## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Paragraph No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Introduction</td>
<td>1-1</td>
</tr>
<tr>
<td>1.2</td>
<td>Features</td>
<td>1-1</td>
</tr>
<tr>
<td>1.3</td>
<td>Software Features</td>
<td>1-2</td>
</tr>
<tr>
<td>1.4</td>
<td>Programming Model</td>
<td>1-3</td>
</tr>
<tr>
<td>1.5</td>
<td>Index Registers (X, Y)</td>
<td>1-3</td>
</tr>
<tr>
<td>1.6</td>
<td>Stack Pointer Registers (U, S)</td>
<td>1-3</td>
</tr>
<tr>
<td>1.7</td>
<td>Program Counter (PC)</td>
<td>1-4</td>
</tr>
<tr>
<td>1.8</td>
<td>Accumulator Registers (A, B, D)</td>
<td>1-4</td>
</tr>
<tr>
<td>1.9</td>
<td>Direct Page Register (DP)</td>
<td>1-4</td>
</tr>
<tr>
<td>1.10</td>
<td>Condition Code Register (CC)</td>
<td>1-4</td>
</tr>
<tr>
<td>1.10.1</td>
<td>Condition Code Bits</td>
<td>1-5</td>
</tr>
<tr>
<td>1.10.1.1</td>
<td>Half Carry (H), Bit 5</td>
<td>1-5</td>
</tr>
<tr>
<td>1.10.1.2</td>
<td>Negative (N), Bit 3</td>
<td>1-5</td>
</tr>
<tr>
<td>1.10.1.3</td>
<td>Zero (Z), Bit 2</td>
<td>1-5</td>
</tr>
<tr>
<td>1.10.1.4</td>
<td>Overflow (V), Bit 1</td>
<td>1-5</td>
</tr>
<tr>
<td>1.10.1.5</td>
<td>Carry (C), Bit 0</td>
<td>1-5</td>
</tr>
<tr>
<td>1.10.2</td>
<td>Interrupt Mask Bits and Stacking Indicator</td>
<td>1-5</td>
</tr>
<tr>
<td>1.10.2.1</td>
<td>Fast Interrupt Request Mask (F), Bit 6</td>
<td>1-5</td>
</tr>
<tr>
<td>1.10.2.2</td>
<td>Interrupt Request Mask (I), Bit 4</td>
<td>1-5</td>
</tr>
<tr>
<td>1.10.2.3</td>
<td>Entire Flag (E), Bit 7</td>
<td>1-6</td>
</tr>
<tr>
<td>1.11</td>
<td>Pin Assignments and Signal Description</td>
<td>1-6</td>
</tr>
<tr>
<td>1.11.1</td>
<td>MC6809 Clocks</td>
<td>1-6</td>
</tr>
<tr>
<td>1.11.1.1</td>
<td>Oscillator (EXTAL, XTAL)</td>
<td>1-6</td>
</tr>
<tr>
<td>1.11.1.2</td>
<td>Enable (E)</td>
<td>1-7</td>
</tr>
<tr>
<td>1.11.1.3</td>
<td>Quadrature (Q)</td>
<td>1-7</td>
</tr>
<tr>
<td>1.11.2</td>
<td>MC6809E Clocks (E and Q)</td>
<td>1-7</td>
</tr>
<tr>
<td>1.11.3</td>
<td>Three State Control (TSC) (MC6809E)</td>
<td>1-7</td>
</tr>
<tr>
<td>1.11.4</td>
<td>Last Instruction Cycle (LIC) (MC6809E)</td>
<td>1-7</td>
</tr>
<tr>
<td>1.11.5</td>
<td>Address Bus (A0-A15)</td>
<td>1-7</td>
</tr>
<tr>
<td>1.11.6</td>
<td>Data Bus (D0-D7)</td>
<td>1-7</td>
</tr>
<tr>
<td>1.11.7</td>
<td>Read/Write (R/W)</td>
<td>1-8</td>
</tr>
<tr>
<td>1.11.8</td>
<td>Processor State Indicators (BA, BS)</td>
<td>1-8</td>
</tr>
<tr>
<td>1.11.8.1</td>
<td>Normal</td>
<td>1-8</td>
</tr>
<tr>
<td>1.11.8.2</td>
<td>Interrupt or Reset Acknowledge</td>
<td>1-8</td>
</tr>
<tr>
<td>1.11.8.3</td>
<td>Sync Acknowledge</td>
<td>1-8</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS
(CONTINUED)

<table>
<thead>
<tr>
<th>Paragraph No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.11.8.4</td>
<td>Halt/Bus Grant</td>
<td>1-8</td>
</tr>
<tr>
<td>1.11.9</td>
<td>Reset (RESET)</td>
<td>1-9</td>
</tr>
<tr>
<td>1.11.10</td>
<td>Interrupts</td>
<td>1-9</td>
</tr>
<tr>
<td>1.11.10.1</td>
<td>Non-Maskable Interrupt (NMI)</td>
<td>1-9</td>
</tr>
<tr>
<td>1.11.10.2</td>
<td>Fast Interrupt Request (FIRQ)</td>
<td>1-9</td>
</tr>
<tr>
<td>1.11.10.3</td>
<td>Interrupt Request (IRQ)</td>
<td>1-9</td>
</tr>
<tr>
<td>1.11.11</td>
<td>Memory Ready (MRDY) (MC6809)</td>
<td>1-9</td>
</tr>
<tr>
<td>1.11.12</td>
<td>Advanced Valid Memory Address (AVMA) (MC6809E)</td>
<td>1-10</td>
</tr>
<tr>
<td>1.11.13</td>
<td>Halt (HALT)</td>
<td>1-10</td>
</tr>
<tr>
<td>1.11.14</td>
<td>Direct Memory Access/Bus Request (DMA/BREQ) (MC6809)</td>
<td>1-10</td>
</tr>
<tr>
<td>1.11.15</td>
<td>Busy (MC6809E)</td>
<td>1-10</td>
</tr>
<tr>
<td>1.11.16</td>
<td>Power</td>
<td>1-11</td>
</tr>
</tbody>
</table>

SECTION 2
ADDRESSING MODES

<table>
<thead>
<tr>
<th>Paragraph No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Introduction</td>
<td>2-1</td>
</tr>
<tr>
<td>2.2</td>
<td>Addressing Modes</td>
<td>2-1</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Inherent</td>
<td>2-1</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Immediate</td>
<td>2-1</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Extended</td>
<td>2-2</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Direct</td>
<td>2-2</td>
</tr>
<tr>
<td>2.2.5</td>
<td>Indexed</td>
<td>2-2</td>
</tr>
<tr>
<td>2.2.5.1</td>
<td>Constant Offset from Register</td>
<td>2-2</td>
</tr>
<tr>
<td>2.2.5.2</td>
<td>Accumulator Offset from Register</td>
<td>2-3</td>
</tr>
<tr>
<td>2.2.5.3</td>
<td>Autoincrement/Decrement from Register</td>
<td>2-3</td>
</tr>
<tr>
<td>2.2.5.4</td>
<td>Indirection</td>
<td>2-4</td>
</tr>
<tr>
<td>2.2.5.5</td>
<td>Extended Indirect</td>
<td>2-4</td>
</tr>
<tr>
<td>2.2.5.6</td>
<td>Program Counter Relative</td>
<td>2-4</td>
</tr>
<tr>
<td>2.2.6</td>
<td>Branch Relative</td>
<td>2-4</td>
</tr>
</tbody>
</table>

SECTION 3
INTERRUPT CAPABILITIES

<table>
<thead>
<tr>
<th>Paragraph No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>3-1</td>
</tr>
<tr>
<td>3.2</td>
<td>Non-Maskable Interrupt (NMI)</td>
<td>3-1</td>
</tr>
<tr>
<td>3.3</td>
<td>Fast Maskable Interrupt Request (FIRQ)</td>
<td>3-2</td>
</tr>
<tr>
<td>3.4</td>
<td>Normal Maskable Interrupt Request (IRQ)</td>
<td>3-2</td>
</tr>
<tr>
<td>3.5</td>
<td>Software Interrupts (SWI, SWI2, SWI3)</td>
<td>3-2</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

(Concluded)

<table>
<thead>
<tr>
<th>Paragraph No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>4-1</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Position-Independence</td>
<td>4-1</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Modular Programming</td>
<td>4-1</td>
</tr>
<tr>
<td>4.1.2.1</td>
<td>Local Storage</td>
<td>4-1</td>
</tr>
<tr>
<td>4.1.2.2</td>
<td>Global Storage</td>
<td>4-2</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Reentrancy/Recursion</td>
<td>4-2</td>
</tr>
<tr>
<td>4.2</td>
<td>M6809 Capabilities</td>
<td>4-2</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Module Construction</td>
<td>4-2</td>
</tr>
<tr>
<td>4.2.1.1</td>
<td>Parameters</td>
<td>4-3</td>
</tr>
<tr>
<td>4.2.1.2</td>
<td>Local Storage</td>
<td>4-3</td>
</tr>
<tr>
<td>4.2.1.3</td>
<td>Global Storage</td>
<td>4-3</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Position-Independent Code</td>
<td>4-4</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Reentrant Programs</td>
<td>4-5</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Recursive Programs</td>
<td>4-5</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Loops</td>
<td>4-5</td>
</tr>
<tr>
<td>4.2.6</td>
<td>Stack Programming</td>
<td>4-6</td>
</tr>
<tr>
<td>4.2.6.1</td>
<td>M6809 Stacking Operations</td>
<td>4-6</td>
</tr>
<tr>
<td>4.2.6.2</td>
<td>Subroutine Linkage</td>
<td>4-7</td>
</tr>
<tr>
<td>4.2.6.3</td>
<td>Software Stacks</td>
<td>4-8</td>
</tr>
<tr>
<td>4.2.7</td>
<td>Real Time Programming</td>
<td>4-8</td>
</tr>
<tr>
<td>4.3</td>
<td>Program Documentation</td>
<td>4-8</td>
</tr>
<tr>
<td>4.4</td>
<td>Instruction Set</td>
<td>4-9</td>
</tr>
</tbody>
</table>

# APPENDIX A

INSTRUCTION SET DETAILS

| A.1 | Introduction                                      | A-1 |
| A.2 | Notation                                          | A-1 |
|     | Instructions (listed in alphabetical order)      | A-3 |

# APPENDIX B

ASSIST09 MONITOR PROGRAM

<p>| B.1 | General Description                             | B-1 |
| B.2 | Implementation Requirements                     | B-1 |
| B.3 | Interrupt Control                               | B-2 |
| B.4 | Initialization                                  | B-3 |</p>
<table>
<thead>
<tr>
<th>Paragraph No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.5</td>
<td>Input/Output Control</td>
<td>B-4</td>
</tr>
<tr>
<td>B.6</td>
<td>Command Format</td>
<td>B-4</td>
</tr>
<tr>
<td>B.7</td>
<td>Command List</td>
<td>B-5</td>
</tr>
<tr>
<td>B.8</td>
<td>Commands</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Breakpoint</td>
<td>B-6</td>
</tr>
<tr>
<td></td>
<td>Call</td>
<td>B-6</td>
</tr>
<tr>
<td></td>
<td>Display</td>
<td>B-7</td>
</tr>
<tr>
<td></td>
<td>Encode</td>
<td>B-7</td>
</tr>
<tr>
<td></td>
<td>Go</td>
<td>B-8</td>
</tr>
<tr>
<td></td>
<td>Load</td>
<td>B-8</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>B-9</td>
</tr>
<tr>
<td></td>
<td>Null</td>
<td>B-10</td>
</tr>
<tr>
<td></td>
<td>Offset</td>
<td>B-10</td>
</tr>
<tr>
<td></td>
<td>Punch</td>
<td>B-11</td>
</tr>
<tr>
<td></td>
<td>Register</td>
<td>B-11</td>
</tr>
<tr>
<td></td>
<td>Stevel</td>
<td>B-12</td>
</tr>
<tr>
<td></td>
<td>Trace</td>
<td>B-12</td>
</tr>
<tr>
<td></td>
<td>Verify</td>
<td>B-13</td>
</tr>
<tr>
<td></td>
<td>Window</td>
<td>B-13</td>
</tr>
<tr>
<td>B.9</td>
<td>Services</td>
<td>B-14</td>
</tr>
<tr>
<td></td>
<td>BKPT</td>
<td>B-15</td>
</tr>
<tr>
<td></td>
<td>INCHP</td>
<td>B-15</td>
</tr>
<tr>
<td></td>
<td>MONITR</td>
<td>B-16</td>
</tr>
<tr>
<td></td>
<td>OUTCH</td>
<td>B-17</td>
</tr>
<tr>
<td></td>
<td>OUT2HS</td>
<td>B-17</td>
</tr>
<tr>
<td></td>
<td>OUT4HS</td>
<td>B-18</td>
</tr>
<tr>
<td></td>
<td>PAUSE</td>
<td>B-18</td>
</tr>
<tr>
<td></td>
<td>PCRLF</td>
<td>B-19</td>
</tr>
<tr>
<td></td>
<td>PDATA</td>
<td>B-19</td>
</tr>
<tr>
<td></td>
<td>PDATA1</td>
<td>B-20</td>
</tr>
<tr>
<td></td>
<td>SPACE</td>
<td>B-21</td>
</tr>
<tr>
<td></td>
<td>VTRSW</td>
<td>B-21</td>
</tr>
<tr>
<td>B.10</td>
<td>Vector Swap Service</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.ACIA</td>
<td>B-22</td>
</tr>
<tr>
<td></td>
<td>.AVTBL</td>
<td>B-23</td>
</tr>
<tr>
<td></td>
<td>.BSDTA</td>
<td>B-24</td>
</tr>
<tr>
<td></td>
<td>.BSOFF</td>
<td>B-24</td>
</tr>
<tr>
<td></td>
<td>.BSON</td>
<td>B-25</td>
</tr>
<tr>
<td></td>
<td>.CIDTA</td>
<td>B-25</td>
</tr>
<tr>
<td></td>
<td>.CIOFF</td>
<td>B-26</td>
</tr>
<tr>
<td></td>
<td>.CION</td>
<td>B-26</td>
</tr>
<tr>
<td></td>
<td>.CMDL1</td>
<td>B-27</td>
</tr>
<tr>
<td></td>
<td>.CMDL2</td>
<td>B-28</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS (CONTINUED)

<table>
<thead>
<tr>
<th>Paragraph No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>.CODTA</td>
<td></td>
<td>B-28</td>
</tr>
<tr>
<td>.COFF</td>
<td></td>
<td>B-29</td>
</tr>
<tr>
<td>.COON</td>
<td></td>
<td>B-29</td>
</tr>
<tr>
<td>.ECHO</td>
<td></td>
<td>B-30</td>
</tr>
<tr>
<td>.FIRQ</td>
<td></td>
<td>B-30</td>
</tr>
<tr>
<td>.HSDATA</td>
<td></td>
<td>B-31</td>
</tr>
<tr>
<td>.IRQ</td>
<td></td>
<td>B-31</td>
</tr>
<tr>
<td>.NMI</td>
<td></td>
<td>B-32</td>
</tr>
<tr>
<td>.PAD</td>
<td></td>
<td>B-32</td>
</tr>
<tr>
<td>.PAUSE</td>
<td></td>
<td>B-33</td>
</tr>
<tr>
<td>.PTM</td>
<td></td>
<td>B-33</td>
</tr>
<tr>
<td>.RESET</td>
<td></td>
<td>B-34</td>
</tr>
<tr>
<td>.RSVD</td>
<td></td>
<td>B-34</td>
</tr>
<tr>
<td>.SWI</td>
<td></td>
<td>B-35</td>
</tr>
<tr>
<td>.SWI2</td>
<td></td>
<td>B-35</td>
</tr>
<tr>
<td>.SWI3</td>
<td></td>
<td>B-36</td>
</tr>
<tr>
<td>B.11</td>
<td>Monitor Listing</td>
<td>B-37</td>
</tr>
</tbody>
</table>

## APPENDIX C

MACHINE CODE TO INSTRUCTION CROSS REFERENCE

| C.1 | Introduction                          | C-1     |

## APPENDIX D

PROGRAMMING AID

| D.1 | Introduction                          | D-1     |

## APPENDIX E

ASCII CHARACTER SET

| E.1 | Introduction                          | E-1     |
| E.2 | Character Representation and Code Identification | E-1     |
| E.3 | Control Characters                    | E-2     |
| E.4 | Graphic Characters                    | E-2     |
# TABLE OF CONTENTS (CONTINUED)

<table>
<thead>
<tr>
<th>Paragraph No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>APPENDIX F</strong></td>
<td></td>
</tr>
<tr>
<td>F.1</td>
<td>Introduction</td>
<td>F-1</td>
</tr>
<tr>
<td>F.2</td>
<td>Opcode Map</td>
<td>F-1</td>
</tr>
<tr>
<td></td>
<td><strong>APPENDIX G</strong></td>
<td></td>
</tr>
<tr>
<td>G.1</td>
<td>Introduction</td>
<td>G-1</td>
</tr>
<tr>
<td></td>
<td><strong>APPENDIX H</strong></td>
<td></td>
</tr>
<tr>
<td>H.1</td>
<td>Introduction</td>
<td>H-1</td>
</tr>
<tr>
<td>H.2</td>
<td>Powers of 2; Powers of 16</td>
<td>H-1</td>
</tr>
<tr>
<td>H.3</td>
<td>Hexadecimal and Decimal Conversion</td>
<td>H-2</td>
</tr>
<tr>
<td>H.3.1</td>
<td>Converting Hexadecimal to Decimal</td>
<td>H-2</td>
</tr>
<tr>
<td>H.3.2</td>
<td>Converting Decimal to Hexadecimal</td>
<td>H-2</td>
</tr>
</tbody>
</table>

# LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Programming Model</td>
<td>1-3</td>
</tr>
<tr>
<td>1-2</td>
<td>Condition Code Register</td>
<td>1-4</td>
</tr>
<tr>
<td>1-3</td>
<td>Processor Pin Assignments</td>
<td>1-6</td>
</tr>
<tr>
<td>2-1</td>
<td>Postbyte Usage for EXG/TFR, PSH/PUL Instructions</td>
<td>2-2</td>
</tr>
<tr>
<td>3-1</td>
<td>Interrupt Processing Flowchart</td>
<td>3-5</td>
</tr>
<tr>
<td>4-1</td>
<td>Stacking Order</td>
<td>4-7</td>
</tr>
<tr>
<td>B-1</td>
<td>Memory Map</td>
<td>B-2</td>
</tr>
<tr>
<td>E-1</td>
<td>ASCII Character Set</td>
<td>E-1</td>
</tr>
<tr>
<td>G-1</td>
<td>Pin Assignments</td>
<td>G-1</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>BA/BS Signal Encoding</td>
<td>1-8</td>
</tr>
<tr>
<td>2-1</td>
<td>Postbyte Usage for Indexed Addressing Modes</td>
<td>2-3</td>
</tr>
<tr>
<td>3-1</td>
<td>Interrupt Vector Locations</td>
<td>3-1</td>
</tr>
<tr>
<td>4-1</td>
<td>Instruction Set</td>
<td>4-9</td>
</tr>
<tr>
<td>4-2</td>
<td>8-Bit Accumulator and Memory Instructions</td>
<td>4-11</td>
</tr>
<tr>
<td>4-3</td>
<td>16-Bit Accumulator and Memory Instructions</td>
<td>4-12</td>
</tr>
<tr>
<td>4-4</td>
<td>Index/Stack Pointer Instructions</td>
<td>4-12</td>
</tr>
<tr>
<td>4-5</td>
<td>Branch Instructions</td>
<td>4-13</td>
</tr>
<tr>
<td>4-6</td>
<td>Miscellaneous Instructions</td>
<td>4-13</td>
</tr>
<tr>
<td>A-1</td>
<td>Operation Notation</td>
<td>A-1</td>
</tr>
<tr>
<td>A-2</td>
<td>Register Notation</td>
<td>A-2</td>
</tr>
<tr>
<td>B-1</td>
<td>Command List</td>
<td>B-5</td>
</tr>
<tr>
<td>B-2</td>
<td>Services</td>
<td>B-14</td>
</tr>
<tr>
<td>B-3</td>
<td>Vector Table Entries</td>
<td>B-22</td>
</tr>
<tr>
<td>C-1</td>
<td>Machine Code to Instruction Cross Reference</td>
<td>C-2</td>
</tr>
<tr>
<td>D-1</td>
<td>Programming Aid</td>
<td>D-1</td>
</tr>
<tr>
<td>E-1</td>
<td>Control Characters</td>
<td>E-2</td>
</tr>
<tr>
<td>E-2</td>
<td>Graphic Characters</td>
<td>E-3</td>
</tr>
<tr>
<td>F-1</td>
<td>Opcode Map</td>
<td>F-2</td>
</tr>
<tr>
<td>F-2</td>
<td>Indexed Addressing Mode Data</td>
<td>F-3</td>
</tr>
<tr>
<td>H-1</td>
<td>Powers of 2; Powers of 16</td>
<td>H-1</td>
</tr>
<tr>
<td>H-2</td>
<td>Hexadecimal and Decimal Conversion Chart</td>
<td>H-2</td>
</tr>
</tbody>
</table>
SECTION 1
GENERAL DESCRIPTION

1.1 INTRODUCTION

This section contains a general description of the Motorola MC6809 and MC6809E Microprocessor Units (MPU). Pin assignments and a brief description of each input/output signal are also given. The term MPU, processor, or M6809 will be used throughout this manual to refer to both the MC6809 and MC6809E processors. When a topic relates to only one of the processors, that specific designator (MC6809 or MC6809E) will be used.

1.2 FEATURES

The MC6809 and MC6809E microprocessors are greatly enhanced, upward compatible, computationally faster extensions of the MC6800 microprocessor.

Enhancements such as additional registers (a Y index register, a U stack pointer, and a direct page register) and instructions (such as MUL) simplify software design. Improved addressing modes have also been implemented.

Upward compatibility is guaranteed as MC6800 assembly language programs may be assembled using the Motorola MC6809 Macro Assembler. This code, while not as compact as native M6809 code, is, in most cases, 100% functional.

Both address and data are available from the processor earlier in an instruction cycle than from the MC6800 which simplifies hardware design. Two clock signals, E (the MC6800 φ2) and a new quadrature clock Q (which leads E by one-quarter cycle) also simplify hardware design.

A memory ready (MRDY) input is provided on the MC6809 for working with slow memories. This input stretches both the processor internal cycle and direct memory access bus cycle times but allows internal operations to continue at full speed. A direct memory access request (DMA/BREQ) input is provided for immediate memory access or dynamic memory refresh operations; this input halts the internal MC6809 clocks. Because the processor's registers are dynamic, an internal counter periodically recovers the bus from direct memory access operations and performs a true processor refresh cycle to allow unlimited length direct memory access operation. An interrupt acknowledge signal is available to allow development of vectoring by interrupt device hardware or detection of operating system calls.
Three prioritized, vectored, hardware interrupt levels are available: non-maskable, fast, and normal. The highest and lowest priority interrupts, non-maskable and interrupt request respectively, are the normal interrupts used in the M6800 family. A new interrupt on this processor is the fast interrupt request which provides faster service to its interrupt input by only stacking the program counter and condition code register and then servicing the interrupt.

Modern programming techniques such as position-independent, system independent, and reentrant programming are readily supported by these processors.

A Memory Management Unit (MMU), the MC6829, allows a M6809 based system to address a two megabyte memory space. Note: An arbitrary number of tasks may be supported — slower — with software.

This advanced family of processors is compatible with all M6800 peripheral parts.

1.3 SOFTWARE FEATURES

Some of the software features of these processors are itemized in the following paragraphs. Programs developed for the MC6800 can be easily converted for use with the MC6809 or MC6809E by running the source code through a M6809 Macro Assembler or any one of the many cross assemblers that are available.

The addressing modes of any microprocessor provide it with the capability to efficiently address memory to obtain data and instructions. The MC6809 and MC6809E have a versatile set of addressing modes which allow them to function using modern programming techniques.

The addressing modes and instructions of the MC6809 and MC6809E are upward compatible with the MC6800. The old addressing modes have been retained and many new ones have been added.

A direct page register has been added which allows a 256 byte “direct” page anywhere in the 64K logical address space. The direct page register is used to hold the most-significant byte of the address used in direct addressing and decrease the time required for address calculation.

Branch relative addressing to anywhere in the memory map (−32768 to +32767) is available.

Program counter relative addressing is also available for data access as well as branch instructions.

The indexed addressing modes have been expanded to include:

- 0-, 5-, 8-, 16-bit constant offsets,
- 8- or 16-bit accumulator offsets,
- autoincrement/decrement (stack operation).
In addition, most indexed addressing modes may have an additional level of indirection added.

Any or all registers may be pushed on to or pulled from either stack with a single instruction.

A multiply instruction is included which multiplies unsigned binary numbers in accumulators A and B and places the unsigned result in the 16-bit accumulator D. This unsigned multiply instruction also allows signed or unsigned multiple precision multiplication.

1.4 PROGRAMMING MODEL

The programming model (Figure 1-1) for these processors contains five 16-bit and four 8-bit registers that are available to the programmer.

![Programming Model Diagram]

Figure 1-1. Programming Model

1.5 INDEX REGISTERS (X, Y)

The index registers are used during the indexed addressing modes. The address information in an index register is used in the calculation of an effective address. This address may be used to point directly to data or may be modified by an optional constant or register offset to produce the effective address.

1.6 STACK POINTER REGISTERS (U, S)

Two stack pointer registers are available in these processors. They are: a user stack pointer register (U) controlled exclusively by the programmer, and a hardware stack pointer register (S) which is used automatically by the processor during subroutine calls.
and interrupts, but may also be used by the programmer. Both stack pointers always point to the top of the stack.

These registers have the same indexed addressing mode capabilities as the index registers, and also support push and pull instructions. All four indexable registers (X, Y, U, S) are referred to as pointer registers.

1.7 PROGRAM COUNTER (PC)

The program counter register is used by these processors to store the address of the next instruction to be executed. It may also be used as an index register in certain addressing modes.

1.8 ACCUMULATOR REGISTERS (A, B, D)

The accumulator registers (A, B) are general-purpose 8-bit registers used for arithmetic calculations and data manipulation.

Certain instructions concatenate these registers into one 16-bit accumulator with register A positioned as the most-significant byte. When concatenated, this register is referred to as accumulator D.

1.9 DIRECT PAGE REGISTER (DP)

This 8-bit register contains the most-significant byte of the address to be used in the direct addressing mode. The contents of this register are concatenated with the byte following the direct addressing mode operation code to form the 16-bit effective address. The direct page register contents appear as bits A15 through A8 of the address. This register is automatically cleared by a hardware reset to ensure M6800 compatibility.

1.10 CONDITION CODE REGISTER (CC)

The condition code register contains the condition codes and the interrupt masks as shown in Figure 1-2.

![Figure 1-2. Condition Code Register](image-url)
1.10.1 CONDITION CODE BITS. Five bits in the condition code register are used to indicate the results of instructions that manipulate data. They are: half carry (H), negative (N), zero (Z), overflow (V), and carry (C). The effect each instruction has on these bits is given in the detail information for each instruction (see Appendix A).

1.10.1.1 Half Carry (H), Bit 5. This bit is used to indicate that a carry was generated from bit three in the arithmetic logic unit as a result of an 8-bit addition. This bit is undefined in all subtract-like instructions. The decimal addition adjust (DAA) instruction uses the state of this bit to perform the adjust operation.

1.10.1.2 Negative (N), Bit 3. This bit contains the value of the most-significant bit of the result of the previous data operation.

1.10.1.3 Zero (Z), Bit 2. This bit is used to indicate that the result of the previous operation was zero.

1.10.1.4 Overflow (V), Bit 1. This bit is used to indicate that the previous operation caused a signed arithmetic overflow.

1.10.1.5 Carry (C), Bit 0. This bit is used to indicate that a carry or a borrow was generated from bit seven in the arithmetic logic unit as a result of an 8-bit mathematical operation.

1.10.2 INTERRUPT MASK BITS AND STACKING INDICATOR. Two bits (I and F) are used as mask bits for the interrupt request and the fast interrupt request inputs. When either or both of these bits are set, their associated input will not be recognized.

One bit (E) is used to indicate how many registers (all, or only the program counter and condition code) were stacked during the last interrupt.

1.10.2.1 Fast Interrupt Request Mask (F), Bit 6. This bit is used to mask (disable) any fast interrupt request line (FIRQ). This bit is set automatically by a hardware reset or after recognition of another interrupt. Execution of certain instructions such as SWI will also inhibit recognition of a FIRQ input.

1.10.2.2 Interrupt Request Mask (I), Bit 4. This bit is used to mask (disable) any interrupt request input (IRQ). This bit is set automatically by a hardware reset or after recognition of another interrupt. Execution of certain instructions such as SWI will also inhibit recognition of an IRQ input.
1.10.2.3 Entire Flag (E), Bit 7. This bit is used to indicate how many registers were stacked. When set, all the registers were stacked during the last interrupt stacking operation. When clear, only the program counter and condition code registers were stacked during the last interrupt.

The state of the E bit in the stacked condition code register is used by the return from interrupt (RTI) instruction to determine the number of registers to be unstacked.

1.11 PIN ASSIGNMENTS AND SIGNAL DESCRIPTION

Figure 1-3 shows the pin assignments for the processors. The following paragraphs provide a short description of each of the input and output signals.

![Processor Pin Assignments Diagram]

1.11.1 MC6809 CLOCKS. The MC6809 has four pins committed to developing the clock signals needed for internal and system operation. They are: the oscillator pins EXTAL and XTAL; the standard M6800 enable (E) clock; and a new, quadrature (Q) clock.

1.11.1.1 Oscillator (EXTAL, XTAL). These pins are used to connect the processor's internal oscillator to an external, parallel-resonant crystal. These pins can also be used for input of an external TTL timing signal by grounding the XTAL pin and applying the input to the EXTAL pin. The crystal or the external timing source is four times the resulting bus frequency.
1.11.1.2 Enable (E). The E clock is similar to the phase 2 (φ2) MC6800 bus timing clock. The leading edge indicates to memory and peripherals that the data is stable and to begin write operations. Data movement occurs after the Q clock is high and is latched on the trailing edge of E. Data is valid from the processor (during a write operation) by the rising edge of E.

1.11.1.3 Quadrature (Q). The Q clock leads the E clock by approximately one half of the E clock time. Address information from the processor is valid with the leading edge of the Q clock. The Q clock is a new signal in these processors and does not have an equivalent clock within the MC6800 bus timing.

1.11.2 MC6809E CLOCKS (E and Q). The MC6809E has two pins provided for the TTL clock signal inputs required for internal operation. They are the standard M6800 enable (E) clock and the quadrature (Q) clock. The Q Input must lead the E input.

Addresses will be valid from the processor (on address delay time after the falling edge of E) and data will be latched from the bus by the falling edge of E. The Q input is fully TTL compatible. The E input is used to drive the internal MOS circuitry directly and therefore requires input levels above the normal TTL levels.

1.11.3 THREE STATE CONTROLS (TSC) (MC6809E). This input is used to place the address and data lines and the R/W line in the high-impedance state and allows the address bus to be shared with other bus masters.

1.11.4 LAST INSTRUCTION CYCLE (LIC) (MC6809E). This output goes high during the last cycle of every instruction and its high-to-low transition indicates that the first byte of an opcode will be latched at the end of the present bus cycle.

1.11.5 ADDRESS BUS (A0-A15). This 16-bit, unidirectional, three-state bus is used by the processor to provide address information to the address bus. Address information is valid on the rising edge of the Q clock. All 16 outputs are in the high-impedance state when the bus available (BA) signal is high, and for one bus cycle thereafter.

When the processor does not require the address bus for a data transfer, it outputs address FFFF16, and read/write (R/W) high. This is a "dummy access" of the least-significant byte of the reset vector which replaces the valid memory address (VMA) functions of the MC6800. For the MC6809, the memory read signal internal circuitry inhibits stretching of the clocks during non-access cycles.

1.11.6 DATA BUS (D0-D7). This 8-bit, bidirectional, three-state bus is the general purpose data path. All eight outputs are in the high-impedance state when the bus available (BA) output is high.
1.11.7 **READ/WRITE (R/W)**. This output indicates the direction of data transfer on the data bus. A low indicates that the processor is writing onto the data bus; a high indicates that the processor is reading data from the data bus. The signal at the R/W output is valid at the leading edge of the Q clock. The R/W output is in the high-impedance state when the bus available (BA) output is high.

1.11.8 **PROCESSOR STATE INDICATORS (BA, BS)**. The processor uses these two output lines to indicate the present processor state. These pins are valid with the leading edge of the Q clock.

The bus available (BA) output is used to indicate that the buses (address and data) and the read/write output are in the high-impedance state. This signal can be used to indicate to bus-sharing or direct memory access systems that the buses are available. When BA goes low, an additional dead cycle will elapse before the processor regains control of the buses.

The bus status (BS) output is used in conjunction with the BA output to indicate the present state of the processor. Table 1-1 is a listing of the BA and BS outputs and the processor states that they indicate. The following paragraphs briefly explain each processor state.

**Table 1-1. BA/BS Signal Encoding**

<table>
<thead>
<tr>
<th>BA</th>
<th>BS</th>
<th>Processor State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Normal (Running)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Interrupt or Reset Acknowledge</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Sync Acknowledge</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Halt/Bus Grant Acknowledged</td>
</tr>
</tbody>
</table>

1.11.8.1 **Normal.** The processor is running and executing instructions.

1.11.8.2 **Interrupt or Reset Acknowledge.** This processor state is indicated during both cycles of a hardware vector fetch which occurs when any of the following interrupts have occurred: RESET, NMI, FIRQ, IRQ, SWI, SWI2, and SWI3.

This output, plus decoding of address lines A3 through A1 provides the user with an indication of which interrupt is being serviced.

1.11.8.3 **Sync Acknowledge.** The processor is waiting for an external synchronization input on an interrupt line. See SYNC instruction in Appendix A.

1.11.8.4 **Halt/Bus Grant.** The processor is halted or bus control has been granted to some other device.
1.11.9 **RESET (RESET).** This input is used to reset the processor. A low input lasting longer than one bus cycle will reset the processor.

The reset vector is fetched from locations $FFFE$ and $FFFF$ when the processor enters the reset acknowledge state as indicated by the BA output being low and the BS output being high.

During initial power-on, the reset input should be held low until the clock oscillator is fully operational.

1.11.10 **INTERRUPTS.** The processor has three separate interrupt input pins: non-maskable interrupt (NMI), fast interrupt request (FIRQ), and interrupt request (IRQ). These interrupt inputs are latched by the falling edge of every Q clock except during cycle stealing operations where only the NMI input is latched. Using this point as a reference, a delay of at least one bus cycle will occur before the interrupt is recognized by the processor.

1.11.10.1 **Non-Maskable Interrupt (NMI).** A negative edge on this input requests that a non-maskable interrupt sequence be generated. This input, as the name indicates, cannot be masked by software and has the highest priority of the three interrupt inputs. After a reset has occurred, a NMI input will not be recognized by the processor until the first program load of the hardware stack pointer. The entire machine state is saved on the hardware stack during the processing of a non-maskable interrupt. This interrupt is internally blocked after a hardware reset until the stack pointer is initialized.

1.11.10.2 **Fast Interrupt Request (FIRQ).** This input is used to initiate a fast interrupt request sequence. Initiation depends on the F (fast interrupt request mask) bit in the condition code register being clear. This bit is set during reset. During the interrupt, only the contents of the condition code register and the program counter are stacked resulting in a short amount of time required to service this interrupt. This interrupt has a higher priority than the normal interrupt request (IRQ).

1.11.10.3 **Interrupt Request (IRQ).** This input is used to initiate what might be considered the “normal” interrupt request sequence. Initiation depends on the I (interrupt mask) bit in the condition code register being clear. This bit is set during reset. The entire machine state is saved on the hardware stack during processing of an IRQ input. This input has the lowest priority of the three hardware interrupts.

1.11.11 **MEMORY READ (MRDY) (MC6809).** This input allows extension of the E and Q clocks to allow a longer data access time. A low on this input allows extension of the E and Q clocks (E high and Q low) in integral multiples of quarter bus cycles (up to 10 cycles) to allow interface with slow memory devices.
Memory ready does not extend the E and Q clocks during non-valid memory access cycles and therefore the processor does not slow down for "don't care" bus accesses. Memory ready may also be used to extend the E and Q clocks when an external device is using the halt and direct memory access/bus request inputs.

1.11.12 ADVANCED VALID MEMORY ADDRESS (AVMA) (MC6809E). This output signal indicates that the MC6809E will use the bus in the following bus cycle. This output is low when the MC6809E is in either a halt or sync state.

1.11.13 HALT. This input is used to halt the processor. A low input halts the processor at the end of the present instruction execution cycle and the processor remains halted indefinitely without loss of data.

When the processor is halted, the BA output is high to indicate that the buses are in the high-impedance state and the BS output is also high to indicate that the processor is in the halt/bus grant state.

During the halt/bus grant state, the processor will not respond to external real-time requests such as FIRQ or IRQ. However, a direct memory access/bus request input will be accepted. A non-maskable interrupt or a reset input will be latched for processing later. The E and Q clocks continue to run during the halt/bus grant state.

1.11.14 DIRECT MEMORY ACCESS/BUS REQUEST (DMA/BREQ) (MC6809). This input is used to suspend program execution and make the buses available for another use such as a direct memory access or a dynamic memory refresh.

A low level on this input occurring during the Q clock high time suspends instruction execution at the end of the current cycle. The processor acknowledges acceptance of this input by setting the BA and BS outputs high to signify the bus grant state. The requesting device now has up to 15 bus cycles before the processor retrieves the bus for self-refresh.

Typically, a direct memory access controller will request to use the bus by setting the DMA/BREQ input low when E goes high. When the processor acknowledges this input by setting the BA and BS outputs high, that cycle will be a dead cycle used to transfer bus mastership to the direct memory access controller. False memory access during any dead cycle should be prevented by externally developing a system DMAVMA signal which is low in any cycle when the BA output changes.

When the BA output goes low, either as a result of a direct memory access/bus request or a processor self-refresh, the direct memory access device should be removed from the bus. Another dead cycle will elapse before the processor accesses memory, to allow transfer of bus mastership without contention.

1.11.15 BUSY (MC6809E). This output indicates that bus re-arbitration should be deferred and provides the indivisible memory operation required for a "test-and-set" primitive.
This output will be high for the first two cycles of any Read-Modify-Write instruction, high during the first byte of a double-byte access, and high during the first byte of any indirect access or vector-fetch operation.

1.11.16 POWER. Two inputs are used to supply power to the processor: VCC is +5.0 ±5%, while VSS is ground or 0 volts.
SECTION 2
ADDRESSING MODES

2.1 INTRODUCTION

This section contains a description of each of the addressing modes available on these processors.

2.2 ADDRESSING MODES

The addressing modes available on the MC6809 and MC6809E are: Inherent, Immediate, Extended, Direct, Indexed (with various offsets and autoincrementing/decrementing), and Branch Relative. Some of these addressing modes require an additional byte after the opcode to provide additional addressing interpretation. This byte is called a postbyte.

The following paragraphs provide a description of each addressing mode. In these descriptions the term effective address is used to indicate the address in memory from which the argument for an instruction is fetched or stored, or from which instruction processing is to proceed.

2.2.1 INHERENT. The information necessary to execute the instruction is contained in the opcode. Some operations specifying only the index registers or the accumulators, and no other arguments, are also included in this addressing mode.

Example: \texttt{MUL}

2.2.2 IMMEDIATE. The operand is contained in one or two bytes immediately following the opcode. This addressing mode is used to provide constant data values that do not change during program execution. Both 8-bit and 16-bit operands are used depending on the size of the argument specified in the opcode.

Example:
\begin{verbatim}
LDA #CR
LDB #7
LDA #$F0
LDB %1110000
LDX #$8004
\end{verbatim}

Another form of immediate addressing uses a postbyte to determine the registers to be manipulated. The exchange (EXG) and transfer (TFR) instructions use the postbyte as shown in Figure 2-1(A). The push and pull instructions use the postbyte to designate the registers to be pushed or pulled as shown in Figure 2-1(B).
2.2.3 EXTENDED. The effective address of the argument is contained in the two bytes following the opcode. Instructions using the extended addressing mode can reference arguments anywhere in the 64K addressing space. Extended addressing is generally not used in position independent programs because it supplies an absolute address.

Example: \texttt{LDA @CAT}

2.2.4 DIRECT. The effective address is developed by concatenation of the contents of the direct page register with the byte immediately following the opcode. The direct page register contents are the most-significant byte of the address. This allows accessing 256 locations within any one of 256 pages. Therefore, the entire addressing range is available for access using a single two-byte instruction.

Example: \texttt{LDA >CAT}

2.2.5 INDEXED. In these addressing modes, one of the pointer registers (X, Y, U, or S), and sometimes the program counter (PC) is used in the calculation of the effective address of the instruction operand. The basic types (and their variations) of indexed addressing available are shown in Table 2-1 along with the postbyte configuration used.

2.2.5.1 Constant Offset from Register. The contents of the register designated in the postbyte are added to a twos complement offset value to form the effective address of
the instruction operand. The contents of the designated register are not affected by this addition. The offset sizes available are:

No offset — designated register contains the effective address

5-bit — 16 to +15
8-bit — 128 to +127
16-bit — 32768 to +32767

Table 2-1. Postbyte Usage for Indexed Addressing Modes

<table>
<thead>
<tr>
<th>Mode Type</th>
<th>Variation</th>
<th>Direct</th>
<th>Indirect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant Offset from Register</td>
<td>No Offset</td>
<td>1RR00100</td>
<td>1RR10100</td>
</tr>
<tr>
<td>(twos Complement Offset)</td>
<td>5-Bit Offset</td>
<td>0RRnmmm</td>
<td>Defaults to 8-bit</td>
</tr>
<tr>
<td></td>
<td>8-Bit Offset</td>
<td>1RR01000</td>
<td>1RR11000</td>
</tr>
<tr>
<td></td>
<td>16-Bit Offset</td>
<td>1RR01001</td>
<td>1RR11001</td>
</tr>
<tr>
<td>Accumulator Offset from Register</td>
<td>A Accumulator Offset</td>
<td>1RR00110</td>
<td>1RR10110</td>
</tr>
<tr>
<td>(twos Complement Offset)</td>
<td>B Accumulator Offset</td>
<td>1RR00101</td>
<td>1RR10101</td>
</tr>
<tr>
<td></td>
<td>D Accumulator Offset</td>
<td>1RR01011</td>
<td>1RR11011</td>
</tr>
<tr>
<td>Auto Increment/Decrement from Register</td>
<td>Increment by 1</td>
<td>1RR00000</td>
<td>Not Allowed</td>
</tr>
<tr>
<td></td>
<td>Increment by 2</td>
<td>1RR00001</td>
<td>1RR10001</td>
</tr>
<tr>
<td></td>
<td>Decrement by 1</td>
<td>JRR00010</td>
<td>Not Allowed</td>
</tr>
<tr>
<td></td>
<td>Decrement by 2</td>
<td>1RR00011</td>
<td>1RR10011</td>
</tr>
<tr>
<td>Constant Offset from Program Counter</td>
<td>8-Bit Offset</td>
<td>1XX01100</td>
<td>1XX11000</td>
</tr>
<tr>
<td></td>
<td>16-Bit Offset</td>
<td>1XX01101</td>
<td>1XX11011</td>
</tr>
<tr>
<td>Extended Indirect</td>
<td>16-Bit Address</td>
<td>--------</td>
<td>10011111</td>
</tr>
</tbody>
</table>

The 5-bit offset value is contained in the postbyte. The 8- and 16-bit offset values are contained in the byte or bytes immediately following the postbyte. If the Motorola assembler is used, it will automatically determine the most efficient offset; thus, the programmer need not be concerned about the offset size.

Examples: LDA ,X LDY 64000,U LDB 0,Y LDA 17,PC LDX 84,000,S LDA There,PCR

2.2.5.2 Accumulator Offset from Register. The contents of the index or pointer register designed in the postbyte are temporarily added to the twos complement offset value contained in an accumulator (A, B, or D) also designated in the postbyte. Neither the designated register nor the accumulator contents are affected by this addition.

Example: LDA A,X LDA D,U LDA B,Y

2.2.5.3 Auto Increment/Decrement from Register. This addressing mode works in a postincrementing or predecrementing manner. The amount of increment or decrement, one or two positions, is designated in the postbyte.
In the autoincrement mode, the contents of the effective address contained in the pointer register, designated in the postbyte, and then the pointer register is automatically incremented; thus, the pointer register is postincremented.

In the autodecrement mode, the pointer register, designated in the postbyte, is automatically decremented first and then the contents of the new address are used; thus, the pointer register is predecremented.

Examples:

<table>
<thead>
<tr>
<th>Autoincrement</th>
<th>Autodecrement</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA ,X+</td>
<td>LDA , -X</td>
</tr>
<tr>
<td>LDA ,Y+</td>
<td>LDA , -Y</td>
</tr>
<tr>
<td>LDA ,S+</td>
<td>LDA , -S</td>
</tr>
<tr>
<td>LDA ,U+</td>
<td>LDA , -U</td>
</tr>
<tr>
<td>LDX ,X+</td>
<td>LDX , -X</td>
</tr>
<tr>
<td>LDX ,Y+</td>
<td>LDX , -Y</td>
</tr>
<tr>
<td>LDX ,U+</td>
<td>LDX , -U</td>
</tr>
<tr>
<td>LDX ,S+</td>
<td>LDX , -S</td>
</tr>
</tbody>
</table>

2.2.5.4 Indirection. When using indirection, the effective address of the base indexed addressing mode is used to fetch two bytes which contain the final effective address of the operand. It can be used with all the indexed addressing modes and the program counter relative addressing mode.

2.2.5.5 Extended Indirect. The effective address of the argument is located at the address specified by the two bytes following the postbyte. The postbyte is used to indicate indirection.

Example: LDA [$F000]

2.2.5.6 Program Counter Relative. The program counter can also be used as a pointer with either an 8- or 16-bit signed constant offset. The offset value is added to the program counter to develop an effective address. Part of the postbyte is used to indicate whether the offset is 8 or 16 bits.

2.2.6 BRANCH RELATIVE. This addressing mode is used when branches from the current instruction location to some other location relative to the current program counter are desired. If the test condition of the branch instruction is true, then the effective address is calculated (program counter plus twos complement offset) and the branch is taken. If the test condition is false, the processor proceeds to the next in-line instruction. Note that the program counter is always pointing to the next instruction when the offset is added. Branch relative addressing is always used in position independent programs for all control transfers.

For short branches, the byte following the branch instruction opcode is treated as an 8-bit signed offset to be used to calculate the effective address of the next instruction if the branch is taken. This is called a short relative branch and the range is limited to plus 127 or minus 128 bytes from the following opcode.

For long branches, the two bytes after the opcode are used to calculate the effective address. This is called a long relative branch and the range is plus 32,767 or minus 32,768
SECTION 3
INTERRUPT CAPABILITIES

3.1 INTRODUCTION

The MC6809 and MC6809E microprocessors have six vectored interrupts (three hardware and three software). The hardware interrupts are the non-maskable interrupt (NMI), the fast maskable interrupt request (FIRQ), and the normal maskable interrupt request (IRQ). The software interrupts consist of SWI, SWI2, and SWI3. When an interrupt request is acknowledged, all the processor registers are pushed onto the hardware stack, except in the case of FIRQ where only the program counter and the condition code register is saved, and control is transferred to the address in the interrupt vector. The priority of these interrupts is, highest to lowest, NMI, SWI, FIRQ, IRQ, SWI2, and SWI3. Figure 3-1 is a detailed flowchart of interrupt processing in these processors. The interrupt vector locations are given in Table 3-1. The vector locations contain the address for the interrupt routine.

Additional information on the SWI, SWI2, and SWI3 interrupts is given in Appendix A. The hardware interrupts, NMI, FIRQ, and IRQ are listed alphabetically at the end of Appendix A.

<table>
<thead>
<tr>
<th>Interrupt Description</th>
<th>Vector Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset (RESET)</td>
<td>FFFE FFFF</td>
</tr>
<tr>
<td>Non-Maskable Interrupt (NMI)</td>
<td>FFFC FFFD</td>
</tr>
<tr>
<td>Software Interrupt (SWI)</td>
<td>FFFA FFFB</td>
</tr>
<tr>
<td>Interrupt Request (IRQ)</td>
<td>FFF8 FFF9</td>
</tr>
<tr>
<td>Fast Interrupt Request (FIRQ)</td>
<td>FFF6 FFF7</td>
</tr>
<tr>
<td>Software Interrupt 2 (SWI2)</td>
<td>FFF4 FFF5</td>
</tr>
<tr>
<td>Software Interrupt 3 (SWI3)</td>
<td>FFF2 FFF3</td>
</tr>
<tr>
<td>Reserved</td>
<td>FFF0 FFF1</td>
</tr>
</tbody>
</table>

3.2 NON-MASKABLE INTERRUPT (NMI)

The non-maskable interrupt is edge-sensitive in the sense that if it is sampled low one cycle after it has been sampled high, a non-maskable interrupt will be triggered. Because the non-maskable interrupt cannot be masked by execution of the non-maskable interrupt handler routine, it is possible to accept another non-maskable interrupt before executing the first instruction of the interrupt routine. A fatal error will exist if a non-maskable interrupt is repeatedly allowed to occur before completing the return from interrupt (RTI) instruction of the previous non-maskable interrupt request, since the stack
will eventually overflow. This interrupt is especially applicable to gaining immediate processor response for powerfail, software dynamic memory refresh, or other non-delayable events.

3.3 FAST MASKABLE INTERRUPT REQUEST (FIRQ)

A low level on the FIRQ input with the F (fast interrupt request mask) bit in the condition code register clear triggers this interrupt sequence. The fast interrupt request provides fast interrupt response by stacking only the program counter and condition code register. This allows fast context switching with minimal overhead. If any registers are used by the interrupt routine then they can be saved by a single push instruction.

After accepting a fast interrupt request, the processor clears the E flag, saves the program counter and condition code register, and then sets both the I and F bits to mask any further IRQ and FIRQ interrupts. After servicing the original interrupt, the user may selectively clear the I and F bits to allow multiple-level interrupts if so desired.

3.4 NORMAL MASKABLE INTERRUPT REQUEST (IRQ)

A low level on the IRQ input with the I (interrupt request mask) bit in the condition code register clear triggers this interrupt sequence. The normal maskable interrupt request provides a slower hardware response to interrupts because it causes the entire machine state to be stacked. However, this means that interrupting software routines can use all processor resources without fear of damaging the interrupted routine. A normal interrupt request, having lower priority than the fast interrupt request, is prevented from interrupting the fast interrupt handler by the automatic setting of the I bit by the fast interrupt request handler.

After accepting a normal interrupt request, the processor sets the E flag, saves the entire machine state, and then sets the I bit to mask any further interrupt request inputs. After servicing the original interrupt, the user may clear the I bit to allow multiple-level normal interrupts.

All interrupt handling routines should return to the formerly executing tasks using a return from interrupt (RTI) instruction. This instruction recovers the saved machine state from the hardware stack and control is returned to the interrupted program. If the recovered E bit is clear, it indicates that a fast interrupt request occurred and only the program counter address and condition code register are to be recovered.

3.5 SOFTWARE INTERRUPTS (SWI, SWI2, SWI3)

The software interrupts cause the processor to go through the normal interrupt request sequence of stacking the complete machine state even though the interrupting source is the processor itself. These interrupts are commonly used for program debugging and for calls to an operating system.
Normal processing of the SWI input sets the I and F bits to prevent either of these interrupt requests from affecting the completion of a software interrupt request. The remaining software interrupt request inputs (SWI2 and SWI3) do not have the priority of the SWI input and therefore do not mask the two hardware interrupt request inputs (FIRQ and IRQ).
SECTION 4
PROGRAMMING

4.1 INTRODUCTION

These processors are designed to be source-code compatible with the M6800 to make
use of the substantial existing base of M6800 software and training. However, this asset
should not overshadow the capabilities built into these processors that allow more
modern programming techniques such as position-independence, modular programming,
and reentrancy/recursion to be used on a microprocessor-based system. A brief
review of these methods is given in the following paragraphs.

4.1.1 POSITION INDEPENDENCE. A program is said to be “position-independent” if it
will run correctly when the same machine code is positioned arbitrarily in memory. Such
a program is useful in many different hardware configurations, and might be copied from
a disk into RAM when the operating system first sees a request to use a system utility.
Position-independent programs never use absolute (extended or direct) addressing: in-
stead, inherent immediate, register, indexed and relative modes are used. In particular,
there should be no jump (absolute) or jump to subroutine instructions nor should ab-
solute addresses be used. A position-independent program is almost always preferable
to a position-dependent program (although position-independent code is usually 5 to
10% slower than normal code).

4.1.2 MODULAR PROGRAMMING. Modular programming is another indication of quality
code. A module is a program element which can be easily disconnected from the rest of
the program either for re-use in a new environment or for replacement. A module is usually
a subroutine (although a subroutine is not necessarily a module); frequently, the pro-
gramer isolates register changes internal to the module by pushing these registers
onto the stack upon entry, and pulling them off the stack before the return. Isolating
register changes in the called module, to that module alone, allows the code in the call-
ing program to be more easily analyzed since it can be assumed that all registers (except
those specifically used for parameter transfer are unchanged by each called module.
This leaves the processor’s registers free at each level for loop counts, address com-
parisons, etc.

4.1.2.1 Local Storage. A clean method for allocating “local” storage is required both by
position-independent programs as well as modular programs. Local or temporary storage
is used to hold values only during execution of a module (or called modules) and is releas-
ed upon return. One way to allocate local storage is to decrement the hardware stack

4-1
pointer(s) by the number of bytes needed. Interrupts will then leave this area intact and it can be de-allocated on exiting the module. A module will almost always need more temporary storage than just the MPU registers.

4.1.2.2 Global Storage. Even in a modular environment there may be a need for “global” values which are accessible by many modules within a given system. These provide a convenient means for storing values from one invocation to another invocation of the same routine. Global storage may be created as local storage at some level, and a pointer register (usually U) used to point at this area. This register is passed unchanged in all subroutines, and may be used to index into the global area.

4.1.3 REENTRY/RECURSION. Many programs will eventually involve execution in an interrupt-driven environment. If the Interrupt handlers are complex, they might well call the same routine which has just been interrupted. Therefore, to protect present programs against certain obsolescence, all programs should be written to be reentrant. A reentrant routine allocates different local variable storage upon each entry. Thus, a later entry does not destroy the processing associated with an earlier entry.

The same technique which was implemented to allow reentrancy also allows recursion. A recursive routine is defined as a routine that calls itself. A recursive routine might be written to simplify the solution of certain types of problems, especially those which have a data structure whose elements may themselves be a structure. For example, a parenthetical equation represents a case where the expression in parenthesis may be considered to be a value which is operated on by the rest of the equation. A programmer might choose to write an expression evaluator passing the parenthetical expression (which might also contain parenthetical expressions) in the call, and receive back the returned value of the expression within the parenthesis.

4.2 M6809 CAPABILITIES

The following paragraphs briefly explain how the MC6809 is used with the programming techniques mentioned earlier.

4.2.1 MODULE CONSTRUCTION. A module can be defined as a logically self-contained and discrete part of a larger program. A properly constructed module accepts well defined inputs, carries out a set of processing actions, and produces a specified output. The use of parameters, local storage, and global storage by a program module is given in the following paragraphs. Since registers will be used inside the module (essentially a form of local storage), the first thing that is usually done at entry to a module is to push (save) them on to the stack. This can be done with one instruction (e.g., PSHS Y, X, B, A). After the body of the module is executed, the saved registers are collected, and a subroutine return is performed, at one time, by pulling the program counter from the stack (e.g., PULS A,B,X,Y,PC).
4.2.1.1 Parameters. Parameters may be passed to or from modules either in registers, if they will provide sufficient storage for parameter passage, or on the stack. If parameters are passed on the stack, they are placed there before calling the lower level module. The called module is then written to use local storage inside the stack as needed (e.g., ADDA offset, S). Notice that the required offset consists of the number of bytes pushed (upon entry), plus two from the stacked return address, plus the data offset at the time of the call. This value may be calculated, by hand, by drawing a “stack picture” diagram representing module entry, and assigning convenient mnemonics to these offsets with the assembler. Returned parameters replace those sent to the routine. If more parameters are to be returned on the stack than would normally be sent, space for their return is allocated by the calling routine before the actual call (if four additional bytes are to be returned, the caller would execute LEAS – 4, S to acquire the additional storage).

4.2.1.2 Local Storage. Local storage space is acquired from the stack while the present routine is executing and then returned to the stack prior to exit. The act of pushing registers which will be used in later calculations essentially saves those registers in temporary local storage. Additional local storage can easily be acquired from the stack e.g., executing LEAS – 2048, S acquires a buffer area running from the 0, S to 2047, S inclusive. Any byte in this area may be accessed directly by any instruction which has an indexed addressing mode. At the end of the routine, the area acquired for local storage is released (e.g., LEAS 2048, S) prior to the final pull. For cleaner programs, local storage should be allocated at entry to the module and released at the exit of the module.

4.2.1.3 Global Storage. The area required for global storage is also most effectively acquired from the stack, probably by the highest level routine in the standard package. Although this is local storage to the highest level routine, it becomes “global” by positioning a register to point at this storage, (sometimes referred to as a stack mark) then establishing the convention that all modules pass that same pointer value when calling lower level modules. In practice, it is convenient to leave this stack mark register unchanged in all modules, especially if global accesses are common. The highest level routine in the standard package would execute the following sequence upon entry (to initialize the global area):

\[
\begin{align*}
& \text{PSHS} \quad U \quad \text{higher level mark, if any} \\
& \text{TFR} \quad S,U \quad \text{new stack mark} \\
& \text{LEAS} \quad -17, U \quad \text{allocate global storage}
\end{align*}
\]

Note that the U register now defines 17-bytes of locally allocated (permanent) globals (which are –1, U through –17, U) as well as other external globals (2, U and above) which have been passed on the stack by the routine which called the standard package. Any global may be accessed by any module using exactly the same offset value at any level (e.g., ROL, RAT, U; where RAT EQU – 11 has been defined). Furthermore, the values stacked prior to invoking the standard package may include pointers to data or I/O peripherals. Any indexed operation may be performed indexed indirect through those pointers, which means, for example, that the module need know nothing about the actual hardware configuration, except that (upon entry) the pointer to an I/O register has been placed at a given location on the stack.
4.2.2 POSITION-INDEPENDENT CODE. Position-independent code means that the same machine language code can be placed anywhere in memory and still function correctly. The M6809 has a long relative (16-bit offset) branch mode along with the common MC6800 branches, plus program-counter relative addressing. Program-counter relative addressing uses the program counter like an indexable register, which allows all instructions that reference memory to also reference data relative to the program counter. The M6809 also has load effective address (LEA) instructions which allow the user to point to data in a ROM in a position-independent manner.

An important rule for generating position-independent code is: NEVER USE ABSOLUTE ADDRESSING.

Program-counter relative addressing on the M6809 is a form of indexed addressing that uses the program counter as the base register for a constant-offset indexing operation. However, the M6809 assembler treats the PCR address field differently from that used in other indexed instructions. In PCR addressing, the assembly time location value is subtracted from the (constant) value of the PCR offset. The resulting distance to the desired symbol is the value placed into the machine language object code. During execution, the processor adds the value of the run-time PC to the distance to get a position-independent absolute address.

The PCR indexed addressing form can be used to point at any location relative to the program regardless of position in memory. The PCR form of indexed addressing allows access to tables within the program space in a position-independent manner via use of the load effective address instruction.

In a program which is completely position-independent, some absolute locations are usually required, particularly for I/O. If the locations of I/O devices are placed on the stack (as globals) by a small setup routine before the standard package is invoked, all internal modules can do their I/O through that pointer (e.g., STA [ACIAD, U]), allowing the hardware to be easily changed, if desired. Only the single, small, and obvious setup routine need be rewritten for each different hardware configuration.

Global, permanent, and temporary values need to be easily available in a position-independent manner. Use the stack for this data since the stacked data is directly accessible. Stack the absolute address of I/O devices before calling any standard software package since the package can use the stacked addresses for I/O in any system.

The LEA instructions allow access to tables, data, or immediate values in the text of the program in a position-independent manner as shown in the following example:

```
LEAX      MSG1,PCR
LBSR      PDATA

MSG1      FCC      /PRINT THIS!
```
Here we wish to point at a message to be printed from the body of the program. By writing "MSG1, PCR" we signal the assembler to compute the distance between the present address (the address of the LBSR) and MSG1. This result is inserted as a constant into the LEA instruction which will be indexed from the program counter value at the time of execution. Now, no matter where the code is located, when it is executed the computer offset from the program counter will point at MSG1. This code is position-independent.

It is common to use space in the hardware stack for temporary storage. Space is made for temporary variables from 0,S through TEMP-1, S by decrementing the stack pointer equal to the length of required storage. We could use:

LEAS – TEMP,S.

Not only does this facilitate position-independent code but it is structured and helps reentrancy and recursion.

4.2.3 REENTRANT PROGRAMS. A program that can be executed by several different users sharing the same copy of it in memory is called reentrant. This is important for interrupt driven systems. This method saves considerable memory space, especially with large interrupt routines. Stacks are required for reentrant programs, and the M6809 can support up to four stacks by using the X and Y index registers as stack pointers.

Stacks are simple and convenient mechanisms for generating reentrant programs. Subroutines which use stacks for passing parameters and results can be easily made to be reentrant. Stack accesses use the indexed addressing mode for fast, efficient execution. Stack addressing is quick.

Pure code, or code that is not self-modifying, is mandatory to produce reentrant code. No internal information within the code is subject to modification. Reentrant code never has internal temporary storage, is simpler to debug, can be placed in ROM, and must be interruptable.

4.2.4 RECURSIVE PROGRAMS. A recursive program is one that can call itself. They are quite convenient for parsing mechanisms and certain arithmetic functions such as computing factorials. As with reentrant programming, stacks are very useful for this technique.

4.2.5 LOOPS. The usual structured loops (i.e., REPEAT...UNTIL, WHILE...DO, FOR..., etc.) are available in assembly language in exactly the same way a high-level language compiler could translate the construct for execution on the target machine. Using a FOR...NEXT loop as an example, it is possible to push the loop count, increment value, and termination value on the stack as variables local to that loop. On each pass through the loop, the working register is saved, the loop count picked up, the increment added in, and the result compared to the termination value. Based on this comparison, the loop counter might be updated, the working register recovered and the loop resumed, or the working register recovered and the loop variables de-allocated. Reasonable macros
could make the source form for loop trivial, even in assembly language. Such macros might reduce errors resulting from the use of multiple instructions simply to implement a standard control structure.

4.2.6 STACK PROGRAMMING. Many microprocessor applications require data stored as contiguous pieces of information in memory. The data may be temporary, that is, subject to change or it may be permanent. Temporary data will most likely be stored in RAM. Permanent data will most likely be stored in ROM.

It is important to allow the main program as well as subroutines access to this block of data, especially if arguments are to be passed from the main program to the subroutines and vice versa.

4.2.6.1 M6800 Stack Operations. Stack pointers are markers which point to the stack and its internal contents. Although all four index registers may be used as stack registers, the S (hardware stack pointer) and the U (user stack pointer) are generally preferred because the push and pull instructions apply to these registers. Both are 16-bit indexable registers. The processor uses the S register automatically during interrupts and subroutine calls. The U register is free for any purpose needed. It is not affected by interrupts or subroutine calls implemented by the hardware.

Either stack pointer can be specified as the base address in indexed addressing. One use of the indirect addressing mode uses stack pointers to allow addresses of data to be passed to a subroutine on a stack as arguments to a subroutine. The subroutine can now reference the data with one instruction. High-level language calls that pass arguments by reference are now more efficiently coded. Also, each stack push or pull operation in a program uses a postbyte which specifies any register or set of registers to be pushed or pulled from either stack. With this option, the overhead associated with subroutine calls in both assembly and high-level language programs is greatly decreased. In fact, with the large number of instructions that use autoincrement and autodecrement, the M6800 can emulate a true stack computer architecture.

Using the S or U stack pointer, the order in which the registers are pushed or pulled is shown in Figure 4-1. Notice that we push “onto” the stack towards decreasing memory locations. The program counter is pushed first. Then the stack pointer is decremented and the “other” stack pointer is pushed onto the stack. Decrementing and storing continues until all the registers requested by the postbyte are pushed onto the stack. The stack pointer points to the top of the stack after the push operation.

The stacking order is specified by the processor. The stacking order is identical to the order used for all hardware and software interrupts. The same order is used even if a subset of the registers is pushed.

Without stacks, most modern block-structured high-level languages would be cumbersome to implement. Subroutine linkage is very important in high-level language generation. Paragraph 4.2.6.2 describes how to use a stack mark pointer for this important task.
Good programming practice dictates the use of the hardware stack for temporary storage. To reserve space, decrement the stack pointer by the amount of storage required with the instruction LEAS – TEMPS, S. This instruction makes space for temporary variables from 0,S through TEMPS – 1,S.

![Stacking Order Diagram](image)

**Figure 4-1. Stacking Order**

4.2.6.2 Subroutine Linkage. In the highest level routine, global variables are sometimes considered to be local. Therefore, global storage is allocated at this point, but access to these same variables requires different offset values depending on subroutine depth. Because subroutine depth changes dynamically, the length may not be known beforehand. This problem is solved by assigning one pointer (U will be used in the following description, but X or Y could also be used) to “mark” a location on the hardware stack by using the instruction TFR S,U. If the programmer does this immediately prior to allocating global storage, then all variables will then be available at a constant negative offset location from this stack mark. If the stack is marked after the global variables are
allocated, then the global variables are available at a constant positive offset from U. Register U is then called the stack mark pointer. Recall that the hardware stack pointer may be modified by hardware interrupts. For this reason, it is fatal to use data referred to by a negative offset with respect to the hardware stack pointer, S.

4.2.6.3 Software Stacks. If more than two stacks are needed, autoincrement and autodecrement mode of addressing can be used to generate additional software stack pointers.

The X, Y, and U index registers are quite useful in loops for incrementing and decrementing purposes. The pointer is used for searching tables and also to move data from one area of memory to another (block moves). This autoincrement and autodecrement feature is available in the indexed addressing mode of the M6809 to facilitate such operations.

In autoincrement, the value contained in the index register (X or Y, U or S) is used as the effective address and then the register is incremented (postincremented). In autodecrement, the index register is first decremented and then used to obtain the effective address (predecremented). Postincrement or predecrement is always performed in this addressing mode. This is equivalent in operation to the push and pull from a stack. This equivalence allows the X and Y index registers to be used as software stack pointers. The indexed addressing mode can also implement an extra level of post indirection. This feature supports parameter and pointer operations.

4.2.7 REAL TIME PROGRAMMING. Real time programming requires special care. Sometimes a peripheral or task demands an immediate response from the processor, other times it can wait. Most real time applications are demanding in terms of processor response.

A common solution is to use the interrupt capability of the processor in solving real time problems. Interrupts mean just that; they request a break in the current sequence of events to solve an asynchronous service request. The system designer must consider all variations of the conditions to be encountered by the system including software interaction with interrupts. As a result, problems due to software design are more common in interrupt implementation code for real time programming than most other situations. Software timeouts, hardware interrupts, and program control interrupts are typically used in solving real time programming problems.

4.3 PROGRAM DOCUMENTATION

Common sense dictates that a well documented program is mandatory. Comments are needed to explain each group of instructions since their use is not always obvious from looking at the code. Program boundaries and branch instructions need full clarification. Consider the following points when writing comments: up-to-date, accuracy, completeness, conciseness, and understandability.
Accurate documentation enables you and others to maintain and adapt programs for updating and/or additional use with other programs.

The following program documentation standards are suggested.

A) Each subroutine should have an associated header block containing at least the following elements:
   1) A full specification for this subroutine — including associated data structures — such that replacement code could be generated from this description alone.
   2) All usage of memory resources must be defined, including:
      a) All RAM needed from temporary (local) storage used during execution of this subroutine or called subroutines.
      b) All RAM needed for permanent storage (used to transfer values from one execution of the subroutine to future executions).
      c) All RAM accessed as global storage (used to transfer values from or to higher-level subroutines).
      d) All possible exit-state conditions, if these are to be used by calling routines to test occurrences internal to the subroutine.

B) Code internal to each subroutine should have sufficient associated line comments to help in understanding the code.

C) All code must be non-self-modifying and position-independent.

D) Each subroutine which includes a loop must be separately documented by a flowchart or pseudo high-level language algorithm.

E) Any module or subroutine should be executable starting at the first location and exit at the last location.

4.4 INSTRUCTION SET

The complete instruction set for the M6809 is given in Table 4-1.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABX</td>
<td>Add Accumulator B into Index Register X</td>
</tr>
<tr>
<td>ADC</td>
<td>Add with Carry into Register</td>
</tr>
<tr>
<td>ADD</td>
<td>Add Memory into Register</td>
</tr>
<tr>
<td>AND</td>
<td>Logical AND Memory into Register</td>
</tr>
<tr>
<td>ASL</td>
<td>Arithmetic Shift Left</td>
</tr>
<tr>
<td>ASR</td>
<td>Arithmetic Shift Right</td>
</tr>
<tr>
<td>BCC</td>
<td>Branch on Carry Clear</td>
</tr>
<tr>
<td>BCS</td>
<td>Branch on Carry Set</td>
</tr>
<tr>
<td>BEO</td>
<td>Branch on Equal</td>
</tr>
<tr>
<td>BGE</td>
<td>Branch on Greater Than or Equal to Zero</td>
</tr>
<tr>
<td>BGT</td>
<td>Branch on Greater</td>
</tr>
<tr>
<td>BHI</td>
<td>Branch if Higher</td>
</tr>
<tr>
<td>BHS</td>
<td>Branch if Higher or Same</td>
</tr>
<tr>
<td>BIT</td>
<td>Bit Test</td>
</tr>
<tr>
<td>BLE</td>
<td>Branch if Less than or Equal to Zero</td>
</tr>
</tbody>
</table>
Table 4-1. Instruction Set (Continued)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLO</td>
<td>Branch on Lower</td>
</tr>
<tr>
<td>BLS</td>
<td>Branch on Lower or Same</td>
</tr>
<tr>
<td>BLT</td>
<td>Branch on Less than Zero</td>
</tr>
<tr>
<td>BMI</td>
<td>Branch on Minus</td>
</tr>
<tr>
<td>BNE</td>
<td>Branch Not Equal</td>
</tr>
<tr>
<td>BPL</td>
<td>Branch on Plus</td>
</tr>
<tr>
<td>BRA</td>
<td>Branch Always</td>
</tr>
<tr>
<td>BRN</td>
<td>Branch Never</td>
</tr>
<tr>
<td>BSR</td>
<td>Branch to Subroutine</td>
</tr>
<tr>
<td>BVC</td>
<td>Branch on Overflow Clear</td>
</tr>
<tr>
<td>BVS</td>
<td>Branch on Overflow Set</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear</td>
</tr>
<tr>
<td>CMP</td>
<td>Compare Memory from a Register</td>
</tr>
<tr>
<td>COM</td>
<td>Complement</td>
</tr>
<tr>
<td>CWAI</td>
<td>Clear CC bits and Wait for Interrupt</td>
</tr>
<tr>
<td>DAA</td>
<td>Decimal Addition Adjust</td>
</tr>
<tr>
<td>DEC</td>
<td>Decrement</td>
</tr>
<tr>
<td>EOR</td>
<td>Exclusive OR</td>
</tr>
<tr>
<td>EXG</td>
<td>Exchange Registers</td>
</tr>
<tr>
<td>INC</td>
<td>Increment</td>
</tr>
<tr>
<td>JMP</td>
<td>Jump</td>
</tr>
<tr>
<td>JSR</td>
<td>Jump to Subroutine</td>
</tr>
<tr>
<td>LD</td>
<td>Load Register from Memory</td>
</tr>
<tr>
<td>LEA</td>
<td>Load Effective Address</td>
</tr>
<tr>
<td>LSL</td>
<td>Logical Shift Left</td>
</tr>
<tr>
<td>LSR</td>
<td>Logical Shift Right</td>
</tr>
<tr>
<td>MUL</td>
<td>Multiply</td>
</tr>
<tr>
<td>NEG</td>
<td>Negate</td>
</tr>
<tr>
<td>NOP</td>
<td>No Operation</td>
</tr>
<tr>
<td>OR</td>
<td>Inclusive OR Memory into Register</td>
</tr>
<tr>
<td>PSH</td>
<td>Push Registers</td>
</tr>
<tr>
<td>PUL</td>
<td>Pull Registers</td>
</tr>
<tr>
<td>ROL</td>
<td>Rotate Left</td>
</tr>
<tr>
<td>ROR</td>
<td>Rotate Right</td>
</tr>
<tr>
<td>RTI</td>
<td>Return from Interrupt</td>
</tr>
<tr>
<td>RTS</td>
<td>Return from Subroutine</td>
</tr>
<tr>
<td>SBC</td>
<td>Subtract with Borrow</td>
</tr>
<tr>
<td>SEX</td>
<td>Sign Extend</td>
</tr>
<tr>
<td>ST</td>
<td>Store Register into Memory</td>
</tr>
<tr>
<td>SUB</td>
<td>Subtract Memory from Register</td>
</tr>
<tr>
<td>SWI</td>
<td>Software Interrupt</td>
</tr>
<tr>
<td>SYNC</td>
<td>Synchronize to External Event</td>
</tr>
<tr>
<td>TFR</td>
<td>Transfer Register to Register</td>
</tr>
<tr>
<td>TST</td>
<td>Test</td>
</tr>
</tbody>
</table>

4-10
The instruction set can be functionally divided into five categories. They are:

- **8-Bit Accumulator and Memory Instructions**
- **16-Bit Accumulator and Memory Instructions**
- **Index Register/Stack Pointer Instructions**
- **Branch Instructions**
- **Miscellaneous Instructions**

Tables 4-2 through 4-6 are listings of the M6809 Instructions and their variations grouped into the five categories listed.

### Table 4-2. 8-Bit Accumulator and Memory Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADCA, ADCCB</td>
<td>Add memory to accumulator with carry</td>
</tr>
<tr>
<td>ADDA, ADDB</td>
<td>Add memory to accumulator</td>
</tr>
<tr>
<td>ANDA, ANDB</td>
<td>And memory with accumulator</td>
</tr>
<tr>
<td>ASL, ASLA, ASLB</td>
<td>Arithmetic shift of accumulator or memory left</td>
</tr>
<tr>
<td>ASR, ASRA, ASRB</td>
<td>Arithmetic shift of accumulator or memory right</td>
</tr>
<tr>
<td>BITA, BITB</td>
<td>Bit test memory with accumulator</td>
</tr>
<tr>
<td>CLR, CLRA, CLRB</td>
<td>Clear accumulator or memory location</td>
</tr>
<tr>
<td>CMPA, CMPB</td>
<td>Compare memory from accumulator</td>
</tr>
<tr>
<td>COM, COMA, COMB</td>
<td>Complement accumulator or memory location</td>
</tr>
<tr>
<td>DAA</td>
<td>Decimal adjust A accumulator</td>
</tr>
<tr>
<td>DEC, DECA, DECB</td>
<td>Decrement accumulator or memory location</td>
</tr>
<tr>
<td>EORA, EORB</td>
<td>Exclusive or memory with accumulator</td>
</tr>
<tr>
<td>EXG RT, R2</td>
<td>Exchange RT with R2 (R1, R2 = A, B, CC, DP)</td>
</tr>
<tr>
<td>INC, INCA, INCB</td>
<td>Increment accumulator or memory location</td>
</tr>
<tr>
<td>LDA, LDB</td>
<td>Load accumulator from memory</td>
</tr>
<tr>
<td>LSL, LSLA, LSLB</td>
<td>Logical shift left accumulator or memory location</td>
</tr>
<tr>
<td>LSR, LSRA, LSRB</td>
<td>Logical shift right accumulator or memory location</td>
</tr>
<tr>
<td>MUL</td>
<td>Unsigned multiply (A × B → D)</td>
</tr>
<tr>
<td>NEG, NEGA, NEGB</td>
<td>Negate accumulator or memory</td>
</tr>
<tr>
<td>ORA, ORB</td>
<td>Or memory with accumulator</td>
</tr>
<tr>
<td>ROL, ROLA, ROLB</td>
<td>Rotate accumulator or memory left</td>
</tr>
<tr>
<td>ROR, RORA, RORB</td>
<td>Rotate accumulator or memory right</td>
</tr>
<tr>
<td>SBCA, SBCB</td>
<td>Subtract memory from accumulator with borrow</td>
</tr>
<tr>
<td>STA, STB</td>
<td>Store accumulator to memory</td>
</tr>
<tr>
<td>SUBA, SUBB</td>
<td>Subtract memory from accumulator</td>
</tr>
<tr>
<td>TST, TSTA, TSTB</td>
<td>Test accumulator or memory location</td>
</tr>
<tr>
<td>TFR R1, R2</td>
<td>Transfer R1 to R2 (R1, R2 = A, B, CC, DP)</td>
</tr>
</tbody>
</table>

**NOTE:** A, B, CC, or DP may be pushed to (pulled from) either stack with PSHS, PSHU (PULS, PULLU) instructions.
Table 4-3. 16-Bit Accumulator and Memory Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDD</td>
<td>Add memory to D accumulator</td>
</tr>
<tr>
<td>CMPD</td>
<td>Compare memory from D accumulator</td>
</tr>
<tr>
<td>EXG D, R</td>
<td>Exchange D with X, Y, S, U, or PC</td>
</tr>
<tr>
<td>LDD</td>
<td>Load D accumulator from memory</td>
</tr>
<tr>
<td>SEX</td>
<td>Sign Extend B accumulator into A accumulator</td>
</tr>
<tr>
<td>STD</td>
<td>Store D accumulator to memory</td>
</tr>
<tr>
<td>SUBD</td>
<td>Subtract memory from D accumulator</td>
</tr>
<tr>
<td>TFR D, R</td>
<td>Transfer D to X, Y, S, U, or PC</td>
</tr>
<tr>
<td>TFR R, D</td>
<td>Transfer X, Y, S, U, or PC to D</td>
</tr>
</tbody>
</table>

NOTE: D may be pushed (pulled) to either stack with PSHS, PSHU (PULS, PULU) instructions.

Table 4-4. Index/Stack Pointer Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPS, CMPU</td>
<td>Compare memory from stack pointer</td>
</tr>
<tr>
<td>CMPX, CMPY</td>
<td>Compare memory from index register</td>
</tr>
<tr>
<td>EXG R1, R2</td>
<td>Exchange D, X, Y, S, U or PC with D, X, Y, S, U or PC</td>
</tr>
<tr>
<td>LEAS, LEAU</td>
<td>Load effective address into stack pointer</td>
</tr>
<tr>
<td>LEAX, LEAY</td>
<td>Load effective address into index register</td>
</tr>
<tr>
<td>LDS, LDU</td>
<td>Load stack pointer from memory</td>
</tr>
<tr>
<td>LDX, LDY</td>
<td>Load index register from memory</td>
</tr>
<tr>
<td>PSHS</td>
<td>Push A, B, CC, DP, D, X, Y, U, or PC onto hardware stack</td>
</tr>
<tr>
<td>PSHU</td>
<td>Push A, B, CC, DP, D, X, Y, X, or PC onto user stack</td>
</tr>
<tr>
<td>PULS</td>
<td>Pull A, B, CC, DP, D, X, Y, U, or PC from hardware stack</td>
</tr>
<tr>
<td>PULU</td>
<td>Pull A, B, CC, DP, D, X, Y, S, or PC from hardware stack</td>
</tr>
<tr>
<td>STS, STU</td>
<td>Store stack pointer to memory</td>
</tr>
<tr>
<td>STX, STY</td>
<td>Store index register to memory</td>
</tr>
<tr>
<td>TFR R1, R2</td>
<td>Transfer D, X, Y, S, U, or PC to D, X, Y, S, U, or PC</td>
</tr>
<tr>
<td>ABX</td>
<td>Add B accumulator to X (unsigned)</td>
</tr>
</tbody>
</table>
### Table 4-5. Branch Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEQ, LBEQ</td>
<td>Branch if equal</td>
</tr>
<tr>
<td>BNE, LBNE</td>
<td>Branch if not equal</td>
</tr>
<tr>
<td>BMI, LBMI</td>
<td>Branch if minus</td>
</tr>
<tr>
<td>BPL, LBPL</td>
<td>Branch if plus</td>
</tr>
<tr>
<td>BCS, LBCS</td>
<td>Branch if carry set</td>
</tr>
<tr>
<td>BCC, LBCC</td>
<td>Branch if carry clear</td>
</tr>
<tr>
<td>BVS, LBVS</td>
<td>Branch if overflow set</td>
</tr>
<tr>
<td>BVC, LBVC</td>
<td>Branch if overflow clear</td>
</tr>
<tr>
<td><strong>SIMPLE BRANCHES</strong></td>
<td></td>
</tr>
<tr>
<td>BGT, LBGT</td>
<td>Branch if greater (signed)</td>
</tr>
<tr>
<td>BVS, LBVS</td>
<td>Branch if invalid twos complement result</td>
</tr>
<tr>
<td>BGE, LBGE</td>
<td>Branch if greater than or equal (signed)</td>
</tr>
<tr>
<td>BEQ, LBEQ</td>
<td>Branch if equal</td>
</tr>
<tr>
<td>BNE, LBNE</td>
<td>Branch if not equal</td>
</tr>
<tr>
<td>BLE, LBLE</td>
<td>Branch if less than or equal (signed)</td>
</tr>
<tr>
<td>BVC, LBVC</td>
<td>Branch if valid twos complement result</td>
</tr>
<tr>
<td>BLT, LBLT</td>
<td>Branch if less than (signed)</td>
</tr>
<tr>
<td><strong>SIGNED BRANCHES</strong></td>
<td></td>
</tr>
<tr>
<td>BHI, LBHI</td>
<td>Branch if higher (unsigned)</td>
</tr>
<tr>
<td>BCC, LBCC</td>
<td>Branch if higher or same (unsigned)</td>
</tr>
<tr>
<td>BHS, LBHS</td>
<td>Branch if higher or same (unsigned)</td>
</tr>
<tr>
<td>BEQ, LBEQ</td>
<td>Branch if equal</td>
</tr>
<tr>
<td>BNE, LBNE</td>
<td>Branch if not equal</td>
</tr>
<tr>
<td>BLS, LBLS</td>
<td>Branch if lower or same (unsigned)</td>
</tr>
<tr>
<td>BCS, LBCS</td>
<td>Branch if lower (unsigned)</td>
</tr>
<tr>
<td>BLO, LBLO</td>
<td>Branch if lower (unsigned)</td>
</tr>
<tr>
<td><strong>UNSIGNED BRANCHES</strong></td>
<td></td>
</tr>
<tr>
<td>BSR, LB5R</td>
<td>Branch to subroutine</td>
</tr>
<tr>
<td>BRA, LBRA</td>
<td>Branch always</td>
</tr>
<tr>
<td>BRN, LBRN</td>
<td>Branch never</td>
</tr>
<tr>
<td><strong>OTHER BRANCHES</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Table 4-6. Miscellaneous Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANDCC</td>
<td>AND condition code register</td>
</tr>
<tr>
<td>CWAIE</td>
<td>AND condition code register, then wait for interrupt</td>
</tr>
<tr>
<td>NOP</td>
<td>No operation</td>
</tr>
<tr>
<td>ORCC</td>
<td>OR condition code register</td>
</tr>
<tr>
<td>JMP</td>
<td>Jump</td>
</tr>
<tr>
<td>JSR</td>
<td>Jump to subroutine</td>
</tr>
<tr>
<td>RTI</td>
<td>Return from interrupt</td>
</tr>
<tr>
<td>RTS</td>
<td>Return from subroutine</td>
</tr>
<tr>
<td>SWI, SWI2, SWI3</td>
<td>Software interrupt (absolute indirect)</td>
</tr>
<tr>
<td>SYNC</td>
<td>Synchronize with interrupt line</td>
</tr>
</tbody>
</table>

4-13/4-14
APPENDIX A
INSTRUCTION SET DETAILS

A.1 INTRODUCTION

This appendix contains detailed information about each instruction in the MC6809 instruction set. They are arranged in an alphabetical order with the mnemonic heading set in larger type for easy reference.

A.2 NOTATION

In the operation description for each instruction, symbols are used to indicate the operation. Table A-1 lists these symbols and their meanings. Abbreviations for the various registers, bits, and bytes are also used. Table A-2 lists these abbreviations and their meanings.

Table A-1. Operation Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>transferred to</td>
</tr>
<tr>
<td>∧</td>
<td>Boolean AND</td>
</tr>
<tr>
<td>∨</td>
<td>Boolean OR</td>
</tr>
<tr>
<td>⊕</td>
<td>Boolean exclusive OR</td>
</tr>
<tr>
<td>¬</td>
<td>Boolean NOT</td>
</tr>
<tr>
<td>:</td>
<td>Concatenation</td>
</tr>
<tr>
<td>+</td>
<td>Arithmetic plus</td>
</tr>
<tr>
<td>-</td>
<td>Arithmetic minus</td>
</tr>
<tr>
<td>X</td>
<td>Arithmetic multiply</td>
</tr>
</tbody>
</table>
### Table A-2. Register Notation

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCA or A</td>
<td>Accumulator A</td>
</tr>
<tr>
<td>ACCB or B</td>
<td>Accumulator B</td>
</tr>
<tr>
<td>ACCA:ACCB or D</td>
<td>Double accumulator D</td>
</tr>
<tr>
<td>ACCX</td>
<td>Either accumulator A or B</td>
</tr>
<tr>
<td>CCR or CC</td>
<td>Condition code register</td>
</tr>
<tr>
<td>DPR or DP</td>
<td>Direct page register</td>
</tr>
<tr>
<td>EA</td>
<td>Effective address</td>
</tr>
<tr>
<td>IFF</td>
<td>If and only if</td>
</tr>
<tr>
<td>IX or X</td>
<td>Index register X</td>
</tr>
<tr>
<td>IY or Y</td>
<td>Index register Y</td>
</tr>
<tr>
<td>LSN</td>
<td>Least significant nibble</td>
</tr>
<tr>
<td>M</td>
<td>Memory location</td>
</tr>
<tr>
<td>MI</td>
<td>Memory immediate</td>
</tr>
<tr>
<td>MSN</td>
<td>Most significant nibble</td>
</tr>
<tr>
<td>PC</td>
<td>Program counter</td>
</tr>
<tr>
<td>R</td>
<td>A register before the operation</td>
</tr>
<tr>
<td>R'</td>
<td>A register after the operation</td>
</tr>
<tr>
<td>TEMP</td>
<td>Temporary storage location</td>
</tr>
<tr>
<td>xxH</td>
<td>Most significant byte of any 16-bit register</td>
</tr>
<tr>
<td>xxL</td>
<td>Least significant byte of any 16-bit register</td>
</tr>
<tr>
<td>Sp or S</td>
<td>Hardware Stack pointer</td>
</tr>
<tr>
<td>Us or U</td>
<td>User Stack pointer</td>
</tr>
<tr>
<td>P</td>
<td>A memory argument with Immediate, Direct, Extended, and Indexed addressing modes</td>
</tr>
<tr>
<td>Q</td>
<td>A read-modify-write argument with Direct, Indexed, and Extended addressing modes</td>
</tr>
<tr>
<td>()</td>
<td>The data pointed to by the enclosed (16-bit address)</td>
</tr>
<tr>
<td>dd</td>
<td>8-bit branch offset</td>
</tr>
<tr>
<td>DDDD</td>
<td>16-bit branch offset</td>
</tr>
<tr>
<td>/</td>
<td>Immediate value follows</td>
</tr>
<tr>
<td>$</td>
<td>Hexadecimal value follows</td>
</tr>
<tr>
<td>!</td>
<td>Indirection</td>
</tr>
<tr>
<td>.</td>
<td>Indicates indexed addressing</td>
</tr>
</tbody>
</table>
Source Form: ABX

Operation: IX' ← IX + ACCB

Condition Codes: Not affected.

Description: Add the 8-bit unsigned value in accumulator B into index register X.

Addressing Mode: Inherent
ADC

Add with Carry into Register

Source Forms: ADCA P; ADCB P

Operation: R' ← R + M + C

Condition Codes:
- H — Set if a half-carry is generated; cleared otherwise.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Set if an overflow is generated; cleared otherwise.
- C — Set if a carry is generated; cleared otherwise.

Description: Adds the contents of the C (carry) bit and the memory byte into an 8-bit accumulator.

Addressing Modes: Immediate
Extended
Direct
Indexed
ADD (8-Bit)  Add Memory into Register  ADD (8-Bit)

Source Forms:  ADDA P; ADDB P

Operation:  \( R' \leftarrow R + M \)

Condition Codes:  
H — Set if a half-carry is generated; cleared otherwise.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a carry is generated; cleared otherwise.

Description:  Adds the memory byte into an 8-bit accumulator.

Addressing Modes:  Immediate
Extended
Direct
Indexed
ADD (16-Bit)  Add Memory Into Register  ADD (16-Bit)

Source Forms: ADD D P

Operation: \( R' \leftarrow R + M:M + 1 \)

Condition Codes:
- \( H \) — Not affected.
- \( N \) — Set if the result is negative; cleared otherwise.
- \( Z \) — Set if the result is zero; cleared otherwise.
- \( V \) — Set if an overflow is generated; cleared otherwise.
- \( C \) — Set if a carry is generated; cleared otherwise.

Description: Adds the 16-bit memory value into the 16-bit accumulator.

Addressing Modes:
- Immediate
- Extended
- Direct
- Indexed
**AND**

Logical AND Memory into Register

**Source Forms:**
ANDA P; ANDB P

**Operation:**
\[ R' \leftarrow R \land M \]

**Condition Codes:**
- H — Not affected.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Always cleared.
- C — Not affected.

**Description:**
Performs the logical AND operation between the contents of an accumulator and the contents of memory location M and the result is stored in the accumulator.

**Addressing Modes:**
Immediate
Extended
Direct
Indexed
Source Form: ANDCC #xx

Operation: R' = R \land M1

Condition Codes: Affected according to the operation.

Description: Performs a logical AND between the condition code register and the immediate byte specified in the instruction and places the result in the condition code register.

Addressing Mode: Immediate
ASL

Arithmetic Shift Left

Source Forms: ASL Q; ASLA; ASLB

Operation: \[ C \leftarrow \begin{array}{cccccc} & & & & & \\ b7 & & & & & b0 \end{array} \leftarrow 0 \]

Condition Codes:
- H — Undefined
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Loaded with the result of the exclusive OR of bits six and seven of the original operand.
- C — Loaded with bit seven of the original operand.

Description: Shifts all bits of the operand one place to the left. Bit zero is loaded with a zero. Bit seven is shifted into the C (carry) bit.

Addressing Modes: Inherent
- Extended
- Direct
- Indexed
Source Forms: ASR Q; ASRA; ASRB

Operation:  

Condition Codes: H — Undefined.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Not affected.
C — Loaded with bit zero of the original operand.

Description: Shifts all bits of the operand one place to the right. Bit seven is held constant. Bit zero is shifted into the C (carry) bit.

Addressing Modes: Inherent
Extended
Direct
Indexed
BCC

Branch on Carry Clear

Source Forms: BCC dd; LBCC DDDD

Operation: TEMP — MI
IFF C = 0 then PC' ← PC + TEMP

Condition Codes: Not affected.

Description: Tests the state of the C (carry) bit and causes a branch if it is clear.

Addressing Mode: Relative

Comments: Equivalent to BHS dd; LBHS DDDD
Source Forms:  
BCS dd; LBCS DDDD

Operation:  
TEMP ← MI  
IFF C = 1 then PC' ← PC + TEMP

Condition Codes:  Not affected.

Description:  Tests the state of the C (carry) bit and causes a branch if it is set.

Addressing Mode:  Relative

Comments:  Equivalent to BLO dd; LBLO DDDD
BEQ

Branch on Equal

Source Forms: BEQ dd; LBEQ DDDD

Operation: TEMP ← MI
IFF Z = 1 then PC' ← PC + TEMP

Condition Codes: Not affected.

Description: Tests the state of the Z (zero) bit and causes a branch if it is set. When used after a subtract or compare operation, this instruction will branch if the compared values, signed or unsigned, were exactly the same.

Addressing Mode: Relative
BGE

Branch on Greater than or Equal to Zero

Source Forms:  BGE dd; LBGE DDDD

Operation:  TEMP ← MI
IFF [N • V] = 0 then PC' ← PC + TEMP

Condition Codes:  Not affected.

Description:  Causes a branch if the N (negative) bit and the V (overflow) bit are either both set or both clear. That is, branch if the sign of a valid twos complement result is, or would be, positive. When used after a subtract or compare operation on twos complement values, this instruction will branch if the register was greater than or equal to the memory operand.

Addressing Mode:  Relative
**BGT**

**Branch on Greater**

**Source Forms:**
BGT dd; LBGT DDDD

**Operation:**
TEMP ← MI

\[
\text{IFF } Z \land [N \lor V] = 0 \text{ then } PC' \leftarrow PC + \text{TEMP}
\]

**Condition Codes:**
Not affected.

**Description:**
Causes a branch if the N (negative) bit and V (overflow) bit are either both set or both clear and the Z (zero) bit is clear. In other words, branch if the sign of a valid two's complement result is, or would be, positive and not zero. When used after a subtract or compare operation on two's complement values, this instruction will branch if the register was greater than the memory operand.

**Addressing Mode:**
Relative
BHI  

Branch If Higher

Source Forms:  BHI dd; LBHI DDDD

Operation:  TEMP ← MI
IFF [C v Z] = 0 then PC' ← PC + TEMP

Condition Codes:  Not affected.

Description:  Causes a branch if the previous operation caused neither a carry nor a zero result. When used after a subtract or compare operation on unsigned binary values, this instruction will branch if the register was higher than the memory operand.

Addressing Mode:  Relative

Comments:  Generally not useful after INC/DEC, LD/TST, and TST/CLR/COM instructions.
BHS

Branch if Higher or Same

Source Forms: BHS dd; LBHS DDDD

Operation:
TEMP ← MI
IFF C = 0 then PC' ← PC + MI

Condition Codes: Not affected.

Description: Tests the state of the C (carry) bit and causes a branch if it is clear. When used after a subtract or compare on unsigned binary values, this instruction will branch if the register was higher than or the same as the memory operand.

Addressing Mode: Relative

Comments: This is a duplicate assembly-language mnemonic for the single machine instruction BCC. Generally not useful after INC/DEC, LD/ST, and TST/CLR/COM instructions.
Source Form: Bit P

Operation: \( \text{TEMP} \leftarrow R \land M \)

Condition Codes:
- H — Not affected.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Always cleared.
- C — Not affected.

Description: Performs the logical AND of the contents of accumulator A or B and the contents of memory location M and modifies the condition codes accordingly. The contents of accumulator A or B and memory location M are not affected.

Addressing Modes: Immediate
- Extended
- Direct
- Indexed
**BLE**

*Branch on Less than or Equal to Zero*

**Source Forms:**

BLE dd; LBLE DDDD

**Operation:**

\[
\text{TEMP} - \text{MI} \\
\text{IFF } Z \text{ v } [N \oplus V] = 1 \text{ then } \text{PC}' = \text{PC} + \text{TEMP}
\]

**Condition Codes:**

Not affected.

**Description:**

Causes a branch if the exclusive OR of the N (negative) and V (overflow) bits is 1 or if the Z (zero) bit is set. That is, branch if the sign of a valid twos complement result is, or would be, negative. When used after a subtract or compare operation on twos complement values, this instruction will branch if the register was less than or equal to the memory operand.

**Addressing Mode:**

Relative
BLO  Branch on Lower  BLO

Source Forms:  BLO dd; LBLO DDDD

Operation:  TEMP ← MI
IFF C = 1 then PC' ← PC + TEMP

Condition Codes:  Not affected.

Description:  Tests the state of the C (carry) bit and causes a branch if it is set. When used after a subtract or compare on unsigned binary values, this instruction will branch if the register was lower than the memory operand.

Addressing Mode:  Relative

Comments:  This is a duplicate assembly-language mnemonic for the single machine instruction BCS. Generally not useful after INC/DEC, LD/ST, and TST/CLR/COM instructions.
BLS

Branch on Lower or Same

Source Forms: BLS dd; LBLS DDDD

Operation: TEMP ← MI
IFF (C v Z) = 1 then PC′ ← PC + TEMP

Condition Codes: Not affected.

Description: Causes a branch if the previous operation caused either a carry or a zero result. When used after a subtract or compare operation on unsigned binary values, this instruction will branch if the register was lower than or the same as the memory operand.

Addressing Mode: Relative

Comments: Generally not useful after INC/DEC, LD/ST, and TST/CLR/COM instructions.
BLT Branch on Less than Zero

Source Forms: BLT dd; LBLT DDDD

Operation: TEMP ← MI
IFF [N ≡ V] = 1 then PC' ← PC + TEMP

Condition Codes: Not affected.

Description: Causes a branch if either, but not both, of the N (negative) or V (overflow) bits is set. That is, branch if the sign of a valid twos complement result is, or would be, negative. When used after a subtract or compare operation on twos complement binary values, this instruction will branch if the register was less than the memory operand.

Addressing Mode: Relative
**BMI**  
**Branch on Minus**  

**Source Forms:**  
BMI dd; LBMI DDDD

**Operation:**  
TEMP ← MI  
IFF N = 1 then PC' ← PC + TEMP

**Condition Codes:**  
Not affected.

**Description:**  
Tests the state of the N (negative) bit and causes a branch if set. That is, branch if the sign of the two's complement result is negative.

**Addressing Mode:**  
Relative

**Comments:**  
When used after an operation on signed binary values, this instruction will branch if the result is minus. It is generally preferred to use the LBLT instruction after signed operations.
**BNE**

Branch Not Equal

**Source Forms:**

BNE dd; LBNE DDDD

**Operation:**

\[
\begin{align*}
\text{TEMP} & \leftarrow \text{MI} \\
\text{IFF } Z = 0 \text{ then } \text{PC}' & \leftarrow \text{PC} + \text{TEMP}
\end{align*}
\]

**Condition Codes:**

Not affected.

**Description:**

Tests the state of the Z (zero) bit and causes a branch if it is clear. When used after a subtract or compare operation on any binary values, this instruction will branch if the register is, or would be, not equal to the memory operand.

**Addressing Mode:**

Relative
BPL

Branch on Plus

Source Forms: BPL dd; LBPL DDDD

Operation: TEMP ← MI
IFF N = 0 then PC' ← PC + TEMP

Condition Codes: Not affected.

Description: Tests the state of the N (negative) bit and causes a branch if it is clear. That is, branch if the sign of the twos complement result is positive.

Addressing Mode: Relative

Comments: When used after an operation on signed binary values, this instruction will branch if the result (possibly invalid) is positive. It is generally preferred to use the BGE instruction after signed operations.
BRA

Branch Always

Source Forms: BRA dd; LBRA DDDD

Operation:

TEMP ← M1
PC' ← PC + TEMP

Condition Codes: Not affected.

Description: Causes an unconditional branch.

Addressing Mode: Relative
Source Forms: BRN dd; LBRN DDDD
Operation: TEMP — MI
Condition Codes: Not affected.
Description: Does not cause a branch. This instruction is essentially a no operation, but has a bit pattern logically related to branch always.
Addressing Mode: Relative
BSR - Branch to Subroutine

**Source Forms:**

BSR dd; LBSR DDDD

**Operation:**

TEMP ← MI  
SP′ ← SP − 1, (SP) ← PCL  
SP′ ← SP − 1, (SP) ← PCH  
PC′ ← PC + TEMP

**Condition Codes:**

Not affected.

**Description:**

The program counter is pushed onto the stack. The program counter is then loaded with the sum of the program counter and the offset.

**Addressing Mode:**

Relative

**Comments:**

A return from subroutine (RTS) instruction is used to reverse this process and must be the last instruction executed in a subroutine.
BVC
Branch on Overflow Clear

Source Forms: BVC dd; LBVC DDDD

Operation: TEMP — M1
IFF V = 0 then PC' = PC + TEMP

Condition Codes: Not affected.

Description: Tests the state of the V (overflow) bit and causes a branch if it is clear. That is, branch if the twos complement result was valid. When used after an operation on twos complement binary values, this instruction will branch if there was no overflow.

Addressing Mode: Relative
BVS

Branch on Overflow Set

Source Forms: BVS dd; LBVS DDDD

Operation: TEMP — MI
IFF V = 1 then PC' — PC + TEMP

Condition Codes: Not affected.

Description: Tests the state of the V (overflow) bit and causes a branch if it is set. That is, branch if the twos complement result was invalid. When used after an operation on twos complement binary values, this instruction will branch if there was an overflow.

Addressing Mode: Relative
CLR

Clear

Source Form: CLR Q

Operation: TEMP — M
           M — 0016

Condition Codes: H — Not affected.
                 N — Always cleared.
                 Z — Always set.
                 V — Always cleared.
                 C — Always cleared.

Description: Accumulator A or B or memory location M is loaded with 00000000.
             Note that the EA is read during this operation.

Addressing Modes: Inherent
                 Extended
                 Direct
                 Indexed
CMP (8-Bit)  Compare Memory from Register  CMP (8-Bit)

Source Forms:  CMPA P; CMPB P

Operation:  TEMP ← R − M

Condition Codes:  H — Undefined.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.

Description:  Compares the contents of memory location to the contents of the specified register and sets the appropriate condition codes. Neither memory location M nor the specified register is modified. The carry flag represents a borrow and is set to the inverse of the resulting binary carry.

Addressing Modes:  Immediate
Extended
Direct
Indexed
CMP (16-Bit) Compare Memory from Register CMP (16-Bit)

Source Forms: CMPD P; CMPX P; CMPY P; CMPU P; CMPS P

Operation: TEMP ← R − M:M + 1

Condition Codes: H — Not affected.  
N — Set if the result is negative; cleared otherwise.  
Z — Set if the result is zero; cleared otherwise.  
V — Set if an overflow is generated; cleared otherwise.  
C — Set if a borrow is generated; cleared otherwise.

Description: Compares the 16-bit contents of the concatenated memory locations M:M + 1 to the contents of the specified register and sets the appropriate condition codes. Neither the memory locations nor the specified register is modified unless autoincrement or autodecrement are used. The carry flag represents a borrow and is set to the inverse of the resulting binary carry.

Addressing Modes: Immediate  
Extended  
Direct  
Indexed
Complement

Source Forms: COM Q; COMA; COMB

Operation: $M' \leftarrow O + \overline{M}$

Condition Codes:
- H — Not affected.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Always cleared.
- C — Always set.

Description: Replaces the contents of memory location M or accumulator A or B with its logical complement. When operating on unsigned values, only BEQ and BNE branches can be expected to behave properly following a COM instruction. When operating on twos complement values, all signed branches are available.

Addressing Modes: Inherent
                   Extended
                   Direct
                   Indexed


Source Form: CWAI #$XX

Operation:
CCR ← CCR ∧ MI (Possibly clear masks)
Set E (entire state saved)
SP' ← SP - 1, (SP) ← PCL
SP' ← SP - 1, (SP) ← PCH
SP' ← SP - 1, (SP) ← USL
SP' ← SP - 1, (SP) ← USH
SP' ← SP - 1, (SP) ← IYL
SP' ← SP - 1, (SP) ← IYH
SP' ← SP - 1, (SP) ← IXL
SP' ← SP - 1, (SP) ← IXH
SP' ← SP - 1, (SP) ← DPR
SP' ← SP - 1, (SP) ← ACCB
SP' ← SP - 1, (SP) ← ACCA
SP' ← SP - 1, (SP) ← CCR

Condition Codes: Affected according to the operation.

Description:
This instruction ANDs an immediate byte with the condition code register which may clear the interrupt mask bits I and F, stacks the entire machine state on the hardware stack and then looks for an interrupt. When a non-masked interrupt occurs, no further machine state information need be saved before vectoring to the interrupt handling routine. This instruction replaced the MC6800 CLI WAI sequence, but does not place the buses in a high-impedance state. A FIRQ (fast interrupt request) may enter its interrupt handler with its entire machine state saved. The RTI (return from interrupt) instruction will automatically return the entire machine state after testing the E (entire) bit of the recovered condition code register.

Addressing Mode: Immediate

Comments:
The following immediate values will have the following results:
FF = enable neither
EF = enable IRQ
BF = enable FIRQ
AF = enable both
DAA

Source Form: DAA

Operation: \( \text{ACCA} \leftarrow \text{ACCA} + \text{CF (MSN)}: \text{CF (LSN)} \)
where \( \text{CF} \) is a Correction Factor, as follows: the \( \text{CF} \) for each nibble
(BCD) digit is determined separately, and is either 6 or 0.

Least Significant Nibble
\( \text{CF (LSN)} = 6 \text{ IFF } 1 \) \( \text{C} = 1 \)
or 2) \( \text{LSN} > 9 \)

Most Significant Nibble
\( \text{CF (MSN)} = 6 \text{ IFF } 1 \) \( \text{C} = 1 \)
or 2) \( \text{MSN} > 9 \)
or 3) \( \text{MSN} > 8 \text{ and } \text{LSN} > 9 \)

Condition Codes:

- **H** — Not affected.
- **N** — Set if the result is negative; cleared otherwise.
- **Z** — Set if the result is zero; cleared otherwise.
- **V** — Undefined.
- **C** — Set if a carry is generated or if the carry bit was set before the
  operation; cleared otherwise.

Description:
The sequence of a single-byte add instruction on accumulator A
(either ADDA or ADCA) and a following decimal addition adjust instruction results in a BCD addition with an appropriate carry bit.
Both values to be added must be in proper BCD form (each nibble such that: \( 0 \leq \text{nibble} \leq 9 \)). Multiple-precision addition must add the
carry generated by this decimal addition adjust into the next higher digit during the add operation (ADCA) immediately prior to the next
decimal addition adjust.

Addressing Mode: Inherent
DEC

Decrement

Source Forms:  DEC Q; DECA; DECB

Operation:  \(M' - M - 1\)

Condition Codes:
- H — Not affected.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Set if the original operand was 10000000; cleared otherwise.
- C — Not affected.

Description: Subtract one from the operand. The carry bit is not affected, thus allowing this instruction to be used as a loop counter in multiple-precision computations. When operating on unsigned values, only BEQ and BNE branches can be expected to behave consistently. When operating on twos complement values, all signed branches are available.

Addressing Modes: Inherent
                   Extended
                   Direct
                   Indexed
EOR

Exclusive OR

Source Forms: EORA P; EORB P

Operation: \( R' = R \cdot M \)

Condition Codes:
- **H** — Not affected.
- **N** — Set if the result is negative; cleared otherwise.
- **Z** — Set if the result is zero; cleared otherwise.
- **V** — Always cleared.
- **C** — Not affected.

Description: The contents of memory location \( M \) is exclusive ORed into an 8-bit register.

Addressing Modes: Immediate
Extended
Direct
Indexed
Source Form: EXG R1, R2

Operation: R1 ← R2

Condition Codes: Not affected (unless one of the registers is the condition code register).

Description: Exchanges data between two designated registers. Bits 3-0 of the postbyte define one register, while bits 7-4 define the other, as follows:

0000 = A:B  
0001 = X  
0010 = Y  
0011 = US  
0100 = SP  
0101 = PC  
0110 = Undefined  
0111 = Undefined

1000 = A  
1001 = B  
1010 = CCR  
1011 = DPR  
1100 = Undefined  
1101 = Undefined  
1110 = Undefined  
1111 = Undefined

Only like size registers may be exchanged. (8-bit with 8-bit or 16-bit with 16-bit.)

Addressing Mode: Immediate
INC

Increment

Source Forms: INC Q; INCA; INCB

Operation: M' ← M + 1

Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if the original operand was 01111111; cleared otherwise.
C — Not affected.

Description: Adds to the operand. The carry bit is not affected, thus allowing this instruction to be used as a loop counter in multiple-precision computations. When operating on unsigned values, only the BEQ and BNE branches can be expected to behave consistently. When operating on twos complement values, all signed branches are correctly available.

Addressing Modes: Inherent
Extended
Direct
Indexed
JMP

Jump

Source Form: JMP EA
Operation: PC' → EA
Condition Codes: Not affected.
Description: Program control is transferred to the effective address.
Addressing Modes: Extended
Direct
Indexed
JSR

Jump to Subroutine

Source Form: JSR EA

Operation:
SP' ← SP − 1, (SP) ← PCL
SP' ← SP − 1, (SP) ← PCH
PC' ← EA

Condition Codes: Not affected.

Description: Program control is transferred to the effective address after storing the return address on the hardware stack. A RTS instruction should be the last executed instruction of the subroutine.

Addressing Modes: Extended
 Direct
 Indexed
LD (8-Bit)  Load Register from Memory  LD (8-Bit)

Source Forms:  LDA P; LDB P

Operation:  \( R' \leftarrow M \)

Condition Codes:  
- \( H \) — Not affected.
- \( N \) — Set if the loaded data is negative; cleared otherwise.
- \( Z \) — Set if the loaded data is zero; cleared otherwise.
- \( V \) — Always cleared.
- \( C \) — Not affected.

Description:  Loads the contents of memory location \( M \) into the designated register.

Addressing Modes:  Immediate
- Extended
- Direct
- Indexed
LD (16-Bit)  Load Register from Memory  LD (16-Bit)

Source Forms: LDD P; LDX P; LDY P; LDS P; LDU P

Operation: $R' \leftarrow M : M + 1$

Condition Codes:  
- H — Not affected. 
- N — Set if the loaded data is negative; cleared otherwise. 
- Z — Set if the loaded data is zero; cleared otherwise. 
- V — Always cleared. 
- C — Not affected.

Description: Load the contents of the memory location $M : M + 1$ into the designated 16-bit register.

Addressing Modes: Immediate 
- Extended 
- Direct 
- Indexed
LEA
Load Effective Address

Source Forms: LEAX, LEAY, LEAS, LEAU

Operation: R’ ← EA

Condition Codes:
H — Not affected.
N — Not affected.
Z — LEAX, LEAY: Set if the result is zero; cleared otherwise.
LEAS, LEAU: Not affected.
V — Not affected.
C — Not affected.

Description: Calculates the effective address from the indexed addressing mode and places the address in an indexable register.

LEAX and LEAY affect the Z (zero) bit to allow use of these registers as counters and for MC6800 INX/DEX compatibility.

LEAU and LEAS do not affect the Z bit to allow cleaning up the stack while returning the Z bit as a parameter to a calling routine, and also for MC6800 INS/DES compatibility.

Addressing Mode: Indexed

Comments: Due to the order in which effective addresses are calculated internally, the LEAX, X+ and LEAX, X+ do not add 2 and 1 respectively to the X register; but instead leave the X register unchanged. This also applies to the Y, U, and S registers. For the expected results, use the faster instruction LEAX 2, X and LEAX 1, X.

Some examples of LEA instruction uses are given in the following table.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEAX 10, X</td>
<td>X + 10 – X</td>
<td>Adds 5-bit constant 10 to X</td>
</tr>
<tr>
<td>LEAX 500, X</td>
<td>X + 500 – X</td>
<td>Adds 16-bit constant 500 to X</td>
</tr>
<tr>
<td>LEAY A, Y</td>
<td>Y + A – Y</td>
<td>Adds 8-bit accumulator to Y</td>
</tr>
<tr>
<td>LEAY D, Y</td>
<td>Y + D – Y</td>
<td>Adds 16-bit D accumulator to Y</td>
</tr>
<tr>
<td>LEAU –10, U</td>
<td>U – 10 – U</td>
<td>Subtracts 10 from U</td>
</tr>
<tr>
<td>LEAS –10, S</td>
<td>S – 10 – S</td>
<td>Used to reserve area on stack</td>
</tr>
<tr>
<td>LEAS 10, S</td>
<td>S + 10 – S</td>
<td>Used to ‘clean up’ stack</td>
</tr>
<tr>
<td>LEAX 5, S</td>
<td>S + 5 – X</td>
<td>Transfers as well as adds</td>
</tr>
</tbody>
</table>

A-45
**LSL**

**Logical Shift Left**

**Source Forms:** LSL Q; LSLA; LSLB

**Operation:**

\[ C \leftarrow \underbrace{\_\_\_\_\_\_\_\_\_\_}_{b7} \underbrace{\_\_\_\_\_\_\_\_\_}_{b0} \leftarrow 0 \]

**Condition Codes:**
- **H** — Undefined.
- **N** — Set if the result is negative; cleared otherwise.
- **Z** — Set if the result is zero; cleared otherwise.
- **V** — Loaded with the result of the exclusive OR of bits six and seven of the original operand.
- **C** — Loaded with bit seven of the original operand.

**Description:** Shifts all bits of accumulator A or B or memory location M one place to the left. Bit zero is loaded with a zero. Bit seven of accumulator A or B or memory location M is shifted into the C (carry) bit.

**Addressing Modes:** Inherent
- Extended
- Direct
- Indexed

**Comments:** This is a duplicate assembly-language mnemonic for the single machine instruction ASL.
LSR

Logical Shift Right

Source Forms: LSR Q; LSRA; LSRB

Operation: \[0 \rightarrow \begin{array}{cccccc}
\vdots \\
b_7 & b_6 & b_5 & b_4 & b_3 & b_2 \\
\vdots \\
C
\end{array} \rightarrow C\]

Condition Codes:
- H — Not affected.
- N — Always cleared.
- Z — Set if the result is zero; cleared otherwise.
- V — Not affected.
- C — Loaded with bit zero of the original operand.

Description: Performs a logical shift right on the operand. Shifts a zero into bit seven and bit zero into the C (carry) bit.

Addressing Modes:
- Inherent
- Extended
- Direct
- Indexed
Source Form: MUL

Operation: ACCA\*ACCB' $\rightarrow$ ACCA x ACCB

Condition Codes:
- **H** — Not affected.
- **N** — Not affected.
- **Z** — Set if the result is zero; cleared otherwise.
- **V** — Not affected.
- **C** — Set if ACCB bit 7 of result is set; cleared otherwise.

Description: Multiply the unsigned binary numbers in the accumulators and place the result in both accumulators (ACCA contains the most-significant byte of the result). Unsigned multiply allows multiple-precision operations.

Addressing Mode: Inherent

Comments: The C (carry) bit allows rounding the most-significant byte through the sequence: MUL, ADCA #0.
NEG

Negate

Source Forms: NEG Q; NEGA; NEGB

Operation: \( M' \leftarrow 0 - M \)

Condition Codes:
- \( H \) — Undefined.
- \( N \) — Set if the result is negative; cleared otherwise.
- \( Z \) — Set if the result is zero; cleared otherwise.
- \( V \) — Set if the original operand was 10000000.
- \( C \) — Set if a borrow is generated; cleared otherwise.

Description: Replaces the operand with its twos complement. The \( C \) (carry) bit represents a borrow and is set to the inverse of the resulting binary carry. Note that 80\(16\) is replaced by itself and only in this case is the \( V \) (overflow) bit set. The value 00\(16\) is also replaced by itself, and only in this case is the \( C \) (carry) bit cleared.

Addressing Modes: Inherent
- Extended
- Direct
NOP

No Operation

Source Form: NOP

Operation: Not affected.

Condition Codes: This instruction causes only the program counter to be incremented. No other registers or memory locations are affected.

Addressing Mode: Inherent
OR

Inclusive OR Memory Into Register

Source Forms: ORA P; ORB P

Operation: R' ← R v M

Condition Codes: H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.

Description: Performs an inclusive OR operation between the contents of accumulator A or B and the contents of memory location M and the result is stored in accumulator A or B.

Addressing Modes: Immediate
Extended
Direct
Indexed
OR  Inclusive OR Memory Immediate into Condition Code Register

Source Form: ORCC #XX

Operation: R' ← R v MI

Condition Codes: Affected according to the operation.

Description: Performs an Inclusive OR operation between the contents of the condition code registers and the immediate value, and the result is placed in the condition code register. This instruction may be used to set interrupt masks (disable interrupts) or any other bit(s).

Addressing Mode: Immediate
PSHS

Push Registers on the Hardware Stack

Source Form:
PSHS register list
PSHS #LABEL
Postbyte:

<table>
<thead>
<tr>
<th>PC</th>
<th>U</th>
<th>Y</th>
<th>X</th>
<th>DP</th>
<th>B</th>
<th>A</th>
<th>CC</th>
</tr>
</thead>
</table>

push order-----

Operation:
IFF b7 of postbyte set, then:  
SP' ← SP - 1, (SP) ← PCL
SP' ← SP - 1, (SP) ← PCH

IFF b6 of postbyte set, then:  
SP' ← SP - 1, (SP) ← USL
SP' ← SP - 1, (SP) ← USH

IFF b5 of postbyte set, then:  
SP' ← SP - 1, (SP) ← IYL
SP' ← SP - 1, (SP) ← IYH

IFF b4 of postbyte set, then:  
SP' ← SP - 1, (SP) ← IXL
SP' ← SP - 1, (SP) ← IXH

IFF b3 of postbyte set, then:  
SP' ← SP - 1, (SP) ← DPR

IFF b2 of postbyte set, then:  
SP' ← SP - 1, (SP) ← ACCB

IFF b1 of postbyte set, then:  
SP' ← SP - 1, (SP) ← ACCA

IFF b0 of postbyte set, then:  
SP' ← SP - 1, (SP) ← CCR

Condition Codes: Not affected.

Description: All, some, or none of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself).

Addressing Mode: Immediate

Comments: A single register may be placed on the stack with the condition codes set by doing an autodecrement store onto the stack (example: STX , − − S).
PSHU

Push Registers on the User Stack

Source Form:
PSHU register list
PSHU #LABEL

Postbyte:
b7 b6 b5 b4 b3 b2 b1 b0

Operation:
IFF b7 of postbyte set, then: US′ ← US − 1, (US) ← PCL
IFF b6 of postbyte set, then: US′ ← US − 1, (US) ← PCH
IFF b5 of postbyte set, then: US′ ← US − 1, (US) ← SPL
IFF b4 of postbyte set, then: US′ ← US − 1, (US) ← SPH
IFF b3 of postbyte set, then: US′ ← US − 1, (US) ← IYL
IFF b2 of postbyte set, then: US′ ← US − 1, (US) ← IYH
IFF b1 of postbyte set, then: US′ ← US − 1, (US) ← IXL
IFF b0 of postbyte set, then: US′ ← US − 1, (US) ← IXH

Condition Codes: Not affected.

Description: All, some, or none of the processor registers are pushed onto the user stack (with the exception of the user stack pointer itself).

Addressing Mode: Immediate

Comments: A single register may be placed on the stack with the condition codes set by doing an autodecrement store onto the stack (example: STX , − − U).
Source Form: 
PULS register list  
PULS #LABEL  
Postbyte: 
b7 b6 b5 b4 b3 b2 b1 b0  
PC U Y X DP B A CC  
←-----pull order

Operation:  
IFF b0 of postbyte set, then:  
CCR' ←(SP), SP'←SP + 1  
IFF b1 of postbyte set, then:  
ACCA' ←(SP), SP'←SP + 1  
IFF b2 of postbyte set, then:  
ACCB' ←(SP), SP'←SP + 1  
IFF b3 of postbyte set, then:  
DPR' ←(SP), SP'←SP + 1  
IFF b4 of postbyte set, then:  
IXH' ←(SP), SP'←SP + 1  
IXL' ←(SP), SP'←SP + 1  
IFF b5 of postbyte set, then:  
IYH' ←(SP), SP'←SP + 1  
IYL' ←(SP), SP'←SP + 1  
IFF b6 of postbyte set, then:  
USH' ←(SP), SP'←SP + 1  
USL' ←(SP), SP'←SP + 1  
IFF b7 of postbyte set, then:  
PCH' ←(SP), SP'←SP + 1  
PCL' ←(SP), SP'←SP + 1

Condition Codes:  
May be pulled from stack; not affected otherwise.

Description:  
All, some, or none of the processor registers are pulled from the 
hardware stack (with the exception of the hardware stack pointer 
itself).

Addressing Mode:  
Immediate

Comments:  
A single register may be pulled from the stack with condition codes 
set by doing an autoincrement load from the stack (example:  
LDX ,S + + ).
PULU

Pull Registers from the User Stack

Source Form:
PULU register list
PULU #LABEL
Postbyte:
b7 b6 b5 b4 b3 b2 b1 b0

PC U Y X DP B A CC

IFF b0 of postbyte set, then: CCR' ←(US), US' ← US + 1
IFF b1 of postbyte set, then: ACCA' ←(US), US' ← US + 1
IFF b2 of postbyte set, then: ACCB' ←(US), US' ← US + 1
IFF b3 of postbyte set, then: DPR' ←(US), US' ← US + 1
IFF b4 of postbyte set, then: IXH' ←(US), US' ← US + 1
IFF b5 of postbyte set, then: IYH' ←(US), US' ← US + 1
IFF b6 of postbyte set, then: SPL' ←(US), US' ← US + 1
IFF b7 of postbyte set, then: PCH ←(US), US' ← US + 1

Condition Codes:
May be pulled from stack; not affected otherwise.

Description:
All, some, or none of the processor registers are pulled from the user stack (with the exception of the user stack pointer itself).

Addressing Mode:
Immediate

Comments:
A single register may be pulled from the stack with condition codes set by doing an autoincrement load from the stack (example: LDX ,U ++ ).
**ROL**

*Rotate Left*

**Source Forms:**

ROL Q; ROLA; RolB

**Operation:**

```
+---+---+---+---+---+---+---+---+
|   |   |   | C |   |   |   |   |
+---+---+---+---+---+---+---+---+
  b7 b6 b5 b4 b3 b2 b1 b0
```

**Condition Codes:**

- **H** — Not affected.
- **N** — Set if the result is negative; cleared otherwise.
- **Z** — Set if the result is zero; cleared otherwise.
- **V** — Loaded with the result of the exclusive OR of bits six and seven of the original operand.
- **C** — Loaded with bit seven of the original operand.

**Description:**

Rotates all bits of the operand one place left through the C (carry) bit. This is a 9-bit rotation.

**Addressing Mode:**

- Inherent
- Extended
- Direct
- Indexed
**ROR**

*Rotate Right*

**Source Forms:**
ROR Q; RORA; RORB

**Operation:**
![Diagram of ROR operation]

**Condition Codes:**
- **H** — Not affected.
- **N** — Set if the result is negative; cleared otherwise.
- **Z** — Set if the result is zero; cleared otherwise.
- **V** — Not affected.
- **C** — Loaded with bit zero of the previous operand.

**Description:**
Rotates all bits of the operand one place right through the C (carry) bit. This is a 9-bit rotation.

**Addressing Modes:**
- Inherent
- Extended
- Direct
- Indexed
RTI

Source Form: RTI

Operation: $\text{CCR}' \leftarrow (\text{SP}), \text{SP}' \leftarrow \text{SP} + 1$, then

  IFF CCR bit E is set, then:
  \begin{align*}
  \text{ACCA}' & \leftarrow (\text{SP}), \text{SP}' \leftarrow \text{SP} + 1 \\
  \text{ACCB}' & \leftarrow (\text{SP}), \text{SP}' \leftarrow \text{SP} + 1 \\
  \text{DPR}' & \leftarrow (\text{SP}), \text{SP}' \leftarrow \text{SP} + 1 \\
  \text{IXH}' & \leftarrow (\text{SP}), \text{SP}' \leftarrow \text{SP} + 1 \\
  \text{IXL}' & \leftarrow (\text{SP}), \text{SP}' \leftarrow \text{SP} + 1 \\
  \text{IYH}' & \leftarrow (\text{SP}), \text{SP}' \leftarrow \text{SP} + 1 \\
  \text{IYL}' & \leftarrow (\text{SP}), \text{SP}' \leftarrow \text{SP} + 1 \\
  \text{USH}' & \leftarrow (\text{SP}), \text{SP}' \leftarrow \text{SP} + 1 \\
  \text{USL}' & \leftarrow (\text{SP}), \text{SP}' \leftarrow \text{SP} + 1 \\
  \text{PCH}' & \leftarrow (\text{SP}), \text{SP}' \leftarrow \text{SP} + 1 \\
  \text{PCL}' & \leftarrow (\text{SP}), \text{SP}' \leftarrow \text{SP} + 1 \\
  \end{align*}

  IFF CCR bit E is clear, then:
  \begin{align*}
  \text{PCH}' & \leftarrow (\text{SP}), \text{SP}' \leftarrow \text{SP} + 1 \\
  \text{PCL}' & \leftarrow (\text{SP}), \text{SP}' \leftarrow \text{SP} + 1 \\
  \end{align*}

Condition Codes: Recovered from the stack.

Description: The saved machine state is recovered from the hardware stack and control is returned to the interrupted program. If the recovered E (entire) bit is clear, it indicates that only a subset of the machine state was saved (return address and condition codes) and only that subset is recovered.

Addressing Mode: Inherent
RTS  Return from Subroutine  RTS

Source Form:  RTS

Operation:  
PCH' ← (SP), SP' ← SP + 1
PCL' ← (SP), SP' ← SP + 1

Condition Codes:  Not affected.

Description:  Program control is returned from the subroutine to the calling program. The return address is pulled from the stack.

Addressing Mode:  Inherent
SBC

Subtract with Borrow

**Source Forms:**
SBC A; SBCB P

**Operation:**
\[ R' \leftarrow R - M - C \]

**Condition Codes:**
- **H** — Undefined.
- **N** — Set if the result is negative; cleared otherwise.
- **Z** — Set if the result is zero; cleared otherwise.
- **V** — Set if an overflow is generated; cleared otherwise.
- **C** — Set if a borrow is generated; cleared otherwise.

**Description:**
Subtracts the contents of memory location M and the borrow (in the C (carry) bit) from the contents of the designated 8-bit register, and places the result in that register. The C bit represents a borrow and is set to the inverse of the resulting binary carry.

**Addressing Modes:**
- Immediate
- Extended
- Direct
- Indexed
SEX

Sign Extended

Source Form: SEX

Operation: If bit seven of ACCB is set then ACCA' ← FF16
            else ACCA' ← 0016

Condition Codes: H — Not affected.
                 N — Set if the result is negative; cleared otherwise.
                 Z — Set if the result is zero; cleared otherwise.
                 V — Not affected.
                 C — Not affected.

Description: This instruction transforms a twos complement 8-bit value in accumulator B into a twos complement 16-bit value in the D accumulator.

Addressing Mode: Inherent
ST (8-Bit)  

Source Forms: STA P; STB P

Operation: M' = R

Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.

Description: Writes the contents of an 8-bit register into a memory location.

Addressing Modes: Extended
                 Direct
                 Indexed

ST (16-Bit)  Store Register Into Memory  ST (16-Bit)

Source Forms:  STD P; STX P; STY P; STS P; STU P

Operation:  \( M':M + 1' \rightarrow R \)

Condition Codes:  
- **H**: Not affected.
- **N**: Set if the result is negative; cleared otherwise.
- **Z**: Set if the result is zero; cleared otherwise.
- **V**: Always cleared.
- **C**: Not affected.

Description:  Writes the contents of a 16-bit register into two consecutive memory locations.

Addressing Modes:  
- Extended
- Direct
- Indexed
**SUB (8-Bit)**  Subtract Memory from Register  **SUB (8-Bit)**

**Source Forms:**  SUBA P; SUBB P

**Operation:**  \( R' \leftarrow R - M \)

**Condition Codes:**  
- **H** — Undefined.
- **N** — Set if the result is negative; cleared otherwise.
- **Z** — Set if the result is zero; cleared otherwise.
- **V** — Set if the overflow is generated; cleared otherwise.
- **C** — Set if a borrow is generated; cleared otherwise.

**Description:**  Subtracts the value in memory location \( M \) from the contents of a designated 8-bit register. The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry.

**Addressing Modes:**  Immediate  
- Extended  
- Direct  
- Indexed
SUB (16-Bit) Subtract Memory from Register SUB (16-Bit)

Source Forms: SUBD P

Operation: R' ← R − M:M + 1

Condition Codes:
- H — Not affected.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Set if the overflow is generated; cleared otherwise.
- C — Set if a borrow is generated; cleared otherwise.

Description:
Subtracts the value in memory location M:M + 1 from the contents of a designated 16-bit register. The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry.

Addressing Modes: Immediate Extended Direct Indexed
SWI

Software Interrupt

Source Form: SWI

Operation: Set E (entire state will be saved)

- SP' ← SP - 1, (SP) ← PCL
- SP' ← SP - 1, (SP) ← PCH
- SP' ← SP - 1, (SP) ← USL
- SP' ← SP - 1, (SP) ← USH
- SP' ← SP - 1, (SP) ← IYL
- SP' ← SP - 1, (SP) ← IYH
- SP' ← SP - 1, (SP) ← IXL
- SP' ← SP - 1, (SP) ← IXH
- SP' ← SP - 1, (SP) ← DPR
- SP' ← SP - 1, (SP) ← ACCB
- SP' ← SP - 1, (SP) ← ACCA
- SP' ← SP - 1, (SP) ← CCR

Set I, F (mask interrupts)

PC' ← (FFFF):(FFFF)

Condition Codes: Not affected.

Description: All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself), and control is transferred through the software interrupt vector. Both the normal and fast interrupts are masked (disabled).

Addressing Mode: Inherent
Source Form: SWI2

Operation:
Set E (entire state saved)
SP' ← SP - 1, (SP) ← PCL
SP' ← SP - 1, (SP) ← PCH
SP' ← SP - 1, (SP) ← USL
SP' ← SP - 1, (SP) ← USH
SP' ← SP - 1, (SP) ← IYL
SP' ← SP - 1, (SP) ← IYH
SP' ← SP - 1, (SP) ← IXL
SP' ← SP - 1, (SP) ← IXH
SP' ← SP - 1, (SP) ← DPR
SP' ← SP - 1, (SP) ← ACCB
SP' ← SP - 1, (SP) ← ACCA
SP' ← SP - 1, (SP) ← CCR
PC' ← (FFFF4):(FFFF5)

Condition Codes: Not affected.

Description: All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself), and control is transferred through the software interrupt 2 vector. This interrupt is available to the end user and must not be used in packaged software. This interrupt does not mask (disable) the normal and fast interrupts.

Addressing Mode: Inherent
Source Form:  SWI 3

Operation:  Set E (entire state will be saved)
            SP' ← SP - 1, (SP) ← PCL
            SP' ← SP - 1, (SP) ← PCH
            SP' ← SP - 1, (SP) ← USL
            SP' ← SP - 1, (SP) ← USH
            SP' ← SP - 1, (SP) ← IYL
            SP' ← SP - 1, (SP) ← IYH
            SP' ← SP - 1, (SP) ← IXL
            SP' ← SP - 1, (SP) ← IXH
            SP' ← SP - 1, (SP) ← DPR
            SP' ← SP - 1, (SP) ← ACCB
            SP' ← SP - 1, (SP) ← ACCA
            SP' ← SP - 1, (SP) ← CCR
            PC' ← (FFFFH) + (FFFF3)

Condition Codes:  Not affected.

Description:  All of the processor registers are pushed onto the hardware stack
              (with the exception of the hardware stack pointer itself), and control
              is transferred through the software interrupt 3 vector. This interrupt
              does not mask (disable) the normal and fast interrupts.

Addressing Mode:  Inherent
SYNC

Synchronize to External Event

Source Form: SYNC
Operation: Stop processing instructions
Condition Codes: Not affected.
Description: When a SYNC instruction is executed, the processor enters a synchronizing state, stops processing instructions, and waits for an interrupt. When an interrupt occurs, the synchronizing state is cleared and processing continues. If the interrupt is enabled, and it lasts three cycles or more, the processor will perform the interrupt routine. If the interrupt is masked or is shorter than three cycles, the processor simply continues to the next instruction. While in the synchronizing state, the address and data buses are in the high-impedance state.

This instruction provides software synchronization with a hardware process. Consider the following example for high-speed acquisition of data:

<table>
<thead>
<tr>
<th>FAST</th>
<th>SYNC</th>
<th>WAIT FOR DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDA</td>
<td>DATA FROM DISC AND CLEAR INTERRUPT</td>
</tr>
<tr>
<td></td>
<td>STA</td>
<td>PUT IN BUFFER</td>
</tr>
<tr>
<td></td>
<td>DECX</td>
<td>COUNT IT, DONE?</td>
</tr>
<tr>
<td></td>
<td>BNE</td>
<td>FAST GO AGAIN IF NOT</td>
</tr>
</tbody>
</table>

The synchronizing state is cleared by any interrupt. Of course, enabled interrupts at this point may destroy the data transfer and, as such, should represent only emergency conditions.

The same connection used for interrupt-driven I/O service may also be used for high-speed data transfers by setting the interrupt mask and using the SYNC instruction as the above example demonstrates.

Addressing Mode: Inherent
TFR
Transfer Register to Register

Source Form: TFR R1, R2
Operation: R1 → R2
Condition Code: Not affected unless R2 is the condition code register.

Description: Transfers data between two designated registers. Bits 7-4 of the postbyte define the source register, while bits 3-0 define the destination register, as follows:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>A</td>
<td>0001</td>
<td>X</td>
</tr>
<tr>
<td>0010</td>
<td>Y</td>
<td>0101</td>
<td>B</td>
</tr>
<tr>
<td>0011</td>
<td>US</td>
<td>1011</td>
<td>DPR</td>
</tr>
<tr>
<td>0100</td>
<td>SP</td>
<td>1100</td>
<td>Undefined</td>
</tr>
<tr>
<td>0101</td>
<td>PC</td>
<td>1101</td>
<td>Undefined</td>
</tr>
<tr>
<td>0110</td>
<td>Undefined</td>
<td>1110</td>
<td>Undefined</td>
</tr>
<tr>
<td>0111</td>
<td>Undefined</td>
<td>1111</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

Only like size registers may be transferred. (8-bit to 8-bit, or 16-bit to 16-bit.)

Addressing Mode: Immediate
TST

Source Forms: TST Q; TSTA; TSTB

Operation: TEMP ← M − 0

Condition Codes: H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.

Description: Set the N (negative) and Z (zero) bits according to the contents of memory location M, and clear the V (overflow) bit. The TST instruction provides only minimum information when testing unsigned values; since no unsigned value is less than zero, BLO and BLS have no utility. While BHI could be used after TST, it provides exactly the same control as BNE, which is preferred. The signed branches are available.

Addressing Modes: Inherent
Extended
Direct
Indexed

Comments: The MC6800 processor clears the C (carry) bit.
FIRQ

Fast Interrupt Request (Hardware Interrupt)

Operation:
IFF F bit clear, then:
SP' ← SP - 1, (SP) ← PCL
SP' ← SP - 1, (SP) ← PCH
Clear E (subset state is saved)
SP' ← SP - 1, (SP) ← CCR
Set F, I (mask further interrupts)
PC' ← (FFF6):(FFF7)

Condition Codes: Not affected.

Description: A FIRQ (fast interrupt request) with the F (fast interrupt request mask) bit clear causes this interrupt sequence to occur at the end of the current instruction. The program counter and condition code register are pushed onto the hardware stack. Program control is transferred through the fast interrupt request vector. An RTI (return from interrupt) instruction returns the processor to the original task. It is possible to enter the fast interrupt request routine with the entire machine state saved if the fast interrupt request occurs after a clear and wait for interrupt instruction. A normal interrupt request has lower priority than the fast interrupt request and is prevented from interrupting the fast interrupt request routine by automatic setting of the I (interrupt request mask) bit. This mask bit could then be reset during the interrupt routine if priority was not desired. The fast interrupt request allows operations on memory, TST, INC, DEC, etc. instructions without the overhead of saving the entire machine state on the stack.

Addressing Mode: Inherent

A-73
**IRQ**  
*Interrupt Request (Hardware Interrupt)*

**Operation:**  
IFF I bit clear, then:  
\[ \text{SP}' \leftarrow \text{SP} - 1, \text{(SP)} \leftarrow \text{PCL} \]
\[ \text{SP}' \leftarrow \text{SP} - 1, \text{(SP)} \leftarrow \text{PCH} \]
\[ \text{SP}' \leftarrow \text{SP} - 1, \text{(SP)} \leftarrow \text{USL} \]
\[ \text{SP}' \leftarrow \text{SP} - 1, \text{(SP)} \leftarrow \text{USH} \]
\[ \text{SP}' \leftarrow \text{SP} - 1, \text{(SP)} \leftarrow \text{IYL} \]
\[ \text{SP}' \leftarrow \text{SP} - 1, \text{(SP)} \leftarrow \text{IYH} \]
\[ \text{SP}' \leftarrow \text{SP} - 1, \text{(SP)} \leftarrow \text{IXL} \]
\[ \text{SP}' \leftarrow \text{SP} - 1, \text{(SP)} \leftarrow \text{IXH} \]
\[ \text{SP}' \leftarrow \text{SP} - 1, \text{(SP)} \leftarrow \text{DPR} \]
\[ \text{SP}' \leftarrow \text{SP} - 1, \text{(SP)} \leftarrow \text{ACCB} \]
\[ \text{SP}' \leftarrow \text{SP} - 1, \text{(SP)} \leftarrow \text{ACCA} \]
Set E (entire state saved)  
\[ \text{SP}' \leftarrow \text{SP} - 1, \text{(SP)} \leftarrow \text{CCR} \]
Set I (mask further IRQ interrupts)  
PC\:' \leftarrow (FFFF)\:(FFF9)

**Condition Codes:**  
Not affected.

**Description:**  
If the I (interrupt request mask) bit is clear, a low level on the IRQ input causes this interrupt sequence to occur at the end of the current instruction. Control is returned to the interrupted program using a RTI (return from interrupt) instruction. A FIRQ (fast interrupt request) may interrupt a normal IRQ (interrupt request) routine and be recognized anytime after the interrupt vector is taken.

**Addressing Mode:**  
Inherent
Non-Maskable Interrupt (Hardware Interrupt)

Operation:

- \( SP' \leftarrow SP - 1 \), \( (SP) \leftarrow PCL \)
- \( SP' \leftarrow SP - 1 \), \( (SP) \leftarrow PCH \)
- \( SP' \leftarrow SP - 1 \), \( (SP) \leftarrow USL \)
- \( SP' \leftarrow SP - 1 \), \( (SP) \leftarrow USH \)
- \( SP' \leftarrow SP - 1 \), \( (SP) \leftarrow IYL \)
- \( SP' \leftarrow SP - 1 \), \( (SP) \leftarrow IYH \)
- \( SP' \leftarrow SP - 1 \), \( (SP) \leftarrow IXL \)
- \( SP' \leftarrow SP - 1 \), \( (SP) \leftarrow IXH \)
- \( SP' \leftarrow SP - 1 \), \( (SP) \leftarrow DPR \)
- \( SP' \leftarrow SP - 1 \), \( (SP) \leftarrow ACCB \)
- \( SP' \leftarrow SP - 1 \), \( (SP) \leftarrow ACCA \)
- Set E (entire state save)
- \( SP' \leftarrow SP - 1 \), \( (SP) \leftarrow CCR \)
- Set I, F (mask interrupts)
- \( PC' \leftarrow (FFFC):(FFFD) \)

Condition Codes: Not affected.

Description: A negative edge on the NMI (non-maskable interrupt) input causes all of the processor's registers (except the hardware stack pointer) to be pushed onto the hardware stack, starting at the end of the current instruction. Program control is transferred through the NMI vector. Successive negative edges on the NMI input will cause successive NMI operations. Non-maskable interrupt operation can be internally blocked by a RESET operation and any non-maskable interrupt that occurs will be latched. If this happens, the non-maskable interrupt operation will occur after the first load into the stack pointer (LDS; TFR r,s; EXG r,s; etc.) after RESET.

Addressing Mode: Inherent
Operation:  
CCR' — X1X1XXXX  
DPR' — 0016  
PC' — (FFFF):(FFFF)

Condition Codes:  Not affected.

Description:  The processor is initialized (required after power-on) to start program execution. The starting address is fetched from the restart vector.

Addressing Mode:  Extended Indirect
APPENDIX B
ASSIST09 MONITOR PROGRAM

B.1 GENERAL DESCRIPTION

The M6809 is a high-performance microprocessor which supports modern programming techniques such as position-independent, reentrancy, and modular programming. For a software monitor to take advantage of such capabilities demands a more refined and sophisticated user interface than that provided by previous monitors. ASSIST09 is a monitor which supports the advanced features that the M6809 makes possible. ASSIST09 features include the following:

- Coded in a position (address) independent manner. Will execute anywhere in the 64K address space.
- Multiple means available for installing user modifications and extensions.
- Full complement of commands for program development including breakpoint and trace.
- Sophisticated monitor calls for completely address-independent user program services.
- RAM work area is located relative to the ASSIST09 ROM, not at a fixed address as with other monitors.
- Easily adapted to real-time environments.
- Hooks for user command tables, I/O handlers, and default specifications.
- A complete user interface with services normally only seen in full disk operating systems.

The concise instruction set of the M6809 allows all of these functions and more to be contained in only 2048 bytes.

The ASSIST09 monitor is easily adapted to run under control of a real-time operating system. A special function is available which allows voluntary time-slicing, as well as forced time-slicing upon the use of several service routines by a user program.

B.2 IMPLEMENTATION REQUIREMENTS

Since ASSIST09 was coded in an address-independent manner, it will properly execute anywhere in the 64K address space of the M6809. However, an assumption must be made regarding the location of a work area needed to hold miscellaneous variables and the default stack location. This work area is called the page work area and it is addressed within ASSIST09 by use of the direct page register. It is located relative to the start of the
ASSIST09 ROM by an offset of $-1900$ hexadecimal. Assuming ASSIST09 resides at the top of the memory address space for direct control of the hardware interrupt vectors, the memory map would appear as shown in Figure B-1.

![Memory Map Diagram](image)

**Figure B-1. Memory Map**

If F800 is not the start of the monitor ROM the addresses would change, but the relative locations would remain the same except for the programmable timer module (PTM) and asynchronous communications interface adapter (ACIA) default addresses which are fixed.

The default console input/output handlers access an ACIA located at E008. For trace commands, a PTM with default address E000 is used to force an NMI so that single instructions may be executed. These default addresses may easily be changed using one of several methods. The console I/O handlers may also be replaced by user routines. The PTM is initialized during the MONITR service call (see Paragraph B.9 SERVICES) to fire up the monitor unless its default address has been changed to zero, in which case no PTM references will occur.

**B.3 INTERRUPT CONTROL**

Upon reset, a vector table is created which contains, among other things, default interrupt vector handler appendage addresses. These routines may easily be replaced by user appendages with the vector swap service described later. The default actions taken by the appendages are as follows:

- **RESET** — Build the ASSIST09 vector table and setup monitor defaults, then invoke the monitor startup routine.
- **SWI** — Request a service from ASSIST09.
- **FI REQ** — An immediate RTI is done.
- **SWI2, SWI3, IRQ, Reserved, NMI** — Force a breakpoint and enter the command processor.
The use of IRQ is recommended as an abort function during program debugging sessions, as breakpoints and other ASSIST09 defaults are reinitialized upon RESET. Only the primary software interrupt instruction (SWI) is used, not the SWI2 or SWI3. This avoids page fault problems which would otherwise occur with a memory management unit as the SWI2 and SWI3 instructions do not disable interrupts.

Counter number one of the PTM is used to cause an NMI interrupt for the trace and breakpoint commands. At RESET the control register for timer one is initialized for tracing purposes. If no tracing or breakpointing is done then the entire PTM is available to the user. Otherwise, only counters two and three are available. Although control register two must be used to initialize control register one, ASSIST09 returns control register two to the same value it has after a RESET occurs. Therefore, the only condition imposed on a user program is that if the "operate/preset" bit in control register one must be turned on, $A7 should be stored, $A6 should be stored if it must be turned off.

B.4 INITIALIZATION

During ASSIST09 execution, a vector table is used to address certain service routines and default values. This table is generated to provide easily changed control information for user modifications. The first byte of the ASSIST09 ROM contains the start of a subroutine which initializes the vector table along with setting up certain default values before returning to the caller.

If the ASSIST09 RESET vector receives control, it does three things:

1. Assigns a default stack in the work space,
2. Calls the aforementioned subroutine to initialize the vector table, and
3. Fires up the ASSIST09 monitor proper with a MONITR SWI service request.

However, a user routine can perform the same functions with a bonus. After calling the vector initialization subroutine, it may examine or alter any of the vector table values before starting normal ASSIST09 processing. Thus, a user routine may "bootstrap" ASSIST09 and alter the default standard values.

Another method of inserting user modifications is to have a user routine reside at an extension ROM location 2K below the start of the ASSIST09 ROM. The vector table initialization routine mentioned above, looks for a "BRA" flag ($20FE) at this address, and if found calls the location following the flag as a subroutine with the U register pointing to the vector table. Since this is done after vector table initialization, any or all defaults may be altered at this time. A big advantage to using this method is that the modifications are "automatic" in that upon a RESET condition the changes are made without overt action required such as the execution of a memory change command.

No special stack is used during ASSIST09 processing. This means that the stack pointer must be valid at all interruptable times and should contain enough room for the stacking of at least 21 bytes of information. The stack in use during the initial MONITR service call to start up ASSIST09 processing becomes the "official" stack. If any later stack validity checks occur, this same stack will be re-based before entering the command handler.
ASSIST09 uses a work area which is addressed at an offset from the start of the ASSIST09 ROM. The offset value is -1900 hexadecimal. This points to the base page used during monitor execution and contains the vector table as well as the start of the default stack. If the default stack is used and it exceeds 81 bytes in size, then contiguous RAM must exist below this base work page for proper extension of the stack.

B5. INPUT/OUTPUT CONTROL

Output generated by use of the ASSIST09 services may be halted by pressing any key, causing a 'FREEZE' mode to be entered. The next keyboard entry will release this condition allowing normal output to continue. Commands which generate large amounts of output may be aborted by entering CANCEL (CONTROL-X). User programs may also monitor for CANCEL along with the 'FREEZE' condition even when not performing console I/O (PAUSE service).

B.6 COMMAND FORMAT

There are three possible formats for a command:

<Command> CR
<Command> <Expression1> CR
<Command> <Expression1> <Expression2> CR

The space character is used as the delimiter between the command and all arguments. Two special quick commands need no carriage return, "." and "//". To re-enter a command once a mistake is made, type the CANCEL (CONTROL-X) key.

Each "expression" above consists of one or more values separated by an operator. Values can be hex strings, the letters "P", "M", and "W", or the result of a function. Each hexadecimal string is converted internally to a 16-bit binary number. The letter "P" stands for the current program counter, "M" for the last memory examine/change address, and "W" for the window value. The window value is set by using the WINDOW command.

One function exists and it is the INDIRECT function. The character "@" following a value replaces that value with the 16-bit number obtained by using that value as an address.

Two operators are allowed, "+" and "-" which cause addition and subtraction. Values are operated on in a left-to-right order.

Examples:

480 — hexadecimal 480
W + 3 — value of window plus three
P-200 — current program counter minus 200 hexadecimal
M-W — current memory pointer minus window value
100@ — value of word addressed by the two bytes at 100 hexadecimal
P + 1@ — value addressed by the word located one byte up from the current program counter
B.7 COMMAND LIST

Table B-1 lists the commands available in the ASSIST09 monitor.

**Table B-1. Command List**

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
<th>Command Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breakpoint</td>
<td>Set, clear, display, or delete breakpoints</td>
<td>B</td>
</tr>
<tr>
<td>Call</td>
<td>Call program as subroutine</td>
<td>C</td>
</tr>
<tr>
<td>Display</td>
<td>Display memory block in hex and ASCII</td>
<td>D</td>
</tr>
<tr>
<td>Encode</td>
<td>Return indexed postbyte value</td>
<td>E</td>
</tr>
<tr>
<td>Go</td>
<td>Start or resume program execution</td>
<td>G</td>
</tr>
<tr>
<td>Load</td>
<td>Load memory from tape</td>
<td>L</td>
</tr>
<tr>
<td>Memory</td>
<td>Examine or alter memory</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>Memory change or examine last referenced</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Memory change or examine</td>
<td></td>
</tr>
<tr>
<td>Null</td>
<td>Set new character and new line padding</td>
<td>N</td>
</tr>
<tr>
<td>Offset</td>
<td>Compute branch offsets</td>
<td>O</td>
</tr>
<tr>
<td>Punch</td>
<td>Punch memory on tape</td>
<td>P</td>
</tr>
<tr>
<td>Registers</td>
<td>Display or alter registers</td>
<td>R</td>
</tr>
<tr>
<td>Stlevel</td>
<td>Alter stack trace level value</td>
<td>S</td>
</tr>
<tr>
<td>Trace</td>
<td>Trace number of instructions</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>Trace one instruction</td>
<td></td>
</tr>
<tr>
<td>Verify</td>
<td>Verify tape to memory load</td>
<td>V</td>
</tr>
<tr>
<td>Window</td>
<td>Set a window value</td>
<td>W</td>
</tr>
</tbody>
</table>

B.8 COMMANDS

Each of the commands are explained on the following pages. They are arranged in alphabetical order by the command name used in the command list. The command name appears at each margin and in slightly larger type for easy reference.
BREAKPOINT

Format:  Breakpoint
         Breakpoint –
         Breakpoint < Address>
         Breakpoint – < Address>

Operation: Set or change the breakpoint table. The first format displays all breakpoints. The second clears the breakpoint table. The third enters an address into the table. The fourth deletes an address from the table. At reset, all breakpoints are deleted. Only instructions in RAM may be breakpointed.

CALL

Format:  Call
         Call < Address>

Operation: Call and execute a user routine as a subroutine. The current program counter will be used unless the address is specified. The user routine should eventually terminate with a “RTS” instruction. When this occurs, a breakpoint will ensue and the program counter will point into the monitor.
DISPLAY

Format:  
Display <From>
Display <From> <Length>
Display <From> <To>

Operation: Display contents of memory in hexadecimal and ASCII characters. The second argument, when entered, is taken to be a length if it is less than the first, otherwise it is the ending address. A default length of 16 decimal is assumed for the first format. The addresses are adjusted to include all bytes within the surrounding modulo 16 address byte boundary. The CANCEL (CONTROL-X) key may be entered to abort the display. Care must be exercised when the last 15 bytes of memory are to be displayed. The <Length> option should always be used in this case to assure proper termination: D FFE0 40

Examples:
D M 10 — Display 16 bytes surrounding the last memory location examined.
D E000 F000 — Display memory from E000 to F000 hex.

ENCODE

Format:  
Encode <Indexed operand>

Operation: The encode command will return the indexing instruction mode postbyte value from the entered assembler-like syntax operand. This is useful when hand coding instructions. The letter “H” is used to indicate the number of hex digits needed in the expression as shown in the following examples:

E ,Y — Return zero offset to Y register postbyte.
E [HHH,PCR] — Return two byte PCR offset using indirection.
E [,S++] — Return autoincrement S by two indirect.
E H,X — Return 5-bit offset from X.

Note that one “H” specifies a 5-bit offset, and that the result given will have zeros in the offset value position. This command does not detect all incorrectly specified syntax or illegal indexing modes.
**GO**

Format:  
Go
Go < Address >

Operation: Execute starting from the address given. The first format will continue from the current program counter setting. If it is a breakpoint no break will be taken. This allows continuation from a breakpoint. The second format will breakpoint if the address specified is in the breakpoint list.

**LOAD**

Format:  
Load
Load < Offset >

Operation: Load a tape file created using the S1-S9 format. The offset option, if used, is added to the address on the tape to specify the actual load address. All offsets are positive, but wrap around memory modulo 64K. Depending on the equipment involved, after the load is complete a few spurious characters may still be sent by the input device and interpreted as command characters. If this happens, a CANCEL (CONTROL-X) should be entered to cause such characters to be ignored. If the load was not successful a "?" is displayed.
MEMORY

Format: MEMORY <Address>/
       <Address>/
       /

Operation: Initiate the memory examine/change function. The second format will not accept an expression for the address, only a hex string. The third format defaults to the address displayed during the last memory change/examine function. (The same value is obtained in expressions by use of the letter “M”.) After activation, the following actions may be taken until a carriage return is entered:

  <Expr>     Replaces the byte with the specified value. The value may be an expression.
  SPACE      Go to next address and print the byte value.
  ,          (Comma) Go to next address without printing the byte value.
  LF         (Line feed) Go to next address and print it along with the byte value on the next line.
  ^          (Circumflex or Up arrow) Go the previous address and print it along with the byte value on the next line.
  /          Print the current address with the byte value on the next line.
  CR         (Carriage return) Terminate the command.
  '<Text>'' Replace succeeding bytes with ASCII characters until the second apostrophe is entered.

If a change attempt fails (i.e., the location is not valid RAM) then a question mark will appear and the next location displayed.
**NULL**

**Format:** Null <Specification>

**Operation:** Set the new line and character padding count values. The expression value is treated as two values. The upper two hex represent the character pad count, and the lower two the new line pad count (triggered by a carriage return). An expression of less than three hex digits will set the character pad count to zero. The values must range from zero to 7F hexadecimal (127 decimal).

Example:

\[
\begin{align*}
N & \quad 3 \quad \text{— Set the character count to zero and new line count to three.} \\
N & \quad 207 \quad \text{— Set character padding count to two and new line count to seven.}
\end{align*}
\]

Settings for TI Silent 700 terminals are:

<table>
<thead>
<tr>
<th>Baud</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>300</td>
<td>4</td>
</tr>
<tr>
<td>1200</td>
<td>317</td>
</tr>
<tr>
<td>2400</td>
<td>72F</td>
</tr>
</tbody>
</table>

**OFFSET**

**Format:** Offset <Offset addr> <To instruction>

**Operation:** Print the one and two byte offsets needed to perform a branch from the first expression to the instruction. Thus, offsets for branches as well as indexed mode instructions which use offsets may be obtained. If only a four byte value is printed, then a short branch count cannot be done between the two addresses.

Example:

\[
0 \quad P+2 \quad A000 \quad \text{— Compute offsets needed from the current program counter plus two to A000.}
\]
PUNCH

Format: Punch <From> <To>

Operation: Punch or record formatted binary object tape in S1-S9 (MIKBUG) format.

REGISTER

Format: Register

Operation: Print the register set and prompt for a change. At each prompt the following may be entered.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPACE</td>
<td>Skip to the next register prompt</td>
</tr>
<tr>
<td>&lt; Expr &gt; SPACE</td>
<td>Replace with the specified value and prompt for the next register.</td>
</tr>
<tr>
<td>&lt; Expr &gt; CR</td>
<td>(carriage return) Replace with the specified value and terminate the command.</td>
</tr>
<tr>
<td>CR</td>
<td>Terminate the command.</td>
</tr>
</tbody>
</table>
STLEVEL

Format:  Stlevel
         Stlevel < Address >

Operation: Set the stack trace level for inhibiting tracing information. As long as the stack is at or above the stack level address, the trace display will continue. However, when lower than the address it is inhibited. This allows tracing of a routine without including all subroutine and lower level calls in the trace information. Note that tracing through a ASSIST09 "SWI" service request may also temporarily suppress trace output as explained in the description of the trace command. The first format sets the stack trace level to the current program stack value.

TRACE

Format:  Trace < Count >
         . (period)

Operation: Trace the specified number of instructions. At each trace, the opcode just executed will be shown along with the register set. The program counter in the register display points to the NEXT instruction to be executed. A CANCEL (CONTROL-X) will prematurely halt tracing. The second format (period) will cause a single trace to occur. Breakpoints have no effect during the trace. Selected portions of a trace may be disabled using the STLEVEL command. Instructions in ROM and RAM may be traced, whereas breakpoints may be done only in RAM. When tracing through a ASSIST09 service request, the trace display will be suppressed starting two instructions into the monitor until shortly before control is returned to the user program. This is done to avoid an inordinate amount of displaying because ASSIST09, at times, performs a sizeable amount of processing to provide the requested services.
VERIFY

Format:  Verify
      Verify <Offset>

Operation: Verify or compare the contents of memory to the tape file. This command has
the same format and operation as a LOAD command except the file is com-
pared to memory. If the verify fails for any reason a "?" is displayed.

WINDOW

Format:  Window <Value>

Operation: Set the window to a value. This value may be referred to when entering ex-
pressions by use of the letter "W". The window may be set to any 16-bit value.
B.9 SERVICES

The following describes services provided by the ASSIST09 monitor. These services are invoked by using the “SWI” instruction followed by a one byte function code. All services are designed to allow complete address independence both in invocation and operation. Unless specified otherwise, all registers are transparent over the “SWI” call. In the following descriptions, the terms “input handler” and “output handler” are used to refer to appendage routines which may be replaced by the user. The default routines perform standard I/O through an ACIA for console operations to a terminal. The ASCII CANCEL code can be entered on most terminals by depressing the CONTROL and X keys simultaneously. A list of services is given in Table B-2.

Table B-2. Services

<table>
<thead>
<tr>
<th>Service</th>
<th>Entry</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obtain input character</td>
<td>INCHP</td>
<td>0</td>
<td>Obtain the input character in register A from the input handler</td>
</tr>
<tr>
<td>Output a character</td>
<td>OUTCH</td>
<td>1</td>
<td>Send the character in the register A to the output handler</td>
</tr>
<tr>
<td>Send string</td>
<td>PDATA1</td>
<td>2</td>
<td>Send a string of characters to the output handler</td>
</tr>
<tr>
<td>Send new line and string</td>
<td>PDATA</td>
<td>3</td>
<td>Send a carriage return, line feed, and string of characters to the output handler</td>
</tr>
<tr>
<td>Convert byte to hex</td>
<td>OUT2HS</td>
<td>4</td>
<td>Display the byte pointed to by the X register in hex</td>
</tr>
<tr>
<td>Convert word to hex</td>
<td>OUT4HS</td>
<td>5</td>
<td>Display the word pointed to by the X register in hex</td>
</tr>
<tr>
<td>Output to next line</td>
<td>PCRLF</td>
<td>6</td>
<td>Send a carriage return and line feed to the output handler</td>
</tr>
<tr>
<td>Send space</td>
<td>SPACE</td>
<td>7</td>
<td>Send a blank to the output handler</td>
</tr>
<tr>
<td>Fireup ASSIST09</td>
<td>MONITR</td>
<td>8</td>
<td>Enter the ASSIST09 monitor</td>
</tr>
<tr>
<td>Vector swap</td>
<td>VCTRSW</td>
<td>9</td>
<td>Examine or exchange a vector table entry</td>
</tr>
<tr>
<td>User breakpoint</td>
<td>BRKPT</td>
<td>10</td>
<td>Display registers and enter the command handler</td>
</tr>
<tr>
<td>Program break and check</td>
<td>PAUSE</td>
<td>11</td>
<td>Stop processing and check for a freeze or cancel condition</td>
</tr>
</tbody>
</table>
BRKPT

User Breakpoint

Code: 10

Arguments: None

Result: A disabled breakpoint is taken. The registers are displayed and the command handler of ASSIST09 is entered.

Description: Establishes user breakpoints. Both SWI2 and SWI3 default appendages cause a breakpoint as well, but do not set the I and F mask bits. However, since they may both be replaced by user routines the breakpoint service always ensures breakpoint availability. These user breakpoints have nothing to do with system breakpoints which are handled differently by the ASSIST09 monitor.

Example: BRKPT EQU 10 INPUT CODE FOR BRKPT
          SWI REQUEST SERVICE
          FCB BRKPT FUNCTION CODE BYTE

INCHP

Obtain Input Character

Code: 0

Arguments: None

Result: Register A contains a character obtained from the input handler.

Description: Control is not returned until a valid input character is received from the input handler. The input character will have its parity bit (bit 7) stripped and forced to a zero. All NULL ($00) and RUBOUT ($7F) characters are ignored and not returned to the caller. The ECHO flag, which may be changed by the vector SWAP service, determines whether or not the input character is echoed to the output handler (full duplex operation). The default at reset is to echo input. When a carriage return ($0D) is received, line feed ($A0) is automatically sent back to the output handler.

Example: INCHNP EQU 0 INPUT CODE FOR INCHP
          SWI PERFORM SERVICE CALL
          FCB FUNCTION FOR INCHNP

          A REGISTER NOW CONTAINS NEXT CHARACTER
MONITR

Startup ASSIST09

Code: 8

Arguments: S→ Stack to become the "official" stack
          DP→ Direct page default for executed user programs
          A=0 Call input and output console initialization handlers and give the
          "ASSIST09" startup message
          A#0 Go directly to the command handler

Result: ASSIST09 is entered and the command handler given control

Description: The purpose for this function is to enter ASSIST09, either after a system
reset, or when a user program desires to terminate. Control is not returned
unless a "GO" or "CALL" command is done without altering the program
counter. ASSIST09 runs on the passed stack, and if a stack error is
detected during user program execution this is the stack that is rebased.
The direct page register value in use remains the default for user program
execution.

The ASSIST09 restart vector routine uses this function to startup monitor
processing after calling the vector build subroutine as explained in INITIALIZATION.

If indicated by the A register, the input and output initialization handlers
are called followed by the sending of the string "ASSIST09" to the output
handler. The programmable timer (PTM) is initialized, if its address is not
zero, such that register 1 can be used for causing an NMI during trace com-
mands. The command handler is then entered to perform the command re-
quest prompt.

Example: MONITR EQU 8
        LOOP
        CLRA
        TFR A,DP
        LEAS STACK, PCR
        SWI
        FCB MONITR
        BRA LOOP

INPUT CODE FOR MONITR

PREPARE ZERO PAGE REGISTER AND
INITIALIZATION PARAMETER
SET DEFAULT PAGE VALUE
SETUP DEFAULT STACK VALUE
REQUEST SERVICE
FUNCTION CODE BYTE
REENTER IF FALLOUT OCCURS
OUTCH

Output a Character

Code: 1

Arguments: Register A contains the byte to transmit.

Result: The character is sent to the output handler

The character is set as follows ONLY if a LINEFEED was the character to transmit:

CC = 0 if normal output occurred.
CC = 1 if CANCEL was entered during output.

Description: If a FREEZE Occurs (any input character is received) then control is not
returned to the user routine until the condition is released. The FREEZE
condition is checked for only when a linefeed is being sent. Padding null
characters ($00) may be sent following the outputted character depending
on the current setting of the NULLS command. For DLE (Data Link Escape),
character nulls are never sent. Otherwise, carriage returns ($00) receive the
new line count of nulls, all other characters the character count of nulls.

Example: OUTCH EQU 1

      LDA #0
      SWI
      FCB OUTCH

INPUT CODE FOR OUTCH
LOAD CHARACTER "0"
SEND OUT WITH MONITOR CODE
SERVICE CODE BYTE

OUT2HS

Convert Byte to Hex

Code: 4

Arguments: Register X points to a byte to display in hex.

Result: The byte is converted to two hex digits and sent to the output handler
followed by a blank.

Example: OUT2HS EQU 4

      LEAX DATA, PCR
      SWI
      FCB OUT2HS

INPUT CODE FOR OUT2HS
POINT TO 'DATA' TO DECODE
REQUEST SERVICE
SERVICE CODE BYTE
OUT4HS

Convert Word to Hex

Code: 5

Arguments: Register X points to a word (two bytes) to display in hex.

Result: The word is converted to four hex digits and sent to the output handler followed by a blank.

Example: OUT4HS EQU 5

        LEAX DATA, PCR
        SWI
        FCB OUT4HS

INPUT CODE FOR OUT4HS

        LOAD 'DATA' ADDRESS TO DECODE
        REQUEST ASSIST09 SERVICE
        SERVICE CODE BYTE

PAUSE

Program Break and Check

Code: 11

Arguments: None

Result: CC = 0 For a normal return.
        CC = 1 If a CANCEL was entered during the interim.

Description: The PAUSE service should be used whenever a significant amount of processing is done by a program without any external interaction (such as console I/O). Another use of the PAUSE service is for the monitoring of FREEZE or CANCEL requests from the input handler. This allows multi-tasking operating systems to receive control and possibly re-dispatch other programs in a timeslice-like fashion. Testing for FREEZE and CANCEL conditions is performed before return. Return may be after other tasks have had a chance to execute, or after a FREEZE condition is lifted. In a one task system, return is always immediate unless a FREEZE occurs.
PCRLF  
Output to Next Line  

Code: 6  
Arguments: None  

Result: A carriage return and line feed are sent to the output handler.  
\[ C = 1 \] if normal output occurred.  
\[ C = 1 \] if CONTROL-X was entered during output.  

Description: If a FREEZE occurs (any input character is received), then control is not returned to the user routine until the condition is released. The string is completely sent regardless of any FREEZE or CANCEL events occurring. Padding characters may be sent as described under the OUTFCH service.  

Example:  

\[
\begin{align*}
\text{PCRLF} & \quad \text{EQU} \quad 6 \\
\text{INPUT CODE PCRLF} \\
\text{SWI} \\
\text{FCT} & \quad \text{PCRLF} \\
\text{REQUEST SERVICE} \\
\text{SERVICE CODE BYTE}
\end{align*}
\]

PDATA  
Send New Line and String  

Code: 3  
Arguments: Register X points to an output string terminated with an ASCII EOT (04).  

Result: The string is sent to the output handler following a carriage return and line feed.  
\[ CC = 0 \] if normal output occurred.  
\[ CC = 1 \] if CONTROL-X was entered during output.  

Description: The output string may contain embedded carriage returns and line feeds thus allowing several lines of data to be sent with one function call. If a FREEZE occurs (any input character is received), then control is not returned to the user routine until the condition is released. The string is completely sent regardless of any FREEZE or CANCEL events occurring. Padding characters may be sent as described by the OUTHCH function.
Send New Line and String
(Continued)

Example:

```
PDATA      EQU 3       INPUT CODE FOR PDATA
MSGOUT FCC  'THIS IS A MULTIPLE LINE MESSAGE.'
FCB $0A, $0D LINE FEED, CARRIAGE RETURN
FCC 'THIS IS THE SECOND LINE.'
FCB $04 STRING TERMINATOR
LEAX MSGOUT, PCR LOAD MESSAGE ADDRESS
SWI REQUEST A SERVICE
FCB PDATA SERVICE CODE BYTE
```

Send String

Code: 2

Arguments: Register X points to an output string terminated with an ASCII EOT ($04).

Result: The string is sent to the output handler.
        CC = 0 if normal output occurred.
        CC = 1 if CONTROL-X was entered during output.

Description: The output string may contain embedded carriage returns and line feeds thus allowing several lines of data to be sent with one function call. If a FREEZE occurs (any input character is received), then control is not returned to the user routine until the condition is released. The string is completely sent regardless of any FREEZE or CANCEL events occurring. Padding characters may be sent as described by the OUTFCH function.

Example:

```
PDATA EQU 2       INPUT CODE FOR PDATA1
MSG FCC  'THIS IS AN OUTPUT STRING'
FCB $04 STRING TERMINATOR
LEAX MSG, PCR LOAD 'MSG' STRING ADDRESS
SWI REQUEST A SERVICE
FCB PDATA1 SERVICE CODE BYTE
```

B-20
SPACE

Single Space Output

Code: 7

Arguments: None

Result: A space is sent to the output handler.

Description: Padding characters may be sent as described under the OUTCH service.

Example: SPACE EQU 7 INPUT CODE SPACE
          SWI REQUEST ASSIST09 SERVICE
          FCB SPACE SERVICE CODE BYTE

VCTRSW

Vector Swap

Code: 9

Arguments: Register A contains the vector swap input code.
Register X contains zero or a replacement value.

Result: Register X contains the previous value for the vector.

Description: The vector swap service examines/alters a word entry in the ASSIST09 vector table. This table contains pointers and default values used during monitor processing. The entry is replaced with the value contained in the X register unless it is zero. The codes available are listed in Table B-3.

Example: VCTRSW EQU 9 INPUT CODE VCTRSW
          .IRQ EQU 12 IRQ APPENDAGE SWAP FUNCTION CODE
          LEAX MYIRQH,PCR LOAD NEW IRQ HANDLER ADDRESS
          LDA #$IRQ LOAD SUBCODE FOR VECTOR SWAP
          SWI REQUEST SERVICE
          FCB VCTRSW SERVICE CODE BYTE
          X NOW HAS THE PREVIOUS APPENDAGE ADDRESS
B.10 VECTOR SWAP SERVICE

The vector swap service allows user modifications of the vector table to be easily installed. Each vector handler, including the one for SWI, performs a validity check on the stack before any other processing. If the stack is not pointing to valid RAM, it is reset to the initial value passed to the MONITR request which fired-up ASSIST09 after RESET. Also, the current register set is printed following a "?" (question mark) and then the command handler is entered. A list of each entry in the vector table is given in Table B-3.

Table B-3. Vector Table Entries

<table>
<thead>
<tr>
<th>Entry</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.AVTBL</td>
<td>0</td>
<td>Returns address of vector table</td>
</tr>
<tr>
<td>.CMDL1</td>
<td>2</td>
<td>Primary command list</td>
</tr>
<tr>
<td>.RSVD</td>
<td>4</td>
<td>Reserved MC6809 interrupt vector appendage</td>
</tr>
<tr>
<td>.SWI3</td>
<td>6</td>
<td>Software interrupt 3 interrupt vector appendage</td>
</tr>
<tr>
<td>.SWI2</td>
<td>8</td>
<td>Software interrupt 2 interrupt vector appendage</td>
</tr>
<tr>
<td>.FIRQ</td>
<td>10</td>
<td>Fast interrupt request vector appendage</td>
</tr>
<tr>
<td>.IRQ</td>
<td>12</td>
<td>Interrupt request vector appendage</td>
</tr>
<tr>
<td>.SWI</td>
<td>14</td>
<td>Software interrupt vector appendage</td>
</tr>
<tr>
<td>.NMI</td>
<td>16</td>
<td>Non-maskable interrupt vector appendage</td>
</tr>
<tr>
<td>.RESET</td>
<td>18</td>
<td>Reset interrupt vector appendage</td>
</tr>
<tr>
<td>.CION</td>
<td>20</td>
<td>Input console initialization routine</td>
</tr>
<tr>
<td>.CIDTA</td>
<td>22</td>
<td>Input data byte from console routine</td>
</tr>
<tr>
<td>.CIOFF</td>
<td>24</td>
<td>Input console shutdown routine</td>
</tr>
<tr>
<td>.COON</td>
<td>26</td>
<td>Output console initialization routine</td>
</tr>
<tr>
<td>.CODTA</td>
<td>28</td>
<td>Output/data byte to console routine</td>
</tr>
<tr>
<td>.COOFF</td>
<td>30</td>
<td>Output console shutdown routine</td>
</tr>
<tr>
<td>.HSDTA</td>
<td>32</td>
<td>High speed display handler routine</td>
</tr>
<tr>
<td>.BSON</td>
<td>34</td>
<td>Punch/load initialization routine</td>
</tr>
<tr>
<td>.BSDTA</td>
<td>36</td>
<td>Punch/load handler routine</td>
</tr>
<tr>
<td>.BSOFF</td>
<td>38</td>
<td>Punch/load shutdown routine</td>
</tr>
<tr>
<td>.PAUSE</td>
<td>40</td>
<td>Processing pause routine</td>
</tr>
<tr>
<td>.CMDL2</td>
<td>44</td>
<td>Secondary command list</td>
</tr>
<tr>
<td>.ACIA</td>
<td>46</td>
<td>Address of ACIA</td>
</tr>
<tr>
<td>.PAD</td>
<td>48</td>
<td>Character and new line pad counts</td>
</tr>
<tr>
<td>.ECHO</td>
<td>50</td>
<td>Echo flag</td>
</tr>
<tr>
<td>.PTM</td>
<td>52</td>
<td>Programmable timer module address</td>
</tr>
</tbody>
</table>

The following pages describe the purpose of each entry and the requirements which must be met for a user replaceable value or routine to be successfully substituted.
**.ACIA**

**ACIA Address**

**Code:** 46

**Description:** This entry contains the address of the ACIA used by the default console input and output device handlers. Standard ASSIST09 initialization sets this value to hexadecimal E008. If this must be altered, then it must be done before the MONITR startup service is invoked, since that service calls the .COON and .COIN input and output device initialization routines which initialize the ACIA pointed to by this vector slot.

---

**.AVTBL**

**Return Address of Vector Table**

**Code:** 0

**Description:** The address of the vector table is returned with this code. This allows mass changes to the table without individual calls to the vector swap service. The code values are identical to the offsets in the vector table. This entry should never be changed, only examined.
.BSDTA  Punch/Load Handler Routine

Code:  36

Description: This entry contains the address of a routine which performs punch, load, and verify operations. The .BSON routine is always executed before the routine is given control. This routine is given the same parameter list documented for .BSON. The default handler uses the .CODTA routine to punch or the .CIDTA routine to read data in S1/S9 (MIKBUG) format. The function code byte must be examined to determine the type request being handled.

A return code must be given which reflects the final processing disposition:

\[ Z = 1 \] Successful completion

or

\[ Z = 0 \] Unsuccessful completion.

The .BSOFF routine will be called after this routine is completed.

.BSOFF  Punch/Load Shutdown Routine

Code:  38

Description: This entry points to a subroutine which is designated to terminate device processing for the punch, load, and verify handler .BSDTA. The stack contains a parameter list as documented for the .BSON entry. The default ASSIST09 routine issues DC4 ($14 or stop) and DC3 ($13 or x-off) followed by a one second delay to give the reader/punch time to stop. Also, an internally used flag by the INCHP service routine is cleared to reverse the effect caused by its setting in the .BSON handler. See that description for an explanation of the proper use of this flag.
.BSON

Punch/Load Initialization Routine

Code:  34

Description: This entry points to a subroutine with the assigned task of turning on the device used for punch, load, and verify processing. The stack contains a parameter list describing which function is requested. The default routine sends an ASCII "reader on" or "punch on" code of DC1 ($11) or DC2 ($12) respectively to the output handler (.CODTA). A flag is also set which disables test for FREEZE conditions during INCHNP processing. This is done so characters are not lost by being interpreted as FREEZE mode indicators. If a user replacement routine also uses the INCHNP service, then it also should set this same byte non-zero and clear it in the .BSOFF routine. The ASSIST09 source listing should be consulted for the location of this byte.

The stack is setup as follows:
- S + 6 = Code byte, VERIFY (-1), PUNCH (0), LOAD (1)
- S + 4 = Start address for punch only
- S + 2 = End address for punch, or offset for READ/LOAD
- S + 0 = Return address

.CIDTA

Input Data Byte from Console Routine

Code:  22

Description: This entry determines the console input handler appendage. The responsibility of this routine is to furnish the requested next input character in the A register, if available, and return with a condition code. The INCHP service routine calls this appendage to supply the next character. Also, a "FREEZE" mode routine calls at various times to test for a FREEZE condition or determine if the CANCEL key has been entered. Processing for this appendage must abide by the following conventions:

Input:  PC → ASSIST09 work page
        S → Return address

Output:  C = 0, A = input character
         C = 1 if no input character is yet available

Volatile Registers:  U, B

The handler should always pass control back immediately even if no character is yet available. This enables other tasks to do productive work while input is unavailable. The default routine reads an ACIA as explained in Paragraph B.2 Implementation Requirements.
.CIOFF
Input Console Shutdown Routine

Code: 24

Description: This entry points to a routine which is called to terminate input processing. It is not called by ASSIST09 at any time, but is included for consistency. The default routine merely does an "RTS". The environment is as follows:

Input: None
Output: Input device terminated
Volatile Registers: None

.CION
Input Console Initialization Routine

Code: 20

Description: This entry is called to initiate the input device. It is called once during the MONITR service which initializes the monitor so the command processor may obtain commands to process. The default handler resets the ACIA used for standard input and output and sets up the following default conditions: 8-bit word length, no parity checking, 2 stop bits, divide-by-16 counter ratio. The effect of an 8-bit word with no parity checking is to accept 7-bit ASCII and ignore the parity bit.

Input: .ACIA Memory address of the ACIA
Output: The output device is initialized
Volatile Registers: A, X
**Description:** User supplied command tables may either substitute or replace the ASSIST09 standard tables. The command handler scans two lists, the primary table first followed by the secondary table. The primary table is pointed to by this entry and contains, as a default, the ASSIST09 command table. The secondary table defaults to a null list. A user may insert their own table into either position. If a user list is installed in the secondary table position, then the ASSIST09 list will be searched first. The default ASSIST09 list contains all one character command names. Thus, a user command "PRINT" would be matched if the letters "PR" are typed, but not just a "P" since the system command list would match first. A user may replace the primary system list if desired. A command is chosen on a first match basis comparing only the character(s) entered. This means that two or more commands may have the same initial characters and that if only that much is entered then the first one in the list(s) is chosen.

Each entry in the users command list must have the following format:

+ 0  FCB   L  Where "L" is the size of the entry including this byte
+ 1  FCC  '<string>'  Where '<string>' is the command name
+ N  FDB  EP - * Where "EP" represents the symbol defining the start of the command routine

The first byte is an entry length byte and is always three more than the length of the command string (one for the length itself plus two for the routine offset). The command string must contain only ASCII alphanumeric characters, no special characters. An offset to the start of the command routine is used instead of an absolute address so that position-independent programs may contain command tables. The end of the command table is a one byte flag. A -1 ($FF) specifies that the secondary table is to be searched, or a -2 ($FE) that command list searching is to be terminated. The table represented as the secondary command list must end with -2. The first list must end with a -1 if both lists are to be searched, or a -2 if only one list is to be used.

A command routine is entered with the following registers set:

DPR - ASSIST09 page work area.
S - a return address to the command processor.
Z = 1 A carriage return terminated the command name.
Z = 0 A space delimiter followed the command name.
A command routine is entered after the delimiter following the command name is typed in. This means that a carriage return may be the delimiter entered with the input device resting on the next line. For this reason the Z bit in the condition code is set so the command routine may determine the current position of the input device. The command routine should ensure that the console device is left on a new line before returning to the command handler.

This entry points to the second list table. The default is a null list followed by a byte of −2. A complete explanation of the use for this entry is provided under the description of the .CMDL1 entry.

The responsibility of this handler is to send the character in the A register to the output device. The default routine also follows with padding characters as explained in the description of the OUTFCH service. If the output device is not ready to accept a character, then the "pause" subroutine should be called repeatedly while this condition lasts. The address of the pause routine is obtained from the .PAUSE entry in the vector table. The character counts for padding are obtained from the .PAD entry in the table. All ASSIST09 output is done with a call to this appendage. This includes punch processing as well. The default routine sends the character to an ACIA as explained in Paragraph B.2 Implementation Requirements. The operating environment is as follows:

**Input:**
- A = Character to send
- DP = ASSIST09 work page
- .PAD = Character and new line padding counts (in vector table)
- .PAUSE = Pause routine (in vector table)

**Output:**
- Character sent to the output device

**Volatile Registers:**
- None. All work registers must be restored
.COOFF  Output Console Shutdown Routine  .COOFF

Code: 30

Description: This entry addresses the routine to terminate output device processing. ASSIST09 does not call this routine. It is included for completeness. The default routine is an "RTS".

Input: DP → ASSIST09 work page
Output: The output device is terminated
Volatile Registers: None

.COON  Output Console Initialization Routine  .COON

Code: 26

Description: This entry points to a routine to initialize the standard output device. The default routine initializes an ACIA and is the very same one described under the .COON vector swap definition.

Input: .ACIA vector entry for the ACIA address
Output: The output device is initialized
Volatile Registers: A, X
.ECHO

Echo Flag

Code: 50

Description: The first byte of this word is used as a flag for the INCHP service routine to determine the requirement of echoing input received from the input handler. A non-zero value means to echo the input; zero not to echo. The echoing will take place even if user handlers are substituted for the default .CIDTA handler as the INCHP service routine performs the echo.

.FIRQ

Fast Interrupt Request Vector Appendage

Code: 10

Description: The fast interrupt request routine is located via this pointer. The MC6809 addresses hexadecimal FFF6 to locate the handler when processing a FIRQ. The stack and machine status is as defined for the FIRQ interrupt upon entry to this appendage. It should be noted that this routine is "jumped" to with an indirect jump instruction which adds eleven cycles to the interrupt time before the handler actually receives control. The default handler does an immediate "RTI" which, in essence, ignores the interrupt.
.HSDTA
High Speed Display Handler Routine

Code: 32

Description: This entry is invoked as a subroutine by the DISPLAY command and passed a parameter list containing the "TO" and "FROM" addresses. The from value is rounded down to a 16 byte address boundary. The default routine displays memory in both hexadecimal and ASCII representations, with a title produced on every 128 byte boundary. The purpose for this vector table entry is for easy implementation of a user routine for special purpose handling of a block of data. (The data could, for example, be sent to a high speed printer for later analysis.) The parameters are all passed on the stack. The environment is as follows:

  Input:  
  S + 4 = Start address
  S + 2 = Stop address
  S + 0 = Return Address
  DP → ASSIST09 work page

  Output: Any purpose desired

  Volatile Registers: X, D

.IRQ
Interrupt Request Vector Appendage

Code: 12

Description: All interrupt requests are passed to the routine pointed to by this vector. Hexadecimal FFF8 is the MC6809 location where this interrupt vector is fetched. The stack and processor status is that defined for the IRQ interrupt upon entry to the handler. Since the routine's address is in the vector table, an indirect jump must be done to invoke it. This adds eleven cycles to the interrupt time before the IRQ handler receives control. The default IRQ handler prints the registers and enters the ASSIST09 command handler.
**.NMI**

Non-Maskable Interrupt Vector Appendage

**Code:** 16

**Description:** This entry points to the non-maskable interrupt handler to receive control whenever the processor branches to the address at hexadecimal FFFC. Since ASSIST09 uses the NMI interrupt during trace and breakpoint processing, such commands should not be used if a user handler is in control. This is true unless the user handler has the intelligence to forward control to the default handler if the NMI interrupt has not been generated due to user facilities. The NMI handler given control will have an eleven cycle overhead as its address must be fetched from the vector table.

---

**.PAD**

Character and New Line Pad Count

**Code:** 48

**Description:** This entry contains the pad count for characters and new lines. The first of the two bytes is the count of nulls for other characters, and the second is the number of nulls ($00) to send out after any line feed is transmitted. The ASCII Escape character ($10) never has nulls sent following it. The default .CODTA handler is responsible for transmitting these nulls. A user handler may or may not use these counts as required.

The "NULLS" command also sets these two bytes with user specified values.
.PAUSE

Processing Pause Routine

Code: 40

Description: In order to support real-time (also known as multi-tasking) environments, ASSIST09 calls a dead-time routine whenever processing must wait for some external change of state. An example would be when the OUTCH service routine attempts the sending of a character to the ACIA through the default .CODTA handler and the ACIA status registers shows that it cannot yet be accepted. The default dead-time routine resides in a reserved four byte area which contains the single instruction, “RTS”. The .PAUSE vector entry points to this routine after standard initialization. This pointer may be changed to point to a user routine which dispatches other programs so that the MC6809 may be utilized more efficiently. Another example of use would be to increment a counter so that dead-time cycle counts may be accumulated for statistical or debugging purposes. The reason for the four byte reserved area (which exists in the ASSIST09 work page) is so other code may be overlaid without the need for another space in the address map to be assigned. For example, a master monitor may be using a memory management unit to assign a complete 64K block of memory to ASSIST09 and the programs being executed/tested under ASSIST09 control. The master monitor wishes, or course, to be reentered when any “dead time” occurs, so it overlays the default routine (“RTS”) with its own “SWI”. Since the master monitor would be “front ending” all “SWI’s” anyway, it knows when a “pause” call is being performed and can redispach other systems on a time-slice basis.

All registers must be transparent across the pause handler. Along with selected points in ASSIST09 user service processing, there is a special service call specifically for user programs to invoke the pause routine. It may be suggested that if no services are being requested for a given time period (say 10 ms) user programs should call the .PAUSE service routine so that fair-task dispatching can be guaranteed.

.PTM

Programmable Timer Module Address

Code: 53

Description: This entry contains the address of the MC6840 programmable timer module (PTM). Alteration of this slot should occur before the MONITR startup service is called as explained in Paragraph B.4 Initialization. If no PTM is available, then the address should be changed to a zero so that no initialization attempt will take place. Note that if a zero is supplied, ASSIST09 Breakpoint and Trace commands should not be issued.
**.RESET**  
Reset Interrupt Vector Appendage

**Code:** 18

**Description:** This entry returns the address of the RESET routine which initializes ASSIST09. Changing it has no effect, but it is included in the vector table in case a user program wishes to determine where the ASSIST09 restart code resides. For example, if ASSIST09 resides in the memory map such that it does not control the MC6809 hardware vectors, a user routine may wish to start it up and thus need to obtain the standard RESET vector code address. The ASSIST09 reset code assigns the default in the work page, calls the vector build subroutine, and then starts ASSIST09 proper with the MONITR service call.

**.RSVD**  
Reserved MC6809 Interrupt Vector Appendage

**Code:** 4

**Description:** This is a pointer to the reserved interrupt vector routine addressed at hexadecimal FFF0. This MC6809 hardware vector is not defined as yet. The default routine setup by ASSIST09 will cause a register display and entrance to the command handler.
.SWI

Software Interrupt Vector Appendage

Code: 14

Description: This vector entry contains the address of the Software Interrupt routine. Normally, ASSIST09 handles these interrupts to provide services for user programs. If a user handler is in place, however, these facilities cannot be used unless the user routine "passes on" such requests to the ASSIST09 default handler. This is easy to do, since the vector swap function passes back the address of the default handler when the switch is made by the user. This "front ending" allows a user routine to examine all service calls, or alter/replace/extend them to his requirements. Of course, the registers must be transparent across the transfer of control from the user to the standard handler. A "JMP" instruction branches directly to the routine pointed to by this vector entry when a SWI occurs. Therefore, the environment is that as defined for the "SWI" interrupt.

.SWI2

Software Interrupt 2 Vector Appendage

Code: 8

Description: This entry contains a pointer to the SWI2 handler entered whenever that instruction is executed. The status of the stack and machine are those defined for the SWI2 interrupt which has its interrupt vector address at FFF4 hexadecimal. The default handler prints the registers and enters the ASSIST09 command handler.
Code: 6

**Description:** This entry contains a pointer to the SWI3 handler entered whenever that instruction is executed. The status of the stack and machine are those defined for the SWI3 interrupt which has its interrupt vector address located at hexadecimal FFF2. The default handler prints the registers and enters the ASSIST09 command handler.
PLEASE NOTE:

I did not scan this ASSIST09 listing from the Motorola book. The listing was very large, and I was scanning a bound book which required each page to be scanned individually. Instead, I assembled the ASSIST09 source code using my own ASM09 assembler and placed the resultant listing into these pages. This has the added advantage that the text is searchable and can be extracted if you wish to use code snippets in other places. It does mean however that this listing is not precisely identical to the one originally printed in the Motorola MC6809-MC6809E Programming Reference manual.

Also note: I found the source code on which this listing is based via an internet search. It appears to be the original Motorola source code, however it had been modified both in function (code changes) and source format (different assembler). I have attempted to restore it as closely as possible to the original - please notify me if you find errors. For the most part, the source was compatible with my assembler. I did have to change the single-quote character constants to double-quote format ('a' instead of 'a'), and some of the directives are slightly different. (TITLE instead of TTL for example).

Dave Dunfield
* THIS IS THE BASE ASSIST09 ROM.
* IT MAY RUN WITH OR WITHOUT THE
* EXTENSION ROM WHICH
* WHEN PRESENT WILL BE AUTOMATICALLY
* INCORPORATED BY THE BLDVTR
* SUBROUTINE.

* GLOBAL MODULE EQUATES

$F800 ROMBEG EQU $F800 ROM START ASSEMBLY ADDRESS
$E700 RAMOFS EQU -$1900 ROM OFFSET TO RAM WORK PAGE
$0800 ROMSIZ EQU 2048 ROM SIZE
$F000 ROM2OF EQU ROMBEG-ROMSIZ START OF EXTENSION ROM
$E008 ACIA EQU $E008 DEFAULT ACIA ADDRESS
$E000 PTM EQU $E000 DEFAULT PTM ADDRESS
$0000 DFTCHP EQU 0 DEFAULT CHARACTER PAD COUNT
$0005 DFTNLP EQU 5 DEFAULT NEW LINE PAD COUNT
$003E PROMPT EQU '>' PROMPT CHARACTER
$0008 NUMBKP EQU 8 NUMBER OF BREAKPOINTS

* MISCELANEOUS EQUATES

$0004 EOT EQU $04 END OF TRANSMISSION
$0007 BELL EQU $07 BELL CHARACTER
$000A LF EQU $0A LINE FEED
$000D CR EQU $0D CARRIAGE RETURN
$0010 DLE EQU $10 DATA LINK ESCAPE
$0018 CAN EQU $18 CANCEL (CTL-X)
$0000 * PTM ACCESS DEFINITIONS

$E001 PTMSTA EQU PTM+1 READ STATUS REGISTER
$E000 PTMC13 EQU PTM CONTROL REGISTERS 1 AND 3
$E001 PTMC2 EQU PTM+1 CONTROL REGISTER 2
$E002 PTMTM1 EQU PTM+2 LATCH 1
$E004 PTMTM2 EQU PTM+4 LATCH 2
$E006 PTMTM3 EQU PTM+6 LATCH 3
$0000 * NEXT SUB-CODES FOR ACCESSING THE VECTOR TABLE.

$0004 INCHNP EQU 0 INPUT CHAR IN A REG - NO PARITY
$0005 OUTCH EQU 1 OUTPUT CHAR FROM A REG
$0005 PDATA1 EQU 2 OUTPUT STRING
$0003 PDATA EQU 3 OUTPUT CR/LF THEN STRING
$0004 OUT2HS EQU 4 OUTPUT TWO HEX AND SPACE
$0005 OUT4HS EQU 5 OUTPUT FOUR HEX AND SPACE
$0006 PCRLF EQU 6 OUTPUT CR/LF
$0007 SPACE EQU 7 OUTPUT A SPACE
$0008 MONITR EQU 8 ENTER ASSIST09 MONITOR
$0009 VCTRST EQU 9 VECTOR EXAMINE/SWITCH
$000A BRKPT EQU 10 USER PROGRAM BREAKPOINT
$000B PAUSE EQU 11 TASK PAUSE FUNCTION
$000B NUMFUN EQU 11 NUMBER OF AVAILABLE FUNCTIONS

* RELATIVE POSITIONING MUST BE MAINTAINED.
0000 69 .AVTBL EQU 0 ADDRESS OF VECTOR TABLE
0002 70 .CMDL1 EQU 2 FIRST COMMAND LIST
0004 71 .RSVD EQU 4 RESERVED HARDWARE VECTOR
0006 72 .SWI3 EQU 6 SWI3 ROUTINE
0008 73 .SWI2 EQU 8 SWI2 ROUTINE
000A 74 .FIRQ EQU 10 FIRQ ROUTINE
000C 75 .IRQ EQU 12 IRQ ROUTINE
000E 76 .SWI1 EQU 14 SWI ROUTINE
0010 77 .NMI EQU 16 NMI ROUTINE
0012 78 .RESET EQU 18 RESET ROUTINE
0014 79 .CICN EQU 20 CONSOLE ON
0016 80 .CIDTA EQU 22 CONSOLE INPUT DATA
0018 81 .CIOFF EQU 24 CONSOLE INPUT OFF
001A 82 .COON EQU 26 CONSOLE OUTPUT ON
001C 83 .CODTA EQU 28 CONSOLE OUTPUT DATA
001E 84 .COOFF EQU 30 CONSOLE OUTPUT OFF
0020 85 .HSRTA EQU 32 HIGH SPEED PRINTDATA
0022 86 .BCON EQU 34 PUNCH/LOAD ON
0024 87 .BSDTA EQU 36 PUNCH/LOAD DATA
0026 88 .BSOFF EQU 38 PUNCH/LOAD OFF
0028 89 .PAUSE EQU 40 TASK PAUSE ROUTINE
002A 90 .EXPAN EQU 42 EXPRESSION ANALYZER
002C 91 .CMDL2 EQU 44 SECOND COMMAND LIST
002E 92 .ACIA EQU 46 ACIA ADDRESS
0030 93 .PAD EQU 48 CHARACTER PAD AND NEW LINE PAD
0032 94 .RECHO EQU 50 ECHO/LOAD AND NULL BKPT FLAG
0034 95 .PTM EQU 52 PTM ADDRESS
0036 96 .NUMVTR EQU 52/2+1 NUMBER OF VECTORS
0038 97 .HIVTR EQU 52 HIGHEST VECTOR OFFSET
0000 0000  99  ******************************************
0000 0000 100  * WORK AREA
0000 0000 101  * THIS WORK AREA IS ASSIGNED TO THE PAGE ADDRESSED BY
0000 0000 102  * -$1800,PCR FROM THE BASE ADDRESS OF THE ASSIST09
0000 0000 103  * ROM. THE DIRECT PAGE REGISTER DURING MOST ROUTINE
0000 0000 104  * OPERATIONS WILL POINT TO THIS WORK AREA. THE STACK
0000 0000 105  * INITIALLY STARTS UNDER THE RESERVED WORK AREAS AS
0000 0000 106  * DEFINED HEREIN.
0000 0000 107  ******************************************
0000 0000 108  WORKPG EQU ROMBEG+RAMOF'S SETUP DIRECT PAGE ADDRESS
0000 0000 109  SETDP =WORKPG NOTIFY ASSEMBLER
0000 0000 110  ORG WORKPG+256 READY PAGE DEFINITIONS
0000 0000 111  * THE FOLLOWING THRU BKPTOP MUST RESIDE IN THIS ORDER
0000 0000 112  * FOR PROPER INITIALIZATION
0000 0000 113  PAUSER EQU * PAUSE ROUTINE
0000 0000 114  ORG *-4
0000 0000 115  SWIBFL EQU * BYPASS SWI AS BREAKPOINT FLAG
0000 0000 116  ORG *-1
0000 0000 117  ORG *-1
0000 0000 118  BKPTCT EQU * BREAKPOINT COUNT
0000 0000 119  ORG *-2
0000 0000 120  SLEVEL EQU * STACK TRACE LEVEL
0000 0000 121  ORG *(NUMVTR*2)
0000 0000 122  VECTAB EQU * VECTOR TABLE
0000 0000 123  ORG *(NUMBKP)
0000 0000 124  BKPTBL EQU * BREAKPOINT TABLE
0000 0000 125  ORG *(2*NUMBKP)
0000 0000 126  BKPTOP EQU * BREAKPOINT OPCODE TABLE
0000 0000 127  ORG *-2
0000 0000 128  WINDOW EQU * WINDOW
0000 0000 129  ORG *-2
0000 0000 130  ADDR EQU * ADDRESS POINTER VALUE
0000 0000 131  ORG *-1
0000 0000 132  BASEPG EQU * BASE PAGE VALUE
0000 0000 133  ORG *-2
0000 0000 134  NUMBER EQU * BINARY BUILD AREA
0000 0000 135  ORG *-2
0000 0000 136  LASTOP EQU * LAST OPCODE TRACED
0000 0000 137  ORG *-2
0000 0000 138  RSTACK EQU * RESET STACK POINTER
0000 0000 139  ORG *-2
0000 0000 140  PSTACK EQU * COMMAND RECOVERY STACK
0000 0000 141  ORG *-2
0000 0000 142  PCNTER EQU * LAST PROGRAM COUNTER
0000 0000 143  ORG *-2
0000 0000 144  TRACEC EQU * TRACE COUNT
0000 0000 145  ORG *-1
0000 0000 146  SWICNT EQU * TRACE "SWI" NEST LEVEL COUNT
0000 0000 147  ORG *-1
0000 0000 148  MISCNT EQU * (MISFLG MUST FOLLOW SWICNT)
0000 0000 149  ORG *-1
0000 0000 150  DELIM EQU * EXPRESSION DELIMITER/WORK BYTE
0000 0000 151  ORG *-40
0000 0000 152  ROM2WK EQU * EXTENSION ROM RESERVED AREA
0000 0000 153  ORG *-21
0000 0000 154  TSTACK EQU * TEMPORARY STACK HOLD
0000 0000 155  STACK EQU * START OF INITIAL STACK
* INITIAL STATE OF THE VECTOR TABLE. ALL ADDRESSES ARE CONVERTED TO ABSOLUTE FORM. THIS TABLE STARTS WITH THE SECOND ENTRY, ENDS WITH STATIC CONSTANT INITIALIZATION DATA WHICH CARRIES BEYOND THE TABLE.

**INITVT** FDB CMDTBL-* DEFAULT FIRST COMMAND TABLE
**RSRVD** FDB RSRVD-* DEFAULT UNDEFINED HARDWARE VECTOR
**SWI3R** FDB SWI3R-* DEFAULT SWI3
**SWI2R** FDB SWI2R-* DEFAULT SWI2
**FIRQR** FDB FIRQR-* DEFAULT FIRQ
**IRQR** FDB IRQR-* DEFAULT IRQ ROUTINE
**SWIR** FDB SWIR-* DEFAULT SWI ROUTINE
**NMIR** FDB NMIR-* DEFAULT NMI ROUTINE
**FIR** FDB FIP-* DEFAULT FUNCTION INTR
**RST** FDB RST-* DEFAULT RESTART ROUTINE
**CION** FDB CION-* DEFAULT CION
**CIDTA** FDB CIDTA-* DEFAULT CIDTA
**CIOFF** FDB CIOFF-* DEFAULT CIOFF
**SWI** FDB SWIR-* DEFAULT SWI ROUTINE
**NMI** FDB NMIR-* DEFAULT NMI ROUTINE
**BREAK** FDB BKPNT-* DEFAULT BREAKPOINT ROUTINE
**EX** FDB EXP1-* DEFAULT EXPRESSION ANALYZER
**CMDTB2** FDB CMDTB2-* DEFAULT SECOND COMMAND TABLE

**INTVS** FDB ACIA DEFAULT ACIA
**DFTP** FDB DFTCHP,DFTNLF DEFAULT NULL PADDs
**ECHON** FDB 0 DEFAULT ECHO
**FTM** FDB FTM DEFAULT FTM
**STL** FDB 0 INIT STACK TRACE LEVEL
**BC** FDB 0 INIT BREAKPOINT COUNT
**SWI** FDB 0 SWI BREAKPOINT LEVEL
**SPACE** FDB $39 DEFAULT PAUSE ROUTINE (RTS)

**INCHNP** FDB INCHNP
**OUTCH** FDB OUTCH
**PDATA1** FDB PDATA1
**PDATA** FDB PDATA
**OUT2HS** FDB OUT2HS
**OUT4HS** FDB OUT4HS
**PCRLF** FDB PCRLF
**PCLF** FDB PCLF
**ZSPACE** FDB ZSPACE
**ZMONTR** FDB ZMONTR
**ZVSMTH** FDB ZVSMTH
**ZBKPFN** FDB ZBKPFN
**ZPAUSE** FDB ZPAUSE

**ASSIST09 SWI HANDLER**

**FOR A USER PROGRAM. A FUNCTION BYTE IS ASSUMED TO FOLLOW THE SWI INSTRUCTION. IT IS BOUND CHECKED AND THE PROPER ROUTINE IS GIVEN CONTROL. THIS INVOCATION MAY ALSO BE A BREAKPOINT INTERRUPT. IF SO, THE BREAKPOINT HANDLER IS ENTERED.**

**OUTPUT: VARIES ACCORDING TO FUNCTION CALLED. PC ON CALLERS STACK INCREMENTED BY ONE IF VALID CALL. VOLATILE REGISTERS: SEE FUNCTIONS CALLED**

**RUNS DISABLED UNLESS FUNCTION CLEAR S I FLAG.**

**SWI FUNCTION VECTOR TABLE**

**INCHNP** FDB INCHNP
**OUTCH** FDB OUTCH
**PDATA1** FDB PDATA1
**PDATA** FDB PDATA
**OUT2HS** FDB OUT2HS
**OUT4HS** FDB OUT4HS
**PCRLF** FDB PCRLF
**PCLF** FDB PCLF
**ZSPACE** FDB ZSPACE
**ZMONTR** FDB ZMONTR
**ZVSMTH** FDB ZVSMTH
**ZBKPFN** FDB ZBKPFN
**ZPAUSE** FDB ZPAUSE
DUNFIELD 6809 ASSEMBLER: ASSIST09

F895  6A 8D E6 F7  293 SWIR  DEC  SWICNT,PCR  UP "SWI" LEVEL FOR TRAC
F899  17 02 25  294 LBSR  LDOP  SETUP PAGE AND VERIFY STACK
F89C  295 * CHECK FOR BREAKPOINT TRAP
F89C  EE 6A  296 LDU  10,8  LOAD PROGRAM COUNTER
F89E  33 5F  297 LEAU  -1,U  BACK TO SWI ADDRESS
F8A0  0D FB  298 TST  SWIBFL  ? THIS "SWI" BREAKPOINT
F8A2  26 11  299 BNE  SWIDNE  BRANCH IF SO TO LET THROUGH
F8A4  17 06 9B  300 LBSR  CBKDR  OBTAIN BREAKPOINT POINTERS
F8A8  50  301 NEGB  OBTAIN POSITIVE COUNT
F8A9  5A  302 SWILP  DECB  COUNT DOWN
F8A9  2B 0A  303 BMI  SWIDNE  BRANCH WHEN DONE
F8AB  11 A3 A1  304 CMPU  ,Y++  ? WAS THIS A BREAKPOINT
F8AD  33 8C B8  305 LEAU  SWITB,PCR  OBTAIN VECTOR BRANCH ADDRESS
F8AF  EC C5  306 JMP  D,U  JUMP TO ROUTINE
F8B0  CC 01 A6  307 LDD  #$01A6  SETUP TIMER 1 MODE
F8B2  A7 01  308 STA  PTMC2-PTM,X  SET OUTPUT ENABLED/
F8B4  6F 01  309 CLR  PTMC2-PTM,X  SET CR2 BACK TO RESET FORM
F8B5  41 53 53 49 53 54 +  310 SIGNON  FCC  /ASSIST09/  SIGNON EYE-CATCHER
F8D1  04  311 FCB  EOT
F8D2  10 97  312 FCB  EO
F8D2  5D 61  313 TST  1,S  ? INIT CONSOLE AND SEND MSG
F8D5  26 0D  314 BNE  ZMONT2  BRANCH IF NOT TO USE A PTM
F8D7  AD 9D E6 F9  315 JSR  [VECTAB+.CION,PCR]  READY CONSOLE INPUT
F8D8  AD 9D E6 FB  316 JSR  [VECTAB+.COON,PCR]  READY CONSOLE OUTPUT
F8E1  30 8C E5  317 LEAX  SIGNON,PCR  READY SIGNON EYE-CATCHER
F8E4  3F  318 SWI  PERFORM
F8E5  03  319 FCB  POATA  PRINT STRING
F8E6  9E F6  320 LDX  VECTAB+.PTM  LOAD PTM ADDRESS
F8E8  27 0D  321 BEQ  CMD  BRANCH IF NOT TO USE A PTM
F8E8  6F 02  322 CLR  PTMTM1-PTM,X  SET LATCH TO CLEAR RESET
F8EC  6F 03  323 CLR  PTMTM1+1-PTM,X  AND SET GATE HIGH
F8EE  CC 01 A6  324 LDD  #$501A6  SETUP TIMER 1 MODE
F8F1  A7 01  325 STA  PTMC2-PTM,X  SETUP FOR CONTROL REGISTER1
F8F3  E7 00  326 STB  PTMC13-PTM,X  SET OUTPUT ENABLED/
F8FB  57  327 * SINGLE SHOT/ DUAL 8 BIT/INTERNAL MODE/OPE
F8F5  6F 01  328 CLR  PTMC2-PTM,X  SET CR2 BACK TO RESET FORM
F8F7  359  329 FALL INTO COMMAND PROCESSOR
F8F7 3F 377  CMD SWI  TO NEW LINE
F8F8 06 378  PSCH PCLR FUNCTION
F8F9 17 06 46 380  CMDNEP LBSR CBKLDI OBTAIN BREAKPOINT POINTERS
F8FC 20 0C 381  BPL CMDNOL BRANCH IF NOT ARMED OR NONE
F8FE 50 382  NEGB MAKE POSITIVE
F8FF 27 5A 383  STB BKPTCT FLAG AS INVALID ARMED?
F901 5A 384  CMDDDL DECUB ? FINISHED
F902 2B 06 385  BMI CMDNOL BRANCH IF SO
F904 A6 30 386  LDA -NUMBR*2,Y LOAD OPCODE STORED
F906 A7 B1 387  STA (Y+Y) STORE BACK OVER "SWI"
F908 80 04 388  BRA CMDNOL LOOP UNTIL DONE
F90A AE 6A 389  CMDNOL LDX 10,S LOAD USERS PROGRAM COUNTER
F90C 9E 93 390  STX PCNTER SAVE FOR EXPRESSION ANALYZER
F90E 86 3E 391  LDA #PROMPT LOAD PROMPT CHARACTER
F910 3F 392  SWI SEND TO OUTPUT HANDLER
F911 01 393  FCB OUTCH FUNCTION
F912 33 E4 394  LEAU ,S REMEMBER STACK RESTORE ADDRESS
F914 DF 95 395  STU PSTACK REMEMBER STACK FOR ERROR USE
F916 4F 396  CLRA CLEAR ZERO
F917 5F 397  CLRUB PREPARE ZERO
F919 DD 9B 398  STD NUMBER CLEAR NUMBER BUILD AREA
F91A DD 8F 399  STD MISFLG CLEAR MISCEL. AND SWICNT FLAGS
F91C DD 91 400  STD TRACCE CLEAR TRACE COUNT
F91E C6 02 401  LDB #2 SET D TO TWO
F920 34 07 402  STU D.CC PLACE DEFUALTS ONTO STACK
F922 17 04 54 404  LBSR READ OBTAIN FIRST CHARACTER
F925 30 8D 05 81 405  LEAX CDOT+2,PCR PRESET FOR SINGLE TRACE
F929 81 2E 406  CMPA [#]'? QUICK TRACE
F92B 27 4F 407  BEQ CMDXQT BRANCH EQUAL FOR TRACE ONE
F92D 30 8D 04 E9 408  LEAX CMXPAD+2,PCR READY MEMORY ENTRY POINT
F931 81 2F 409  CMPA [#]'? OPEN LAST USED MEMORY
F933 32 02 410  BEQ CMDXQT BRANCH TO DO IT IF SO
F935 81 20 412  CMPA CMMD #? BLANK OR DELIMITER
F937 23 14 413  BLS CMDGOT BRANCH YES, WE HAVE IT
F939 34 02 414  PSHS A BUILD ONTO STACK
F93B 6C 5F 415  INC -1,U COUNT THIS CHARACTER
F93D 81 2F 416  CMPA [#]'? MEMORY COMMAND
F93F 27 4F 417  BEQ CMDMEM BRANCH IF SO
F941 17 04 0B 418  LBSR BLTDC MCTOR TREAT AS HEX VALUE
F944 27 02 419  BEQ CMD3 BRANCH IF STILL VALID NUMBER
F946 6A 5E 420  DEC -2,U FLG AS INVALID NUMBER
F948 17 04 2E 421  CMD3 LBSR READ OBTAIN NEXT CHARACTER
F94B 20 E8 422  BRA CMD2 TEST NEXT CHARACTER
F94D 80 0D 424  CMDGOT SUBRA #CR SET ZERO IF CARRIAGE RETURN
F94F A7 5D 425  STA -3,U SETUP FLAG
F951 9E C4 426  LDX VECTAB+.CMDL1 START WITH FIRST CMD LIST
F953 06 8D 427  CMSCH LDB ,X+ LOAD ENTRY LENGTH
F955 2A 10 428  BPL CMDSME BRANCH IF NOT LIST END
F957 9E EE 429  LDS  VECTOR+.CMDL2  NOW TO SECOND CMD LIST
F959 5C 430  INCB  ? TO CONTINUE TO DEFAULT LIST
F95A 27 F7 431  CMDSCH  BRANCH IF 50
F95C 10 DE 95 432  CMDBAD  LDS  PSTACK  RESTORE STACK
F95F 30 8D 01 5A 433  LEAX  ERRMSG,PCR  POINT TO ERROR STRING
F963 3F 434  SWI  SEND OUT
F964 02 435  FCB  PDATA1  TO CONSOLE
F965 20 90 436  BRA  CMD  AND TRY AGAIN
F967 5A 437  * SEARCH NEXT ENTRY
F968 4A 5F 439  CMPB  -1,U  ? ENTERED LONGER THAN ENTRY
F96A 24 03 440  BHS  CMDSIZ  BRANCH IF NOT TOO LONG
F96C 3A 441  CMDFLS  ABX  SKIP  TO NEXT ENTRY
F96D 20 E4 442  BRA  CMD  AND TRY NEXT
F96F 31 5D 443  CMDSIZ  LEAY  -3,U  PREPARE TO COMPARE
F971 A6 5F 444  LDA  -1,U  LOAD SIZE+2
F973 80 02 445  SUBA  #2  TO ACTUAL SIZE ENTERED
F975 A7 5E 446  STA  -2,U  SAVE SIZE FOR COUNTDOWN
F977 5A 447  CMDCMP  DECB  DOWN ONE BYTE
F978 A6 80 448  LDA  ,X+  NEXT COMMAND CHARACTER
F97A 4A 6E 449  CMPA  ,-Y  ? SAME AS THAT ENTERED
F97C 26 EE 450  BNE  CMDFLS  BRANCH TO FLUSH IF NOT
F97E 5A 5F 451  BHS  CMDSIZ  BRANCH IF NOT TOO LONG
F980 2A 5E 452  BRA  CMD  AND TRY NEXT
F982 3A 453  ABX  TO  NEXT ENTRY
F983 4E 454  LDD  -2,X  LOAD OFFSET
F985 30 8B 455  LEAX  D,X  COMPUTE ROUTINE ADDRESS+2
F987 6D 5D 456  CMXQT  TST  -3,U  SET CC FOR CARRIAGE RETURN TEST
F989 32 C4 457  LEAS  ,U  DELETE STACK WORK AREA
F98B AD 1E 458  JSR  -2,X  CALL COMMAND
F98D 16 FF 7A 459  LBRA  CMDNOL  GO GET NEXT COMMAND
F990 6D 5E 460  CMDEM  TST  -2,U  ? VALID HEX NUMBER ENTERED
F992 2B C8 461  BMI  CMDBAD  BRANCH ERROR IF NOT
F994 30 88 AE 462  LEAX  <CMEMN-CMPADP,X TO DIFFERENT ENTRY
F997 DC 9B 463  LDD  NUMBER  LOAD NUMBER ENTERED
F999 20 EC 464  BRA  CMDXQT  AND ENTER MEMORY COMMAND
F99B 465  ** COMMANDS ARE ENTERED AS A SUBROUTINE WITH:
F99B 466  ** DPR->ASSIST09 DIRECT PAGE WORK AREA
F99B 467  ** Z=1 CARRIAGE RETURN ENTERED
F99B 468  ** Z=0 NON CARRIAGE RETURN DELIMITER
F99B 469  ** S=NORMAL RETURN ADDRESS
F99B 470  ** THE LABEL "CMDBAD" MAY BE ENTERED TO ISSUE AN
F99B 471  ** AN ERROR FLAG (*).
F99B 472  ** THE LABEL "CMDBAD" MAY BE ENTERED TO ISSUE AN
F99B 473  ** AN ERROR FLAG (*).
F99B 474  ************************************************************
F99B 475  * ASSIST09 COMMAND TABLES
F99B 476  * THESE ARE THE DEFAULT COMMAND TABLES.  EXTERNAL
F99B 477  * TABLES OF THE SAME FORMAT MAY EXTEND/REPLACE
F99B 478  * THESE BY USING THE VECTOR SWAP FUNCTION.
F99B 479  *
F99B 480  * ENTRY FORMAT:
F99B 481  * +0...TOTAL SIZE OF ENTRY (INCLUDING THIS BYTE)
F99B 482  * +1...COMMAND STRING
F99B 483  * +N...TWO BYTE OFFSET TO COMMAND (ENTRYADDR-*)
F99B 484  *
F99B 485  * THE TABLES TERMINATE WITH A ONE BYTE -1 OR -2.
F99B 486  * THE -1 CONTINUES THE COMMAND SEARCH WITH THE
F99B 487  * SECOND COMMAND TABLE.
F99B 488  * THE -2 TERMINATES COMMAND SEARCHES.
F99B 489  ************************************************************
F99B 490  *
F99B 491  * THIS IS THE DEFAULT LIST FOR THE SECOND COMMAND
F99B 492  * LIST ENTRY.
F99B 493  CMDTB2  FCB  -2  STOP COMMAND SEARCHES
F99C 494  *
F99C 495  * THIS IS THE DEFAULT LIST FOR THE FIRST COMMAND
F99C 496  * LIST ENTRY.
F99C  497  CMDTBL  EQU *                     MONITOR COMMAND TABLE
F99C  04  498  FCB  4                     'BREAKPOINT' COMMAND
F99D  42  499  FCC  /B/                        'CALL' COMMAND
F99E  05  500  FDB  CBKPT-*                    'DISPLAY' COMMAND
F9A0  04  501  FCB  4                     'ENCOD' COMMAND
F9A1  43  502  FCC  /C/                         'LOAD' COMMAND
F9A2  05  503  FDB  CCALL-*                    'DISPLAY' COMMAND
F9A4  04  504  FCB  4                      'LOAD' COMMAND
F9A5  44  505  FCC  /D/                         'LOAD' COMMAND
F9A6  04  506  FDB  CDISP-*                    'DISPLAY' COMMAND
F9A8  04  507  FCB  4                      'LOAD' COMMAND
F9A9  45  508  FCC  /E/                          'LOAD' COMMAND
F9AA  05  509  FDB  CENCDE-*                  'DISPLAY' COMMAND
F9AC  04  510  FCB  4                       'LOAD' COMMAND
F9AD  47  511  FCC  /G/                           'LOAD' COMMAND
F9AE  03  512  FDB  CGO-*                     'LOAD' COMMAND
F9B0  04  513  FCB  4                      'LOAD' COMMAND
F9B1  4C  514  FCC  /H/                             'LOAD' COMMAND
F9B2  04  515  FDB  LOAD-*                     'LOAD' COMMAND
F9B4  04  516  FCB  4                       'LOAD' COMMAND
F9B5  4D  517  FCC  /M/                                  'LOAD' COMMAND
F9B6  04  518  FDB  CHEM-*                   'LOAD' COMMAND
F9B8  04  519  FCB  4                      'LOAD' COMMAND
F9B9  4E  520  FCC  /N/                                    'LOAD' COMMAND
F9BA  04  521  FDB  CNULLS-*                  'LOAD' COMMAND
F9BC  04  522  FCB  4                      'LOAD' COMMAND
F9BD  4F  523  FCC  /O/                                  'LOAD' COMMAND
F9BE  05  524  FDB  COFFS-*                   'LOAD' COMMAND
F9C0  04  525  FCB  4                      'LOAD' COMMAND
F9C1  50  526  FCC  /P/                                'LOAD' COMMAND
F9C2  04  527  FDB  CPUNCH-*                   'LOAD' COMMAND
F9C4  04  528  FCB  4                      'LOAD' COMMAND
F9C5  52  529  FCC  /R/                             'LOAD' COMMAND
F9C6  02  530  FDB  CREG-*                      'LOAD' COMMAND
F9C8  04  531  FCB  4                      'LOAD' COMMAND
F9C9  53  532  FCC  /S/                                   'LOAD' COMMAND
F9CA  04  533  FDB  CSTLEV-*                   'LOAD' COMMAND
F9CC  04  534  FCB  4                      'LOAD' COMMAND
F9CD  54  535  FCC  /T/                                    'LOAD' COMMAND
F9CE  04  536  FDB  CTTRACE-*                  'LOAD' COMMAND
F9DD  04  537  FCB  4                      'LOAD' COMMAND
F9DE  56  538  FCC  /U/                                   'LOAD' COMMAND
F9DF  04  539  FDB  CVER-*                     'LOAD' COMMAND
F9D4  04  540  FCB  4                      'LOAD' COMMAND
F9D5  57  541  FCC  /W/                                   'LOAD' COMMAND
F9D6  04  542  FDB  CWINDO-*                    'LOAD' COMMAND
F9DF  FF  543  FCB  -1                                  END, CONTINUE WITH THE SECOND
F9D9  45  544  **************************************************

F9DB  34  06  555  PSHS  D                                 SAVE - DO NOT REREAD
F9DC  C6  10  556  LDB  #16                                SHIFT BY 4 BITS
F9DD  3D  557  MUL                              WITH MULTIPLY
F9DE  8D  04  558  BSR  ZOUTHX                             SEND OUT AS HEX
F9E2  35  06  559  PULS  D                                RESTORE BYTES
F9E4  84  0F  560  ANDA  #$5F                             ISOLATE RIGHT HEX
F9E6  BB  90  561  ZOUTHX  ADDA  #$90                     PREPARE A-P ADJUST
F9EB  19  562  DAA  ADJUST                               PREPARE CHARACTER BITS
F9EB  19  563  ADCA  #$40                             PREPARE CHARACTER BITS
F9E9  89  40  564  DAA  ADJUST                             PREPARE CHARACTER BITS
F9EC 6E 9D EE  EE  565 SEND  JMP [VECTAB+.CODTA,PCR] SEND TO OUT HANDLER
F9F0  566
F9F0  8D E7  567 ZOT4HS BSR ZOUT2H CONVERT FIRST BYTE
F9F2  8D E5  568 ZOT2HS BSR ZOUT2H CONVERT BYTE TO HEX
F9F4  AF 64  569 STX 4,S UPDATE USERS X REGISTER
F9F6  570 * FALL INTO SPACE ROUTINE
F9F6  571
F9F6  572 *************************************************
F9F6  573 * [SWI FUNCTION 7]
F9F6  574 * SPACE - SEND BLANK TO OUTPUT HANDLER
F9F6  575 * INPUT: NONE
F9F6  576 * OUTPUT: BLANK SEND TO CONSOLE HANDLER
F9F6  577 *************************************************
F9F6  86 20  578 ZSPACE LDA #' ' LOAD BLANK
F9F8  20 3D  579 BRA ZOTCH2 SEND AND RETURN
F9FA  580
F9FA  581 *************************************************
F9FA  582 * [SWI FUNCTION 9]
F9FA  583 * SWAP VECTOR TABLE ENTRY
F9FA  584 * INPUT: A=VECTOR TABLE CODE (OFFSET)
F9FA  585 * X=0 OR REPLACEMENT VALUE
F9FA  586 * OUTPUT: X=PREVIOUS VALUE
F9FA  587 *************************************************
F9FA  A6 61  588 ZVSWH LDA 1,S LOAD REQUESTERS A
F9FC  81 34  589 CMPA #IVTR ? SUB-CODE TOO HIGH
F9FE  22 39  590 BHI ZOTCH3 IGNORE CALL IF SO
FA00  10 9E C2  591 LDY VECTAB+.AVTBL LOAD VECTOR TABLE ADDRESS
FA02  83 66  592 LDU A,Y U=OLD ENTRY
FA04  EF 64  593 STU 4,S RETURN OLD VALUE TO CALLERS X
FA05  EF 64  593 STU 4,S RETURN OLD VALUE TO CALLERS X
FA06  AF A6  594 STX -2,S ? X=0
FA07  AF 7E  595 BEQ ZOTCH3 YES, DO NOT CHANGE ENTRY
FA09  27 E2  596 STX A,Y REPLACE ENTRY
FA0A  AA 2A  597 BRA ZOTCH3 RETURN FROM SWI
FA0F  598 *D
FA0F  599
FA0F  600 *************************************************
FA0F  601 * [SWI FUNCTION 0]
FA0F  602 * INCHNP - OBTAIN INPUT CHAR IN A (NO PARITY)
FA0F  603 * NULLS AND RUBOUTS ARE IGNORED.
FA0F  604 * AUTOMATIC LINE FEED IS SENT UPON RECEIVING A
FA0F  605 * CARRIAGE RETURN.
FA0F  606 * UNLESS WE ARE LOADING FROM TAPE.
FA0F  607 *************************************************
FA0F  8D 5D  608 ZINCHP BSR XQPAUS RELEASE PROCESSOR
FA11  8D 5F  609 ZINCH BSR XQCIDT CALL INPUT DATA APPENDAGE
FA13  24 FA  610 BCC ZINCHP LOOP IF NONE AVAILABLE
FA15  4D  611 STA ? TEST FOR NULL
FA16  27 F9  612 BEQ ZINCH IGNORE NULL
FA18  81 7F  613 CMPA #$7F ? RUBOUT
FA1A  27 F5  614 BEQ ZINCH BRANCH YES TO IGNORE
FA1C  A7 61  615 STA 1,S STORE INTO CALLERS A
FA1E  DD 8F  616 TST MISFLG ? LOAD IN PROGRESS
FA20  26 17  617 BNE ZOT3 CH BRANCH IF SO TO NOT ECHO
FA22  81 OD  618 CMPA #CR ? CARRIAGE RETURN
FA24  26 04  619 BNE ZIN2 NO, TEST ECHO BYTE
FA26  86 0A  620 LDA #LF LOAD LINE FEED
FA28  8D C2  621 BSR SEND ALWAYS ECHO LINE FEED
FA2A  DD F4  622 ZIN2 TST VECTAB+.ECHO ? ECHO DESIRED
FA2C  26 0B  623 BNE ZOTCH3 NO, RETURN
FA2E  624 * FALL THROUGH TO OUTCH
FA2E  625
FA2E  626 *************************************************
FA2E  627 * [SWI FUNCTION 1]
FA2E  628 * OUTCH - OUTPUT CHARACTER FROM A
FA2E  629 * INPUT: NONE
FA2E  630 * OUTPUT: IF LINEFEED IS THE OUTPUT CHARACTER THEN
FA2E  631 * C=0 NO CTL-X RECEIVED, C=1 CTL-X RECEIVED
FA2E  632
FA2E  A6 61  633  ZOTCH1  LDA  1,S  LOAD CHARACTER TO SEND
FA30  30 8C 09  634  LEAX  <2PRLS,PCR  DEFAULT FOR LINE FEED
FA33  81 0A  635  CMPA  #LF  ?  LINE FEED
FA35  27 0F  636  BEQ  2PDTLP  BRANCH TO CHECK PAUSE IF SO
FA37  8D B3  637  ZOTCH2  BSR  SEND  SEND TO OUTPUT ROUTINE
FA39  0C 90  638  ZOTCH3  INC  SWICNT  BUMP UP "SWI" TRACE NEST LEVEL
FA3B  3B  639  RTI  RETURN  FROM "SWI" FUNCTION
FA3C  04  640
FA3C  04  641  **************************************************
FA3C  04  642  *              [SWI FUNCTION 6]
FA3C  04  643  *        PCRLF - SEND CR/LF TO CONSOLE HANDLER
FA3C  04  644  *  INPUT: NONE
FA3C  04  645  *  OUTPUT: CR AND LF SENT TO HANDLER
FA3C  04  646  *  C=0 NO CTL-X, C=1 CTL-X RECEIVED
FA3C  04  647  **************************************************
FA3C  04  648
FA3D  30 8C FC  649  ZPCRLF  LEAX  ZPCRLS,PCR  READY CR,LF STRING
FA3D  30 8C FC  650  * FALL INTO CR/LF CODE
FA40  86 0D  651  ZPDATA  LDA  #CR  LOAD CARRIAGE RETURN
FA42  8D A8  652  BSR  SEND  SEND IT
FA44  86 0A  653  LDA  #LF  LOAD LINE FEED
FA46  86 0A  654  LDA  #LF  LOAD LINE FEED
FA46  86 0A  655  LDA  #LF  LOAD LINE FEED
FA46  86 0A  656  LDA  #LF  LOAD LINE FEED
FA46  86 0A  657  LDA  #LF  LOAD LINE FEED
FA46  86 0A  658  LDA  #LF  LOAD LINE FEED
FA46  86 0A  659  LDA  #LF  LOAD LINE FEED
FA46  86 0A  660  LDA  #LF  LOAD LINE FEED
FA46  86 0A  661  LDA  #LF  LOAD LINE FEED
FA46  86 0A  662  LDA  #LF  LOAD LINE FEED
FA46  86 0A  663  LDA  #LF  LOAD LINE FEED
FA46  86 0A  664  LDA  #LF  LOAD LINE FEED
FA46  86 0A  665  LDA  #LF  LOAD LINE FEED
FA46  86 0A  666  LDA  #LF  LOAD LINE FEED
FA46  86 0A  667  LDA  #LF  LOAD LINE FEED
FA46  86 0A  668  LDA  #LF  LOAD LINE FEED
FA46  86 0A  669  LDA  #LF  LOAD LINE FEED
FA46  86 0A  670  LDA  #LF  LOAD LINE FEED
FA46  86 0A  671  LDA  #LF  LOAD LINE FEED
FA46  86 0A  672  LDA  #LF  LOAD LINE FEED
FA46  86 0A  673  LDA  #LF  LOAD LINE FEED
FA46  86 0A  674  LDA  #LF  LOAD LINE FEED
FA46  86 0A  675  LDA  #LF  LOAD LINE FEED
FA46  86 0A  676  LDA  #LF  LOAD LINE FEED
FA46  86 0A  677  LDA  #LF  LOAD LINE FEED
FA46  86 0A  678  LDA  #LF  LOAD LINE FEED
FA46  86 0A  679  LDA  #LF  LOAD LINE FEED
FA46  86 0A  680  LDA  #LF  LOAD LINE FEED
FA46  86 0A  681  LDA  #LF  LOAD LINE FEED
FA46  86 0A  682  LDA  #LF  LOAD LINE FEED
FA46  86 0A  683  LDA  #LF  LOAD LINE FEED
FA46  86 0A  684  LDA  #LF  LOAD LINE FEED
FA46  86 0A  685  LDA  #LF  LOAD LINE FEED
FA46  86 0A  686  LDA  #LF  LOAD LINE FEED
FA46  86 0A  687  LDA  #LF  LOAD LINE FEED
FA46  86 0A  688  LDA  #LF  LOAD LINE FEED
FA46  86 0A  689  LDA  #LF  LOAD LINE FEED
FA46  86 0A  690  LDA  #LF  LOAD LINE FEED
FA46  86 0A  691  LDA  #LF  LOAD LINE FEED
FA46  86 0A  692  LDA  #LF  LOAD LINE FEED
FA46  86 0A  693  LDA  #LF  LOAD LINE FEED
FA46  86 0A  694  LDA  #LF  LOAD LINE FEED
FA46  86 0A  695  LDA  #LF  LOAD LINE FEED
FA46  86 0A  696  LDA  #LF  LOAD LINE FEED
FA46  86 0A  697  LDA  #LF  LOAD LINE FEED
FA46  86 0A  698  LDA  #LF  LOAD LINE FEED
FA46  86 0A  699  LDA  #LF  LOAD LINE FEED
FA46  86 0A  700  STB  ,S  OVERLAY OLD ONE ON STACK
FA58  2D 18  706  CHKABT  BSR  XQCIDT  ATTEMPT INPUT
FA58  8D 18  707  BCC  CHKRTN  BRANCH NO TO RETURN
FA58  8D 02  709  BNE  CHKWT  BRANCH NO TO PAUSE
FA60  53  710  CHKSEC  COMB  SET  CARRY
FA61  39  711  CHKRTN  RTS  RETURN TO CALLER WITH CC SET
FA62  8D 0A  712  CHKWT  BSR  XQPAUS  PAUSE FOR A MOMENT
FA64  8D 0C  713  BSR  XQCIDT  ? KEY FOR START
FA63  24  FA  714  BCC  CHKWT  LOOP UNTIL RECEIVED
FA68  81 18  715  CMPA  #CAN  ? ABORT SIGNALED FROM WAIT
FA6A  27 F4  716  BEQ  CHKSEC  BRANCH YES
FA6C  4F  717  CLRA  SET C=0 FOR NO ABORT
FA6D  39  718  RTS  AND RETURN
FA6E  6E 9D E5 78  721  XQPAUS  JMP  [VECTAB+.PAUSE,PCR]  TO PAUSE ROUTINE
FA72  AD 9D E5 62  722  XQCIDT  JSR  [VECTAB+.CIDTA,PCR]  TO INPUT ROUTINE
FA76  4F 50 2D 04  723  MSHOWP  FCB  'O','P','-',EOT OPCODE PREP
FA7D  8D 42  737  NMIR  BSR  LDDP  LOAD PAGE AND VERIFY STACK
FA7F  0D 8F  738  TST  MISFLG  ? THRU A BREAKPOINT
FA81  26 34  739  BNE  NMICON  BRANCH IF SO TO CONTINUE
FA83  0D 90  740  TST  SWICNT  ? INHIBIT "SWI" DURING TRACE
FA85  2B 29  741  BMI  NMITRC  BRANCH YES
FA87  30 6C  742  LEAX  12,S  OBTAIN USERS STACK POINTER
FA89  9C F8  743  CMPX  SLEVEL  ? TO TRACE HERE
FA8B  25 23  744  BLO  NMITRC  BRANCH IF TOO LOW TO DISPLAY
FA8D  30 8C E9  745  LEAX  MSHOWP,PCR  LOAD OP PREP
FA90  3F  746  SWI  SEND TO CONSOLE
FA91  02  748  FCB  POATA1  FUNCTION
FA92  09 8E  749  LEAX  LASTOP,PCR  POINT TO LAST OP
FA94  3F  750  SWI  SEND OUT AS HEX
FA99  05  751  FCB  OUT4HS  FUNCTION
FA99  25 17  752  BSR  REGPRS  FOLLOW MEMORY WITH REGISTERS
FA9C  25 37  753  BCS  ZBKCMD  BRANCH IF "CANCEL"
FA9E  06 8E  754  ROR  DELIM  RESTORE CARRY BIT
FAA0  25 33  755  BCS  ZBKCMD  BRANCH IF "CANCEL"
FAA2  9E 91  756  LDX  TRACEC  LOAD TRACE COUNT
FAA4  27 2F  757  BEQ  ZBKCMD  IF ZERO TO COMMAND HANDLER
FAA6  30 1F  758  LEAX  -1,X  MINUS ONE
FAA8  9F 91  759  STX  TRACEC  REFRESH
FAAA  27 29  760  BEQ  ZBKCMD  STOP TRACE WHEN ZERO
FAAC  8D AA  761  BSR  CHKABT  ? ABORT THE TRACE
FAAE  25 25  762  BCS  ZBKCMD  BRANCH YES TO COMMAND HANDLER
FABA  16 03 F7  763  NMITRC  LBRA  CTRCE3  NO, TRACE ANOTHER INSTRUCTION
FAB3  17 01 B9  765  REGPRS  LBSR  REGPR  PRINT REGISTERS AS FROM COMMAND
FAB6  39  766  RTS  RETURN TO CALLER
FAB7  767  768  * JUST EXECUTED THRU A BRKPNT.  NOW CONTINUE NORMALLY
FAB7  0F 8F  769  NMICON  CLR  MISFLG  CLEAR THRU FLAG
FAB9  17 02 EB  770  LBSR  ARMBK2  ARM BREAKPOINTS
FABC  3B  771  RTI  RTI AND  CONTINUE USERS PROGRAM
FABD  772
FABD  773  * LDDP - SETUP DIRECT PAGE REGISTER, VERIFY STACK.
FABD  774  * AN INVALID STACK CAUSES A RETURN TO THE COMMAND
FABD  775  * HANDLER.
FABD  776  * INPUT: FULLY STACKED REGISTERS FROM AN INTERRUPT
FABD  777  * OUTPUT: DPR LOADED TO WORK PAGE
FABD  778
FABD  3F 07 20 04  779  ERRMSG  FCB  '?',BELL,$20,EOT ERROR RESPONSE
FAC1  780
FAC1  E6 8D E4 D8  781  LDDP  LDB  BASEPG,PCR  LOAD DIRECT PAGE HIGH BYTE
FACS  1F 9B  782  TFR  B,DP  SETUP DIRECT PAGE REGISTER
FAC7  A1 63  783  CMPA  3,S  ? IS STACK VALID
FAC9  27 25  784  BEQ  RTS  YES, RETURN
FACB  10 DE 97  785  LDS  RSTACK  RESET TO INITIAL STACK POINTER
FACE  30 8C EC  786  ERROR  LEAX  ERRMSG,PCR  LOAD ERROR REPORT
FAD1  3F  787  SWI  SEND OUT BEFORE REGISTERS
FAD2  03  788  FCB  PDATA  ON NEXT LINE
FAD3  789  * FALL INTO BREAKPOINT HANDLER
FAD3  790
FAD3  791  ***********************************************
FAD3  792  * SWI FUNCTION 10]
FAD3  793  * BREAKPOINT PROGRAM FUNCTION
FAD3  794  * PRINT REGISTERS AND GO TO COMMAND HANDLER
FAD3  795  ***********************************************
FAD3  8D DE  796  ZBKPNT  BSR  REGPRS  PRINT OUT REGISTERS
FAD5  16 FE 21  797  ZBKCMD  LBRA  CMDNEP  NOW ENTER COMMAND HANDLER
FAD8  798
FAD8  799  ***********************************************
FAD8  800  * IRQ, RESERVED, SWI2 AND SWI3 INTERRUPT HANDLERS
FAD8  801  * THE DEFAULT HANDLING IS TO CAUSE A BREAKPOINT.
FAD8  802  ***********************************************
FAD8  803  SWI2R  EQU  *  SWI2 ENTRY
FAD8  804  SWI3R  EQU  *  SWI3 ENTRY
FAD8  805  IRQR  EQU  *  IRQ ENTRY
FAD8  806  RSRVDR  BSR  LDDP  SET BASE PAGE, VALIDATE STACK
FADA  20 F7  807  BRA  ZBKPNT  FORCE A BREAKPOINT
FADC  808
FADC  809  ***********************************************
FADC  810  * FIRQ HANDLER
FADC  811  * JUST RETURN FOR THE FIRQ INTERRUPT
FADC  812  ***********************************************
FABC  813  FIRQ  EQU  RTI  IMMEDIATE RETURN
FADC  814
FADC  815  ***********************************************
FADC  816  * DEFAULT I/O DRIVERS
FADC  817  ***********************************************
FADC  818
FADC  819  * CIDTA - RETURN CONSOLE INPUT CHARACTER
FADC  820  * OUTPUT: C=0 IF NO DATA READY, C=1 A=CHARACTER
FADC  821  * U VOLATILE
FADC  DE F0  822  CIDTA  LDU  VECTAB+.ACIA  LOAD ACIA ADDRESS
FADE  A6 C4  823  LDA  ,U  LOAD STATUS REGISTER
FAED  44  824  LSRA  TEST RECEIVER REGISTER FLAG
FAEE  24 02  825  BCC  CIRTN  RETURN IF NOTHING
FAEF  A6 41  826  LDA  1,U  LOAD DATA BYTE
FAE5  39  827  CIRTN  RTS  RETURN TO CALLER
FAE6  828
FAE6  829  * CION - INPUT CONSOLE INITIALIZATION
FAE6  830  * COON - OUTPUT CONSOLE INITIALIZATION
FAE6  831  * A,X VOLATILE
FAE6  832  CION  EQU  *
FAE6  833  COON  LDA  #3  RESET ACIA CODE
FAAB  9E F0  834  LDX  VECTAB+.ACIA  LOAD ACIA ADDRESS
FAEA  A7 84  835  STA  ,X  STORE INTO STATUS REGISTER
FAEC  86 51  836  LDA  #551  SET CONTROL
FAEE A7 84 837 STA ,X REGISTER UP
FAF0 39 838 RTS RTS RETURN TO CALLER
FAF1 839
FAF1 840 * THE FOLLOWING HAVE NO DUTIES TO PERFORM
FAF0 841 CIOFF EQU RTS CONSOLE INPUT OFF
FAF0 842 COOFF EQU RTS CONSOLE OUTPUT OFF
FAF1 843
FAF1 844 * CODTA - OUTPUT CHARACTER TO CONSOLE DEVICE
FAF1 845 * INPUT: A=CHARACTER TO SEND
FAF1 846 * OUTPUT: CHAR SENT TO TERMINAL WITH PROPER PADDING
FAF1 847 * ALL REGISTERS TRANSPARENT
FAF1 848
FAF1 34 47 849 CODTA PSHS U,D,CC SAVE REGISTERS,WORK BYTE
FAF3 DE F0 850 LDU VECTAB+.ACIA ADDRESS ACIA
FAF5 8D 1B 851 BSR CODTAO CALL OUTPUT CHAR SUBROTINE
FAF7 81 10 852 CMPA #DLE ? DATA LINE ESCAPE
FAF9 27 12 853 BEQ CODTRT YES, RETURN
FAFB 81 0D 855 CMPA #CR ? CR
FAFF 26 02 856 BNE CODTPD BRANCH NO
FB01 D6 F3 857 LDB VECTAB+.PAD+1 LOAD NEW LINE PAD COUNT
FB03 4F 858 CODTPD CLRA CREATE NULL
FB04 E7 E4 859 STD ,S SAVE COUNT
FB06 BC 860 FCB SKIP2 ENTER LOOP
FB07 8D 09 861 BSR CODTAO SEND NULL
FB09 6A E4 862 DEC ,S ? FINISHED
FB0B 2A FA 863 BFL CODTPL NO, CONTINUE WITH MORE
FB0D 39 C7 864 CODTRT PULS PC,U,D,CC RESTORE REGISTERS AND RETURN
FB0F 865
FB0F 17 FF 5C 866 CODTAD LBSR XPQAUS TEMPORARY GIVE UP CONTROL
FB12 E6 C4 867 CODTAO LDB ,U LOAD ACIA CONTROL REGISTER
FB14 C5 02 868 BITB #502 ? TX REGISTER CLEAR >LSAB FIXME
FB16 26 F7 869 BNE CODTAD RELEASE CONTROL IF NOT
FB18 A7 41 870 STA 1,U STORE INTO DATA REGISTER
FB1A 39 871 RTS RETURN TO CALLER
FB1B 872 *E
FB1B 873
FB1B 874 * BSON - TURN ON READ/VERIFY/PUNCH MECHANISM
FB1B 875 * A IS VOLATILE
FB1B 876
FB1B 86 11 877 BSON LDA #$511 SET READ CODE
FB1D 66 FF 878 TST 6,S ? READ OR VERIFY
FB1F 26 01 879 BNE BSON2 BRANCH YES
FB21 4C 880 INCA SET TO WRITE
FB22 3F 881 BSON2 SWI PERFORM OUTPUT
FB23 01 882 FCB OUTCH FUNCTION
FB24 0C 8F 883 INC MISFLG SET LOAD IN PROGRESS FLAG
FB26 39 884 RTS RETURN TO CALLER
FB27 885
FB27 886 * BSOFF - TURN OFF READ/VERIFY/PUNCH MECHANISM
FB27 887 * A,X VOLATILE
FB27 888
FB27 86 14 888 BSOFF LDA #$514 TO DC4 - STOP
FB29 3F 889 SWI SEND OUT
FB2A 01 890 FCB OUTCH FUNCTION
FB2B 4A 891 DECA CHANGE TO DC3 (X-OFF)
FB2C 3F 892 SWI SEND OUT
FB2D 01 893 FCB OUTCH FUNCTION
FB2E 0A 8F 894 DEC MISFLG CLEAR LOAD IN PROGRESS FLAG
FB30 8E 61 A8 895 LDX #$25000 DELAY 1 SECOND (2MHZ CLOCK)
FB33 30 1F 896 BSOFSP LEAX -1,X COUNT DOWN
FB35 26 FC 897 BNE BSOFSP LOOP TILL DONE
FB37 39 898 RTS RETURN TO CALLER
FB38 899
FB38 900 * BSDTA - READ/VERIFY/PUNCH HANDLER
FB38 901 * INPUT: S+6-CODE BYTE, VERIFY(-1),PUNCH(0),LOAD(1)
FB38 902 S+4=START ADDRESS
FB38 903 * S+2=STOP ADDRESS
FB38 904 * S+0=RETURN ADDRESS
FB38 905 * OUTPUT: Z=1 NORMAL COMPLETION, Z=0 INVALID LOAD/VER
FB38 906 * REGISTERS ARE VOLATILE
FB38 907
FB38 EE 62 908 BSDTA LDU 2,S U=TO ADDRESS OR OFFSET
FB3A 6D 66 909 TST 6,S ? PUNCH
FB3C 27 54 910 BEQ BSDPUN BRANCH YES
FB3E 911 * DURING READ/VERIFY: S+2=MSB ADDRESS SAVE BYTE
FB3E 912 * S+1=BYTE COUNTER
FB3E 913 * S+0=CHECKSUM
FB3E 914 * U HOLDS OFFSET
FB3E 32 7D 915 LEAS -3,S ROOM FOR WORK/COUNTER/CHECKSUM
FB40 3F 916 BSDLD1 SWI GET NEXT CHARACTER
FB41 61 917 FCB INCHNP FUNCTION
FB42 81 53 918 BSDLD2 CMPA #'S'? START OF S1/S9
FB44 26 FA 919 BNE BSDLD1 BRANCH NOT
FB46 90 920 SWI GET NEXT CHARACTER
FB47 00 921 FCB INCHNP FUNCTION
FB48 81 39 922 CMPA #'9'? HAVE S9
FB4A 27 22 923 BEQ BSDSRT YES, RETURN GOOD CODE
FB4C 81 31 924 CMPA #'1'? HAVE NEW RECORD
FB4E 6F E4 925 BNE BSDLD2 BRANCH IF NOT
FB50 6F E4 926 CLR ,S CLEAR CHECKSUM
FB52 27 BSF BYTE OBTAIN BYTE COUNT
FB54 EE 61 928 STB 1,S SAVE FOR DECREMENT
FB56 8D 1D 929 * READ ADDRESS
FB56 EE 62 930 BSDNXT BSDR BYTE NEXT BYTE
FB56 27 0C 931 BEQ BSDEOL BRANCH IF CHECKSUM
FB58 6D 69 938 TST 9,S ? VERIFY ONLY
FB5E 6D 02 939 BMI BSDCMP YES, ONLY COMPARE
FB60 EE 64 940 STB ,Y STORE INTO MEMORY
FB62 8A 1B 941 BSDCMP CMPB ,Y+ ? VALID RAM
FB64 27 F2 942 BEQ BSDNXT YES, CONTINUE READING
FB66 35 92 943 BSDSRT PULS PC,X,A RETURN WITH Z SET PROPER
FB70 944
FB70 4C 945 BSDEOL INCA ? VALID CHECKSUM
FB71 81 CD 946 BEQ BSDLD1 BRANCH S9S
FB73 20 F9 947 BRA BSDSRT RETURN 2=0 INVALID
FB75 948
FB75 8D 12 949 * BYTE BUILDS 8 BIT VALUE FROM TWO HEX DIGITS IN
FB75 EE 60 950 BYTE BSDR BYTHEX OBTAIN FIRST HEX
FB77 7D 16 951 LDB #16 PREPARE SHIFT
FB79 3D 952 MUL OVER TO A
FB7A 8D 0D 953 BSDR BYTHEX OBTAIN SECOND HEX
FB7C 34 04 954 PSHS B SAVE HIGH HEX
FB7E AB E0 955 ADDA ,S+ COMBINE BOTH SIDES
FB80 1F 89 956 TFR A,B SEND BACK IN B
FB82 AB 62 957 ADDA 2,S COMPUTE NEW CHECKSUM
FB84 A7 62 958 STA 2,S STORE BACK
FB86 6A 63 959 DEC 3,S DECREMENT BYTE COUNT
FB88 39 960 BYTRTS RTS RETURN TO CALLER
FB89 961
FB89 3F 962 BYTHEX SWI GET NEXT HEX
FB8A 00 963 FCB INCHNP CHARACTER
FB8B 17 01 D4 964 LBSR CNVHEX CONVERT TO HEX
FB8E 27 F8 965 BEQ BYTRTS RETURN IF VALID HEX
FB90 35 F2 966 PULS PC,U,Y,X,A RETURN TO CALLER WITH Z=0
FB92 967
FB92 968 * PUNCH STACK USE: S+8=TO ADDRESS
FB92 969 * S+6=RETURN ADDRESS
FB92 970 * S+4=SAVED PADDING VALUES
FB92 971 * S+2 FROM ADDRESS
FB92 972 * S+1=FRAME COUNT/CHECKSUM
FB92 973  * S+0=BYTE COUNT
FB92 DE F2 974 BSDPUN LDU VECTAB+.PAD LOAD PADDING VALUES
FB94 AE 64 975 LDX 4,S X+FROM ADDRESS
FB96 34 56 976 PSHS U,X,D CREATE STACK WORK AREA
FB98 CC 00 18 977 LDD #24 SET A=0, B=24
FB9B D7 F2 978 STB VECTAB+.PAD SETUP 24 CHARACTER PADS
FB9D 3F 979 SWI SEND NULLS OUT
FB9E 01 980 FCB OUTCH FUNCTION
FB9F C6 04 981 LDB #4 SETUP NEW LINE PAD TO 4
FBA1 DD F2 982 STD VECTAB+.PAD SETUP PUNCH PADDING
FBA3 983  * CALCULATE SIZE
FBA3 EC 68 984 BSDGO LDD 8,S LOAD TO
FBA5 A3 62 985 SUBD 2,S MINUS FROM.LENGTH
FBA7 10 83 00 18 986 CMPD #24 ? MORE THAN 23
FBA9 25 02 987 BLO BSPOK NO, OK
FBAE C6 17 988 LDB #23 FORCE TO 23 MAX
FBAF 5C 989 BSDPUN INCB PREPARE COUNTER
FBB0 E7 E4 990 STB ,S STORE BYTE.COUNT
FBB2 CB 03 991 ADDD #3 ADJUST TO FRAME COUNT
FBB4 E7 61 992 STB 1,S SAVE
FBB6 30 8C 33 994 LEAX <BSPSTR,PCR LOAD START RECORD HEADER
FBB9 3F 995 SWI SEND OUT
FBBB 5F 996 FCB PDATA FUNCTION
FBBB 5P 998 CLRB INITIALIZE CHECKSUM
FBBE 8D 27 1000 BSDPUN2 SEND FRAME COUNT
FBC0 1001  *DATA ADDRESS
FBC2 8D 25 1002 BSR BSDPUN2 SEND ADDRESS HI
FBC3 8D 23 1003 BSR BSDPUN2 SEND ADDRESS LOW
FBC4 AE 62 1004  *PUNCH DATA
FBC6 8D 1F 1005 LDX 2,S LOAD START DATA ADDRESS
FBC9 8A E4 1006 BSMRE BSR BSDPUN2 SEND OUT NEXT BYTE
FBCA 26 FA 1007 DEC ,S ? FINAL BYTE
FBB3 54 62 1008 BNE BSMRE LOOP IF NOT DONE
FBB6 AF 62 1009 STX 2,S UPDATE FROM ADDRESS VALUE
FBBE 1010  *PUNCH CHECKSUM
FBC2 53 1011 COMB COMPLEMENT
FBC5 E7 61 1012 STB 1,S STORE FOR SENDOUT
FBCD 30 61 1013 LEAX 1,S POINT TO IT
FBCD 8D 14 1014 BSR BSDPUNC SEND OUT EX HEX
FBD5 AE 68 1015 LDX 8,S LOAD TOP ADDRESS
FBD7 AC 62 1016 CMPX 2,S ? DONE
FBD9 24 C8 1017 BHS BSDPUN BRANCH NOT
FBD9 30 8C 11 1018 LEAX <BSPEOF,PCR PREPARE END OF FILE
FBB4 56 62 1019 SWI SEND OUT STRING
FBCF 03 1020 FCB PDATA FUNCTION
FBD0 EC 64 1021 LDD 4,S RECOVER PAD COUNTS
FBD2 DD F2 1022 STD VECTAB+.PAD RESTORE
FBD4 4F 1023 CLR A SET Z=1 FOR OK RETURN
FBD5 35 D6 1024 PULS PC,U,X,D RETURN WITH OK CODE
FBE7 1025
FBE8 EB 84 1026 BSDPUN2 ADDD ,X ADD TO CHECKSUM
FBE9 16 FD ED 1027 BSDPUN LBRA ZOUT2H SEND OUT AS HEX AND RETURN
FBEA 1028
FBEA 53 31 04 1029 BSDPSTR FCB 'S',1,EOT CR,LF,NULLS,S,1
FBEF 53 39 30 33 30 30 + 1030 BSPEOF FCC /S9030000FC/ EOF STRING
FBF9 DD 0A 04 1031 FCB CR,LF,EOT
FBCF 1032
FBCF 1033  * HSDTA - HIGH SPEED PRINT MEMORY
FBCF 1034  * INPUT: S+4=START ADDRESS
FBCF 1035  * S+2=STOP ADDRESS
FBCF 1036  * S+0=RETURN ADDRESS
FBCF 1037  * X,D VOLATILE
FBCF 1038
FBCF 1039  * SEND TITLE
FBCF 3F 1040 HSDTA SWI SEND NEW LINE
**DUNFIELD 6809 ASSEMBLER: ASSIST09**

**FBFD 06** 1041  FCB  PCRLF  FUNCTION
**FBFE C6 06** 1042  LDB  #6  PREPARE 6 SPACES
**FC00 3F 06** 1043  HSBLNK  SWI  SEND BLANK
**FC01 07** 1044  FCB  SPACE  FUNCTION
**FC02 5A** 1045  DECB  COUNT DOWN
**FC03 26 FB** 1046  BNE  HSBLNK  LOOP IF MORE
**FC05 5F** 1047  CLRB  SETUP BYTE COUNT
**FC06 1F 9B** 1048  HSHTTL  TFR  B,A  PREPARE FOR CONVERT
**FC08 17 FD DB** 1049  LBSR  ZOUTHX  CONVERT TO A HEX DIGIT
**FC0B 3F** 1050  SWI  SEND BLANK
**FC0C 07** 1051  FCB  SPACE  FUNCTION
**FC0D 3F** 1052  SWI  SEND ANOTHER
**FC0E 07** 1053  FCB  SPACE  BLANK
**FC0F 5C** 1054  INCB  UP ANOTHER
**FC10 C1 10** 1055  CMPB  #$10  ? PAST 'F'
**FC12 25 F2** 1056  BLO  HSHTTL  LOOP UNTIL SO
**FC14 3F** 1057  HSHTNLN  SWI  TO NEXT LINE
**FC15 3F** 1058  SWI  PRINT OUT ADDRESS
**FC16 25 2F** 1059  BCS  HSDRTN  RETURN IF USER ENTERED CTL-X
**FC18 30 64** 1060  LEAX  4,S  POINT AT ADDRESS TO CONVERT
**FC1A 3F** 1061  SWI  PRINT OUT ADDRESS
**FC1B 05** 1062  FCB  OUT4HS  FUNCTION
**FC1C AE 64** 1063  LDX  4,S  LOAD ADDRESS PROPER
**FC1E C6 10** 1064  LDB  #16  NEXT SIXTEEN
**FC20 3F** 1065  HSHTNLX  SWI  CONVERT BYTE TO HEX AND SEND
**FC21 04** 1066  FCB  OUT2HS  FUNCTION
**FC22 5A** 1067  DECB  COUNT DOWN
**FC23 26 FB** 1068  BNE  HSHTNLX  LOOP IF NOT SIXTEENTH
**FC25 3F** 1069  SWI  SEND BLANK
**FC26 07** 1070  FCB  SPACE  FUNCTION
**FC27 AE 64** 1071  LDX  4,S  RELOAD FROM ADDRESS
**FC29 C6 10** 1072  LDB  #16  COUNT
**FC2B A6 80** 1073  HSHCHR  LDA ,X+  NEXT BYTE
**FC2D 2B 04** 1074  BMI  HSHDOTT  TOO LARGE, TO A DOT
**FC2F B1 20** 1075  CMPA  '#.'  ? LOWER THAN A BLANK
**FC31 24 02** 1076  BHS  HSHCOK  NO, BRANCH OK
**FC33 3E 2E** 1077  HSHDOTT  LDA  '#.'  CONVERT INVALID TO A BLANK
**FC35 3F** 1078  HSHCOK  SWI  SEND CHARACTER
**FC36 01** 1079  FCB  OUTCH  FUNCTION
**FC37 5A** 1080  DECB  ? DONE
**FC38 26 F1** 1081  BNE  HSHCHR  BRANCH NO
**FC3A AC 62** 1082  CMPX  2,S  ? PAST LAST ADDRESS
**FC3C 24 09** 1083  BHS  HSDRTN  QUIT IF SO
**FC3E AF 64** 1084  STX  4,S  UPDATE FROM ADDRESS
**FC40 A6 65** 1085  LDA  5,S  LOAD LOW BYTE ADDRESS
**FC42 4B** 1086  ASLA  ?  TO SECTION BOUNDARY
**FC43 CE 6F** 1087  BNE  HSHLRN  BRANCH IF NOT
**FC45 20 85** 1088  BRA  HSDTA  BRANCH IF SO
**FC47 3F** 1089  HSDRTN  SWI  SEND NEW LINE
**FC48 06** 1090  FCB  PCRLF  FUNCTION
**FC49 39** 1091  RTS  RETURN TO CALLER
**FC4A 01** 1092  *F*
**FC4A** 1093
**FC4A** 1094  ************************************************************************
**FC4A** 1095  * A S S I S T 0 9  C O M M A N D S *
**FC4A** 1096  ************************************************************************
**FC4A 06** 1097
**FC4A** 1098  **************************************************************************
**FC4A 8D 23** 1099  BSR  REGFRT  PRINT REGISTERS
**FC4C 4C** 1100  INCA  SET FOR CHANGE FUNCTION
**FC4D 8D 21** 1101  BSR  REGCCHG  GO CHANGE, DISPLAY REGISTERS
**FC50 39** 1102  RTS  RETURN TO COMMAND PROCESSOR
**FC50 1103
**FC50** 1104  ************************************************************************
**FC50** 1105  * REGFRT - PRINT/CHANGE REGISTERS SUBROUTINE
**FC50** 1106  * WILL ABDICT TO 'CMDBAD' IF OVERFLOW DETECTED DURING
**FC50** 1107  * A CHANGE OPERATION. CHANGE DISPLAYS REGISTERS WHEN
**FC50** 1108  * DONE.

---

**IC0C 07** 1051  FCB  SPACE  FUNCTION
**IC0D 3F** 1052  SWI  SEND ANOTHER
**IC0E 07** 1053  FCB  SPACE  BLANK
**IC0F 5C** 1054  INCB  UP ANOTHER
**IC10 C1 10** 1055  CMPB  #$10  ? PAST 'F'
**IC12 25 F2** 1056  BLO  HSHTTL  LOOP UNTIL SO
**IC14 3F** 1057  HSHTNLN  SWI  TO NEXT LINE
**IC15 3F** 1058  SWI  PRINT OUT ADDRESS
**IC16 25 2F** 1059  BCS  HSDRTN  RETURN IF USER ENTERED CTL-X
**IC18 30 64** 1060  LEAX  4,S  POINT AT ADDRESS TO CONVERT
**IC1A 3F** 1061  SWI  PRINT OUT ADDRESS
**IC1B 05** 1062  FCB  OUT4HS  FUNCTION
**IC1C AE 64** 1063  LDX  4,S  LOAD ADDRESS PROPER
**IC1E C6 10** 1064  LDB  #16  NEXT SIXTEEN
**IC20 3F** 1065  HSHTNLX  SWI  CONVERT BYTE TO HEX AND SEND
**IC21 04** 1066  FCB  OUT2HS  FUNCTION
**IC22 5A** 1067  DECB  COUNT DOWN
**IC23 26 FB** 1068  BNE  HSHTNLX  LOOP IF NOT SIXTEENTH
**IC25 3F** 1069  SWI  SEND BLANK
**IC26 07** 1070  FCB  SPACE  FUNCTION
**IC27 AE 64** 1071  LDX  4,S  RELOAD FROM ADDRESS
**IC29 C6 10** 1072  LDB  #16  COUNT
**IC2B A6 80** 1073  HSHCHR  LDA ,X+  NEXT BYTE
**IC2D 2B 04** 1074  BMI  HSHDOTT  TOO LARGE, TO A DOT
**IC2F B1 20** 1075  CMPA  '#.'  ? LOWER THAN A BLANK
**IC31 24 02** 1076  BHS  HSHCOK  NO, BRANCH OK
**IC33 3E 2E** 1077  HSHDOTT  LDA  '#.'  CONVERT INVALID TO A BLANK
**IC35 3F** 1078  HSHCOK  SWI  SEND CHARACTER
**IC36 01** 1079  FCB  OUTCH  FUNCTION
**IC37 5A** 1080  DECB  ? DONE
**IC38 26 F1** 1081  BNE  HSHCHR  BRANCH NO
**IC3A AC 62** 1082  CMPX  2,S  ? PAST LAST ADDRESS
**IC3C 24 09** 1083  BHS  HSDRTN  QUIT IF SO
**IC3E AF 64** 1084  STX  4,S  UPDATE FROM ADDRESS
**IC40 A6 65** 1085  LDA  5,S  LOAD LOW BYTE ADDRESS
**IC42 4B** 1086  ASLA  ?  TO SECTION BOUNDARY
**IC43 CE 6F** 1087  BNE  HSHLRN  BRANCH IF NOT
**IC45 20 85** 1088  BRA  HSDTA  BRANCH IF SO
**IC47 3F** 1089  HSDRTN  SWI  SEND NEW LINE
**IC48 06** 1090  FCB  PCRLF  FUNCTION
**IC49 39** 1091  RTS  RETURN TO CALLER
**IC4A 01** 1092  *F*
**IC4A** 1093
**IC4A** 1094  ************************************************************************
**IC4A** 1095  * A S S I S T 0 9  C O M M A N D S *
**IC4A** 1096  ************************************************************************
**IC4A 06** 1097
**IC4A** 1098  **************************************************************************
FC50 1109 * REGISTER MASK LIST CONSISTS OF:

FC50 1110  A) CHARACTERS DENOTING REGISTER

FC50 1111  B) ZERO FOR ONE BYTE, -1 FOR TWO

FC50 1112  C) OFFSET ON STACK TO REGISTER POSITION

FC50 1113  * INPUT: SP+4=STACKED REGISTERS

FC50 1114  A=0 PRINT, A#0 PRINT AND CHANGE

FC50 1115  OUTPUT: (ONLY FOR REGISTER DISPLAY)

FC50 1116  C=1 CONTROL-X ENTERED, C=0 OTHERWISE

FC50 1117  * VOLATILE: D,X (CHANGE)

FC50 1118  B,X (DISPLAY)

FC50 1119  *******************************************

FC50 50 43 FF 13 1120 REGMSK FCB 'P','C',-1,19 PC REG

FC54 41 00 0A 1121          FCB     'A',0,10 A REG

FC57 42 00 0B 1122          FCB     'B',0,11 B REG

FC5A 58 FF 0D 1123          FCB     'X',-1,13 X REG

FC5D 59 FF 0E 1124          FCB     'Y',-1,15 Y REG

FC60 55 FF 11 1125          FCB     'U',-1,17 U REG

FC63 53 FF 01 1126          FCB     'S',-1,1 S REG

FC66 43 43 00 09 1127          FCB     'C','C',0,9 CC REG

FC6A 44 50 00 0C 1128          FCB     'D','P',0,12 DP REG

FC6E 00 1129          FCB     0 END OF LIST

FC6F                      1130

FC6F 4F                  1131  REGPRT CLRA SETUP PRINT ONLY FLAG

FC70 30 E8 10 1132  REGCHG LEAX 4+12,S READY STACK VALUE

FC73 34 32 1133  PSHS Y,X,A SAVE ON STACK WITH OPTION

FC75 31 8C D8 1134  LEAY REGMSK,PCR LOAD REGISTER MASK

FC78 EC A0 1135  LDD ?,Y+ LOAD NEXT CHAR OR <=0

FC7A 4D 04 1136  TSTA ? END OF CHARACTERS

FC7B 2F 04 1137  BLE REGP2 BRANCH NOT CHARACTER

FC7D 3F 1138  SWI SEND TO CONSOLE

FC7E 01 1139  FCB OUTFCH FUNCTION BYTE

FC7F 20 F7 1140  BRA REGF1 CHECK NEXT

FC81 86 2D 1141  REGP2 LDA #'-' READY '-'

FC83 3F 1142  SWI SEND OUT

FC84 01 1143  FCB OUTFCH WITH OUTFCH

FC85 30 E5 1144  LEAX B,S X->REGISTER TO PRINT

FC87 2E 12 1145  TST ? CHANGE OPTION

FC89 26 12 1146  BNE REGCNG BRANCH YES

FC8B 6D 3F 1147  TST -1,Y ? ONE OR TWO BYTES

FC8D 27 03 1148  BEQ REGF3 BRANCH ZERO MEANS ONE

FC8F 3F 1149  SWI PERFORM WORD HEX

FC90 05 1150  FCB OUT4HS FUNCTION

FC91 8C 1151  FCB SKP2 SKIP BYTE PRINT

FC92 3F 1152  REGF3 SWI PERFORM BYTE HEX

FC93 04 1153  FCB OUT2HS FUNCTION

FC94 EC A0 1154  LDD ?,Y+ TO FRONT OF NEXT ENTRY

FC96 5D 1155  TSTB ? END OF ENTRIES

FC97 26 DF 1156  BNE REGF1 LOOP IF MORE

FC99 3F 1157  SWI FORCE NEW LINE

FC9A 06 1158  FCB PCRLF FUNCTION

FC9B 35 B2 1159  REGRTN PULS PC,Y,X,A RESTORE STACK AND RETURN

FC9D 1160

FC9D 8D 40 1161  REGCNG BSR BLDNBN INPUT BINARY NUMBER

FC9F 27 10 1162  BEQ REGNXC IF CHANGE THEN JUMP

FCA1 81 0D 1163  CMPA #CR ? NO MORE DESIRED

FCA3 27 1E 1164  BEQ REGAGN BRANCH NOPE

FCA5 E6 3F 1165  LDB -1,Y LOAD SIZE FLAG

FCA7 5A 1166  DECB MINUS ONE

FCA8 50 1167  NEGB MAKE POSITIVE

FCA9 58 1168  ASLB TIMES TWO (=2 OR =4)

FCA4 5F 1169  REGSKP SWI PERFORM SPACES

FCA7 1170  FCB SPACE FUNCTION

FCA4 5A 1171  DECB

FCD6 26 FB 1172  BNE REGSKP LOOP IF MORE

FCF0 1173  BRA REG4 CONTINUE WITH NEXT REGISTER

FCB7 A7 E4 1174  REGNXC STA ,S SAVE DELIMITER IN OPTION

FCB3 DC 9B 1175  (ALWAYS > 0)
FCB5 6D 3F 1177  TST  -1,Y  ? TWO BYTES WORTH
FCB7 26 02 1178  BNE  REGTWO  BRANCH YES
FCB9 A6 82 1179  LDA  ,-X  SETUP FOR TWO
FCBB ED 84 1180  REGTWO  STD  ,X  STORE IN NEW VALUE
FCBD A6 E4 1181  LDA  ,-S  RECOVER DELIMITER
FCBF 81 0D 1182  CMPA  #CR  ? END OF CHANGES
FCC1 26 D1 1183  BNE  REG4  NO, KEEP ON TRUCK'N
FCC3 30 8D E2 8A 1186  REGAGN  LEAX  TSTACK,PCR  LOAD TEMP AREA
FCC7 C6 15 1187  LDB  #21  LOAD COUNT
FCC9 35 02 1188  REGTF1  PULS  A  NEXT BYTE
FCCB A7 80 1189  STA  ,-X  STORE INTO TEMP
FCCD 5A 1190  DECB  COUNT DOWN
FCD0 10 EE 88 EC 1192  LDS  -20,X  LOAD NEW STACK POINTER
FCD4 C6 15 1193  LDB  #21  LOAD COUNT AGAIN
FCD6 A6 82 1194  REGTF2  LDA  ,-X  NEXT TO STORE
FCD8 34 02 1195  PSHS  A  BACK ONTO NEW STACK
FCD9 4F 1216  BLDNNB  CLRA  NO DYNAMIC DELIMITER
FCE0 8C 1217  FCB  SKIP2  SKIP NEXT INSTRUCTION
FCE1 86 20 1219  BLDNUM  LDA  #' '  ALLOW LEADING BLANKS
FCE3 97 8E 1220  STA  DELIM  STORE AS DELIMITER
FCE5 6E 9D E3 03 1221  JMP  [VECTAB+.EXPAN,PCR]  TO EXP ANALYZER
FCE9 222  
FCE9 223  * THIS IS THE DEFAULT SINGLE ROM ANALYZER. WE ACCEPT:
FCE9 224  * 1) HEX INPUT
FCE9 225  * 2) 'M' FOR LAST MEMORY EXAMINE ADDRESS
FCE9 226  * 3) 'P' FOR PROGRAM COUNTER ADDRESS
FCE9 227  * 4) 'W' FOR WINDOW VALUE
FCE9 228  * 5) '@' FOR INDIRECT VALUE
FCE9 34 14 1229  EXP1  PSHS  X,B  SAVE REGISTERS
FCEB 8D 5C 1230  EXPDLM  BSR  BLDHXI  CLEAR NUMBER, CHECK FIRST CHAR
FCED 27 18 1231  BEQ  EXP2  IF HEX DIGIT CONTINUE BUILDING
FCEF 32 1232  * SKIP BLANKS IF DESIRED
FCEF 91 8E 1233  CMPA  DELIM  ? CORRECT DELIMITER
FCF1 27 F8 1234  BEQ  EXPDLM  YES, IGNORE IT
FCF3 32 1235  * TEST FOR M OR P
FCF3 9E 9E 1236  LDX  ADDR  DEFAULT FOR 'M'
FCF5 81 4D 1237  CMPA  #M'  ? MEMORY EXAMINE ADDR WANTED
FCFF 27 16 1238  BEQ  EXPDL  BRANCH IF SO
FCF9 9E 93 1239  LDX  PCNTER  DEFAULT FOR 'P'
FCFB 81 50 1240  CMPA  #P'  ? LAST PROGRAM COUNTER WANTED
FCFD 27 10 1241  BEQ  EXPDL  BRANCH IF SO
FCFF 9E A0 1242  LDX  WINDOW  DEFAULT TO WINDOW
FD01 81 57 1243  CMPA  #W'  ? WINDOW WANTED
FD03 27 0A 1244  BEQ  EXPDL
FD05  35 94  1245  EXPRTN  PULS  PC,X,B  RETURN AND RESTORE REGISTERS
FD07  1246  *  GOT HEX, NOW CONTINUE BUILDING
FD07  8D 44  1247  EXP2  BSR  BLDHEX  COMPUTE NEXT DIGIT
FD09  27 FC  1248  BEQ  EXP2  CONTINUE IF MORE
FD0B  20 0A  1249  BRA  EXPCDL  SEARCH FOR +/-
FD0D  1250  *  STORE VALUE AND CHECK IF NEED DELIMITER
FD0D  AE 84  1251  EXPFDI  LDX ,X  INDIRECT DESIRED
FD0F  9F 9B  1252  EXPTDL  STX  NUMBER  STORE RESULT
FD11  0D 0E  1253  TST  DELIM  ?  TO FORCE A DELIMITER
FD13  27 F0  1254  BEQ  EXPRTN  RETURN IF NOT WITH VALUE
FD15  8D 62  1255  BSR  READ  OBTAIN NEXT CHARACTER
FD17  1256  *  TEST FOR + OR -
FD17  9E 9B  1257  EXPCDL  LDX  NUMBER  LOAD LAST VALUE
FD19  81 2B  1258  CMPA  #'+'  ?  ADD OPERATOR
FD1B  26 0E  1259  BNE  EXPCHM  BRANCH NOT
FD1D  8D 23  1260  BSR  READTRM  COMPUTE NEXT TERM
FD1F  34 02  1261  PSHS  A  SAVE DELIMITER
FD21  DC 9B  1262  LDD  NUMBER  LOAD NEW TERM
FD23  30 8B  1263  EXPADD  LEAX  D,X  ADD TO X
FD25  9F 9B  1264  STX  NUMBER  STORE AS NEW RESULT
FD27  30 02  1265  PULS  A  RESTORE DELIMITER
FD29  20 EC  1266  BRA  EXPCDL  NOW TEST IT
FD2B  1267  EXPCHM  CMPA  #'-'  ?  SUBTRACT OPERATOR
FD2D  8D 9D  1268  BSR  EXPTRM  OBTAIN NEXT TERM
FD2F  81 40  1269  CMPA  #'@'  ?  INDIRECTION DESIRED
FD31  27 DA  1270  BEQ  EXPCHM  BRANCH IF SO
FD33  9F 9C  1271  CLR     NUMBER  CLEAR NUMBER
FD35  8D 11  1272  BEQ  EXPFDI  BRANCH IF SO
FD37  20 0A  1273  BRA  EXPCDL  NOW TEST IT
FD39  0D 0E  1274  TST  DELIM  ?  TO FORCE A DELIMITER
FD3B  AE 84  1275  LDX  ,X  INDIRECT DESIRED
FD3D  9F 9B  1276  STX  NUMBER  STORE RESULT
FD3F  0D 0E  1277  TST  DELIM  ?  TO FORCE A DELIMITER
FD41  27 F0  1278  BEQ  EXPRTN  RETURN IF NOT WITH VALUE
FD43  8D 62  1279  BSR  READ  OBTAIN NEXT CHARACTER
FD45  1280  *  COMPUTE NEXT EXPRESSION TERM
FD47  26 0E  1281  BNE  EXPADD  ADD TO EXPRESSION
FD49  20 EC  1282  BRA  EXPADD  ADD TO EXPRESSION
FD4B  1283  BSR  BLDNUM  OBTAIN NEXT VALUE
FD4D  27 32  1284  BEQ  CNVRTS  RETURN IF VALID NUMBER
FD4F  16 FC  13  1285  BLDBAD  LBRA  CMDBAD  ABORT COMMAND IF INVALID
FD51  1286  1287  *********************************************
FD51  0F 9B  1288  *  BUILD BINARY VALUE USING INPUT CHARACTERS.
FD53  1289  *  INPUT:  A=ASCII HEX VALUE OR DELIMITER
FD55  1290  *  SP+0=RETURN ADDRESS
FD57  C6 10  1291  LDB     #16  PREPARE SHIFT
FD59  3D  1292  MUL     NUMBER  MULTIPLY
FD5B  86 04  1293  LDA     #4  ROTATE BINARY INTO VALUE
FD5D  1300  BNE  CNVRTS  RETURN IF NOT A NUMBER
FD5F  1301  LDB     #16  PREPARE SHIFT
FD61  3D  1302  MUL     NUMBER  MULTIPLY
FD63  1303  LDA     #4  ROTATE BINARY INTO VALUE
FD65  1304  BSR  ASLB  OBTAIN NEXT BIT
FD67  1305  ROL     NUMBER+1  INTO LOW BYTE
FD69  1306  ROL     NUMBER  INTO HI BYTE
FD6B  1307  DECA                    COUNT DOWN
FD6D  1308  BRA  BLDBAD  BRANCH IF MORE TO DO
FD6F  1309  BRA  CNVOK  SET GOOD RETURN CODE
FD71  1310  *********************************************
FD73  0F 9B  1294  *  VOLTILE: D
FD75  1295  *********************************************
FD77  0F 9C  1296  BLDHXI  CLR  NUMBER  CLEAR NUMBER
FD79  1297  CLR  NUMBER+1  CLEAR NUMBER
FD7B  8D 2A  1298  BLDHXC  BSR  READ  GET INPUT CHARACTER
FD7D  1300  BNE  CNVRTS  RETURN IF NOT A NUMBER
FD7F  1301  LDB     #16  PREPARE SHIFT
FD81  3D  1302  MUL     NUMBER  MULTIPLY
FD83  1303  LDA     #4  ROTATE BINARY INTO VALUE
FD85  1304  BSR  ASLB  OBTAIN NEXT BIT
FD87  1305  ROL     NUMBER+1  INTO LOW BYTE
FD89  1306  ROL     NUMBER  INTO HI BYTE
FD8B  1307  DECA                    COUNT DOWN
FD8D  1308  BRA  BLDBAD  BRANCH IF MORE TO DO
FD8F  1309  BRA  CNVOK  SET GOOD RETURN CODE
FD91  1310  *********************************************
FD93  0F 9B  1311  *********************************************
FD95  1312  *  CONVERT ASCII CHARACTER TO BINARY BYTE
FD62 1313  * INPUT:  A=ASCII
FD64 25 12 1320 BLO CNVRTS BRANCH NOT VALUE
FD66 81 39 1321 CMPA #9' ? POSSIBLE A-F
FD68 22 06 1322 BLE CNVGOT BRANCH NO TO ACCEPT
FD70 22 06 1324 CMPA #F' ? NOT TOO LARGE
FD72 80 07 1325 SUBA #7 DOWN TO BINARY
FD74 81 30 1327 CNVHEX CMPA #0' ? LOWER THAN A ZERO
FD76 81 39 1328 CMPA #A' ? LESS THEN TEN
FD78 81 46 1330 CMPA #F' ? NOT TOO LARGE
FD7A 81 46 1332 CNVOK ORCC #4 FORCE ZERO ON FOR VALID HEX
FD7C 81 3F 1334 CMPA #$3F READY "SWI" OPCODE
FD80 81 18 1336 CMPA #CAN ABORT COMMAND
FD82 27 C7 1338 BEQ BLDBAD BRANCH TO ABORT IF SO
FD84 81 41 1339 CMPA #A' ? LESS THEN TEN
FD86 25 0A 1340 BLO CNVRTS RETURN IF MINUS (INVALID)
FD88 81 46 1342 CMPA #F' ? NOT TOO LARGE
FD8A 81 46 1344 CMPA #F' ? NOT TOO LARGE
FD8C AE 6C 1346 LDX 12,S LOAD PROGRAM COUNTER
FD8E AD F1 1348 JSR [,S++] CALL USER SUBROUTINE
FD90 A6 B4 1349 BMI CNVRTS RETURN WHEN DONE
FD92 2B FA 1350 STA [,Y] STORE INTO OPCODE TABLE
FD94 A9 F7 1352 STA SWIBFL SHOW UPCOMING SWI NOT BRKPNT
FD96 17 01 B6 1353 LBSR CBKLRD SEARCH BREAKPOINTS
FD98 AE 6C 1355 LDX 12,S LOAD PROGRAM COUNTER
FD9A 17 01 98 1356 ARMBK2 DECBO COUNT DOWN
FD9C 2B C9 1358 BMI CNVRTS RETURN WHEN DONE
FD9E AD F1 1360 JSR [,S++] CALL USER SUBROUTINE
FDAB 8D C8 1362 LDA #0 CALL ADDRESS AS SUBROUTINE
FDAC 28 07 1364 FULS U,Y,X,DP,CC RESTORE USERS REGISTERS
FDAC 28 07 1366 LDA #0 CALL ADDRESS AS SUBROUTINE
FDAC 28 07 1368 LDA #0 CALL ADDRESS AS SUBROUTINE
FDAC 28 07 1370 LDA #0 CALL ADDRESS AS SUBROUTINE
FDAC 28 07 1372 LDA #0 CALL ADDRESS AS SUBROUTINE
FDAC 28 07 1374 LDA #0 CALL ADDRESS AS SUBROUTINE
FDAC 28 07 1376 LDA #0 CALL ADDRESS AS SUBROUTINE
FDAC 28 07 1378 LDA #0 CALL ADDRESS AS SUBROUTINE
**DUNFIELD 6809 ASSEMBLER: ASSIST09**

**PAGE: 22**

0A          1381  FCB  BRKPT  FUNCTION
20          1382  BRA  CGOBRK  LOOP UNTIL USER CHANGES PC
0A          1383

0A          1384  **************MEMORY - DISPLAY/CHANGE MEMORY

0A          1385  * CMEMN AND CMPADP ARE DIRECT ENTRY POINTS FROM

0A          1386  * THE COMMAND HANDLER FOR QUICK COMMANDS

0A          1387  CMEM  LBSR  CDNUM  OBTAIN ADDRESS
0A          1388  CMEMN  STD  ADDR  STORE DEFAULT
0A          1389  CMEM2  LDX  ADDR  LOAD POINTER
0A          1390  LSRB  ZOUT2H  SEND OUT HEX VALUE OF BYTE
0A          1391  LDA  #'-'  LOAD DELIMITER
0A          1392  SWI  SEND OUT
0A          1393  FCB  OUCH  FUNCTION
0A          1394  CMEM4  LSRB  BLDNNB  OBTAIN NEW BYTE VALUE
0A          1395  BEQ  CMENUM  BRANCH IF NUMBER
0A          1396  * COMA - SKIP BYTE
0A          1397  CMPA  #',,'  ? COMMA
0A          1398  BRN  CMNOC  BRANCH NOT
0A          1399  STX  ADDR  UPDATE POINTER
0A          1400  LEAX  1,X  TO NEXT BYTE
0A          1401  BRA  CMEM4  AND INPUT IT
0A          1402  CMENUM  LDB  NUMBER+1  LOAD LOW BYTE VALUE
0A          1403  BSR  MUPDAT  GO OVERLAY MEMORY BYTE
0A          1404  CMPA  #',,'  ? CONTINUE WITH NO DISPLAY
0A          1405  BEQ  CMEM4  BRANCH YES
0A          1406  * QUOTED STRING
0A          1407  CMNOC  CMPA  #$27  ? QUOTED STRING
0A          1408  BRN  CMNOC  BRANCH NO
0A          1409  CMESTR  BSR  READ  OBTAIN NEXT CHARACTER
0A          1410  CMPA  #$27  ? END OF QUOTED STRING
0A          1411  BEQ  CMSFCE  YES, QUIT STRING MODE
0A          1412  TFR  A,B  TO B FOR SUBROUTINE
0A          1413  BSR  MUPDAT  GO UPDATE BYTE
0A          1414  BRA  CMESTR  GET NEXT CHARACTER
0A          1415  * BLANK - NEXT BYTE
0A          1416  CMNOC  CMPA  #$20  ? BLANK FOR NEXT BYTE
0A          1417  BRN  CMNOC  BRANCH NOT
0A          1418  STX  ADDR  UPDATE POINTER
0A          1419  CMSFCE  SWI  GIVE SPACE
0A          1420  FCB  SPACE  FUNCTION
0A          1421  BRA  CMEM2  NOW PROMPT FOR NEXT
0A          1422  * LINE FEED - NEXT BYTE WITH ADDRESS
0A          1423  CMNOCB  CMPA  #$LF  ? LINE FEED FOR NEXT BYTE
0A          1424  BNE  CMNOCB  BRANCH NO
0A          1425  LDA  #$CR  GIVE CARRIAGE RETURN
0A          1426  SWI  TO CONSOLE
0A          1427  FCB  OUCH  HANDLER
0A          1428  STX  ADDR  STORE NEXT ADDRESS
0A          1429  BRA  CMPADP  BRANCH TO SHOW
0A          1430  * UP ARROW - PREVIOUS BYTE AND ADDRESS
0A          1431  CMNOC  CMPA  #$A  ? UP ARROW FOR PREVIOUS BYTE
0A          1432  BRN  CMNOC  BRANCH NOT
0A          1433  LEAX  #,X  DOWN TO PREVIOUS BYTE
0A          1434  STX  ADDR  STORE NEW POINTER
0A          1435  CMPADS  SWI  FORCE NEW LINE
0A          1436  FCB  PCTRL  FUNCTION
0A          1437  CMPADP  BSR  PRTADR  GO PRINT ITS VALUE
0A          1438  BRA  CMEM2  THEN PROMPT FOR INPUT
0A          1439  * SLASH - NEXT BYTE WITH ADDRESS
0A          1440  CMNOC  CMPA  #$/'  ? SLASH FOR CURRENT DISPLAY
0A          1441  BNE  CMPADP  YES, SEND ADDRESS
0A          1442  RTS  RETURN FROM COMMAND
0A          1443
0A          1444  * PRINT CURRENT ADDRESS
0A          1445  PRTADR  LDX  ADDR  LOAD POINTER VALUE
0A          1446  PSHS  X  SAVE X ON STACK
0A          1447  LEAX  #,S  POINT TO IT FOR DISPLAY
0A          1448  SWI  DISPLAY POINTER IN HEX
DUNFIELD 6809 ASSEMBLER: ASSIST09  PAGE: 23

FE28 05  1449  FCB  OUT4HS  FUNCTION
FE29 35 90  1450  PULS  PC,X  RECOVER POINTER AND RETURN
FE2B  1451
FE2B  1452  * UPDATE BYTE
FE2B  9E 9E  1453  MUPDAT  LDX  ADDR  LOAD NEXT BYTE POINTER
FE2D  05  1454  STB  ,X+  STORE AND INCREMENT X
FE2F  E1 1F  1455  CMFB  -1,X  ? SUCCESSFUL STORE
FE31  26 03  1456  BNE  MUPBAD  BRANCH FOR '!' IF NOT
FE33  9F 9E  1457  STX  ADDR  STORE NEW POINTER VALUE
FE35  39  1458  RTS  BACK TO CALLER
FE36  34 02  1459  MUPBAD  PSHS  A  SAVE A REGISTER
FE38  86 3F  1460  LDA  #$?  SHOW INVALID
FE3A  3F  1461  SWI  SEND OUT
FE3B  01  1462  FCB  OUTCH  FUNCTION
FE3C  35 82  1463  PULS  PC,A  RETURN TO CALLER
FE3E  1464
FE3E  1465  ********************WINDOW - SET WINDOW VALUE
FE3E  8D 20  1466  CWINDO  BSR  CDNUM  OBTAIN WINDOW VALUE
FE40  DD A0  1467  STD  WINDOW  STORE IT IN
FE42  39  1468  RTS  END COMMAND
FE43  1469
FE43  1470  ********************DISPLAY - HIGH SPEED DISPLAY MEMORY
FE43  8D 1B  1471  CDISP  BSR  CDNUM  OBTAIN ADDRESS
FE45  C4 F0  1472  ANDB  #$F0  FORCE TO 16 BOUNDARY
FE47  1F 02  1473  TFR  D,Y  SAVE IN Y
FE49  30 2F  1474  LEAX  15,Y  DEFAULT LENGTH
FE4B  25 04  1475  BCS  CDISPS  BRANCH IF END OF INPUT
FE4D  8D 11  1476  BSR  CDNUM  OBTAIN COUNT
FE4F  30 AB  1477  LEAX  D,Y  ASSUME COUNT, COMPUTE END ADDR
FE51  34 30  1478  CDISPS  PSHS  Y,X  SETUP PARAMETERS FOR HSDATA
FE53  10 A3 62  1479  CMPD  2,S  ? WAS IT COUNT
FE55  23 02  1480  BLS  CDCNT  BRANCH YES
FE57  ED E4  1481  STD  ,S  STORE HIGH ADDRESS
FE58  AD 9D E1 84  1482  CDCNT  JSR  [VECTAB+.HSDTA,PCR]  CALL PRINT ROUTINE
FE5E  35 E0  1483  PULS  PC,U,Y  CLEAN STACK AND END COMMAND
FE60  1484
FE60  1485  * OBTAIN NUMBER - ABORT IF NONE
FE60  1486  * ONLY DELIMITERS OF CR, BLANK, OR '/' ARE ACCEPTED
FE60  1487  * OUTPUT: D=VALUE, C=1 IF CARRIAGE RETURN DELIMITER,
FE60  1488  * ELSE C=0
FE60  17 7E  1489  CDNUM  LBSR  BLNUM  OBTAIN NUMBER
FE63  26 09  1490  BNE  CDBADN  BRANCH IF INVALID
FE65  81 2F  1491  CMPA  #'/'  ? VALID DELIMITER
FE67  22 05  1492  BHI  CDBADN  BRANCH IF NOT FOR ERROR
FE69  81 0E  1493  CMPA  #CR+1  LEAVE COMPARE FOR CARRIAGE RET
FE6B  DC 9B  1494  LDD  NUMBER  LOAD NUMBER
FE6D  9B  1495  RTS  RETURN WITH COMPARE
FE6E  16 9B  1496  CDBADN  LBRA  CMDBAD  RETURN TO ERROR MECHANISM
FE71  1497
FE71  1498  **********************PUNCH - PUNCH MEMORY IN S1-S9 FORMAT
FE71  8D ED  1499  CPUNCH  BSR  CDNUM  OBTAIN START ADDRESS
FE73  1F 02  1500  TFR  D,Y  SAVE IN Y
FE75  8D E9  1501  BSR  CDNUM  OBTAIN END ADDRESS
FE77  6F E2  1502  CLR  ,S  SETUP PUNCH FUNCTION CODE
FE79  34 26  1503  PSHS  Y,D  STORE VALUES ON STACK
FE7B  AD 9D E1 65  1504  CICALBS  JSR  [VECTAB+.BSON,PCR]  INITIALIZE HANDLER
FE7F  AD 9D E1 63  1505  JSR  [VECTAB+.BSDTA,PCR]  PERFORM FUNCTION
FE83  34 01  1506  PSHS  CC  SAVE RETURN CODE
FE85  AD 9D E1 5F  1507  JSR  [VECTAB+.BSOFF,PCR]  TURN OFF HANDLER
FE89  35 01  1508  PULS  CC  OBTAIN CONDITION CODE SAVED
FE8B  26 E1  1509  BNE  CDBADN  BRANCH IF ERROR
FE8D  35 B2  1510  PULS  PC,Y,X,A  RETURN FROM COMMAND
FE8F  1511
FE8F  1512  *******************LOAD - LOAD MEMORY FROM S1-S9 FORMAT
FE8F  8D 01  1513  CLOAD  BSR  CLVOFS  CALL SETUP AND PASS CODE
FE91  01  1514  FCB  1  LOAD FUNCTION CODE FOR PACKET
FE92  1515
FE92  33 F1  1516  CLVOFS  LEAU  [,S++]  LOAD CODE IN HIGH BYTE OF U
DUNFIELD 6809 ASSEMBLER: ASSIST09                                     PAGE: 24

FE94  33 D4              1517  LEAU  [,U] NOT CHANGING CC AND RESTORE S
FE96  27 03              1518  BEQ  CLVDFT BRANCH IF CARRIAGE RETURN NEXT
FE98  BD C6              1519  BSR  CDNUM OBTAIN OFFSETS
FE9A  BC                 1520  FCB  SKIP2 SKIP DEFAULT OFFSET
FE9B  4F                 1521  CLVDFT CLR A CREATE ZERO OFFSET
FE9C  5F                 1522  CLRB AS DEFAULT
FE9D  34 4E              1523  PSHS  U,DP,D SETUP CODE, NULL WORD, OFFSET
FE9F  20 DA              1524  BRA  CCALBS ENTER CALL TO BS ROUTINES
FEA1                          1525
FEA1                          1526  ******************VERIFY - COMPARE MEMORY WITH FILES
FEA1  8D EF              1527  CVER  BSR  CLVOFS COMPUTE OFFSET IF ANY
FEA3                          1528
FEA4                          1529
FEA4                          1530  ******************trace - trace instructions
FEA4                          1531
FEA4                          1532 ********************nulls - set new line and char padding
FEA4                          1533
FEA4                          1534
FEA4                          1535
FEA4                          1536
FEA4                          1537
FEA4                          1538
FEA4                          1539
FEA4                          1540
FEA4                          1541
FEA6                          1542
FEA6                          1543
FEA6                          1544
FEA6                          1545
FEA6                          1546
FEA6                          1547
FEA6                          1548
FEA6                          1549
FEA6                          1550
FEA6                          1551
FEA6                          1552
FEA6                          1553
FEA6                          1554
FEA6                          1555
FEA6                          1556
FEA6                          1557
FEA6                          1558
FEA6                          1559
FEA6                          1560
FEA6                          1561
FEA6                          1562
FEA6                          1563
FEA6                          1564
FEA6                          1565
FEA6                          1566
FEA6                          1567
FEA6                          1568
FEA6                          1569
FEA6                          1570
FEA6                          1571
FEA6                          1572
FEA6                          1573
FEA6                          1574
FEA6                          1575
FEA6                          1576
FEA6                          1577
FEA6                          1578
FEA6                          1579
FEA6                          1580
FEA6                          1581
FEA6                          1582
FEA6                          1583
FEA6                          1584
DUNFIELD 6809 ASSEMBLER: ASSIST09

PAGE: 26

FF5F 6D 84 1653 CENLP1 TST ,X ? END OF TABLE

FF61 2B D2 1654 BM1 CBKERR BRANCH ERROR IF SO

FF63 A1 81 1655 CMPA ,X++ ? THIS THE CHARACTER

FF65 26 F8 1656 BNE CENLP1 BRANCH IF NOT

FF67 EB 1F 1657 ADDB -1,X ADD THIS VALUE

FF69 20 EE 1658 BRA CENGET GET NEXT INPUT

FF6B 30 8C 49 1659 CEND1 LEAX <CONV2,PCR POINT AT TABLE 2

FF6E 1F 98 1660 TFR B,A SAVE COPY IN A

FF70 84 60 1661 ANDA #$560 ISOLATE REGISTER MASK

FF72 AA E4 1662 ORA ,S ADD IN INDICTION BIT

FF74 A7 E4 1663 STA ,S SAVE BACK AS POSTBYTE SKELETON

FF76 C4 9F 1664 ANDB #$9F CLEAR REGISTER BITS

FF78 6D 84 1665 CENLP2 TST ,X ? END OF TABLE

FF7A 27 B9 1666 BEQ CBKERR BRANCH ERROR IF SO

FF7C E1 81 1667 CMPB ,X++ ? SAME VALUE

FF7E 26 F8 1668 BNE CENLP2 LOOP IF NOT

FF80 B6 1F 1669 LDB -1,X LOAD RESULT VALUE

FF82 1700 ORB ,S ADD TO BASE SKELETON

FF84 E7 E4 1671 STB ,S SAVE POSTBYTE ON STACK

FF86 30 E4 1672 LEAX ,S POINT TO IT

FF88 3F 1673 SWI SEND OUT AS HEX

FF89 04 1674 FCB OUT2HS FUNCTION

FF8A 3F 1675 SWI TO NEXT LINE

FF8B 06 1676 FCB PCRLF FUNCTION

FF8C 35 84 1677 PULS PC,B END OF COMMAND

FF8E 1678

FF8E 41 04 42 05 44 06 + 1679 * TABLE ONE DEFINES VALID INPUT IN SEQUENCE

FF9E 48 01 48 01 48 00 + 1680 CONV1 FCX 'A',$04,'B',$05,'D',S','06,'$,H',S01

FF9E 2D 09 2D 01 53 70 + 1682 FCX '-',S09,'-',S01,'S','70','Y',S30

FFA6 55 50 58 10 2B 07 + 1683 FCX 'U',S50,'X',S10,'+',S07,'+',S01

FFAE 50 80 43 00 52 00 + 1684 FCX 'P',S80,'C',S00,'R',S00,'R',S00

FFB3 0F 1685 FCX $FF END OF TABLE

FFB7 10 84 11 00 1688 CONV2 FDB $1084,$1100 R,R R,R

FFB8 12 8E 13 89 1689 FDB $1288,$1309 HH,R HHHH,R

FFBF 14 86 15 85 1690 FDB $1486,$1585 A,R R+,R

FFC3 16 8B 17 80 1691 FDB $168B,$1780 D,R R+,R

FFC7 18 81 19 82 1692 FDB $1881,$1982 R++,R R

FFCB 1A 83 82 8C 1693 FDB $1A83,$828C ,R++ R HHHH,PCR

FFCD 83 8D 03 9F 1694 FDB $838D,$039F HHHH,PCR HHHH

FFD2 00 1695 FCB 0 END OF TABLE

FFD4 1696

FFD4 1697**************************************************************

FFD4 6E 9D DF EE 1700 Rsrvd jmp [vectab+.rsvid,pcr] RESERVED VECTOR

FFD8 6E 9D DF EC 1701 Swi3 jmp [vectab+.swi3,pcr] SWI3 VECTOR

FFDC 6E 9D DF EA 1702 Swi2 jmp [vectab+.swi2,pcr] SWI2 VECTOR

FFE0 6E 9D DF EB 1703 Fire jmp [vectab+.fire,pcr] FIRE VECTOR

FFE4 6E 9D DF E6 1704 Irq jmp [vectab+.irq,pcr] IRQ VECTOR

FFE8 6E 9D DF E4 1705 Swi jmp [vectab+.swi,pcr] SWI VECTOR

FFEC 6E 9D DF E2 1706 Nmi jmp [vectab+.nmi,pcr] NMI VECTOR

FFF0 1707

FFF0 1708**************************************************************************

FFF0 1709* ASSIST09 HARDWARE VECTOR TABLE

FFF0 1710* THIS TABLE IS USED IF THE ASSIST09 ROM ADDRESSES

FFF0 1711* THE MC6809 HARDWARE VECTORS.

FFF0 1712**************************************************************************

FFF0 1713 ORG ROMSEG+ROMSIZ-16 SETUP HARDWARE VECTORS

FFF2 1714 FDB Rsrvd RESERVED SLOT

FFF4 1715 FDB Swi1 SOFTWARE INTERRUPT 3

FFF4 1715 FDB Swi2 SOFTWARE INTERRUPT 2

FFF6 1717 FDB Fire FAST INTERRUPT REQUEST

FFF8 1718 FDB Irq INTERRUPT REQUEST

FFFA 1719 FDB Swi SOFTWARE INTERRUPT

FFFC 1720 FDB Nmi NON-MASKABLE INTERRUPT
<table>
<thead>
<tr>
<th>FFDE F8 37</th>
<th>1721</th>
<th>FDB</th>
<th>RESET</th>
<th>RESTART</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1722</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYMBOL</td>
<td>ADDRESS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ACIA</td>
<td>-002E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.AVTBL</td>
<td>-0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDTA</td>
<td>-0024</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-0026</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-0026</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSON</td>
<td>-0022</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.CIDTA</td>
<td>-0016</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.CIOFF</td>
<td>-0018</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.CION</td>
<td>-0014</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.CMDL1</td>
<td>-0002</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.CMDL2</td>
<td>-002C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.CODTA</td>
<td>-001C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.COOFF</td>
<td>-001E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.COON</td>
<td>-001A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ECHO</td>
<td>-0032</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.EXPAN</td>
<td>-002A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.FIRQ</td>
<td>-000A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.HSDTA</td>
<td>-0020</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.IRQ</td>
<td>-000C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.NMI</td>
<td>-0010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.PAD</td>
<td>-0030</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.PAUSE</td>
<td>-0028</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.PTM</td>
<td>-0034</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.RESET</td>
<td>-0012</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.RSVD</td>
<td>-0004</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.SWI</td>
<td>-000E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.SW12</td>
<td>-0008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.SW13</td>
<td>-0006</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ACIA</td>
<td>-0008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ADDR</td>
<td>-DF9E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBK2</td>
<td>-FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMB2</td>
<td>FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBLP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FDAC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMNSW</td>
<td>-FD9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BASEPG</td>
<td>-DF9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BELL</td>
<td>-0007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTBL</td>
<td>-DFB2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTCT</td>
<td>-DFFA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTOP</td>
<td>-DFA2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BLD2</td>
<td>-F815</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BLDSRT</td>
<td>-FB6E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDEOL</td>
<td>-FB70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD1</td>
<td>-FB40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD2</td>
<td>-FB42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDTA</td>
<td>-FB38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSON</td>
<td>-FB1B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDEOL</td>
<td>-FB70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD1</td>
<td>-FB40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD2</td>
<td>-FB42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDTA</td>
<td>-FB38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ACIA</td>
<td>-E008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ADDR</td>
<td>-DF9E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBK2</td>
<td>-FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMB2</td>
<td>FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBLP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FDAC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMNSW</td>
<td>-FD9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BASEPG</td>
<td>-DF9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BELL</td>
<td>-0007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTBL</td>
<td>-DFB2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTCT</td>
<td>-DFFA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTCT</td>
<td>-DFFA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTOP</td>
<td>-DFA2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BLD2</td>
<td>-F815</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BLDSRT</td>
<td>-FB6E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDEOL</td>
<td>-FB70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD1</td>
<td>-FB40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD2</td>
<td>-FB42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDTA</td>
<td>-FB38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSON</td>
<td>-FB1B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDEOL</td>
<td>-FB70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD1</td>
<td>-FB40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD2</td>
<td>-FB42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDTA</td>
<td>-FB38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ACIA</td>
<td>-E008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ADDR</td>
<td>-DF9E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBK2</td>
<td>-FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMB2</td>
<td>FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBLP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FDAC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMNSW</td>
<td>-FD9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BASEPG</td>
<td>-DF9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BELL</td>
<td>-0007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTBL</td>
<td>-DFB2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTCT</td>
<td>-DFFA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTCT</td>
<td>-DFFA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTOP</td>
<td>-DFA2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BLD2</td>
<td>-F815</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BLDSRT</td>
<td>-FB6E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDEOL</td>
<td>-FB70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD1</td>
<td>-FB40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD2</td>
<td>-FB42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDTA</td>
<td>-FB38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ACIA</td>
<td>-E008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ADDR</td>
<td>-DF9E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBK2</td>
<td>-FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMB2</td>
<td>FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBLP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FDAC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMNSW</td>
<td>-FD9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BASEPG</td>
<td>-DF9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BELL</td>
<td>-0007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTBL</td>
<td>-DFB2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTCT</td>
<td>-DFFA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTCT</td>
<td>-DFFA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTOP</td>
<td>-DFA2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BLD2</td>
<td>-F815</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BLDSRT</td>
<td>-FB6E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDEOL</td>
<td>-FB70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD1</td>
<td>-FB40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD2</td>
<td>-FB42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDTA</td>
<td>-FB38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ACIA</td>
<td>-E008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ADDR</td>
<td>-DF9E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBK2</td>
<td>-FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMB2</td>
<td>FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBLP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FDAC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMNSW</td>
<td>-FD9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BASEPG</td>
<td>-DF9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BELL</td>
<td>-0007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTBL</td>
<td>-DFB2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTCT</td>
<td>-DFFA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTCT</td>
<td>-DFFA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTOP</td>
<td>-DFA2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BLD2</td>
<td>-F815</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BLDSRT</td>
<td>-FB6E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDEOL</td>
<td>-FB70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD1</td>
<td>-FB40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD2</td>
<td>-FB42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDTA</td>
<td>-FB38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ACIA</td>
<td>-E008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ADDR</td>
<td>-DF9E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBK2</td>
<td>-FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMB2</td>
<td>FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBLP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FDAC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMNSW</td>
<td>-FD9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BASEPG</td>
<td>-DF9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BELL</td>
<td>-0007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTBL</td>
<td>-DFB2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTCT</td>
<td>-DFFA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTCT</td>
<td>-DFFA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTOP</td>
<td>-DFA2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BLD2</td>
<td>-F815</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BLDSRT</td>
<td>-FB6E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDEOL</td>
<td>-FB70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD1</td>
<td>-FB40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD2</td>
<td>-FB42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDTA</td>
<td>-FB38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ACIA</td>
<td>-E008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ADDR</td>
<td>-DF9E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBK2</td>
<td>-FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMB2</td>
<td>FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBLP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FDAC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMNSW</td>
<td>-FD9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BASEPG</td>
<td>-DF9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BELL</td>
<td>-0007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTBL</td>
<td>-DFB2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTCT</td>
<td>-DFFA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTCT</td>
<td>-DFFA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTOP</td>
<td>-DFA2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BLD2</td>
<td>-F815</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BLDSRT</td>
<td>-FB6E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDEOL</td>
<td>-FB70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD1</td>
<td>-FB40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD2</td>
<td>-FB42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDTA</td>
<td>-FB38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ACIA</td>
<td>-E008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ADDR</td>
<td>-DF9E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBK2</td>
<td>-FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMB2</td>
<td>FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBLP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FDAC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMNSW</td>
<td>-FD9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BASEPG</td>
<td>-DF9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BELL</td>
<td>-0007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTBL</td>
<td>-DFB2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTCT</td>
<td>-DFFA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTCT</td>
<td>-DFFA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BKPTOP</td>
<td>-DFA2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BLD2</td>
<td>-F815</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BLDSRT</td>
<td>-FB6E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDEOL</td>
<td>-FB70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD1</td>
<td>-FB40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDLD2</td>
<td>-FB42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDTA</td>
<td>-FB38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>-FB27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ACIA</td>
<td>-E008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ADDR</td>
<td>-DF9E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBK2</td>
<td>-FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMLOP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMB2</td>
<td>FAD7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ARMBLP</td>
<td>-FD8E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symbol</td>
<td>Define</td>
<td>References</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ACIA</td>
<td>92</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.AVTBL</td>
<td>69</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSDTA</td>
<td>87</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSOFF</td>
<td>88</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.BSON</td>
<td>86</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.CIDTA</td>
<td>80</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.CIOFF</td>
<td>81</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.CION</td>
<td>79</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.CMDL1</td>
<td>70</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.CMDL2</td>
<td>91</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.CODTA</td>
<td>83</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.COOFF</td>
<td>84</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.COON</td>
<td>82</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ECHO</td>
<td>94</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.EXPAN</td>
<td>90</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.FIRQ</td>
<td>74</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.HSDTA</td>
<td>85</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.IRQ</td>
<td>75</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.NMI</td>
<td>77</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.PAD</td>
<td>93</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.PAUSE</td>
<td>89</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.PTM</td>
<td>95</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.RESET</td>
<td>78</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.RSVD</td>
<td>71</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.SWI</td>
<td>76</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.SWI2</td>
<td>73</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.SWI3</td>
<td>72</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACIA</td>
<td>21</td>
<td>253 822 834 850</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDR</td>
<td>130 1236 1388 1399 1418 1428 1434 1445 1453 1457</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARMBK2</td>
<td>1366 770 1354</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARMBLP</td>
<td>1353 1357</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARMLOP</td>
<td>1368 1374</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARMSW</td>
<td>1361 1359</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BASEPG</td>
<td>132 183 781</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BELL</td>
<td>33 779</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BKPTBL</td>
<td>124 1635</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BKPTCT</td>
<td>118 383 1367 1591 1604 1631 1636</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BKPTOP</td>
<td>126</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLD2</td>
<td>189 193</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLD3</td>
<td>195 198</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLDBAD</td>
<td>1285 1336</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLDEXE</td>
<td>1298 1247</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLDHXC</td>
<td>1299 418</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLDHXI</td>
<td>1296 1230</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLDNNB</td>
<td>1216 1161 1394</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLDNUM</td>
<td>1219 1283 1489 1585 1589</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLDRTN</td>
<td>204 202</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLDSHF</td>
<td>1304 1308</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLDVTR</td>
<td>180 215</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRKPT</td>
<td>63 1381</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSDCMP</td>
<td>941 939</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSDEOL</td>
<td>945 937</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSDLD1</td>
<td>916 919 946</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSDLD2</td>
<td>918 925</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSDNXT</td>
<td>936 942</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSDFUN</td>
<td>974 910</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSDSRT</td>
<td>943 923 947</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSDTA</td>
<td>908 247 1505</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSOFF</td>
<td>888 248 1507</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSOFLP</td>
<td>896 897</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSON2</td>
<td>877 246 1504</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSON3</td>
<td>881 879</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSPEOF</td>
<td>1030 1018</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSPGO</td>
<td>984 1017</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSRPRE</td>
<td>1006 1008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSFK</td>
<td>989 987</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSPSTR</td>
<td>1029 994</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSPUN2</td>
<td>1026 1000 1002 1003 1006</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSPUNC</td>
<td>1027 1014</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BYTE</td>
<td>950 927 930 932 936</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BYTHEX</td>
<td>962 950 953</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BYTRTS</td>
<td>960 965</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN</td>
<td>37 708 715 1335</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBKADD</td>
<td>1616 1586</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBKADL</td>
<td>1624 1627</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBKADT</td>
<td>1629 1625</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBKDEL</td>
<td>1594 1590</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBKDLM</td>
<td>1600 1603</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBKDLP</td>
<td>1595 1598</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBKDSL</td>
<td>1607 1611</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBKDSF</td>
<td>1605 1584 1632</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBKERR</td>
<td>1628 1588 1596 1618 1622 1654 1666</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX C
MACHINE CODE TO INSTRUCTION CROSS REFERENCE

C.1 INTRODUCTION

This appendix contains a cross reference between the machine code, represented in hexadecimal and the instruction and addressing mode that it represents. The number of MPU cycles and the number of program bytes is also given. Refer to Table C-1.
Table C-1. Machine Code to Instruction Cross Reference

<table>
<thead>
<tr>
<th>OP</th>
<th>Mnemonic</th>
<th>Mode</th>
<th>#</th>
<th>OP</th>
<th>Mnemonic</th>
<th>Mode</th>
<th>#</th>
<th>OP</th>
<th>Mnemonic</th>
<th>Mode</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>NEG</td>
<td>Direct</td>
<td>6</td>
<td>2</td>
<td>30</td>
<td>LEAX</td>
<td>Indexed</td>
<td>4</td>
<td>2</td>
<td>60</td>
<td>NEG</td>
</tr>
<tr>
<td>01</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>31</td>
<td>LEAY</td>
<td></td>
<td></td>
<td></td>
<td>61</td>
<td>*</td>
</tr>
<tr>
<td>02</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>32</td>
<td>LEAS</td>
<td></td>
<td></td>
<td></td>
<td>62</td>
<td>*</td>
</tr>
<tr>
<td>03</td>
<td>COM</td>
<td></td>
<td></td>
<td></td>
<td>33</td>
<td>LEAU</td>
<td>Indexed</td>
<td>4</td>
<td>2</td>
<td>83</td>
<td>COM</td>
</tr>
<tr>
<td>04</td>
<td>LSR</td>
<td></td>
<td></td>
<td></td>
<td>34</td>
<td>PSHS</td>
<td>Immed</td>
<td>5</td>
<td>2</td>
<td>64</td>
<td>LSR</td>
</tr>
<tr>
<td>05</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>35</td>
<td>PULS</td>
<td></td>
<td></td>
<td></td>
<td>65</td>
<td>*</td>
</tr>
<tr>
<td>06</td>
<td>ROR</td>
<td></td>
<td></td>
<td></td>
<td>36</td>
<td>PSHU</td>
<td></td>
<td></td>
<td></td>
<td>66</td>
<td>*</td>
</tr>
<tr>
<td>07</td>
<td>ASR</td>
<td></td>
<td></td>
<td></td>
<td>37</td>
<td>PULU</td>
<td>Immed</td>
<td>5</td>
<td>2</td>
<td>67</td>
<td>ASR</td>
</tr>
<tr>
<td>08</td>
<td>ASL, LSL</td>
<td></td>
<td></td>
<td></td>
<td>38</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>68</td>
<td>ASL, LSL</td>
</tr>
<tr>
<td>09</td>
<td>ROL</td>
<td></td>
<td></td>
<td></td>
<td>39</td>
<td>RTS</td>
<td></td>
<td></td>
<td></td>
<td>69</td>
<td>ROL</td>
</tr>
<tr>
<td>0A</td>
<td>DEC</td>
<td></td>
<td></td>
<td></td>
<td>3A</td>
<td>ABX</td>
<td></td>
<td></td>
<td></td>
<td>70</td>
<td>DEC</td>
</tr>
<tr>
<td>0B</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>3B</td>
<td>RTI</td>
<td></td>
<td></td>
<td></td>
<td>71</td>
<td>*</td>
</tr>
<tr>
<td>0C</td>
<td>INC</td>
<td></td>
<td></td>
<td></td>
<td>3C</td>
<td>CWAI</td>
<td></td>
<td></td>
<td></td>
<td>72</td>
<td>INC</td>
</tr>
<tr>
<td>0D</td>
<td>TST</td>
<td></td>
<td></td>
<td></td>
<td>3D</td>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
<td>73</td>
<td>TST</td>
</tr>
<tr>
<td>0E</td>
<td>JMP</td>
<td></td>
<td></td>
<td></td>
<td>3E</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>74</td>
<td>JMP</td>
</tr>
<tr>
<td>0F</td>
<td>CLR</td>
<td>Direct</td>
<td>6</td>
<td>2</td>
<td>3F</td>
<td>SWI</td>
<td>Inherent</td>
<td>1</td>
<td>6</td>
<td>75</td>
<td>CLR</td>
</tr>
<tr>
<td>10</td>
<td>Page 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inherent</td>
<td>2</td>
<td>1</td>
<td>76</td>
<td>NEG</td>
</tr>
<tr>
<td>11</td>
<td>Page 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inherent</td>
<td>2</td>
<td>1</td>
<td>77</td>
<td>NEG</td>
</tr>
<tr>
<td>12</td>
<td>NOP</td>
<td>Inherent</td>
<td>2</td>
<td>1</td>
<td>42</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>78</td>
<td>*</td>
</tr>
<tr>
<td>13</td>
<td>SYNC</td>
<td>Inherent</td>
<td>4</td>
<td>1</td>
<td>43</td>
<td>COMA</td>
<td></td>
<td></td>
<td></td>
<td>79</td>
<td>COM</td>
</tr>
<tr>
<td>14</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>44</td>
<td>LSRA</td>
<td></td>
<td></td>
<td></td>
<td>80</td>
<td>LSR</td>
</tr>
<tr>
<td>15</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>45</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>81</td>
<td>*</td>
</tr>
<tr>
<td>16</td>
<td>LBRA</td>
<td>Relative</td>
<td>5</td>
<td>3</td>
<td>46</td>
<td>RORA</td>
<td></td>
<td></td>
<td></td>
<td>82</td>
<td>ROR</td>
</tr>
<tr>
<td>17</td>
<td>LBSR</td>
<td>Relative</td>
<td>9</td>
<td>3</td>
<td>47</td>
<td>ASRA</td>
<td></td>
<td></td>
<td></td>
<td>83</td>
<td>ASR</td>
</tr>
<tr>
<td>18</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>48</td>
<td>ASLA, LSL</td>
<td></td>
<td></td>
<td></td>
<td>84</td>
<td>ASL, LSL</td>
</tr>
<tr>
<td>19</td>
<td>DAA</td>
<td>Inherent</td>
<td>2</td>
<td>1</td>
<td>49</td>
<td>ROLA</td>
<td></td>
<td></td>
<td></td>
<td>85</td>
<td>ROL</td>
</tr>
<tr>
<td>1A</td>
<td>ORCC</td>
<td>Immed</td>
<td>3</td>
<td>2</td>
<td>4A</td>
<td>DECA</td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>DEC</td>
</tr>
<tr>
<td>1B</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>4B</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>87</td>
<td>*</td>
</tr>
<tr>
<td>1C</td>
<td>ANDCC</td>
<td>Immed</td>
<td>3</td>
<td>2</td>
<td>4C</td>
<td>INCA</td>
<td></td>
<td></td>
<td></td>
<td>88</td>
<td>INC</td>
</tr>
<tr>
<td>1D</td>
<td>SEX</td>
<td>Inherent</td>
<td>2</td>
<td>1</td>
<td>4D</td>
<td>TSTA</td>
<td></td>
<td></td>
<td></td>
<td>89</td>
<td>TST</td>
</tr>
<tr>
<td>1E</td>
<td>EXG</td>
<td>Immed</td>
<td>8</td>
<td>2</td>
<td>4E</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>90</td>
<td>JMP</td>
</tr>
<tr>
<td>1F</td>
<td>TFR</td>
<td>Immed</td>
<td>6</td>
<td>2</td>
<td>4F</td>
<td>CLRA</td>
<td>Inherent</td>
<td>2</td>
<td>1</td>
<td>91</td>
<td>CLR</td>
</tr>
</tbody>
</table>

LEGEND:
- Number of MPU cycles (less possible push pull or indexed-mode cycles)
# Number of program bytes
* Denotes unused opcode
| OP | Mnm | Mode  | → | # | OP | Mnm | Mode  | → | # | OP | Mnm | Mode  | → | # |
|----|-----|-------|---|---|----|-----|-------|---|---|----|-----|-------|---|---|----|-----|-------|---|---|----|-----|-------|---|---|
| 90 | SUBA | Direct | 4 | 2 | 90 | SUBA | Direct | 4 | 2 | 90 | SUBA | Direct | 4 | 2 |
| 91 | CMPA | Indexed | 4+ | 2+ | 91 | CMPA | Indexed | 4+ | 2+ | 91 | CMPA | Indexed | 4+ | 2+ |
| 92 | SBCA | Direct | 4 | 2 | 92 | SBCA | Direct | 4 | 2 | 92 | SBCA | Direct | 4 | 2 |
| 93 | SUBD | Indexed | 4+ | 2+ | 93 | SUBD | Indexed | 4+ | 2+ | 93 | SUBD | Indexed | 4+ | 2+ |
| 94 | ANDA | Direct | 4 | 2 | 94 | ANDA | Direct | 4 | 2 | 94 | ANDA | Direct | 4 | 2 |
| 95 | BITA | Indexed | 4+ | 2+ | 95 | BITA | Indexed | 4+ | 2+ | 95 | BITA | Indexed | 4+ | 2+ |
| 96 | LDA  | Direct | 4 | 2 | 96 | LDA  | Direct | 4 | 2 | 96 | LDA  | Direct | 4 | 2 |
| 97 | STA  | Direct | 4 | 2 | 97 | STA  | Direct | 4 | 2 | 97 | STA  | Direct | 4 | 2 |
| 98 | EORA | Indexed | 4+ | 2+ | 98 | EORA | Indexed | 4+ | 2+ | 98 | EORA | Indexed | 4+ | 2+ |
| 99 | ADCCA| Indexed | 4+ | 2+ | 99 | ADCCA| Indexed | 4+ | 2+ | 99 | ADCCA| Indexed | 4+ | 2+ |
| 9A | ORA  | Direct | 4 | 2 | 9A | ORA  | Direct | 4 | 2 | 9A | ORA  | Direct | 4 | 2 |
| 9B | ADDA | Direct | 4 | 2 | 9B | ADDA | Direct | 4 | 2 | 9B | ADDA | Direct | 4 | 2 |
| 9C | CPMX | Direct | 4 | 2 | 9C | CPMX | Direct | 4 | 2 | 9C | CPMX | Direct | 4 | 2 |
| 9D | JSR  | Direct | 4 | 2 | 9D | JSR  | Direct | 4 | 2 | 9D | JSR  | Direct | 4 | 2 |
| 9E | LDSX | Indexed | 4+ | 2+ | 9E | LDSX | Indexed | 4+ | 2+ | 9E | LDSX | Indexed | 4+ | 2+ |
| 9F | STX  | Direct | 5 | 2 | 9F | STX  | Direct | 5 | 2 | 9F | STX  | Direct | 5 | 2 |
| A0 | SUBA | Indexed | 4+ | 2+ | A0 | SUBA | Indexed | 4+ | 2+ | A0 | SUBA | Indexed | 4+ | 2+ |
| A1 | CMPA | Indexed | 4+ | 2+ | A1 | CMPA | Indexed | 4+ | 2+ | A1 | CMPA | Indexed | 4+ | 2+ |
| A2 | SBCA | Indexed | 4+ | 2+ | A2 | SBCA | Indexed | 4+ | 2+ | A2 | SBCA | Indexed | 4+ | 2+ |
| A3 | SUBD | Indexed | 4+ | 2+ | A3 | SUBD | Indexed | 4+ | 2+ | A3 | SUBD | Indexed | 4+ | 2+ |
| A4 | ANDA | Indexed | 4+ | 2+ | A4 | ANDA | Indexed | 4+ | 2+ | A4 | ANDA | Indexed | 4+ | 2+ |
| A5 | BITA | Indexed | 4+ | 2+ | A5 | BITA | Indexed | 4+ | 2+ | A5 | BITA | Indexed | 4+ | 2+ |
| A6 | LDA  | Indexed | 4+ | 2+ | A6 | LDA  | Indexed | 4+ | 2+ | A6 | LDA  | Indexed | 4+ | 2+ |
| A7 | STA  | Indexed | 4+ | 2+ | A7 | STA  | Indexed | 4+ | 2+ | A7 | STA  | Indexed | 4+ | 2+ |
| A8 | EORA | Indexed | 4+ | 2+ | A8 | EORA | Indexed | 4+ | 2+ | A8 | EORA | Indexed | 4+ | 2+ |
| A9 | ADCCA| Indexed | 4+ | 2+ | A9 | ADCCA| Indexed | 4+ | 2+ | A9 | ADCCA| Indexed | 4+ | 2+ |
| AA | ORA  | Indexed | 4+ | 2+ | AA | ORA  | Indexed | 4+ | 2+ | AA | ORA  | Indexed | 4+ | 2+ |
| AB | ADDA | Indexed | 4+ | 2+ | AB | ADDA | Indexed | 4+ | 2+ | AB | ADDA | Indexed | 4+ | 2+ |
| AC | CPMX | Indexed | 4+ | 2+ | AC | CPMX | Indexed | 4+ | 2+ | AC | CPMX | Indexed | 4+ | 2+ |
| AD | JSR  | Indexed | 4+ | 2+ | AD | JSR  | Indexed | 4+ | 2+ | AD | JSR  | Indexed | 4+ | 2+ |
| AE | LDSX | Indexed | 4+ | 2+ | AE | LDSX | Indexed | 4+ | 2+ | AE | LDSX | Indexed | 4+ | 2+ |
| AF | STX  | Indexed | 4+ | 2+ | AF | STX  | Indexed | 4+ | 2+ | AF | STX  | Indexed | 4+ | 2+ |
| B0 | SUBA | Extended | 5 | 3 | B0 | SUBA | Extended | 5 | 3 | B0 | SUBA | Extended | 5 | 3 |
| B1 | CMPA | Extended | 5 | 3 | B1 | CMPA | Extended | 5 | 3 | B1 | CMPA | Extended | 5 | 3 |
| B2 | SBCA | Extended | 5 | 3 | B2 | SBCA | Extended | 5 | 3 | B2 | SBCA | Extended | 5 | 3 |
| B3 | SUBD | Extended | 5 | 3 | B3 | SUBD | Extended | 5 | 3 | B3 | SUBD | Extended | 5 | 3 |
| B4 | ANDA | Extended | 5 | 3 | B4 | ANDA | Extended | 5 | 3 | B4 | ANDA | Extended | 5 | 3 |
| B5 | BITA | Extended | 5 | 3 | B5 | BITA | Extended | 5 | 3 | B5 | BITA | Extended | 5 | 3 |
| B6 | LDA  | Extended | 5 | 3 | B6 | LDA  | Extended | 5 | 3 | B6 | LDA  | Extended | 5 | 3 |
| B7 | STA  | Extended | 5 | 3 | B7 | STA  | Extended | 5 | 3 | B7 | STA  | Extended | 5 | 3 |
| B8 | EORA | Extended | 5 | 3 | B8 | EORA | Extended | 5 | 3 | B8 | EORA | Extended | 5 | 3 |
| B9 | ADCCA| Extended | 5 | 3 | B9 | ADCCA| Extended | 5 | 3 | B9 | ADCCA| Extended | 5 | 3 |
| BA | ORA  | Extended | 5 | 3 | BA | ORA  | Extended | 5 | 3 | BA | ORA  | Extended | 5 | 3 |
| BB | ADDA | Extended | 5 | 3 | BB | ADDA | Extended | 5 | 3 | BB | ADDA | Extended | 5 | 3 |
| BC | CPMX | Extended | 5 | 3 | BC | CPMX | Extended | 5 | 3 | BC | CPMX | Extended | 5 | 3 |
| BD | JSR  | Extended | 5 | 3 | BD | JSR  | Extended | 5 | 3 | BD | JSR  | Extended | 5 | 3 |
| BE | LDSX | Extended | 5 | 3 | BE | LDSX | Extended | 5 | 3 | BE | LDSX | Extended | 5 | 3 |
| BF | STX  | Extended | 5 | 3 | BF | STX  | Extended | 5 | 3 | BF | STX  | Extended | 5 | 3 |

NOTE: All unused opcodes are both undefined and illegal.
# APPENDIX D
## PROGRAMMING AID

### D.1 INTRODUCTION

This appendix contains a compilation of data that will assist you in programming the M6809 processor. Refer to Table D-1.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Forms</th>
<th>Addressing Mode</th>
<th>Relative</th>
<th>Description</th>
<th>S</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCC</td>
<td>BCC</td>
<td>24 3 2</td>
<td>Branch C = 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBC</td>
<td>10 5/6 4</td>
<td>Long Branch C = 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCS</td>
<td>BCS</td>
<td>26 3 2</td>
<td>Branch C = 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBCS</td>
<td>10 5/6 4</td>
<td>Long Branch C = 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEQ</td>
<td>BEQ</td>
<td>27 3 2</td>
<td>Branch Z = 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBEQ</td>
<td>10 5/6 4</td>
<td>Long Branch Z = 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BGE</td>
<td>BGE</td>
<td>2C 3 2</td>
<td>Branch &amp; Zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBGE</td>
<td>10 5/6 4</td>
<td>Long Branch &amp; Zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BGT</td>
<td>BGT</td>
<td>2E 3 2</td>
<td>Branch &gt; Zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBGT</td>
<td>10 5/6 4</td>
<td>Long Branch &gt; Zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BHI</td>
<td>BHI</td>
<td>22 3 2</td>
<td>Branch, higher</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBHI</td>
<td>10 5/6 4</td>
<td>Long Branch higher</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BHS</td>
<td>BHS</td>
<td>24 3 2</td>
<td>Branch Higher</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBHS</td>
<td>10 5/6 4</td>
<td>Long Branch Higher</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLE</td>
<td>BLE</td>
<td>2F 3 2</td>
<td>Branch &amp; Zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBLE</td>
<td>10 5/6 4</td>
<td>Long Branch &amp; Zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLO</td>
<td>BLO</td>
<td>25 3 2</td>
<td>Branch Lower</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBLO</td>
<td>10 5/6 4</td>
<td>Long Branch Lower</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Forms</th>
<th>Addressing Mode</th>
<th>Relative</th>
<th>Description</th>
<th>S</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLS</td>
<td>BLS</td>
<td>23 3 2</td>
<td>Branch Lower or Same</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBS</td>
<td>10 5/6 4</td>
<td>Long Branch Lower or Same</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLT</td>
<td>BLT</td>
<td>20 3 2</td>
<td>Branch &lt; Zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBLT</td>
<td>10 5/6 4</td>
<td>Long Branch &lt; Zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BMI</td>
<td>BMI</td>
<td>28 3 2</td>
<td>Branch Minus</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBMI</td>
<td>10 5/6 4</td>
<td>Long Branch Minus</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNE</td>
<td>BNE</td>
<td>26 3 2</td>
<td>Branch Z ≠ 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBNE</td>
<td>10 5/6 4</td>
<td>Long Branch Z ≠ 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BPL</td>
<td>BPL</td>
<td>2A 3 2</td>
<td>Branch Plus</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBPL</td>
<td>10 5/6 4</td>
<td>Long Branch Plus</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRA</td>
<td>BRA</td>
<td>20 3 2</td>
<td>Branch Always</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBRA</td>
<td>16 5 3</td>
<td>Long Branch Always</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRN</td>
<td>BRN</td>
<td>21 3 2</td>
<td>Branch Never</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBHN</td>
<td>10 5 4</td>
<td>Long Branch Never</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSR</td>
<td>BSR</td>
<td>1D 7 2</td>
<td>Branch to Subroutine</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBSR</td>
<td>17 9 3</td>
<td>Long Branch to Subroutine</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BVC</td>
<td>BVC</td>
<td>28 3 2</td>
<td>Branch V = 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBVC</td>
<td>10 5/6 4</td>
<td>Long Branch V = 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BVS</td>
<td>BVS</td>
<td>29 3 2</td>
<td>Branch V = 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBVS</td>
<td>10 5/6 4</td>
<td>Long Branch V = 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table D-1. Programming Aid (Continued)

### SIMPLE BRANCHES

<table>
<thead>
<tr>
<th>OP</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BRA</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>LBRA</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>BRN</td>
<td>21</td>
<td>3</td>
</tr>
<tr>
<td>LBRN</td>
<td>1021</td>
<td>5</td>
</tr>
<tr>
<td>BSR</td>
<td>8D</td>
<td>7</td>
</tr>
<tr>
<td>LBSR</td>
<td>17</td>
<td>9</td>
</tr>
</tbody>
</table>

### SIMPLE CONDITIONAL BRANCHES (Notes 1-4)

<table>
<thead>
<tr>
<th>Test</th>
<th>True</th>
<th>OP</th>
<th>False</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>N = 1</td>
<td>BMI</td>
<td>2B</td>
<td>BPL</td>
<td>2A</td>
</tr>
<tr>
<td>Z = 1</td>
<td>BEQ</td>
<td>27</td>
<td>BNE</td>
<td>26</td>
</tr>
<tr>
<td>V = 1</td>
<td>BVS</td>
<td>29</td>
<td>BVC</td>
<td>28</td>
</tr>
<tr>
<td>C = 1</td>
<td>BCS</td>
<td>25</td>
<td>BCC</td>
<td>24</td>
</tr>
</tbody>
</table>

### SIGNED CONDITIONAL BRANCHES (Notes 1-4)

<table>
<thead>
<tr>
<th>Test</th>
<th>True</th>
<th>OP</th>
<th>False</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>r &gt; m</td>
<td>BGT</td>
<td>2E</td>
<td>BLE</td>
<td>2F</td>
</tr>
<tr>
<td>r ≥ m</td>
<td>BGE</td>
<td>2C</td>
<td>BLT</td>
<td>2D</td>
</tr>
<tr>
<td>r = m</td>
<td>BEQ</td>
<td>27</td>
<td>BNE</td>
<td>26</td>
</tr>
<tr>
<td>r ≤ m</td>
<td>BLE</td>
<td>2F</td>
<td>BGT</td>
<td>2E</td>
</tr>
<tr>
<td>r &lt; m</td>
<td>BLT</td>
<td>2D</td>
<td>BGE</td>
<td>2C</td>
</tr>
</tbody>
</table>

### UNSIGNED CONDITIONAL BRANCHES (Notes 1-4)

<table>
<thead>
<tr>
<th>Test</th>
<th>True</th>
<th>OP</th>
<th>False</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>r &gt; m</td>
<td>BHI</td>
<td>22</td>
<td>BLS</td>
<td>23</td>
</tr>
<tr>
<td>r ≥ m</td>
<td>BHS</td>
<td>24</td>
<td>BLO</td>
<td>25</td>
</tr>
<tr>
<td>r = m</td>
<td>BEQ</td>
<td>27</td>
<td>BNE</td>
<td>26</td>
</tr>
<tr>
<td>r ≤ m</td>
<td>BLS</td>
<td>23</td>
<td>BHI</td>
<td>22</td>
</tr>
<tr>
<td>r &lt; m</td>
<td>BLO</td>
<td>25</td>
<td>BHS</td>
<td>24</td>
</tr>
</tbody>
</table>

Notes:

1. All conditional branches have both short and long variations.
2. All short branches are 2 bytes and require 3 cycles.
3. All conditional long branches are formed by prefixing the short branch opcode with $10 and using a 16-bit destination offset.
4. All conditional long branches require 4 bytes and 6 cycles if the branch is taken or 5 cycles if the branch is not taken.
### Table D-1. Programming Aid (Continued)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Forms</th>
<th>Immediate</th>
<th>Direct</th>
<th>Indexed</th>
<th>Extended</th>
<th>Inherent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABX</td>
<td>3A 3</td>
<td>1</td>
<td>B + X − X (Unsigned)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC</td>
<td>A</td>
<td>3A 3</td>
<td>A + M − C − A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC</td>
<td>A</td>
<td>3A 3</td>
<td>A + M − C − B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>A</td>
<td>3A 3</td>
<td>A + M − A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>A</td>
<td>3A 3</td>
<td>A + M − B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>A</td>
<td>3A 3</td>
<td>A + M + 1 − D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>A</td>
<td>3A 3</td>
<td>A + M − A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>A</td>
<td>3A 3</td>
<td>A + M − B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASL</td>
<td>3A 3</td>
<td>1</td>
<td>0100 00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASL</td>
<td>3A 3</td>
<td>1</td>
<td>0100 00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASR</td>
<td>3A 3</td>
<td>1</td>
<td>0100 00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASR</td>
<td>3A 3</td>
<td>1</td>
<td>0100 00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIT</td>
<td>B</td>
<td>3A 3</td>
<td>Bit Test 1/M</td>
<td>BIT 1/M</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIT</td>
<td>B</td>
<td>3A 3</td>
<td>Bit Test 2/M</td>
<td>BIT 2/M</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR</td>
<td>3A 3</td>
<td>1</td>
<td>0100 00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMP</td>
<td>CMPA</td>
<td>3A 3</td>
<td>Compare M from A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMPB</td>
<td>CMPD</td>
<td>3A 3</td>
<td>Compare M from B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMPS</td>
<td>CMPS</td>
<td>3A 3</td>
<td>Compare M + 1 from D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMPU</td>
<td>CMPX</td>
<td>3A 3</td>
<td>Compare M + 1 from S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMPPU</td>
<td>CMPPX</td>
<td>3A 3</td>
<td>Compare M + 1 from T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COM</td>
<td>COMA</td>
<td>3A 3</td>
<td>Compare M + 1 from X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMB</td>
<td>COMB</td>
<td>3A 3</td>
<td>Compare M + 1 from Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CWA</td>
<td>CWA</td>
<td>3A 3</td>
<td>CC 1 IMM − CC Wait for Interrupt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAA</td>
<td>DAA</td>
<td>3A 3</td>
<td>Decimal Adjust A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC</td>
<td>DEC</td>
<td>3A 3</td>
<td>DEC 1 − A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECB</td>
<td>DECB</td>
<td>3A 3</td>
<td>DEC 1 − B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EOR</td>
<td>EOR</td>
<td>3A 3</td>
<td>EOR M + A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXG</td>
<td>EXG</td>
<td>3A 3</td>
<td>EXG 1 − R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC</td>
<td>INC</td>
<td>3A 3</td>
<td>INC 1 − A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCB</td>
<td>INCB</td>
<td>3A 3</td>
<td>INC 1 − B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>JMP</td>
<td>3A 3</td>
<td>JMP 1 − M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSR</td>
<td>JSR</td>
<td>3A 3</td>
<td>JSR 1 − M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>LDA</td>
<td>3A 3</td>
<td>LD 1 − A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDB</td>
<td>LDB</td>
<td>3A 3</td>
<td>LD 1 − B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDD</td>
<td>LDD</td>
<td>3A 3</td>
<td>LD 1 − M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDS</td>
<td>LDS</td>
<td>3A 3</td>
<td>LD 1 − S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDU</td>
<td>LDU</td>
<td>3A 3</td>
<td>LD 1 − U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDL</td>
<td>LDL</td>
<td>3A 3</td>
<td>LD 1 − Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEA</td>
<td>LEAS</td>
<td>3A 3</td>
<td>LEA 1 − S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEAX</td>
<td>LEAX</td>
<td>3A 3</td>
<td>LEA 1 − U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEAY</td>
<td>LEAY</td>
<td>3A 3</td>
<td>LEA 1 − X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3A 3</td>
<td>LEA 1 − Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Legend:**
- **OP**: Operation Code (Hexadecimal)
- **−**: Number of MPU Cycles
- **#**: Number of Program Bytes
- **+**: Arithmetic Plus
- **−**: Arithmetic Minus
- **•**: Multiply
- **M**: Complement of M
- **i**: Test and set if true, cleared otherwise
- **H**: Not Affected
- **N**: Negative (sign bit)
- **Z**: Zero (Reset)
- **V**: Overflow, 2's complement
- **C**: Carry from ALU

**Diagrams:**
- 0: Unaffected
- 1: Affected
- 2: Negative
- 3: Zero
- 4: Carry
- 5: Overflow
- 6: Carry
- 7: Negative
- 8: Zero
- 9: Carry
- 10: Overflow
- 11: Carry
- 12: Overflow
- 13: Carry
- 14: Overflow
- 15: Carry

**Condition Codes:**
- **Z**: Zero
- **N**: Negative
- **V**: Overflow
- **C**: Carry
- **H**: Half-Carry
- **A**: Carry

**Logical Operations:**
- **AND**: Logical AND
- **OR**: Logical OR
- **XOR**: Logical XOR
- **NOT**: Logical NOT

**Arithmetic Operations:**
- **ADD**: Add
- **ADC**: Add with Carry
- **SUB**: Subtract
- **SBB**: Subtract with Borrow
- **MUL**: Multiply
- **DIV**: Divide
- **SAD**: Subtract Add
- **SBB**: Subtract Subtract
- **SHR**: Shift Right
- **SLL**: Shift Left
- **RAR**: Rotate Right
- **RLL**: Rotate Left

**Other Operations:**
- **JMP**: Jump
- **JSR**: Jump and Return
- **BFR**: Branch on Flag
- **BLR**: Branch and Return on Flag
- **CLI**: Clear Interrupts
- **SEI**: Set Interrupts

**Miscellaneous:**
- **LD**: Load
- **ST**: Store
- **PIN**: Push
- **POP**: Pop
- **PUSH**: Push
- **POP**: Pop
- **DOW**: Dwell
- **UPR**: UPR
- **TST**: Test
- **RTS**: Return from System Call
### Table D-1. Programming Aid (Continued)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Forms</th>
<th>Immediate</th>
<th>Direct</th>
<th>Indexed&lt;sup&gt;1&lt;/sup&gt;</th>
<th>Extended</th>
<th>Inherent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LSL</strong></td>
<td>LSLA</td>
<td>08 6 2</td>
<td>68 6+ 2+</td>
<td>78 7 3</td>
<td>A B c</td>
<td>M b7 d0 0</td>
<td><img src="image1.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>LSLB</td>
<td>58 6 2</td>
<td>68 6+ 2+</td>
<td>78 7 3</td>
<td>A B c</td>
<td>M b7 d0 0</td>
<td><img src="image2.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>LSL</td>
<td>58 6 2</td>
<td>68 6+ 2+</td>
<td>78 7 3</td>
<td>A B c</td>
<td>M b7 d0 0</td>
<td><img src="image3.png" alt="Image" /></td>
</tr>
<tr>
<td><strong>LSR</strong></td>
<td>LSRAD</td>
<td>04 6 2</td>
<td>64 6+ 2+</td>
<td>74 7 3</td>
<td>A B c</td>
<td>M b7 d0 0</td>
<td><img src="image4.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>LSRB</td>
<td>54 6 2</td>
<td>64 6+ 2+</td>
<td>74 7 3</td>
<td>A B c</td>
<td>M b7 d0 0</td>
<td><img src="image5.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>LSR</td>
<td>54 6 2</td>
<td>64 6+ 2+</td>
<td>74 7 3</td>
<td>A B c</td>
<td>M b7 d0 0</td>
<td><img src="image6.png" alt="Image" /></td>
</tr>
<tr>
<td><strong>MUL</strong></td>
<td>NEGA</td>
<td>40 6 2</td>
<td>60 6+ 2+</td>
<td>70 7 3</td>
<td>A B+1 A</td>
<td>B+1 B</td>
<td><img src="image7.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>NEGB</td>
<td>50 6 2</td>
<td>60 6+ 2+</td>
<td>70 7 3</td>
<td>A B+1 A</td>
<td>B+1 B</td>
<td><img src="image8.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>NEG</td>
<td>00 6 2</td>
<td>60 6+ 2+</td>
<td>70 7 3</td>
<td>A B+1 A</td>
<td>B+1 B</td>
<td><img src="image9.png" alt="Image" /></td>
</tr>
<tr>
<td><strong>NOR</strong></td>
<td>ORA</td>
<td>12 2 2</td>
<td>9A 4 2</td>
<td>AA 4+ 2+</td>
<td>BA 5 3</td>
<td>A V M - A</td>
<td><img src="image10.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>ORB</td>
<td>3A 4 2</td>
<td>EA 4+ 2+</td>
<td>FA 5 3</td>
<td>A V M - B</td>
<td><img src="image11.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ORCC</td>
<td>1A 3 2</td>
<td></td>
<td></td>
<td></td>
<td>C V M - CC</td>
<td><img src="image12.png" alt="Image" /></td>
</tr>
<tr>
<td><strong>PSH</strong></td>
<td>PSHS</td>
<td>3A 5+1 2</td>
<td></td>
<td></td>
<td></td>
<td>Push Registers on Stack</td>
<td><img src="image13.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>PSHU</td>
<td>36 5+1 2</td>
<td></td>
<td></td>
<td></td>
<td>Push Registers on U Stack</td>
<td><img src="image14.png" alt="Image" /></td>
</tr>
<tr>
<td><strong>PUL</strong></td>
<td>PULS</td>
<td>35 5+1 2</td>
<td></td>
<td></td>
<td></td>
<td>Pull Registers from Stack</td>
<td><img src="image15.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>PULU</td>
<td>37 5+1 2</td>
<td></td>
<td></td>
<td></td>
<td>Pull Registers from U Stack</td>
<td><img src="image16.png" alt="Image" /></td>
</tr>
<tr>
<td><strong>ROL</strong></td>
<td>ROL</td>
<td>06 6 2</td>
<td>69 6+ 2+</td>
<td>79 7 3</td>
<td>A B c</td>
<td>M b7 d0 0</td>
<td><img src="image17.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>ROLB</td>
<td>09 6 2</td>
<td>69 6+ 2+</td>
<td>79 7 3</td>
<td>A B c</td>
<td>M b7 d0 0</td>
<td><img src="image18.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>ROLR</td>
<td>43 6 2</td>
<td>69 6+ 2+</td>
<td>79 7 3</td>
<td>A B c</td>
<td>M b7 d0 0</td>
<td><img src="image19.png" alt="Image" /></td>
</tr>
<tr>
<td><strong>ROR</strong></td>
<td>RORA</td>
<td>46 6 2</td>
<td>66 6+ 2+</td>
<td>76 7 3</td>
<td>A B c</td>
<td>M b7 d0 0</td>
<td><img src="image20.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>RORB</td>
<td>56 6 2</td>
<td>66 6+ 2+</td>
<td>76 7 3</td>
<td>A B c</td>
<td>M b7 d0 0</td>
<td><img src="image21.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>RORR</td>
<td>06 6 2</td>
<td>66 6+ 2+</td>
<td>76 7 3</td>
<td>A B c</td>
<td>M b7 d0 0</td>
<td><img src="image22.png" alt="Image" /></td>
</tr>
<tr>
<td><strong>RTI</strong></td>
<td>RTI</td>
<td>38 6 1B</td>
<td>1 Return From Interrupt</td>
<td><img src="image23.png" alt="Image" /></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RTS</strong></td>
<td>RTS</td>
<td>39 5 1</td>
<td>1 Return From Subroutine</td>
<td><img src="image24.png" alt="Image" /></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SBC</strong></td>
<td>SBCA</td>
<td>82 2 2</td>
<td>92 4 2</td>
<td>A2 4+ 2+</td>
<td>B2 5 3</td>
<td>A-M-C-A</td>
<td><img src="image25.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>SBCB</td>
<td>82 2 2</td>
<td>92 4 2</td>
<td>A2 4+ 2+</td>
<td>B2 5 3</td>
<td>B-M-C-B</td>
<td><img src="image26.png" alt="Image" /></td>
</tr>
<tr>
<td><strong>SEX</strong></td>
<td>SEX</td>
<td>1D 2 2</td>
<td>1 Sign Extend B into A</td>
<td><img src="image27.png" alt="Image" /></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ST</strong></td>
<td>STA</td>
<td>97 4 2</td>
<td>A7 4+ 2+</td>
<td>B7 5 3</td>
<td>A-M</td>
<td><img src="image28.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STB</td>
<td>97 4 2</td>
<td>A7 4+ 2+</td>
<td>B7 5 3</td>
<td>B-M</td>
<td><img src="image29.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STD</td>
<td>10 6 3</td>
<td>10 6+ 3+</td>
<td>10 7 4</td>
<td>D-M M+1</td>
<td><img src="image30.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STS</td>
<td>10 6 3</td>
<td>10 6+ 3+</td>
<td>10 7 4</td>
<td>S-M M+1</td>
<td><img src="image31.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><strong>STU</strong></td>
<td>STU</td>
<td>10 6 3</td>
<td>10 6+ 3+</td>
<td>10 7 4</td>
<td>U-M M+1</td>
<td><img src="image32.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STX</td>
<td>9F 5 2</td>
<td>AF 5+ 2+</td>
<td>BF 6 3</td>
<td>X-M M+1</td>
<td><img src="image33.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STY</td>
<td>9F 5 2</td>
<td>AF 5+ 2+</td>
<td>BF 6 3</td>
<td>Y-M M+1</td>
<td><img src="image34.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td><strong>SUB</strong></td>
<td>SUBA</td>
<td>80 2 2</td>
<td>90 4 2</td>
<td>A0 4+ 2+</td>
<td>B0 5 3</td>
<td>A-M-A</td>
<td><img src="image35.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>SUBB</td>
<td>80 2 2</td>
<td>90 4 2</td>
<td>A0 4+ 2+</td>
<td>B0 5 3</td>
<td>B-M-B</td>
<td><img src="image36.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>SUBD</td>
<td>83 4 3</td>
<td>93 6 2</td>
<td>A3 6+ 2+</td>
<td>B3 7 3</td>
<td>D-M M+1-D</td>
<td><img src="image37.png" alt="Image" /></td>
</tr>
<tr>
<td><strong>SWI</strong></td>
<td>SWI&lt;sup&gt;1&lt;/sup&gt;</td>
<td>3F 19 1</td>
<td>Software Interrupt 1</td>
<td><img src="image38.png" alt="Image" /></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SWI&lt;sup&gt;2&lt;/sup&gt;</td>
<td>10 20 2</td>
<td>Software Interrupt 2</td>
<td><img src="image39.png" alt="Image" /></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SWI&lt;sup&gt;3&lt;/sup&gt;</td>
<td>11 20 1</td>
<td>Software Interrupt 3</td>
<td><img src="image40.png" alt="Image" /></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SYNC</strong></td>
<td>SYNC</td>
<td>13 2 4</td>
<td>1 Synchronize to Interrupt</td>
<td><img src="image41.png" alt="Image" /></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TFR</strong></td>
<td>TFR</td>
<td>1F 6 2</td>
<td>1 Set Carry</td>
<td><img src="image42.png" alt="Image" /></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TST</strong></td>
<td>TSTA</td>
<td>4D 2 1</td>
<td>1 Test A</td>
<td><img src="image43.png" alt="Image" /></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TSTB</td>
<td>5D 2 1</td>
<td>1 Test B</td>
<td><img src="image44.png" alt="Image" /></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TST</td>
<td>0D 6 2</td>
<td>6D 6+ 2+</td>
<td>7D 7 3</td>
<td>Test M</td>
<td><img src="image45.png" alt="Image" /></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
1. This column gives a base cycle and byte count. To obtain total count, add the values obtained from the INDEXED ADDRESSING MODE table, in Appendix F.
2. R1 and R2 may be any pair of 8 bit or any pair of 16 bit registers.
   The 8 bit registers are: A, B, CC, DP
   The 16 bit registers are: X, Y, U, S, D, PC
3. EA is the effective address.
4. The PSH and PUL instructions require 5 cycles plus 1 cycle for each byte pushed or pulled.
5. 5(6) means: 5 cycles if branch taken, 6 cycles if taken (Branch instructions).
6. SWI sets I and F bits. SWI2 and SWI3 do not affect I and F.
7. Conditions Codes set as a direct result of the instruction.
8. Value of half-carry flag is undefined.
9. Special Case — Carry set if b7 is SET.

---

D-4
APPENDIX E  
ASCII CHARACTER SET

E.1 INTRODUCTION

This appendix contains the standard 112 character ASCII character set (7-bit code).

E.2 CHARACTER REPRESENTATION AND CODE IDENTIFICATION

The ASCII character set is given in Figure E-1.

![Figure E-1. ASCII Character Set](image-url)
Each 7-bit character is represented with bit seven as the high-order bit and bit one as the low-order bit as shown in the following example:

\[
\begin{array}{cccccccc}
  b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\
  1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]

The bit representation for the character “A” is developed from the bit pattern for bits seven through five found above the column designated 4 and the bit pattern for bits four through one found to the left of the row designated 1.

A hexadecimal notation is commonly used to indicate the code for each character. This is easily developed by assuming a logic zero in the non-existant bit eight position for the column numbers and using the hexadecimal number for the row numbers.

**E.3 CONTROL CHARACTERS**

The characters located in columns zero and one of Figure E-1 are considered control characters. By definition, these are characters whose occurrence in a particular context initiates, modifies, or stops an action that affects the recording, processing, transmission, or interpretation of data. Table E-1 provides the meanings of the control characters.

**Table E-1. Control Characters**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Mnemonic</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUL</td>
<td>Null</td>
<td>DLE</td>
<td>Data Link Escape</td>
</tr>
<tr>
<td>SOH</td>
<td>Start of Heading</td>
<td>DC1</td>
<td>Device Control 1</td>
</tr>
<tr>
<td>STX</td>
<td>Start of Text</td>
<td>DC2</td>
<td>Device Control 2</td>
</tr>
<tr>
<td>ETX</td>
<td>End of Text</td>
<td>DC3</td>
<td>Device Control 3</td>
</tr>
<tr>
<td>EOT</td>
<td>End of Transmission</td>
<td>DC4</td>
<td>Device Control 4</td>
</tr>
<tr>
<td>ENQ</td>
<td>Enquiry</td>
<td>NAK</td>
<td>Negative Acknowledge</td>
</tr>
<tr>
<td>ACK</td>
<td>Acknowledge</td>
<td>SYN</td>
<td>Synchronous Idle</td>
</tr>
<tr>
<td>BEL</td>
<td>Bell</td>
<td>ETB</td>
<td>End of Transmission Block</td>
</tr>
<tr>
<td>BS</td>
<td>Backspace</td>
<td>CAN</td>
<td>Cancel</td>
</tr>
<tr>
<td>HT</td>
<td>Horizontal Tabulation</td>
<td>EM</td>
<td>End of Medium</td>
</tr>
<tr>
<td>LF</td>
<td>Line Feed</td>
<td>SUB</td>
<td>Substitute</td>
</tr>
<tr>
<td>VT</td>
<td>Vertical Tabulation</td>
<td>ESC</td>
<td>Escape</td>
</tr>
<tr>
<td>FF</td>
<td>Form Feed</td>
<td>FS</td>
<td>File Separator</td>
</tr>
<tr>
<td>CR</td>
<td>Carriage Return</td>
<td>GS</td>
<td>Group Separator</td>
</tr>
<tr>
<td>SO</td>
<td>Shift Out</td>
<td>RS</td>
<td>Record Separator</td>
</tr>
<tr>
<td>SI</td>
<td>Shift In</td>
<td>US</td>
<td>Unit Separator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DEL</td>
<td>Delete</td>
</tr>
</tbody>
</table>

**E.4 GRAPHIC CHARACTERS**

The characters in columns two through seven are considered graphic characters. These characters have a visual representation which is normally displayed or printed. These characters and their names are given in Table E-2.
Table E-2. Graphic Characters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
<td>Space (Normally Nonprinting)</td>
</tr>
<tr>
<td>!</td>
<td>Exclamation Point</td>
</tr>
<tr>
<td>&quot;</td>
<td>Quotation Marks (Diaeresis)</td>
</tr>
<tr>
<td>#</td>
<td>Number Sign</td>
</tr>
<tr>
<td>$</td>
<td>Dollar Sign</td>
</tr>
<tr>
<td>%</td>
<td>Percent Sign</td>
</tr>
<tr>
<td>&amp;</td>
<td>Ampersand</td>
</tr>
<tr>
<td>'</td>
<td>Apostrophe (Closing Single Quotation Mark; Acute Accent)</td>
</tr>
<tr>
<td>(</td>
<td>Opening Parenthesis</td>
</tr>
<tr>
<td>)</td>
<td>Closing Parenthesis</td>
</tr>
<tr>
<td>*</td>
<td>Asterisk</td>
</tr>
<tr>
<td>+</td>
<td>Plus</td>
</tr>
<tr>
<td>,</td>
<td>Comma (Cedilla)</td>
</tr>
<tr>
<td>-</td>
<td>Hyphen (Minus)</td>
</tr>
<tr>
<td>.</td>
<td>Period (Decimal Point)</td>
</tr>
<tr>
<td>/</td>
<td>Slant</td>
</tr>
<tr>
<td>0...9</td>
<td>Digits 0 Through 9</td>
</tr>
<tr>
<td>:</td>
<td>Colon</td>
</tr>
<tr>
<td>;</td>
<td>Semicolon</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less Than</td>
</tr>
<tr>
<td>=</td>
<td>Equals</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater Than</td>
</tr>
<tr>
<td>?</td>
<td>Question Mark</td>
</tr>
<tr>
<td>@</td>
<td>Commercial At</td>
</tr>
<tr>
<td>A...Z</td>
<td>Uppercase Latin Letters A Through Z</td>
</tr>
<tr>
<td>[</td>
<td>Opening Bracket</td>
</tr>
<tr>
<td>\</td>
<td>Reverse Slant</td>
</tr>
<tr>
<td>]</td>
<td>Closing Bracket</td>
</tr>
<tr>
<td>^</td>
<td>Circumflex</td>
</tr>
<tr>
<td>_</td>
<td>Underline</td>
</tr>
<tr>
<td>'</td>
<td>Opening Single Quotation Mark (Grave Accent)</td>
</tr>
<tr>
<td>a...z</td>
<td>Lowercase Latin Letters a Through z</td>
</tr>
<tr>
<td>{</td>
<td>Opening Brace</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>Closing Brace</td>
</tr>
<tr>
<td>~</td>
<td>Tilde</td>
</tr>
</tbody>
</table>
APPENDIX F
OPCODE MAP

F.1 INTRODUCTION

This appendix contains the opcode map and additional information for calculating required machine cycles.

F.2 OPCODE MAP

Table F-1 is the opcode map for M6809 processors. The number(s) by each instruction indicates the number of machine cycles required to execute that instruction. When the number contains an "l" (e.g., 4 + l), it indicates that the indexed addressing mode is being used and that an additional number of machine cycles may be required. Refer to Table F-2 to determine the additional machine cycles to be added.

Some instructions in the opcode map have two numbers, the second one in parenthesis. This indicates that the instruction involves a branch. The parenthetical number applies if the branch is taken.

The "page 2, page 3" notation in column one means that all page 2 instructions are preceded by a hexadecimal 10 opcode and all page 3 instructions are preceded by a hexadecimal 11 opcode.
## Table F-1. Opcode Map

<table>
<thead>
<tr>
<th>DIR</th>
<th>REL</th>
<th>ACCA</th>
<th>ACCB</th>
<th>IND</th>
<th>IMM</th>
<th>DIR</th>
<th>IND</th>
<th>EXT</th>
<th>IMM</th>
<th>DIR</th>
<th>IND</th>
<th>EXT</th>
<th>IMM</th>
<th>DIR</th>
<th>IND</th>
<th>EXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>0000</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>

### Most-Significant Four Bits

<table>
<thead>
<tr>
<th>DIR</th>
<th>REL</th>
<th>ACCA</th>
<th>ACCB</th>
<th>IND</th>
<th>IMM</th>
<th>DIR</th>
<th>IND</th>
<th>EXT</th>
<th>IMM</th>
<th>DIR</th>
<th>IND</th>
<th>EXT</th>
<th>IMM</th>
<th>DIR</th>
<th>IND</th>
<th>EXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>0000</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>

### Least-Significant Four Bits

<table>
<thead>
<tr>
<th>DIR</th>
<th>REL</th>
<th>ACCA</th>
<th>ACCB</th>
<th>IND</th>
<th>IMM</th>
<th>DIR</th>
<th>IND</th>
<th>EXT</th>
<th>IMM</th>
<th>DIR</th>
<th>IND</th>
<th>EXT</th>
<th>IMM</th>
<th>DIR</th>
<th>IND</th>
<th>EXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>0000</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>

---

*Note: The table entries represent opcodes with their corresponding instructions.*
Table F-2. Indexed Addressing Mode Data

<table>
<thead>
<tr>
<th>Type</th>
<th>Forms</th>
<th>Non Indirect</th>
<th>Indirect</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Assembler Form</td>
<td>Postbyte</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OP Code</td>
</tr>
<tr>
<td>Constant Offset From R (two's complement offset)</td>
<td>No Offset</td>
<td>.R</td>
<td>1RR00100</td>
</tr>
<tr>
<td></td>
<td>5 Bit Offset</td>
<td>n, R</td>
<td>ORRnnnnn</td>
</tr>
<tr>
<td></td>
<td>8 Bit Offset</td>
<td>n, R</td>
<td>1RR01000</td>
</tr>
<tr>
<td></td>
<td>16 Bit Offset</td>
<td>n, R</td>
<td>1RR01001</td>
</tr>
<tr>
<td>Accumulator Offset From R (two's complement offset)</td>
<td>A — Register Offset</td>
<td>A, R</td>
<td>1RR00110</td>
</tr>
<tr>
<td></td>
<td>B — Register Offset</td>
<td>B, R</td>
<td>1RR01011</td>
</tr>
<tr>
<td></td>
<td>D — Register Offset</td>
<td>D, R</td>
<td>1RR01111</td>
</tr>
<tr>
<td>Auto Increment/Decrement R</td>
<td>Increment By 1</td>
<td>.R+</td>
<td>1RR00000</td>
</tr>
<tr>
<td></td>
<td>Increment By 2</td>
<td>.R++</td>
<td>1RR00001</td>
</tr>
<tr>
<td></td>
<td>Decrement By 1</td>
<td>.R</td>
<td>1RR00010</td>
</tr>
<tr>
<td></td>
<td>Decrement By 2</td>
<td>..R</td>
<td>1RR00011</td>
</tr>
<tr>
<td>Constant Offset From PC (two's complement offset)</td>
<td>8 Bit Offset</td>
<td>n, PCR</td>
<td>1XX01100</td>
</tr>
<tr>
<td></td>
<td>16 Bit Offset</td>
<td>n, PCR</td>
<td>1XX01101</td>
</tr>
<tr>
<td>Extended Indirect</td>
<td>16 Bit Address</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R = X, Y, U or S  
X = Don’t Care  
U = 10  
S = 11  
X = 00  
Y = 01  

+ and # Indicate the number of additional cycles and bytes for the particular variation.
APPENDIX G
PIN ASSIGNMENTS

G.1 INTRODUCTION

This appendix is provided for a quick reference of the pin assignments for the MC6809 and MC6809E processors. Refer to Figure G-1. Descriptions of these pin assignments are given in Section 1.

![PIN ASSIGNMENTS Diagram]

Figure G-1. Pin Assignments
APPENDIX H
CONVERSION TABLES

H.1 INTRODUCTION

This appendix provides some conversion tables for your convenience.

H.2 POWERS OF 2, POWERS OF 16

Refer to Table H-1.

Table H-1. Powers of 2; Powers of 16

<table>
<thead>
<tr>
<th>16^m m =</th>
<th>2^n n =</th>
<th>Value</th>
<th>16^m m =</th>
<th>2^n n =</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>16</td>
<td>65,536</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>2</td>
<td>-</td>
<td>17</td>
<td>131,072</td>
</tr>
<tr>
<td>-2</td>
<td>2</td>
<td>4</td>
<td>-</td>
<td>18</td>
<td>262,144</td>
</tr>
<tr>
<td>-3</td>
<td>3</td>
<td>8</td>
<td>-</td>
<td>19</td>
<td>524,288</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>16</td>
<td>5</td>
<td>20</td>
<td>1,048,576</td>
</tr>
<tr>
<td>-5</td>
<td>5</td>
<td>32</td>
<td>-</td>
<td>21</td>
<td>2,097,152</td>
</tr>
<tr>
<td>-6</td>
<td>6</td>
<td>64</td>
<td>-</td>
<td>22</td>
<td>4,194,304</td>
</tr>
<tr>
<td>-7</td>
<td>7</td>
<td>128</td>
<td>-</td>
<td>23</td>
<td>8,388,608</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>256</td>
<td>6</td>
<td>24</td>
<td>16,777,216</td>
</tr>
<tr>
<td>-9</td>
<td>9</td>
<td>512</td>
<td>-</td>
<td>25</td>
<td>33,554,432</td>
</tr>
<tr>
<td>-10</td>
<td>1,024</td>
<td>-</td>
<td>26</td>
<td>67,108,864</td>
<td></td>
</tr>
<tr>
<td>-11</td>
<td>2,048</td>
<td>-</td>
<td>27</td>
<td>134,217,728</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4,096</td>
<td>7</td>
<td>28</td>
<td>268,435,456</td>
<td></td>
</tr>
<tr>
<td>-13</td>
<td>8,192</td>
<td>-</td>
<td>29</td>
<td>536,870,912</td>
<td></td>
</tr>
<tr>
<td>-14</td>
<td>16,384</td>
<td>-</td>
<td>30</td>
<td>1,073,741,824</td>
<td></td>
</tr>
<tr>
<td>-15</td>
<td>32,768</td>
<td>-</td>
<td>31</td>
<td>2,147,483,648</td>
<td></td>
</tr>
</tbody>
</table>
H.3 HEXADECIMAL AND DECIMAL CONVERSION

Table H-2 is a chart that can be used for converting numbers from either hexadecimal to decimal or decimal to hexadecimal.

H.3.1 CONVERTING HEXADECIMAL TO DECIMAL. Find the decimal weights for corresponding hexadecimal characters beginning with the least-significant character. The sum of the decimal weights is the decimal value of the hexadecimal number.

H.3.2 CONVERTING DECIMAL TO HEXADECIMAL. Find the highest decimal value in the table which is lower than or equal to the decimal number to be converted. The corresponding hexadecimal character is the most-significant digit of the final number. Subtract the decimal value found from the decimal number to be converted. Repeat the above step to determine the hexadecimal character. Repeat this process to find the subsequent hexadecimal numbers.

Table H-2. Hexadecimal and Decimal Conversion Chart

<table>
<thead>
<tr>
<th>15 Byte</th>
<th>8 Byte</th>
<th>7 Byte</th>
<th>0 Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>4,096</td>
<td>1</td>
<td>256</td>
</tr>
<tr>
<td>2</td>
<td>8,192</td>
<td>2</td>
<td>512</td>
</tr>
<tr>
<td>3</td>
<td>12,288</td>
<td>3</td>
<td>768</td>
</tr>
<tr>
<td>4</td>
<td>16,384</td>
<td>4</td>
<td>1,024</td>
</tr>
<tr>
<td>5</td>
<td>20,480</td>
<td>5</td>
<td>1,280</td>
</tr>
<tr>
<td>6</td>
<td>24,576</td>
<td>6</td>
<td>1,536</td>
</tr>
<tr>
<td>7</td>
<td>28,672</td>
<td>7</td>
<td>1,792</td>
</tr>
<tr>
<td>8</td>
<td>32,768</td>
<td>8</td>
<td>2,048</td>
</tr>
<tr>
<td>9</td>
<td>36,864</td>
<td>9</td>
<td>2,304</td>
</tr>
<tr>
<td>A</td>
<td>40,960</td>
<td>A</td>
<td>2,560</td>
</tr>
<tr>
<td>B</td>
<td>45,056</td>
<td>B</td>
<td>2,816</td>
</tr>
<tr>
<td>C</td>
<td>49,152</td>
<td>C</td>
<td>3,072</td>
</tr>
<tr>
<td>D</td>
<td>53,248</td>
<td>D</td>
<td>3,328</td>
</tr>
<tr>
<td>E</td>
<td>57,344</td>
<td>E</td>
<td>3,584</td>
</tr>
<tr>
<td>F</td>
<td>61,440</td>
<td>F</td>
<td>3,840</td>
</tr>
</tbody>
</table>

H-2