010A</td>
<td>001000</td>
<td>011000</td>
<td>010100</td>
<td>001001</td>
<td>011001</td>
</tr>
<tr>
<td>010B</td>
<td>001000</td>
<td>011000</td>
<td>010100</td>
<td>001001</td>
<td>011001</td>
</tr>
<tr>
<td>010C</td>
<td>001000</td>
<td>011000</td>
<td>010100</td>
<td>001001</td>
<td>011001</td>
</tr>
<tr>
<td>010D</td>
<td>001000</td>
<td>011000</td>
<td>010100</td>
<td>001001</td>
<td>011001</td>
</tr>
<tr>
<td>010E</td>
<td>001000</td>
<td>011000</td>
<td>010100</td>
<td>001001</td>
<td>011001</td>
</tr>
<tr>
<td>010F</td>
<td>001000</td>
<td>011000</td>
<td>010100</td>
<td>001001</td>
<td>011001</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>G1F</td>
<td>01F92</td>
<td>01F98</td>
<td>01F9B</td>
<td>01F9F</td>
<td>01FA3</td>
</tr>
<tr>
<td>G1E</td>
<td>01FA7</td>
<td>01FAE</td>
<td>01FB0</td>
<td>01FB4</td>
<td>01FB7</td>
</tr>
<tr>
<td>G1D</td>
<td>01FC3</td>
<td>01FC6</td>
<td>01FC8</td>
<td>01FCC</td>
<td>01FD0</td>
</tr>
<tr>
<td>G1C</td>
<td>01FD4</td>
<td>01FD6</td>
<td>01FD8</td>
<td>01FD9</td>
<td>01FDB</td>
</tr>
<tr>
<td>G1B</td>
<td>01FDD</td>
<td>01FDE</td>
<td>01FDF</td>
<td>01FE0</td>
<td>01FE1</td>
</tr>
<tr>
<td>G1A</td>
<td>01FE3</td>
<td>01FE4</td>
<td>01FE5</td>
<td>01FE6</td>
<td>01FE7</td>
</tr>
<tr>
<td>G0F</td>
<td>01FE9</td>
<td>01FEA</td>
<td>01FEB</td>
<td>01FEC</td>
<td>01FED</td>
</tr>
<tr>
<td>G0E</td>
<td>01FEF</td>
<td>01FF0</td>
<td>01FF1</td>
<td>01FF2</td>
<td>01FF3</td>
</tr>
<tr>
<td>G0D</td>
<td>01FF5</td>
<td>01FF6</td>
<td>01FF7</td>
<td>01FF8</td>
<td>01FF9</td>
</tr>
<tr>
<td>G0C</td>
<td>01FFB</td>
<td>01FFC</td>
<td>01FFD</td>
<td>01FFE</td>
<td>01FFF</td>
</tr>
<tr>
<td>G0B</td>
<td>01G01</td>
<td>01G02</td>
<td>01G03</td>
<td>01G04</td>
<td>01G05</td>
</tr>
<tr>
<td>G0A</td>
<td>01G07</td>
<td>01G08</td>
<td>01G09</td>
<td>01G0A</td>
<td>01G0B</td>
</tr>
<tr>
<td>FGF</td>
<td>01G0D</td>
<td>01G0E</td>
<td>01G0F</td>
<td>01G10</td>
<td>01G11</td>
</tr>
<tr>
<td>FGE</td>
<td>01G13</td>
<td>01G14</td>
<td>01G15</td>
<td>01G16</td>
<td>01G17</td>
</tr>
<tr>
<td>FGD</td>
<td>01G19</td>
<td>01G1A</td>
<td>01G1B</td>
<td>01G1C</td>
<td>01G1D</td>
</tr>
<tr>
<td>FGC</td>
<td>01G1F</td>
<td>01G20</td>
<td>01G21</td>
<td>01G22</td>
<td>01G23</td>
</tr>
<tr>
<td>FGB</td>
<td>01G25</td>
<td>01G26</td>
<td>01G27</td>
<td>01G28</td>
<td>01G29</td>
</tr>
<tr>
<td>FGA</td>
<td>01G2B</td>
<td>01G2C</td>
<td>01G2D</td>
<td>01G2E</td>
<td>01G2F</td>
</tr>
<tr>
<td>F0F</td>
<td>01G31</td>
<td>01G32</td>
<td>01G33</td>
<td>01G34</td>
<td>01G35</td>
</tr>
<tr>
<td>F0E</td>
<td>01G37</td>
<td>01G38</td>
<td>01G39</td>
<td>01G3A</td>
<td>01G3B</td>
</tr>
<tr>
<td>F0D</td>
<td>01G3D</td>
<td>01G3E</td>
<td>01G3F</td>
<td>01G40</td>
<td>01G41</td>
</tr>
<tr>
<td>F0C</td>
<td>01G43</td>
<td>01G44</td>
<td>01G45</td>
<td>01G46</td>
<td>01G47</td>
</tr>
<tr>
<td>F0B</td>
<td>01G49</td>
<td>01G4A</td>
<td>01G4B</td>
<td>01G4C</td>
<td>01G4D</td>
</tr>
<tr>
<td>F0A</td>
<td>01G4F</td>
<td>01G50</td>
<td>01G51</td>
<td>01G52</td>
<td>01G53</td>
</tr>
<tr>
<td>F1F</td>
<td>01G55</td>
<td>01G56</td>
<td>01G57</td>
<td>01G58</td>
<td>01G59</td>
</tr>
<tr>
<td>F1E</td>
<td>01G5B</td>
<td>01G5C</td>
<td>01G5D</td>
<td>01G5E</td>
<td>01G5F</td>
</tr>
<tr>
<td>F1D</td>
<td>01G61</td>
<td>01G62</td>
<td>01G63</td>
<td>01G64</td>
<td>01G65</td>
</tr>
<tr>
<td>F1C</td>
<td>01G67</td>
<td>01G68</td>
<td>01G69</td>
<td>01G6A</td>
<td>01G6B</td>
</tr>
<tr>
<td>F1B</td>
<td>01G6D</td>
<td>01G6E</td>
<td>01G6F</td>
<td>01G70</td>
<td>01G71</td>
</tr>
<tr>
<td>F1A</td>
<td>01G73</td>
<td>01G74</td>
<td>01G75</td>
<td>01G76</td>
<td>01G77</td>
</tr>
<tr>
<td>F0F</td>
<td>01G79</td>
<td>01G7A</td>
<td>01G7B</td>
<td>01G7C</td>
<td>01G7D</td>
</tr>
<tr>
<td>E0F</td>
<td>01G7F</td>
<td>01G80</td>
<td>01G81</td>
<td>01G82</td>
<td>01G83</td>
</tr>
<tr>
<td>E0E</td>
<td>01G85</td>
<td>01G86</td>
<td>01G87</td>
<td>01G88</td>
<td>01G89</td>
</tr>
<tr>
<td>E0D</td>
<td>01G8B</td>
<td>01G8C</td>
<td>01G8D</td>
<td>01G8E</td>
<td>01G8F</td>
</tr>
<tr>
<td>E0C</td>
<td>01G91</td>
<td>01G92</td>
<td>01G93</td>
<td>01G94</td>
<td>01G95</td>
</tr>
<tr>
<td>E0B</td>
<td>01G97</td>
<td>01G98</td>
<td>01G99</td>
<td>01G9A</td>
<td>01G9B</td>
</tr>
<tr>
<td>E0A</td>
<td>01G9D</td>
<td>01G9E</td>
<td>01G9F</td>
<td>01GA0</td>
<td>01GA1</td>
</tr>
<tr>
<td>E1F</td>
<td>01GA3</td>
<td>01GA4</td>
<td>01GA5</td>
<td>01GA6</td>
<td>01GA7</td>
</tr>
<tr>
<td>E1E</td>
<td>01GA9</td>
<td>01GAA</td>
<td>01GAB</td>
<td>01GAC</td>
<td>01GAD</td>
</tr>
<tr>
<td>E1D</td>
<td>01GAF</td>
<td>01GB0</td>
<td>01GB1</td>
<td>01GB2</td>
<td>01GB3</td>
</tr>
<tr>
<td>E1C</td>
<td>01GB5</td>
<td>01GB6</td>
<td>01GB7</td>
<td>01GB8</td>
<td>01GB9</td>
</tr>
<tr>
<td>E1B</td>
<td>01GBB</td>
<td>01GBC</td>
<td>01GBD</td>
<td>01GBE</td>
<td>01GBF</td>
</tr>
<tr>
<td>E1A</td>
<td>01GCH</td>
<td>01GCI</td>
<td>01GCJ</td>
<td>01GCK</td>
<td>01GCL</td>
</tr>
<tr>
<td>E0F</td>
<td>01GCD</td>
<td>01GCE</td>
<td>01GCF</td>
<td>01GG0</td>
<td>01GG1</td>
</tr>
<tr>
<td>D0F</td>
<td>01GG3</td>
<td>01GG4</td>
<td>01GG5</td>
<td>01GG6</td>
<td>01GG7</td>
</tr>
<tr>
<td>D0E</td>
<td>01GG9</td>
<td>01GGA</td>
<td>01GGB</td>
<td>01GGC</td>
<td>01GGD</td>
</tr>
<tr>
<td>D0D</td>
<td>01GGF</td>
<td>01GHI</td>
<td>01GJJ</td>
<td>01GJK</td>
<td>01GJL</td>
</tr>
<tr>
<td>D0C</td>
<td>01GN0</td>
<td>01GNO</td>
<td>01GNN</td>
<td>01GNO</td>
<td>01GNS</td>
</tr>
<tr>
<td>D0B</td>
<td>01GNU</td>
<td>01GNU</td>
<td>01GNV</td>
<td>01GNW</td>
<td>01GNX</td>
</tr>
<tr>
<td>D0A</td>
<td>01GNZ</td>
<td>01GNZ</td>
<td>01GNZ</td>
<td>01GNZ</td>
<td>01GNZ</td>
</tr>
</tbody>
</table>

HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE (Cont'd)
Converting to hexadecimal may be simplified by using the following table.

To convert \( (61275)_{10} \) to hexadecimal, using the table: the table entry closest to, but not greater than, \( (61275)_{10} \) is \( (61184)_{10} \), which equals \( (EF00)_{16} \) from the table. Subtracting 61,184 from the original number \( (61275-61184)_{10} \) leaves a remainder of \( (91)_{10} \), which equals \( (5B)_{16} \). Therefore, \( (61275)_{10} = (EF5B)_{16} \).
APPENDIX D

HEXADECIMAL ADDITIONS

In the following Hexadecimal Addition Table, all values represent the result of an addition of a hexadecimal character from the column across the top and the column down the left side. The result of the addition is found where the two characters to be added intersect within the table. All values above the slanted line represent the result of an addition with no carry generated; all those values below the slanted line represent the result of an addition with a carry of one generated into the next character position of the hexadecimal result.

<table>
<thead>
<tr>
<th>HEXADECIMAL ADDITION TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 A B C D E F</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8 9 A B C D E F 0</td>
</tr>
<tr>
<td>2 3 4 5 6 7 8 9 A B C D E F 0 1</td>
</tr>
<tr>
<td>3 4 5 6 7 8 9 A B C D E F 0 1 2</td>
</tr>
<tr>
<td>4 5 6 7 8 9 A B C D E F 0 1 2 3</td>
</tr>
<tr>
<td>5 6 7 8 9 A B C D E F 0 1 2 3 4</td>
</tr>
<tr>
<td>6 7 8 9 A B C D E F 0 1 2 3 4 5</td>
</tr>
<tr>
<td>7 8 9 A B C D E F 0 1 2 3 4 5 6</td>
</tr>
<tr>
<td>8 9 A B C D E F 0 1 2 3 4 5 6 7</td>
</tr>
<tr>
<td>9 A B C D E F 0 1 2 3 4 5 6 7 8</td>
</tr>
<tr>
<td>A B C D E F 0 1 2 3 4 5 6 7 8 9</td>
</tr>
<tr>
<td>B C D E F 0 1 2 3 4 5 6 7 8 9 A</td>
</tr>
<tr>
<td>C D E F 0 1 2 3 4 5 6 7 8 9 A B</td>
</tr>
<tr>
<td>D E F 0 1 2 3 4 5 6 7 8 9 A B C</td>
</tr>
<tr>
<td>E F 0 1 2 3 4 5 6 7 8 9 A B C D</td>
</tr>
<tr>
<td>F 0 1 2 3 4 5 6 7 8 9 A B C D E</td>
</tr>
</tbody>
</table>
### Appendix E

#### Numerical Information

<table>
<thead>
<tr>
<th>$n^2$</th>
<th>$2^{1/n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>1.41421</td>
</tr>
<tr>
<td>3</td>
<td>1.73205</td>
</tr>
<tr>
<td>4</td>
<td>2.0000</td>
</tr>
<tr>
<td>5</td>
<td>2.23607</td>
</tr>
<tr>
<td>6</td>
<td>2.44949</td>
</tr>
<tr>
<td>7</td>
<td>2.64575</td>
</tr>
<tr>
<td>8</td>
<td>2.82843</td>
</tr>
<tr>
<td>9</td>
<td>3.0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$2^{n}$</th>
<th>$2^{1/n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>2.0000</td>
</tr>
<tr>
<td>4</td>
<td>2.0000</td>
</tr>
<tr>
<td>8</td>
<td>2.0000</td>
</tr>
<tr>
<td>16</td>
<td>2.0000</td>
</tr>
<tr>
<td>32</td>
<td>2.0000</td>
</tr>
<tr>
<td>64</td>
<td>2.0000</td>
</tr>
<tr>
<td>128</td>
<td>2.0000</td>
</tr>
<tr>
<td>256</td>
<td>2.0000</td>
</tr>
<tr>
<td>512</td>
<td>2.0000</td>
</tr>
</tbody>
</table>

#### Table of Powers of Two

<table>
<thead>
<tr>
<th>$2^{n}$</th>
<th>$2^{1/n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>2.0000</td>
</tr>
<tr>
<td>4</td>
<td>2.0000</td>
</tr>
<tr>
<td>8</td>
<td>2.0000</td>
</tr>
<tr>
<td>16</td>
<td>2.0000</td>
</tr>
<tr>
<td>32</td>
<td>2.0000</td>
</tr>
<tr>
<td>64</td>
<td>2.0000</td>
</tr>
<tr>
<td>128</td>
<td>2.0000</td>
</tr>
<tr>
<td>256</td>
<td>2.0000</td>
</tr>
<tr>
<td>512</td>
<td>2.0000</td>
</tr>
</tbody>
</table>

---

**E-1/E-2**
### APPENDIX F

#### TABLE OF POWERS OF SIXTEEN

<table>
<thead>
<tr>
<th>$16^n$</th>
<th>$n$</th>
<th>$16^{-n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0.10000 0000 0000 0000 0000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.62500 0000 0000 0000 0000</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>0.39062 5000 0000 0000 0000</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.24414 0625 0000 0000 0000</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0.15258 7890 2500 0000 0000</td>
</tr>
<tr>
<td>65</td>
<td>5</td>
<td>0.95367 4316 0625 0000 0000</td>
</tr>
<tr>
<td>1048</td>
<td>6</td>
<td>5.9604 6447 7306 2500 0000</td>
</tr>
<tr>
<td>777</td>
<td>7</td>
<td>5.37252 9029 4619 4062 5000</td>
</tr>
<tr>
<td>4294</td>
<td>8</td>
<td>2.5383 0644 5390 6289 1000</td>
</tr>
<tr>
<td>719</td>
<td>9</td>
<td>0.94551 9999 2814 7007 3000</td>
</tr>
<tr>
<td>1099</td>
<td>10</td>
<td>0.89999 9999 9999 9999 9999</td>
</tr>
<tr>
<td>592</td>
<td>11</td>
<td>0.8189 3478 6467 2816 4096</td>
</tr>
<tr>
<td>474</td>
<td>12</td>
<td>0.7421 0073 4827 3789 8191</td>
</tr>
<tr>
<td>599</td>
<td>13</td>
<td>0.6805 1847 5761 4827 3789</td>
</tr>
<tr>
<td>268</td>
<td>14</td>
<td>0.6280 7816 0000 0000 0000</td>
</tr>
<tr>
<td>152</td>
<td>15</td>
<td>0.5873 0000 0000 0000 0000</td>
</tr>
</tbody>
</table>

#### TABLE OF POWERS OF TEN

<table>
<thead>
<tr>
<th>$10^n$</th>
<th>$n$</th>
<th>$10^{-n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1.0000 0000 0000 0000 0000</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0.1999 9999 9999 9999 9999</td>
</tr>
<tr>
<td>64</td>
<td>2</td>
<td>0.28F5 C28F 5C28 F5C3 16-1</td>
</tr>
<tr>
<td>3E8</td>
<td>3</td>
<td>0.4189 3748 C6A7 EF9E 16-2</td>
</tr>
<tr>
<td>2710</td>
<td>4</td>
<td>0.68DB 8BAC 710C B296 16-3</td>
</tr>
<tr>
<td>86A0</td>
<td>5</td>
<td>0.47C5 AC47 1847 8423 16-4</td>
</tr>
<tr>
<td>4240</td>
<td>6</td>
<td>0.10C6 F7A0 B3ED 8D37 16-4</td>
</tr>
<tr>
<td>98</td>
<td>7</td>
<td>0.1AD7 F29A BCAF 4858 16-5</td>
</tr>
<tr>
<td>5F5 E100</td>
<td>8</td>
<td>0.2AF3 1DC4 6118 73BF 16-6</td>
</tr>
<tr>
<td>389A CA00</td>
<td>9</td>
<td>0.44B8 2FA0 985A 52CC 16-7</td>
</tr>
<tr>
<td>2 540B E400</td>
<td>10</td>
<td>0.6DF3 7F67 5EF6 EADF 16-8</td>
</tr>
<tr>
<td>17 4876 E800</td>
<td>11</td>
<td>0.AFEB FF0B CB24 AAFF 16-9</td>
</tr>
<tr>
<td>8E D4A5 1000</td>
<td>12</td>
<td>0.11C9 9B81 2DEA 1119 16-9</td>
</tr>
<tr>
<td>918 4E72 A000</td>
<td>13</td>
<td>0.1C25 C268 4976 81C2 16-10</td>
</tr>
<tr>
<td>5AF3 107A 4000</td>
<td>14</td>
<td>0.2C09 370D 4257 3604 16-11</td>
</tr>
<tr>
<td>3 8D7E A4C6 8000</td>
<td>15</td>
<td>0.480E BE78 9D58 566D 16-12</td>
</tr>
<tr>
<td>23 86F2 6FC1 0000</td>
<td>16</td>
<td>0.724A CAF5 6226 F0AE 16-13</td>
</tr>
<tr>
<td>163 4578 5D8A 0000</td>
<td>17</td>
<td>0.8677 AA32 36A4 8449 16-14</td>
</tr>
<tr>
<td>DF9 8683 A764 0000</td>
<td>18</td>
<td>0.1272 5D01 D243 ABA1 16-14</td>
</tr>
<tr>
<td>BAC7 2304 89E8 0000</td>
<td>19</td>
<td>0.1D33 C94F 86D2 AC35 16-15</td>
</tr>
</tbody>
</table>
## APPENDIX G

### ASCII INTERCHANGE CODE SET WITH CARD PUNCH CODES

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

<table>
<thead>
<tr>
<th>Row</th>
<th>Col</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>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>NUL</td>
<td>DLE</td>
<td>SP</td>
<td>0</td>
<td>@</td>
<td>P</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
<td>!</td>
<td>1</td>
<td>A</td>
<td>Q</td>
<td>a</td>
<td>q</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>STX</td>
<td>DC2</td>
<td>&quot;</td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>r</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>c</td>
<td>s</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>t</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>ENQ</td>
<td>NAK</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>e</td>
<td>u</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>ACK</td>
<td>SYN</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>v</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>BEL</td>
<td>ETB</td>
<td>.</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>w</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>BS</td>
<td>CAN</td>
<td>(</td>
<td>8</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>x</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>HT</td>
<td>EM</td>
<td>)</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>y</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
<td>LF</td>
<td>SUB</td>
<td>^</td>
<td>5</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
<td>VT</td>
<td>ESC</td>
<td>+</td>
<td>5</td>
<td>K</td>
<td>l</td>
<td>k</td>
<td>l</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
<td>FF</td>
<td>FS</td>
<td>L</td>
<td>0</td>
<td>8</td>
<td>+</td>
<td>l</td>
<td>l</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
<td>CR</td>
<td>GS</td>
<td>=</td>
<td>0</td>
<td>8</td>
<td>-</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
<td>SO</td>
<td>RS</td>
<td>&gt;</td>
<td>0</td>
<td>N</td>
<td>A</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
<td>SI</td>
<td>US</td>
<td>/</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>o</td>
<td>DEL</td>
</tr>
</tbody>
</table>

G-1
Some positions in the ASCII code chart may have a different graphic representation on various devices as:

<table>
<thead>
<tr>
<th>ASCII</th>
<th>IBM 029</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Control Characters:

- **NUL** – Null
- **SOH** – Start of Heading (CC)
- **STX** – Start of Text (CC)
- **ETX** – End of Text (CC)
- **EOT** – End of Transmission (CC)
- **ENQ** – Enquiry (CC)
- **ACK** – Acknowledge (CC)
- **BEL** – Bell (audible or attention signal)
- **BS** – Backspace (FE)
- **HT** – Horizontal Tabulation (punch card skip) (FE)
- **LF** – Line Feed (FE)
- **VT** – Vertical Tabulation (FE)
- **FF** – Form Feed (FE)
- **CR** – Carriage Return (FE)
- **SO** – Shift Out
- **SI** – Shift In
- **DLE** – Data Link Escape (CC)
- **DC1** – Device Control 1
- **DC2** – Device Control 2
- **DC3** – Device Control 3
- **DC4** – Device Control 4 (stop)
- **NAK** – Negative Acknowledge (CC)
- **SYN** – Synchronous Idle (CC)
- **ETB** – End of Transmission Block (CC)
- **CAN** – Cancel
- **EM** – End of Medium
- **SS** – Start of Special Sequence
- **ESC** – Escape
- **FS** – File Separator (IS)
- **GS** – Group Separator (IS)
- **RS** – Record Separator (IS)
- **US** – Unit Separator (IS)
- **DEL** – Delete
- **SP** – Space (normally nonprinting)
- **(CC)** – Communication Control
- **(FE)** – Format Effector
- **(IS)** – Information Separator
Reader's Comment Form

Date ____________

Manual Title: _______________________________________________________________
Publication Number __________________________________________________________

- How do you use this publication?
  - [ ] As an introduction to the subject.
  - [ ] Emergency Maintenance.
  - [ ] Other ____________________________

- Is the material:
  - Easy to read? ____________________________________________ [ ] Yes [ ] No
  - Well organized? __________________________________________ [ ] Yes [ ] No
  - Complete? ______________________________________________ [ ] Yes [ ] No
  - Well illustrated? _________________________________________ [ ] Yes [ ] No
  - Accurate? _______________________________________________ [ ] Yes [ ] No
  - Suitable for its intended use? ______________________________ [ ] Yes [ ] No

- Please check the items that describe your position:
  - [ ] Customer Personnel
  - [ ] SYSTEMS Personnel
  - [ ] Engineer
  - [ ] Sales Representative
  - [ ] Technician
  - [ ] Field Service
  - [ ] Operator
  - [ ] Programmer
  - [ ] Instructor
  - [ ] Trainee
  - [ ] Other ____________________________

- Please check specific criticism(s), give page number(s), and explain below:
  - [ ] Clarification on page(s) ______________
  - [ ] Deletion on page(s) ______________
  - [ ] Addition on page(s) ______________
  - [ ] Error on page(s) ______________

  Explanation:

Your Name: _______________________________________________________________
Your Company: ____________________________________________________________
May we have your comments?

This publication is one of a series of SYSTEMS Engineering Laboratories technical documents written to serve each of a wide variety of users. Your completion of the attached form will aid SYSTEMS in the continued production of complete, easily referenced material in each of these various technical publications.

Thank You