MC6801
8-Bit Single-Chip Microcomputer

REFERENCE MANUAL

(including MC6803 and MC68701)
MC6801
8-Bit Single-Chip Microcomputer

REFERENCE MANUAL

Prepared By
MICROPROCESSOR OPERATIONS
AUSTIN, TEXAS

Motorola reserves the right to make changes to any product herein to improve reliability, function, or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others.

Second Edition
©Motorola Inc., 1983
All Rights Reserved
TABLE OF CONTENTS

Chapter 1
The MC6801 Microcomputer: An Overview

1.0 Introduction ................................................................. 1-1
1.1 The MC6801 Instruction Set .............................................. 1-3
1.2 MC6801 System Configurations ....................................... 1-4
1.3 Mode Independent MCU Resources .................................. 1-6
1.3.1 Serial Communications Interface (SCI) .......................... 1-6
1.3.2 Programmable Timer ................................................... 1-8
1.4 Summary of Features ...................................................... 1-9/1-10

Chapter 2
Operating Modes and Memory Maps

2.0 Introduction ..................................................................... 2-1
2.1 MC6801 Fundamental Modes ........................................... 2-3
2.1.1 Single Chip Mode (Mode 7) .......................................... 2-4
2.1.2 Expanded Non-Multiplexed Mode (Mode 5) .................... 2-4
2.1.3 Expanded Multiplexed Mode (Modes 1, 2, 3, 6) .......... 2-6
2.1.4 Test Modes (Modes 0 and 4) ....................................... 2-8
2.2 Mode Associated Memory Maps ........................................ 2-8
2.2.1 Internal Register Area Exclusions ................................. 2-19
2.2.2 Relocatable ROM Options: Modes 1R and 6R ............... 2-20
2.3 Programming The Mode .................................................. 2-20
2.4 MC6801 Comparisons ..................................................... 2-20
2.4.1 MC6800 Bus Comparison ........................................... 2-21
2.4.2 Comparison with MC6803 .......................................... 2-21
2.4.3 Comparison with MC68701 ........................................ 2-21
2.4.4 Comparison with MC68120 ......................................... 2-22

Chapter 3
Functional Pin Description

3.0 Introduction ..................................................................... 3-1
3.1 Mode Independent Pins .................................................. 3-4
3.1.1 XTAL1 and EXTAL2: MCU Clock Inputs .................... 3-4
3.1.2 E: MCU Clock Output ............................................... 3-6
3.1.3 RESET ......................................................................... 3-7
TABLE OF CONTENTS (Continued)

<table>
<thead>
<tr>
<th>Paragraph No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.4</td>
<td>VCC and VSS: MCU Power</td>
<td>3-11</td>
</tr>
<tr>
<td>3.1.5</td>
<td>VCC Standby: RAM Standby Power</td>
<td>3-11</td>
</tr>
<tr>
<td>3.1.5.1</td>
<td>Standby Power Operations</td>
<td>3-11</td>
</tr>
<tr>
<td>3.1.5.1.1</td>
<td>RAM Enable (RAME) Bit</td>
<td>3-11</td>
</tr>
<tr>
<td>3.1.5.1.2</td>
<td>Standby Power (STBY PWR) Bit</td>
<td>3-12</td>
</tr>
<tr>
<td>3.1.5.1.3</td>
<td>RAM Control Register</td>
<td>3-13</td>
</tr>
<tr>
<td>3.1.6</td>
<td>NMI: Non-Maskable Interrupt Request</td>
<td>3-13</td>
</tr>
<tr>
<td>3.1.7</td>
<td>IRQI: Maskable Interrupt Request</td>
<td>3-14</td>
</tr>
<tr>
<td>3.1.8</td>
<td>P10-P17: Port 1</td>
<td>3-14</td>
</tr>
<tr>
<td>3.1.9</td>
<td>P20-P24: Port 2</td>
<td>3-18</td>
</tr>
<tr>
<td>3.1.9.1</td>
<td>Port 2 Data Register Input/Output</td>
<td>3-18</td>
</tr>
<tr>
<td>3.1.9.2</td>
<td>Mode Selection Pins: P20, P21, P22</td>
<td>3-19</td>
</tr>
<tr>
<td>3.1.9.3</td>
<td>Timer Interface: P20, P21</td>
<td>3-19</td>
</tr>
<tr>
<td>3.1.9.4</td>
<td>Serial Communications Interface: P22, P23, P24</td>
<td>3-21</td>
</tr>
<tr>
<td>3.2</td>
<td>Single Chip Mode Pin Description</td>
<td>3-23</td>
</tr>
<tr>
<td>3.2.1</td>
<td>SC1: Input Strobe 3 (IS3)</td>
<td>3-23</td>
</tr>
<tr>
<td>3.2.2</td>
<td>SC2: Output Strobe 3 (OS3)</td>
<td>3-24</td>
</tr>
<tr>
<td>3.2.3</td>
<td>P30-P37: Port 3</td>
<td>3-25</td>
</tr>
<tr>
<td>3.2.3.1</td>
<td>Port 3 Control and Status Register</td>
<td>3-27</td>
</tr>
<tr>
<td>3.2.4</td>
<td>P40-P47: Port 4</td>
<td>3-27</td>
</tr>
<tr>
<td>3.3</td>
<td>Expanded Non-Multiplexed Mode Pin Description</td>
<td>3-29</td>
</tr>
<tr>
<td>3.3.1</td>
<td>SC1: Input/Output Select (IOS)</td>
<td>3-29</td>
</tr>
<tr>
<td>3.3.2</td>
<td>SC2: R/W (Read/Write)</td>
<td>3-30</td>
</tr>
<tr>
<td>3.3.3</td>
<td>P30-P37: Port 3 Data Bus (D0-D7)</td>
<td>3-30</td>
</tr>
<tr>
<td>3.3.4</td>
<td>P40-P47: Port 4 Address Bus/Inputs</td>
<td>3-32</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Expanded Non-Multiplexed Bus Timing</td>
<td>3-33</td>
</tr>
<tr>
<td>3.3.6</td>
<td>Monitoring the Expanded Non-Multiplexed Bus</td>
<td>3-34</td>
</tr>
<tr>
<td>3.4</td>
<td>Expanded Multiplexed Modes Pin Descriptions</td>
<td>3-37</td>
</tr>
<tr>
<td>3.4.1</td>
<td>SC1: Address Strobe (AS)</td>
<td>3-37</td>
</tr>
<tr>
<td>3.4.2</td>
<td>SC2: R/W (Read/Write)</td>
<td>3-37</td>
</tr>
<tr>
<td>3.4.3</td>
<td>P30-P37: Port 3 Multiplexed Address/Data Bus</td>
<td>3-37</td>
</tr>
<tr>
<td>3.4.3.1</td>
<td>Port 3 In Expanded Multiplexed Modes 1, 2, 3 and 6</td>
<td>3-38</td>
</tr>
<tr>
<td>3.4.3.2</td>
<td>Port 3 In Expanded Multiplexed Test Mode 0</td>
<td>3-39</td>
</tr>
<tr>
<td>3.4.4</td>
<td>P40-P47: Port 4 Address Bus/Data Inputs</td>
<td>3-40</td>
</tr>
<tr>
<td>3.4.5</td>
<td>Expanded Multiplexed Bus Timing</td>
<td>3-42</td>
</tr>
<tr>
<td>3.4.6</td>
<td>Monitoring the Expanded Multiplexed Bus</td>
<td>3-45</td>
</tr>
</tbody>
</table>

Chapter 4
The MC6801 Microprocessor Unit (MPU)

<table>
<thead>
<tr>
<th>Paragraph No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0</td>
<td>Introduction</td>
<td>4-1</td>
</tr>
<tr>
<td>4.1</td>
<td>Assembler Source Statements</td>
<td>4-2</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Labels</td>
<td>4-3</td>
</tr>
<tr>
<td>4.2</td>
<td>Addressing Modes</td>
<td>4-3</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Inherent Addressing Mode</td>
<td>4-4</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Immediate Addressing Mode</td>
<td>4-4</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>4.2.3</td>
<td>Direct and Extended Addressing Modes</td>
<td>4-5</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Relative Addressing Mode</td>
<td>4-6</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Indexed Addressing Mode</td>
<td>4-7</td>
</tr>
<tr>
<td>4.3</td>
<td>MC6801 Instruction Set</td>
<td>4-7</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Condition Code Register Instructions</td>
<td>4-10</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Accumulator and Memory Instructions</td>
<td>4-10</td>
</tr>
<tr>
<td>4.3.2.1</td>
<td>Arithmetic Instructions</td>
<td>4-11</td>
</tr>
<tr>
<td>4.3.2.2</td>
<td>Logic Instructions</td>
<td>4-11</td>
</tr>
<tr>
<td>4.3.2.3</td>
<td>Data Test Instructions</td>
<td>4-13</td>
</tr>
<tr>
<td>4.3.2.4</td>
<td>Data Handling Instructions</td>
<td>4-14</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Program Control Instructions</td>
<td>4-16</td>
</tr>
<tr>
<td>4.3.3.1</td>
<td>Index Register Instructions</td>
<td>4-16</td>
</tr>
<tr>
<td>4.3.3.2</td>
<td>Stack Pointer Instructions</td>
<td>4-17</td>
</tr>
<tr>
<td>4.3.3.3</td>
<td>Jump and Branch Instructions</td>
<td>4-19</td>
</tr>
<tr>
<td>4.4</td>
<td>Programming Examples</td>
<td>4-25</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Use of the Index Register</td>
<td>4-25</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Number Systems</td>
<td>4-29</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Two's Complement Overflow</td>
<td>4-32</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Arithmetic Instructions Revisited</td>
<td>4-34</td>
</tr>
<tr>
<td>4.4.4.1</td>
<td>Use of Arithmetic Instructions Revisited</td>
<td>4-35</td>
</tr>
<tr>
<td>4.4.5</td>
<td>Multi-Byte Addition and Subtraction</td>
<td>4-36</td>
</tr>
<tr>
<td>4.4.6</td>
<td>Multiplication</td>
<td>4-38</td>
</tr>
<tr>
<td>4.4.6.1</td>
<td>Multiplication Using the MUL Instruction</td>
<td>4-45</td>
</tr>
<tr>
<td>4.4.6.2</td>
<td>Multiplication Using Booth's Algorithm</td>
<td>4-50</td>
</tr>
<tr>
<td>4.4.7</td>
<td>Division</td>
<td>4-53</td>
</tr>
</tbody>
</table>

Chapter 5
The MC6801 Interrupt Structure

<table>
<thead>
<tr>
<th>Paragraph No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>Introduction</td>
<td>5-1</td>
</tr>
<tr>
<td>5.1</td>
<td>Interrupt Considerations</td>
<td>5-1</td>
</tr>
<tr>
<td>5.2</td>
<td>MC6801 Interrupt Generation</td>
<td>5-3</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Non-Maskable Interrupt (NMI)</td>
<td>5-3</td>
</tr>
<tr>
<td>5.2.1.1</td>
<td>System Considerations in Using NMI</td>
<td>5-3</td>
</tr>
<tr>
<td>5.2.1.2</td>
<td>Using NMI as a Maskable Interrupt</td>
<td>5-4</td>
</tr>
<tr>
<td>5.2.2</td>
<td>MC6801 Maskable Interrupts (IRQ1 and IRQ2)</td>
<td>5-6</td>
</tr>
<tr>
<td>5.3</td>
<td>MC6801 Interrupt Response</td>
<td>5-7</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Selection of Interrupt Vectors</td>
<td>5-15</td>
</tr>
<tr>
<td>5.3.2</td>
<td>MC6801 Operating Mode and Interrupt Vector</td>
<td>5-17</td>
</tr>
<tr>
<td>5.4</td>
<td>MC6801 Interrupt Instructions</td>
<td>5-18</td>
</tr>
<tr>
<td>5.4.1</td>
<td>MC6801 Instructions Affecting the I-Bit</td>
<td>5-18</td>
</tr>
<tr>
<td>5.4.1.1</td>
<td>The CLI and SEI Instructions</td>
<td>5-18</td>
</tr>
<tr>
<td>5.4.1.2</td>
<td>The TAP Instruction</td>
<td>5-18</td>
</tr>
<tr>
<td>5.4.1.3</td>
<td>The RTI Instruction</td>
<td>5-19</td>
</tr>
<tr>
<td>5.4.1.4</td>
<td>The SWI Instruction</td>
<td>5-19</td>
</tr>
<tr>
<td>5.4.2</td>
<td>The Wait for Interrupt (WAI) Instruction</td>
<td>5-19</td>
</tr>
<tr>
<td>5.5</td>
<td>Providing Interrupt Service</td>
<td>5-21</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>5.6</td>
<td>Program Restartability</td>
<td>5-23</td>
</tr>
<tr>
<td>5.6.1</td>
<td>Re-Entrant Routines</td>
<td>5-23</td>
</tr>
<tr>
<td>5.6.2</td>
<td>Resource Sharing</td>
<td>5-24</td>
</tr>
</tbody>
</table>

Chapter 6

Serial Communications Interface (SCI)

6.0   Introduction .......................................................... 6-1
6.1   Serial Communications Interface Registers ......................... 6-1
6.1.1  Rate and Mode Control Register .................................... 6-2
6.1.2  Transmit/Receive Control and Status Register ....................... 6-5
6.1.3  Transmit and Receive Data Registers ................................ 6-6
6.2   SCI Clocking Options .................................................. 6-6
6.2.1  Using the Internal SCI Clock ......................................... 6-6
6.2.2  Using an External SCI Clock .......................................... 6-7
6.2.3  Providing a Serial Output Clock ..................................... 6-7
6.2.4  Clocking Multiple SCIs ................................................. 6-7
6.3   Serial Data Formats .................................................... 6-8
6.4   Serial Communication Operations .................................... 6-9
6.4.1  Transmitter Operation ............................................... 6-9
6.4.2  Receiver Operation ..................................................... 6-9
6.5   The Wake-Up Feature .................................................... 6-10
6.5.1  Transmitter Duties During Wake-Up Operation ....................... 6-11
6.5.2  Receiver Duties During Wake-Up Operation ......................... 6-12
6.6   Providing SCI Interrupt Service ...................................... 6-12
6.7   Two SCI Software Examples ............................................ 6-13
6.7.1  Exercising the Serial Communications Interface .................... 6-13
6.7.2  Demonstrating the Wake-Up Feature ................................... 6-13

Chapter 7

The MC6801 Programmable Timer

7.0   Introduction ................................................................... 7-1
7.1   Programmable Timer Registers .......................................... 7-1
7.1.1  Counter Register ($09-0A) ........................................... 7-2
7.1.2  Output Compare Register ($0B:0C) .................................... 7-3
7.1.3  Input Capture Register ($0D:0E) ..................................... 7-4
7.1.4  Timer Control and Status Register ($08) ............................ 7-5
7.2   Selected Programmable Timer Examples ................................ 7-6
7.2.1  Reading the Counter Register ........................................ 7-6
7.2.2  Generating an Output Waveform ...................................... 7-7
7.2.3  Generating a Synchronized Output Compare ......................... 7-10
7.2.4  Echoing an Input Signal .............................................. 7-12
7.2.5  Generating an Input Capture Using LiLbug (TM)* ................... 7-13
# TABLE OF CONTENTS (Concluded)

<table>
<thead>
<tr>
<th>Paragraph No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.0</td>
<td>Introduction</td>
<td>8-1</td>
</tr>
<tr>
<td>8.1</td>
<td>Interface to Static RAM (Modes 1, 2, 3, 6)</td>
<td>8-1</td>
</tr>
<tr>
<td>8.1.1</td>
<td>Expanded Multiplexed Bus Timing</td>
<td>8-1</td>
</tr>
<tr>
<td>8.1.2</td>
<td>An MC6801 Interface with MCM2114 Static RAM</td>
<td>8-2</td>
</tr>
<tr>
<td>8.1.3</td>
<td>Final Remarks</td>
<td>8-3</td>
</tr>
<tr>
<td>8.2</td>
<td>Port 3 Parallel Interfaces (Mode 7)</td>
<td>8-8</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Line Printer Interface to Port 3</td>
<td>8-8</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Keyboard Interface to Port 3</td>
<td>8-10</td>
</tr>
<tr>
<td>8.3</td>
<td>Prioritized Interrupt Vectors (Modes 1, 2, 3)</td>
<td>8-14</td>
</tr>
<tr>
<td>8.3.1</td>
<td>General Considerations</td>
<td>8-14</td>
</tr>
<tr>
<td>8.3.2</td>
<td>8-Level Prioritizing Scheme</td>
<td>8-15</td>
</tr>
<tr>
<td>8.3.2.1</td>
<td>Generating the Address of the Vector</td>
<td>8-15</td>
</tr>
<tr>
<td>8.3.2.2</td>
<td>Providing the Interrupt Vector</td>
<td>8-15</td>
</tr>
<tr>
<td>8.3.3</td>
<td>Final Remarks</td>
<td>8-17</td>
</tr>
<tr>
<td>8.4</td>
<td>Memory and ACIA Interface (Mode 5)</td>
<td>8-19</td>
</tr>
<tr>
<td>8.4.1</td>
<td>Obtaining 256 Additional Bytes of Read-Only Memory</td>
<td>8-20</td>
</tr>
<tr>
<td>8.5</td>
<td>Period Measurement</td>
<td>8-22</td>
</tr>
<tr>
<td>8.5.1</td>
<td>Measuring Periods Less than 65,536 Cycles</td>
<td>8-23</td>
</tr>
<tr>
<td>8.5.2</td>
<td>Measuring Periods Exceeding 65,535 Cycles</td>
<td>8-24</td>
</tr>
<tr>
<td>8.5.3</td>
<td>Period Measurement Sample Programs</td>
<td>8-26</td>
</tr>
<tr>
<td>8.6</td>
<td>SCI Parallel Interfaces (All Modes)</td>
<td>8-33</td>
</tr>
<tr>
<td>8.6.1</td>
<td>SCI Parallel-to-Serial Input Interface</td>
<td>8-33</td>
</tr>
<tr>
<td>8.6.2</td>
<td>SCI Serial-to-Parallel Output Interface</td>
<td>8-36</td>
</tr>
<tr>
<td>8.7</td>
<td>Dual Processor Parallel Interfaces (Mode 7)</td>
<td>8-39</td>
</tr>
<tr>
<td>8.7.1</td>
<td>Interface Control Schemes</td>
<td>8-39</td>
</tr>
<tr>
<td>8.7.2</td>
<td>8-Bit Half Duplex Interface</td>
<td>8-40</td>
</tr>
<tr>
<td>8.7.3</td>
<td>4-Bit Full Duplex Interface</td>
<td>8-45</td>
</tr>
<tr>
<td>8.7.3.1</td>
<td>Full Duplex with Input Capture Function</td>
<td>8-46</td>
</tr>
<tr>
<td>8.7.3.2</td>
<td>Full Duplex with Exclusive-OR Function</td>
<td>8-49</td>
</tr>
</tbody>
</table>

## Appendices

A Definition of the Executable Instructions
B MC6801 Operation Code Map
C ASCII Conversion Table
D Selected Powers of 2 and 16
E The MC68701 Microcomputer Unit
F Cycle-by-Cycle Bus Activity
G Glossary
H Summary of Instruction E-Cycle Counts
I Expanded Multiplexed Bus Clocking
J Reset Vector Chip Select Circuit for Mode 0
K MC6801 System Development Tools

## Index
# LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Chapter 1</strong></td>
<td></td>
</tr>
<tr>
<td>1-1</td>
<td>MC6801 Single Chip Microcomputer</td>
<td>1-2</td>
</tr>
<tr>
<td>1-2</td>
<td>MC6801 Programming Model</td>
<td>1-3</td>
</tr>
<tr>
<td>1-3</td>
<td>Single Chip Mode</td>
<td>1-4</td>
</tr>
<tr>
<td>1-4</td>
<td>Expanded Non-Multiplexed Mode</td>
<td>1-5</td>
</tr>
<tr>
<td>1-5</td>
<td>Expanded Multiplexed Mode</td>
<td>1-5</td>
</tr>
<tr>
<td>1-6</td>
<td>SCI Register Organization</td>
<td>1-7</td>
</tr>
<tr>
<td>1-7</td>
<td>Transmit/Receive Control and Status Register</td>
<td>1-7</td>
</tr>
<tr>
<td>1-8</td>
<td>Programmable Timer</td>
<td>1-8</td>
</tr>
<tr>
<td>1-9</td>
<td>Timer Control and Status Register</td>
<td>1-9/1-10</td>
</tr>
<tr>
<td></td>
<td><strong>Chapter 2</strong></td>
<td></td>
</tr>
<tr>
<td>2-1</td>
<td>Summary of Operating Mode Characteristics</td>
<td>2-2</td>
</tr>
<tr>
<td>2-2</td>
<td>Single Chip Mode</td>
<td>2-4</td>
</tr>
<tr>
<td>2-3</td>
<td>Expanded Non-Multiplexed Mode</td>
<td>2-5</td>
</tr>
<tr>
<td>2-4</td>
<td>System Configuration — Expanded Non-Multiplexed Mode</td>
<td>2-6</td>
</tr>
<tr>
<td>2-5</td>
<td>Expanded Multiplexed Mode</td>
<td>2-7</td>
</tr>
<tr>
<td>2-6</td>
<td>System Configuration for Expanded Multiplexed Modes</td>
<td>2-7</td>
</tr>
<tr>
<td>2-7</td>
<td>Memory Map for Mode 0</td>
<td>2-9</td>
</tr>
<tr>
<td>2-8</td>
<td>Memory Map for Mode 1</td>
<td>2-10</td>
</tr>
<tr>
<td>2-9</td>
<td>Memory Map for Mode 1R</td>
<td>2-11</td>
</tr>
<tr>
<td>2-10</td>
<td>Memory Map for Mode 2</td>
<td>2-12</td>
</tr>
<tr>
<td>2-11</td>
<td>Memory Map for Mode 3</td>
<td>2-13</td>
</tr>
<tr>
<td>2-12</td>
<td>Memory Map for Mode 4</td>
<td>2-14</td>
</tr>
<tr>
<td>2-13</td>
<td>Memory Map for Mode 5</td>
<td>2-15</td>
</tr>
<tr>
<td>2-14</td>
<td>Memory Map for Mode 6</td>
<td>2-16</td>
</tr>
<tr>
<td>2-15</td>
<td>Memory Map for Mode 6R</td>
<td>2-17</td>
</tr>
<tr>
<td>2-16</td>
<td>Memory Map for Mode 7</td>
<td>2-18</td>
</tr>
<tr>
<td>2-17</td>
<td>MCU Internal Register Area</td>
<td>2-19</td>
</tr>
<tr>
<td></td>
<td><strong>Chapter 3</strong></td>
<td></td>
</tr>
<tr>
<td>3-1</td>
<td>MC6801 Pin Diagram</td>
<td>3-1</td>
</tr>
<tr>
<td>3-2</td>
<td>MC6801 Block Diagram</td>
<td>3-2</td>
</tr>
<tr>
<td>3-3</td>
<td>MC6801 Recommended Crystal Parameters</td>
<td>3-5</td>
</tr>
<tr>
<td>3-4</td>
<td>RESET Timing</td>
<td>3-9</td>
</tr>
<tr>
<td>3-5</td>
<td>Mode Programming Levels and Timing</td>
<td>3-9</td>
</tr>
<tr>
<td>3-6</td>
<td>Programming the Mode with Diodes</td>
<td>3-10</td>
</tr>
</tbody>
</table>
### LIST OF ILLUSTRATIONS (Continued)

<table>
<thead>
<tr>
<th>Figure No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-7</td>
<td>Programming the Mode with Analog Switches and Diodes</td>
<td>3-10</td>
</tr>
<tr>
<td>3-8</td>
<td>Data Port Timing for MPU Read</td>
<td>3-15</td>
</tr>
<tr>
<td>3-9</td>
<td>Data Port Timing for MPU Write</td>
<td>3-15</td>
</tr>
<tr>
<td>3-10</td>
<td>Logic Diagram for Port 1</td>
<td>3-16</td>
</tr>
<tr>
<td>3-11</td>
<td>Port 2 Data Register</td>
<td>3-19</td>
</tr>
<tr>
<td>3-12</td>
<td>Logic Diagram for Port 2 Bit 0</td>
<td>3-20</td>
</tr>
<tr>
<td>3-13</td>
<td>Logic Diagram for Port 2 Bit 1</td>
<td>3-20</td>
</tr>
<tr>
<td>3-14</td>
<td>Logic Diagram for Port 2 Bit 2</td>
<td>3-21</td>
</tr>
<tr>
<td>3-15</td>
<td>Logic Diagram for Port 2 Bit 3</td>
<td>3-22</td>
</tr>
<tr>
<td>3-16</td>
<td>Logic Diagram for Port 2 Bit 4</td>
<td>3-23</td>
</tr>
<tr>
<td>3-17</td>
<td>Port 3 Latch Setup and Hold Times</td>
<td>3-24</td>
</tr>
<tr>
<td>3-18</td>
<td>Output Strobe 3 (OS3) Timing</td>
<td>3-25</td>
</tr>
<tr>
<td>3-19</td>
<td>Logic Diagram for Port 3</td>
<td>3-26</td>
</tr>
<tr>
<td>3-20</td>
<td>Logic Diagram for Port 4</td>
<td>3-29</td>
</tr>
<tr>
<td>3-21</td>
<td>Logic Diagram for Port 3 (Repeated)</td>
<td>3-31</td>
</tr>
<tr>
<td>3-22</td>
<td>Logic Diagram for Port 4 (Repeated)</td>
<td>3-33</td>
</tr>
<tr>
<td>3-23</td>
<td>Expanded Non-Multiplexed Bus Timing</td>
<td>3-34</td>
</tr>
<tr>
<td>3-24</td>
<td>Typical Expanded Non-Multiplexed System</td>
<td>3-35</td>
</tr>
<tr>
<td>3-25</td>
<td>External Bus — Expanded Non-Multiplexed Mode</td>
<td>3-36</td>
</tr>
<tr>
<td>3-26</td>
<td>Logic Diagram for Port 3 (Repeated)</td>
<td>3-38</td>
</tr>
<tr>
<td>3-27</td>
<td>Logic Diagram for Port 4 (Repeated)</td>
<td>3-41</td>
</tr>
<tr>
<td>3-28</td>
<td>Expanded Multiplexed Bus Timing</td>
<td>3-42</td>
</tr>
<tr>
<td>3-29</td>
<td>Typical Bus De-Multiplexing Latch Arrangement</td>
<td>3-43</td>
</tr>
<tr>
<td>3-30</td>
<td>Typical Expanded Multiplexed System</td>
<td>3-44</td>
</tr>
<tr>
<td>3-31</td>
<td>External Bus — Expanded Multiplexed Mode</td>
<td>3-46</td>
</tr>
</tbody>
</table>

### Chapter 4

<table>
<thead>
<tr>
<th>Section No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-1</td>
<td>MC6801 MPU Programming Model</td>
<td>4-1</td>
</tr>
<tr>
<td>4-2</td>
<td>MC6801 Instruction Set Summary</td>
<td>4-8</td>
</tr>
<tr>
<td>4-3</td>
<td>Condition Code Register Bit Definitions</td>
<td>4-11</td>
</tr>
<tr>
<td>4-4</td>
<td>Condition Code Register Instructions</td>
<td>4-11</td>
</tr>
<tr>
<td>4-5</td>
<td>Arithmetic Instructions</td>
<td>4-12</td>
</tr>
<tr>
<td>4-6</td>
<td>Logic Instructions</td>
<td>4-12</td>
</tr>
<tr>
<td>4-7</td>
<td>Data Test Instructions</td>
<td>4-13</td>
</tr>
<tr>
<td>4-8</td>
<td>Data Handling Instructions</td>
<td>4-14</td>
</tr>
<tr>
<td>4-9</td>
<td>Index Register Instructions</td>
<td>4-17</td>
</tr>
<tr>
<td>4-10</td>
<td>Stack Pointer Instructions</td>
<td>4-18</td>
</tr>
<tr>
<td>4-11</td>
<td>Operation of Push Instruction</td>
<td>4-20</td>
</tr>
<tr>
<td>4-12</td>
<td>Operation of Pull Instruction</td>
<td>4-21</td>
</tr>
<tr>
<td>4-13</td>
<td>Jump and Branch Instructions</td>
<td>4-22</td>
</tr>
<tr>
<td>4-14</td>
<td>Operation of JSR (Extended) Instruction</td>
<td>4-22</td>
</tr>
<tr>
<td>4-15</td>
<td>Operation of JSR (Indexed) Instruction</td>
<td>4-23</td>
</tr>
<tr>
<td>4-16</td>
<td>Operation of BSR Instruction</td>
<td>4-23</td>
</tr>
<tr>
<td>4-17</td>
<td>Operation of RTS Instruction</td>
<td>4-24</td>
</tr>
<tr>
<td>4-18</td>
<td>BCD Addition: BCD1</td>
<td>4-28</td>
</tr>
</tbody>
</table>
## LIST OF ILLUSTRATIONS (Continued)

<table>
<thead>
<tr>
<th>Figure No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-19</td>
<td>Block Move Routine: BLOCKC</td>
<td>4-30</td>
</tr>
<tr>
<td>4-20</td>
<td>Block Move Routine: BLOCKM</td>
<td>4-31</td>
</tr>
<tr>
<td>4-21</td>
<td>Arithmetic Instructions (Repeated)</td>
<td>4-34</td>
</tr>
<tr>
<td>4-22</td>
<td>Operation of DAA Instruction</td>
<td>4-35</td>
</tr>
<tr>
<td>4-23</td>
<td>BCD Addition Routine: BCDAD1</td>
<td>4-39</td>
</tr>
<tr>
<td>4-24</td>
<td>BCD Addition Routine: BCDAD2</td>
<td>4-40</td>
</tr>
<tr>
<td>4-25</td>
<td>BCD Subtraction Routine: BCDSB1</td>
<td>4-42</td>
</tr>
<tr>
<td>4-26</td>
<td>BCD Subtraction Routine: BCDSB2</td>
<td>4-43</td>
</tr>
<tr>
<td>4-27</td>
<td>Unsigned Multiplication: MUL16A</td>
<td>4-46</td>
</tr>
<tr>
<td>4-28</td>
<td>Signed Multiplication Routine: MUL16B</td>
<td>4-48</td>
</tr>
<tr>
<td>4-29</td>
<td>Multiplication Using Booth's Algorithm</td>
<td>4-50</td>
</tr>
<tr>
<td>4-30</td>
<td>Flowchart for Booth's Algorithm</td>
<td>4-51</td>
</tr>
<tr>
<td>4-31</td>
<td>Signed Multiplication Routine: MUL16</td>
<td>4-52</td>
</tr>
<tr>
<td>4-32</td>
<td>Flowchart for Unsigned Division</td>
<td>4-54</td>
</tr>
<tr>
<td>4-33</td>
<td>Unsigned Division Routine: DIV16B</td>
<td>4-55</td>
</tr>
</tbody>
</table>

## Chapter 5

5-1 Polling Loop Sequence ........................................................................... 5-2
5-2 Hierarchical Polling Loop Sequence....................................................... 5-2
5-3 An NMI Mask Circuit ............................................................................... 5-5
5-4 Software for an NMI Mask ....................................................................... 5-5
5-5 Generalized Interrupt Control and Generation ......................................... 5-6
5-6 Logic Diagrams for Interrupt Sampling .................................................... 5-7
5-7 Interrupt Recognition Windows ............................................................... 5-9
5-8 MC6801 Processor Flowchart ................................................................... 5-10
5-9 MC6801 Non-Interrupt Flowchart ............................................................. 5-11
5-10 MC6801 Interrupt Flowchart ................................................................... 5-12
5-11 Stacking the Machine State .................................................................... 5-13
5-12 MC6801 Interrupt Sequence ..................................................................... 5-14
5-13 Pulsing the IRQ2 Interrupt Line ........................................................... 5-17
5-14 MC6801 Interrupt Vectors ....................................................................... 5-17
5-15 WAI Instruction Sequence ................................................................------- 5-20
5-16 A Routine to Skip the WAI Instruction ................................................ 5-22
5-17 A "Who-Done-It" Routine ....................................................................... 5-23
5-18 Example of Re-Entrant Programming .................................................... 5-26

## Chapter 6

6-1 Serial Communications Interface Registers .............................................. 6-2
6-2 Block Diagram of SCI ............................................................................... 6-3
6-3 Rate and Mode Control Register ............................................................... 6-4
6-4 Selected Internal Bit Times and Rates ..................................................... 6-4
6-5 Format and Clock Source Control ............................................................. 6-4
6-6 Transmit/Receive Control and Status Register .......................................... 6-5
## LIST OF ILLUSTRATIONS (Continued)

<table>
<thead>
<tr>
<th>Figure No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-7</td>
<td>SCI Serial Data Formats</td>
<td>6-8</td>
</tr>
<tr>
<td>6-8</td>
<td>Exercising the SCI: SERIAL</td>
<td>6-15</td>
</tr>
<tr>
<td>6-9</td>
<td>Example of SERIAL Program Output</td>
<td>6-19</td>
</tr>
<tr>
<td>6-10</td>
<td>Demonstrating the Wake-Up Feature: OKBAD</td>
<td>6-20</td>
</tr>
<tr>
<td>6-11</td>
<td>Example of OKBAD Program Output</td>
<td>6-24</td>
</tr>
</tbody>
</table>

### Chapter 7

<table>
<thead>
<tr>
<th>Figure No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-1</td>
<td>MC6801 Programming Timer Registers</td>
<td>7-1</td>
</tr>
<tr>
<td>7-2</td>
<td>Block Diagram of Programmable Timer</td>
<td>7-2</td>
</tr>
<tr>
<td>7-3</td>
<td>Output Compare Timing</td>
<td>7-4</td>
</tr>
<tr>
<td>7-4</td>
<td>Input Capture Timing</td>
<td>7-4</td>
</tr>
<tr>
<td>7-5</td>
<td>Timer Control and Status Register (TCSR)</td>
<td>7-5</td>
</tr>
<tr>
<td>7-6</td>
<td>Counter Register Write and Read Diagram</td>
<td>7-7</td>
</tr>
<tr>
<td>7-7</td>
<td>WAVGEN Default Output Signal</td>
<td>7-7</td>
</tr>
<tr>
<td>7-8</td>
<td>Generating a Waveform: WAVGEN</td>
<td>7-8</td>
</tr>
<tr>
<td>7-9</td>
<td>Synchronized Loop Timing</td>
<td>7-10</td>
</tr>
<tr>
<td>7-10</td>
<td>Synchronized Output Compare: SYNLUP</td>
<td>7-11</td>
</tr>
<tr>
<td>7-11</td>
<td>Immediate Output Compare Timing</td>
<td>7-12</td>
</tr>
<tr>
<td>7-12</td>
<td>Equipment Arrangement for Program ECHO</td>
<td>7-13</td>
</tr>
<tr>
<td>7-13</td>
<td>Echoing an Input: ECHO</td>
<td>7-14</td>
</tr>
</tbody>
</table>

### Chapter 8

<table>
<thead>
<tr>
<th>Figure No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-1</td>
<td>Expanded Multiplexed Bus Timing (Repeated)</td>
<td>8-2</td>
</tr>
<tr>
<td>8-2</td>
<td>Typical Data for MCM2114 Static RAM</td>
<td>8-4</td>
</tr>
<tr>
<td>8-3</td>
<td>Expanded Multiplexed Bus Interface with MCM2114</td>
<td>8-7</td>
</tr>
<tr>
<td>8-4</td>
<td>Line Printer Interface Connection Diagram</td>
<td>8-9</td>
</tr>
<tr>
<td>8-5</td>
<td>Line Printer Interface Signals</td>
<td>8-10</td>
</tr>
<tr>
<td>8-6</td>
<td>Line Printer Interface Driver: PINZ, POUTCH</td>
<td>8-11</td>
</tr>
<tr>
<td>8-7</td>
<td>Keyboard Interface Signals</td>
<td>8-12</td>
</tr>
<tr>
<td>8-8</td>
<td>Keyboard Interface Connection Diagram</td>
<td>8-12</td>
</tr>
<tr>
<td>8-9</td>
<td>Keyboard Interface Drive: KEYINZ, KEYIN</td>
<td>8-13</td>
</tr>
<tr>
<td>8-10</td>
<td>8-Level Priority Encoder</td>
<td>8-16</td>
</tr>
<tr>
<td>8-11</td>
<td>Priority Encoder Interrupt Vectors</td>
<td>8-17</td>
</tr>
<tr>
<td>8-12</td>
<td>Priority Encoder Timing</td>
<td>8-18</td>
</tr>
<tr>
<td>8-13</td>
<td>Expanded Non-Multiplexed Bus Timing (Repeated)</td>
<td>8-19</td>
</tr>
<tr>
<td>8-14</td>
<td>Memory and ACIA Interface in Expanded Non-Multiplexed Mode</td>
<td>8-21</td>
</tr>
<tr>
<td>8-15</td>
<td>PROM Interface in Mode 5</td>
<td>8-22</td>
</tr>
<tr>
<td>8-16</td>
<td>Minimum Period Measurement Using MC6801 Timer</td>
<td>8-24</td>
</tr>
<tr>
<td>8-17</td>
<td>Special Cases During Period Measurement</td>
<td>8-26</td>
</tr>
<tr>
<td>8-18</td>
<td>Period Measurement Sample Program: TIM24</td>
<td>8-27</td>
</tr>
<tr>
<td>8-19</td>
<td>Example of TIM24 and TIM16 Output</td>
<td>8-33</td>
</tr>
<tr>
<td>8-20</td>
<td>Period Measurement Sample Program: TIM16</td>
<td>8-34</td>
</tr>
</tbody>
</table>
## LIST OF ILLUSTRATIONS (Concluded)

<table>
<thead>
<tr>
<th>Figure No.</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-21</td>
<td>SCI Parallel-to-Serial Interface</td>
<td>8-37</td>
</tr>
<tr>
<td>8-22</td>
<td>SCI Serial-to-Parallel Interface</td>
<td>8-38</td>
</tr>
<tr>
<td>8-23</td>
<td>Half-Duplex Interface</td>
<td>8-41</td>
</tr>
<tr>
<td>8-24</td>
<td>Request/Grant Control Signals</td>
<td>8-41</td>
</tr>
<tr>
<td>8-25</td>
<td>Half Duplex Routines</td>
<td>8-42</td>
</tr>
<tr>
<td>8-26</td>
<td>Half Duplex Data Transfer Rate</td>
<td>8-46</td>
</tr>
<tr>
<td>8-27</td>
<td>Interface Using Input Capture Function</td>
<td>8-47</td>
</tr>
<tr>
<td>8-28</td>
<td>Flowchart for Interface with Input Capture Function</td>
<td>8-48</td>
</tr>
<tr>
<td>8-29</td>
<td>Interface Using Exclusive-OR Function</td>
<td>8-49</td>
</tr>
<tr>
<td>8-30</td>
<td>Full Duplex Routine Using Exclusive-OR</td>
<td>8-51</td>
</tr>
<tr>
<td>8-31</td>
<td>Flowchart for Interface</td>
<td>8-53</td>
</tr>
<tr>
<td>8-32</td>
<td>Full Duplex Data Transfer Rates (Exclusive-OR)</td>
<td>8-54</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>Chapter 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-1</td>
<td>MC6801 Additional Instructions</td>
<td>4-10</td>
</tr>
<tr>
<td>4-2</td>
<td>Branch Instructions</td>
<td>4-24</td>
</tr>
<tr>
<td>4-3</td>
<td>Overflow Rules for Addition</td>
<td>4-32</td>
</tr>
<tr>
<td>4-4</td>
<td>Overflow Rules for Subtraction</td>
<td>4-33</td>
</tr>
<tr>
<td>4-5</td>
<td>Truth Table for “Add with Carry”</td>
<td>4-36</td>
</tr>
<tr>
<td>4-6</td>
<td>Truth Table for “Subtract with Borrow”</td>
<td>4-36</td>
</tr>
</tbody>
</table>
CHAPTER 1
THE MC6801 MICROCOMPUTER: AN OVERVIEW

1.0 INTRODUCTION

The Motorola MC6801 Microcomputer Unit (MCU)* is the most versatile and powerful single-chip microcomputer currently available to the system designer. The variety of MC6801 operating modes offers the designer an unexcelled measure of freedom in configuring a microcomputer to specific system requirements. In addition to operational flexibility, the MCU also provides an extremely powerful set of internal resources. These resources can provide a significant cost savings by a reduction in the system total part count. They include the following:

- 2048 bytes of ROM,
- 128 bytes of RAM,
- a maximum of 29 parallel I/O and 2 control lines,
- a three function 16-bit timer, and
- a full duplex Serial Communications Interface.

The extraordinary flexibility of the MC6801 is provided by its ability to be operated in a variety of modes depending upon application requirements. Selection of the hardware-programmed mode is based upon the available internal resources for a particular mode and those required by the application. The details involved in mode selection are contained in Chapter 2, but this discussion provides an overview of the available modes.

The MC6801 provides three fundamental operating modes:

- Single Chip,
- Expanded Non-Multiplexed,
- Expanded Multiplexed.

The Single Chip mode utilizes only on-chip resources while maximizing the number of available input/output lines. The Expanded Non-Multiplexed mode offers a modest increase in the external address space (256 external read/write locations) with separate address and data buses. The Expanded Multiplexed mode time-multiplexes the address and data buses which provides a 64K-byte address space while requiring only a simple latch to de-multiplex the bus.

The MC6801 is a complete monolithic microcomputer housed in a single 40-pin package and is the product of state-of-the-art advances in scaled NMOS process technology. A block diagram, shown in Figure 1-1, illustrates the integration of the on-chip resources into a complete powerful microcomputer. The MCU contains an enhanced 8-bit MC6800 MPU, 2048 bytes of ROM, 128 bytes of RAM, four Parallel I/O Ports, a Serial Communications Interface (SCI), a Programmable Timer with three functions, and an internal clock generator. As shown in Figure 1-1, several pin configurations depend upon the MCU operating mode. All of the pins associated with Port 3, Port 4, SCI, and SC2 are mode dependent.

*The MC6801 Microcomputer is also referred to as the MCU, MC6801, and/or MC6801 MCV throughout this manual.
Figure 1-1. MC6801 Single Chip Microcomputer
The concept of an integrated family of devices is predicated on continuity in both design and development. In the design of a third generation product — such as the MC6801 — one of the most desirable objectives is to achieve compatibility with existing software and hardware. The Motorola MC6801 satisfies this goal and is compatible with the entire M6800 Family of components. In addition, it requires only a single +5 volt power supply and will directly interface with both TTL and MOS peripheral devices.

As a central member of the M6800 Family, the MCU shares many attributes of the MC6800 MPU. For example, the MCU implements the entire MC6800 instruction set. Additional instructions have been incorporated, however, which provide both greater system capability and ease in programming. These enhancements can result in increased throughput, simplified software conversion effort, and reduced development time.

1.1 THE MC6801 INSTRUCTION SET

The MC6801 instruction set will be very familiar to those readers having experience with the MC6800. Those who have not had this experience, however, will appreciate the easy-to-learn instruction set. The MC6800 is upward compatible with the MC6801 with respect to both source and object code. Significant improvements have been incorporated in the MC6801 to improve instruction throughput. In addition, several new instructions provide more capability in implementing 16-bit arithmetic operations.

A programming model of the MC6801 is shown in Figure 1-2. Readers familiar with the MC6800 should note that the significant difference between programming models of the two MPUs is that the A and B accumulators can be concatenated into a single double byte accumulator called the D accumulator. The two accumulators can be accessed separately or be referenced jointly by several

![Figure 1-2. MC6801 Programming Model](image-url)
new instructions. Double accumulator instructions include: Load, Store, Add, Subtract, Logical Shift Left, and Logical Shift Right. Each of these instructions utilizes the same addressing modes that are available for its analogous single accumulator instruction.

Indexing is greatly enhanced by the addition of three new instructions which interface with the Index Register. These new instructions provide the capability of adding a single unsigned byte to the Index Register (ABX), pushing the contents of the Index Register onto the stack (PSHX), and pulling the top two bytes of the stack into the Index Register (PULX).

Integer multiplication is greatly improved by the addition of a new 8-bit by 8-bit unsigned multiply instruction. The MUL instruction multiplies the two accumulators together, overwrites them with a double byte result, and executes in 10 MPU E-cycles.

Throughput improvements are achieved by reducing execution times for certain key instructions. These improvements affect all stores, the indexed addressing mode, and branches.

1.2 MC6801 SYSTEM CONFIGURATIONS

The versatility of the MC6801 is attained by offering the designer a variety of configurations which can be obtained with hardware-programming using a minimal amount of external circuitry. The three functional operating modes are:

- Single Chip,
- Expanded Non-Multiplexed, and
- Expanded Multiplexed.

The operating mode affects the configuration of two of the four MCU I/O Ports. While Ports 1 and 2 are functionally independent of the mode, Ports 3 and 4 are configured by the operating mode.

The Single Chip Mode is illustrated in Figure 1-3. In this mode, the device is totally self-reliant, uses only on-chip resources, and provides no external address or data bus. The Single Chip mode provides a maximum of 29 input/output lines including an interrupt-capable parallel I/O port with two handshake control lines. Ports 3 and 4 also function as data Input/Output ports in this mode.
In the Expanded Non-Multiplexed mode, Port 3 functions as an 8-bit bidirectional data bus while Port 4 provides up to eight data input or address bus output lines. If address outputs are selected (by writing ones to the port Data Direction Register), the MCU will provide up to eight of the least significant lines of the address bus. The eight most significant address lines are decoded internally and the resultant signal, Input/Output Select (I/O Select), provides a means for controlling an external memory space access. The expanded non-multiplexed bus will interface with M6800 family peripheral parts to directly access a maximum of 256 external locations. The MC6801 pin configuration in the Expanded Non-Multiplexed mode is shown in Figure 1-4.

![Figure 1-4. Expanded Non-Multiplexed Mode](image)

When configured in the Expanded Multiplexed mode, Port 3 provides the eight least significant lines of the address bus multiplexed with a bidirectional 8-bit data bus. Port 4 provides the remaining eight bits of the address bus. A simple latch is required to de-multiplex the Port 3 address/data bus which is controlled by the MCU signal, AS (Address Strobe). After de-multiplexing, the bus interfaces with all MC6800 Family peripheral parts. Figure 1-5 illustrates the MC6801 in the Expanded Multiplexed modes. In this configuration, the MCU can access a 64K byte memory space.

![Figure 1-5. Expanded Multiplexed Mode](image)
1.3 MODE INDEPENDENT MCU RESOURCES

Several MC6801 on-chip resources are functionally independent of the operating mode. These resources include:

- Port 1,
- Port 2,
- the Serial Communications Interface (SCI), and
- the Programmable Timer.

Port 1 is configured as an 8-bit parallel input/output port where each bit can be individually defined as an input or an output. Definition of each data port bit is accomplished by writing to the port Data Direction Register where a “1” defines an output and a “0” defines an input.

Port 2 consists of five lines which can be utilized as data input/output lines except that Port 2 bit 1 cannot be used as a data output line. If certain functions are enabled, however, up to all five Port lines are dedicated to the Programmable Timer and Serial Communications Interface. A brief overview of the SCI and Timer follows with a more detailed discussion provided in Chapters 6 and 7.

1.3.1 Serial Communications Interface (SCI)

The Serial Communications Interface (SCI) provides a full duplex capability with two formats and a variety of bit rates. External access to the SCI is provided by three of the Port 2 pins which interface with the serial transmit, receive, and bit rate clock lines.

The SCI provides two programmable formats: industry standard NRZ and Bi-Phase. Several bit rate clocking options are also provided:

- an internal clock can be utilized which divides the MPU clock frequency to obtain a set of four program selectable bit rates (convenient values depend upon judicious selection of the MPU crystal or external clock frequency),
- the SCI can be driven by an external clock, or
- the SCI can be programmed to provide a bit rate clock as an output.

The SCI register organization is shown in Figure 1-6 where the addressable registers are included in the MCU internal register area. The bit rate, clocking source, and format are controlled by the SCI Rate and Mode Control Register (“Mode”, in this case, refers to the SCI and not the MCU).

Data written to the Transmit Data Register is transferred to the transmit shift register and presented serially to the transmit pin. Serial data at the receive pin is clocked to the receive shift register and transferred to the Receive Data Register where it can be read by the MCU. Separate flags in the Transmit/Receive Control and Status Register indicate when the Transmit Data Register is empty (TDRE) and the Receive Data Register is full (RDRF). Serial data overrun and framing error protection is also provided and is indicated by the ORFE bit in the SCI Control and Status Register. A summary of the bits in the Transmit/Receive Control and Status Register is shown in Figure 1-7 and a more detailed account is provided in Chapter 6.
Figure 1-6. SCI Register Organization

Figure 1-7. Transmit/Receive Control and Status Register

RDRF — Receive Data Register Full
ORFE — Overrun or Framing Error
TDRE — Transmit Data Register Empty
RIE — Receive Interrupt Enable
RE — Receive Enable
TIE — Transmit Interrupt Enable
TE — Transmit Enable
WU — Wake Up
A "wake-up" feature allows the SCI receiver to remain passive until its line goes idle which can be interpreted as the end of the current "message". This optional feature allows an MCU to ignore the remainder of any "message" for which it is not an addressee by enabling the "wake-up" feature. It is provided as a tool which can be used, in some cases, to enhance MPU utilization in multiprocessor configurations.

1.3.2 Programmable Timer

The MC6801 includes a Programmable Timer which is functionally independent of the operating mode of the MCU. Possible timer applications include the following:
- measurement of elapsed time,
- providing an elapsed time interrupt,
- generation of an output waveform, and
- measurement of time between input signal level transitions.

The central element in the Programmable Timer is a 16-bit free running counter which is incremented by the MPU E-clock. The counter can be read by the MPU and an overflow flag (TOF) is set each time it contains all ones. One application of the free-running counter is to measure elapsed time. Other applications involve using the overflow flag (TOF) to generate a periodic interrupt every 65,536 MPU E-cycles. In addition to the counter, there are two other 16-bit registers associated with the timer as illustrated in Figure 1-8.

```
15 7    0
 ICF OCF TOF EICI EOCI ETOI IEDG OLVL

Counter High Byte | Counter Low Byte $08
Output Compare High Byte | Output Compare Low Byte $09:0A
Input Capture High Byte | Input Capture Low Byte $0B:OC

Figure 1-8. Programmable Timer
```

The Output Compare Register and the Output Level Bit (OLVL) in the Timer Control and Status Register can be utilized to control an output waveform. An interrupt, signifying an arbitrary timeout, can also be generated using the output compare function. The free-running counter is transferred to the read-only Input Capture Register whenever a proper level transition is sensed. The "proper" transition is defined by the IEDG bit in the Timer Control and Status Register. This feature can be used to measure periods or pulse widths. If both an output waveform is generated and the input capture function is used, the timer requires one external input and provides one output using two pins of Port 2.
A summary of the Timer Control and Status Register is shown in Figure 1-9. The Input Capture Flag (ICF) indicates if a transfer of the free-running counter to the Input Capture Register has occurred. The Timer Overflow Flag (TOF) is set when the free-running counter contains all ones. The Output Compare Flag (OCF) indicates if equality exists between the value in the Output Compare Register and the free-running counter. In addition, the OLVL (Output Level) bit will be clocked to an output latch whenever this occurs. The remaining bits in the Timer Control/Status register enable or disable individual interrupts associated with each status flag. Each timer interrupt uses an individual prioritized interrupt vector. If all timer interrupts are pending, they will be serviced in the following order: input capture (ICF), output compare (OCF), and timer overflow (TOF).

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>008</td>
<td></td>
</tr>
<tr>
<td>ICF</td>
<td>OCF</td>
</tr>
<tr>
<td>TOF</td>
<td>EICI</td>
</tr>
<tr>
<td>EOCI</td>
<td>ETOI</td>
</tr>
<tr>
<td>IEDG</td>
<td>OLVL</td>
</tr>
</tbody>
</table>

ICF — Input Capture Flag
OCF — Output Compare Flag
TOF — Timer Overflow Flag
EICI — Enable Input Capture Interrupt
EOCI — Enable Output Compare Interrupt
ETOI — Enable Timer Overflow Interrupt
IEDG — Input Edge
OLVL — Output Level

Figure 1-9. Timer Control and Status Register

1.4 SUMMARY OF FEATURES

When used as a single chip microcomputer or coupled with the complete spectrum of Motorola off-the-shelf peripheral parts, the MC6801 is an extremely cost-effective and powerful tool. The MC6801 MCU is an integral component in the Motorola M6800 Family of components. The Motorola tradition of excellence assures the highest standards of reliability, performance, and ease of interfacing with other parts. A summary of features of the MC6801 Single Chip Microcomputer Unit includes the following:

- Enhanced M6800 instruction set
- $8 \times 8$ multiply instruction
- Serial Communications Interface (SCI)
- Upward compatible with MC6800 object and source code
- 16-bit three function Programmable Timer
- Single chip or expandable to 64K byte address space
- Bus compatible with M6800 Family
- 2048 bytes of ROM
- 128 bytes of RAM (64 bytes retainable on powerdown)
- 29 parallel I/O and two handshake control lines
- Internal clock generator with divide-by-four output
- TTL compatible inputs and outputs
- Single +5 volt power supply
- External and internal interrupts

1-9/1-10
CHAPTER 2
OPERATING MODES AND MEMORY MAPS

2.0 INTRODUCTION

There are several possible starting points from which to begin a detailed discussion of the MC6801. Three such possibilities include:

- a description of the operating modes,
- the memory map associated with each mode, and
- a functional pin description.

This chapter discusses the MCU with respect to its operating modes and associated memory maps. The functional pin description is contained in Chapter 3. To a large degree, all three starting points treat the same general topics: only the viewpoint is different.

The MC6801 can be operated in a variety of configurations with varying types and amounts of on-chip resources. The facility which provides this extraordinary flexibility is the ability of the MCU to be hardware programmed into one of eight different operating modes.

The configuration of any particular operating mode can then be further defined using software to initialize an I/O port with respect to which of its bits are to be utilized as inputs and outputs. This is accomplished by writing a byte to the write-only Data Direction Register of a port in which 1's indicate outputs and 0's specify inputs.

As a preface to this discussion, consider a list of MCU characteristics which are affected by the operating mode. This list admittedly contains some interdependent items, but this is unimportant to our overall objective. MCU parameters which are affected by the operating mode include the following:

- number of I/O lines available,
- amount and location of addressable external memory,
- availability of on-chip RAM,
- availability of on-chip ROM,
- physical location of interrupt vectors,
- configuration of Port 3,
- configuration of Port 4,
- addressability of three associated Port 3 registers,
- addressability of two associated Port 4 registers,
- availability and type of external bus, and
- number and type of bus control signals.
It is also advantageous to know which features of the MC6801 are not affected by the operating mode and, therefore, function identically in every mode. Mode-independent features include:

- MPU (instruction execution unit),
- most of reserved register area,
- Port 1 configuration,
- Port 2 configuration,
- Serial Communications Interface (SCI), and
- Programmable Timer.

Mode characteristics are summarized in Figure 2-1 for convenient reference.

Common to All Modes:
- MPU (Instruction Execution Unit)
- Reserved Register Area
- Port 1
- Port 2
- Programmable Timer
- Serial Communications Interface

Single Chip — Mode 7
- 128 Bytes of RAM; 2048 Bytes of ROM
- SC1 is Input Strobe 3 (lS3)
- SC2 is Output Strobe 3 (OS3)
- Port 3 is a Parallel I/O Port with Two Control Lines
- Port 4 is a Parallel I/O Port

Expanded Memory Space/Non-Multiplexed Bus — Mode 5
- 128 Bytes of RAM; 2048 Bytes of ROM
- 256 Bytes of Directly Addressable External Memory Space
- SC1 is Input/Output Select (lOS)
- SC2 is Read/Write (R/W)
- Port 3 is an 8-Bit Data Bus
- Port 4 is an Input Port/Address Bus

Expanded Memory Space/Multiplexed Bus — Modes 1, 2, 3, 6
- Four Memory Space Options (64K Byte Address Space):
  (1) No Internal RAM or ROM (Mode 3)
  (2) Internal RAM, No ROM (Mode 2)
  (3) Internal RAM and ROM (Mode 1)
  (4) Internal RAM, ROM with Optional Partial Address Bus (Mode 6)
- SC1 is Address Strobe (AS)
- SC2 is Read/Write (R/W)
- Port 3 is a Multiplexed Address/Data Bus
- Port 4 is an Address Bus (Inputs/Address in Mode 6)

Test — Modes 0 and 4
- Expanded Test — Mode 0
  May be Used to Test Internal ROM and RAM
- Single Chip and Non-Multiplexed Test — Mode 4
  (1) May be Changed from Mode 4 to Mode 5
  (2) May be Used to Test Port 3 and 4 Operation

Figure 2-1. Summary of Operating Mode Characteristics
A common task in the design of an MC6801-based system is to determine the number of available I/O lines for a particular operating mode. A minimum of eight I/O lines are available using Port 1 regardless of the mode. The five lines of Port 2, however, are somewhat special. While P20 is used by the Timer input capture function, this does not prevent it from also being used for other purposes. The remaining four lines, however, are dedicated to SCI or Timer functions if these functions are enabled. If a particular line is not utilized for an SCI or Timer function, it can be used for either data input or output with one exception: Bit 1 cannot be used as a data output line. The Port 2 bits and functions include:

- Port 2 Bit 1 — Used as Timer Output (OLVL) if Port 2 DDR bit 1 is set,
- Port 2 Bit 2 — Used as Serial-Clock-Out or External-Clock-In if CC1 of Rate and Mode Control Register is set,
- Port 2 Bit 3 — Used as serial data input if RE of Transmit/Receive Data Register is set, and
- Port 2 Bit 4 — Used as serial data output if TE of Transmit/Receive Data Register is set.

While Port 2 Bit 0 can also be used for the Timer input capture function, its use is not a dedicated one. Port 2 Bit 0 can be configured as either an input or an output depending upon the state of its bit in the Port 2 Data Direction Register. The Programmable Timer input capture edge detector is a passive "listener" of this line and functions identically regardless of whether the bit is defined as an input or an output. If configured as an input, an MPU read of the Port 2 Data Register will result in reading the level at P20 regardless of whether or not the input capture function is being used.

Bit 1 of Port 2 can be used as a data input line but it cannot be used as a data output line. If its DDR bit is set, the output pin is dedicated to the output compare function output level register.

2.1 MC6801 FUNDAMENTAL MODES

The MCU can be hardware-programmed into one of eight operating modes, which are referred to numerically as modes 0 through 7. While there are eight different operating modes, there are but three fundamental ones. The remaining five may be considered variations of the fundamental modes. The three fundamental operating modes have been given the following names which correspond to the type of bus associated with each of them:

- single chip,
- expanded non-multiplexed, and
- expanded multiplexed.
2.1.1 Single Chip Mode (Mode 7)

In Single Chip Mode, illustrated in Figure 2-2, all four ports are configured as parallel input/output ports. The MCU functions as a self-contained microcomputer in this mode and has no external address or data bus. Mode dependent MCU resources for Single Chip mode include the following:

- 128 bytes of RAM,
- 2048 bytes of ROM,
- a maximum of 29 and a minimum of 24 parallel I/O lines, and
- two handshake control lines.

![Figure 2-2. Single Chip Mode](image)

Of the available 29 I/O lines, five of these lines are "shared" with the SCI and Timer. If SCI or Timer functions are enabled, however, the associated dedicated lines must be deducted from the total available I/O port lines.

In addition to a maximum of 29 data port lines, there are also two handshake control lines called IS3 and OS3. These two lines are intended for use with Port 3 but can also be used for a variety of other purposes. In addition, IS3 and OS3 allow Port 3 to be used as an 8-bit data port with handshaking capability.

2.1.2 Expanded Non-Multiplexed Mode (Mode 5)

The expanded non-multiplexed mode, illustrated in Figure 2-3, provides a modest amount of directly addressable external memory space (up to 256 bytes) while retaining significant on-chip capabilities. Mode dependent resources in the expanded non-multiplexed mode include the following:

- 128 bytes of RAM
- 2048 bytes of ROM,
- a minimum of eight I/O port lines (Port 1) and a maximum of 13 input and 12 output lines (Port 1 and Port 2) in addition to any Port 4 input lines which are not required as address outputs.
The **RESET** input configures Port 3 as an 8-bit bidirectional data bus and Port 4 as an 8-bit data input port. Any combination of the eight least significant lines of the address bus can be obtained by setting the appropriate bits in the Port 4 Data Direction Register, where Data Direction bits 0 through 7 correspond to address lines \(A_0\) through \(A_7\), respectively. Internal pullup resistors provide a logic high for Port 4 pins until software configures any desired lines as address outputs.

**NOTE**

No external address bus is provided until the Port 4 Data Direction Register has been configured.

A maximum of 256 external read/write memory locations are available in this mode. These locations reside in the MCU memory map at addresses \$100\) and \$1FF, inclusively. Up to eight external address lines (\(A_0\)-\(A_7\)) are available from Port 4 while the remaining eight lines (\(A_8\)-\(A_{15}\)) are decoded internally. The output of this internal decoder is provided as Input/Output Select (\(I_{OS}\)). The signal is active (low) whenever an address between \$0100\) and \$01FF\) is sensed on the internal address bus. The \(I_{OS}\) signal can be used in a chip select circuit for devices on the expanded non-multiplexed bus.

The expanded non-multiplexed bus is compatible with M6800 family parts. It consists of the following MCU signals which are defined in detail in Chapter 3:

- **E** (Enable),
- **D0-D7** (Data Bus),
- **A0-A7** (Address Bus),
- **R/W** (Read/Write),
- **I_{OS}** (Input/Output Select),

Figure 2-4 illustrates a typical system configuration using the expanded non-multiplexed mode.
2.1.3 Expanded Multiplexed Mode (Modes 1, 2, 3, 6)

In the expanded multiplexed modes, a 16-bit address bus is provided and the MCU can address the entire 64K byte address space. These modes offer a large memory space while providing several significant on-chip resources. The following mode dependent MCU resources are available in the expanded multiplexed modes:

- 128 bytes of RAM (Modes 1, 2 and 6)
- 2048 bytes of ROM in Mode 6 and 2032 bytes in Mode 1 (External vectors replace the most significant 16 bytes of ROM in Mode 1)
- any unused address outputs (A8-A15) may be used as data input lines (Port 4 — Mode 6 only)

The MCU configuration for the expanded multiplexed modes is illustrated in Figure 2-5. Port 3 is configured as a multiplexed 8-bit address and data bus in all expanded multiplexed modes. The least significant eight bits of address (A0-A7) are multiplexed with the entire data bus (D0-D7). A simple external latch is required to de-multiplex the two buses. In addition, an MCU bus timing signal called Address Strobe (AS) is provided to control the latch.

Port 4 provides the eight most significant lines of the address bus for all expanded multiplexed modes except Mode 6. For this exception, Port 4 is configured from Reset as an 8-bit parallel data input port. By using software, however, the configuration can be changed to provide any combination of the eight most significant lines of the address bus (A0-A7). This is accomplished by setting the appropriate bits in the Port 4 Data Direction Register. Bits 0 through 7 of the Data Direction Register correspond to address lines A8 through A15, respectively.

NOTE
The eight most significant address lines are not provided in Mode 6 until the Port 4 Data Direction Register has been configured.

Figure 2-4. System Configuration — Expanded Non-Multiplexed Mode
This configuration allows unused Port 4 address lines to be used for additional data input lines. Internal pull-up resistors provide a logic high for the eight most significant address lines until software configures the port.

The expanded multiplexed bus is compatible with the M6800 family and consists of the following signals which are defined in detail in Chapter 3:
- E (Enable)
- A0/D0-A7/D7 (Multiplexed Address and Data)
- A8-A15 (Address Bus)
- AS (Address Strobe)
- R/W (Read/Write)

A typical system configuration using the expanded multiplexed mode is illustrated in Figure 2-6. This configuration requires an 8-bit latch to de-multiplex the address and data bus to interface with standard M6800 family parts. The MCU signal, Address Strobe (AS), is used to control the demultiplexing latch.

Figure 2-5. Expanded Multiplexed Mode

Figure 2-6. System Configuration for Expanded Multiplexed Modes

2-7
2.1.4 Test Modes (Modes 0 and 4)

While not fundamentally different than the other six modes, the two MC6801 Test Modes provide a means for testing selected features of the MCU. It should be noted that many features of the MCU can be tested in one of the expanded modes for which a large external memory space is available for the test program. Mode 4, however, provides the only configuration for testing Ports 3 and 4 in the single chip and expanded non-multiplexed modes without a ROM-resident "self-check" program.

The Mode 4 test sequence is facilitated by first loading a small program into the MCU internal RAM using Modes 0, 1, or 2. The MCU is then Reset and re-programmed into Mode 4. In this mode, the ROM is excluded from the MCU internal memory map and the eight most significant bits of the RAM address decoder are treated as "don't cares". This results in the Reset vector being fetched from the two most significant bytes of the RAM. Presumably, this is a vector to the start of the RAM-resident test program. The RAM is addressable from $XX80 to $XXFF in Mode 4 and all of the single chip resources are accessible except the ROM.

Mode 5 can be entered from Mode 4 without having to Reset the MCU. If the PC0 bit in the Port 2 Data Register (bit 5) is set while in Mode 4, the mode will irreversibly change to Mode 5. This mechanism is intended to be used for testing purposes.

The remaining test mode, Mode 0, is a variation of the expanded multiplexed mode. Two significant features of this mode make it particularly suitable for testing purposes: (1) the Reset vector is decoded as external memory space for only the first two E-cycles after RESET goes high, and (2) data read during internal MPU reads will appear on the Port 3 external Data Bus while E (Enable) is high.

The significance of the first feature is that an external program can be used to obtain control of the MCU from Reset and all other references to the interrupt vector area (such as MPU reads) will access internal ROM. This characteristic provides a method for reading the entire ROM including all of the interrupt vectors from a program which resides in external memory. The latter feature allows the internal data bus to be monitored with automated test equipment.

A memory map restriction must be observed when operating in Mode 0. No peripheral device can be enabled to the data bus as a response to any address in the MCU internal memory map. If this restriction is not observed, electrical damage can occur due to data bus contention.

2.2 MODE ASSOCIATED MEMORY MAPS

Another viewpoint from which to examine the operating modes of the MC6801 is the memory map associated with each mode. A memory map for each mode is depicted in Figures 2-7 through 2-16. Significant details associated with each operating mode are included with each of the maps for convenient reference.

Hatched areas on the maps depict internal addresses while open areas refer to available external memory space. Unusable addresses are designated as such on the applicable memory maps.

Most of the information presented in the memory maps has been previously mentioned. Two aspects, however, have not been discussed and are worthy of special mention: (1) several notes refer to addresses in the internal register area which are excluded in several modes, and (2) there is a variant mode for Mode 1 and Mode 6 called Mode 1R and Mode 6R, respectively. These two aspects are discussed in the next two sections.
Notes:

(1) Excludes the following addresses which may be used externally: $04, $05, $06, $07 and $0F.
(2) Addresses $FFFE and $FFFF are considered external if accessed within 2 cycles after a positive edge of \texttt{RESET} and internal at all other times.
(3) After 2 MPU cycles, there must be no overlapping of internal and external memory spaces to avoid driving the data bus with more than one device.
(4) This mode is the only mode which may be used to examine the interrupt vectors in internal ROM using an external \texttt{RESET} vector.
(5) Assumes RAME (RAM Enable bit) is set.

\textbf{Figure 2-7. Memory Map for Mode 0}
Notes:

1. Excludes the following addresses which may be used externally: $04, $05, $06, $07, and $0F.
2. Internal ROM addresses $FF0 to $FFFF are not usable.
3. Assumes RAME (RAM Enable bit) is set.

Figure 2-8. Memory Map for Mode 1
Notes:

(1) Excludes the following addresses which may be used externally: $04, $05, $06, $07, and $0F.
(2) Starting addresses for the internal ROM may be $C800,$D800 or $E800 as a mask option.
(3) Assumes RAME (RAM Enable bit) is set.

Figure 2-9. Memory Map for Mode 1R
Notes:

(1) Excludes the following addresses which may be used externally: $04, $05, $06, $07, and $0F.
(2) Assumes RAME (RAM Enable bit) is set.

Figure 2-10. Memory Map for Mode 2
MC6801 MODE 3
MULTIPLEXED/NO RAM OR ROM

Notes:

(1) Excludes the following addresses which may be used externally: $04, $05, $06, $07, and $0F.

Figure 2-11. Memory Map for Mode 3
Notes:
(1) The internal ROM is disabled.
(2) Mode 4 may be changed to Mode 5 without having to assert RESET by writing a "1" to bit 5 (PC0) of Port 2 Data Register.
(3) Addresses A8 to A15 are treated as "don't cares" to decode internal RAM.
(4) Internal RAM will appear as $XX80 to $XXFF.
(5) MPU reads of Port 3 Data Direction Register will access Port 3 Data Register instead.

Figure 2-12. Memory Map for Mode 4
Notes:
(1) Excludes the following addresses which are decoded as external: $04, $06, $0F.
(2) This mode may be entered without going through Reset by using Mode 4 and subsequently writing a "1" into the PC0 bit of Port 2 Data Register.
(3) Address lines A0-A7 will not contain addresses until the Data Direction Register for Port 4 has been written with 1's in the appropriate bits. These address lines will provide 1's until software configures Port 4 Data Direction Register.
(4) This area cannot be directly addressed or written.
(5) Assumes RAME (RAM Enable bit) is set.

Figure 2-13. Memory Map for Mode 5
Notes:

(1) Excludes the following addresses which may be used externally: $04, $06, $0F.

(2) Address lines A8-A15 will not contain addresses until the Data Direction Register for Port 4 has been written with “1”s in the appropriate bits. These address lines will assert “1”s until made outputs by writing the Data Direction Register.

(3) Assumes RAME (RAM Enable bit) is set.

Figure 2-14. Memory Map for Mode 6
Notes:

(1) Excludes the following addresses which may be used externally: $04, $06, $0F.

(2) Address lines A8-A15 will not contain addresses until the Data Direction Register for Port 4 has been written with "1"s in the appropriate bits. These address lines will assert "1"s until made outputs by writing the Data Direction Register.

(3) Starting addresses for the internal ROM may be $C800, $D800 or $E800.

(4) Assumes RAME (RAM Enable bit) is set.

Figure 2-15. Memory Map for Mode 6R
Notes:

(1) MPU reads of Port 3 Data Direction Register will access Port 3 Data Register instead.
(2) Assumes RAME (RAM Enable bit) is set.

Figure 2-16. Memory Map for Mode 7
2.2.1 Internal Register Area Exclusions

The internal register area is present in every MCU memory map and consists of the registers shown in Figure 2-17. From this map, it should be noted that all four ports have an assigned Data Direction Register and a Data Register. Port 3 has one additional register which is used to define its configuration in the Single Chip Mode: the Port 3 Control and Status Register.

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Port 1 Data Direction Register</td>
</tr>
<tr>
<td>01</td>
<td>Port 2 Data Direction Register</td>
</tr>
<tr>
<td>02</td>
<td>Port 1 Data Register</td>
</tr>
<tr>
<td>03</td>
<td>Port 2 Data Register</td>
</tr>
<tr>
<td>04*</td>
<td>Port 3 Data Direction Register</td>
</tr>
<tr>
<td>05**</td>
<td>Port 4 Data Direction Register</td>
</tr>
<tr>
<td>06*</td>
<td>Port 3 Data Register</td>
</tr>
<tr>
<td>07**</td>
<td>Port 4 Data Register</td>
</tr>
<tr>
<td>08</td>
<td>Timer Control and Status Register (TCSR)</td>
</tr>
<tr>
<td>09</td>
<td>Counter Register (MSB)</td>
</tr>
<tr>
<td>0A</td>
<td>Counter Register (LSB)</td>
</tr>
<tr>
<td>0B</td>
<td>Output Compare Register (MSB)</td>
</tr>
<tr>
<td>0C</td>
<td>Output Compare Register (LSB)</td>
</tr>
<tr>
<td>0D</td>
<td>Input Capture Register (MSB)</td>
</tr>
<tr>
<td>0E</td>
<td>Input Capture Register (LSB)</td>
</tr>
<tr>
<td>0F*</td>
<td>Port 3 Control and Status Register</td>
</tr>
<tr>
<td>10</td>
<td>SCI Rate and Mode Control Register (RMCRI)</td>
</tr>
<tr>
<td>11</td>
<td>Transmit/Receive Control and Status Register</td>
</tr>
<tr>
<td>12</td>
<td>SCI Receiver Data Register</td>
</tr>
<tr>
<td>13</td>
<td>SCI Transmitter Data Register</td>
</tr>
<tr>
<td>14</td>
<td>RAM Control Register</td>
</tr>
<tr>
<td>15-1F</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

* External addresses in all modes except Modes 4 and 7
** External address in Modes 0, 1, 2, and 3

Figure 2-17. MCU Internal Register Area

When a port functions as an I/O port, the associated Data Direction Register is used to define which bits are configured as inputs or outputs. All of the MCU Data Direction Registers are cleared during Reset, which configures all data port lines as inputs. Any particular bit can be changed to an output, however, by setting its corresponding bit in the Data Direction Register. When configured as a data port, the Data Register allows the programmer to directly access its associated pins.

In those operating modes that utilize Ports 3 and 4 as dedicated address or data buses and not as data ports, the addresses of the registers associated with them are decoded by the MCU as external addresses. For example, in modes 0 through 3, Ports 3 and 4 provide the address and data bus. In these modes, the locations reserved for the Port 3 Data Register ($06), Data Direction Register ($04), and Control and Status Register ($0F), are decoded as external addresses. Similarly, the Port 4 Data Register ($07) and Data Direction Register ($05) are also decoded as external memory locations. This feature allows external hardware to emulate Ports 3 and 4 as data ports. An expanded multiplexed mode is used and the emulated ports respond correctly to their associated register addresses.
2.2.2 Relocatable ROM Options: Modes 1R and 6R

The internal ROM can be relocated by a mask option from $F800-$FFFF to one of the following address ranges:

- $C800-$CFFF,
- $D800-$DFFF, or
- $E800-$EFFF.

As shown in Figures 2-9 and 2-15, the physical location of the interrupt vector area associated with each mode, however, is not changed. The net result is that the relocated ROM option only has applications value in two Modes: 1 and 6. The modes are labeled as 1R and 6R in the memory maps. In Mode 1R, one effect of this change is to reclaim the 16 bytes of internal ROM that could not be accessed in Mode 1 due to the external interrupt vector area. In Mode 6R, the value of the option is to obtain both the use of external interrupt vectors and the ability to configure the Port 4 unused address lines as data input lines. If planning to use the MC68701 EPROM as a tool in the prototype phase, the reader should check the current Data Sheet to determine if it supports the “R” option.

2.3 PROGRAMMING THE MODE

Having discussed the characteristics associated with each of the MCU operating modes, it is now appropriate to briefly describe how the MCU is programmed into a given mode. The following remarks are intended to provide a general overview of how this is accomplished. Circuit details are contained in the discussion for the RESET pin in Chapter 3.

The MC6801 operating mode is controlled by the levels present at pins 8, 9, and 10 during the rising edge of RESET. These same three pins, however, also function as the least three significant bits of Port 2. The operating mode is latched into the MCU Program Control Register on the rising edge of RESET after which time the levels can be removed and the pins used for other purposes. The operating mode can be read from the Port 2 Data Register where the values PC0 (Pin 8), PC1 (Pin 9), and PC2 (Pin 10) appear as data bits D5 through D7, respectively.

The Program Control register may be considered a read-only register with a single exception: Mode 4 will be irreversibly changed to Mode 5 if a “1” is written to the PC0 bit in the Port 2 Data Register. This feature is included for testing purposes and provides a mechanism to enable changing from Mode 4 to Mode 5 without having to Reset the MCU.

2.4 MC6801 COMPARISONS

Having concluded the discussion of the MCU operating modes, it is considered appropriate to contrast the MC6801 with several other M6800 family parts. This will enable the reader to see how the MCU has been integrated with other parts in the family. For a more detailed and current description of any particular part, the reader is referred to the appropriate Data Sheet.

*An additional mask option allows for partially decoding the address bus by making address lines, A13-A12, "don't cares."
2.4.1 MC6800 Bus Comparison

While the two types of MC6801 external buses are compatible with M6800 family parts, it is not a signal-for-signal replacement for the MC6800 MPU.

The E (Enable) output of the MC6801 is similar and functionally equivalent to the MC6800 φ2 clock signal. The φ1 clock signal, however, is derived internally and is not provided as an output.

The MC6800 VMA (Valid Memory Address) signal is not provided by the MCU and is not required in MC6801 systems. VMA is used in MC6800 systems to indicate bus cycles during which an invalid address could exist on the address bus. In MC6801 systems, these unused bus cycles are replaced with those which produce an MPU read of address $FFFF. Because this corresponds to the Reset vector and the R/W (Read/Write) line is forced high (read), the VMA signal is not required.

The MC6800 Three State Control (TSC), Data Bus Enable (DBE), HALT, and Bus Available (BA) signals have no counterpart in the MC6801.

The MC6801 expanded multiplexed data bus differs somewhat from its MC6800 predecessor. The MCU multiplexed data bus exists only when E (Enable) is high. When E is low, the lines are part of the address bus. Peripheral parts interfacing with this data bus must be controlled such that they are enabled onto the bus when E (Enable) is high and are removed at all other times. If this timing is not observed, peripheral parts could be enabled to the data bus while the MCU is supplying address.

2.4.2 Comparison with MC6803

The MC6803 is functionally identical to an MC6801 which is limited to selected modes. The MC6803 can be considered an MC6801 operating in Modes 2 or 3. Operation in modes other than these (excluding Mode 0) is undefined. In Mode 0, the contents of the ROM is undefined.

2.4.3 Comparison with MC68701

One application for the MC68701 is as a tool in the prototype phase for MC6801-based systems which utilize the internal ROM. It is nearly a pin-for-pin surrogate for the MC6801 with its most distinctive feature being the substitution of the internal ROM with a 2048-byte EPROM. The EPROM offers the designer a convenient tool with which to develop and test a ROM-resident program before committing it to production. Minor differences between the MC6801 and the MC68701 include the following:

- the ROM has been replaced with a 2048-byte EPROM,
- the interrupt vector space has been modified in Mode 0,
- the RESET pin is electrically different, and
- two bits have been added to the RAM Control Register to support programming of the EPROM.

A description of the MC68701 is presented in Appendix E.
2.4.4 Comparison with MC68120

The MC68120 Intelligent Peripheral Controller (IPC) is a general purpose mask programmable, 48-pin, single-chip or expandable peripheral controller. The IPC provides the interface between an MC68000, MC6800, MC6801, or MC6809 microprocessor and a peripheral device through a system bus, shared dual-ported RAM, and associated control lines. The MC68120 is architecturally similar to the MC6801 with many additional features which enhance its operation as a peripheral controller. Software for the MC68120 is both source and object code compatible with the MC6801. Features of the MC68120 include the following:

- Local Bus Compatible with the M6800 and M68000 Family
- 128 Bytes of Dual-Ported RAM
- Six Semaphore Registers
- System Bus Interface Compatible with M6800 and M68000 Families
- Single Chip or Expandable to 64K Byte Address Space
- 21 Parallel I/O and Two Handshake Control Lines
- Serial Communications Interface (SCI)
- 16-Bit Three Function Programmable Timer
CHAPTER 3
FUNCTIONAL PIN DESCRIPTION

3.0 INTRODUCTION

While the first two chapters were written for the system designer, this discussion is intended primarily for the system implementor. When supplemented with a current MC6801 Data Sheet, it is sufficiently detailed to support circuit design of MC6801-based systems.

A diagram of the MC6801 40-pin package is shown in Figure 3-1 and a functional block diagram is illustrated in Figure 3-2. Twenty-nine pins are organized as three 8-bit ports and one 5-bit port. Each port consists of at least a Data Register, a write-only Data Direction Register, and an output driver. The Data Direction Register is used to define whether corresponding bits in the Data Register are configured as an input (clear) or output (set). Port pins are labeled as Pij where i identifies the port and j indicates the particular bit.

The terms "I/O port" or "data port" have a very specific meaning when applied to the MC6801. When the port is used as a "data port" or, equivalently, "I/O port," it is controlled by its Data Direction Register and the programmer has direct access to its pins using the port associated with the Data Register. Address and data buses and associated control lines are never referred to simply as "I/O" or "data" in this discussion.

<table>
<thead>
<tr>
<th>VSS</th>
<th>1</th>
<th>40</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTAL1</td>
<td>2</td>
<td>39</td>
<td>SC1</td>
</tr>
<tr>
<td>EXTAL2</td>
<td>3</td>
<td>38</td>
<td>SC2</td>
</tr>
<tr>
<td>NMI</td>
<td>4</td>
<td>37</td>
<td>P30</td>
</tr>
<tr>
<td>IRQ01</td>
<td>5</td>
<td>36</td>
<td>P31</td>
</tr>
<tr>
<td>RESET</td>
<td>6</td>
<td>35</td>
<td>P32</td>
</tr>
<tr>
<td>VCC</td>
<td>7</td>
<td>34</td>
<td>P33</td>
</tr>
<tr>
<td>P20</td>
<td>8</td>
<td>33</td>
<td>P34</td>
</tr>
<tr>
<td>P21</td>
<td>9</td>
<td>32</td>
<td>P35</td>
</tr>
<tr>
<td>P22</td>
<td>10</td>
<td>31</td>
<td>P36</td>
</tr>
<tr>
<td>P23</td>
<td>11</td>
<td>30</td>
<td>P37</td>
</tr>
<tr>
<td>P24</td>
<td>12</td>
<td>29</td>
<td>P40</td>
</tr>
<tr>
<td>P10</td>
<td>13</td>
<td>28</td>
<td>P41</td>
</tr>
<tr>
<td>P11</td>
<td>14</td>
<td>27</td>
<td>P42</td>
</tr>
<tr>
<td>P12</td>
<td>15</td>
<td>26</td>
<td>P43</td>
</tr>
<tr>
<td>P13</td>
<td>16</td>
<td>25</td>
<td>P44</td>
</tr>
<tr>
<td>P14</td>
<td>17</td>
<td>24</td>
<td>P45</td>
</tr>
<tr>
<td>P15</td>
<td>18</td>
<td>23</td>
<td>P46</td>
</tr>
<tr>
<td>P16</td>
<td>19</td>
<td>22</td>
<td>P47</td>
</tr>
<tr>
<td>P17</td>
<td>20</td>
<td>21</td>
<td>VCC</td>
</tr>
<tr>
<td>Standby</td>
<td>21</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3-1. MC6801 Pin Diagram
Figure 3-2. MC6801 Block Diagram
An "MPU read" or "MPU write" is defined as any access of a particular location with the R/W (Read/Write) line high or low, respectively. Examples of instructions which perform an MPU read include LDAA, TST, and ORAB. Examples of MPU write instructions include STAA and STX. Both an MPU read and write are performed in such "read-modify-write" instructions as NEG, INC, and COM.

Of the 40 MCU pins, 22 function identically in all eight operating modes. These pins include all but those 18 illustrated on the far left of Figure 3-2. The configuration of the 18 pins (Ports 3 and 4, SC1, SC2) depends upon the operating mode. The organizational scheme of this chapter is to first present descriptions for each of the 22 mode independent pins. The remaining 18 pins are then discussed according to their relationship to each of the three MCU fundamental operating modes. Some repetition is necessarily introduced with this scheme. It is intended, however, that the reader be required to reference only the material pertaining to the intended mode of operation.

Not all of the remaining details of the MC6801 are discussed in this chapter. The excluded topics constitute the basis for the remainder of this manual and include:
- the instruction set,
- the interrupt structure,
- the Serial Communications Interface (SCI), and
- the Programmable Timer

One aspect of the following discussion is unique to Motorola microprocessor documentation. Logic diagrams for each of the four MCU ports are included as part of the presentation. It is recognized that these diagrams might not be understood by all readers and those could be left with the impression that a critical part of the discussion has been missed. No critical information is presented solely by the use of logic diagrams and, therefore, they can be totally ignored without serious consequences.

The logic diagrams have been included (1) to provide insight as to why the MCU functions as it does, (2) to provide a method to obtain non-critical information about topics not considered to be of sufficient interest to include in the discussion, and (3) to reinforce the explanations. Because Port 1 is discussed prior to other ports and is common to all modes, its logic diagram is described in greater detail. Some aspects of the Port 1 logic diagram and description are not repeated in the discussion for the remaining ports.

Finally, one last aspect of this chapter must be mentioned. A current MC6801 Data Sheet is intended to summarize and supplement this discussion. The symbols used in this chapter are defined quantitatively in this document. This chapter provides amplification and explanation for most material presented in the MC6801 Data Sheet.
3.1 MODE INDEPENDENT PINS

Twenty-two of the 40 MCU pins are not affected by the operating mode and thus function identically in all modes. These mode independent pins include:

- XTAL1 and EXTAL2
- VCC, VSS, VCC Standby
- NMI
- P10-P17 (Port 1)
- E
- RESET
- IRQI
- P20-P24 (Port 2)

The function of these 22 pins is discussed in the following sections. It should be remembered that they are applicable to every MC6801-based system regardless of its operating mode.

3.1.1 XTAL1 and EXTAL2: MCU Clock Inputs

The XTAL1 and EXTAL2 pins are used to drive the MCU internal clock generator and produces two clock signals. The two clock input pins can be driven by either of two different types of devices:

- a quartz crystal resonator, or
- a TTL-compatible external clock source.

The MCU internal clock generator consists of an oscillator synchronized with an external crystal or other reference and has a divide-by-four circuit in the output. The output is provided as the E (Enable) signal. Two non-overlapping clocks are derived from it and used as the primary MCU clocks. The internal clock, $\phi_2$, is in phase with E (Enable). The division-by-four circuit facilitates driving the MCU with higher frequency components which are usually less expensive.

Considerations involved in the selection of a particular external clock frequency include:

- it cannot exceed the recommended operating frequency (either $4f_0$ or $f_{XTAL}$), and
- if a particular SCI baud is desired when using the internal bit rate generator, the external frequency must be chosen to produce the desired rate.

If a crystal is used to drive the MCU, it should be manufactured with an AT cut, operated in the parallel resonance mode, and have a fundamental frequency within the range specified for $f_{XTAL}$. Note that the internal divide-by-four circuitry allows use of the inexpensive standard 3.58 MHz or 4.4336 MHz color burst TV crystals. A capacitor must be connected between each crystal pin and ground to ensure reliable startup and operation.

The crystal and capacitors should be mounted as close to the XTAL1 and EXTAL2 pins as possible and drive only the MCU. This will minimize output distortion and startup stabilization time. Nominal recommended crystal specifications are shown in Figure 3-3; however, a MC6801 data sheet should be consulted for more details. The MCU is compatible with many commercially available quartz crystal resonators.
(a) Nominal Recommended Crystal Parameters

See Data Sheet for capacitor value.

(b) Oscillator Stabilization Time ($t_{RC}$)

MC6801 Nominal Crystal Parameters

<table>
<thead>
<tr>
<th></th>
<th>3.58 MHz</th>
<th>4.00 MHz</th>
<th>5.0 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td>60 Ω</td>
<td>50 Ω</td>
<td>30-50 Ω</td>
</tr>
<tr>
<td>$C_0$</td>
<td>3.5 pF</td>
<td>6.5 pF</td>
<td>4.8 pF</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.015 pF</td>
<td>0.025 pF</td>
<td>0.01-0.02 pF</td>
</tr>
<tr>
<td>Q</td>
<td>&gt;40 k</td>
<td>&gt;30 k</td>
<td>&gt;20 k</td>
</tr>
</tbody>
</table>

*Note: These are representative AT-cut crystal parameters only. Crystals of other types of cuts may also be used.

Figure 3-3. MC6801 Recommended Crystal Parameters
The MCU can also be driven by an external TTL-compatible clock source. In this configuration, the clock source is connected to the EXTAL2 input and the XTAL1 input must be tied to ground. The external clock source frequency must be in the range specified for $4f_0$ with a duty cycle of 50% ($\pm 10\%$)*.

Consideration must be given to the time required for the MCU clock generator to stabilize during powerup when designing the Reset circuit. RESET must be kept below 0.8 volt until the clock generator has stabilized. For the recommended crystal, the time is $t_{RC}$ after $V_{CC}$ reaches 4.75 volts as shown in Figure 3-3.

If an external clock source is used, it is possible to "stretch" the MCU clock input (i.e., temporarily extend the period) but certain operating limitations must be observed. Typically, this is only desirable when operating in the expanded multiplexed modes and interfacing with slow memories, refreshing dynamic memories, or performing multiplexed DMA transfers. Design considerations include:

1. the "stretched" half-cycle must be synchronized with E,
2. duty cycle limitations must be observed,
3. no half-cycle can be stretched beyond one-half of $t_{cyc}$ (maximum),
4. the Programmable Timer is clocked by E and will be affected by altering the MCU input frequency, and
5. the SCI internal bit rate generator is clocked by the Programmable Timer and serial operations can be adversely affected by altering the MCU input frequency.

3.1.2 E: MCU Clock Output

The E (Enable) clock is provided as a timing signal to synchronize Data Bus transfers. An "MPU E-cycle" (or bus cycle) consists of a negative half-cycle of E followed by a positive half-cycle. For any given bus cycle, address will become valid during the negative half-cycle of E and the selected device must be enabled to the Data Bus during the next positive half-cycle. The data bus is active only while E is high or, equivalently, "during E." Specific details of bus timing are discussed in sections 3.3.5 and 3.4.5.

Enable (E) is the primary MCU system timing signal and all timing data specified as cycles with respect to this clock unless otherwise noted. The frequency of E is equivalent to the MCU input frequency divided by four. Due to propagation delays in the clock generator, however, some skew will exist between the external clock source or crystal and E. Enable (E) is derived from alternating negative edges of the MCU input clock and is unaffected by the clock duty cycle. The Enable output will drive one Schottky TTL load and 90 pF and is functionally equivalent to an M6800 system $\phi 2$ clock.

*A more detailed discussion of how the duty cycle affects the expanded multiplexed bus timing is included in Appendix I.
3.1.3 **RESET**

The Reset function is used for three primary purposes in an MC6801 system:
1. to provide an orderly and defined startup of MPU activity from a powerdown condition,
2. to return a system to startup conditions without an intervening powerdown condition, and
3. as a control signal to latch the operating mode.

In this discussion, a distinction is made between the MCU input signal, **RESET**, and the internal Reset state by using the two preceding forms of capitalization. The bar (or overscore) indicates that the signal is active when pulled low.

When **RESET** is pulled low, execution of the current instruction is aborted and the MPU enters a "Reset state." No registers are pushed onto the stack and their contents are not defined while in this state.

While **RESET** is held low, instruction execution is suspended and other MCU activity includes the following:
- the MPU I-bit is set which masks (disables) both IRQ1 and IRQ2 interrupts;
- the NMI interrupt latch is cleared which effectively disregards NMI interrupts occurring while the MPU is held in Reset;
- the E (Enable) clock is active;
- all Data Direction Registers are cleared;
- the SCI Rate and Mode Control Register is cleared;
- the SCI Transmit/Receive Control and Status Register is preset to $20;
- the Receive Data Register is cleared;
- the Timer Control and Status Register is cleared;
- the free-running Counter is cleared;
- the buffer for the LSB of the Counter and output level register are cleared;
- the Output Compare Register is preset to $FFFF;
- the Port 3 Control and Status Register is cleared;
- Ports 1, 2 and 3 are forced to the high impedance state;
- Port 4 is also held in a high impedance state but internal pull-up resistors are provided to pull the lines high;
- SC1 is held high in a high impedance state with an internal pull-up resistor if the inputs to P20, P21, and P22 indicate the Single Chip modes; otherwise it is actively held high;
- SC2 is actively held high.

Note that only SC1 is affected by levels on the mode programming pins while **RESET** is held low and will be configured by the levels present on pins 8, 9, and 10. Finally, it should be noted that the MCU Data Registers are NOT cleared by Reset.

When a positive edge of **RESET** is detected, the MCU will latch the operating mode and complete its Reset sequence. This consists of configuring Port 3, Port 4, SC1, and SC2; and fetching a vector (address) from locations $FFFE and $FFFF. The physical location of this vector is defined by the operating mode of the MCU and can reside in either internal or external memory space. After the Reset vector has been fetched, it is transferred to the Program Counter and instruction execution begins at this location.
Reset timing is illustrated in Figure 3-4 where the buses shown refer to the MCU internal buses. External bus activity, however, is mode dependent. RESET is internally synchronized with E and requires a setup time of tPCS prior to the negative edge of E to be recognized on the following negative edge.

From a powerdown condition, RESET must be held below 0.8 volt for a period sufficient to allow the MCU clock generator to stabilize. The time required is dependent on the characteristics of the external clocking device. For typical crystals, the time is not more than tRC and begins when VCC reaches 4.75 volts as shown in Figure 3-4. In addition to the oscillator stabilization time, RESET must be held below 0.8 volt until VCC Standby has reached 4.75 volts.

If the MCU is already in a powerup condition, the oscillator stabilization time is no longer a factor and RESET must be held below 0.8 volt for at least three E-cycles to complete its entire internal Reset sequence. An external RC-network can be used to obtain the required timing.

The MCU operating mode is latched from three pins of Port 2 on the positive edge of RESET. The levels should be provided on pins 8, 9 and 10 while RESET is low to configure SCI. They must be valid, however, for setup time, tMPS, before RESET goes high and — if the rise time is less than that specified in the data sheet — must remain so for hold time tMPH. If a diode is used to program the mode, a voltage differential of VMPDD is provided between a logic low (VMPL) and the RESET voltage at the point when the mode logic level is latched. The diode forward voltage drop (Vf) must not exceed VMPDD minimum. While RESET is high, the three pins are not used for mode selection. Mode programming timing and voltage level requirements are shown in Figure 3-5 and it should be noted that the programming voltage levels are not the same as VIL and VIH. The operating mode can be read with software from bits 5-7 in the Port 2 Data Register where bit 7 corresponds to the level latched from pin 10.

Circuitry for programming the mode typically requires only RESET as a control signal while component selection depends primarily on the normal system usage of Port 2 bits 0, 1, and 2. If the pin(s) are unused, they can be tied high or low; however, care must be taken in software to ensure that they are not unintentionally defined as outputs. If the pin(s) normally provide outputs, they can be programmed using a diode arrangement as shown in Figure 3-6. The diodes provide a path to ground while RESET is low to program a low level and prevents those pins from pulling RESET low during normal operation.

If the pin(s) are normally used as inputs, it is typically necessary to employ devices which electrically isolate them until after the mode has been programmed. The circuit shown in Figure 3-7 uses MC14066B bidirectional analog switches to isolate the pin(s) while diodes and pullup resistors are used to provide the programming levels. External circuitry must ensure that the mode input levels do not change prior to hold time, tMPH. In the circuit shown in Figure 3-7, the hold time is met by the relatively long enable time of the MC14066B analog switches.

Finally, one must consider pin(s) used bidirectionally in normal operation. In this case, three-state buffers are most likely already included to handle the bidirectionality of the pin. To program the mode, the buffers should be controlled by RESET to force isolation of the pins while the programming levels are presented and provide for the required setup and hold times.
Figure 3-4. RESET Timing

Figure 3-5. Mode Programming Levels and Timing
Notes:
1. Mode 7 as shown
2. $R_2\cdot C$ = Reset time constant
3. $R_1 = 10\, \text{k}$ (typical)
4. $D$ = See Data Sheet for diode type.

Figure 3-6. Programming the Mode with Diodes

Notes:
1. Mode 7 is shown (switches open)
2. $R = 4.7\, \text{k}\, \text{ohm}$ (typical)
3. $R_1 = 10\, \text{k}$ (typical)
4. $C = 50\, \mu\text{F}$ (typical)

Figure 3-7. Programming the Mode with Analog Switches and Diodes
3.1.4 VCC and VSS: MCU Power

Power is provided to a large portion of the MCU using the VCC and VSS pins where VSS is tied to ground. The remaining power is provided by VCC Standby. The power supply must provide +5 volts (±5%) for VCC. Total MCU power dissipation (including VCC Standby) will not exceed PD milliwatts.

In some applications, it is desirable to design a large RC time constant into the VCC power supply. This provides time to implement an orderly powerdown sequence which detects the drop in level, generates NMI, and executes a short powerdown subroutine. Circuitry must be designed to keep RESET below 0.8 volt until VCC reaches 4.75 volts plus an additional period, tRC, to allow the clock generator to stabilize.

3.1.5 VCC Standby: RAM Standby Power

The VCC Standby pin provides power to the standby portion of the RAM, and the STBY PWR and RAME bits of the RAM Control Register. The standby portion of the RAM consists of 64 bytes located from $80 through $BF. The power source should provide VSB volt during powerup operation while during powerdown it must in the range specified for VSBB. The maximum current in powerdown (standby) operation will not exceed ISBB for 64 bytes of RAM.

NOTE

Power must be supplied to VCC Standby if the internal RAM is to be used regardless of whether standby power operation is anticipated. In Mode 3, VCC Standby should be tied to ground.

If the standby power source has a larger RC time constant than VCC during powerup, then the Reset circuit must be designed to keep RESET below 0.8 volt until the standby power source has reached VSB volts.

3.1.5.1 STANDBY POWER OPERATIONS. When standby power operation is anticipated, there are two vital issues of concern to the system designer:

- how can the standby RAM be protected from spurious writes as control of the MPU is lost during the powerdown sequence, and
- how can it be determined whether the RAM was adequately powered after a period of standby operation?

The RAM Enable (RAME) and Standby Power (STBY PWR) bits of the RAM Control Register provide assistance with these problems.

3.1.5.1.1 RAM Enable (RAME) Bit. Two of the bits contained in the RAM Control Register ($14) are functionally independent bits which control accesses and provide the status of the standby RAM. The RAM Enable (RAME) bit is used to remove the entire internal RAM from the MCU internal memory map. This "removal" is effectively accomplished by deleting it from the MCU internal memory address decoder. When "removed," references to the RAM addresses ($80 to $FF) are decoded as external. Data in the RAM, however, will remain intact while the RAM is "disabled," provided adequate power is maintained to VCC and VCC Standby.
The RAME bit is both readable and writeable. It is set (enabled) during the Reset sequence if \( V_{CC} \) Standby is above 4.75 volts when the positive edge of \( \text{RESET} \) reaches 4.0 volts. If RAME is cleared, the entire internal RAM is disabled (i.e., deselected) and all subsequent accesses to the internal RAM reference external memory. If the RAM Enable bit is set (and not in Mode 3), the RAM is included in the MCU internal memory space.

It is intended that RAME be cleared during execution of a powerdown service routine. While powering down, however, an uncontrollable MPU could set the RAME bit and subsequently alter the RAM. The probability of this occurrence, however, is usually within acceptable limits.

It should be noted that the RAME bit is powered by the standby power source. It will not function correctly if \( V_{CC} \) Standby has not reached 4.75 volts before the positive edge of \( \text{RESET} \) reaches 4.0 volts.

3.1.5.1.2 Standby Power (STBY PWR) Bit. The STBY PWR bit can be used to monitor \( V_{CC} \) Standby during powerdown operation and determine, by inference, the validity of data in the standby RAM. The STBY PWR bit is readable and writeable and is not affected by Reset. The bit is cleared by an MPU write or whenever \( V_{CC} \) Standby decreases to some value which is less than \( V_{SBB} \) minimum. However, it can be set only by an MPU write. A typical sequence in utilizing the STBY PWR bit is as follows:

1. Software should set the STBY PWR bit during a powerdown sequence.
2. If the STBY PWR bit is found to be set during a subsequent power-up recovery procedure, then it can be assumed that the standby power source has remained above \( V_{SBB} \) minimum during standby operation and, by inference, data in the standby portion of the RAM is valid. However, if the STBY PWR bit is clear, the contents of the RAM must be suspect. This condition indicates that \( V_{CC} \) Standby has fallen below the standby level threshold at some time during standby operation.

The standby power sense circuit will not clear the STBY PWR bit if \( V_{CC} \) Standby remains above \( V_{SBB} \) (minimum). The threshold voltage for clearing the STBY PWR bit is process related but will always exceed the level at which the RAM fails to retain data.

A cleared STBY PWR bit does not necessarily mean that data in the RAM has been altered. The RAM will retain data even if the \( V_{CC} \) Standby is decreased to a level somewhat below the standby sense threshold. This feature provides an engineering margin of reliability to ensure that the STBY PWR bit will clear before any RAM cell fails to retain data. For this reason, the only valid interpretation of the STBY PWR bit is: if the bit remains set, standby RAM data can be assumed to be valid. If the bit is clear, one cannot be certain of the data validity. In the absence of any other information, data would normally be assumed to be invalid.

The STBY PWR bit can also be used to determine if \( V_{CC} \) Standby is below \( V_{SBB} \) minimum at any particular moment. If the STBY PWR bit is set and then read, a "0" for the result indicates that the voltage is below \( V_{SBB} \) minimum.
3.1.5.1.3 RAM Control Register. The RAM Control Register ($14) contains two bits: STBY PWR and RAME. These bits can be used to determine the status and control accesses to the internal RAM. The register is defined as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>0-5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>b7</td>
<td>x</td>
<td>x</td>
<td>X</td>
</tr>
<tr>
<td>b6</td>
<td>x</td>
<td>x</td>
<td>X</td>
</tr>
<tr>
<td>b5</td>
<td>x</td>
<td>x</td>
<td>X</td>
</tr>
<tr>
<td>b4</td>
<td>x</td>
<td>x</td>
<td>X</td>
</tr>
<tr>
<td>b3</td>
<td>x</td>
<td>x</td>
<td>X</td>
</tr>
<tr>
<td>b2</td>
<td>x</td>
<td>x</td>
<td>X</td>
</tr>
<tr>
<td>b1</td>
<td>x</td>
<td>x</td>
<td>X</td>
</tr>
<tr>
<td>b0</td>
<td>x</td>
<td>x</td>
<td>X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$14</th>
<th>STBY PWR</th>
<th>RAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>b7</td>
<td>STBY</td>
<td>X</td>
</tr>
<tr>
<td>b6</td>
<td>PWR</td>
<td>X</td>
</tr>
<tr>
<td>b5</td>
<td>RAME</td>
<td>X</td>
</tr>
<tr>
<td>b4</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>b3</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Bit 0-5

Unused

Bit 6

RAME. The RAM Enable bit allows removal of the RAM from the MCU internal memory space. This read/write bit is set (RAM enabled) during Reset, provided that standby power is available on the rising edge of RESET. The RAME bit is defined as follows:

- RAME = 0 Internal RAM addresses disabled; data read/written from external memory,
- RAME = 1 Internal RAM addresses enabled.

Bit 7

STBY PWR. The purpose of the Standby Power bit is to indicate whether the standby power supply has adequately preserved the data in the standby RAM after a period of standby power operation has terminated. The Standby Power bit is designed to clear whenever VCC Standby decreases to some value less than VSBB. This voltage is above the minimum required to assure the integrity of data in standby RAM. The STBY PWR bit can be set or cleared by software. Reset does not affect this bit. The STBY PWR bit is defined as follows:

- STBY PWR = 0 The standby power voltage has decreased to a value less than VSBB (minimum)
- STBY PWR = 1 The standby power voltage has remained above the VSBB minimum value

3.1.6 NMI: Non-Maskable Interrupt Request

A negative edge on the NMI pin signifies an interrupt request but the MCU will complete the current instruction before responding. After completing the instruction, the MCU will begin an NMI interrupt sequence regardless of whether the I-bit in the Condition Code Register is set or clear.

The NMI interrupt sequence pushes the Program Counter, Index Register, Accumulator A, Accumulator B, and Condition Code Register on the stack. These registers constitute the MPU "machine state" and stacking them allows restoration of the former state after the interrupt service routine is completed. After the registers have been stacked, the I-bit in the Condition Code Register is set which inhibits any additional interrupts except another NMI. Finally, a vector is fetched from $FFFC and $FFFF, transferred to the Program Counter, and instruction execution is resumed at this location.
An external pull-up resistor to \( V_{CC} \) must typically be used on the \( NMI \) line because the MCU has no internal \( NMI \) pull-up resistor. The value of this resistor is limited by the current capability of the device causing the interrupt and values between 3\( \mathrm{k}\Omega \) and 10\( \mathrm{k}\Omega \) are typical. \( NMI \) must be held low for at least one E-cycle to be recognized under all conditions. Because it is edge-triggered, \( NMI \) is very intolerant of electrical noise.

**NOTE**

\( NMI \) must not remain unconnected in any MC6801 system. The MCU has no internal \( NMI \) pull-up resistor.

Further information regarding the Non-Maskable interrupt is provided in Chapter 5.

### 3.1.7 IRQ1: Maskable Interrupt Request

\( \text{IRQ1} \) is an active-low level-sensitive input which is used to initiate a request for an interrupt sequence. The MPU will complete execution of the current instruction before responding to the request. If the interrupt mask bit (I-bit) in the Condition Code Register is clear, the MPU will begin an interrupt sequence after completing the current instruction. The Program Counter, Index Register, Accumulator A, Accumulator B, and Condition Code Register are pushed on the stack. The top seven locations of the stack contain the MPU "machine state" which can be restored later to resume execution of the interrupted routine. The I-bit in the Condition Code Register is then set, preventing additional responses to the same or other maskable interrupts. Finally, a vector is fetched from \$FFFF8 and \$FFFF9, transferred to the Program Counter, and instruction execution is resumed at this location.

There is no internal \( \text{IRQ1} \) pull-up resistor and an external one to \( V_{CC} \) must be provided if there is any possibility of the I-bit being cleared. The value of this resistor is limited by the current capability of the device causing the interrupt; values between 3\( \mathrm{k}\Omega \) and 10\( \mathrm{k}\Omega \) are typical. A more detailed discussion of the maskable interrupt sequence is contained in Chapter 5.

### 3.1.8 P10-P17: Port 1

Port 1 provides a versatile 8-bit parallel I/O port with Schottky TTL drive capability. Internal current limiting resistors for each output driver enable the port to also interface with Darlington transistors. The current limiting resistors are unique to Port 1; none of the other three ports have Darlington drive capability. Port 1 can be interfaced with CMOS devices if external pull-up resistors are provided.

Each bit of the port can be individually configured as an input or output as defined by Port 1 Data Direction Register. During Reset, the Port 1 Data Direction Register is cleared, configuring the eight lines as inputs. Software can then be used to change any or all of the eight lines to outputs by setting the corresponding bits in the Port 1 Data Direction Register. Port 1 can drive one Schottky TTL load and 30\( \mathrm{p}\)F or it can source one milliampere (minimum) of current for Darlington drive at 1.5 volts.
Timing diagrams for MPU reads and writes to Port 1 are shown in Figures 3-8 and 3-9, respectively. During an MPU read of Port 1, valid data must be presented by setup time, $t_{PDSU}$ before the positive edge of $E$ of the MPU read cycle. Data is latched into Port 1 on the rising edge of $E$ of the read cycle and data is required to remain valid for at least hold time, $t_{PDH}$, after the same positive edge. Data is latched to prevent it from changing while being read and is latched only during the MPU read.

**Figure 3-8. Data Port Timing for MPU Read**

During an MPU write cycle to Port 1, data becomes valid at the output pin by $t_{PWD}$, after the negative edge of $E$ of the write cycle. However, when interfacing with CMOS devices, data is valid by $t_{CMOS}$ which allows levels to reach 0.7 VCC.

**Figure 3-9. Data Port Timing for MPU Write**

All MCU input and output Data Registers are accessed using only a single address ($02$ for Port 1). The particular Data Register (input or output) is implied by the nature of the access: a “read” addresses the input Data Register whereas a “write” will reference the output Data Register. Typically, programmers can consider them both a single “Data Register” without incurring any problems.
All MCU Data Direction Registers are write-only registers. If an attempt is made to read them, the value, $FF$, will always be obtained for all except Port 3. If the Port 3 Data Direction Register is read (modes 4 and 7), the Port 3 Data Register will be read instead.

A logic diagram for Port 1 is shown in Figure 3-10. The PDB1 signal refers to the internal Peripheral Data Bus which is shown in Figure 3-2. The PDB interfaces with the MPU data bus through a bus multiplexer in the MPU. Signals in the logic diagrams are defined as follows:

- POR is an active high internal reset signal which is synchronized with E;
- WP1 goes high during E of an MPU write to either the Port 1 Data or Data Direction Register;
- WIO1 goes high during E after WP1 for an MPU write to the Port 1 Data Register;
- DDR1 goes high during E after WP1 for an MPU write to the Port 1 Data Direction Register;
- PDBi is the ith line of the internal Peripheral Data Bus (PDB);
- RIO1 enables a read of Port 1 input Data Register.

The logic diagram utilizes symbols representing discrete components and logic gates. Logic gates are implemented with depletion-load static-logic circuits in standard n-channel silicon-gate technology. Discrete MOS transistor symbols are used to indicate three functions: protective devices, push-pull drivers, and couplers.

A protective device is a grounded-gate enhancement-mode transistor connected in series with a resistor between the pin pad and ground. One such device is shown as Q1 in Figure 3-10.

![Figure 3-10. Logic Diagram for Port 1](image-url)
A push-pull driver comprises two enhancement-mode transistors connected in series between VDD and ground. A series current-limiting resistor is sometimes connected to VDD. The driver is depicted as Q2 and Q3 in Figure 3-10 with current-limiting resistor, R.

The MOS coupler is an enhancement-mode transistor used as a switch. It is either "on" — when operating in saturation with its gate voltage high — or "off" — in cutoff with its gate voltage low. Implicit in coupler operation is a parasitic capacitance (Cp in Figure 3-10) which charges to the level of the driving source when the coupler is on and holds the charge when the coupler is off. When a coupler with VDD on its gate is connected between the output and input of a two-stage latch (depicted as Q4 in Figure 3-10), it serves merely to compensate for leakage from the input node and performs no logic function.

During the E time (i.e., the positive half-cycle of E) of a read of the input Data Registers*, RIO1 is driven high while RIO1 goes low. RIO1 prevents data from changing while it is being read by latching it into the parasitic capacitance of the FET. When the RIO1 coupler is "on," data from inverter I2 is transferred to the Peripheral Data Bus. The resistor, R, is unique to Port 1 and functions as a current limiter which enables the port to drive Darlington transistors. It should be noted that if Port 1 is driving Darlington transistors and the output data is a "1" (i.e., acting as a current source), the voltage at the pin could be too low to indicate a "1" if the input Data Register is read by the MPU.

The FET, Q1, acts as a reverse voltage protector and is a common element for all pins of the MCU except the three power pins (VCC, VCC Standby, and VSS). The FET conducts when the reverse (negative) voltage exceeds its threshold and attempts to prevent damage to the part.

During E of a write operation, the WP1 coupler is "on" and data on the Peripheral Data Bus is transferred to the PDB inverter where it is stored on the parasitic capacitance, Cp. On the following half-cycle of E (E), either the WIO1 or DDR1 coupler is "on" depending upon whether the Port 1 output Data Register (DR) or Data Direction Register (DDR) is being addressed.

If WIO1 is high, data is transferred from the Peripheral Data Bus and latched in the output Data Register. The output of this register is connected to the input of the push-pull output driver which is controlled by the Data Direction Register.

If DDR1 is high, data is transferred from the PDB to the Data Direction Register. If the value is a "0" (on the PDB), both output devices (Q2 and Q3) are off, and the output driver assumes the high impedance state. This allows data to be read from the input buffer, I2. However, if the PDB data is a "1", when DDR1 is high, the output driver is enabled and data read from the input buffer reflects the level at the pin.

Data from the PDB is delayed from reaching its assigned register by one-half E-cycle. The rationale for this delay can be explained by considering the following. Assume that a Data Register bit contains a "0" and a "0" is being rewritten to it from the PDB. The Peripheral Data Bus (PDB) is a dynamic bus for which its parasitic capacitance is precharged high when E is low. Thus at the beginning of a transfer, the bus is always high until data becomes valid. The level will then either remain high or be reduced to zero by discharging the capacitance. If the bus were directly coupled to the output Data Register or Data Direction Register, the output of the driver would go high and then low which would appear at the output as a transient "spike."

*Technically, the actual input Data Register is implemented as a buffer using the parasitic capacitance of an inverter (I2) as opposed to an MOS latch. Nevertheless, it is referred to as a register.
Note that the port Data Register has a single address (see Figure 2-17) but consists of separate input and output Data Registers. When the Data Register is accessed, the Read/Write line is used to select the appropriate register. When a port line is configured as an output, a read of its input Data Register will reflect the voltage level at the output pin. Alternatively, when a port line is configured as an input, writing to its output Data Register has no effect on the port operation because the output driver is held in the high impedance (off) state. If the Data Direction Register is read, the Peripheral Data Bus precharge will always provide $FF as the result.

3.1.9 P20-P24: Port 2

Port 2 provides five multifunctional lines where each can be dedicated to a different function. When configured as an output, each line is capable of driving one Schottky TTL load and 30 pF. Port 2 can also drive CMOS devices if external pull-up resistors are provided.

Timing diagrams for MPU reads and writes to Port 2 are shown in Figures 3-8 and 3-9, respectively. During an MPU read, data must be valid by setup time, \( t_{PD_{SU}} \), before the positive edge of E of the MPU read cycle. Data is latched into Port 2 and must remain valid for at least hold time, \( t_{PD_{H}} \), after the same positive edge. Data remains latched only during the read cycle.

During an MPU write to Port 2, data becomes valid at the output pin no later than \( t_{PW_{D}} \) after the negative edge of E of the write cycle. If interfacing to CMOS devices, however, data is not valid until a somewhat longer period shown as \( t_{CMOS} \). This is necessary to allow the output to reach 0.7 of \( V_{DD} \) (CMOS supply voltage) with the assistance of external pull-up resistors.

The Logic diagrams for Port 2 are shown in Figures 3-12 through 3-15. Many of the signals shown are common to all five bits while others are peculiar to a particular bit. Signals of the latter type are explained within the discussion for each bit whereas the common signals are defined as follows:

- \( POR \) is an active high internal reset signal which is synchronized with E;
- \( WP2 \) goes high during E of an MPU write to either the Port 2 Data or Data Direction Register;
- \( WIO2 \) goes high during \( \bar{E} \) after \( WP2 \) for an MPU write to the Port 2 Data Register;
- \( DDR2 \) goes high during \( \bar{E} \) after \( WP2 \) for an MPU write to the Port 2 Data Direction Register;
- \( PDB_{i} \) is the ith line of the internal Peripheral Data Bus (PDB);
- \( RIO2 \) enables a read of the Port 2 input Data Register.

3.1.9.1 PORT 2 DATA REGISTER INPUT/OUTPUT. During Reset, all five bits of the Port 2 Data Register are configured as inputs and all bits, except bit 1, can be re-configured as Data Register outputs. Bit 1 cannot be used as an output from the Port 2 Data Register and, if configured as an output, is dedicated to the Timer Output Level Register. This is the reason for excepting P21 in the MCU write timing diagram for Port 2 shown in Figure 3-9. The bit can be used, however, for an input to the Data Register.

During E of an MPU read of the Data Register, \( RIO2 \) is driven high while \( \bar{RIO2} \) goes low. The \( RIO2 \) coupler latches the data while being read. When the \( RIO2 \) coupler is "on," data from inverter I2 is transferred to the Peripheral Data Bus.
During E of a write operation, the WP2 coupler is “on” and data on the PDB is transferred to an inverter where it is dynamically stored. On the following half-cycle of E (E) either the WIO2 or the DDR2 coupler is “on” depending on whether the output Data Register or Data Direction Register is being addressed. Data is then transferred from the inverter to the selected register.

Data is delayed from reaching its final destination by one half-cycle. The explanation for this delay is similar to that described for Port 1: to prevent transient “spikes” in the output of the Data or Data Direction register when rewriting a “0” to either of them.

3.1.9.2 MODE SELECT PINS: P20, P21, P22 On the positive edge of \textsc{reset}, the levels on pins 8, 9, and 10 are latched in a 3-bit Program Control Register and used to define the MCU operating mode. The Program Control Register can be read by accessing the Port 2 Data Register. The mode appears as the three most significant data bits as shown in Figure 3-11 where PC2 is the level that was latched from pin 10. Details concerning mode programming were presented in Section 3.1.3.

![Figure 3-11. Port 2 Data Register](image)

3.1.9.3 TIMER INTERFACE: P20, P21. Bits 0 and 1 of Port 2 also provide an interface for two Programmable Timer functions. Bit 0 interfaces with the input capture edge detector and bit 1 can be driven by the Output Level Register. To enable typical use of these functions, the Port 2 Data Direction Register should be configured such that bit 0 is a “0” (input) and bit 1 is a “1” (output).

The input capture edge detector is always sensing the P20 line regardless of its configuration. Note that an input capture will also be generated by configuring bit 0 as an output and toggling P20 by writing to bit 0 of the Port 2 Data Register.

A logic diagram for bit 0 is shown in Figure 3-12. TIC provides a buffered input to the Programmable Timer input capture edge detector. It should be noted that this input is always provided regardless of bit configuration.

A logic diagram for bit 1 is shown in Figure 3-13. Note that the output Data Register is not connected to the Peripheral Data Bus. When P21 is configured as an output, any MPU writes to this bit have no effect on MCU operation. The TOL (Timer Output Level) signal is driven from a 1-bit Output Level Register which originates from the OLVL bit in the Timer Control and Status Register and is clocked by successful output compare.
Figure 3-12. Logic Diagram for Port 2 Bit 0

Figure 3-13. Logic Diagram for Port 2 Bit 1
3.1.9.4 SERIAL COMMUNICATIONS INTERFACE: P22, P23, P24. Bits 2, 3 and 4 of Port 2 provide an interface for the SCI bit rate clock, receiver input, and transmitter output, respectively. The SCI has exclusive use of bits 2, 3 and 4 of Port 2 if certain SCI functions are enabled.

If the Transmitter Enable (TE) bit is set, bit 4 will be forced to an output and P24 will provide serial output data. If the Receiver Enable (RE) bit is set, bit 3 will be forced to an input and P23 will provide serial input data to the SCI receiver.

The SCI clocking options are controlled by bits CC0 and CC1 in the Rate and Mode Control Register. These bits are defined in Chapter 6 and control P22 if certain clocking options are enabled. If CC1 is set, P22 is dedicated to the SCI and implements either the external-clock-in or internal-clock-out option as defined by CC0.

When the internal-clock-out option is selected, it is provided regardless of whether the SCI transmitter and/or receiver have been enabled. If CC1 is clear, the configuration of P22 is controlled by the Port 2 Data Direction Register.

A logic diagram for bit 2 is shown in Figure 3-14 where P22 can interface with:
- an input or output from the Data Register,
- an external bit rate (X8) clock input, or
- an internal bit rate clock output.

![Logic Diagram for Port 2 Bit 2](image)

Figure 3-14. Logic Diagram for Port 2 Bit 2
CC1 and CC0 are static control signals which originate from corresponding bits in the Rate and Mode Control Register. Dynamic signals include DDR2, WP2, WIO2, and RIO2 which control accesses to the input and output Data and Data Direction Registers.

TEX is derived from the SCI bit rate clock and is used for the internal-clock-out option. TIN8 provides an eight times bit rate clock to drive the SCI for the external-clock-in option.

If CC1 is "0", CC0 and TEX are de-coupled from the circuit and P22 functions as a data port controlled by DDR2, WP2, WIO2, and RIO2. In this configuration, P22 provides input or output from its Data Direction Register as defined by bit 2 of its Data Direction Register.

If CC1 is a "1", the CP1 and CP2 couplers are disabled and isolate the Data and Data Direction Registers from the Peripheral Data Bus. In addition, coupler CP3 is enabled and CC0 is connected to the Data Direction Register to appropriately configure the bit. If CC0 is a "0", coupler CP4 is enabled and TEX is connected to the output Data Register and provides the SCI output bit rate clock. If CC0 is a "1", the output driver is forced to the high impedance state and an external eight times bit rate clock (TIN8) input is coupled to the SCI.

A logic diagram for bit 3 is shown in Figure 3-15 where RE is the corresponding bit in the Transmit/Receive Control and Status Register. If RE is not set, P23 functions as a data port bit and is controlled by DDR2, WP2, WIO2 and RIO2. If RE is set, P23 is forced to an input and provides serial data, RX, to the SCI receiver. While RE is set, the DDR bit for P23 is not affected by any MPU writes. It should be noted that RX can be read from the input Data Register and will reflect the value of the serial data line at the time of the read.

![Figure 3-15. Logic Diagram for Port 2 Bit 3](image-url)
A logic diagram for bit 4 is shown in Figure 3-16 where TE is the corresponding bit in the TRCS Register. When TE is not set, the bit functions as a data port bit controlled by DDR2, WP2, WI02 and RIO2. If TE is set, however, Data Direction Register bit 4 is set which configures P24 as an output. While TE is set, the DDR bit for P24 is not affected by any MPU writes. Coupler CP1 is turned “off” which isolates the Data Register bit from the Peripheral Data Bus (PDB) and makes it inaccessible to MPU writes. Coupler CP2 is turned on which connects the output of the serial shift register, TDST, to the output Data Register and driver.

Figure 3-16. Logic Diagram for Port 2 Bit 4

3.2 SINGLE CHIP MODE PIN DESCRIPTION

In Single Chip Mode, the MCU functions as a monolithic microcomputer without external address or data buses. As shown in Figure 1-3, all four ports are configured as parallel data ports. The four MCU data ports provide a maximum of 29 input/output lines and SC1 and SC2 are configured as control lines which can be used with Port 3. The operational characteristics of Port 3 are controlled by its Control and Status Register which is located at $0F$ (see Figure 2-17).

3.2.1 SC1: Input Strobe 3 (IS3)

SC1 is configured as an input only in Single Chip Mode and is configured as an output in all other modes. In Single Chip Mode, SC1 interfaces a negative edge detector with an active low asynchronous input signal called IS3. IS3 is pulled to VCC by an internal active device and can be used as an input data strobe or a data acknowledgement from another device depending upon the configuration of Port 3.
An $\overline{IS3}$ negative edge will set the IS3 FLAG in the Port 3 Control and Status Register. This register also controls two options which are associated with IS3. First, an IRQ1 interrupt can be enabled whenever IS3 FLAG is set. Note that this is the only MCU internally generated interrupt which results in an IRQ1 interrupt and is effectively "wire-ORed" with the external $\overline{IRQ1}$ line. Software can be used to distinguish between internal and external IRQ1 interrupts by inspection of IS3 FLAG in the Port 3 Control and Status Register. Note that this interrupt can occur only in Single Chip Mode.

Finally, an $\overline{IS3}$ negative edge can be used to latch data into Port 3 by setting the LATCH ENABLE bit in the Port 3 Control and Status Register. Latch setup and hold times with respect to IS3 are shown in Figure 3-17 where IS3 is an asynchronous input. External data must be valid at Port 3 by the data setup time ($t_{IS}$) prior to the negative edge of IS3 and remain valid for at least hold time, $t_{IH}$. The width of the active low strobe cannot be less than $t_{PWIS}$.

![Figure 3-17. Port 3 Latch Setup and Hold Times](image)

Port 3 can also be used in a configuration which utilizes some bits as outputs and the remainder as latched inputs. There is no conflict in this configuration. An MPU write to the Port 3 Data Register will not "re-open" the latch while an MPU read will obtain the latched input levels and the current output levels.

### 3.2.2 SC2: Output Strobe 3 ($\overline{OS3}$)

SC2 is configured as an output signal in all modes and, in Single Chip Mode, it provides an active low output strobe called $\overline{OS3}$. This signal can be used to strobe outputs to an external device or provide a data acknowledgement depending on the configuration of Port 3. The active low strobe (pulse) is approximately one E-cycle wide and cannot be varied.

$\overline{OS3}$ can be generated by either an MPU read or write to the Port 3 Data Register and is controlled by the Output Strobe Select (OSS) bit in the Port 3 Control and Status Register. If the bit is clear, a pulse is generated by an MPU read of the Port 3 Data Register; if set, it is generated by an MPU write.

A timing diagram for $\overline{OS3}$ is shown in Figure 3-18. Data is valid at the port not more than $t_{PWD}$ after the negative edge of E of the write cycle as shown in Figure 3-9. The active low output strobe occurs $t_{OSD1}$ after the next positive edge of E. It should be noted that if Port 3 is being used as an output port, data can be considered valid on either edge of the $\overline{OS3}$ pulse. $\overline{OS3}$ is capable of driving one Schottky TTL load and 90 pF.
3.2.3 P30-P37: Port 3

Depending on the mode, Port 3 can be configured as (1) a parallel I/O port with two handshake control lines, (2) a bidirectional data bus, or (3) as a multiplexed address and data bus. In Single Chip Mode, it is configured as an I/O port controlled by its Data Direction and Control and Status Register. The output drivers of Port 3 are designed for one Schottky TTL load and 90 pF.

Timing diagrams for an MPU read and write to Port 3 are shown in Figures 3-8 and 3-9, respectively, where the read shown is for unlatched (LATCH ENABLE = 0) operation. Timing for MPU reads using the input latch is shown in Figure 3-17. During an unlatched MPU read, valid data must be presented by setup time, \( t_{PD_SU} \), before the negative edge of E and must remain valid for at least hold time, \( t_{PH} \), after the negative edge. It should be noted that the read timing for Ports 1, 2 and 4 is with respect to the positive edge of E whereas the timing for an unlatched read of Port 3 is with respect to the negative edge.

During an MPU write cycle, data becomes valid at the output of the port not more than \( t_{PWD} \) after the negative edge of E of the write cycle. If interfacing to a CMOS device, however, data is not valid until a somewhat longer period, \( t_{PMOS} \), which allows the output to reach 0.7 V\(_{CC}\) using external pull-up resistors.

Port 3 consists of two major functional elements: an Input/Output buffer and associated control logic. The buffer can be further divided into a three-state output driver, input and output Data Registers, a Data Direction Register, and a bus arbitrator for the MCU data bus. The Data and Data Direction Registers are directly accessible by MPU instructions from the Peripheral Data Bus only in the Single Chip Modes.

A logic diagram of the Port 3 Input/Output buffer is shown in Figure 3-19. Control signals which are active in Single Chip Mode are defined as follows:

- M47 is high only in Modes 4 and 7 (Single Chip Modes);
- POR is an active high internal reset signal which is synchronized with E;
- WP3 goes high during E of an MPU write to either the Port 3 Data or Data Direction Register;
- WI03 goes high during E after WP3 for an MPU write to the Port 3 output Data Register;
- DDR3 goes high during E after WP3 for an MPU write to the Port 3 Data Direction Register;
- DIBP couples the internal Peripheral Data Bus to the MPU Data Bus during an internal MPU read of an internal address excluding the Port 3 Data and Data Direction Registers;
Figure 3-19. Logic Diagram for Port 3

- DIB3 couples data from the Port 3 input Data Register to the MPU internal Data Bus during an MPU read whenever DIBP is not active;
- P3R enables the input latch and is controlled by the Latch Enable bit of the Port 3 Control and Status Register. If this bit is set, P3R will go low on a negative transition of IS3 and latch the input data. It will remain low until the Port 3 Register is read by the MPU. If the LATCH ENABLE bit is clear, P3R will remain high;
- DBRW3, DWR and LADD are not active in the Single Chip Mode and are held low.

During an MPU write operation, WP3 goes high on the rising edge of E during an access of either the Port 3 Data or Data Direction Register. One-half E-cycle later (E), either WIO3 or DDR3 will go high depending on whether the output Data or Data Direction Register is being accessed. The half-cycle delay in the data path ensures that no transient "spikes" will appear in the output when a "0" is written to either the output Data or Data Direction Register.

During an MPU read, DIB3 is high and the input Data Register is coupled to the MPU Data Bus.* If the LATCH ENABLE bit of the Port 3 Control and Status Register is set, an IS3 negative edge will cause P3R to go low and latch data in the input Data Register. Following an MPU read of the Port 3 Data Register, P3R will return high on the rising edge of E and make the latch transparent. P3R is not affected by an MPU write to the Port 3 Data Register.

*Note that an MPU read of the Port 3 Data Direction Register will result in reading the Port 3 Data Register instead. However, this operation does not affect any Port 3 control functions such as IS3, IS3 Flag, GS3 or latch operation.
The input Data Register in Port 3 (Figure 3-19) is similar to the Port 1 input Data Register shown in Figure 3-10. Note that the P3R coupler in the Port 3 diagram appears where the R101 coupler is located in the Port 1 diagram. In an unlatched MPU read of the Port 3 Data Register, however, P3R remains high which buffers unlatched data between Port 3 and the internal data bus. This accounts for the difference in read timing between Port 3 and the other three ports as shown in Figure 3-8.

3.2.3.1 PORT 3 CONTROL AND STATUS REGISTER. The Port 3 Control and Status Register ($0F) is accessible only in Single Chip Mode and consists of four bits which control Port 3 operating characteristics. Three of the four bits are both readable and writeable while IS3 FLAG bit is a read-only status bit. The bits in the register are defined as follows:

<table>
<thead>
<tr>
<th>Bit 0</th>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Bit 3</th>
<th>Bit 4</th>
<th>Bit 5</th>
<th>Bit 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS3 FLAG</td>
<td>IS3 IRQ1 ENABLE</td>
<td>X</td>
<td>OSS</td>
<td>LATCH ENABLE</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

- **Bit 0**: Not used.
- **Bit 1**: Not used.
- **Bit 2**: Not used.
- **Bit 3**: LATCH ENABLE. This bit controls the input latch for Port 3. If set, input data to Port 3 is latched on the negative edge of Input Strobe (IS3). The latch is transparent on the rising edge of \( \overline{E} \) following an MPU read of the Port 3 Data Register (a read of the Port 3 Control and Status Register is not required to make the latch transparent). If clear, the latch is transparent. LATCH ENABLE is cleared by Reset.
- **Bit 4**: OSS (Output Strobe Select). This bit controls whether \( \overline{OSS} \) will be generated by an MPU read or write to Port 3. When clear, the strobe is generated by a read; when set, the strobe is generated by a write. OSS is cleared by Reset.
- **Bit 5**: Not used.
- **Bit 6**: IS3 IRQ1 ENABLE. This bit can be used to enable an IRQ1 interrupt in response to a negative edge of Input Strobe 3 (IS3). When clear, IS3 FLAG will be set but an interrupt will not be enabled. If IS3 IRQ1 ENABLE is set, an IRQ1 interrupt will be enabled whenever IS3 FLAG is set. IS3 IRQ1 ENABLE is cleared by Reset.
- **Bit 7**: IS3 FLAG. This bit is a read-only status bit which is set by the MCU in response to a negative edge of Input Strobe 3 (IS3). It is cleared by a read of the Port 3 Control and Status Register (with IS3 FLAG set) followed by either a read or write of the Port 3 Data Register. This bit is cleared by Reset.

3.2.4 P40-P47: Port 4

Depending on the mode, Port 4 can be configured as (1) a parallel data port, (2) an address/input port, or (3) as a dedicated address output port. In Single Chip Mode, Port 4 is an 8-bit data port where each bit can be configured as an input or output. During Reset, the Port 4 Data Direction Register is cleared which configures all of the lines as inputs. Lines can be changed to outputs by setting the desired bits in the Data Direction Register.
When configured as outputs, Port 4 can drive one Schottky TTL load and 90 pF. The Port 4 output driver also includes a depletion load transistor for each line. This can be considered equivalent to a pull-up resistor to $V_{CC}$ and allows it to directly interface with CMOS devices at $V_{CC}$ levels. Port 4 should not be pulled higher than $V_{CC}$, however, due to the depletion load transistor which is coupled directly to $V_{CC}$.

Timing diagrams for an MPU read and write to Port 4 are shown in Figures 3-8 and 3-9, respectively. During an MPU read, valid data must be presented for at least the setup time ($t_{PDUS}$) before the positive edge of $E$ corresponding to the MPU read cycle. Data is latched into Port 4 on the rising edge of $E$ of the read cycle, and must remain valid for at least the hold time ($t_{PDH}$). Data is latched only during the MPU read cycle.

During an MPU write, data becomes valid at the output of the port by $t_{PWD}$ after the negative edge of $E$ of the write cycle. When interfacing to CMOS (at $V_{CC}$ levels), however, it is valid by $t_{CMOS}$ after the negative edge in order to allow levels to reach 0.7 $V_{CC}$.

Port 4 consists of two major functional elements: an Input/Output buffer and associated control logic. The Port 4 control logic generates a read signal, $RI04$, and three write signals, $WP4$, $DDR4$ and $WIO4$. These signals control accesses to the Port 4 input and output Data Registers and Data Direction Register. A logic diagram of the Input/Output buffer is shown in Figure 3-20 and consists of an input and output Data Register, Data Direction Register, and an output driver. The output Data Register is accessible from the internal Peripheral Data Bus only in Single Chip Mode and the input Data and Data Direction Registers are accessible only in Modes 4 to 7. In other modes, half of the address bus (either A0-A7 or A8-A15) is connected directly to the output driver. Definitions of signals in the logic diagram include:

- $POR$ is an active high internal reset signal which is synchronized with $E$;
- $WP4$ goes high during $E$ of an MPU write to either the Port 4 Data or Data Direction Register;
- $WIO4$ goes high during $\bar{E}$ after $WP4$ for an MPU write to the Port 4 Data Register;
- $DDR4$ goes high during $\bar{E}$ after $WP4$ for an MPU write to the Port 4 Data Direction Register;
- $PDBi$ is the $i$th line of the internal Peripheral Data Bus (PDB)
- $RI04$ enables a read of the Port 4 input Data Register;
- $PC2$ is the most significant bit of the operating mode (set in Modes 4 to 7);
- $MOD5$ and $MXM$ are inactive in Single Chip Mode and are held low.

In Modes 4 through 7, $PC2$ is held high which allows the port to be configured by its Data Direction Register. During an MPU read of the Port 4 input Data Register, $RI04$ will go high during $E$ while $RIO4$ is driven low to prevent the data from changing while it is being read. When the $RI04$ coupler is "on," data from the inverter is transferred to the Peripheral Data Bus.

During an MPU write to either the output Data or Data Direction Registers, $WP4$ will go high during $E$. One-half $E$-cycle later ($\bar{E}$), either the $WIO4$ or the $DDR4$ coupler will turn on depending on whether the Data or Data Direction Register is being accessed. This will couple data (one half-cycle delayed) to the selected register. The data path is delayed one-half $E$-cycle to prevent transient pulses at the output Data or Data Direction Register when writing a "0" to either register. The pulse is caused by precharging the Peripheral Data Bus as explained in the description for Port 1.
3.3 EXPANDED NON-MULTIPLEXED MODE PIN DESCRIPTION

In the expanded non-multiplexed mode, the MCU can directly address an external memory space of 256 bytes located from $100$ through $1FF$. SC1 is configured as an output and provides an external address select signal called Input/Output Select (IOS). SC2 is always configured as an output and provides a Read/Write data bus control signal. Port 3 is configured as an 8-bit bidirectional Data Bus while Port 4 is configured from Reset as an 8-bit input data port. Using software, however, any combination of the Port 4 eight least significant address lines can be provided as outputs by setting the corresponding bits in its Data Direction Register. Stated alternatively, any of the Port 4 unrequired address lines can be used as additional data input lines. Internal pull-up resistors are provided to pull the Port 4 lines high until software configures the Data Direction Register.

3.3.1 SC1: Input/Output Select (IOS)

In the Expanded Non-Multiplexed Mode, SC1 provides an active low output called IOS (Input/Output Select) which indicates when a certain range of addresses in the external memory space is sensed on the internal address bus. The signal is the output of an internal address decoder which uses lines A8 through A15 as inputs and is active low whenever an address from $0100$ through $01FF$ appears on the internal address bus. In other words, IOS is active low when $01$ is detected on lines A8 to A15, of the internal address bus. IOS is capable of driving one Schottky TTL load and 90 pF and can be used similar to an address pin. IOS timing is shown in Figure 3-23.
Because $\text{IOS}$ must be used to determine when an external address is being referenced, software must address external read/write memory only from $\$100$ to $\$1FF$. While the Port 3 register addresses are not internal in the Expanded Non-Multiplexed Mode, $\text{IOS}$ is not active if they are referenced because their addresses are not in the range from $\$100$ to $\$1FF$.

### 3.3.2 SC2: R/W (Read/Write)

The R/W (Read/Write) output signal is used to control the direction of transfers on the external Data Bus. A low level (Write) on the Read/Write line enables the Port 3 output drivers to the external Data Bus. A high level (Read) on the Read/Write line forces the drivers to a high impedance state and enables data to be read from an external device. The R/W output is capable of driving one Schottky TTL load and 90 pF. Timing is shown in Figure 3-23.

### 3.3.3 P30-P37: Port 3 Data Bus (D0-D7)

Depending on the mode, Port 3 can be configured as (1) a parallel I/O port with two handshake control lines, (2) a bidirectional data bus, or (3) as a multiplexed address and data bus. In the Expanded Non-Multiplexed Mode, Port 3 functions as a bidirectional data bus. The output driver is a three-state device which remains in the high impedance (off) state except when the MCU performs a write operation. It is activated when R/W is low and E is high. Each output is designed to drive one Schottky TTL load and 90 pF. Data bus timing is shown in Figure 3-23.

Three registers associated with Port 3 are decoded as external memory in the Expanded Non-Multiplexed Mode. However, these three locations cannot be used for external read/write memory because $\text{IOS}$ is not active when the addresses appear on the bus. Furthermore, even if $\text{IOS}$ were active for these addresses, it would not be possible to distinguish between an access of $\$06$ (Port 3 Data Register) and $\$106$ (external memory reference).

The Port 3 data bus drivers are enabled only during a write to either an internal or external address. Because Port 3 will remain in a high impedance state, it is not possible to monitor the internal data bus during an MPU read of an internal location.

**NOTE**

The internal Data Bus cannot be monitored during an MPU read of an internal location except in Mode 0.

The reader should also note that whenever the MPU detects a read from an address not in its internal address map, it reads from the Port 3 external Data Bus. This characteristic makes it possible to indirectly address an additional 256 external read-only locations in the expanded non-multiplexed mode as discussed in Chapter 8 (Section 8.4).

Port 3 consists of two major functional elements: an Input/Output buffer and associated control logic. The buffer can be further divided into a three-state output driver, input and output Data Registers, a Data Direction Register, and a bus arbitrator for the Peripheral Data Bus. The Data and Data Direction Registers are not accessible from the internal Peripheral Data Bus in the expanded...
non-multiplexed mode and Port 3 is configured as a dedicated 8-bit bidirectional data bus. A logic diagram for Port 3 is provided in Figure 3-21. The active signals for the expanded non-multiplexed mode are as follows:

- POR is an active high internal reset signal which is synchronized with E;
- DWR couples the internal Peripheral Data Bus (PDB) to the Port 3 output driver and is active only during E;
- DIBP couples the internal Peripheral Data Bus to the MPU Data Bus during an internal MPU read;
- DIB3 couples data from the Port 3 input Data Register to the MPU Data Bus during an MPU read, whenever DIBP is not active;
- DBRW3 controls the state of the Port 3 output driver;
- P3R is inactive in the expanded non-multiplexed mode and is pulled high;
- M47, WP3, WI03, DDR3 and LADD are not active in the expanded non-multiplexed modes and are held low.

In expanded non-multiplexed mode, M47 is low which isolates the output Data and Data Direction Registers from the output drivers. DWR is active during E while LADD is inactive and pulled low. These couplers connect the Peripheral Data Bus to the output drivers which, in turn, are controlled by DBRW3. DBRW3 is a function of Read/Write and E: when E is high DBRW3 follows Read/Write and is high at all other times.

![Logic Diagram for Port 3](image-url)
During an MPU read, DBRW3 keeps the output driver in a high impedance state. The P3R coupler is held high ("on") and the bus arbitrator enables either the DIBP or DIB3 couplers depending on the address associated with the read. If the address is internal, DIBP is activated; otherwise, DIB3 is activated which links the external and internal Data Buses.

During an MPU write, the Peripheral Data Bus is connected to the output driver through the DWR coupler. DBRW3 is low during E of a write cycle which enables the output driver.

### 3.3.4 P40-P47: Port 4 Address Bus/Inputs

Depending on the mode, Port 4 can be configured as (1) a parallel Input/Output port, (2) an address/input port, or (3) as dedicated address outputs. In the expanded non-multiplexed mode, Port 4 is configured from Reset as an 8-bit data input port. Using software, however, the port can be re-configured to provide any combination of address lines, A0 to A7, of the address bus. Note that any bit configured as an output is an address line; otherwise, it is a data input line. Each output buffer has an internal pull-up resistor which pulls the output high if no external device is driving the port. Port 4 is capable of driving one Schottky TTL load and 90 pF.

A timing diagram for an MPU read of the Port 4 Data Register is shown in Figure 3-8. During an MPU read of Port 4, valid data must be present for at least setup time, $t_{PDSU}$, before the positive edge of E of the read cycle. Data is latched into Port 4 by this same edge and must remain valid for at least hold time, $t_{PDH}$. Data is latched only during the read cycle.

Port 4 consists of two major functional elements: an Input/Output buffer and associated control logic. The Port 4 control logic generates a read signal (RI04) and two write signals (WP4 and DDR4) which control accesses to the input Data Register and Data Direction Register. A logic diagram of the Input/Output buffer is provided in Figure 3-22. The diagram shows the input and output Data Registers, Data Direction Register, and an output driver. The output Data Register is not accessible from the internal Peripheral Data Bus in Expanded Non-Multiplexed Mode (5). The input Data and Data Direction Registers are accessible, however, in Modes 4 to 7. Definitions in the logic diagram which are active in the Expanded Non-Multiplexed Mode include:

- **POR** is an active high internal reset signal which is synchronized with E;
- **WP4** goes high during E of an MPU write to the Port 4 Data Register;
- **DDR4** goes high during $\overline{E}$ after WP4 for an MPU write to the Port 4 Data Direction Register;
- **PDBi** is the ith line of the internal Peripheral Data Bus (PDB);
- **RI04** causes a read of the Port 4 input Data Register;
- **MOD5** is high only in Mode 5;
- **ALi** is the ith line of the address bus where i varies from 0 to 7 (A0-A7);
- **AHi** is the ith line of the address bus where i varies from 8 to 15 (A8-A15);
- **PC2** is the most significant bit of the operating mode;
- **MXM** and **WI04** are inactive in the expanded non-multiplexed mode and are held low.

In the expanded non-multiplexed mode, the WIO4 coupler isolates the internal Peripheral Data Bus from the port output Data Register. The WP4 and DDR4 couplers are active during MPU writes to the Data Direction Register from the Peripheral Data Bus. The MOD5 coupler is on and connects the eight least significant addresses (ALi) to the output Data Register and driver. If the Data Direction Register bit is clear, however, the output driver will remain in a high impedance (off) state and input data can be read from the PDB. If the DDR bit is set, the ALi address line will appear as an output.
During an MPU read of the Port 4 input Data Register, $RI04$ is driven high while $RI04$ goes low. The $RI04$ coupler prevents data from changing while it is being read by latching it into the FET parasitic capacitance. If configured as an output, the read will reflect the level associated with the address line. An MPU write to the Port 4 Data Register has no effect on MCU operation in the expanded non-multiplexed mode.

**NOTE**

Addresses A0 to A7 are not provided in the Expanded Non-Multiplexed Mode (5) until the Port 4 Data Direction Register has been configured by software.

### 3.3.5 Expanded Non-Multiplexed Bus Timing

The Expanded Non-Multiplexed bus is compatible with the M6800 family of parts. It is a synchronous bus clocked by E (Enable) where every bus cycle is either a read or write cycle. Occasionally, the MPU does not require a memory reference during execution of an instruction and the idle bus cycle appears as an MPU read of an internal location (SFFFF). While reset is held low, IOS, R/W, and Port 4 are pulled high and the Data Bus (Port 3) is held in a high impedance state.

Figure 3-23 illustrates timing for the Expanded Non-Multiplexed Bus. Address, R/W, and IOS are valid by $t_{AD}$ after the negative edge of E. During an MPU write, data on the Data Bus (D0-D7) is valid by $t_{D\overline{W}}$ after the next positive edge of E and must be captured by an external device no later than hold time, $t_{HW}$, after the next negative edge of E. Note that A0-A7, R/W, and IOS remain valid for at least hold time, $t_{AH}$, after the negative edge of E.
During an MPU read cycle, Address, R/W, and IOS are valid by tAD time after the negative edge of E. The external device has a maximum access time of tACCN before valid data must be presented to the data bus. Data must be valid by at least setup time, tDSR, before the next negative edge of E and remain valid by hold time, tHR, after the next negative edge. Address, R/W, and IOS remain valid for at least tAH after the negative edge of E. The access time, tACCN, is a derived quantity and is computed from when the address bus becomes valid. Although provided in the Data Sheet, it can be computed from the following equation:

$$t_{ACCN} = PWEH + t_{AS} - t_{DSR}$$

A typical system implementation is shown in Figure 3-24 and additional examples are included in Chapter 8.

### 3.3.6 Monitoring the Expanded Non-Multiplexed Bus

The MC6801 Data Sheet and Appendix F contains a cycle-by-cycle description of the bus activity for each instruction with respect to the MCU internal buses. This table, however, requires some interpretation to obtain the exact description for either of the two types of MCU buses.

When using this table, the reader must remember that the internal data bus cannot be monitored during an MPU read unless in Mode 0. During these read cycles, Port 3 — which provides the data bus — is held in the high impedance state.

In the expanded non-multiplexed mode, the external data bus is driven by the Port 3 output drivers during an MPU write to any location. The external data bus is not active, however, during MPU reads from internal locations. In a typical application using this mode, the program resides in the internal ROM. A significant portion of the total bus cycles, therefore, are internal MPU reads in this case which tends to limit the usefulness of a logic analyzer. A logic analyzer can be used to monitor all accesses to the external memory space in the range from $100$ to $1FF$.  

3-34
Figure 3-24. Typical Expanded Non-Multiplexed System
From the cycle-by-cycle description, the reader should note that the address bus provides “Op Code Address” and “Op Code Address + 1” during the first two cycles of every MC6801 instruction. Note that this is also true for single byte inherent instructions such as NOP or MUL. While this may appear to be an anomaly, the explanation is quite simple: when “Op Code Address + 1” appears, the operation code has not yet been decoded to determine the addressing mode of the instruction. The reader should also be able to recognize idle bus cycles such as the last eight cycles of the MUL instruction.

Figure 3-25 presents a sample instruction sequence for a ROM-resident program and indicates its bus activity on a cycle-by-cycle basis when operating in the expanded non-multiplexed mode. If the reader is not familiar with the instruction set, perhaps it would be prudent to skip this section on first reading and return to it after reading Chapter 4.

1. Instruction Sequence

<table>
<thead>
<tr>
<th>Address</th>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>F800</td>
<td>86 AA</td>
<td>LOOP</td>
<td>ORG</td>
<td>$F800</td>
</tr>
<tr>
<td>F802</td>
<td>97 80</td>
<td></td>
<td>LDAA</td>
<td>#$AA</td>
</tr>
<tr>
<td>F804</td>
<td>96 80</td>
<td></td>
<td>STAA</td>
<td>$80</td>
</tr>
<tr>
<td>F808</td>
<td>B7 01 00</td>
<td></td>
<td>STAA</td>
<td>$100</td>
</tr>
<tr>
<td>F809</td>
<td>B6 01 00</td>
<td></td>
<td>LDAA</td>
<td>$100</td>
</tr>
<tr>
<td>F80C</td>
<td>20 F2</td>
<td></td>
<td>BRA</td>
<td>LOOP</td>
</tr>
</tbody>
</table>

2. Bus Activity when Executing the Instruction Sequence

<table>
<thead>
<tr>
<th>Internal Address</th>
<th>Internal Data Bus</th>
<th>External Address Bus</th>
<th>External Data Bus</th>
<th>R/W</th>
<th>IOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>F800</td>
<td>86</td>
<td>00</td>
<td>**</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F801</td>
<td>AA</td>
<td>01</td>
<td>**</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F802</td>
<td>97</td>
<td>02</td>
<td>**</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F803</td>
<td>80</td>
<td>03</td>
<td>**</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0080</td>
<td>AA</td>
<td>80</td>
<td>AA</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F804</td>
<td>96</td>
<td>04</td>
<td>**</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F805</td>
<td>80</td>
<td>05</td>
<td>**</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0080</td>
<td>AA</td>
<td>80</td>
<td>**</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F806</td>
<td>B7</td>
<td>06</td>
<td>**</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F807</td>
<td>01</td>
<td>07</td>
<td>**</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F808</td>
<td>00</td>
<td>08</td>
<td>**</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0100</td>
<td>AA</td>
<td>00</td>
<td>AA</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F809</td>
<td>B6</td>
<td>08</td>
<td>**</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F80A</td>
<td>01</td>
<td>0A</td>
<td>**</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F80B</td>
<td>00</td>
<td>0B</td>
<td>**</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0100</td>
<td>AA</td>
<td>00</td>
<td>AA</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>F80C</td>
<td>20</td>
<td>0C</td>
<td>**</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F80D</td>
<td>F2</td>
<td>0D</td>
<td>**</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FFFF</td>
<td>(FFFF)</td>
<td>FF</td>
<td>**</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

“**" indicates that the Port 3 Data Bus drivers are held in the high impedance state. 
“(FFFF)” indicates the contents of location $FFFF.

Figure 3-25. External Bus — Expanded Non-Multiplexed Mode
3.4 EXPANDED MULTIPLEXED MODES PIN DESCRIPTIONS

In the expanded multiplexed modes, the MCU has the capability to access a 64K byte address space. SC1 is configured as an output and provides a timing signal called Address Strobe which is used to control de-multiplexing of the address and data buses. SC2 is also configured as an output and provides a Data Bus control signal called Read/Write. In Modes 0 through 3, Port 3 is configured as a time multiplexed address/data bus and Port 4 provides the remaining address lines (A8-A15). In Mode 6, however, Port 4 is configured from Reset as an 8-bit data input port. It can then be reconfigured by setting bits in the Port 4 Data Direction Register to provide any combination of address lines, A8 to A15.

3.4.1 SC1: Address Strobe (AS)

In all expanded multiplexed modes, the eight least significant bits of the address bus (A0-A7) are time multiplexed with the data bus. SC1 is configured as AS (Address Strobe) which can be used as a control signal to demultiplex the two buses using a transparent latch such as a 74LS373 or MC6882. AS is derived from alternating transitions of the MCU input clock except that it is held low while E is high.

AS is also used internally to enable the eight least significant address lines to the port. Between logic highs of AS and E, Port 3 is forced to a high impedance state to prevent possible bus contention. AS is designed to drive one Schottky TTL load and 90 pF, and timing is shown in Figure 3-28*.

3.4.2 SC2: R/W (Read/Write)

SC2 is configured as Read/Write in the expanded multiplexed modes and can be used to control the direction of Data Bus transfers. A low level (write) on the R/W line enables the Port 3 Data Bus driver and allows data to be transferred from the MCU to an external device. A high level (read) on the R/W line forces Port 3 to a high impedance state (except in Mode 0) and enables reading the Data Bus. The R/W line is capable of driving one Schottky TTL load and 90 pF, and timing is shown in Figure 3-28.

3.4.3 P30-P37: Port 3 Multiplexed Address/Data Bus

Depending upon the mode, Port 3 can be configured as (1) a parallel I/O port with two handshake control lines, (2) an 8-bit bidirectional data bus, or (3) as a multiplexed address and data bus. In the expanded multiplexed modes, Port 3 is configured to drive a time multiplexed address and data bus. While AS is high, Port 3 provides addresses A0 to A7. While E is high, Port 3 functions as an 8-bit bidirectional Data Bus controlled by R/W. Between AS and E, the port is held in a high impedance state to prevent possible bus conflicts.

With respect to the external data bus, the Port 3 output drivers are three-state devices which remain in the high-impedance (off) state except:

1. during a write operation when R/W is low and E is high (modes 1, 2, 3 and 6), or
2. during a write operation when R/W is low and E is high and during a read operation of an internal MCU location when R/W and E are high (Mode 0).

Each output buffer is capable of driving one Schottky TTL load and 90 pF. Port 3 functions identically in all expanded multiplexed modes except Mode 0 and this exception is discussed in Section 3.4.3.2*

*A more detailed discussion of the relationship between the MCU input clock, E, and AS is included in Appendix I.
Port 3 consists of two major functional elements: an Input/Output buffer and associated control logic. The buffer can be further divided into a three-state output driver, input and output Data Registers, a Data Direction Register, and a bus arbitrator for the MCU data bus. The Data and Data Direction Registers are accessible from the internal Peripheral Data Bus only in the single chip modes. In all other modes, the output signals (either data or addresses) are connected directly to the output drivers.

3.4.3.1 PORT 3 IN EXPANDED MULTIPLEXED MODES 1, 2, 3 AND 6. A logic diagram for Port 3 is provided in Figure 3-26 where signal definitions are as follows:
- POR is an active high internal reset signal which is synchronized with E;
- DBRW3 controls the state of the Port 3 output drivers. It is identical to \( \overline{AS} \) until E; it then follows \( R/W \). Between AS and E, it is low;
- LADD couples the eight least significant address lines to the Port 3 output drivers;
- DWR couples the internal Peripheral Data Bus (PDB) to the Port 3 output driver and is active only during E;
- DIBP couples the Peripheral Data Bus (PDB) to the MPU data bus during an internal MPU read;

![Logic Diagram for Port 3](image)

**Figure 3-26. Logic Diagram for Port 3 (Repeated)**
• DIB3 couples data from the Port 3 input Data Register to the MPU data bus during an MPU read whenever DIBP is not active;
• ALi is the ith line of the address bus where i varies from 0 to 7 (A0 to A7);
• M47, WIO3, DDR3 and WP3 are inactive in the expanded multiplexed modes and are held low;
• P3R is inactive in the expanded multiplexed mode and is held high.

In the expanded multiplexed modes, M47 is low which isolates the output Data Register from the output driver and inhibits the Data Direction Register from controlling the state of the output driver. The eight least significant address lines are coupled to the Port 3 output drivers through the LADD couplers. AS, through DBRW3, enables the output drivers to provide the eight least significant address outputs, ALi.

After hold time, t_AHL, following the negative edge of AS, the output drivers assume a high impedance state. If the MPU is writing, however, DBRW3 will go low during E and couple data from the internal Peripheral Data Bus to the output drivers through the DWR coupler. Therefore when the MPU writes, it drives both the peripheral and external data buses.

During E of an MPU read, DBRW3 keeps the output drivers in a high impedance (off) state. If the address is an internal location, the DIBP signal couples the Peripheral Data Bus to the MPU data bus during E and the output drivers will remain in a high impedance state. If the location is not an internal location, DIB3 will couple the external data bus from the Port 3 input Data Register to the MPU Data Bus.

It should be noted that there is no bus conflict associated with enabling devices to the MCU Data Bus in response to internal MCU addresses in Modes 1, 2, 3, and 6; if this occurs, those devices will be accessed during MPU writes. During MPU reads, however, the Port 3 bus arbitrator will force DIBP high while DIB3 is pulled low and the MPU will read only from the Peripheral Data Bus. Although the address of referenced internal locations will appear on the Address Bus, the Port 3 output drivers will remain in a high-impedance state during E. As a consequence, the external data bus will then be driven only if an external device responds to the address. This characteristic must be kept in mind if attempting to monitor the Data Bus with a logic analyzer. Only in Mode 0 can the Peripheral Data Bus be monitored during an MPU read of an internal address.

NOTE
A logic analyzer can be used to monitor the Peripheral Data Bus during MPU reads of internal locations only in Mode 0.

3.4.3.2 PORT 3 IN EXPANDED MULTIPLEXED TEST MODE 0. In Mode 0, Port 3 and its associated control signals function identical to the other expanded multiplexed modes with two notable exceptions:

First, a two-cycle delay is added to the reset (POR) signal in the DBRW3 and the DIB3 circuit. During the initial two cycles after the positive edge of RESET, DBRW3 forces Port 3 to be an input during E and the DIB3 coupler connects the external and MPU Data Buses. An external address decoder must trap $FFFFE and $FFFFFF on the address bus, only during two cycles after Reset, and provide a Reset vector during these two cycles. Because the Reset vector is fetched from external
memory only during these two cycles, control can be passed to a program residing in external memory which is able to access the entire ROM. Subsequent accesses to the interrupt vector area (SFFFO-$FFFF), such as MPU reads, will address the internal ROM. This feature facilitates testing of the entire ROM pattern.

Finally, in order to monitor the Peripheral Data Bus during MPU reads of internal locations, appropriate logic is incorporated for DBRW3 which enables the Port 3 output drivers during MPU reads of internal locations. In order to avoid bus contention, however, no external device can be enabled to the Data Bus during an MPU read of an internal address in Mode 0.*

3.4.4 P40-P47: Port 4 Address Bus/Data Inputs

Depending upon the mode, Port 4 can be configured as (1) a parallel Input/Output port, (2) an address/input port, or (3) as dedicated address outputs. In the expanded multiplexed modes, it can function either as part of the address bus or as an address/input data port. Port 4 functions significantly different in Modes 0 through 3 than it does in Mode 6. In Modes 0 through 3, the port always provides address lines A8 through A15 as outputs. The port Data and Data Direction Register are not accessible from the Peripheral Data Bus and their addresses are decoded as external memory locations.

In Mode 6, Port 4 is configured from Reset as data port input lines which can be re-configured by software to provide any combination of the eight most significant address lines (A8-A15). In this mode, the output Data Register is isolated from the internal Peripheral Data Bus but both the input Data and Data Direction Register are accessible. The Port 4 Data Direction Register controls the configuration of the port and is cleared during Reset. This configures the port as eight parallel input data lines with internal pullup resistors to VCC. Any combination of address lines A8 through A15 can be obtained by setting the corresponding bits in the Port 4 Data Direction Register where bit 0 controls address line A8.

A timing diagram for MPU reads of the Port 4 input Data Registers is shown in Figure 3-8. During an MPU read of Port 4, valid data must be presented for at least setup time, tPDSU, before the positive edge of E of the read cycle. Data is latched into Port 4 by this same edge and must remain valid for at least hold time, tPDH, after this edge. Data is latched by Port 4 only during the read cycle.

Port 4 consists of two major functional elements: an Input/Output buffer and associated control logic. The Port 4 control logic generates a read signal, R04, and two write signals, WP4 and DDR4, which control accesses to the input Data Register and Data Direction Register. A logic diagram of the Input/Output buffer is repeated in Figure 3-27 and consists of input and output Data Registers, Data Direction Register, and output driver. The output Data Register is accessible from the internal Peripheral Data Bus only in Single Chip Mode. The input Data and Data Direction Registers are accessible only in Modes 4 to 7. In other modes, the most significant half of the address bus is connected directly to the output drivers. Definitions in the logic diagram which are active in Expanded Multiplexed Mode include:

- POR is an active high internal reset signal which is synchronized with E;
- WP4 goes high during E of an MPU write to the Port 4 Data Direction Register (Active only in Mode 6);

*A mode 0 chip select circuit is discussed in Appendix J.
Figure 3-27. Logic Diagram for Port 4 (Repeated)

- DDR4 goes high during $\overline{E}$ after WP4 for an MPU write to the Port 4 Data Direction Register (Active only in Mode 6);
- PDB\textsubscript{i} is the \textit{i}th line of the internal Peripheral Data Bus (PDB);
- RI04 enables a read of the Port 4 input Data Register (Active only in Mode 6);
- MXM is pulled high in all Expanded Multiplexed Modes;
- PC2 is the most significant bit of the operating mode;
- AH\textsubscript{i} are the eight most significant lines of the address bus where \textit{i} varies from 8 to 15 (A8 to A15);
- MOD5 and WI04 are inactive in the expanded multiplexed mode and are held low.

In modes 0 through 3, all of the register control signals (RI04, WP4, DDR4, WI04) are inactive and their associated couplers isolate the Data and Data Direction Register from the internal Peripheral Data Bus. PC2 is held low in these modes and configures the port as address outputs (A8-A15). The MOD5 coupler is inactive and the MXM coupler connects the most significant eight address lines, AH\textsubscript{i}, to the Data Register and output drivers.

In Mode 6, WI04 remains inactive and isolates the output Data Register from the Peripheral Data Bus. The remaining Port 4 register control signals (WP4, DDR4, and RI04) are active and allow the input Data Register and Data Direction Register to be accessed from the internal Peripheral Data Bus. PC2 is high in Mode 6, which allows the Port 4 Data Direction Register to control the configuration of each bit. Reset clears the Data Direction Register which configures the port as eight parallel input lines.
NOTE
In Mode 6, address lines, A8 through A15, are not provided until the Port 4 Data Direction Register is configured using software. Internal pull-up resistors are intended to pull the lines high until this step is performed.

3.4.5 Expanded Multiplexed Bus Timing

The Expanded Multiplexed bus is compatible with the M6800 family of parts. It is a synchronous bus clocked by E (Enable) where every bus cycle is either a read or write cycle. Occasionally, the MPU does not require a memory reference during an E-cycle of a particular instruction. The last cycle of a branch instruction, for example, is an idle bus cycle. All of the address lines and R/W are forced high during idle bus cycles which appears as a read of $FFFF. While RESET is held low, R/W, AS, and address lines A8 to A15 are pulled high while the Port 3 output driver is forced to a high-impedance state.

An expanded multiplexed bus timing diagram is shown in Figure 3-28. During an MPU write cycle, address lines A8 to A15 and R/W become valid tAD after the negative edge of E. The least significant eight lines of the address bus do not become valid, however, until tASM before the rising edge

![Expanded Multiplexed Bus Timing Diagram](image-url)

Figure 3-28. Expanded Multiplexed Bus Timing
of E. An external transparent latch must be provided to capture address lines, A0 to A7, by tAHL after the negative edge of AS. AS can be used to control the latch as shown in Figure 3-29. After the negative edge of AS plus the hold time tAHL, Port 3 is held in a high-impedance state until tDDW after the next positive edge of E at which time the Data Bus becomes valid. Data will remain valid for at least tHW after the next negative edge of E. Note that both the entire address bus (A0-A7 are latched) and R/W are valid tASM before the positive edge of E and remain valid for tHW after the negative edge of E.

During a read bus cycle, addresses A8 to A15 and R/W are valid tAD after the negative edge of E. A transparent latch must be employed to capture A0 through A7 and these addresses become valid tASM before the next positive edge of E. The selected device can be enabled to the Data bus no earlier than tAHL after the negative edge of Address Strobe to avoid contention with the address bus. The address bus and R/W are valid for access time, tACCM, before data is required to be valid. Data must remain valid for at least hold time, tHR, after the next negative edge of E. Note that the address bus remains valid until tAH after the negative edge of E. The maximum access time, tACCM, is a derived amount which is computed from when the latter of the two halves of the address bus becomes valid. Although it is given in the Data Sheet, it can be computed from the following equation:

\[ t_{ACCM} = P_{WEH} + t_{ASM} - t_{DSR} \]

A typical system implementation is shown in Figure 3-30. Additional examples are included in Chapter 8.
Figure 3-30. Typical Expanded Multiplexed System
A somewhat unusual situation can exist while RESET is held low. Expanded Multiplexed Bus activity during Reset will appear as MPU reads of $FFXX where XX is determined by external devices. Further complexity is introduced because AS is held high which means that the least significant lines are never latched. Any "data" that appears during E, therefore, is passed through the latch as "A0 to A7." It is unlikely that any bus contention will result in this situation, however, provided not more than one device responds to addresses $FF00 to $FFFF. This situation will not occur if external pull-up resistors are used for Port 3 or if chip selects are qualified with a low level AS signal.

3.4.6 Monitoring the Expanded Multiplexed Bus

The MC6801 Data Sheet and Appendix F contain a cycle-by-cycle description of the bus activity for each instruction where the data shown is with respect to the MCU internal buses. Some interpretation of this table is required, however, to obtain an exact description for a particular MC6801 bus. While the external address bus will always appear as indicated, the external data bus may not always agree with what is shown in the table. Except in Mode 0, Port 3 — which provides the data bus — is held in the high impedance state during MPU read cycles of internal locations. If executing a program from the internal ROM, these cycles will constitute a significant portion of bus activity.

When using a logic analyzer with the expanded multiplexed bus, the reader should remember that:

1. it should be triggered with the negative edge of E (Enable) and be connected to the de-multiplexed bus,
2. the external and internal address buses will always have the same value,
3. the external and internal data buses will always have the same value for MPU writes and MPU reads of external locations,
4. unless in Mode 0, the external data bus is not valid during MPU read cycles of internal locations.

When examining the cycle-by-cycle description, the reader should note that the address bus always provides "Op Code Address" and "Op Code Address + 1" during the first two cycles of every MC6801 instruction. While this may appear unusual, the explanation is quite simple: when the "Op Code Address + 1" appears on the Address Bus, the MPU has not yet decoded the operation code to determine the addressing mode of the instruction. The reader should also be able to recognize idle bus cycles such as the last eight cycles of the MUL instruction.

The sample instruction sequence presented in Figure 3-31 is included to illustrate the bus traffic on a cycle-by-cycle basis for Mode 1 where the program resides in external memory at $1000. If it was resident in the ROM at, for example, $F800, the bus activity would be identical to that shown in Figure 3-25 except: (1) the column for IOS is not applicable, and (2) the external address bus would be identical to the internal address bus. If the reader is not familiar with the instruction set, perhaps it would be wise to skip this section and return to it after reading Chapter 4.
1. Instruction Sequence

<table>
<thead>
<tr>
<th>Address</th>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>B6 AA</td>
<td>LOOP</td>
<td>LDAA</td>
<td>#$AA</td>
</tr>
<tr>
<td>1002</td>
<td>97 80</td>
<td>STAA</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td>96 80</td>
<td>LDA</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td>B7 01 00</td>
<td>STAA</td>
<td>$100</td>
<td></td>
</tr>
<tr>
<td>1009</td>
<td>B6 01 00</td>
<td>LDA</td>
<td>$100</td>
<td></td>
</tr>
<tr>
<td>100C</td>
<td>20 F2</td>
<td>BRA</td>
<td>LOOP</td>
<td></td>
</tr>
</tbody>
</table>

2. Bus Activity when Executing the Instruction Sequence (For Mode 1)

<table>
<thead>
<tr>
<th>Internal Address</th>
<th>Internal Data Bus</th>
<th>External Address</th>
<th>External Data Bus</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>86</td>
<td>1000</td>
<td>86</td>
<td>1</td>
</tr>
<tr>
<td>1001</td>
<td>AA</td>
<td>1001</td>
<td>AA</td>
<td>1</td>
</tr>
<tr>
<td>1002</td>
<td>97</td>
<td>1002</td>
<td>97</td>
<td>1</td>
</tr>
<tr>
<td>1003</td>
<td>80</td>
<td>1003</td>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>0080</td>
<td>AA</td>
<td>0080</td>
<td>AA</td>
<td>0</td>
</tr>
<tr>
<td>1004</td>
<td>96</td>
<td>1004</td>
<td>96</td>
<td>1</td>
</tr>
<tr>
<td>1005</td>
<td>80</td>
<td>1005</td>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>0080</td>
<td>AA</td>
<td>0080</td>
<td>**</td>
<td>1</td>
</tr>
<tr>
<td>1006</td>
<td>B7</td>
<td>1006</td>
<td>B7</td>
<td>1</td>
</tr>
<tr>
<td>1007</td>
<td>01</td>
<td>1007</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>1008</td>
<td>00</td>
<td>1008</td>
<td>00</td>
<td>1</td>
</tr>
<tr>
<td>0100</td>
<td>AA</td>
<td>0100</td>
<td>AA</td>
<td>0</td>
</tr>
<tr>
<td>1009</td>
<td>B6</td>
<td>1009</td>
<td>B6</td>
<td>1</td>
</tr>
<tr>
<td>100A</td>
<td>01</td>
<td>100A</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>100B</td>
<td>00</td>
<td>100B</td>
<td>00</td>
<td>1</td>
</tr>
<tr>
<td>0100</td>
<td>AA</td>
<td>0100</td>
<td>AA</td>
<td>1</td>
</tr>
<tr>
<td>100C</td>
<td>20</td>
<td>100C</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>100D</td>
<td>F2</td>
<td>100B</td>
<td>F2</td>
<td>1</td>
</tr>
<tr>
<td>FFFF</td>
<td>(FFFF)</td>
<td>FFFF</td>
<td>(FFFF)</td>
<td>1</td>
</tr>
</tbody>
</table>

"***" indicates that the Port 3 Data Bus drivers are held in the high impedance state.
"(FFFF)" indicates the contents of location $FFFF.

Figure 3-31. External Bus — Expanded Multiplexed Mode
4.0 INTRODUCTION

The Microprocessor Unit (MPU) is that portion of the MC6801 which executes the instruction set and can be considered an enhanced version of the MC6800. Several new instructions have been added and its internal organization has been improved to yield greater throughput for many instructions. Both the source and object code of the MC6800 are upward compatible with the MC6801. Instruction cycle counts, however, have been reduced for many instructions.

A programming model of the MC6801 is shown in Figure 4-1. The MPU includes two 8-bit accumulators, A and B, which can be concatenated to form a double byte accumulator referred to as accumulator D or A:B where accumulator A contains the most significant byte. The MPU also includes a 16-bit Index Register, a 16-bit Stack Pointer, a 6-bit Condition Code Register, and a 16-bit Program Counter. The MC6801 programming model is identical to the model for the MC6800 except that the two accumulators can be concatenated for double byte instructions.

![Figure 4-1. MC6801 MPU Programming Model](image-url)
This chapter is concerned with the definition and application of the MC6801 instruction set and assumes that the reader has no experience with the MC6800. However, many comparisons between the two MPUs are offered for the benefit of those readers who are familiar with the MC6800. Before beginning a detailed discussion of the instruction set, however, it is first necessary to introduce a concept which will be used extensively in this discussion: symbolic addressing.

While entire programs can be written in the MC6801 natural machine code language, it is usually a tedious and error-prone operation. It is much more convenient to prepare a program using mnemonics and symbols with which to reference both machine operations and memory locations. The resultant program can then be processed by yet another program — called an Assembler — which will interpret the symbols and produce equivalent MC6801 machine code. The use of symbols to act as surrogates for actual memory locations is known as symbolic addressing. This discussion will utilize the symbolic addressing capability of the MC6801 Assembly Language as the vehicle on which to base this MPU discussion.

The reader should be advised, however, that while the MC6801 machine code is fixed, assembly language can vary according to the whims of the programmer writing the assembler program. This discussion uses the assembly language which is compatible with Motorola's M6800 family assemblers and is intended to be supplemented with the MDOS Macroassembler Reference Manual [Publication No. M68MASR(D)].

4.1 ASSEMBLER SOURCE STATEMENTS

While programs can be written in the MC6801 machine code language, there is no convenient method available to associate the operations with their corresponding machine code value. For this reason, machine instructions are assigned a three or four letter mnemonic which is intended to be suggestive of its operation. The fundamental entity in an assembly language program is the “statement” which can be translated into no more than one machine instruction. A program is written as a series of statements using symbolic language. The symbols can be manually translated into machine code using a table of mnemonics and symbols and their corresponding machine code. Typically, however, the translation is performed by an Assembler.

During the process of assembly, each statement invoking a machine instruction is converted to one, two or three bytes of machine code depending upon the addressing mode of the instruction. A special type of statement, an assembly directive, is useful in controlling and documenting the program but generates no machine code. Typical assembler directives include PAGE (go to top of next page), ORG (set assembler location counter), and SPC (space). These and additional assembler directives are defined in the Macroassembler Reference Manual.

An assembly language statement consists of one to four fields: label, operation, operand, and comment. An optional fifth field, a sequence number, can also be included but is omitted in this discussion. The four fields are separated by one or more spaces as illustrated in the following statement:

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>7D 10 00</td>
<td>BEGIN1</td>
<td>TST</td>
<td>DATA</td>
<td>TEST BYTE AT DATA</td>
</tr>
</tbody>
</table>
This instruction causes the MPU to test the contents of the memory location associated with the symbol, DATA, and set the Condition Code Register bits accordingly. The corresponding three bytes of machine code are also shown (in hexadecimal) where DATA is located at 1000 (hex). The machine code is the result of translation and is not part of the statement.

Every statement must include at least a mnemonic in the operation field. A symbol in the label field is required if the location is to be symbolically addressed in the operand field of another instruction. An operand field could be required depending upon the type of instruction. The comment field is always optional — at the convenience of the programmer — for describing and documenting the program.

4.1.1 Labels

Labels can correspond with either a specific numerical value (Equate directive) or the address of a memory location. The memory location can represent the destination of a branch instruction or the start of a data area. This use of symbolic references to memory allows statements to be written without specifying actual memory locations. For instance, the symbol, DATA, in the above example can reside anywhere in memory. An entry in the label field is required for all statements which are the destination of jump and branch instructions and in statements using the EQU (Equate) directive. In the above example, BEGIN1 serves to identify the location if it is used as the destination of a branch or jump instruction located elsewhere in the program. That instruction will, in turn, have BEGIN1 in its operand field.

Labels consist of one to six characters using any alphanumeric combination of the letters A-Z, digits 0-9, or the two special characters, “.” (period) and “$” (dollar). The first character in any label, however, must be either a letter or the character, “.”. Exceptions include three single character labels, A, B, and X which are reserved by the assembler for referencing accumulator A, accumulator B, and the Index Register, respectively. Note that “D” is not a reserved symbol. The programmer cannot put a space before the D in the three double accumulator shift instructions (LSLD, ASLD, LSRD) in order to avoid ambiguities with the single byte shift instruction and a symbol called “D.”

If the label field is included, it must begin in the first character position of the statement. Two other characters which can also occupy this position have special meaning to the Assembler as indicated below.

<table>
<thead>
<tr>
<th>First Character</th>
<th>Assembler Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(blank)</td>
<td>No label field</td>
</tr>
<tr>
<td>*</td>
<td>Comment statement; print only</td>
</tr>
<tr>
<td>A-Z, “.”</td>
<td>First character of label field</td>
</tr>
</tbody>
</table>

The Macroassembler Reference Manual should be consulted for more detailed information.

4.2 ADDRESSING MODES

Memory references always appear in the operand field of a statement where the particular method used is called an “addressing mode.” The number of different addressing modes is often an indicator of the flexibility and power of the processor instruction set. The MC6801 has a total of six addressing modes: (1) inherent, (2) immediate, (3) extended, (4) direct, (5) relative and (6) indexed.
Each of the addressing modes (except inherent) results in an internally generated double byte value referred to in this discussion as the instruction "effective address." This is the resultant value of a statement operand field and is the value which appears on the address bus during the memory reference cycle. The addressing mode is an implicit part of every MC6801 opcode.

4.2.1 Inherent Addressing Mode

Many MC6801 instructions do not require an operand because the effective address is inherent in the instruction itself. For instance, the instruction ABA causes the MPU to add the contents of accumulators A and B and place the result in accumulator A. The instruction INCB causes the contents of accumulator B to be increased by one. Similarly, INX, causes the Index Register to be increased (incremented) by one. These three examples of inherent instructions do not require an operand. The statements and machine code generated for them are shown below.

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td></td>
<td>ABA</td>
<td>A+B-A</td>
<td></td>
</tr>
<tr>
<td>5C</td>
<td></td>
<td>INCB</td>
<td>B+1-B</td>
<td></td>
</tr>
<tr>
<td>1B</td>
<td></td>
<td>INX</td>
<td>X+1-X</td>
<td></td>
</tr>
</tbody>
</table>

The reader should note that all inherent instructions require only a single byte of machine code and have no operand field in the statement.

4.2.2 Immediate Addressing Mode

In the immediate addressing mode, the machine code byte(s) which follow the operation code is the value of the statement operand field rather than the address of a value. The effective address of the instruction in this case is specified by the "#" sign and implicitly points to the byte following the opcode. The immediate value is limited to either one or two bytes depending only on the size of the register also included in the statement. Examples of several statements which use the immediate addressing mode are shown below. Symbols used in these statements are defined immediately after the examples.

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>86 16</td>
<td></td>
<td>LDAA</td>
<td>#22</td>
<td>22~ACCA</td>
</tr>
<tr>
<td>C8 34</td>
<td></td>
<td>EORB</td>
<td>#$34</td>
<td>XOR ($34,ACCB)</td>
</tr>
<tr>
<td>81 24</td>
<td>CAT</td>
<td>CMPA</td>
<td>#10100</td>
<td>CMPA #124</td>
</tr>
<tr>
<td>86 07</td>
<td></td>
<td>LDA A</td>
<td>#CAT</td>
<td>CAT SAME AS 7</td>
</tr>
<tr>
<td>86 12</td>
<td></td>
<td>LDD</td>
<td>#$1234</td>
<td>7~ACCA:ACCB</td>
</tr>
<tr>
<td>86 41</td>
<td></td>
<td>LDA A</td>
<td>#A</td>
<td>OCTAL</td>
</tr>
<tr>
<td>CE 10 00</td>
<td></td>
<td>LDX</td>
<td>#TABLE</td>
<td>ASCII</td>
</tr>
</tbody>
</table>

The reader should examine the above machine code and note that the value of each statement operand field appears in the byte(s) immediately following the opcode. Note also that the operand field for immediate addressing begins with the character, "#." The "#" is used by the assembler to detect the immediate mode of addressing.
A variety of symbols and expressions can be used following the "#" sign. The *Macroassembler Reference Manual* should be consulted for a complete list of possibilities. The prefixes used in the above example have the following meanings.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Decimal</td>
</tr>
<tr>
<td>$</td>
<td>Hexadecimal</td>
</tr>
<tr>
<td>@</td>
<td>Octal</td>
</tr>
<tr>
<td>%</td>
<td>Binary</td>
</tr>
<tr>
<td>'</td>
<td>Single ASCII character</td>
</tr>
</tbody>
</table>

In the last statement of the above example, the immediate bytes consist of the "value" of the symbol, TABLE. The value of any symbol is equal to its address except when it is used in the label field of an EQU (Equate) statement. The value of a symbol appearing in label field of an EQU directive is defined by the value in the operand field of the statement.

### 4.2.3 Direct and Extended Addressing Modes

In the extended addressing mode, the effective address of the instruction appears explicitly in the two bytes following the opcode. Therefore, the length of all instructions using the extended addressing mode is three bytes: one for the opcode and two for the effective address.

In the direct addressing mode, the most significant byte of the effective address is assumed to be zero (000) and the least significant byte is specified in the byte following the operation code. The length of all instructions using the direct addressing mode is two bytes: one for the opcode and one for the least significant byte of the effective address. Thus, the extended and direct addressing modes differ in two respects: (1) the range of memory that can be accessed and (2) the length of the instruction. Using direct addressing, an instruction can reference memory only within the range $0000-$00FF whereas in the extended mode of addressing the entire memory space can be accessed.

The addressing mode (with respect to direct or extended) is selected by the assembler although the programmer can indirectly affect its decision by judicious placement of statements. If the symbol being referenced in the operand field appears in the label field after the current statement (forward reference), the extended mode of addressing will be selected regardless of the symbol value. If, however, the label has already appeared in the label field (backward reference) then the choice depends only on the value of the symbol. If it is less than $100 then the direct mode of addressing will be used; otherwise, the assembler will select the extended mode of addressing.

There are some instructions which provide an extended addressing mode but not a direct mode. These instructions are members of a group called "read-modify-write" instructions* which operate directly on memory, M, and have the form

\[ <\text{operation}> M \rightarrow M \]

The instructions INC, DEC, CLR, and COM are members of this group: each has an extended addressing mode but no direct mode. The following sequence illustrates the direct and extended mode of addressing.

*Opcodes $40 to $7F (except JMP) in Appendix B.
In the above sequence, the first reference to the symbol, CAT, was a forward reference and the assembler selected the extended addressing mode. The second reference was a backward reference which enabled the assembler to know its value when processing the statement and it selected the direct addressing mode. Note that while the last reference to CAT is also a backward reference to a symbol in the direct area, the extended addressing mode was selected because the particular instruction does not have a direct addressing mode.

### 4.2.4 Relative Addressing Mode

In both the direct and extended modes, the address contained in the operand byte(s) is an absolute numerical address. The relative addressing mode is used only for branch instructions and specifies a location relative to the current value of the Program Counter. The Program Counter will always point to the next statement in line while the addition is being performed. A zero offset byte, therefore, will result in no branch regardless of the test involved.

Branch instructions always generate two bytes of machine code: one for the opcode and one for the relative offset. Because it is desirable to branch in either direction, the offset byte is a signed two's complement offset with a range of $-128$ to $+127$ bytes. The effective branch range, however, must be computed with respect to the address of the next instruction in line. A branch instruction consists of two bytes which always places the next location at PC + 2. If R is defined as the address of the branch destination, the range is then given by:

$$ \text{PC} + 2 - 128 \leq R \leq \text{PC} + 2 + 127 $$

or

$$ \text{PC} - 126 \leq R \leq \text{PC} + 129 $$

This result indicates that the destination of the branch instruction must be within $-126$ to $+129$ memory locations of the first byte of the branch instruction itself. If it is desired to transfer control beyond this range, then the JMP or JSR instruction must be used. Examples of relative addressing are shown in the following sequence.

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 08</td>
<td>THERE</td>
<td>BCC</td>
<td>LBCC</td>
<td>L-O-N-G BCC</td>
</tr>
<tr>
<td>20 00</td>
<td>WHERE</td>
<td>BRA</td>
<td>WHERE</td>
<td>FORWARD BRANCH</td>
</tr>
<tr>
<td>22 FC</td>
<td>WHERE</td>
<td>BHI</td>
<td>THERE</td>
<td>BACKWARD BRANCH</td>
</tr>
<tr>
<td>27 FE</td>
<td>HANG</td>
<td>BEQ</td>
<td>HANG</td>
<td>BRANCH TO SELF</td>
</tr>
<tr>
<td>27 FE</td>
<td>HANG</td>
<td>BEQ</td>
<td>*</td>
<td>*MEANS &quot;HERE&quot;</td>
</tr>
<tr>
<td>7E 10 00</td>
<td>LBCC</td>
<td>JMP</td>
<td>$1000</td>
<td></td>
</tr>
<tr>
<td>8D F7</td>
<td>BSR</td>
<td>HANG</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.2.5 Indexed Addressing Mode

With indexed addressing, the effective address is variable and depends upon two factors: (1) the current contents of the Index Register and (2) the offset contained in the second byte of the instruction. In microprocessor-based systems, instructions usually reside in Read-Only-Memory (ROM). Therefore, the offset in the instruction should be considered a static value determined at assembly time rather than during program execution. The use of a dynamic single byte offset is facilitated with the use of the ABX (Add ACCB to Index Register) instruction.

Every indexed instruction requires two bytes regardless of the value of the offset. If no offset is specified or desired, the instruction will contain $00 in the offset byte. The offset is an unsigned single byte value which when added to the current value in the Index Register yields the effective address of the instruction leaving the Index Register unchanged. Note that because the offset byte is unsigned, a negative offset cannot be specified.

Examples of the indexed addressing mode are shown in the following statements where “EA” indicates “effective address.”

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>E3 00</td>
<td></td>
<td>ADDD</td>
<td>X</td>
<td>EA=(X)</td>
</tr>
<tr>
<td>E3 00</td>
<td></td>
<td>ADDD</td>
<td>, X</td>
<td>EA=(X)</td>
</tr>
<tr>
<td>E3 00</td>
<td></td>
<td>ADDD</td>
<td>0, X</td>
<td>EA=(X)</td>
</tr>
<tr>
<td>E3 04</td>
<td>CAT</td>
<td>ADDD</td>
<td>4, X</td>
<td>EA=(X)+4</td>
</tr>
<tr>
<td>E3 07</td>
<td></td>
<td>EQU</td>
<td>7</td>
<td>DEFINE CAT = 7</td>
</tr>
<tr>
<td>E3 22</td>
<td></td>
<td>ADDD</td>
<td>CAT, X</td>
<td>EA=(X)+7</td>
</tr>
<tr>
<td>E3 22</td>
<td></td>
<td>ADDD</td>
<td>$22, X</td>
<td>EA=(X)+$22</td>
</tr>
<tr>
<td>E3 22</td>
<td></td>
<td>ADDD</td>
<td>CAT*8/2+6, X</td>
<td>EA=(X)+(CAT*8/2+6)</td>
</tr>
</tbody>
</table>

4.3 MC6801 INSTRUCTION SET

The MC6801 instruction set is described in detail in Appendix A. This section will provide a brief introduction and discuss its use in developing MC6801 programs.

The instruction set is shown in summary form in Figure 4-2. Instruction sets are often divided into three general classifications: (1) memory reference, so called because they access memory; (2) inherent instructions which function without a memory reference, and (3) I/O instructions for transferring data between the MPU and peripheral devices. A summary of MC6801 instructions which are different from the MC6800 is shown in Table 4-1 for the convenience of those readers who desire a comparison of the two instruction sets.

For many instructions, the MC6801 performs the same operation using either its internal accumulators or external memory locations. In addition, M6800 Family parts allow the MPU to treat peripheral devices nearly like other memory locations and no specific I/O instructions are required. Because of these features, another classification is considered more suitable for introducing the MC6801 instruction set: (1) Condition Code Register instructions, (2) Accumulator and Memory instructions, and (3) Program Control instructions.
<table>
<thead>
<tr>
<th>Accumulator &amp; Memory Operations</th>
<th>MNE</th>
<th>Immed</th>
<th>Direct</th>
<th>Index</th>
<th>Extend</th>
<th>Inher</th>
<th>Boolean Expression</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Accum</td>
<td>ABA</td>
<td>1B</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + B = A</td>
<td>1 1 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>Add B to X</td>
<td>ABX</td>
<td>3A</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0.0 B + X = X</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Add with Carry</td>
<td>ADDC</td>
<td>8B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = C</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Shift, Logical</td>
<td>ASL</td>
<td>48</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Add</td>
<td>ADDA</td>
<td>8B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Add Double</td>
<td>ADDDB</td>
<td>C2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>B + M = B</td>
</tr>
<tr>
<td>AND</td>
<td>ANDA</td>
<td>8B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + B = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Shift, Logical</td>
<td>ASL</td>
<td>6B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Bit Test</td>
<td>BITA</td>
<td>8B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Compare</td>
<td>CMPA</td>
<td>8B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>1's Complement</td>
<td>CLRA</td>
<td>4F</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Decrement</td>
<td>DEC</td>
<td>6B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Exclusive OR</td>
<td>EORB</td>
<td>8B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Increment</td>
<td>INC</td>
<td>6B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Load Accs</td>
<td>LDAA</td>
<td>8B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Load Double</td>
<td>LDD</td>
<td>6B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Logical Shift, Logical</td>
<td>LSL</td>
<td>6B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Shift Right, Logical</td>
<td>ASR</td>
<td>6B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Multiply</td>
<td>MUL</td>
<td>6B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>No Operation</td>
<td>NOP</td>
<td>6B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Inclusive OR</td>
<td>ORAA</td>
<td>8A</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Push Data</td>
<td>PSHA</td>
<td>8A</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Pull Data</td>
<td>PULA</td>
<td>6B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Rotate Left</td>
<td>ROL</td>
<td>6B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Rotate Right</td>
<td>ROR</td>
<td>6B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Subtract Accm</td>
<td>SBA</td>
<td>82</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Subtract with Carry</td>
<td>SBCA</td>
<td>82</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Store Accm</td>
<td>STA</td>
<td>C2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Subtract</td>
<td>SUBA</td>
<td>82</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Subtract Double</td>
<td>SUBAD</td>
<td>82</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>Transfer Accm</td>
<td>TAB</td>
<td>6D</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>A + M = A</td>
<td>1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

Figure 4-2. MC6801 Instruction Set Summary

4-8
### Jump and Branch Operations

<table>
<thead>
<tr>
<th>Branch Always</th>
<th>Branch If Carry Clear</th>
<th>Branch If Carry Set</th>
<th>Branch If Equal Zero</th>
<th>Branch If Not Equal Zero</th>
<th>Branch If Overflow Clear</th>
<th>Branch If Overflow Set</th>
<th>Branch If Plus</th>
<th>Branch Never</th>
<th>Branch If Higher or Same</th>
<th>Branch If Lower or Same</th>
<th>Branch to Subroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRA</td>
<td>BCC</td>
<td>BCS</td>
<td>BEQ</td>
<td>BGE</td>
<td>BGT</td>
<td>BHI</td>
<td>BLE</td>
<td>BNE</td>
<td>BVS</td>
<td>BPL</td>
<td>BRN</td>
</tr>
<tr>
<td>20 3 2</td>
<td>24 3 2</td>
<td>25 3 2</td>
<td>27 3 2</td>
<td>2C 3 2</td>
<td>2E 3 2</td>
<td>22 3 2</td>
<td>2F 3 2</td>
<td>26 3 2</td>
<td>28 3 2</td>
<td>2A 3 2</td>
<td>21 3 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Index Register Operations

<table>
<thead>
<tr>
<th>MNE</th>
<th>Instruction</th>
<th>M</th>
<th>Relative</th>
<th>Direct</th>
<th>Index</th>
<th>Extend</th>
<th>Inherent</th>
<th>Branch Test</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Op − #</td>
<td>Op − #</td>
<td>Op − #</td>
<td>Op − #</td>
<td>Op − #</td>
<td>Op − #</td>
<td>H</td>
</tr>
<tr>
<td>Compare Index Register</td>
<td>CPX</td>
<td>8C 4 3</td>
<td>9C 5 2</td>
<td>AC 6 2</td>
<td>BC 6 3</td>
<td></td>
<td></td>
<td>X = M+1</td>
<td>*11111</td>
</tr>
<tr>
<td>Decrement Index Register</td>
<td>DEX</td>
<td>0B 3 1</td>
<td>X − M−1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X − M−1</td>
<td>*11111</td>
</tr>
<tr>
<td>Increment Index Register</td>
<td>IX</td>
<td>07 3 1</td>
<td>X − M−1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X − M−1</td>
<td>*11111</td>
</tr>
<tr>
<td>Load Index Register</td>
<td>LX</td>
<td>9B 3 1</td>
<td>X − M−1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X − M−1</td>
<td>*11111</td>
</tr>
<tr>
<td>Store Index Register</td>
<td>SX</td>
<td>9C 3 1</td>
<td>X − M−1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X − M−1</td>
<td>*11111</td>
</tr>
</tbody>
</table>

### Stack Pointer Operations

<table>
<thead>
<tr>
<th>MNE</th>
<th>Instruction</th>
<th>M</th>
<th>Relative</th>
<th>Direct</th>
<th>Index</th>
<th>Extend</th>
<th>Inherent</th>
<th>Branch Test</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Op − #</td>
<td>Op − #</td>
<td>Op − #</td>
<td>Op − #</td>
<td>Op − #</td>
<td>Op − #</td>
<td>H</td>
</tr>
<tr>
<td>Decrement Stack Pointer</td>
<td>DES</td>
<td>8B 3 1</td>
<td>9E 4 2</td>
<td>AE 5 2</td>
<td>BE 5 3</td>
<td></td>
<td></td>
<td>S = 0</td>
<td>*11000</td>
</tr>
<tr>
<td>Increment Stack Pointer</td>
<td>INS</td>
<td>31 3 1</td>
<td>SP+1−SP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S = 0</td>
<td>*11000</td>
</tr>
<tr>
<td>Load Stack Pointer</td>
<td>LDS</td>
<td>34 3 1</td>
<td>M+1+SP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S = 0</td>
<td>*11000</td>
</tr>
<tr>
<td>Store Stack Pointer</td>
<td>STS</td>
<td>35 3 1</td>
<td>X − M+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S = 0</td>
<td>*11000</td>
</tr>
<tr>
<td>Transfer X to SP</td>
<td>TXS</td>
<td>30 3 1</td>
<td>X − M−1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S = 0</td>
<td>*11000</td>
</tr>
<tr>
<td>Transfer SP to X</td>
<td>TTX</td>
<td>36 3 1</td>
<td>M+1+SP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S = 0</td>
<td>*11000</td>
</tr>
</tbody>
</table>

### Condition Code Register Operations

<table>
<thead>
<tr>
<th>MNE</th>
<th>Instruction</th>
<th>M</th>
<th>Relative</th>
<th>Direct</th>
<th>Index</th>
<th>Extend</th>
<th>Inherent</th>
<th>Branch Test</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Op − #</td>
<td>Op − #</td>
<td>Op − #</td>
<td>Op − #</td>
<td>Op − #</td>
<td>Op − #</td>
<td>H</td>
</tr>
<tr>
<td>Clear Carry</td>
<td>CLC</td>
<td>0C 2 1</td>
<td>0−C</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Clear Interrupt Mask</td>
<td>CLI</td>
<td>0E 2 1</td>
<td>0−C</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Clear Overflow</td>
<td>CLV</td>
<td>0A 2 1</td>
<td>0−V</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Set Carry</td>
<td>SEC</td>
<td>0D 2 1</td>
<td>1−C</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Set Interrupt Mask</td>
<td>SEI</td>
<td>0F 2 1</td>
<td>1−C</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Set Overflow</td>
<td>SEV</td>
<td>0B 2 1</td>
<td>1−V</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Accumulator A → CCR</td>
<td>TAP</td>
<td>06 2 1</td>
<td>A−CCR</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>CCR → Accumulator A</td>
<td>TPA</td>
<td>07 2 1</td>
<td>CCR−A</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>

**Legend:**
- **OP**: Operation Code (Hexadecimal)
- **#**: Number of Program Bytes
- **+**: Arithmetic Plus
- **~**: Arithmetic Minus
- **=**: Boolean AND
- **Boolean Inclusive OR**: OR
- **Boolean Exclusive OR**: A
- **M**: Memory References
- **Z**: Zero (Byte)
- **V**: Overflow (2's Complement)
- **A**: Accumulator A
- **C**: Carry From Bit 7
- **H**: Half Carry From Bit 3
- **S**: Set Always
- **I**: Affected Boolean
- **R**: Reset Boolean

**Figure 4-2. MC6801 Instruction Set Summary (Continued)**
Table 4-1. MC6801 Additional Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABX</td>
<td>Unsigned addition of Accumulator B to Index Register</td>
</tr>
<tr>
<td>ADDD</td>
<td>Adds (without carry) the double accumulator to memory and leaves the sum in the double accumulator</td>
</tr>
<tr>
<td>ASLD or LSLD</td>
<td>Shifts the double accumulator left (towards MSB) one bit; the LSB is cleared and the MSB is shifted into the C-bit.</td>
</tr>
<tr>
<td>BHS</td>
<td>Branch if Higher or Same; Unsigned conditional branch (same as BCC)</td>
</tr>
<tr>
<td>BLO</td>
<td>Branch if Lower; Unsigned conditional branch (same as BCS)</td>
</tr>
<tr>
<td>BRN</td>
<td>Branch Never</td>
</tr>
<tr>
<td>JSR</td>
<td>Additional addressing mode: direct</td>
</tr>
<tr>
<td>LDD</td>
<td>Loads double accumulator from memory</td>
</tr>
<tr>
<td>LSL</td>
<td>Shifts memory or accumulator left (towards MSB) one bit: the LSB is cleared and the MSB is shifted into the C-bit (same as ASL)</td>
</tr>
<tr>
<td>LSRD</td>
<td>Shifts the double accumulator right (towards LSB) one bit; the MSB is cleared and the LSB is shifted into the C-bit.</td>
</tr>
<tr>
<td>MUL</td>
<td>Unsigned multiply: multiplies the two accumulators and leaves the product in the double accumulator.</td>
</tr>
<tr>
<td>PSHX</td>
<td>Pushes the Index Register on the stack</td>
</tr>
<tr>
<td>PULX</td>
<td>Pulls the Index Register from the stack</td>
</tr>
<tr>
<td>STD</td>
<td>Stores the double accumulator in memory.</td>
</tr>
<tr>
<td>SUBD</td>
<td>Subtracts memory from the double accumulator and leaves the difference in the double accumulator.</td>
</tr>
<tr>
<td>CPX</td>
<td>Internal processing modified to permit its use with any conditional branch instruction.</td>
</tr>
</tbody>
</table>

4.3.1 Condition Code Register Instructions

The Condition Code Register (CCR) is discussed first because it is affected during the execution of many other instructions in addition to the specific operations shown in Figure 4-2. The five flag bits and one control bit of the CCR are defined in Figure 4-3.

The instructions shown in Figure 4-4 can be used to directly manipulate the CCR. In addition, the MPU automatically sets or clears the appropriate status bits as other instructions are executed. The effect on the condition code register of these instructions will be indicated as they are introduced and is also shown in the Instruction Set Summary of Figure 4-2.

Instructions directly affecting the I-bit (such as CLI and SEI) affect the interrupt structure and are discussed in more detail in Chapter 5. Bits 6 and 7 of the CCR are effectively read-only bits fixed at “1” and have no effect on processor operation.

4.3.2 Accumulator and Memory Instructions

For familiarization purposes, Accumulator and Memory instructions can be further subdivided into four categories: (1) Arithmetic Operations; (2) Logic Operations; (3) Data Testing; and (4) Data Handling.
Half-Carry; set whenever a carry from $b_3$ to $b_4$ of the result is generated by ADD, ABA, ADC; cleared if no $b_3$ to $b_4$ carry; not affected by other instructions.

Interrupt Mask; set by hardware or software interrupt or SEI instruction; cleared by CLI instruction. (Normally not used in arithmetic instructions.) Restored to a zero as a result of an RTI instruction if $I_7$ stored on the stack is low.

Negative, set if high order bit (b7) of result is set; cleared otherwise.

Zero; set if result = 0; cleared otherwise.

Overflow; set if there was arithmetic overflow as a result of the operation, cleared otherwise.

Carry; set if there was a carry from the most significant bit (b7) of the result; cleared otherwise. For subtraction, the C-bit represents the binary borrow.

Figure 4-3. Condition Code Register Bit Definitions

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Mnemonic</th>
<th>Boolean Operation</th>
<th>CCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Carry</td>
<td>CLC</td>
<td>0 → C</td>
<td>• • • • R</td>
</tr>
<tr>
<td>Clear Interrupt Mask</td>
<td>CLI</td>
<td>0 → I</td>
<td>• R • • •</td>
</tr>
<tr>
<td>Clear Overflow</td>
<td>CLV</td>
<td>0 → V</td>
<td>• • • R</td>
</tr>
<tr>
<td>Set Carry</td>
<td>SEC</td>
<td>1 → C</td>
<td>• • • S</td>
</tr>
<tr>
<td>Set Interrupt Mask</td>
<td>SEI</td>
<td>1 → I</td>
<td>• S • •</td>
</tr>
<tr>
<td>Set Overflow</td>
<td>SEV</td>
<td>1 → V</td>
<td>• • • S</td>
</tr>
<tr>
<td>ACCA → CCR</td>
<td>TAP</td>
<td>ACCA → CCR</td>
<td>•</td>
</tr>
<tr>
<td>CCR → ACCA</td>
<td>TPA</td>
<td>CCR → ACCA</td>
<td>•</td>
</tr>
</tbody>
</table>

See Figure 4-2 for legend.

1 (ALL) Set according to the contents of Accumulator A.

Figure 4-4. Condition Code Register Instructions

4.3.2.1 ARITHMETIC INSTRUCTIONS. The arithmetic instructions and their effect on the CCR are shown in Figure 4-5. The MC6801 supports arithmetic operations with single byte values using a single accumulator or double byte values using the D accumulator. Multibyte arithmetic is supported through memory reference instructions.

Number systems directly supported include two's complement binary, unsigned binary, and packed BCD. Further applications of the arithmetic instructions are included in Section 4.4.4.

4.3.2.2 LOGIC INSTRUCTIONS. The logic instructions and their effect on the CCR are shown in Figure 4-6. The COM instruction differs from the other logic instructions in two respects: (1) it can be used to operate directly on memory in addition to the accumulators, and (2) it always sets the carry bit.

4-11
### Boolean/Arithmetic Instructions

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Mnemonic</th>
<th>Boolean/Arithmetic Operation</th>
<th>CCR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Add</strong></td>
<td>ADDA</td>
<td>A + M → A</td>
<td>1 1 1 1 1</td>
</tr>
<tr>
<td></td>
<td>ADDB</td>
<td>B + M → B</td>
<td>1 1 1 1 1</td>
</tr>
<tr>
<td><strong>Add Accumulators</strong></td>
<td>ABA</td>
<td>A + B → A</td>
<td>1 1 1 1 1</td>
</tr>
<tr>
<td><strong>Add Double</strong></td>
<td>ADDD</td>
<td>D + M:M+1 → D</td>
<td>* * 1 1 1</td>
</tr>
<tr>
<td><strong>Add with Carry</strong></td>
<td>ADCA</td>
<td>A + M + C → A</td>
<td>* * 1 1 1</td>
</tr>
<tr>
<td></td>
<td>ADCB</td>
<td>B + M + C → B</td>
<td>* * 1 1 1</td>
</tr>
<tr>
<td><strong>Complement, 2's</strong></td>
<td>NEG</td>
<td>00 - M → M</td>
<td>* * 1 1 1 1 1</td>
</tr>
<tr>
<td><strong>(Negate)</strong></td>
<td>NEGA</td>
<td>00 - A → A</td>
<td>* * 1 1 1 1 1 1</td>
</tr>
<tr>
<td></td>
<td>NEGB</td>
<td>00 - B → B</td>
<td>* * 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td><strong>Decimal Adjust, A</strong></td>
<td>DAA</td>
<td>Converts Binary Add. of BCD Characters into BCD Format (See Figure 4-22)</td>
<td></td>
</tr>
<tr>
<td><strong>Subtract</strong></td>
<td>SUBA</td>
<td>A - M → A</td>
<td>* * 1 1 1 1</td>
</tr>
<tr>
<td></td>
<td>SUBB</td>
<td>B - M → B</td>
<td>* * 1 1 1 1</td>
</tr>
<tr>
<td><strong>Subtract Accumulators</strong></td>
<td>SBA</td>
<td>A - B → A</td>
<td>* * 1 1 1 1</td>
</tr>
<tr>
<td><strong>Subtract Double</strong></td>
<td>SUBD</td>
<td>D - M:M+1 → D</td>
<td>* * 1 1 1 1</td>
</tr>
<tr>
<td><strong>Subtract with Carry</strong></td>
<td>SCBA</td>
<td>A - M - C → A</td>
<td>* * 1 1 1 1</td>
</tr>
<tr>
<td></td>
<td>SBCB</td>
<td>B - M - C → B</td>
<td>* * 1 1 1 1</td>
</tr>
<tr>
<td><strong>Multiply</strong></td>
<td>MUL</td>
<td>A*B → D</td>
<td>* * * * *</td>
</tr>
</tbody>
</table>

See Figure 4-2 for legend.

(Bit set if test is true and cleared otherwise)

1. (Bit V) Test: Result = 10000000?
2. (Bit C) Test: Result = 00000000?
3. (Bit C) Test: Decimal value of most significant BCD Character greater than nine?
   (Not cleared if previously set.)

**Figure 4-5. Arithmetic Instructions**

---

### Logic Instructions

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Mnemonic</th>
<th>Boolean/Arithmetic Operation</th>
<th>CCR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>And</strong></td>
<td>ANDA</td>
<td>A*M → A</td>
<td>* * 1 1 R</td>
</tr>
<tr>
<td></td>
<td>ANDB</td>
<td>B*M → B</td>
<td>* * 1 1 R</td>
</tr>
<tr>
<td><strong>Complement, 1's</strong></td>
<td>COM</td>
<td>M → M</td>
<td>* * 1 1 R S</td>
</tr>
<tr>
<td></td>
<td>COMA</td>
<td>A → A</td>
<td>* * 1 1 R S</td>
</tr>
<tr>
<td></td>
<td>COMB</td>
<td>B → B</td>
<td>* * 1 1 R S</td>
</tr>
<tr>
<td><strong>Exclusive OR</strong></td>
<td>EORA</td>
<td>A • M → A</td>
<td>* * 1 1 R</td>
</tr>
<tr>
<td></td>
<td>EORB</td>
<td>B • M → B</td>
<td>* * 1 1 R</td>
</tr>
<tr>
<td><strong>Or, Inclusive</strong></td>
<td>ORAA</td>
<td>A + M → A</td>
<td>* * 1 1 R</td>
</tr>
<tr>
<td></td>
<td>ORAB</td>
<td>B + M → B</td>
<td>* * 1 1 R</td>
</tr>
</tbody>
</table>

See Figure 4-2 for Legend

**Figure 4-6. Logic Instructions**
A value can be complemented without affecting the carry bit by using the EOR instruction. This is shown in the following example which illustrates complementing the value in the A accumulator two different ways.

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>COMA</td>
<td>EORA</td>
<td>#SFF</td>
<td>C-BIT NOT AFFECTED</td>
</tr>
</tbody>
</table>

4.3.2.3 DATA TEST INSTRUCTIONS. The data test instructions are shown in Figure 4-7. These instructions differ from logic instructions because they do not overwrite either memory or an accumulator with the result. Instead, their effect is to act only on the Condition Code Register.

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Mnemonic</th>
<th>Boolean/Arithmetic Operation</th>
<th>CCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit Test</td>
<td>BITA</td>
<td>A·M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BITB</td>
<td>A·M</td>
<td></td>
</tr>
<tr>
<td>Compare</td>
<td>CMPA</td>
<td>A - M</td>
<td></td>
</tr>
<tr>
<td>Compare Accumulators</td>
<td>CMPB</td>
<td>B - M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TST</td>
<td>M - 00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TSTA</td>
<td>A - 00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TSTB</td>
<td>B - 00</td>
<td></td>
</tr>
</tbody>
</table>

See Figure 4-2 for legend.

**Figure 4-7. Data Test Instructions**

Most of the data test instructions have an analogous logic instruction sequence which affects the CCR in a similar manner. The data test and corresponding logic instructions are shown below:

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>85 0F</td>
<td>BITA</td>
<td>#%1111</td>
<td>A·M</td>
<td></td>
</tr>
<tr>
<td>84 0F</td>
<td>ANDA</td>
<td>#%1111</td>
<td>A·M - A</td>
<td></td>
</tr>
<tr>
<td>81 0F</td>
<td>CMPA</td>
<td>#$F</td>
<td>A - M</td>
<td></td>
</tr>
<tr>
<td>80 0F</td>
<td>SUBA</td>
<td>#$F</td>
<td>A - M - A</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>CBA</td>
<td></td>
<td>A - B</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>SBA</td>
<td></td>
<td>A - B - A</td>
<td></td>
</tr>
<tr>
<td>4D</td>
<td>TSTA</td>
<td></td>
<td>A - 00</td>
<td></td>
</tr>
<tr>
<td>80 00</td>
<td>SUBA</td>
<td>#0</td>
<td>A - 00 - A</td>
<td></td>
</tr>
</tbody>
</table>

It should be noted that the TST instruction clears the C-bit which may be undesirable within loops involving multi-byte arithmetic. The LDAA or LDAB instruction, however, can also be used to test memory locations without clearing the C-bit. Within the data test group, note that the TST instruction is the only one which can be used to directly reference memory.
4.3.2.4 DATA HANDLING INSTRUCTIONS. The Data Handling instructions are summarized in Figure 4-8. The reader should note several effects on the CCR from this set of instructions. First, note that the CLRA instruction also clears the carry bit which could cause some concern within multi-byte arithmetic loops. Accumulator A can also be cleared, however, using LDAA #0 without affecting the C-bit.

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Mnemonic</th>
<th>Boolean/Arithmetic Operations</th>
<th>CCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear</td>
<td>CLR</td>
<td>00 → M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLRA</td>
<td>00 → A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLRb</td>
<td>00 → b</td>
<td></td>
</tr>
<tr>
<td>Decrement</td>
<td>DEC</td>
<td>M → M − 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DECA</td>
<td>A → A − 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DECB</td>
<td>B → B − 1</td>
<td></td>
</tr>
<tr>
<td>Increment</td>
<td>INC</td>
<td>M → M + 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INCA</td>
<td>A → A + 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INCb</td>
<td>B → B + 1</td>
<td></td>
</tr>
<tr>
<td>Load Accumulator</td>
<td>LDAA</td>
<td>M → A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDAB</td>
<td>M → B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDD</td>
<td>M:M + 1 → D</td>
<td></td>
</tr>
<tr>
<td>Push Data</td>
<td>PSHA</td>
<td>A → Msp, SP − 1 → SP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PSHB</td>
<td>B → Msp, SP − 1 → SP</td>
<td></td>
</tr>
<tr>
<td>Pull Data</td>
<td>PULA</td>
<td>SP + 1 → SP,Msp → A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PULB</td>
<td>SP + 1 → SP,Msp → B</td>
<td></td>
</tr>
<tr>
<td>Rotate Left</td>
<td>ROL</td>
<td>M → M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ROLA</td>
<td>A → C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ROLB</td>
<td>B → C</td>
<td></td>
</tr>
<tr>
<td>Rotate Right</td>
<td>ROR</td>
<td>M → M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RORA</td>
<td>A → C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RORB</td>
<td>B → C</td>
<td></td>
</tr>
<tr>
<td>Shift Left</td>
<td>ASL, LSL</td>
<td>M → M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASLA, LSLA</td>
<td>A → C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASLB, LSLB</td>
<td>B → C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASDL, LSLD</td>
<td>D → C</td>
<td></td>
</tr>
<tr>
<td>Shift Right, Arithmetic</td>
<td>ASR</td>
<td>M → M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASRA</td>
<td>A → b7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASRB</td>
<td>B → b7</td>
<td></td>
</tr>
<tr>
<td>Shift Right, Logical</td>
<td>LSR</td>
<td>M → M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LSRRA</td>
<td>A → b7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LSRB</td>
<td>B → b7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LSRD</td>
<td>D → b0</td>
<td></td>
</tr>
<tr>
<td>Store Accumulator</td>
<td>STAA</td>
<td>A → M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STAB</td>
<td>B → M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STD</td>
<td>D → M:M + 1</td>
<td></td>
</tr>
<tr>
<td>Transfer Accumulators</td>
<td>TAB</td>
<td>A → B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TBA</td>
<td>B → A</td>
<td></td>
</tr>
</tbody>
</table>

(3) (Bit V) Test: Operand = 10000000 prior to execution?
(5) (Bit V) Test: Operand = 01111111 prior to execution?
(6) (Bit V) Test: Set equal to result of N • C after shift has occurred.

See Figure 4-2 for legend.

Figure 4-8. Data Handling Instructions
It should also be noted that neither INC nor DEC affect the C-bit. It is intended that INC and DEC be used to control the loop counter variables within multi-byte arithmetic loops. It would, therefore, be inconsistent with this intention if these two instructions affected the carry bit. For this reason, INC and DEC have been included in the Data Handling instructions as opposed to the Arithmetic instructions.

Although they do not affect the carry bit, the INC and DEC instructions can be used to perform multi-byte arithmetic when used with a suitable branch instruction. This is shown in the following examples where the first four sequences illustrate incrementing the double accumulator D and the second four sequences illustrate decrementing it.

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>93 00</td>
<td></td>
<td>SUBD</td>
<td>$0</td>
<td>D - (-1) - D</td>
</tr>
<tr>
<td>C3 00 01</td>
<td></td>
<td>ADDD</td>
<td>#1</td>
<td>D + 1 - D</td>
</tr>
<tr>
<td>CB 01</td>
<td></td>
<td>ADDB</td>
<td>#1</td>
<td>B + 1 - B</td>
</tr>
<tr>
<td>89 00</td>
<td></td>
<td>ADCA</td>
<td>#0</td>
<td>A - 00 + C - A</td>
</tr>
<tr>
<td>5C</td>
<td></td>
<td>INCB</td>
<td></td>
<td>B + 1 - B</td>
</tr>
<tr>
<td>26 01</td>
<td></td>
<td>BNE</td>
<td>NXT</td>
<td>NO CARRY</td>
</tr>
<tr>
<td>4C</td>
<td></td>
<td>INCA</td>
<td></td>
<td>A + 1 - A</td>
</tr>
<tr>
<td>D3 00</td>
<td>NXT</td>
<td>ADDD</td>
<td>$0</td>
<td>D - (-1) - D</td>
</tr>
<tr>
<td>83 00 01</td>
<td></td>
<td>SUBD</td>
<td>#1</td>
<td>D - 1 - D</td>
</tr>
<tr>
<td>C0 01</td>
<td></td>
<td>SUBB</td>
<td>#1</td>
<td>B - 1 - B</td>
</tr>
<tr>
<td>82 00</td>
<td></td>
<td>SBCA</td>
<td>#0</td>
<td>A - 00 - C - A</td>
</tr>
<tr>
<td>5D</td>
<td></td>
<td>TSTB</td>
<td></td>
<td>B - 00</td>
</tr>
<tr>
<td>26 01</td>
<td></td>
<td>BNE</td>
<td>NXT2</td>
<td>NO BORROW</td>
</tr>
<tr>
<td>4A</td>
<td></td>
<td>DECA</td>
<td></td>
<td>A - 1 - A</td>
</tr>
<tr>
<td>5A</td>
<td>NXT2</td>
<td>DECB</td>
<td></td>
<td>B - 1 - B</td>
</tr>
</tbody>
</table>

In the preceding examples, the first and fifth instruction sequences (SUBD and ADDD) require only two bytes but it is probably not clear as to how they perform their respective operations. Addresses, $00 and $01, reference the two write-only Data Direction Registers for Port 1 and Port 2. When read, however, they always provide the result, $FF, which is actually the precharge of the internal Peripheral Data Bus (see Chapter 3)*. With respect to two’s complement numbers, the MC6801 double byte location $00:01 contains $01.

Finally, it should be mentioned that the Rotate instructions implement a 9-bit rather than an 8-bit rotate. The 9-bit operation facilitates rotation of multi-byte values. As an example, suppose the A accumulator and C-bit contained the values $AA and 1, respectively. After the RORA instruction is executed, the following situation would exist.

<table>
<thead>
<tr>
<th>C-Bit b7 b6 b5 b4 b3 b2 b1 b0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before RORA</td>
</tr>
<tr>
<td>1 1 0 1 0 1 0 1 0</td>
</tr>
<tr>
<td>After RORA</td>
</tr>
<tr>
<td>0 1 1 0 1 0 1 0 1</td>
</tr>
</tbody>
</table>

*The Port 3 DDR does not always provide $FF as a result; therefore, it should not be used for this purpose.
Eight bit rotation (instead of 9-bit) for a single accumulator can be achieved by using the appropriate three instruction sequence as shown in the following example. The first three instructions perform a left 8-bit rotation whereas the second sequence results in a right 8-bit rotation.

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td></td>
<td>ASLA</td>
<td></td>
<td>B0 = 0</td>
</tr>
<tr>
<td>24 01</td>
<td></td>
<td>BCC</td>
<td>* + 3</td>
<td>B0 OK</td>
</tr>
<tr>
<td>4C</td>
<td></td>
<td>INCA</td>
<td></td>
<td>SET B0</td>
</tr>
<tr>
<td>44</td>
<td></td>
<td>LSRA</td>
<td></td>
<td>B7 = 0</td>
</tr>
<tr>
<td>24 02</td>
<td></td>
<td>BCC</td>
<td>* + 4</td>
<td>B7 OK</td>
</tr>
<tr>
<td>8A 80</td>
<td></td>
<td>ORAA</td>
<td>#$80</td>
<td>SET B7</td>
</tr>
</tbody>
</table>

A 9-bit rotation is required when working with multi-byte values. The following sequence illustrates a left rotation of four bytes stored at locations BYT to BYT + 3 ($100-$103):

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>78 01 03</td>
<td></td>
<td>ASL</td>
<td>BYT + 3</td>
<td>INSURE B0 = 0</td>
</tr>
<tr>
<td>79 01 02</td>
<td></td>
<td>ROL</td>
<td>BYT + 2</td>
<td></td>
</tr>
<tr>
<td>79 01 01</td>
<td></td>
<td>ROL</td>
<td>BYT + 1</td>
<td></td>
</tr>
<tr>
<td>79 01 00</td>
<td></td>
<td>ROL</td>
<td>BYT</td>
<td></td>
</tr>
<tr>
<td>24 03</td>
<td></td>
<td>BCC</td>
<td>* + 5</td>
<td>IF C = 0, DONE</td>
</tr>
<tr>
<td>7C 01 03</td>
<td></td>
<td>INC</td>
<td>BYT + 3</td>
<td>ELSE SET B0</td>
</tr>
</tbody>
</table>

4.3.3 Program Control Instructions

Program control instructions can be subdivided into three categories: (1) Index Register, (2) Stack Pointer, and (3) Jump and Branch instructions.

4.3.3.1 INDEX REGISTER INSTRUCTIONS. Seven MPU instructions change the contents of the Index Register: LDX, INX, DEX, TSX, PULX, ABX, and RTI. Three other instructions involve the Index Register but do not change its value. The CPX instruction compares the Index Register with a double byte value and affects only the Condition Code Register. The PSHX instruction pushes the contents of the Index Register onto the stack (low byte first) but does not affect any bits in the CCR. Finally, the Index Register can be transferred to the Stack Pointer using the TXS instruction.

References to the Index Register in a statement can refer to its contents or to where it is “pointing.” When the Index Register appears in the operation field of a statement, it refers to the contents of the register itself. When the Index Register appears in the operand field, however, it refers to the location to which the Index Register is “pointing.” The Index Register instructions are summarized in Figure 4-9.

The effective address of an indexed instruction is determined by adding the unsigned offset byte to the current value of the Index Register. The result of the addition is transferred to an internal register which is not accessible to the programmer and the Index Register is not changed.
The ABX (Add B to X) instruction provides the capability to obtain a dynamic offset with indexing. After the offset is calculated using the B accumulator, the ABX instruction can be utilized to add the offset to the Index Register to form the effective address. The use of this offset is analogous to the offset byte in the indexed addressing mode: the ABX instruction performs an unsigned addition of the B accumulator with the Index Register. Note, however, that the ABX instruction overwrites the Index Register with the result of the addition whereas the Index Register remains unchanged in the indexed addressing mode.

4.3.3.2 STACK POINTER INSTRUCTIONS. The stack pointer (SP) manages a last-in-first-out (LIFO) queue or, equivalently, a "stack." Putting a byte onto the stack is known as a "push" while taking a byte from the stack is called a "pull." The last byte "pushed" will always be the first byte "pulled" regardless of its source or destination. In the MC6801, as in the MC6800, the stack builds in the direction of decreasing address and the Stack Pointer always points to the next "empty" location. Note that the Stack Pointer is decremented each time a byte is "pushed" onto the stack and incremented each time a byte is "pulled" from the stack.

The Stack Pointer can be transferred to the Index Register (TSX), and the Index Register can be transferred to the Stack Pointer (TXS). During the transfer of the Stack Pointer to the Index Register, the value is incremented by one which results in the Index Register pointing to the top of the stack or the last byte pushed. In a TXS instruction, the Index Register is assumed to be pointing to the top of stack. The value is decremented during transfer to the stack pointer and points to the next available location in the stack.

The functions of TSX and TXS are illustrated in the two sequences which are approximately functionally equivalent. The essential difference between them is their effect on the Condition Code Register: the PSH instructions do not affect the CCR whereas STAB and STAA do affect it.
The TXS instructions must be placed as shown in the above sequence to prevent stacked data from being overwritten by an interrupt. It is avoided in this procedure by "reserving" room on the stack before writing either accumulator to it.

The accumulators, Index Register, and Program Counter can be individually pushed or pulled. The entire machine state (except for the stack pointer itself) can be pushed onto the stack with the SWI and WAI instructions and interrupts. The entire machine state (again, except for the stack pointer itself) is pulled from the stack during the RTI instruction. The following instructions affect the stack pointer:

1. LDS, TXS, DES, INS
2. PSHA, PULA, PSHB, PULB, PSHX, PULX
3. JSR, BSR, RTS
4. SWI, WAI, RTI.

The first group of instructions directly manipulate the value of the stack pointer without affecting the contents of the stack and are summarized in Figure 4-10. The second group transfers the A and B accumulators and the Index Register to the stack. The third group pushes and pulls the Program Counter and, finally, the last group pushes and pulls the entire machine state (except for the stack pointer).

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Mnemonic</th>
<th>Boolean/Arithmetic Operation</th>
<th>CCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decrement Stack Pntr</td>
<td>DES</td>
<td>SP-1 → SP</td>
<td>1 1 0 0 1 0 1</td>
</tr>
<tr>
<td>Increment Stack Pntr</td>
<td>INS</td>
<td>SP+1 → SP</td>
<td>1 1 0 0 1 0 1</td>
</tr>
<tr>
<td>Load Stack Pntr</td>
<td>LDS</td>
<td>M:M+1 → SP</td>
<td>1 1 0 0 1 0 1</td>
</tr>
<tr>
<td>Store Stack Pntr</td>
<td>STS</td>
<td>SP → M:M+1</td>
<td>1 1 0 0 1 0 1</td>
</tr>
<tr>
<td>Index Reg ← Stack Pntr</td>
<td>TXS</td>
<td>X-1 → SP</td>
<td>1 1 0 0 1 0 1</td>
</tr>
<tr>
<td>Stack Pntr ← Index Reg</td>
<td>TSX</td>
<td>SP+1 → X</td>
<td>1 1 0 0 1 0 1</td>
</tr>
</tbody>
</table>

See Figure 4-2 for legend.

Figure 4-10. Stack Pointer Instructions

A few words of caution are appropriate concerning stack operations. None of the instructions in groups 2-4 of the example can be used (including hardware interrupts) until one of the instructions in group 1 has been used to initialize the stack pointer to a block of RAM memory of sufficient length to be overwritten without degradation to the program. It is the programmer's responsibility to ensure that the stack pointer is always pointing to a valid stack area whenever one of the instructions in groups 2-4 are used or whenever the possibility of interrupt exists.
Operation of the Stack Pointer with the PSH and PUL instructions is illustrated in Figures 4-11 and 4-12. The PSHA instruction causes the contents of Accumulator A to be stored in memory at the location indicated by the stack pointer. The stack pointer is automatically decremented following the store operation and is then "pointing" to the next empty location. The PULA instruction reverses the operation and causes the last byte stacked to be loaded into Accumulator A. The stack pointer is automatically incremented during the data transfer so that it points to the next available location. Note that the PULA instruction does not actually "remove" data from the stack; the value still remains in memory. The next value placed on the stack, however, will overwrite it.

4.3.3.3 JUMP AND BRANCH INSTRUCTIONS. The Jump and Branch instructions are shown in Figure 4-13. These instructions can be used to transfer program control to another point in the program. Execution of the Branch to Subroutine (BSR) and Jump to Subroutine (JSR) instructions cause the current Program Counter (i.e., the return address) to be pushed on the stack as shown in Figures 4-14 through 4-16. The Program Counter is then loaded with the effective address of the instruction which transfers control to the subroutine. The Return from Subroutine (RTS) instruction causes the return address to be pulled from the stack into the Program Counter as shown in Figure 4-17.

The BSR instruction uses only relative addressing and can be used to transfer control to -126 to +129 locations with respect to the first byte of the instruction. The JSR instruction, however, can use indexed, extended, or direct addressing and can be used to reach any location.

The No Operation instruction, NOP, while included here is a jump operation in a very limited sense. Its only effect is to increment the Program Counter by one. It is useful during the debug phase as a machine code replacement for a removed instruction and in equalizing the execution time through alternate paths in a program to establish synchronization.

The conditional branch instructions consist of eight pairs of complementary instructions. They are used to test the results of preceding operations and, on the basis of these results, either continue with the next instruction or transfer control to another point in the program as defined by the offset byte in the branch instruction. The eight pairs of instructions and the associated test conditions are shown in Figure 4-13.

The BRN (Branch Never) instruction can be used in a similar manner to the NOP instruction. It is also useful during debugging operations to nullify the machine code of another branch instruction while preserving the value of the offset byte. Finally, it completes the symmetry of the set of branch instructions by providing a complement to the BRA instruction. This feature is useful when implementing compilers for the MC6801.

In using conditional branch statements with arithmetic data, there could be some confusion as to which branch instructions should be used with a particular number representation. If signed two's complement values are being used, then the branch instructions involving the V-bit (BGT, BLT, BGE, BLE) should be used. If unsigned values are being used, then the branches which disregard the V-bit (BLS, BHS, BHI, BLO) should be used.
Figure 4-11. Operation of Push Instruction
Figure 4-12. Operation of Pull Instruction
Instructions | Mnemonic | Branch Test
--- | --- | ---
Branch Always | BRA | None
Branch If Carry Clear | BCC | C=0
Branch If Carry Set | BCS | C=1
Branch If = Zero | BEQ | Z=1
Branch If ≥ Zero | BGE | N·V=0
Branch If > Zero | BGT | Z+(N·V)=0
Branch If Higher | BHI | C+Z=0
Branch If ≤ Zero | BLE | Z+(N·V)=1
Branch If Lower or Same | BLS | C+Z=1
Branch If < Zero | BLT | N·V=1
Branch If Minus | BMI | N=1
Branch If Not Equal Zero | BNE | Z=0
Branch If Overflow Clear | BVC | V=0
Branch If Overflow Set | BVS | V=1
Branch If Plus | BPL | N=0
Branch Never | BRN | None
Branch If Higher or Same | BHS | C=0
Branch If Lower | BLO | C=1
Branch to Subroutine | BSR | 
Jump | JMP | 
Jump to Subroutine | JSR | Advances Program Counter Only
No Operation | NOP | 
Return From Interrupt | RTI | 
Return from Subroutine | RTS | 
Software Interrupt | SWI | 
Wait for Interrupt | WAI | 

See Figure 4-2 for legend.

1. (All) Load Condition Code Register from Stack. (See Special Operations)
2. (Bit 1) Set when interrupt occurs. If previously set, a Non-Maskable Interrupt is required to exit the wait state.

**Figure 4-13. Jump and Branch Instructions**

**Figure 4-14. Operation of JSR (Extended) Instruction**
Figure 4-15. Operation of JSR (Indexed) Instruction

Figure 4-16. Operation of BSR Instruction
Table 4-2 divides the branch instructions into three categories: simple, signed, and unsigned branches. The simple branches involve those based only on the value of a single bit in the CCR. The signed and unsigned branches refer to the appropriate branch with respect to the number system being used.

Table 4-2. Branch Instructions

Simple Conditional Branches:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEQ</td>
<td>BNE</td>
</tr>
<tr>
<td>BMI</td>
<td>BPL</td>
</tr>
<tr>
<td>BCS</td>
<td>BCC</td>
</tr>
<tr>
<td>BVS</td>
<td>BVC</td>
</tr>
</tbody>
</table>

Signed Conditional Branches:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGT</td>
<td>BLE</td>
</tr>
<tr>
<td>BGE</td>
<td>BLT</td>
</tr>
<tr>
<td>BEQ</td>
<td>BNE</td>
</tr>
<tr>
<td>BLE</td>
<td>BGT</td>
</tr>
<tr>
<td>BLT</td>
<td>BGE</td>
</tr>
</tbody>
</table>

Unsigned Conditional Branches:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHI</td>
<td>BLS</td>
</tr>
<tr>
<td>BHS</td>
<td>BLO</td>
</tr>
<tr>
<td>BEQ</td>
<td>BNE</td>
</tr>
<tr>
<td>BLS</td>
<td>BHI</td>
</tr>
<tr>
<td>BLO</td>
<td>BHS</td>
</tr>
</tbody>
</table>
Note that a branch instruction does not have to immediately follow a particular operation provided subsequent instructions do not affect the bits to be used in the branch test. In the following example, the branch will be taken or not taken based solely on the effect of the LDAA instruction on the Z-bit. None of the instructions between LDAA and BEQ affect any bit in the CCR.

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>96 20</td>
<td></td>
<td>LDAA</td>
<td>BYTE</td>
<td>GET BYTE</td>
</tr>
<tr>
<td>33</td>
<td></td>
<td>PULB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>TSX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3A</td>
<td></td>
<td>ABX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27 04</td>
<td></td>
<td>BEQ</td>
<td>* + 2</td>
<td></td>
</tr>
</tbody>
</table>

The Compare Index Register (CPX) instruction can also be used effectively with the conditional branch instructions. Readers familiar with the M6800 will recall that this instruction can be used with only a few of the branches. In the MC6801, however, internal processing has been modified such that it can be used for branching similar to the single byte comparisons. A typical use of the CPX instruction is shown in the following example.

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE 80 00</td>
<td></td>
<td>LDX</td>
<td>#$8000</td>
<td></td>
</tr>
<tr>
<td>8C 7F FF</td>
<td></td>
<td>CPX</td>
<td>#$7FFF</td>
<td></td>
</tr>
<tr>
<td>22 01</td>
<td></td>
<td>BHI</td>
<td>HANG</td>
<td>WILL TAKE BRANCH</td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 FE</td>
<td>HANG</td>
<td>BRA</td>
<td>*</td>
<td>FOR NOW</td>
</tr>
</tbody>
</table>

4.4 PROGRAMMING EXAMPLES

In this section, several programming examples are presented which focus on use of the Index Register and multi-byte arithmetic.

4.4.1 Use of the Index Register

The indexed addressing mode provides the capability to construct powerful routines which require a minimum amount of memory. This is most evident when writing routines which involve an interactive algorithm.

The effective address of an indexed instruction is formed by adding the 8-bit unsigned offset of the instruction to the Index Register. The result of this addition is placed in an internal unaddressable temporary register; the value in the Index Register is not changed.

Because the offset is part of the instruction, it should be considered a static value determined at assembly time. While a self-modifying instruction sequence could be used to provide a dynamic offset, it should be remembered that such a capability cannot be realized if the instruction resides in Read-Only-Memory (ROM). In addition, self-modifying code is generally considered poor programming practice.

The operand field in a statement which uses indexed addressing can take any of five possible forms:

- X
- X 0
- SYMBOL
- EXPRESSION
- X

The first three forms are equivalent and will produce a zero offset byte. Note that the "#" sign is never used in the operand field of a statement which uses the indexed addressing mode. The "#"
sign can be used, however, to load the Index Register with the address of a label. The addressing
mode for this type of instruction, however, is immediate rather than indexed. For example, the in-
struction

\[
\text{LDX} \quad \#\text{TABLE}
\]

will result in the Index Register being loaded with the address of the label, TABLE. Another impor-
tant instruction is one like the following:

\[
\text{LDX} \quad X
\]

This instruction causes a double byte value, located at where the Index Register is pointing, to over-
write the current value of the Index Register. This type of instruction enables a programmer to use
"indirection." To illustrate this technique, suppose that it is desired to load a byte into Ac-
cumulator A from an address pointed to by a double byte value stored at the location ADDR. The
location of ADDR, however, is also a variable which can be located by a double byte address stored
at the label, POINTR. The byte pointed to by ADDR can be loaded into Accumulator A using the
following sequence.

```
Machine Code Label Operation Operand Comments
FE 10 00 LDX POINTR GET ADDR POINTR
EE 00 LDX X X HAS ADDR
A6 00 LDAA X ACCA HAS BYTE
```

A typical application of indirection is subroutine linkage where the calling routine provides the
subroutine an address in the Index Register which points to a table of addresses. The addresses in the
table point to the values of calling parameters for the subroutine. Several routines in this chapter use
this method of subroutine linkage.

When used for manipulating a table of values, the indexed addressing mode can be used directly in
one of two modes:

1. the Index Register can be loaded with the starting address of the table and the offset byte can
   be contained in the instruction, or
2. the offset byte in the instruction can be used to indicate the starting address of the table and the
   Index Register used to contain the offset.

It is important to realize that both of the above methods have restrictions involving their use. In the
first method, the offset byte must generally be considered a static value determined at assembly
time. In addition, the offset byte can only reach 256 locations in the table.

In the second method, the starting address of the table must reside in the direct area ($00-FF). If this
restriction can be met, however, all values in the table can be accessed. These two methods are il-
lustrated in the following examples where TABLE starts at $00F0.

```
Machine Code Label Operation Operand Comments
CE 00 F0 LDX \#TABLE ADDR (TABLE)-X
A6 04 LDAA 4, X GET 5TH BYTE
CE 00 04 LDX \#4 OFF-X
A6 F0 LDAA TABLE, X GET 5TH BYTE
```
Two other methods can also be used to manipulate a table of values which are not as direct but do alleviate some of the restrictions associated with the aforementioned methods. First, a dynamic offset can be obtained by using Accumulator B to compute the offset and then using the ABX (Add ACCB to X) instruction to obtain the address of the element. The considerations in using this method include:

1. the ABX instruction involves an unsigned single byte addition which limits indexing to forward references not greater than 255 bytes, and
2. the ABX instruction overwrites the Index Register with the result of the addition and requires the Index Register to be reinitialized if used within a loop.

Finally, the most general method for accessing a table is to compute the absolute address of the location using a signed double byte offset. This approach requires storage of the Index Register, a double byte signed addition, and replacement of the former Index Register value with the new one. It requires more memory and execution time but overcomes most of the limitations associated with the aforementioned methods. The following example illustrates the use of both methods.

Both the JSR and JMP instructions provide an indexed addressing mode. Computation of the effective address for these instructions is identical to that performed for other instructions. The final destination of the results for these two instructions is different than for the other instructions: it is transferred to the Program Counter. This operation can be described by the following expression:

\[ \text{PC} + \text{OFFSET} \]

These instructions are useful to jump to an address specified in a certain memory location as illustrated in the following example. In this example, if the value in the A accumulator is equal to an ASCII "A," a jump is made to location, $1040. If the two values are not equal, the program remains in a loop on itself.

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE 00 F0</td>
<td></td>
<td>LDX</td>
<td>#TABLE</td>
<td>ADDR (TABLE)-X</td>
</tr>
<tr>
<td>C6 04</td>
<td></td>
<td>LDAB</td>
<td>#4</td>
<td>OFF-ACCB</td>
</tr>
<tr>
<td>3A</td>
<td></td>
<td>ABX</td>
<td></td>
<td>GET ADDR</td>
</tr>
<tr>
<td>A6 00</td>
<td></td>
<td>LDAA</td>
<td>X</td>
<td>GET 5TH BYTE</td>
</tr>
<tr>
<td>CE 00 F0</td>
<td></td>
<td>LDX</td>
<td>#TABLE</td>
<td></td>
</tr>
<tr>
<td>CC 00 04</td>
<td></td>
<td>LDD</td>
<td>#4</td>
<td>OFF-A:B</td>
</tr>
<tr>
<td>3C</td>
<td></td>
<td>PSHX</td>
<td></td>
<td>X-STACK</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>TSX</td>
<td></td>
<td>SP+1-X</td>
</tr>
<tr>
<td>E3 00</td>
<td></td>
<td>ADDD</td>
<td>X</td>
<td>GET ADDR</td>
</tr>
<tr>
<td>ED 00</td>
<td></td>
<td>STD</td>
<td>X</td>
<td>REPLACE</td>
</tr>
<tr>
<td>38</td>
<td></td>
<td>PULX</td>
<td>ADDR-X</td>
<td></td>
</tr>
<tr>
<td>A6 00</td>
<td></td>
<td>LDAA</td>
<td>X</td>
<td>GET 5TH BYTE</td>
</tr>
</tbody>
</table>

These instructions are useful to jump to an address specified in a certain memory location as illustrated in the following example. In this example, if the value in the A accumulator is equal to an ASCII "A," a jump is made to location, $1040. If the two values are not equal, the program remains in a loop on itself.

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE 10 00</td>
<td></td>
<td>LDX</td>
<td>#TABLE</td>
<td>ADDR (TABLE)-X</td>
</tr>
<tr>
<td>A6 00</td>
<td></td>
<td>LDAA</td>
<td>X</td>
<td>GET CHAR</td>
</tr>
<tr>
<td>B1 41</td>
<td></td>
<td>CMPA</td>
<td>#A</td>
<td>CHECK FOR AN &quot;A&quot;</td>
</tr>
<tr>
<td>26 FE</td>
<td></td>
<td>BNE</td>
<td>*</td>
<td>IF NE., HANG</td>
</tr>
<tr>
<td>EE 01</td>
<td></td>
<td>LDX</td>
<td>1, X</td>
<td>ELSE GET ADDR</td>
</tr>
<tr>
<td>6E 00</td>
<td></td>
<td>JMP</td>
<td>X</td>
<td>GO TO IT!</td>
</tr>
<tr>
<td>41</td>
<td>TABLE</td>
<td>FCC</td>
<td>/A/</td>
<td>CHAR &quot;A&quot;</td>
</tr>
<tr>
<td>10 40</td>
<td></td>
<td>FDB</td>
<td>ADDR</td>
<td>ADDR AT $1040</td>
</tr>
</tbody>
</table>

Three routines will now be presented to further illustrate possible uses of the Index Register. The first one is shown in Figure 4-18 and illustrates a BCD addition utilizing the indexed addressing mode in a somewhat unique fashion. However, the limitations of the routine are quite severe. For
BCD1

***BCD1 ADD***

CALLING CONVENTION:

LDX  #P
JSR  BCD1

WHERE P IS THE FIRST BYTE OF AN AREA

DEFINED AS

P  RMB  NB  FIRST ADDEND
Q  RMB  NB  SECOND ADDEND
RES RMB NB  RESULT

AND VARIABLE NB IS THE LENGTH IN_BYTES OF

BOTH ADDENDS AND THE RESULT.

RESTRICTIONS AND NOTES:

1. THE ROUTINE IS RE-ENTRANT PROVIDING EACH
CALLER PROVIDES A UNIQUE DATA AREA
2. THE ROUTINE MUST BE ASSEMBLED FOR A FIXED
VALUE OF NB
3. THE STORAGE LOCATIONS OF P, Q, AND RES
MUST BE CONTIGUOUS
4. THE VARIABLES P, Q, AND RES MAY RESIDE
ANYWHERE IN THE MEMORY MAP.
5. THE CARRY BIT MAY BE TESTED UPON
EXIT. IF C=0, THERE WAS NO CARRY
6. FROM THE LAST ADDITION; ELSE THE
SUM OVERFLOWED NB BYTES.
7. THE VALUE (3*NB-1) MUST BE LESS THAN
256. THEREFORE, NB MUST NOT EXCEED 85.

A NB EQU 4 SPECIFIES 4-BYTE VALUES

ORG $1000

LDAB  #NB B HAS NUMBER OF BYTES
CLC START WITH IT CLEARED
LDAA NB-1,X GET NEXT ADDEND
ADCA 2*NB-1,X ADD TO NEXT ONE
DAA ADJUST FOR BCD
STA 3*NB-1,X STORE NEXT BYTE OF RESULT
DEX NEXT BYTE
DECB MAYBE DONE
BNE NEXT NO, NOT YET
RTS
END

TOTAL ERRORS 00000--00000

Figure 4-18. BCD Addition: BCD1
instance, it must be assembled for only one length of BCD value wherein the symbol NB (Number of Bytes) is defined. Another limitation is that it assumes a specific data storage arrangement of the addends and the result: they must be contiguous and in the order assumed by the routine. The execution time of the routine is approximately 110 cycles for 4-byte (NB = 4) values.

The next two examples involve the use of the ABX instruction in two table handling routines. The first routine, shown in Figure 4-19, illustrates a simple block move. The routine has the limitation, however, that it always moves the last byte first. The implication of this limitation is that the routine can not be used to move the table to an area of lesser address ("up" in memory) unless the following restriction is observed:

TO address + length < FROM address

If this restriction is not met, part of the table will be overwritten during the move. The routine is made more general in Figure 4-22 where an inspection is made as to whether the table should be moved from the "top" or "bottom" thus alleviating the restriction of the first example.

Relative execution time of the two routines is consistent with their flexibility. To copy 16 bytes, the BLOCKC routine (Figure 4-19) executes in approximately 800 cycles whereas the BLOCKM routine (Figure 4-20) requires approximately 1100 cycles.

4.4.2 Number Systems

The ALU (Arithmetic and Logic Unit) always performs standard binary addition of either one or two bytes using two's complement arithmetic. The MPU instruction set and hardware flags support direct arithmetic operations using any of the following three different representations for numbers:

1. Each byte can be interpreted as a signed 2's complement number in the range −128 to +127:

<table>
<thead>
<tr>
<th>b7</th>
<th>b6</th>
<th>b5</th>
<th>b4</th>
<th>b3</th>
<th>b2</th>
<th>b1</th>
<th>b0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

2. Each byte can be interpreted as an unsigned binary number in the range 0 to 255:

<table>
<thead>
<tr>
<th>b7</th>
<th>b6</th>
<th>b5</th>
<th>b4</th>
<th>b3</th>
<th>b2</th>
<th>b1</th>
<th>b0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

3. Each byte can contain two 4-bit binary coded decimal (BCD) numbers. With this interpretation, each byte represents numbers in the range 0-99:

<table>
<thead>
<tr>
<th>b7</th>
<th>b6</th>
<th>b5</th>
<th>b4</th>
<th>b3</th>
<th>b2</th>
<th>b1</th>
<th>b0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
**Figure 4-19. Block Move Routine: BLOCKC**

```
PAGE 001 BLOCKC *SA:1 BLOCKC *** RE-ENTRANT BLOCK COPY ***

00001  NAM  BLOCKC
00002  OPT  Z01,LLEN=80
00003  TTL  *** RE-ENTRANT BLOCK COPY ***

00005  *******************************************************
00006  *
00007  * B L O C K C -- A RE-ENTRANT 256-BYTE BLOCK COPY *
00008  *
00009  * THIS ROUTINE WILL COPY A BLOCK OF *
00010  * MEMORY UP TO 256 BYTES IN LENGTH *
00011  * FROM ONE LOCATION TO ANOTHER. *
00012  *
00013  * CALLING CONVENTION: *
00014  *
00015  *  LDX #PACKET *
00016  *  JSR  BLOCKC *
00017  *
00018  * WHERE PACKET IS DEFINED AS: *
00019  *
00020  *  PACKET FCB L LENGTH *
00021  *  FDB FROM -FROM- ADDRESS *
00022  *  FDB TO -TO- ADDRESS *
00023  * NOTES: *
00024  *
00025  *  A ZERO LENGTH WILL RESULT IN COPYING 256 BYTES. *
00026  *  IF TO < FROM THEN TO + L MUST BE < FROM BECAUSE *
00027  *  THE ROUTINE MOVES THE LAST LOCATION FIRST. *
00028  *  THE ROUTINE IS RE-ENTRANT. *
00029  *
00030  *******************************************************

00031  0003A 1000  ORG $1000
00032  00033A 1000 E6 00  A BLOCKC LDAB X GET THE LENGTH
00033  00034A 1002 5A  DEC B ONE LESS FOR AN OFFSET
00034  00035A 1003 3C  PSHX SAVE PACKET POINTER

00035  00037A 1004 38  BLK000 PULX RESTORE PACKET POINTER
00036  00038A 1005 3C  PSHX PUT IT BACK
00037  00039A 1006 EE 01 A LDX 1,X GET -FROM- ADDR
00038  00040A 1008 3A  ABX ADD THE OFFSET
00039  00041A 1009 A6 00 A LDAAX X GOT THE -FROM- BYTE

00040  00043A 100B 38  PULX RESTORE PACKET POINTER
00041  00044A 100C 3C  PSHX
00042  00045A 100D EE 03 A LDX 3,X GET -TO- ADDR
00043  00046A 100F 3A  ABX ADD THE OFFSET
00044  00047A 1010 A7 00 A STAA X STASH -FROM- AT -TO-
00045  00048A 1012 5A  DECB DEC THE OFFSET
00046  00049A 1013 CF FF A CMPB #$FF ZERO IS THE LAST OFFSET
00047  00050A 1015 26 ED 1004 BNE BLK000 DONE YET?

00052  * ALL DONE, CLEAN UP AND LEAVE

00053  00054A 1017 38  PULX GET POINTER OFF STACK
00054  00055A 1018 39  RTS
00055  00056  END

TOTAL ERRORS 00000--00000

4-30
*** RE-ENTRANT BLOCK MOVE ***

**CALLING CONVENTION:**

LDX #PACKET
JSR BLOCKM

WHERE PACKET IS DEFINED AS:

PACKET FCB L LENGTH
FDB FDB FROM -FROM- ADDR
FDB TO -TO- ADDRESS

NOTES:

A ZERO LENGTH WILL RESULT IN COPYING 256 BYTES.

THIS ROUTINE MAY BE USED TO MOVE A BLOCK OF

MEMORY IN EITHER DIRECTION WITHOUT DESTROYING

ANY OF THE BLOCK DURING THE MOVE.

THE ROUTINE IS RE-ENTRANT.

*****

**ORG $1000**

LDD 3,X WHICH WAY TO MOVE, SET C-BIT

SUBD 1,X

LDAB X GET LENGTH

PSHB PUT LENGTH INTO STACK

PSHX SAVE PACKET POINTER

DECL MAKE A GUESS

BCC BLK001 MOVE UP IN MEMORY

PULX RESTORE PACKET POINTER

BLK001 PUT IT BACK

LDX 1,X GET -FROM- ADDR

AXX ADD THE OFFSET

LDAA X GET THE -FROM- BYTE

PULX RESTORE PACKET POINTER

LDX 3,X GET -TO- ADDR

AXX STASH -FROM- AT -TO-

INCB GUESS

BCS BLK002 IF FROM > TO ADDR

**Figure 4-20. Block Move Routine: BLOCKM**
4.4.3 Two’s Complement Overflow

The MPU sets or clears selected flags in the Condition Code Register during instruction execution as shown in Figure 4-2. The interpretation of most of these bits is rather straightforward. The Overflow flag (V-bit), however, is an exception and requires clarification. Two’s complement overflow occurs when an operation yields a result beyond the −128 to +127 range of a single signed two’s complement byte. By examining the sign bits of the operands and the result, it is possible to determine whether an arithmetic overflow has occurred. In Table 4-3, for example, the operands and result are shown for the ABA (Add B to A) instruction. Note that the value for the V-bit at the conclusion of the addition is determined by the sign bits (bit 7) of the two operands and the result.

Table 4-3. Overflow Rules for Addition

<table>
<thead>
<tr>
<th>A7</th>
<th>B7</th>
<th>R7</th>
<th>V-Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(A + B) = R
If the signs of the addends, A7 and B7, are different, no overflow can occur and the V-bit is clear after the operation. If the addend sign bits are alike and the result exceeds the byte capacity, the sign bit of the result (R7) will change and the overflow bit will be set.

The following three sequences illustrate the results of the ALU and V-bit from several additions using the ABA instruction. The first two additions result in valid two’s complement sums and, therefore, the V-bit remains clear. The third addition, however, attempts to add two negative values which should have produced the result, \(-132\). This value exceeds eight bits and the V-bit is set to indicate this condition.

$$
\begin{array}{cccccccc}
  & b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\
V\text{-Bit} & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
& 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
& 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
$$

$$
R = A + B = -67
$$

$$
\begin{array}{cccccccc}
  & b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\
V\text{-Bit} & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
& 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
& 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
\end{array}
$$

$$
R = A + B = -100
$$

$$
\begin{array}{cccccccc}
  & b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\
V\text{-Bit} & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
& 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
& 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
$$

$$
R = A + B = +124
$$

In subtraction instructions, the possibility of overflow exists whenever the operands differ in sign. Overflow conditions for A \(- B\) are illustrated in Table 4-4.

Note that Table 4-4 is identical to the addition case (Table 4-3) except that B7 has been replaced by the complement of B7, \(\overline{B7}\). The MC6801 ALU performs subtraction by adding the two’s complement of the subtrahend B to the minuend A. The subtraction table with B7 complemented then reflects the sign bits of two numbers that are to be added. If A7 and \(\overline{B7}\) are alike, overflow has occurred if the sign of the result does not match the sign of both A7 and B7.

The MC6801 also has the capability of adding and subtracting double byte (16-bit) two’s complement values. The range of the double accumulator A:B (or D) is \(-32,768\) to \(32,767\). The discussion for the V-bit is equally valid for the double byte case if all references to bit 7 are replaced with bit 15.

**Table 4-4. Overflow Rules for Subtraction**

<table>
<thead>
<tr>
<th>A7</th>
<th>B7</th>
<th>R7</th>
<th>V-Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\((A - B) = R\)
The V-bit is set or cleared during all shift and rotate instructions according to the result of an exclusive OR of the N-bit (new sign) and the C-bit after the operation has been completed. Within the context of signed arithmetic, the shift instructions can be used to multiply or divide an integer by a power of two. The V-bit can be used to detect two's complement overflow after a left shift which is equivalent to a multiply by two. In this case, the C-bit contains the former sign and the V-bit formula functions correctly.

Right logical shifts, however, do not set the V-bit properly with the exclusive OR because the C-bit contains the former value of bit 0 instead of bit 7. This represents no real difficulty, however, because the programmer is always expected to utilize the right arithmetic shift (ASR) for signed values which guarantee no two's complement overflow.

### 4.4.4 Arithmetic Instructions Revisited

Figure 4-21 summarizes the instructions used primarily for arithmetic operations. The effect of each operation on memory and the MPU accumulators is shown along with how the result of each operation affects the Condition Code Register.

The carry bit (C-bit) is used as both a carry for addition and a borrow for subtraction. It is added to the accumulators in the Add with Carry Instructions (ADCA, ADCB) and subtracted from the accumulators in the Subtract with Carry instructions (SBCA, SBCB).

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Mnemonic</th>
<th>Boolean/Arithmetic Operation</th>
<th>CCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>ADDA</td>
<td>A + M → A</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>ADDB</td>
<td>B + M → B</td>
<td>1</td>
</tr>
<tr>
<td>Add Accumulators</td>
<td>ABA</td>
<td>A + B → A</td>
<td>1</td>
</tr>
<tr>
<td>Add Double</td>
<td>ADDD</td>
<td>D + M:M + 1 → D</td>
<td></td>
</tr>
<tr>
<td>Add with Carry</td>
<td>ADCA</td>
<td>A + M + C → A</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>ADCB</td>
<td>B + M + C → B</td>
<td>1</td>
</tr>
<tr>
<td>Complement, 2's</td>
<td>NEG</td>
<td>00 - M → M</td>
<td>1</td>
</tr>
<tr>
<td>(Negate)</td>
<td>NEGA</td>
<td>00 - A → A</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>NEGB</td>
<td>00 - B → B</td>
<td>1</td>
</tr>
<tr>
<td>Decimal Adjust, A</td>
<td>DAA</td>
<td>Converts Binary Add. of BCD</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Characters into BCD Format</td>
<td>1</td>
</tr>
<tr>
<td>Subtract</td>
<td>SUBA</td>
<td>A - M → A</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SUBB</td>
<td>B - M → B</td>
<td>1</td>
</tr>
<tr>
<td>Subtract Accumulators</td>
<td>SBA</td>
<td>A - B → A</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SUBD</td>
<td>D - M:M + 1 → D</td>
<td>1</td>
</tr>
<tr>
<td>Subtract Double</td>
<td>SBCA</td>
<td>A - M → C + A</td>
<td>1</td>
</tr>
<tr>
<td>Subtract with Carry</td>
<td>SBCB</td>
<td>B - M → C + B</td>
<td>1</td>
</tr>
<tr>
<td>Multiply</td>
<td>MUL</td>
<td>A*B → D</td>
<td>1</td>
</tr>
</tbody>
</table>

See Figure 4-2 for legend.

- (Bit set if test is true and cleared otherwise)
- (Bit V) Test: Result = 10000000?
- (Bit C) Test: Result = 00000000?
- (Bit C) Test: Decimal value of most significant BCD Character greater than nine?
  (Not cleared if previously set.)

**Figure 4-21. Arithmetic Instructions (Repeated)**

4-34
The Decimal Adjust instruction, DAA, can be used to adjust the binary result of a BCD addition in the A accumulator. Following the three operations, ABA, ADD, and ADC, the DAA instruction adjusts the contents of the accumulator and the C-bit to represent the correct BCD result. Because the DAA instruction properly affects the C-bit, it can be used for multi-byte BCD addition. Figure 4-22 shows the details of the DAA instruction and its relationship to other Condition Code Register bits.

Operation: Adds hexadecimal numbers 00, 06, 60, or 66 to ACCA, and may also set the carry bit, as indicated in the following table:

| State of | Upper | Initial | Lower | Number Added | State of |
| C-Bit | Half-Byte | H-Bit | Half-Byte | to ACCA | C-Bit |
| Before DAA | (Bits 4-7) | (Col. 3) | (Bits 0-3) | By DAA | After DAA |
| (Col. 1) | (Col. 2) | (Col. 4) | (Col. 5) | (Col. 6) |
| 0 | 0-9 | 0 | 0-9 | 00 | 0 |
| 0 | 0-8 | 0 | A-F | 06 | 0 |
| 0 | 0-9 | 1 | 0-3 | 06 | 0 |
| 0 | A-F | 0 | 0-9 | 60 | 1 |
| 0 | 9-F | 0 | A-F | 66 | 1 |
| 0 | A-F | 1 | 0-3 | 66 | 1 |
| 1 | 0-2 | 0 | 0-9 | 60 | 1 |
| 1 | 0-2 | 0 | A-F | 66 | 1 |
| 1 | 0-3 | 1 | 0-3 | 66 | 1 |

Note: Columns (1) to (4) of the above table represent all possible cases which can result from any of the operations ABA, ADD, or ADC, with initial carry either set or clear, applied to two binary-coded-decimal operands. The table shows hexadecimal values.

Effect on Condition Code Register:
- H Not affected.
- I Not affected.
- N Set if most significant bit of the result is set; cleared otherwise.
- Z Set if all bits of the result are cleared; cleared otherwise.
- V Not defined.
- C Set or reset according to the same rule as if the DAA and an immediately preceding ABA, ADD, or ADC were replaced by a hypothetical binary-coded-decimal addition.

**Figure 4-22. Operation of DAA Instruction**

### 4.4.4.1 USE OF ARITHMETIC INSTRUCTIONS. Typical use of the arithmetic instructions is shown in the following examples:

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>86 AA</td>
<td></td>
<td>LDAA</td>
<td>#$AA</td>
<td>1010 1010</td>
</tr>
<tr>
<td>C6 CC</td>
<td></td>
<td>LDAB</td>
<td>#$CC</td>
<td>1100 1100</td>
</tr>
<tr>
<td>82 04</td>
<td></td>
<td>ADCA</td>
<td>#4</td>
<td>0111 0110 C CLR</td>
</tr>
</tbody>
</table>

The V-bit and C-bit for the ADCA instruction are set according to Table 4-5.

The SUBA instruction subtracts a single byte value defined by the operand from Accumulator A:

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>86 65</td>
<td></td>
<td>LDAA</td>
<td>#$65</td>
<td>0110 0101</td>
</tr>
<tr>
<td>80 67</td>
<td></td>
<td>SUBA</td>
<td>#$67</td>
<td>1111 1110 C SET</td>
</tr>
<tr>
<td>82 04</td>
<td></td>
<td>SBCE</td>
<td>#4</td>
<td>1111 1001 C CLR</td>
</tr>
</tbody>
</table>
### Table 4-5. Truth Table for “Add with Carry”

<table>
<thead>
<tr>
<th>B7 ACC After</th>
<th>B7 ACC Before</th>
<th>B7 Operand (or ACCB) Before</th>
<th>V-Bit After</th>
<th>C-Bit After</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The two’s complement overflow and carry bits are set as shown in Table 4-6 as a result of a subtract with borrow.

### Table 4-6. Truth Table for “Subtract with Borrow”

<table>
<thead>
<tr>
<th>B7 ACC After</th>
<th>B7 ACC Before</th>
<th>B7 Operand (or ACCB) Before</th>
<th>V-Bit After</th>
<th>C-Bit After</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The NEG instruction can be used to obtain the two’s complement of a single byte. For example, if the A accumulator contains $03$, after the NEGA instruction it would contain $SFD$. Upon completion of the NEG instruction, the carry bit is cleared only if the result is zero. This characteristic can be used to negate the double byte accumulator A:B as shown in the following examples.

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td></td>
<td>NEGA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>NEGB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>82 00</td>
<td></td>
<td>SBCA</td>
<td>#$0</td>
<td>0000 - A:B - A:B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td></td>
<td>COMA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>NEGB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>82 FF</td>
<td></td>
<td>SBCA</td>
<td>#$FF</td>
<td></td>
</tr>
</tbody>
</table>

### 4.4.5 Multi-Byte Addition and Subtraction

Typically microcomputer-based systems require arithmetic instructions to be combined into routines which operate on values larger than one byte. Double byte arithmetic is facilitated by use of the ADDD and SUBD instructions which perform addition and subtraction using the A:B double accumulator. Double byte and multi-byte arithmetic can also be performed using single byte arithmetic instructions. Several algorithms will now be presented showing their implementation using the MC6801 instruction set.
The basic arithmetic operations are binary addition and subtraction as shown in the following example. The two sequences illustrate performing the following two single byte operations:

\[
\text{ALPHA} + \text{BETA} \rightarrow \text{OMEGA} \\
\text{OMEGA} - \text{BETA} \rightarrow \text{ALPHA}
\]

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>B6 01 00</td>
<td></td>
<td>LDAA</td>
<td>ALPHA</td>
<td>ALPHA + BETA</td>
</tr>
<tr>
<td>BB 02 00</td>
<td></td>
<td>ADDA</td>
<td>BETA</td>
<td></td>
</tr>
<tr>
<td>B7 03 00</td>
<td></td>
<td>STAA</td>
<td>OMEGA</td>
<td></td>
</tr>
<tr>
<td>B6 03 00</td>
<td></td>
<td>LDAA</td>
<td>OMEGA</td>
<td>OMEGA - BETA</td>
</tr>
<tr>
<td>B0 02 00</td>
<td></td>
<td>SUBA</td>
<td>BETA</td>
<td></td>
</tr>
<tr>
<td>B7 01 00</td>
<td></td>
<td>STAA</td>
<td>ALPHA</td>
<td></td>
</tr>
</tbody>
</table>

These sequences are so short that they are usually performed in line. Addition of packed BCD single bytes requires only one more instruction. The DAA instruction is used immediately after ADD, ADC or ABA instructions to adjust the binary value generated in Accumulator A to the correct BCD value:

\[
\begin{align*}
\text{Machine Code} & \quad \text{Label} \quad \text{Operation} \quad \text{Operand} \quad \text{Comments} \\
86 & \quad 09 \quad \text{LDAA} \quad \#59 \quad 09 + 06 = 15 \\
80 & \quad 06 \quad \text{ADDA} \quad \#56 \quad \text{ACCA} = 0F \\
19 & \quad & \text{DAA} \quad \text{ACCA} = 15 \\
80 & \quad 06 \quad \text{ADDA} \quad \#56 \quad \text{ACCA} = 1B \\
19 & \quad & \text{DAA} \quad 15 + 06 = 21
\end{align*}
\]

Subtraction of packed BCD values requires the use of a conversion technique. Because there is no "Decimal Adjust" for subtraction of BCD values, it is necessary to convert them to 9's complement form and then add them. The following sequence illustrates subtraction of two packed BCD single byte numbers:

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>86 99</td>
<td></td>
<td>LDAA</td>
<td>#599</td>
<td>FIND: 21 - 16</td>
</tr>
<tr>
<td>80 16</td>
<td></td>
<td>SUBA</td>
<td>#516</td>
<td>99 - 16 = 83</td>
</tr>
<tr>
<td>0D</td>
<td></td>
<td>SEC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>89 21</td>
<td></td>
<td>ADCA</td>
<td>#521</td>
<td>82 + 21 + 1 = A5</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>DAA</td>
<td></td>
<td>ANS: 05 = ACCA</td>
</tr>
</tbody>
</table>

The computational technique implemented with these instructions is illustrated by the following sequence:

\[
\text{ANS} = 21 - 16 \\
= (99 - 16) + 21 - 99 \\
= (99 - 16 + 1) + 21 - 100 \\
= 5
\]

One is added to the 9's complement of the subtrahend by setting the carry bit which is subsequently added to the minuend during execution of the ADCA instruction. The DAA instruction adjusts the result to the proper BCD value by effectively subtracting 100 because the resulting carry is discarded.

Multi-byte arithmetic involves operations where both of the operands and results occupy more than a single byte of memory. The simplest multi-byte routines are addition and subtraction of 16-bit two's complement numbers. These operations can be performed using either a single accumulator or
the double accumulator, D, and the following example illustrates both methods. The first two sequences perform addition with single and double byte instruction whereas the second two sequences illustrate subtraction using both single and double byte instructions.

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>96 51</td>
<td></td>
<td>LDAA</td>
<td>ALPHA+1</td>
<td>GET LO BYTE</td>
</tr>
<tr>
<td>9B 61</td>
<td>ADDA</td>
<td>BET+1</td>
<td></td>
<td>ADD LO BYTES</td>
</tr>
<tr>
<td>97 71</td>
<td>STAA</td>
<td>GAMMA+1</td>
<td></td>
<td>STASH LO BYTE</td>
</tr>
<tr>
<td>96 50</td>
<td></td>
<td>LDAA</td>
<td>ALPHA</td>
<td>GET HI BYTE</td>
</tr>
<tr>
<td>99 60</td>
<td>ADDC</td>
<td>BETA</td>
<td></td>
<td>ADD HI BYTES</td>
</tr>
<tr>
<td>97 70</td>
<td>STAA</td>
<td>GAMMA</td>
<td></td>
<td>STASH HI BYTE</td>
</tr>
<tr>
<td>DC 50</td>
<td>LDD</td>
<td>ALPHA</td>
<td></td>
<td>GET ALPHA</td>
</tr>
<tr>
<td>D3 60</td>
<td>ADDD</td>
<td>BETA</td>
<td></td>
<td>ADD TO BETA</td>
</tr>
<tr>
<td>DD 70</td>
<td>STD</td>
<td>GAMMA</td>
<td></td>
<td>STASH IN GAMMA</td>
</tr>
<tr>
<td>96 51</td>
<td></td>
<td>LDAA</td>
<td>ALPHA+1</td>
<td>GET LO BYTE</td>
</tr>
<tr>
<td>90 61</td>
<td>SUBA</td>
<td>BET+1</td>
<td></td>
<td>SUB LO BYTES</td>
</tr>
<tr>
<td>97 71</td>
<td>STAA</td>
<td>GAMMA+1</td>
<td></td>
<td>STASH LO BYTE</td>
</tr>
<tr>
<td>96 50</td>
<td></td>
<td>LDAA</td>
<td>ALPHA</td>
<td>GET HI BYTE</td>
</tr>
<tr>
<td>92 60</td>
<td>SBCA</td>
<td>BETA</td>
<td></td>
<td>SUB HI BYTE</td>
</tr>
<tr>
<td>97 70</td>
<td>STAA</td>
<td>GAMMA</td>
<td></td>
<td>STASH HI BYTE</td>
</tr>
<tr>
<td>DC 50</td>
<td>LDD</td>
<td>ALPHA</td>
<td></td>
<td>GET ALPHA</td>
</tr>
<tr>
<td>93 60</td>
<td>SUBD</td>
<td>BETA</td>
<td>ALPHA-BETA</td>
<td>RESULT TO GAMMA</td>
</tr>
<tr>
<td>DD 70</td>
<td>STD</td>
<td>GAMMA</td>
<td></td>
<td>RESULT TO GAMMA</td>
</tr>
</tbody>
</table>

Subroutines can be developed for multi-byte BCD addition with a variety of calling conventions and restrictions. Two such routines are shown in Figures 4-23 and 4-24. The first routine, BCDAD1, is very short and serves to illustrate the basic algorithm. Its restrictions, however, are somewhat severe. The second routine is much more general, but the generality results in both increased length and execution time. The BCDAD1 routine (Figure 4-23) executes in approximately 100 cycles whereas the BCDAD2 routine (Figure 4-24) requires approximately 650 cycles for 4-byte addends.

Expanding BCD subtraction to multi-byte arithmetic can be accomplished in a similar manner to BCD addition. The routine in Figure 4-25, BCDSB1, illustrates a simple but restrictive BCD subtraction routine. It involves one loop but care must be taken to preserve the value of the C-bit. Accumulator B in BCDSB1 is used for this purpose. The statements at lines 38 and 39 effectively implement an Add with carry. Accumulator B is cleared if the former addition generated no carry and set to one if a carry was generated.

A more general routine is illustrated in Figure 4-26 where the routine BCDSB2 also uses Accumulator B as a pseudo carry bit. Execution times are approximately 140 cycles for BCDSB1 (Figure 4-25) and approximately 700 cycles for BCDSB2 (Figure 4-26) for 4-byte arguments.

4.4.6 Multiplication

A variety of multiplication routines can be written for the MC6801 with the principle considerations being:

1. the calling convention and associated restrictions,
2. compatibility with signed or unsigned arguments,
3. size of arguments and product, and
4. whether minimum length or minimum execution speed is sought.
**Figure 4-23. BCD Addition Routine: BCDAD1**
PAGE 001 BCDAD2 *SA:1 BCDAD2 *** RE-ENTRANT GENERAL BCDADD ROUTINE ***

00001 NAM  BCDAD2
00002 OPT  Z01,LLEN=80
00003 TTL  *** RE-ENTRANT GENERAL BCDADD ROUTINE ***

*********************************************************
* *
* BCDAD2 -- A RE-ENTRANT BCD ADDITION ROUTINE *
* *
* CALLING CONVENTION: *
* *
LDX #PACKET
JSR BCDAD2

* WHERE PACKET IS DEFINED AS:

FCB LENGTH (BYTES)
FDB ADDRESS OF LSB OF ARG1
FDB ADDRESS OF LSB OF ARG2
FDB ADDRESS OF LSB OF RESULT

* RESTRICTIONS:

ARG1, ARG2, AND RESULT MUST BE THE SAME LENGTH
ROUTINE DESTROYS X, ACCA, AND ACCB

* NOTE:

OVERFLOW MAY BE DETECTED BY TESTING THE C-BIT
ON RETURN. IF C=0, NO OVERFLOW OCCURRED;
IF C=1, THE SUM OVERFLOWED.

*********************************************************

ORG $1000

0003A 1000

A BCDAD2
LDAB #7
MOVE IT ALL TO STACK

A BCD000
LDAA 6,X
NEXT ONE

1004
PSHA

1005
DEX
BACK OFF ONE

1006
DECB
MAYBE DONE

1007
BNE  BCD000
NOW USE X TO POINT AT STACK

1009
TSX

100A
CLC
INITIALIZE CARRY BIT

100B
LDX 1,X
GET ARG 1

100D
LDAA X

Figure 4-24. BCD Addition Routine: BCDAD2

4-40
PAGE 002  BCDAD2  .SA:1  BCDAD2  *** RE-ENTRANT GENERAL BCDADD ROUTINE ***

00057A 100F 30  TSX  ADD TO ARG 2
00058A 1010 EE 03  A  LDX  3,X
00059A 1012 A9 00  A  ADCA  X
00060A 1014 19  DAA  ADJUST FOR BCD
00061A 1015 30  TSX
00062A 1016 EE 05  A  LDX  5,X  PUT IN RESULT
00063A 1018 A7 00  A  STAA  X

00065  * BUMP ALL ADDRESSES BACK ONE

00067A 101A C6 03  A  LDAB  #3  DO IT THREE TIMES
00068A 101C 30  TSX  POINT AT STACK AGAIN
00069A 101D A6 02  A  BCD002  LDAA  2,X  TRAP ROLLOVER
00070A 101F 26 02 1023  BNE  BCD003  NOT YET
00071A 1021 6A 01  A  DEC  1,X  DEC HIGH BYTE TOO
00072A 1023 6A 02  A  BCD003  DEC  2,X  DEC LOW BYTE
00073A 1025 08  INX  NEXT ADDR
00074A 1026 08  INX
00075A 1027 5A  DECB  ONE LESS NOW
00076A 1028 1F 101D  BNE  BCD002  MORE TO GO
00078A 102A 30  TSX
00079A 102B 6A 00  A  DEC  X  SEE IF DONE
00080A 102D 26 DC 100B  BNE  BCD001  NOT YET

00082  * ALL DONE, CLEAN UP & LEAVE

00084A 102F C6 07  A  LDAB  #7  BUMP SP
00085A 1031 3A  ABX
00086A 1032 35  TXS  NOW TRANSFER TO SP
00087A 1033 39  RTS

00089  END

TOTAL ERRORS 00000--00000

Figure 4-24. BCD Addition Routine: BCDAD2 (Continued)
Figure 4-25. BCD Subtraction Routine: BCDSB1
*** RE-ENTRANT GENERAL BCDSUB ROUTINE ***

*********************************************************
* B C D S B 2 — A RE-ENTRANT BCD SUBTRACTION ROUTINE *
* CALLING CONVENTION:                                     *
* WHERE PACKET IS DEFINED AS:                             *
* FCB LENGTH (BYTES)                                      *
* FDB ADDRESS OF LSB OF SUBTRAHEND                        *
* FDB ADDRESS OF LSB OF MINUEND                           *
* FDB ADDRESS OF LSB OF RESULT                            *
* RESTRICTIONS:                                           *
* SUBTRAHEND, MINUEND, AND RESULT MUST BE THE SAME LENGTH *
* ROUTINE DESTROYS X, ACCA, AND ACCB                      *
* NOTE:                                                   *
* UNDERFLOW MAY BE DETECTED BY TESTING THE C-BIT          *
* ON RETURN. IF C=1, NO UNDERFLOW OCCURRED;               *
* IF C=0, MINUEND > SUBTRAHEND.                          *
*********************************************************

ORG $1000

A BCDSB2 LDAB #7 MOVE IT ALL TO STACK
A BCDO00 LDAA 6,X NEXT ONE
PSHA
DEX BACK OFF ONE
DECB MAYBE DONE
BNE BCD000 NOT YET
TSX NOW USE X TO POINT AT STACK

* STACK NOW LOOKS LIKE THIS

* LENGTH (BYTES)
* MS BYTE - ADDR OF SUBTRAHEND
* +2 LS BYTE
* MS BYTE - ADDR OF MINUEND
* +4 LS BYTE
* +5 MS BYTE - ADDR OF RESULT
* +6 LS BYTE

A BCDO01 LDAB #1
A BCDO02 LDAA #99

Figure 4-26. BCD Subtraction Routine: BCDSB2
Figure 4-26. BCD Subtraction Routine: BCDSB2 (Continued)
One of the decisions which must be made is whether to use the MC6801 unsigned multiply instruction. Multi-byte routines which utilize this instruction will typically be somewhat longer than the alternate method but will execute faster.

### 4.4.6.1 MULTIPLICATION USING THE MUL INSTRUCTION

The MUL instruction multiplies the A accumulator by the B accumulator and puts the result in the double accumulator, D. Both integers in the multiplication are treated as unsigned binary values. Multiplication of two unsigned bytes by two other unsigned bytes is illustrated below.

\[
\begin{array}{c}
01 & 02 \\
\times & 03 & 04 \\
\hline
00 & 08 & (02*04) \\
00 & 04 & (01*04) \\
00 & 06 & (02*03) \\
00 & 03 & (01*03) \\
00 & 03 & 0A & 08 \\
\end{array}
\]

Another example which involves a carry out during the addition is as follows:

\[
\begin{array}{c}
FF & FF \\
\times & FF & FF \\
\hline
FE & 01 \\
FE & 01 \\
FE & 01 & \text{--- partial sum: FD 00 01} \\
FE & 01 & \text{with carry out} \\
FF & FE & 00 & 01 \\
\end{array}
\]

A routine which implements this procedure is shown in Figure 4-27. The routine contains no loops and executes in approximately 140 cycles. The execution time is nearly independent of the data involved in the multiplication.

A signed binary multiply routine can utilize either the MUL instruction or a technique known as Booth’s algorithm. An example using the MUL instruction will be discussed first. The MUL instruction can be used to multiply two signed integers together by adding a suitable entry and exit to the unsigned routine previously discussed. The entry converts negative integers to positive integers and determines the sign of the product by performing an exclusive OR of the sign bits of the two values.

With this procedure, the two integers are converted to their absolute value and then multiplied using the unsigned method shown in Figure 4-27. After the multiplication has been completed, if the resulting sign is positive then the multiplication is finished; otherwise, the two’s complement of the result must be taken. While this routine is somewhat lengthy, it should be noted that it contains no loops and is therefore considerably faster than alternative methods such as Booth’s algorithm. The resultant algorithm is shown in Figure 4-28. Execution time for the routine is approximately 225 cycles.
MUL16A *** 16 X 16 UNSIGNED MULTIPLY ***

00001
00002
00003
00005
00006
00007
00008
00009
00010
00011
00012
00013
00014
00015
00016
00017
00018
00019
00020
00021
00022
00023
00024
00025
00026
00027
00028
00029
00030
00031
00032
00033
00034
00035
00036
00037
00038
00039
00040
00041
00042
00043
00044
00045
00046
00047
00048
00049
00050
00051
00052
00053
00054
00055
00056
00057
00058

00001 00002 00003 00005 00006 00007 00008 00009 00010 00011 00012 00013 00014 00015 00016 00017 00018 00019 00020 00021 00022 00023 00024 00025 00026 00027 00028 00029 00030 00031

00032
00033
00034
00035
00036
00037
00039
00040
00041
00042
00043
00044
00045
00046
00047
00048
00049
00050
00051
00052
00053
00054
00055
00056
00057
00058

00001 00002 00003 00005 00006 00007 00008 00009 00010 00011 00012 00013 00014 00015 00016 00017 00018 00019 00020 00021 00022 00023 00024 00025 00026 00027 00028 00029 00030 00031 00032 00033 00034 00035 00036 00037 00039 00040 00041 00042 00043 00044 00045 00046 00047 00048 00049 00050 00051 00052 00053 00054 00055 00056 00057 00058

*********************************************************
* MUL16A -- A 16 X 16 MULTIPLY (32 BIT PRODUCT) *
* THIS ROUTINE MULTIPLIES THE TWO UNSIGNED 16-BIT INTEGERS P & Q AND STORES THE PRODUCT IN THE 32-BIT UNSIGNED RESULT, R *
* CALLING CONVENTION: *
* LDX #DATA *
* JSR MUL16A *
* WHERE DATA IS DEFINED AS *
* DATA RMB 2 P *
* RMB 2 Q *
* RMB 4 R P * Q GOES HERE *
* RESTRICTIONS: *
* 1. THE ROUTINE IS RE-ENTRANT PROVIDING EACH CALLER SUPPLIES ITS OWN DATA AREA. *
* 2. THE DATA AREA MUST BE DEFINED AS ABOVE. *
*********************************************************

ORG $1000

00000 A P EQU 0 OFFSET FOR P
00002 A Q EQU 2 OFFSET FOR Q
00004 A R EQU 4 OFFSET FOR R

00000 A P EQU 0 OFFSET FOR P
00002 A Q EQU 2 OFFSET FOR Q
00004 A R EQU 4 OFFSET FOR R

00035
00036
00037
00039 A 1000 4F MUL16A CLRA CLR TWO HIGH BYTES OF RESULT
00040 A 1001 5F CLRB
00041 A 1002 ED 04 A STD R,X
00043 A 1004 A6 01 A LDA P+1,X P+1 * Q+1
00044 A 1006 E6 03 A LDA B Q+1,X
00045 A 1008 3D MUL
00046 A 1009 ED 06 A STD R+2,X
00048 A 100A 00 A LDA P,X P * Q+1
00049 A 100D E6 03 A LDA Q+1,X
00050 A 100F 3D MUL
00051 A 1010 E3 05 A ADDD R+1,X
00052 A 1012 ED 05 A STD R+1,X
00053 A 1014 24 02 1018 BCC MUL002 CHK FOR CARRY
00054 A 1016 6C 04 A INC R,X
00056 A 1018 A6 01 A MUL002 LDA P+1,X P+1 * Q
00057 A 101A E6 02 A LDA Q,X
00058 A 101C 3D MUL

Figure 4-27. Unsigned Multiplication: MUL16A

4-46
Figure 4-27. Unsigned Multiplication: MUL16A (Continued)
MUL16B -- A 16 X 16 MULTIPLY (32 BIT PRODUCT)

** THIS ROUTINE MULTIPLIES THE TWO
** SIGNED 16-BIT INTEGERS P & Q AND
** STORES THE PRODUCT IN THE 32-BIT
** SIGNED RESULT, R

** CALLING CONVENTION:

** WHERE DATA IS DEFINED AS

** DATA RMB 2 P
** RMB 2 Q
** RMB 4 R  P * Q GOES HERE
** RMB 5 TEMP

** RESTRICTIONS:

** 1. THE ROUTINE IS RE-ENTRANT PROVIDING EACH
** CALLER SUPPLIES ITS OWN DATA AREA.
** 2. THE DATA AREA MUST BE DEFINED AS ABOVE.

** THIS EMBEDDED SUBROUTINE TAKES THE TWO'S
** COMPLEMENT OF THE A:B ACCUMULATOR AND
** SETS THE SIGN FLAG FOR THE RESULT.

** THE MUL16B ENTRY POINT IS HERE

Figure 4-28. Signed Multiplication Routine: MUL16B

4-48
Figure 4-28. Signed Multiplication Routine: MUL16B (Continued)
4.4.6.2 MULTIPLICATION USING BOOTH’S ALGORITHM. When memory space is critical and execution time is of little consequence, then Booth’s algorithm can be effectively used to sacrifice speed for memory space. The procedure can be simply stated as:

1. Clear the product (result area) and initialize the shift count;
2. Test the next bit of the multiplier (from LSB to MSB) assuming an imaginary 0 bit to the immediate right of the LSB of the multiplier;
3. If the bits are equal, then go to step 6;
4. If there is a 0 to 1 transition, subtract the multiplicand from the product and go to step 6;
5. If there is a 1 to 0 transition, add the multiplicand to the product;
6. Clear the previous LSB of the multiplier;
7. Shift the multiplier right (toward the LSB) one bit with the LSB going to LSB of multiplier;
8. Shift the product right (toward the LSB) one bit with the most significant bit remaining the same;
9. If any bits remain to be tested in the multiplier, go to step 2; else finished.

Figure 4-29 illustrates the typical steps involved in an actual calculation. A flowchart and assembly listing for a routine using Booth’s algorithm with the MC6801 instruction set is shown in Figures 4-30 and 4-31. The worst case execution time results when alternate additions and subtractions are required in 16 operations. Execution time is on the order of 1300 cycles and is data dependent. This time compares with 225 cycles when using the MUL16B routine shown in Figure 4-28.

![Figure 4-29. Multiplication Using Booth’s Algorithm](image-url)
MULT16

Clear the Working Registers
This includes the Previous LS Bit of the Multiplier Test Byte
Initialize the Shift Count to 16

Does the LS Bit of the Multiplier = the Previous LS Bit?
Yes

Does the LS Bit of the Multiplier = 0?
Yes 1 to 0
No 0 to 1

Add the Multiplicand to the Product with the MS Bytes Lined Up
Subtract the Multiplicand from the Product with the MS Bytes Lined Up

Clear the Previous LS Bit of the Multiplier Test Byte

Shift the Multiplier Right One Bit with the LS Bit Going into the LS Bit of the Multiplier Test Byte

Shift the Product Right One Bit, the MS Bit Remaining the Same

Decrement the Shift Counter

Return from Subroutine

Does the Shift Counter = 0?
Yes
No

Note: "Product" in this flowchart is synonymous with "Result"

Figure 4-30. Flowchart for Booth's Algorithm

4-51
MULT16

PAGE 001

00001  NAM    MULT16
00002      OPT     Z01,LLEN=80

00004

*********************************************************
00005          M U L T 1 6   -- 16 X 16 2'S COMPLEMENT MULTIPLY
00007          (32-BIT 2'S COMP PRODUCT)
00008
00009          THIS ROUTINE USES BOOTH'S ALGORITHM
00010          TO MULTIPLY TWO 16-BIT 2'S COMPLEMENT
00011          INTEGERS YIELDING A 32-BIT SIGNED
00012          PRODUCT.
00013
00014          CALLING CONVENTION:
00015
00016          LDX #DATA
00017          JSR MULT16
00018
00019          WHERE DATA IS DEFINED AS:
00020
00021          DATA RMB 2 P MULTIPLIER
00022          RMB 2 Q MULTIPLICAND
00023          RMB 4 R RESULT
00024          RMB 4 TEMP
00025
00026          RESTRICTIONS AND NOTES:
00027
00028          THE ROUTINE IS RE-ENTRANT IF EACH CALLER
00029          PROVIDES ITS OWN DATA AREA
00030
00031

*********************************************************

A        P1 EQU 0 MULTIPLIER
00034       A Q EQU 2 MULTIPLICAND
00035       A R EQU 4 RESULT
00036       A S EQU 9 PREVIOUS LS BIT
00037       A T EQU 10 SHIFT COUNT
00038       A P EQU 11 WORKING COPY OF MULTIPLIER

00040A 1000 ORG $1000

00042A 1000 4F MULT16 CLRA CLEAR THE WORKING REGS
00043A 1001 5F CLRB
00044A 1002 ED 04 A STD R,X CLEAR OUT RESULT AREA
00045A 1004 ED 06 A STD R+2,X
00046A 1006 A7 09 A STAA S,X CLEAR PREVIOUS LS BIT TOO
00047A 1008 86 10 A LDAA #16 INZ SHIFT COUNT
00048A 100A A7 0A A STAA T,X
00049A 100C EC 00 A LDD P1,X MAKE WORK COPY OF MULTIPLIER
00050A 100E ED 0B A STD P,X

00052
00053          * BOOTH'S MAIN LINE BEGINS HERE
00054
00056A 1010 A6 0C A MULT01 LDAA P+1,X GET LSB OF MULTIPLIER
00057A 1012 84 01 A ANDA #1 ISOLATE LSB
00058A 1014 16 TAB SAVE

Figure 4-31. Signed Multiplication Routine: MULT16

4-52
4.4.7 Division

Unsigned binary division can be performed using the self-restoring method; there is no hardware divide instruction available on the MC6801. The algorithm used for this method is:

1. Left justify the divisor while keeping track of how many times the divisor was shifted.
2. Subtract the divisor from the dividend; the carry bit indicates whether the dividend is larger or smaller than the divisor.
3. If the C-bit is clear (dividend larger than divisor) then go to step 4; else shift the quotient one left with the least significant bit cleared and add the divisor back to (restore) the dividend. Go to step 5.
4. Shift the quotient one bit left with the least significant bit set.
5. Shift the divisor one right.
6. Decrement the shift counter; if not zero, go to step 2.

A flowchart outlining this algorithm is shown in Figure 4-32. A routine which implements this algorithm using the MC6801 instruction set is shown in Figure 4-33. This routine, DIV16B, performs an unsigned division and provides both the quotient and remainder. If a division by zero is attempted, however, the routine returns with the carry bit (C-bit) set; otherwise the C-bit is clear. Instruction timing for the routine is data dependent but is on the order of 280 cycles (approximately).
Figure 4-32. Flowchart for Unsigned Division
*** 16 BY 16 UNSIGNED DIVIDE ***

00001  NAM   DIV16B
00002  OPT   Z01, LLEN=80
00003  TTL   *** 16 BY 16 UNSIGNED DIVIDE ***

*****************************************************************************
* D I V 1 6 B -- 16-BIT BY 16-BIT UNSIGNED DIVIDE
* THIS ROUTINE DIVIDES TWO UNSIGNED 16-BIT INTEGERS PROVIDING A 16-BIT QUOTIENT AND REMAINDER.
* CALLING CONVENTION:
*   LDX  #DATA
*   JSR  DIV16B
* WHERE DATA IS DEFINED AS:
*   DATA RMB 2   DIVIDEND
*   RMB 2   DIVISOR
*   RMB 2   QUOTIENT
* RETURNS:
*   1. QUOTIENT IN LOCATION SPECIFIED, AND
*   2. REMAINDER IN THE A:B ACCUMULATOR.
*   3. THE CARRY BIT IS CLEARED IF THE DIVISOR IS NON-ZERO; IF DIVISOR IS ZERO, THE CARRY BIT IS SET AND THE QUOTIENT AND REMAINDER ARE UNDEFINED.
* NOTES:
*   1. THE ROUTINE IS RE-ENTRANT PROVIDING EACH CALLER FURNISHES ITS OWN UNIQUE DATA AREA.
*****************************************************************************

000040  0000  A  KT   EQU  0 OFFSET FOR COUNTER
000041  0003  A  DIV   EQU  3 OFFSET FOR DIVISOR
000042  0001  A  QUO   EQU  1 OFFSET FOR QUOTIENT
000043  0005  A  PTR   EQU  5 OFFSET FOR QUOTIENT POINTER

00045A  1000  ORG  $1000

00047A  1000  3C  DIV16B  PSHX MOVE IT ALL TO STACK
00048A  1001  C6  04  A   LDAB #4
00049A  1003  A6  03  A   DIV000 LDAA 3,X NEXT ONE
00050A  1005  36   PSHA
00051A  1006  09   DEX BACK OFF ONE
00052A  1007  5A   DECB
00053A  1008  26  P9  1003  BNE DIV000
00054A  100A  5C   INCB INZ KT TO 1
00055A  100B  37   PSHB
00056A  100C  30   TSX NOW POINT AT STACK WITH X

Figure 4-33. Unsigned Division Routine: DIV16B
00058  * STACK NOW LOOKS LIKE THIS
00059
00060  * +0 COUNTER
00061  * +1 MS BYTE - DIVIDEND & QUOTIENT
00062  * +2 LS BYTE
00063  * +3 MS BYTE - DIVISOR
00064  * +4 LS BYTE
00065  * +5 MS BYTE - POINTER TO DATA
00066  * +6 LS BYTE

00068A 100D EC 03 A LDD DIV,X LEFT JUSTIFY THE DIVISOR
00069A 100F 27 32 1043 BEQ DIV006 UH, OH ... DIVISOR IS ZERO !!
00070A 1011 2B 07 101A BMI DIV002 ALREADY DONE
00071A 1013 6C 00 A DIV001 INC X BUMP COUNT
00072A 1015 05 ASLD
00073A 1016 2A FB 1013 BPL DIV001 DO IT AGAIN
00074A 1018 ED 03 A STD DIV,X
00075A 101A EC 01 A DIV002 LDD QUO,X
00076A 101C 6F 01 A CLR QUO,X
00077A 101E 6F 02 A CLR QUO+1,X

00079A 1020 A3 03 A DIV003 SUBD DIV,X
00080A 1022 24 05 1029 BCC DIV004 DIVISOR STILL OK
00081A 1024 E3 03 A ADDD DIV,X TOO LARGE - RESTORE
00082A 1026 0C CLC
00083A 1027 20 01 102A BRA DIV005

00085A 1029 0D DIV004 SEC
00086A 102A 69 02 A DIV005 ROL QUO+1,X
00087A 102C 69 01 A ROL QUO,X
00088A 102E 64 03 A LSR DIV,X ADJUST DIVISOR
00089A 1030 66 04 A ROR DIV+1,X
00090A 1032 6A 00 A DEC KT,X
00091A 1034 26 EA 1020 BNE DIV003

00093  * CLEAN UP THE STACK AND EXIT

00095A 1036 37 PSHB SAVE THE REMAINDER
00096A 1037 36 PSHA
00097A 1038 EC 01 A LDD QUO,X GET QUOTIENT
00098A 103A EE 05 A LDX PTR,X GET WHERE TO PUT IT
00099A 103C ED 04 A STD 4,X PUT IT BACK
00100A 103E 32 PULA RESTORE REMAINDER
00101A 103F 33 PULB
00102A 1040 0C CLC FLAG OK
00103A 1041 20 01 1044 BRA DIV007
00104A 1043 0D DIV006 SEC FLAG PROBLEM
00105A 1044 38 DIV007 PULX
00106A 1045 38 PULX
00107A 1046 38 PULX
00108A 1047 31 INS
00109A 1048 39 RTS ALL DONE

00111 END

TOTAL ERRORS 00000--00000

Figure 4-33. Unsigned Division Routine: DIV16B (Continued)
CHAPTER 5
THE MC6801 INTERRUPT STRUCTURE

5.0 INTRODUCTION

MC6801 interrupt requests can be generated by any of three different methods: (1) by presenting an appropriate external signal, (2) by enabling interrupts from the Programmable Timer, Serial Communications Interface or Port 3 input strobe, or (3) by executing a Software Interrupt (SWI) instruction. A special type of interrupt, Reset, is excluded from the following discussion.

Seven hardware interrupts and one software interrupt can be generated from all of the possible sources. The interrupts and any associated flag bits are as follows:

1. Non-Maskable Interrupt (NMI)
2. Interrupt Request 1 (IRQ1)
3. Port 3 Input Strobe (IS3 FLAG)
4. Timer Input Capture (ICF)
5. Timer Output Compare (OCF)
6. Timer Overflow (TOF)
7. Serial Communications Interface (RDRF, ORFE, TDRE)
8. Software Interrupt (SWI) Instruction

5.1 INTERRUPT CONSIDERATIONS

Interrupts, in general, can be considered a mechanism with which to provide “service” to a peripheral device. The “service,” for instance, could merely consist of reading the Receiver Data Register in an MC6850 Asynchronous Communications Interface Adapter (ACIA) in response to the keystroke on a keyboard. This same service, however, could also be provided by continuously testing the RDRF (Receiver Data Register Full) flag bit for a “full” condition. When the flag bit indicates “full,” the Receiver Data Register would be read just as in the interrupt example.

Software utilizing the former method of service is usually referred to as “interrupt-driven” while the latter is called “polling.” Either method could be appropriate depending on the application. The program sequence shown in Figure 5-1 illustrates a polling sequence. Note there is an implied assumption that the MPU can remain in the polling loop for an indefinite period of time without system degradation. Examined from a different viewpoint, the MPU is minimally utilized, but the idle time cannot be diverted to performing other tasks because it is totally dedicated. Interrupt-driven service provides a means to utilize the excess MPU capacity.
Figure 5-1. Polling Loop Sequence

MPU utilization is not, however, always enhanced by using interrupts. MPU interrupt service always incurs a fixed "overhead" of execution time. In the MC6801, this "overhead" consumes 22 MPU E-cycles of which 12 cycles are needed to execute the interrupt sequence while the remaining 10 cycles are required to restore the former machine state at the end of the service routine. This interrupt response time latency could be intolerable in servicing a very high speed peripheral. If the time between requests for service is reduced sufficiently, it can be shown that using interrupt-driven service, in preference to polling, can result in reduced MPU performance. Interrupt servicing is most effective when the response time latency is insignificant relative to the average time between interrupt requests. Fortunately, this condition is met by a great many practical applications.

Interrupts can also be used to provide a hierarchical response in systems characterized by a large variation in the time between service requests. In these applications, service is usually provided such that devices requesting service more often are given a higher priority.

Polling loops, however, can also be structured to provide some hierarchical capability. Consider, for example, the program sequence shown in Figure 5-2. In this example, service requests are honored with a hierarchy of Device 1, Device 2, and Device 3. Note that if Device 1 requests service at a sufficiently high rate, however, service requests for Device 2 and Device 3 will never be honored.

Another significant feature of polling is that once service has begun for either Device 2 or Device 3, requests for service by Device 1 will not be honored again until service has been completed. This could produce an intolerable situation which can possibly be remedied by the use of interrupts. If interrupts were used instead of polling, the service for either Device 2 or Device 3 could be suspended whenever Device 1 needed service. This prioritization of service cannot be efficiently achieved with polling. A combination of polling and interrupt service can also be used to advantage in some applications.

*CHECK DEVICE 1
DEVI LDA STAT1 CHECK DEVICE 1
  BPL DEV2 NO SERVICE REQUIRED
  JSR SVDEV1 SERVICE DEVICE 1
  BRA DEV1 CHECK IT AGAIN

*CHECK DEVICE 2
DEV2 LDA STAT2 CHECK DEVICE 2
  BPL DEV2 NO SERVICE REQUIRED
  JSR SVDEV2 SERVICE DEVICE 2
  BRA DEV1 CHECK DEVICE 1 AGAIN

*CHECK DEVICE 3
DEV3 LDA STAT3 CHECK DEVICE 3
  BPL DEV1 LAST ONE
  JSR SVDEV3 SERVICE DEVICE 3
  BRA DEV1 START OVER

Figure 5-2. Hierarchical Polling Loop Sequence
5.2 MC6801 INTERRUPT GENERATION

MC6801 interrupts have three attributes: (1) priority of service, (2) edge or level sensitivity, and (3) whether the interrupt can or cannot be masked. The interrupt with the highest priority, excluding Reset, is the Non-Maskable Interrupt (NMI) which can be generated by a high-to-low voltage transition on the MCU NMI pin. This can be considered an edge-sensitive interrupt and is one of three such interrupts in the MC6801. The Port 3 Input Strobe (IS3) and the Timer Input Capture interrupts are also edge-triggered. All other interrupts are level-sensitive for which the request line must be held low long enough for the MCU to both recognize and respond to them.

The MC6801 design philosophy which is inherent in level-sensitive interrupt response is that a peripheral device requesting service will "pull down" the appropriate interrupt line and the service routine, as part of its duties, will clear or reset the interrupt request. Pulse inputs as level-sensitive interrupts are strongly discouraged and require precise timing and a detailed knowledge of the MC6801 internal interrupt circuitry in order to achieve predictable results.

5.2.1 Non-Maskable Interrupt (NMI)

The MC6801 Non-Maskable Interrupt, NMI, is the highest prioritized interrupt (excluding Reset) and, as its name implies, can be masked only with external circuitry. The NMI interrupt can be used to signal loss of power to the processor thereby facilitating an orderly power-down sequence.

An NMI request is generated by a high-to-low voltage transition (negative edge). The generation mechanism need not be designed such that the service routine resets the device producing the NMI interrupt, although it may. If the NMI request is asynchronous with respect to E, it should remain low for at least one MPU E-cycle to be recognized under all conditions.

The NMI input must not remain unconnected in any MC6801-based system because there is no internal pullup resistor. Therefore, an unconnected NMI input could result in spurious NMI requests. Pullup resistors between 3 k and 10 k ohms are typically used and depend upon the current capability of the circuit element generating the interrupt.

5.2.1.1 SYSTEM CONSIDERATIONS IN USING NMI. When using the NMI interrupt, there are several considerations which must be kept in mind in order to design error-free systems. The first consideration is that an MC6801 cannot internally control when it is ready to recognize its initial NMI interrupts. Prior to responding to any interrupt, it is first necessary to initialize the MPU stack pointer. A system will usually fail if it allows an NMI to occur between an MPU Reset (during which the contents of the stack pointer are undefined) and the stack pointer initialization instruction.

NOTE

NMI negative edges which occur while RESET is low are ignored. If an NMI negative edge occurs within two E-cycles following the positive edge of RESET, however, the MPU will execute an NMI interrupt sequence prior to executing the instruction at the Reset vector.
Another important consideration is that an NMI service routine can be interrupted at any time by yet another edge of the NMI pin after the 10th cycle of an NMI interrupt sequence. If an MC6801 system attempts to service NMI requests for which the average time between requests is less than the service routine execution, the program will eventually fail for lack of sufficient stack memory.

While more than one M6800 family device can be wire-ORed to the NMI line, difficulties can arise if mutual exclusion of interrupt requests is not guaranteed. Suppose more than one NMI request exists when the request for the device currently being serviced is cleared. The NMI line will remain low but no new negative edge will be generated. Without this edge, any remaining pending requests on the NMI line will be ignored. The NMI line can be configured as a level-sensitive maskable interrupt line with minimal additional external hardware and would alleviate the difficulties associated with this problem. This technique is discussed in detail in the next section.

System development tools sometime utilize the NMI interrupt for various debugging functions. The designer should investigate whether the development system will permit "user" NMI interrupts in order to avoid unpleasant surprises during checkout.

5.2.1.2 USING NMI AS A MASKABLE INTERRUPT. The NMI interrupt can be used as a level-sensitive highest priority maskable interrupt with minimal external hardware. Suppose the following design objectives are proposed for an external NMI mask circuit:

1. NMI must be masked when the MPU comes out of Reset,
2. NMI must be maskable by using software to set and clear the mask,
3. Inputs to the circuits can be considered level-sensitive, and
4. Nested (multiple) NMI interrupts cannot occur.

Further suppose that the following is a requirement on hardware devices which utilize the NMI line: external devices must "pull down" the NMI line and hold it down until reset by software.

The design can be implemented by using an input/output line as a mask control line, an externally connected OR-gate as the masking device, and proper software control. Consider the circuit shown in Figure 5-3. Any available data line serves as the mask control line. The pullup resistor on the mask control line ensures that the line will be high coming out of Reset which inhibits any NMI interrupt until software enables the circuit by clearing the mask control line.

Software initialization of the mask control line must be performed carefully to avoid enabling a premature interrupt. The Data Direction Register for all of the MC6801 I/O ports is cleared by Reset which configures them as input data ports. Prior to configuring the mask control bit as an output, it must be set by writing to the Data Register. If this order is reversed, the line could go low (enabling NMI) until a "1" is written to the Data Register. The NMI line can be unmasked by clearing the mask control bit. If an NMI is pending, the output state of the OR-gate will change from a high to a low and generate the interrupt.

One of the device restrictions stated that it should keep the line low until reset by software. This restriction serves to prevent further NMI interrupts while the service routine is executing. No new edge can be generated on the NMI pin while the level is held low.

Software must be used correctly in servicing the interrupting devices in order to achieve the desired results. A software example which utilizes the NMI mask circuit is shown in Figure 5-4.
Figure 5-3. An NMI Mask Circuit

*INITIALIZE DATA PORT
LDAA #%10000000  KEEP MASK BIT HIGH
STAA P1DATA    WRITE PORT DATA REG FIRST
STAA P1DDR     WRITE DATA DIRECTION REG

*READY FOR NMI NOW, CLEAR MASK
LDAA P1DATA    GET P17
ANDA #%01111111 CLEAR B7 ONLY
STAA P1DATA    ARM NMI

*SERVICE NMI HERE
NMISVC LDAA #%10000000  SET NMI MASK FIRST
STAA P1DATA    MASK HAS BEEN SET
LDAA DEVST     CLEAR DEVICE FLAG
LDAA DEVDAT    READ STATUS, READ DATA

Figure 5-4. Software for an NMI Mask

The NMI service routine must not reset or "re-arm" the external device until after the mask control bit has been set. The following steps should be performed during interrupt service:

1. the mask bit must be set,
2. the device must be reset (NMI cleared),
3. other services may be performed as required,
4. the mask bit must be cleared and then be followed immediately by a "Return from Interrupt" (RTI) instruction.

The last step will generate a new edge if another NMI is pending but it will occur too late to be immediately acted upon. One instruction will be executed before taking the interrupt; that instruction must be the RTI instruction to prevent nesting of NMI interrupts.
5.2.2 MC6801 Maskable Interrupts (IRQ1 and IRQ2)

Maskable interrupts can be generated by (1) pulling the processorIRQ1 line low, (2) by pulling down Port 3 input strobe (IS3) or, (3) by the Programmable Timer or Serial Communications Interface (SCI). The timer and SCI generate interrupts by using an internal interrupt line, IRQ2, which is not accessible to external hardware. This line is shown in Figure 1-1. IRQ1 has priority over any IRQ2 interrupt if both occur at the same time. All maskable interrupts are level-sensitive except the Port 3 Input Strobe (IS3), and Timer Input Capture (ICF) interrupts which are both edge sensitive.

Maskable interrupts are controllable at two different levels in the MC6801. At the processor level, a bit in the Condition Code Register, the I-bit, either inhibits (I-bit set) or enables (I-bit clear) all maskable interrupts. This bit is always set by the MPU during the Reset and interrupt sequences, and during the execution of certain instructions. The I-bit can be cleared only by executing specific MC6801 instructions.

Interrupts can also be controlled individually in MC6800 family parts by utilizing the mask or interrupt enable bit of the device. The device interrupt control and flag bit are both used to generate an interrupt request as depicted in Figure 5-5. Operation of the flag bit, however, is functionally independent of whether or not interrupts are being used.

Figure 5-5. Generalized Interrupt Control and Generation
Several MC6800-family devices (or devices with open collector outputs) can be wire-ORed to a common IRQ1 line. If more than one device is connected to the IRQ1 input, it should be pulled high with a pullup resistor to VCC for good performance. There is no internal IRQ1 pullup resistor in the MC6801. If the IRQ1 input is not used, it can be pulled high to VCC using a pullup resistor for good noise immunity. Values between 3.3 k and 10 k ohm are typically used for pullup resistors and depend on the current capacity of the circuit element generating the interrupt.

5.3 MC6801 INTERRUPT RESPONSE

The MC6801 samples the level on each of its three interrupt request lines (NMI, IRQ1, IRQ2) on each negative edge of E (Enable). The level is then clocked through circuits which establish synchronization and refine its electrical characteristics. For the Non-Maskable Interrupt Request, the level is coupled to an edge detector. Finally, each sampling is presented to the “Set” input of an S-R flip-flop. It is the output of these three S-R flip-flops which is used to indicate to the MPU whether an interrupt is, or is not, pending.

Note that because interrupts are sampled on each negative edge of E (Enable), an asynchronous signal (with respect to E) must have a pulse width of at least one E-cycle to be sure the level is active when sampled. Logic diagrams for the sampling circuits for NMI and IRQ1 are shown in Figure 5-6.

Figure 5-6. Logic Diagrams for Interrupt Sampling
A description of the elements used in the logic diagram was presented in Section 3.1.8. The active signals in the diagram are:

- \( \phi 2 \) is an internal primary clock which is derived from and is in phase with E.
- \( \phi 1 \) is a second primary internal clock which is derived from \( \phi 2 \) and is a non-overlapping clock with respect to \( \phi 2 \).
- IS3 is active low whenever (1) the MCU is operating in Modes 4 or 7, (2) IS3 FLAG is set and (3) IS3 IRQ1 ENABLE is set. If all three conditions are not true, IS3 is held high.

One should note from Figure 5-6 that the logic diagrams for capturing the IRQ1 and NMI levels are nearly identical. When \( \phi 2 \) is high, the \( \phi 2 \) coupler is "on" while the two \( \phi 1 \) couplers are "off" and the input level is coupled to the transparent latch. When the negative edge of \( \phi 2 \) (or E) occurs, the two couplers assume the opposite state which latches the level and transmits it to subsequent logic stages.

A separate S-R flip-flop is provided for each of the NMI, IRQ1, and IRQ2 interrupts. Each S-R flip-flop is "Set" only by an interrupt request. The NMI flip-flop is "Reset" during Reset and the tenth cycle of an interrupt sequence excluding one generated by an SWI instruction. The two maskable interrupt flip-flops are "Reset" during Reset and whenever the I-bit is set. Because all interrupt sequences set the I-bit in the tenth cycle, they also "Reset" the IRQ1 and IRQ2 flip-flops. While it is set, the I-bit holds both IRQ1 and IRQ2 interrupt flip-flops in "Reset." By design, if both "Set" and "Reset" conditions exist, the flip-flop is reset.

Note that all three interrupt flip-flops are cleared by any interrupt sequence with the exception that the NMI flip-flop is not cleared by an SWI sequence. An NMI request is never lost with this scheme because of its service priority. If one is attempting to pulse level-sensitive interrupts, however, this characteristic can lead to problems. To illustrate the difficulty, suppose that the IRQ1 line is pulsed such that the IRQ1 interrupt flip-flop is set. Prior to the interrupt being serviced, however, an NMI request occurs. During the interrupt sequence, all interrupt flip-flops are reset and the pulsed IRQ1 interrupt request is "lost"!

Each MC6801 instruction is executed to completion before acting upon any interrupt. Interrupts are clocked into their respective S-R flip-flops during the intervals, W1 and W2, shown in Figure 5-7 where the LDAA and STAA instructions were arbitrarily chosen for this example. The two windows, W1 and W2, are the intervals of time during which an interrupt request must be presented in order for the interrupt sequence to begin upon completion of the STAA instruction. If the request is presented slightly before this window, the interrupt will occur during the STAA instruction. If presented slightly after this window, the MPU will execute one more instruction before responding to the interrupt. Time interval, W1, is for IRQ1 while NMI and IRQ2 use the window, W2.

Note that if an unmasked IRQ1 request occurs between the last cycle of the LDAA instruction and the third cycle of STAA, the next instruction will not be executed. If, however, IRQ1 is not active until the last cycle of the STAA, the next instruction will be executed before responding to the interrupt. During the first cycle of every instruction, the operation code (opcode) is fetched from memory. However, this instruction will not be executed at this time if the output state of the interrupt flip-flops indicate that an interrupt is pending; instead, the MPU will fetch the same opcode again and then stack the machine state. The interrupt service routine will be executed before again accessing the instruction.
The MC6801 interrupt response is depicted in the processor flowchart shown in Figure 5-8. For discussion purposes, it is convenient to divide the flowchart into two autonomous sections. The left portion of the flowchart represents the non-interrupt operation of the processor whereas the right portion represents the interrupt response sequence.

The non-interrupt section of the flowchart is shown in Figure 5-9. If no interrupts occur during program execution, this flowchart adequately describes MPU activity. Note that the I-bit is set during the Reset sequence. Instruction processing begins with the main loop in the flowchart.

The interrupt sequence is initiated by any of the following actions:
1. pulling the level down on the NMI line, or
2. clearing the I-bit and pulling the level down for either IRQ1 or IRQ2, or
3. executing either an SWI or WAI instruction.

The reader should note that while NMI, IRQ1, and IRQ2 can occur asynchronously with respect to MPU activity, interrupt response occurs only at discrete times.

The interrupt section of the flowchart is shown in Figure 5-10. Assuming that one of the above interrupt actions has occurred, the processor updates the I-bit from ITMP (see Figure 5-8 and supplemental notes). It then stacks the machine state as shown in Figure 5-11.

After the registers have been stacked, the processor will set the I-bit, examine the external interrupts again, and select the vector (address pointer to a service routine) corresponding to the current highest priority interrupt. If the interrupt sequence began in response to an IRQ1 and an NMI edge occurred before the vector was fetched, the NMI vector would be selected even though the sequence was initiated in response to IRQ1.
Supplemental Notes to MC6801 Interrupt Flowchart

1. IRQ2 is accessible only to the MC6801 internal bus and is used by the Programmable Timer and Serial Communications Interface (SCI).
2. ITMP is a 1-bit buffer register for the MPU i-bit in the Condition Code Register. Instructions which affect the i-bit put their result in ITMP. ITMP is transferred to the i-bit as shown in the flowchart. The overall effect of this buffering operation can be stated as follows: the effect of any operation which clears the i-bit is delayed one E-cycle. The effect of any operation which sets the i-bit, however, is not delayed.
3. The i-bit masks (inhibits) or enables both IRQ1 and IRQ2 interrupts.
4. The Reset sequence will be entered from any point in the flowchart if a low level is sensed on the RESET line.

Figure 5-8. MC6801 Processor Flowchart
Figure 5-9. MC6801 Non-Interrupt Flowchart
Figure 5-10. MC6801 Interrupt Flowchart
The interrupt sequence requires 12 MPU E-cycles to complete once it has begun, as shown in Figure 5-12. Note that the response time for the NMI and IRQ2 interrupts is one cycle less than that for IRQ1. The IRQ1 recognition circuit includes an additional cycle for synchronization. In the absence of other interrupts (and SWI), the response time to an IRQ1 asynchronous interrupt, therefore, varies from 14 to 23 E-cycles (13 to 22 for NMI or IRQ2) from activation of the signal to service routine opcode fetch. The actual number of E-cycles depends upon which instruction is being executed and how far it has progressed. The shorter response time results from the interrupt occurring during the next to the last cycle of the current instruction. The "worst case" response time results from the IRQ1 request occurring during the last cycle of an instruction which precedes an SWI instruction (the most time consuming MC6801 instruction).

The minimum interrupt response time can, however, be shorter than 14 MPU E-cycles for IRQ1 or 13 for NMI and IRQ2 when operating with at least one other lower priority interrupt enabled. This would occur if the MPU was stacking the machine state in response to a lower priority interrupt when a higher priority interrupt is requested. In the MC6801, the interrupt which triggers the interrupt sequence is not necessarily the interrupt serviced by the MPU. If a higher priority interrupt is recognized by the tenth cycle of the interrupt sequence, it will be serviced instead. Furthermore, a higher priority IRQ2 interrupt will be serviced even if it occurs during the eleventh cycle of the interrupt sequence.

The shortest possible interrupt response times, therefore, are three E-cycles for IRQ2 (Input Capture or Output Compare), four E-cycles for NMI, and five E-cycles for IRQ1. These cycle counts are from signal activation to the service routine opcode fetch cycle and assume a lower priority interrupt had already triggered the interrupt sequence. These response times also include a two 1 cycle delay from the interrupt flip-flop to the priority encoder. The reader should note that if an NMI request occurs no later than the end of the ninth cycle of the interrupt sequence (with setup time, tPCS), it will be serviced immediately. If the NMI request occurs in the tenth cycle, however, it will not be serviced until after completion of the current interrupt sequence.
Figure 5-12. MC6801 Interrupt Sequence
The reader should note that these two extremes of interrupt response time were labeled as "worst" and "shortest." The ability to service higher priority interrupts after the interrupt sequence has been initiated is a decided advantage in certain applications. It should be recognized, however, that "shortest" is not necessarily the "best" in all possible applications. For example, if the NMI interrupt is being used to perform an "approximate" input capture function, (see Chapter 7), it is most certainly not "best" if other interrupts are also enabled. For this function, the response time variability to NMI should be as short as possible in order to make the time measurement more accurate. The characteristic of selecting the highest priority interrupt vector, however, increases the response time uncertainty. When an NMI interrupt can occur with other interrupts, the minimum possible response time is four E-cycles instead of 13 when it is the only interrupt being used.

5.3.1 Selection of Interrupt Vectors

In the tenth cycle of an interrupt sequence, the MPU selects an interrupt vector according to the priority of the interrupt. Except during Reset and execution of the SWI instruction, a priority encoder circuit maps pending interrupts into address lines A1-A3, during the tenth cycle of an interrupt sequence. This mapping, effectively, selects the appropriate interrupt vector. The inputs to the priority encoder circuit are:

1. the output of the NMI interrupt flip-flop,
2. the output of the IRQ1 interrupt flip-flop,
3. ICF • EICI,
4. OCF • EOCI, and
5. TOF • ETOI.

Note that the SCI interrupts \[(RORF + ORFE)•RIE + (TDRE•TIE)\] are treated as a "default" case to the priority encoder: if an IRQ2 interrupt exists and it is not one of the previous cases, then it is assumed to be an SCI interrupt.

All IRQ2 interrupts are generated by combining certain status and control bits from the Programmable Timer and the Serial Communications Interface which are defined in Chapters 6 and 7. The status and control bits are combined to form an IRQ2 interrupt signal analogous to the method shown in Figure 5-5. A proper combination of control and flag signals can cause the IRQ2 flip-flop to be set. Removal of the active level, however, does not clear the flip-flop. The flip-flop is cleared only by the I-bit being set or by Reset.

The IRQ2 inputs to the priority encoder are not latched and have no synchronization delays in the path from the register to the encoder. It is possible, therefore, to generate an IRQ2 interrupt and subsequently remove the identity of the interrupt to the priority encoder. This will occur if either of the following two actions are taken after generating an IRQ2 interrupt but before servicing it (without an intervening higher priority interrupt): (1) the flag bit is cleared or (2) the interrupt enable bit is cleared.

As an illustration of these details, suppose an MC6801 is executing with the I-bit clear and all interrupts enabled when all of the interrupts, except NMI, simultaneously become pending. Now further suppose that during execution of the IRQ1 service routine, the program arbitrarily decides to clear the interrupt enable bits for all of the remaining interrupts. Now consider MPU activity after the Return-from-Interrupt (RTI) instruction is executed.

5-15
During the interrupt sequence for IRQ1, the interrupt flip-flops for IRQ1 and IRQ2 are reset by the I-bit being set which holds the IRQ1 and IRQ2 flip-flops in "Reset." After the IRQ1 is serviced and all of the remaining interrupt enable bits are cleared, the former processor state is resumed (with I-bit clear) by the RTI instruction. With the IRQ2 flip-flop cleared and no active IRQ2 signal to set it again, all of the previous Timer and SCI interrupts are disregarded. From this discussion, it should be apparent that while the I-bit is set, any pending interrupt (other than NMI) can be removed.

As a second example of interrupt operation, suppose that an MC6801 system is executing with the I-bit clear and an IRQ2 interrupt enabled. Further suppose that the program wishes to inhibit this interrupt by clearing its interrupt enable bit. Now suppose the interrupt to be inhibited occurs during execution of the instruction which clears its interrupt enable bit.

Because the IRQ2 flip-flop is set, an IRQ2 interrupt will occur after execution of this instruction, providing a higher priority interrupt does not preempt the interrupt and thereby clear the IRQ2 flip-flop. Assuming this doesn't happen, a problem now exists because the inputs to the priority encoder do not match the interrupt being serviced. In fact, the inputs to the priority encoder could indicate no interrupt at all. In the absence of any interrupt, the priority encoder will always select $FF00:FF11 (SCI interrupt).

**NOTE**

An SEI instruction should precede any instruction which clears an enabled IRQ2 interrupt in order to avoid a spurious SCI interrupt.

An example of this technique is shown in the following sequence.

<table>
<thead>
<tr>
<th>Machine Code</th>
<th>Label</th>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>96 08</td>
<td></td>
<td>LDAA</td>
<td>$8</td>
<td>GET TCSR</td>
</tr>
<tr>
<td>84 FB</td>
<td></td>
<td>ANDA</td>
<td>#$FB</td>
<td>CLR ETOI</td>
</tr>
<tr>
<td>0F</td>
<td></td>
<td>SEI</td>
<td></td>
<td>NO IRQ1 OR IRQ2</td>
</tr>
<tr>
<td>97 08</td>
<td>STAA</td>
<td>$8</td>
<td>ETOI INHIBITED</td>
<td></td>
</tr>
<tr>
<td>0E</td>
<td>CLI</td>
<td></td>
<td>IRQ1 OR IRQ2 ON</td>
<td></td>
</tr>
</tbody>
</table>

As a final example, suppose an MC6801 system is executing with the I-bit clear and any or all maskable interrupts enabled. Now suppose that a control and status register and the data register are in adjacent locations as is the case for the Timer Control and Status Register and the Counter Register ($08 and $09:0A). Let us assume that the programmer is using the LDD (Load Double Accumulator) instruction to access the Control and Status Register in the A accumulator and the most significant byte of the counter in the B accumulator. Finally, let us assume that a timer overflow interrupt occurs during execution of the LDD instruction.

The result of this operation is to indirectly "pulse" the IRQ2 line as shown in Figure 5-13. This will cause the IRQ2 interrupt flip-flop to be set but provide no input to the priority encoder. Thus, when responding to the timer overflow interrupt, $FF00:FF11 will be selected as the interrupt vector. Admittedly, this illustration is pathological inasmuch as it is difficult to imagine a real application which combines both interrupts and polling. It does illustrate, however, that it is possible to indirectly pulse the IRQ2 interrupt using somewhat dubious software.

*Alternatively, one could provide an RTI for a "spurious interrupt" in the service routine. A "spurious interrupt" is defined as a serial interrupt with [RIE*(RDRF + ORFE) + (TIE*TDRE)] false.
5.3.2 MC6801 Operating Mode and Interrupt Vectors

The eight interrupt vectors (including Reset) used by the MC6801 are shown in Figure 5-14. The physical location of these vectors, however, depends upon the operating mode of the processor and can exist as:

1. external memory space,
2. internal ROM,
3. or internal RAM.

<table>
<thead>
<tr>
<th>Vector (MSB:LSB)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFDF:FFFF</td>
<td>Reset</td>
</tr>
<tr>
<td>FFDC:FFFD</td>
<td>Non-Maskable Interrupt (NMI)</td>
</tr>
<tr>
<td>FFFA:FFFB</td>
<td>Software Interrupt (SWI)</td>
</tr>
<tr>
<td>FFFA:FFFB</td>
<td>IRQ1 Interrupt (IRQ1, IS3)</td>
</tr>
<tr>
<td>FFF6:FFFF</td>
<td>IRQ2/Timer Input Capture (ICF)</td>
</tr>
<tr>
<td>FFF4:FFFF</td>
<td>IRQ2/Timer Output Compare (OCF)</td>
</tr>
<tr>
<td>FFF2:FFFF</td>
<td>IRQ2/Timer Overflow (TOF)</td>
</tr>
<tr>
<td>FF5:FFFF</td>
<td>IRQ2/SCI (RDRF, ORFE, TDRE)</td>
</tr>
</tbody>
</table>

Figure 5-14. MC6801 Interrupt Vectors

The MC6801 memory maps (Figures 2-7 through 2-16) indicate the location of the interrupt vectors with respect to the operating mode. It should be noted that Mode 0 is the only MC6801 operating mode for which the interrupt vector location varies as a function of time. In Mode 0, the interrupt vectors are fetched from external memory for the first two cycles following Reset and from the internal memory space thereafter. The result is that the Reset vector will always be fetched from external memory but all accesses to any interrupt vector, except during an MPU Reset, will be to internal memory.
5.4 MC6801 INTERRUPT INSTRUCTIONS

This section discusses specific MC6801 instructions which directly affect the interrupt structure. It includes all MC6801 instructions which set or clear the I-bit or cause the interrupt sequence to be executed.

5.4.1 MC6801 Instructions Affecting the I-Bit

The MC6801 I-bit (bit 4 in the Condition Code Register) serves as the processor interrupt mask and either enables (I-bit clear) or inhibits (I-bit set) both IRQ1 and IRQ2 interrupts. MC6801 instructions which affect the I-bit include:

1. Clear Interrupt Mask (CLI),
2. Set Interrupt Mask (SEI),
3. Transfer Accumulator A to Condition Code Register (TAP),
4. Software Interrupt (SWI), and
5. Return from Interrupt (RTI).

The SEI and SWI instructions always set the I-bit. The CLI instruction always clears the I-bit while the TAP and RTI instructions can either clear or set the I-bit. All instructions which affect the I-bit put their result into a I-bit buffer register, ITMP. If the contents of ITMP is a "1", ITMP is immediately transferred to the I-bit; otherwise, the transfer to the I-bit is delayed for one MPU E-cycle. The reason for the delay is to facilitate use of the Wait for Interrupt instruction and is further discussed in Section 5.4.2.

5.4.1.1 THE CLI AND SEI INSTRUCTIONS. The Clear Interrupt Mask (CLI) instruction is used to enable all maskable interrupts. The clearing of the I-bit is buffered through the ITMP register and is, therefore, delayed one MPU E-cycle. Assuming the I-bit is not already clear and no NMI occurs during instruction execution, the instruction following CLI will always be executed prior to servicing any maskable interrupt.

The Set Interrupt Mask (SEI) instruction results in the I-bit being set and is not delayed. When the I-bit is set, it holds the interrupt flip-flops for IRQ1 and IRQ2 in reset which, effectively, inhibits any maskable interrupt from being recognized. To further illustrate operations affecting the I-bit, the reader should note that a pending maskable interrupt will never be serviced during the following three-instruction sequence (assuming no NMI):

```
LOOP CLI CLEAR INTERRUPT MASK
  SEI SET IT AGAIN
  BRA LOOP DO IT AGAIN
```

5.4.1.2 THE TAP INSTRUCTION. The Transfer Accumulator A to Condition Code Register (TAP) instruction can either clear or set the I-bit depending upon the value of bit 4 in the A accumulator. When the TAP instruction is executed, the I-bit in the CCR is set and bit 4 of the A accumulator is put into the I-bit buffer, ITMP. The value of ITMP overwrites the I-bit during the first cycle of the next instruction, but it occurs too late to trigger an interrupt sequence until after this instruction has been completed.
The next instruction in line following single or multiple TAP instructions will always be executed (unless an NMI occurs) regardless of whether bit 4 is a "0" or a "1". Note that a maskable interrupt will never occur in the following sequence:

<table>
<thead>
<tr>
<th>LOOP</th>
<th>CLRA</th>
<th>0—A</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAP</td>
<td>0—ITMP</td>
<td>DITTO</td>
</tr>
<tr>
<td>TAP</td>
<td>DITTO</td>
<td></td>
</tr>
<tr>
<td>SEI</td>
<td>1—1</td>
<td>--the interrupt would occur here but the SEI instruction set the I-bit</td>
</tr>
<tr>
<td>BRA</td>
<td>LOOP</td>
<td>DO IT AGAIN</td>
</tr>
</tbody>
</table>

5.4.1.3 THE RTI INSTRUCTION. The Return from Interrupt (RTI) instruction can clear or set the I-bit depending upon the value of the I-bit position (bit 4) of the stacked Condition Code Register. This is normally the value of the I-bit prior to the interrupt sequence. If the stacked I-bit is clear and a maskable interrupt is pending, the processor will immediately begin another interrupt sequence after the RTI instruction has been completed. If the I-bit is cleared when the Condition Code Register is restored, the one E-cycle delay is absorbed by the remaining cycles of the instruction.

5.4.1.4 THE SWI INSTRUCTION. The Software Instruction (SWI) instruction is used primarily as a mechanism for writing the machine state to memory on demand from software. It is also the MC6801 instruction requiring the longest execution time: 12 MPU E-cycles.

A Software Interrupt instruction will always be executed to completion before any other interrupt is recognized. The instruction is "complete" when the SWI vector has been placed into the Program Counter. If an NMI is pending after completing the SWI instruction, the processor will immediately begin a second interrupt sequence. If a masked interrupt becomes pending during the execution of an SWI instruction, however, it will remain pending due to the I-bit being set as part of the instruction.

While it may appear from Figure 5-8 that the SWI instruction has the highest priority, this appearance is deceiving. The SWI instruction is mutually exclusive from all of the other interrupts. Priority does apply, however, if considering the order in which service routines are executed if all possible interrupts (except Reset) became pending while executing an SWI instruction. Under such conditions, (1) after fetching the SWI vector, the machine state is stacked again for the NMI, (2) the NMI service routine is executed, (3) the SWI service routine is executed, and (4) finally, the service routines for the maskable interrupts are executed in the prioritized order. The setting of the I-bit during execution of the SWI instruction results in masking all other interrupts except NMI.

5.4.2 The Wait for Interrupt (WAI) Instruction

In response to the Wait for Interrupt (WAI) instruction, the MC6801 stacks the machine state and then enters a "Wait" state, as shown in Figure 5-15. Normal processing resumes only after receipt of an unmasked interrupt request or by Reset. By stacking the machine state before the interrupt, the interrupt response time latency is reduced by the stacking time. Assuming that the machine state
Figure 5-15. WAI Instruction Sequence
has been stacked, the MC6801 requires only five or six E-cycles to fetch the first opcode in the service routine after receipt of an unmasked interrupt. This depends upon whether the interrupt is NMI or IRQ2 (five cycles) or IRQ1 (six cycles). This time should be compared with at least 13 E-cycles when the WAI instruction is not used.

While the MPU is in the "Wait" state, its bus state will appear as a series of MPU reads of an address which is seven locations less than the original contents of the Stack Pointer. Contrary to the MC6800, none of the ports are driven to the high impedance state by a WAI instruction.

A possible problem involving the WAI instruction must be considered by the system designer. Remember, the CLI instruction results in the I-bit being cleared during the MPU E-cycle following completion of the instruction. A specific instruction sequence which requires this delay is:

```
CLI
WAI
= INTERRUPT =
```

If the MPU did not delay clearing the I-bit for one E-cycle, it would be possible for the interrupt to become pending prior to execution of the CLI instruction, and then be serviced between the CLI and WAI instructions. The program would then wait for an interrupt it had already serviced.

It is still possible, however, for this situation to occur even with the delay. This would happen if and only if

1. an NMI is serviced between the CLI and WAI instructions, and
2. the maskable interrupt intended to terminate the WAI instruction becomes pending (or is serviced) during execution of the NMI service routine.

If this situation is unavoidable, then the following solution is offered for consideration. Recall that the critical element in this problem is that a maskable interrupt (not NMI) can return to a WAI instruction after restoring the machine state. If a sequence is incorporated in the maskable interrupt service routine (again, not NMI) to prevent returning to a WAI instruction, then the problem can be circumvented. The routine shown in Figure 5-16 performs this function.

### 5.5 PROVIDING INTERRUPT SERVICE

Consider an MC6801 system that has three interrupt capable devices, all of which are connected to the IRQ1 line. While there are three devices on the IRQ1 line, remember there is but one IRQ1 vector. The function of determining which of the three devices is requesting service and then vectoring to the corresponding service routine can be performed by a "who-done-it" software routine. This routine is normally a polling sequence which inspects the interrupt request flag of each of the candidate devices, as shown in Figure 5-17. The polling order reflects the desired priority of service.

While the "who-done-it" routine is simple to implement, the time required to execute it adds directly to the interrupt response time latency. If several devices are connected to the interrupt line and the polling sequence becomes too lengthy, device vectored interrupts can be obtained in certain MC6801 operating modes with the use of additional external hardware. The Programmable Timer utilizes prioritized hardware vectored interrupts which require no "who-done-it" routine.
Figure 5-16. A Routine to Skip the WAI Instruction
**Figure 5-17. A “Who-Done-It” Routine**

External hardware designed to provide prioritized vectored interrupts generally employs the following elements:

1. a decoder to trap $FFF8 or $FFF9 (IRQ1) on the address bus,
2. a priority encoder circuit to map the current interrupts into a prioritized output,
3. circuitry to map the output of the priority encoder into the address in a memory element, and
4. a memory element to furnish the two bytes of the vector.

Hardware for performing this service can be used only in the expanded multiplexed modes of the MC6801: Modes 1, 2, 3 and 6. An example of an 8-level priority interrupt encoder is included in Chapter 8.

### 5.6 PROGRAM RESTARTABILITY

The MC6801 interrupt sequence provides for (1) saving the machine state, (2) vectoring to a service routine, (3) restoring the machine state, and (4) resuming the former routine. This procedure cannot, however, guarantee that all routines can be interrupted and subsequently restarted with equivalent results.

Some routines logically cannot be interrupted and restarted with equivalent results. Consider a program which controls a peripheral device with a time-dependent sequence such as a timing loop. If the routine must perform an operation precisely at the end of a software timing loop, an interrupt would alter the elapsed time. Routines can receive immunity from maskable interrupts (but not NMI) during time critical sequences by preceding them with an SEI instruction and following them with a CLI instruction.

### 5.6.1 Re-Entrant Routines

Another instance in which an MC6801 program cannot be successfully restarted after an interrupt is one which has its state variables altered by the interrupt routine. Routines which can be interrupted and subsequently re-entered (called again) must be written such that they are “re-entrant”. Programs which are re-entrants use only registers or stack as working storage; routines which are not re-entrant utilize local temporary memory locations as working storage.
As an example, consider a subroutine which converts binary data to hexadecimal. Suppose that this subroutine was executing in background (non-interrupted state) when the processor received an interrupt. During servicing of the interrupt, a binary to hexadecimal conversion is needed and the same subroutine was entered a second time. After execution resumes in the background program, the validity of the result obtained by the background caller depends solely on how the subroutine was written. If the subroutine used only registers and stack, both callers received a correct result. If temporary variables with fixed locations were used, however, the background routine most likely received an incorrect result because execution of the interrupt service routine resulted in altering the fixed memory locations.

Routines written to be re-entrant generally make extensive use of the stack. After pushing the appropriate values onto the stack, the stack pointer can be transferred to the Index Register using the TSX instruction. The indexed mode of addressing can then be used to manipulate values on the stack. An example of a re-entrant routine for the MC6801 is shown in Figure 5-18.

5.6.2 Resource Sharing

Another common difficulty in implementing interrupt-driven programs is that of resource sharing. For purposes of this discussion, a resource is anything that is used by more than one routine. This can include a memory location, peripheral device, or register.

By way of introduction into resource sharing, a short example is given. Suppose that two bits (bit 0 and bit 7) in the Port 1 Data Register are used to generate output waveforms. Further assume bit 0 is toggled by a routine running in background (non-interrupt driven) while bit 7 is toggled in response to some interrupt. Instruction fragments for these two operations are as follows:

```
LDAA #1 BACKGROUND
EORA PIDATA TOGGLE BIT 0
STAA PIDATA

INTRPT LDAA #$80 INTERRUPT
EORA PIDATA TOGGLE BIT 7
STAA PIDATA
RTI
```

The problem with this seemingly trivial example is that it contains a programming error which can result in an erroneous signal in the bit 7 output. The error will always occur if the interrupt is serviced between the EORA and STAA instructions of the background routine:

```
LDAA #1 BACKGROUND
EORA PIDATA TOGGLE BIT 7
STAA PIDATA
```

—interrupt service
When the interrupt occurs, ACCA contains a copy of the Port 1 Data Register with bit 0 toggled which is then pushed onto the stack during the interrupt sequence. In the meantime, the interrupt service routine obtains a copy of the Port 1 Data Register, toggles bit 7, overwrites the Data Register with this value, and returns to the background routine. The former contents of ACCA is restored during the RTI instruction and the background routine then resumes by overwriting the Data Register with the former copy before bit 7 was toggled. Hence, bit 7 is toggled by the interrupt service routine and then toggled back again by the background routine.

To correct this error, it is important to realize its cause: the inability to obtain exclusive access to a shared resource (the Port 1 Data Register) while modifying it. Because this is necessary to prevent the error, a scheme must be implemented whereby other routines sharing the resource are "locked out" while the change is taking place.

For example, if the INTRPT routine is invoked by a maskable interrupt, the I-bit could be set to prevent an interrupt and thus "lock out" the interrupt routine between the EORA and STAA instructions. The background portion of the fragment would then appear as follows:

```
LDAA #1    BACKGROUND
SEI        LOCK OUT INTRPT
EORA PIDATA TOGGLE BIT 7
STAA PIDATA
CLI        TAKE LOCK OFF
```

A complete discussion of this important topic is beyond the scope of this text but can typically be found in those books which treat the design of operating systems. For a more thorough treatment, the reader is referred to an appropriate textbook*.

*Two books are recommended. The first book uses PASCAL extensively while the second does not.


/* M U L 1 6 -- THIS ROUTINE MULTIPLIES THE CONTENTS OF A:B BY 16 BITS POINTED TO BY THE X-REGISTER AND PUTS THE RESULT INTO A:B. THE X-REGISTER IS SAVED. */

/* THE ROUTINE IS RE-ENTRANT */

org $400
mul16

; SAVE X-REGISTER ON STACK
pshx

; PUSH MULTIPLICAND ONTO STACK
pshb

; LOAD UP THE MULTIPLIER
ldx x

; PUSH ONTO THE STACK
pshx

; USE 16 AS THE COUNT
ldaa #16

; USE X TO POINT AT STACK
tsx

; THE STACK NOW LOOKS LIKE THIS:

; +0 COUNT

; +1 MS BYTE MULTIPLICAND

; +2 LS BYTE

; +3 MS BYTE MULTIPLIER

; +4 LS BYTE

; +5 MS BYTE X-REGISTER

; +6 LS BYTE

; +7 MS BYTE RETURN ADDRESS

; +8 LS BYTE

; FORM THE RESULT
ldaa 3,x

; SHIFT MULTIPLICAND
asl 2,x

; BCC MUL004
bpl mul002

; ADD MULTIPLIER
add 3,x

; BNE MUL002
bne mul004

; BUMP STACK POINTER
ins

; BUMP IT TWICE
pulx

; BUMP IT TWICE
pulx

; ONCE MORE TO RESTORE X
pulx

; END
rts

/* CLEAN UP THE STACK */

/* TOTAL ERRORS 00000---00000 */

Figure 5-18. Example of Re-Entrant Programming
CHAPTER 6
SERIAL COMMUNICATIONS INTERFACE (SCI)

6.0 INTRODUCTION

The Serial Communications Interface (SCI) allows the MC6801 to be efficiently interfaced with a variety of devices which require an asynchronous serial data format. The SCI provides two serial data formats at a variety of bit rates and can be clocked by either an internal bit rate generator which is dependent on the MPU E-cycle or by an independent external clock. Three pins of Port 2 provide the SCI Interface with external devices and it is functionally independent of the operating mode.

The SCI provides a great deal of flexibility which is obtained by software programmability. The following six features of the SCI are optional using software control:

- format: standard mark/space (NRZ) or Bi-phase,
- serial clock source: internal or external,
- baud rate: one of four for a given MCU E-clock frequency or one eighth of an external clock rate,
- wake-up feature: enabled or disabled,
- interrupt requests: enabled or masked individually for transmitter and receiver, and
- clock output: internal bit rate clock enabled or disabled to Port 2 bit 2.

6.1 SERIAL COMMUNICATIONS INTERFACE REGISTERS

The programmer interface with the SCI is provided by the four addressable registers shown in Figure 6-1. These registers include:

- a Rate and Mode Control Register,
- a Transmit/Receive Control and Status Register,
- a Transmit Data Register, and
- a Receive Data Register.

To interface with external devices, the SCI uses Port 2 bit 3 for serial input and bit 4 for serial output. Port 2 bit 2 is also used by the SCI if either the internal-clock-out or external-clock-in options are selected. The SCI internally configures the appropriate bits in Port 2 Data Direction Register according to the options selected. A block diagram of the SCI is shown in Figure 6-2.
6.1.1 Rate and Mode Control Register

The Rate and Mode Control Register (RMCR) is a four-bit write-only register which can be used to control the following serial data characteristics:

- bit rate,
- format,
- clocking source, and
- configuration of Port 2 bit 2.
Figure 6-2. Block Diagram of SCI
The four bits of the Rate and Mode Control Register can be considered as a pair of two-bit fields. The least two significant bits determine the bit rate when using the internal clock and the remaining two bits control the format (NRZ or Bi-Phase) and clock source (internal or external). The register is depicted in Figure 6-3 and individual bits are defined as follows:

![Figure 6-3. Rate and Mode Control Register](image)

Bit 1: Bit 0 SS1:SS0 Speed Select — These two bits select the Baud when using the internal clock. Four rates which can be selected are a function of the MCU E-cycle frequency. Figure 6-4 lists bit times and rates for three selected MCU input frequencies. If the MCU is driven by other frequencies, the appropriate divisor must be used to determine the bit rate. If external clocking is selected (CC1 = CC0 = 1), then speed select bits are ignored.

![Figure 6-4. Selected Internal Bit Times and Rates](image)

Bit 3: Bit 2 CC1:CC0 Clock Control and Format Select — These two bits control the serial clocking source and format. If CC1 is set, the DDR value for Port 2 bit 2 (P22) is forced to the complement of CC0 and cannot be changed until CC1 is cleared. If CC1 is cleared after having been set, its DDR value remains unchanged. Figure 6-5 defines the formats, clock source, and use of P22.

![Figure 6-5. Format and Clock Source Control](image)
If CC1:CC0 = 10, the internal serial bit rate clock is provided at P22 regardless of the values for the Transmit Enable (TE) and Receive Enable (RE) bits. The Rate and Mode Control Register is cleared by Reset which configures the clocking, format, and bit rate as: internal clock, and Bi-phase format at the highest rate.

6.1.2 Transmit/Receive Control and Status Register

The eight bits of the Transmit/Receive Control and Status (TRCS) Register provide control for several SCI options and reveal information regarding the state of current serial operations. The five control bits of the register enable and disable the following SCI functions:

- serial transmitter and receiver (separately),
- transmitter and/or receiver interrupts (separately), and,
- the wake-up feature.

The remaining three status bits provide a means for determining the state of serial operations and are set when:

- the Transmitter Data Register is empty,
- the Receiver Data Register is full, and
- a Receiver overrun or framing error occurs.

All eight bits of the register can be read but only the five least significant bits can be written. During Reset, the register is preset to $20$ thereby clearing all of the bits except TDRE which is set. The Transmit/Receive Control and Status Register is depicted in Figure 6-6 where individual bits are defined as follows:

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDRF</td>
<td>ORFE</td>
<td>TDRE</td>
<td>RIE</td>
<td>RE</td>
<td>TIE</td>
<td>TE</td>
<td>WU</td>
</tr>
</tbody>
</table>

Figure 6-6. Transmit/Receive Control and Status Register

**Bit 0 WU**  
"Wake-up" on Idle Line — When set, WU enables the wake-up function. It is cleared by the SCI after receipt of ten consecutive 1's or by Reset. WU will not remain set if the line is idle.*

**Bit 1 TE**  
Transmit Enable — If set, the DDR value for Port 2 bit 4 is set, cannot be cleared while TE is set, and will remain set if TE is subsequently cleared. A preamble of nine consecutive 1's is produced when TE is changed from clear to set. While TE is set, the transmitter output is coupled to Port 2 bit 4. TE is cleared by Reset.

**Bit 2 TIE**  
Transmit Interrupt Enable — If set, an IRQ2 interrupt is enabled when TDRE is set; when clear, the interrupt is inhibited. TIE is cleared by Reset.

**Bit 3 RE**  
Receive Enable — If set, the DDR value for Port 2 bit 3 is cleared, cannot be set while RE is set, and will remain clear if RE is cleared. While RE is set, the SCI receiver is enabled. RE is cleared by Reset.

**Bit 4 RIE**  
Receive Interrupt Enable — If set, an IRQ2 interrupt is enabled when RDRF and/or ORFE is set; when clear, the interrupt is inhibited. RIE is cleared by Reset.

*WU will be cleared by the next bit rate clock transition if the line is idle.
Bit 5 TDRE  Transmit Data Register Empty — TDRE is set when the Transmit Data Register is transferred to the output serial shift register and by Reset. If clear, it indicates that transfer has not yet taken place and MPU writes to the Transmit Data Register will overwrite the last value. Byte transfer is synchronized with the bit rate clock. TDRE is cleared by reading the TRCS Register (with TDRE set), then writing a new byte to the Transmit Data Register. Data will not be transmitted if it is written to the Transmit Data Register without previously reading the TRCS Register.

Bit 6 ORFE  Overrun-Framing Error — If set, ORFE indicates either an overrun or framing error. An overrun occurs when the next byte is ready for transfer to the Receiver Data Register and the RDRF flag bit is set. A receiver framing error results when a stop bit (1) is not found in the tenth bit time. An overrun can be distinguished from a framing error by noting the value of RDRF. If RDRF = ORFE = 1, then an overrun has occurred; if RDRF = 0 and ORFE = 1, a framing error has been detected. Data transfer is inhibited during an overrun. For a framing error, the misframed byte is transferred but RDRF is not set. The ORFE bit is cleared by reading the TRCS Register, then the Receive Data Register, or by Reset.

Bit 7 RDRF  Receive Data Register Full — RDRF is set when the input serial shift register is transferred to the Receive Data Register. It is cleared by reading the TRCS Register, then reading the Receive Data Register, or by Reset.

6.1.3 Transmit and Receive Data Registers

Two 8-bit registers are used for the next byte to be transmitted and the last received byte. Providing TDRE is clear, the write-only Transmit Data Register is transferred to the transmit shift register after the current byte has been transmitted. The transfer is synchronized with the bit rate clock.

The Receive Data Register is a read-only 8-bit register and contains the last byte received. RDRF is set to indicate when a byte has been transferred from the receive shift register to the Receive Data Register.

6.2 SCI CLOCKING OPTIONS

There are several options associated with SCI clocking functions. An internal clock can be used to provide a serial bit rate as a function of the MCU E-cycle. As another option, this clock can be coupled to an output pin and used external to the MCU. Finally, an external clock can be used to drive the SCI. Regardless of clock option, however, both the receiver and transmitter operate at the same bit rate and format.

6.2.1 Using the Internal SCI Clock

The source of the internal clock is the Programmable Timer free-running counter which is driven by the MCU E-cycle clock. The SS1:SS0 bits in the Rate and Mode Control Register define one of four possible divisions of the MCU E-clock for four separate SCI bit rates. While a wide range of MCU E-clock frequencies can be used, a judicious choice of frequency must be made if a particular baud rate is desired. Figure 6-4 provides the Baud rates achievable with four different frequencies driving the MCU.
NOTE
The MCU Programmable Timer is the source of the internal bit rate clock. MPU writes to the free running counter should be avoided if the SCI is being used with the internal clock option.

6.2.2 Using an External SCI Clock
An external clock can be used with the SCI to obtain a particular Baud rate which is independent of the MCU internal E-cycle. If an external clock is used with the SCI, the following requirements are applicable:
1. the CC1:CC0 field in the Rate and Mode Control Register must be set to 11,
2. the input clock frequency must be eight times (8X) desired bit rate,
3. the frequency must not exceed that of the MCU E-clock with a 50% duty cycle (±10%), and
4. only NRZ format can be used.

6.2.3 Providing a Serial Output Clock
The SCI serial bit rate clock can be obtained as an output by suitable programming of the Rate and Mode Control Register. If the CC1:CC0 bit field of the Rate and Mode Control Register is written to 10, the SCI internal bit rate clock will be coupled to P22. If this option is selected, the DDR value for this bit is internally set by the SCI and cannot be changed. The output bit rate clock will be provided at P22 regardless of the values for either TE or RE.

The bit rate clock output can be used for purposes other than serial clocking. It is a free running clock source that requires no software service. The clock has a 50% duty cycle at the frequency determined by both the MCU E-cycle frequency and the SS1:SS0 field of the RMCR. A positive edge occurs at the mid-bit time during each bit interval as shown in Figure 6-7.

6.2.4 Clocking Multiple SCIs
If it is desired to drive several SCIs with a common clock, it is recommended that a separate external clock be used. One possible clock source is the E-clock output from the MCU being driven at the lowest frequency. This choice provides compliance with the restriction that the frequency of the external serial clock cannot exceed any MCU E-cycle.

It is difficult to clock another MC6801 SCI with the serial bit clock output due to the different clock rates involved. The serial bit clock output is at the desired bit rate whereas the serial input clock requirement is eight times (8X) the desired bit rate.
Simultaneous serial transmit and receive (full duplex) capability is provided using either of two software selectable formats:

- industry standard mark/space (NRZ) which is used typically with terminals or modems, and
- self-clocking bi-phase which is intended primarily for use between processors.

Both formats begin with a single "start" bit (always 0) and end with a "stop" bit (always 1). The standard mark/space (NRZ) format produces a level corresponding to the bit value during each bit time. The level is sampled by the receiver at the middle of each bit interval. This format can tolerate a theoretical range of $3.75 \times 10^{-6}$ mismatch between the transmitter and receiver clock rates for correct operation.

The second format, Bi-phase, is included to enable processor-to-processor communications with a much higher tolerance clocking mismatch. Bi-phase format — also called Bi-Phase-M, FM, F/2F, and Manchester — requires a transition (in either direction) at every bit time. An additional transition at the half bit time is required whenever the bit value is a "1". This format can tolerate a theoretical range of 25% difference in the transmitter and receiver clock frequency and/or phase. Both formats are illustrated in Figure 6-7 for comparison. Note that an idle line in NRZ format is represented by a constant mark (1) on the line whereas an idle line in Bi-phase will toggle at every half-bit time.

It should be apparent from this description that no SCI serial format includes a parity bit or provides more than one stop bit. If a parity bit is desired, it can be generated using software before the byte is written to the Transmit Data Register. Similarly, parity can be checked with software during data reception.

Figure 6-7. SCI Serial Data Formats
6.4 SERIAL COMMUNICATION OPERATIONS

The SCI must be initialized prior to operation by a sequence which
- selects the clock source and format in the Rate and Mode Control Register and then
- writes the desired operational control bits to the Transmit/Receive Control and Status Register.

6.4.1 Transmitter Operation

Serial transmitter operation is enabled by the TE bit in the Transmit/Receive Control and Status Register. When set, this bit couples the transmit shift register to P24 and activates transmitter operation. Transmission begins with a preamble consisting of nine 1’s after which internal synchronization is established and the transmitter is ready for operation. At this point, one of two situations exists:
1. if the Transmit Data Register is empty (TDRE = 1), a continuous string of ones will be transmitted indicating an idle line, or
2. if data has been stored into the Transmit Data Register (TDRE = 0), the byte is transferred to the transmit shift register and transmission will begin.

The Transmit Data Register is not transferred to the transmit shift register until the next bit time after it has been emptied. TDRE is then set and transmission of the byte begins. The Start bit, eight data bits (beginning with the least significant bit) and finally the Stop bit are transmitted. If TDRE is still set when the next transfer should occur, 1’s are transmitted until more data is provided.

NOTE
If a byte is written to the Transmit Data Register without previously reading the Transmit/Receive Control and Status Register, the TDRE flag will not be cleared and data will not be transferred to the transmit shift register.

6.4.2 Receiver Operation

The SCI receiver is enabled by the RE bit in the Transmit/Receive Control and Status Register. In NRZ format, the receiver is ready to accept data after the line has been idle for 1/4 bit time. In Bi-phase format it is necessary to present an idle line (toggling at half-bit times) to the receiver for at least one bit time.

The receiver bit interval is divided into eight sub-intervals for internal operation. In NRZ format, the received bit stream is synchronized with the leading edge of the first 0 (space) encountered. The serial input is then sampled at the approximate center of each bit time interval for ten consecutive bits. Providing the tenth bit is a “1” and RDRF is clear, the byte is transferred to the Receiver Data Register and RDRF is set. This sequence is the normal receiver response.

If the tenth bit is not a 1 (i.e., a Stop bit), a framing error is assumed and ORFE is set, reflecting this condition.* The byte with the framing error is transferred to the Receive Data Register but RDRF is

*In NRZ format, holding the receiver line low (e.g., with BREAK key), and clearing ORFE, results in ORFE being set after receipt of ten 0’s.
not set. If the next serial bit is also a 0, the SCI will treat it as a new Start bit. A framing error is cleared by reading the TRCS Register followed by reading the Receive Data Register. Application programs should always allow for the possibility of a framing error because RDRF will not be set until the framing error is cleared.

If the Stop bit is present, but the receiver has not been serviced (RDRF is still set) ORFE will be set, indicating that an overrun has occurred. Data transfer from the receive shift register to the Receive Data Register is inhibited in an overrun condition. An overrun is cleared by reading the TRCS Register followed by reading the Receive Data Register.

There is no difference in programming procedures for NRZ and Bi-phase formats. However, internal processing is somewhat different. In Bi-phase format, sub-interval sampling is initiated by a level transition. The receiver must then determine whether the transition interval is more or less than six of eight sub-intervals. Short intervals are defined as 1’s, while long intervals are defined as 0’s. This sequence is repeated after receipt of the next transition.

### 6.5 THE WAKE-UP FEATURE

The Wake-up feature is intended to provide a means for coping with a somewhat specialized problem. Attributes of the target application include the following:

- three or more processors are communicating via a common serial line,
- communications are “message” oriented where a message can be transmitted with no significant idle line time within the interior of the message,
- the “address” of the recipient is included at the beginning of the message, and
- it is probable that typical messages are not of interest to all processors on the common line.

Given this scenario, non-interested parties are required to respond to every byte of all messages. The Wake-up feature is provided to allow non-interested MCUs to disregard the remainder of a message if serial communications are structured in accordance with the above conditions. If the Wake-up feature is not used, it may be ignored without consequence. The Wake-up bit, however, should not be set unintentionally.

In the target multi-processor application, the message contains the identification of the addressee in its initial bytes. In order to ignore the remainder of the message, the Wake-up bit (WU) must be set to inhibit all further flag processing until the beginning of the next message. An SCI in Wake-up (e.g., WU is set) is reenabled (WU is cleared) by a string of ten consecutive 1’s which necessarily must also be used as a message delimiter. The ten consecutive 1’s can be generated only by an idle line which ensures that no data byte can accidently reenable an SCI. Software must provide for the idle period between any two consecutive messages. In addition, no ten-bit idle period can be allowed to occur within a message in order not to disturb message synchronization.

An SCI “listener” desiring to invoke the Wake-up feature is required to set the WU control bit in the Transmit/Receive Control and Status Register while the message is in the process of being transmitted. When WU is set, the SCI effectively ignores the remainder of the message. Upon receipt of ten consecutive 1’s, the SCI clears WU and resumes normal receiver operation.

*If a valid character is overrun by a misframed character, RDRF and ORFE are set (overrun) with the valid character stored in the Receive Data Register. If a framing error occurs followed by one or more overrunning characters, the result is a framing error (RDRF clear, ORFE set) with the first (misframed) character stored in Receive Data Register.
6.5.1 Transmitter Duties During Wake-Up Operation

Software controlling the transmitter during Wake-up operation is required to meet certain obligations. A “message” consists of a character string transmitted such that the serial line does not go idle within the message for a period sufficient to enable any receiver. The Wake-up (WU) bit is cleared by the SCI when its receiver detects a string of ten consecutive ones.* Ones included in the eight data bits, the stop bit, and the idle line count towards the accumulation of ten consecutive ones.

Software servicing the transmitter normally stores the next byte in the Transmit Data Register as the response to TDRE being set. The line will be kept busy shifting out the current byte for 10 bit times (1 Start bit, 8 data bits, 1 Stop bit) then the line will go idle if the transmitter has not again been serviced.

If the last data bit transmitted was all ones ($FF$), transmitter software has a total of 10 bit times to respond to the TDRE flag before a receiver “sees” ten consecutive ones. If the most significant bit of the data was a zero, however, the transmitter software has a total of 18 bit times to respond to the TDRE flag. These two cases, 10 and 18, represent minimum and maximum transmitter software response times to avoid prematurely clearing a receiver Wake-up bit.

After the last character of a given message has been transmitted, the line must remain idle for at least a 10-bit delay. This delay, when measured from when TDRE is set, is data dependent because both the data bits and the Stop bit are included in the count. The TDRE flag that is used to signal the start of the delay is the one which appears after the last byte of the message is written to the Transmit Data Register. This corresponds with transfer of the last byte of the message to the transmit shift register.

If the last byte is all ones, the transmitter must wait a total of 11 bit times (from when TDRE is set for the final time) before beginning the next message (1 Start bit, 8 data bits, 1 Stop bit, 1 Idle bit). The maximum waiting time is required if the most significant bit of the last byte transmitted was a zero. With this condition, the transmitter must delay a total of 19 bit times before beginning the next message to be certain that all receivers have been enabled (1 Start bit, 8 data bits, 1 Stop bit, and 9 Idle bits).

In writing software for use with the Wake-up feature, it can be convenient to have a table of bit times at the various baud rates. The bit times could then be converted into the number of MCU E-cycles necessary to obtain the minimum delay between messages. If the internal clock is used, however, it is much more convenient to program timeout loops with respect to the number of MCU E-cycles per bit rather than bit times. It should be realized that the number of MCU E-cycles per bit is a constant for a given value of the Speed Select bits (SS1:SS0) regardless of the frequency of the crystal or external clock driving the MCU. The number of MCU E-cycles per bit using the internal bit rate clock is as follows:

<table>
<thead>
<tr>
<th>SS1:SS0</th>
<th>MCU E-cycles Per Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>16</td>
</tr>
<tr>
<td>01</td>
<td>128</td>
</tr>
<tr>
<td>10</td>
<td>1024</td>
</tr>
<tr>
<td>11</td>
<td>4096</td>
</tr>
</tbody>
</table>

*To be precise, WU is cleared when the Start bit, serial shift register, and Stop bit flip-flops all contain 1's. There is no counter dedicated to this function.
Finally, remember the above discussion assumed that both the transmitter and receiver bit rate clocks were synchronized. For a worst case, one must assume that both clocks are at opposite extremes of the acceptable clock skew. This possible skew requires the programmer to be conservative with respect to timing when designing software which utilizes the Wake-up feature.

6.5.2 Receiver Duties During Wake-Up Operation

Software using the Wake-up feature will invoke it after determining that it is not an addressee for the current message. Receiver software can then effectively ignore the remainder of the message by clearing RDRF and setting WU. With the WU bit set, RDRF will no longer be affected until the line goes idle for ten consecutive bit times. Providing that RDRF is clear when wake-up is invoked, the ORFE bit will also be inhibited by wake-up; otherwise, it will be set upon receipt of the next byte. When the receiver detects ten consecutive ones, the SCI will clear WU and upon receipt of the next byte, the RDRF flag bit will be set.

6.6 PROVIDING SCI INTERRUPT SERVICE

If the SCI is interrupt-driven, there are several factors which should be taken into account. First, it should be noted that there are two interrupt enable bits in the TRCS register which control three types of receiver interrupts and one transmitter interrupt. There is, however, only one interrupt vector associated with all SCI interrupts*.

In the SCI interrupt service routine, it is usually necessary to establish a software mechanism to ensure "fair" service of both transmitter and receiver interrupts. It is usually undesirable to service only one of the interrupts using the same order of preference. This could result in no response to the other flag if the preferred interrupt occurs often enough. It is generally preferable to either alternate the order or provide service to both transmitter and receiver each time.

The SCI receiver has three states which can invoke an interrupt if the Receiver Interrupt Enable (RIE) bit is set and the MCU I-bit is clear. These states are determined by the values of RDRF and ORFE and are as follows:

- RDRF alone — normal receipt of the next byte,
- ORFE alone — a framing error, and
- both RDRF and ORFE — an overrun.

All three conditions are cleared (reset) by the same sequence: an MPU read of the TRCS register followed by a read of the Receiver Data Register.

While the transmitter has only one associated interrupt condition (TDRE set), it must be treated somewhat differently than receiver interrupts. When no data is being transmitted and none is available, the state of the TDRE bit is a "1". If the Transmitter Interrupt Enable bit (TIE) is also set, an interrupt would occur which requires no service. Therefore, the TIE bit should be set only when there is more data to transmit and should be cleared after the last byte of the message has been written to the Transmit Data Register.

*It is also possible to have no SCI interrupt enabled and be vectored to $FFF0:$FFFI under certain circumstances. See Section 5.3.1.
6.7 TWO SCI SOFTWARE EXAMPLES

Two examples are provided, in Figure 6-8 and 6-10 which illustrates functions performed by the SCI. The examples have been chosen such that they involve only one MCU and only one teleprinter. Due to this self-imposed constraint, they may seem trivial; however, the purpose of these two examples is to demonstrate software procedures and for this limited purpose it is hoped they are adequate. In both examples, the output of the SCI is jumpered to the serial input and, in essence, the MCU is talking to itself. If LILbug (TM)* is used in these examples, an external MC6850 ACIA terminal interface must be employed because the SCI is dedicated to the two examples.

6.7.1 Exercising the Serial Communications Interface

The SERIAL program, illustrated in Figure 6-8, can be used to exercise all SCI functions except for the Wake-up feature. It is assumed that the SCI transmitter is jumpered to its receiver.

The program transmits a byte, waits for it to reach the receiver, and then prints both the transmitted and received values as the most and least significant bytes of a 16-bit hexadecimal value. The program internally compares both transmitted and received values and increments an error counter if a discrepancy is found. An example of the output from the program is shown in Figure 6-9.

The serial rates, formats, and clocking options are controlled by the Rate and Mode Control Register. Two program variables allow altering the contents of this 4-bit register. The variable, MODE, indicates the starting values for the CC1:CC0 and SS1:SS0 fields. The value for CC1 is never changed. A 3-bit variable called MASK controls whether the CC0, SS1, and SS0 field are changed after each set of 256 bytes. A description in the program header indicates how these bits are affected by the MASK byte.

The program can be used to drive the SCI through all four internal baud rates. No difference in the speed of the printer will be observed, however, unless the output baud rate is higher than the SCI. This is more readily observed with a CRT operating at 9600 baud. The baud rate of the terminal has no affect on the logical operation of the program.

6.7.2 Demonstrating the Wake-Up Feature

The OKBAD program demonstrates the SCI Wake-up feature and assumes that the transmitter is connected to the receiver. The program is shown in Figure 6-10 and performs the following functions:

1. the SCI output routine always transmits the character string, “OK! ZBAD”,
2. the receiver routine writes all that it “sees” (except “Z”) to an internal buffer,
3. upon receiving a “Z”, the receiver checks a program flag byte (WAKEUP) and, if clear, sets the SCI Wake-up bit (WU).

After transmitting the last character and delaying until Wake-up has been cleared, the contents of the receiver buffer are written to the output printer. Buffering the receiver input makes the program independent from the baud rate of the console printer. Only one of two messages will appear in the

---

* LILbug (TM) is a monitor programmed in the ROM of the MC6801L1. See the LILbug Manual for details.
receiver buffer. If WAKEUP was clear, then only “OK!” should be received because the Wake-up feature inhibited the RDRF flag while “BAD” was being transmitted. If WAKEUP was not clear, then the entire message will appear in the buffer as “OK! BAD”. An example of the output from the OKBAD program is shown in Figure 6-11.

The character, “Z”, is used in this example as a control byte to indicate when to check the internal flag byte. The character is not written to the receiver input buffer.

The variable, MODE, controls the value used in configuring the Rate and Mode Control Register. A table containing loop index values corresponding to a timeout of 19 bit times is used to provide an idle time sufficient to reenable the receiver between messages. The b1:b0 field (SS1:SS0) of MODE indicates which value in the table is to be used for the timeout. Because the SS1:SS0 field is unused with an external clock, this program will not function correctly, if bit 3 in MODE (CC1) is a “1”.

6-14
Figure 6-8. Exercising the SCI: SERIAL
**EQUATES**

- TRCS EQU $0011 TRANSMIT/RECEIVE CONTROL & STATUS
- TX EQU $0013 TRANSMIT DATA REGISTER
- RX EQU $0012 RECEIVE DATA REGISTER
- RMCR EQU $0010 RATE & MODE CONTROL REGISTER
- OUT4HS EQU $F815 LILBUG OUTPUT 4 HEX & SPACE
- CRLF EQU $F818 LILBUG CR AND LF
- F815 A OUT4HS EQU $F815 LILBUG OUTPUT 4 HEX & SPACE
- F818 A CRLF EQU $F818 LILBUG CR AND LF
- F80F A PDATA EQU $F80F LILBUG STRING PRINT

**STORAGE CELLS**

- ORG $1000
- JMP BEGIN A VECTOR TO START-UP

**MODE, MASK, & KT ARE USER VARIABLES**

**FORMATTED PRINT LINE**

- HDR0 FCC /RMCR:0/ HEADER FOR RATE & MODE REGISTER
- RM EMB 1 WORKING COUNTER FOR KT
- CHARTX RMB 1 TRANSMITTED BYTE
- CHARRX RMB 1 RECEIVED BYTE
- ERRKT RMB 2 ERROR COUNTER

**MAIN LINE BEGINS HERE**

**Figure 6-8. Exercising the SCI: SERIAL (Continued)**
0009 * PRINT THE HEADER
0010 * FIX THE RATE & MODE CONTROL REGISTER
0011 * FOR NEXT PASS.

00113A 1043 7F 0011 A START CLR TRCS TURN OFF RE & TE
00114A 1046 B6 1003 A LDAA MODE GET THE MODE
00115A 1049 84 0F A ANDA #$F CLEAR OUT DON'T CARES
00116A 104B 85 08 A BITA #$8 SEE IF BIT 3 IS ON
00117A 104D 27 08 1057 BEQ OK NOT ON
00118A 104F 6D 1004 A LDAB MASK INSURE MASK <= 3
00119A 1052 C4 03 A ANDB #$3
00120A 1054 F7 1004 A STAB MASK

00122A 1057 16 OK TAB MAKE 2ND COPY
00123A 1058 97 10 A STAA RMCR STASH IN RMCR
00124A 105A 8B 30 A ADDA #$30 CONVERT LOWER FOUR TO ASCII
00125A 105C 81 39 A CMPA #$39 MAKE SURE NUMERIC
00126A 105E 23 02 1062 BLS ST2 IT IS
00127A 1060 8B 07 A ADDA #$7 IT IS NOW
00128A 1062 B7 1011 A ST2 STAA RM USE FOR PRINT
00129A 1065 86 0A A LDAA #$9A NOW TURN ON TE & RE
00130A 1067 97 11 A STAA TRCS ALL SET NOW
00131A 1069 C5 04 A BITB #$4 CHECK FORMAT
00132A 106B 26 07 1074 BNE NXT1 IT'S NRZ
00133A 106D CE 4249 A LDX #$4249 "BI" INTO X
00134A 1070 86 50 A LDAA #$P
00135A 1072 20 05 1079 BRA NXT2

00137A 1074 CE 4E52 A NXT1 LDX #$4E52 "NR" INTO X
00138A 1077 86 5A A LDAA #$Z
00139A 1079 FF 1019 A NXT2 STX FMT
00140A 107C B7 101B A STAA FMT+2 FORMAT NOW FIXED
00142A 107F 86 54 A LDAA #$T INT & EXT END IN "T"
00143A 1081 B7 1025 A STAA CLK+2
00144A 1084 C1 0B A CMPB #11 WHICH CLOCK
00145A 1086 23 05 108D BLS NXT3
00146A 1088 CE 4558 A LDX #$4558 "EX" INTO X
00147A 108B 20 03 1090 BRA NXT4

00149A 108D CE 494E A NXT3 LDX #$494E "IN" INTO X
00150A 1090 FF 1023 A NXT4 STX CLK CLK NOW SET
00151A 1093 CE 100B A LDX #HDR0 PRINT THE WHOLE MESS
00152A 1096 BD F80F A JSR PDATA
00153A 1099 CE 1009 A LDX #ERRKT NOW SHOW ERRORS
00154A 109C BD F815 A JSR OUT4HS
00155A 109F BD F818 A JSR CRLF

00157 * DO DUMMY READ ON RECEIVER TO ELIMINATE
00158 * POSSIBLE FALSE RECEIVER FLAG (RDRF) CAUSED
00159 * BY CHANGING THE RATE & MODE CONTROL REGISTER

00161A 10A2 DC 11 A LDD TRCS READ STATUS AND DATA

00163 * WAIT ON TRANSMITTER

00165A 10A4 86 20 A LOOP LDAA #$20 TDRE FLAG BIT
00166A 10A6 95 11 A LOOP1 BITA TRCS MAKE SURE TDRE=1

Figure 6-8. Exercising the SCI: SERIAL (Continued)
Figure 6-8. Exercising the SCI: SERIAL (Continued)
<table>
<thead>
<tr>
<th>RMCR:02</th>
<th>FMT:BIP</th>
<th>CLK:INT</th>
<th>ERRS:0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0101</td>
<td>0202</td>
<td>0303</td>
</tr>
<tr>
<td>0404</td>
<td>0505</td>
<td>0606</td>
<td>0707</td>
</tr>
<tr>
<td>0808</td>
<td>0909</td>
<td>OA0A</td>
<td>OB0B</td>
</tr>
<tr>
<td>OC0C</td>
<td>OD0D</td>
<td>OE0E</td>
<td>OF0F</td>
</tr>
<tr>
<td>1010</td>
<td>1111</td>
<td>1212</td>
<td>1313</td>
</tr>
<tr>
<td>1414</td>
<td>1515</td>
<td>1616</td>
<td>1717</td>
</tr>
<tr>
<td>1818</td>
<td>1919</td>
<td>1A1A</td>
<td>1B1B</td>
</tr>
<tr>
<td>1C1C</td>
<td>1D1D</td>
<td>1E1E</td>
<td>1F1F</td>
</tr>
<tr>
<td>2020</td>
<td>2121</td>
<td>2222</td>
<td>2323</td>
</tr>
<tr>
<td>2424</td>
<td>2525</td>
<td>2626</td>
<td>2727</td>
</tr>
<tr>
<td>2828</td>
<td>2929</td>
<td>2A2A</td>
<td>2B2B</td>
</tr>
<tr>
<td>2C2C</td>
<td>2D2D</td>
<td>2E2E</td>
<td>2F2F</td>
</tr>
<tr>
<td>3030</td>
<td>3131</td>
<td>3232</td>
<td>3333</td>
</tr>
<tr>
<td>3434</td>
<td>3535</td>
<td>3636</td>
<td>3737</td>
</tr>
<tr>
<td>3838</td>
<td>3939</td>
<td>3A3A</td>
<td>3B3B</td>
</tr>
<tr>
<td>3C3C</td>
<td>3D3D</td>
<td>3E3E</td>
<td>3F3F</td>
</tr>
<tr>
<td>4040</td>
<td>4141</td>
<td>4242</td>
<td>4343</td>
</tr>
<tr>
<td>4444</td>
<td>4545</td>
<td>4646</td>
<td>4747</td>
</tr>
<tr>
<td>4848</td>
<td>4949</td>
<td>4A4A</td>
<td>4B4B</td>
</tr>
<tr>
<td>4C4C</td>
<td>4D4D</td>
<td>4E4E</td>
<td>4F4F</td>
</tr>
<tr>
<td>5050</td>
<td>5151</td>
<td>5252</td>
<td>5353</td>
</tr>
<tr>
<td>5454</td>
<td>5555</td>
<td>5656</td>
<td>5757</td>
</tr>
<tr>
<td>5858</td>
<td>5959</td>
<td>5A5A</td>
<td>5B5B</td>
</tr>
<tr>
<td>5C5C</td>
<td>5D5D</td>
<td>5E5E</td>
<td>5F5F</td>
</tr>
<tr>
<td>6060</td>
<td>6161</td>
<td>6262</td>
<td>6363</td>
</tr>
<tr>
<td>6464</td>
<td>6565</td>
<td>6666</td>
<td>6767</td>
</tr>
<tr>
<td>6868</td>
<td>6969</td>
<td>6A6A</td>
<td>6B6B</td>
</tr>
<tr>
<td>6C6C</td>
<td>6D6D</td>
<td>6E6E</td>
<td>6F6F</td>
</tr>
<tr>
<td>7070</td>
<td>7171</td>
<td>7272</td>
<td>7373</td>
</tr>
<tr>
<td>7474</td>
<td>7575</td>
<td>7676</td>
<td>7777</td>
</tr>
<tr>
<td>7878</td>
<td>7979</td>
<td>7A7A</td>
<td>7B7B</td>
</tr>
<tr>
<td>7C7C</td>
<td>7D7D</td>
<td>7E7E</td>
<td>7F7F</td>
</tr>
<tr>
<td>8080</td>
<td>8181</td>
<td>8282</td>
<td>8383</td>
</tr>
<tr>
<td>8484</td>
<td>8585</td>
<td>8686</td>
<td>8787</td>
</tr>
<tr>
<td>8888</td>
<td>8989</td>
<td>8A8A</td>
<td>8B8B</td>
</tr>
<tr>
<td>8C8C</td>
<td>8D8D</td>
<td>8E8E</td>
<td>8F8F</td>
</tr>
<tr>
<td>9090</td>
<td>9191</td>
<td>9292</td>
<td>9393</td>
</tr>
<tr>
<td>9494</td>
<td>9595</td>
<td>9696</td>
<td>9797</td>
</tr>
<tr>
<td>9898</td>
<td>9999</td>
<td>9A9A</td>
<td>9B9B</td>
</tr>
<tr>
<td>9C9C</td>
<td>9D9D</td>
<td>9E9E</td>
<td>9F9F</td>
</tr>
<tr>
<td>A0A0</td>
<td>A1A1</td>
<td>A2A2</td>
<td>A3A3</td>
</tr>
<tr>
<td>A4A4</td>
<td>A5A5</td>
<td>A6A6</td>
<td>A7A7</td>
</tr>
<tr>
<td>A8A8</td>
<td>A9A9</td>
<td>AAAA</td>
<td>ABA0</td>
</tr>
<tr>
<td>ACAC</td>
<td>ADAD</td>
<td>AEAE</td>
<td>AFAF</td>
</tr>
<tr>
<td>B0B0</td>
<td>B1B1</td>
<td>B2B2</td>
<td>B3B3</td>
</tr>
<tr>
<td>B4B4</td>
<td>B5B5</td>
<td>B6B6</td>
<td>B7B7</td>
</tr>
<tr>
<td>B8B8</td>
<td>B9B9</td>
<td>BA0A</td>
<td>BBBB</td>
</tr>
<tr>
<td>BCBC</td>
<td>BDDB</td>
<td>BEBE</td>
<td>BFBB</td>
</tr>
<tr>
<td>C0C0</td>
<td>C1C1</td>
<td>C2C2</td>
<td>C3C3</td>
</tr>
<tr>
<td>C4C4</td>
<td>C5C5</td>
<td>C6C6</td>
<td>C7C7</td>
</tr>
<tr>
<td>C8C8</td>
<td>C9C9</td>
<td>CA0A</td>
<td>CBCB</td>
</tr>
<tr>
<td>CDCC</td>
<td>CDCD</td>
<td>CECE</td>
<td>CFCC</td>
</tr>
<tr>
<td>D0D0</td>
<td>D1D1</td>
<td>D2D2</td>
<td>D3D3</td>
</tr>
<tr>
<td>D4D4</td>
<td>D5D5</td>
<td>D6D6</td>
<td>D7D7</td>
</tr>
<tr>
<td>D8D8</td>
<td>D9D9</td>
<td>DADA</td>
<td>DBDB</td>
</tr>
<tr>
<td>DCDC</td>
<td>DDDD</td>
<td>DEDE</td>
<td>DFDF</td>
</tr>
<tr>
<td>E0E0</td>
<td>E1E1</td>
<td>E2E2</td>
<td>E3E3</td>
</tr>
<tr>
<td>E4E4</td>
<td>E5E5</td>
<td>E6E6</td>
<td>E7E7</td>
</tr>
<tr>
<td>E8E8</td>
<td>E9E9</td>
<td>EAE0</td>
<td>EBEB</td>
</tr>
<tr>
<td>ECEC</td>
<td>EDED</td>
<td>EE0E</td>
<td>EFEF</td>
</tr>
<tr>
<td>F0F0</td>
<td>F1F1</td>
<td>F2F2</td>
<td>F3F3</td>
</tr>
<tr>
<td>F4F4</td>
<td>F5F5</td>
<td>F6F6</td>
<td>F7F7</td>
</tr>
<tr>
<td>F8F8</td>
<td>F9F9</td>
<td>FA0A</td>
<td>FBFB</td>
</tr>
<tr>
<td>FCFC</td>
<td>FDFD</td>
<td>FEFE</td>
<td>FFFF</td>
</tr>
</tbody>
</table>

**Figure 6-9. Example of SERIAL Program Output**
**WAKEUP DEMO PROGRAM**

**THIS PROGRAM ILLUSTRATES THE M6801 WAKEUP FEATURE.**


THE PROGRAM HAS TWO VARIABLES:

- **MODE** — RATE & MODE DESIRED
- **WAKEUP** — FLAG TO INDICATE SHOULD BE RUN WITH OR WITHOUT WAKEUP.

THE PROGRAM USES LILBUG I/O AND IS INDEPENDENT OF THE TERMINAL'S BAUD RATE.

IT ASSUMED THAT THE SERIAL OUTPUT IS CONNECTED TO SERIAL INPUT. AN EXTERNAL BAUD RATE CLOCK MAY NOT BE USED WITH THIS PROGRAM.

**EQUATES**

- **F809 A OUTCH EQU $F809** LILBUG OUTPUT CHAR ROUTINE
- **0011 A TRCS EQU $0011** TX/RX CONTROL/STATUS REG
- **0013 A TX EQU $0013** TX DATA REGISTER
- **0012 A RX EQU $0012** RX DATA REGISTER
- **0010 A RMCR EQU $0010** RATE & MODE CNTRL REG
- **F818 A CRFL EQU $F818** LILBUG CR AND LF
- **F80C A PDATA1 EQU $F80C** LILBUG PRINT STRING ROUTINE

**STORAGE CELLS**

- **ORG **$1000

**START-UP VECTOR**

**Figure 6-10. Demonstrating the Wake-Up Feature: OKBAD**

6-20
**Figure 6-10. Demonstrating the Wake-Up Feature: OKBAD (Continued)**
**Figure 6-10. Demonstrating the Wake-Up Feature: OKBAD (Continued)**

6-22
**Figure 6-10.** Demonstrating the Wake-Up Feature: OKBAD (Continued)
Figure 6-11. Example of OKBAD Program Output
CHAPTER 7
THE MC6801 PROGRAMMABLE TIMER

7.0 INTRODUCTION

The MC6801 Programmable Timer can be used for many purposes including measuring the pulse width of an input signal and simultaneously generating an output signal. Pulse widths for both input and output signals can vary from several microseconds to many seconds. The Programmable Timer is also capable of generating periodic interrupts or indicating passage of an arbitrary number of MPU E-cycles.

In order to implement these types of applications, the reader must become familiar with the registers which control and access the Programmable Timer. This chapter provides a detailed description of the operation of the four addressable registers used to interface with the Programmable Timer. It concludes with a few examples which are intended to illustrate various features.

7.1 PROGRAMMABLE TIMER REGISTERS

The capabilities of the Programmable Timer are obtained using the following four addressable registers:

- the Counter Register,
- the Output Compare Register,
- the Input Capture Register, and
- the Timer Control and Status Register (TCSR).

The overall organization of the programmable Timer Registers is shown in Figure 7-1 while Figure 7-2 depicts a block diagram of the Programmable Timer.

There are two ways to discuss the Programmable Timer, depending upon whether the Timer Control and Status Register is mentioned first or last. Unfortunately, either choice leads to forward

![Figure 7-1. MC6801 Programmable Timer Registers](image-url)
references but it is felt that the latter one is somewhat preferable. Therefore, the five control bits (EICI, EOCI, ETOI, IEDG, WU) and three status bits (ICF, OCF, TOF) referred to in the next three sections are not defined until the discussion of the register in which they reside: namely, the Timer Control and Status Register (TCSR).

7.1.1 Counter Register ($09:0A$)

The key element in the Programmable Timer is in a 16-bit free-running counter, or Counter Register, which is clocked to increasing values during each negative half-cycle of the MPU E-clock. Software can read the Counter Register at any time without affecting its value. The Counter Register is clocked and read during opposite half cycles of the MPU E-cycle clock.

The Counter Register must be read using an instruction which first addresses its most significant byte ($09$). An MPU read of this address causes the least significant byte to be transferred to a buffer. This buffer is cleared by Reset and is accessed when reading the Counter Register least significant byte ($0A$). For double byte read instructions, these two accesses occur on consecutive bus cycles. Note that unless the Counter Register most significant byte is also read, the same value will be obtained from more than one read of the least significant byte.
The Counter Register is cleared during Reset and is a read-only register with a single exception: any MPU write to its most significant byte ($09) will always preset it to $FFFF8 regardless of the value involved in the write. This preset feature is intended for use in testing but could be of value in some applications.

NOTE

The Counter Register also provides a bit rate clock for the Serial Communications Interface (SCI). MPU writes to the Counter Register should be avoided if the SCI is being used with the internal clock.

The 16-bit Counter Register repeats every 65,536 MPU E-cycles. When the Counter Register contains all ones, the Timer Overflow Flag (TOF) bit is set during the same half cycle. An interrupt can also be enabled when rollover occurs by setting its interrupt enable bit, ETOI.

7.1.2 Output Compare Register ($0B:0C)

The Output Compare Register is a 16-bit read/write register which is initialized to $FFFF by Reset and can be used for several purposes. Two possible applications include controlling an output waveform and indicating when a period of time has elapsed. Of the four Programmable Timer registers, the Output Compare Register is unique in that all bits are readable and writable and are not altered by the Timer hardware (except during Reset). If the compare function is not utilized, the two bytes of the Output Compare Register can be used simply as a storage location.

The Output Compare Register and the Counter Register are compared during each negative half-cycle of the MPU E-clock. If a match is found, the Output Compare Flag (OCF) bit is set and the Output Level (OLVL) bit is clocked to an output level register. Providing Port 2 bit 1 is defined as an output by its corresponding bit in the Data Direction Register, the value of the output level register will appear at P21. The values in the Output Compare Register and Output Level bit must be changed after each successful comparison to control an output waveform or establish a new elapsed timeout. An interrupt can also accompany a successful output compare providing the interrupt enable bit, EOCI, is set.

After an MPU write cycle to the most significant byte of the Output Compare Register ($0B), the output compare function is inhibited for one E-cycle. This allows both bytes to be written on consecutive E-cycles before making the next comparison. Therefore, if desiring to change both bytes of the register, a double byte write instruction should be used in order to take advantage of the compare inhibit feature.

MPU writes can be made to either byte of the Output Compare Register, however, without affecting the other byte. The Output Level bit (OLVL) is clocked to the output level register regardless of whether the Output Compare Flag (OCF) is set or clear. A timing diagram for the output compare function is shown in Figure 7-3.
7.1.3 Input Capture Register (S0D:0E)

The Input Capture Register is a 16-bit read-only register which is cleared by Reset and is used to latch the value of the Counter Register when a defined transition is sensed by the input capture edge detector. The level transition which triggers a Counter Register transfer is controlled by the Input Edge bit (IEDG).

External devices interface with the input capture function using P20. Typically, Port 2 bit 0 is configured as an input but the edge detector is always sensing this line even if configured as an output.

The result obtained by an input capture corresponds to the value of the Counter Register on the second negative half-cycle of the MPU E-clock following the transition, as shown in Figure 7-4. This one cycle delay is required for internal synchronization.

The Counter Register is transferred to the Input Capture Register on each proper signal transition regardless of whether the Input Capture Flag (ICF) is set or clear. The register always contains the counter value corresponding to the most recent input capture.

After a read of the most significant byte of the Input Capture Register, Counter Register transfer is inhibited during the next negative half-cycle of E (Enable). During double byte reads, this inhibited transfer will occur between consecutive read cycles of the most and least significant bytes. This characteristic requires input pulse widths be at least two MPU E-cycles for recognition under all conditions. If the application is such that it is possible to guarantee that the Input Capture Register will not be read during an input capture, then one MPU E-cycle is sufficient.
7.1.4 Timer Control and Status Register ($08)

The Timer Control and Status Register (TCSR) is an 8-bit register which contains three status bits and five control bits. The three most significant bits contain read-only status information and indicate that:

- a proper transition has taken place at P20 with an accompanying transfer of the Counter Register to the Input Capture Register,
- a match has been found between the Counter Register and the Output Compare Register, and
- a value of $FFFF has been sensed in the Counter.

Of the remaining five bits of the TCSR, three of the bits control interrupts associated with each of the three flag bits. The other two bits control (1) which edge is significant to the input capture edge detector and (2) the next value to be clocked to the output level register in response to a successful output compare.

Each of three Programmable Timer functions can generate an IRQ2 interrupt if its individual enable bit is set and each has a separate prioritized interrupt vector. The MCU interrupt mask bit (I-bit in the Condition Code Register) controls all maskable interrupts and must be clear to enable any Programmable Timer interrupt. The associated interrupt vectors, in order of decreasing priority, are:

1. Input Capture Vector: $FFF6:FFF7
2. Output Compare Vector: $FFF4:FFF5
3. Timer Overflow Vector: $FFF2:FFF3

```
<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OLVL - Output Level</td>
</tr>
<tr>
<td>1</td>
<td>IEDG - Input Edge</td>
</tr>
<tr>
<td>2</td>
<td>ETOI - Enable Timer</td>
</tr>
<tr>
<td>3</td>
<td>EOCI - Enable Output</td>
</tr>
<tr>
<td>4</td>
<td>EICI - Enable Input</td>
</tr>
<tr>
<td>5</td>
<td>TOF - Timer Overflow</td>
</tr>
</tbody>
</table>
```

The TCSR is illustrated in Figure 7-5 where each bit is defined as follows:

- **Bit 0 OLVL**: Output Level — This is the next value to be clocked to the output level register by a successful output compare and will appear at P21 if Bit 1 of Port 2 Data Direction Register is set. OLVL is cleared by Reset. The Output Level Register is not affected by Reset.

- **Bit 1 IEDG**: Input Edge — IEDG is cleared by Reset and controls which level transition will trigger a Counter Register transfer to the Input Capture Register:
  - IEDG = 0 Transfer on a negative-going edge
  - IEDG = 1 Transfer on a positive-going edge

- **Bit 2 ETOI**: Enable Timer Overflow Interrupt — If set, an IRQ2 interrupt is enabled whenever TOF is set; if clear, the interrupt is inhibited. It is cleared by Reset.

- **Bit 3 EOCI**: Enable Output Compare Interrupt — If set, an IRQ2 interrupt is enabled whenever OCF is set; if clear, the interrupt is inhibited. It is cleared by Reset.

- **Bit 4 EICI**: Enable Input Capture Interrupt — If set, an IRQ2 interrupt is enabled whenever ICF is set; if clear, the interrupt is inhibited. It is cleared by Reset.

- **Bit 5 TOF**: Timer Overflow Flag — TOF is set when the Counter Register contains $FFFF. It is cleared by reading the TCSR (with TOF set) followed by the Counter Register most significant byte ($09), or by Reset.
Bit 6 OCF  Output Compare Flag — OCF is set when the Output Compare Register matches the Counter Register. It is cleared by reading the TCSR (with OCF set) and then writing to the Output Compare Register ($0B to $0C), or by Reset.*

Bit 7 ICF  Input Capture Flag — ICF is set when a proper edge has been sensed by the input capture edge detector. It is cleared by an MPU read of the TCSR (with ICF set) followed by the Input Capture Register most significant byte ($D), or by Reset.

Note that reading the TCSR satisfies the first condition required to clear any status bits which happen to be set during the read. The only remaining step to clear the status bit is to make an access of the appropriate register. Typically, this presents no problem for the input capture and output compare functions.

A problem can occur, however, when using the timer overflow function and reading the Counter Register at random times to, say, measure an elapsed time. Without incorporating the proper precautions into software, the Timer Overflow Flag could unintentionally be cleared if (1) the TCSR is read when TOF is set and (2) the most significant byte of the Counter Register is read but not for the purpose of servicing the flag. Solutions to this problem are application-dependent, and, typically, involve always reading the TCSR before the Counter Register. Based on the value found for the overflow flag, software must then perform some action which ensures that the flag will be serviced.

Finally, if any Programmable Timer function is operated interrupt-driven, the programmer must be familiar with and observe the precautions noted in Section 5.3 concerning characteristics of the IRQ2 interrupt.

7.2 SELECTED PROGRAMMABLE TIMER EXAMPLES

While the preceding discussion defines Programmable Timer operation, it is recognized that examples are valuable in demonstrating its various features. The following examples illustrate several aspects of Programmable Timer operation. Most of the examples are tutorial in nature; their goal is to inform rather than to provide solutions to specific applications. The examples also present analysis techniques which could be helpful in some applications.

7.2.1 Reading the Counter Register

The free-running counter (Counter Register) can be considered a read-only register with one exception: any write to the Counter Register will cause it to preset to $FFF8. If the write is immediately followed by a double byte read of the Counter Register, the value $FFFF will always be obtained as shown in the following instruction sequence:

```
STX $09 /WRITE TO COUNTER REGISTER
LDX $09 /READ THE COUNTER REGISTER
```

The timing description in Figure 7-6 illustrates why the Index Register will contain $FFFF after the Counter Register is read.

*The output level register is also cleared by Reset.
7.2.2 Generating an Output Waveform

Many possible output waveforms can be generated by using software to control the Output Compare Register and the OLVL bit in the TCSR. Software written to generate an output waveform must satisfy the following requirements:

1. Port 2 bit 1 must be configured as an output by writing a “1” into the appropriate bit in its Data Direction Register,
2. the Output Level Bit (OLVL) must be toggled after each successful compare, and
3. the Output Compare Register must be updated before the Counter Register reaches the new value.

The program, WAVGEN, shown in Figure 7-8, illustrates use of the output compare function. The purpose of the program is to generate the waveform shown in Figure 7-7. The output signal can be observed by connecting Port 2 bit 1 (P21) to an oscilloscope. The duty cycle and period of the output can be varied by changing the values of the double byte variables, OFF1 and OFF2.
** Figure 7-8. Generating a Waveform: WAVGEN**
PAGE 002 WAVGEN:SA:1 WAVGEN **** OUTPUT COMPARE FUNCTION GENERATOR ****

00055

00056
00057
00058
00059

00061A 0300 86 02 A START LDAA #2 CHANGE PORT 2 BIT 1 TO OUTPUT
00062A 0302 97 01 A STAA PZDDR DDR NOW CONFIGURED

00064

00066A 0304 96 08 A LOOP LDAA TCSR
00067A 0306 85 40 A BITA #$40 CHECK OCF
00068A 0308 27 FA 0304 BEQ LOOP KEEP CHECKING

00070

00072A 030A 88 01 A EORA #1 TOGGLE OLVL BIT
00073A 030C 97 08 A STAA TCSR
00074A 030E 44 A LSRA .SAVE OLVL BIT IN CARRY

00076

00077

00079A 030F DC 0B A LDD OUTCMP GET OUTPUT COMPARE REG
00080A 0311 24 05 0318 BCC OFFS2 LAST OLVL SAVED IN CARRY BIT
00081A 0313 F3 02FC A ADDD OFF1 ADD TO OFFSET 1
00082A 0316 20 03 031B BRA ALLDUN .

00084A 0318 F3 02FE A OFFS2 ADDD OFF2 .ADD OFFSET 2
00085A 031B DD 0B A ALLDUN STD OUTCMP BACK INTO COMPARE REGISTER
00086A 031D 20 E5 0304 BRA LOOP GO AGAIN

00088

0300 A END START

TOTAL ERRORS 00000--00000

Figure 7-8. Generating a Waveform: WAVGEN (Continued)
7.2.3 Generating a Synchronized Output Compare

This example is presented only for instructional purposes and admittedly has minimal useful applications value. Its value lies in understanding the timing considerations required for the program to function.

The objective of this example is to generate a square wave output using the output compare function without any reference to the Output Compare Flag (OCF). This requires that the program be synchronized with the Counter Register such that a successful compare always occurs on the cycle following the write to the Output Compare Register.

The solution is shown in Figure 7-10 and the timing involved in its iterative loop is illustrated in Figure 7-9. The loop requires exactly 15 MPU cycles. Once the loop is entered correctly it will remain synchronized because 15 (the cycle count of the loop) is constantly being added to the value in the Output Compare Register. The key task is to determine how to enter the loop with the correct value.

![Figure 7-9. Synchronized Loop Timing](image)

The loop is synchronized with the Counter Register by reading it 19 cycles before the first compare and then adding four to obtain the value for the next compare. The addition of four cycles is equivalent to reading the Counter Register exactly 15 MPU E-cycles before the compare within the loop. When the routine enters the loop, the number of cycles remaining until the next compare is identical to that occurring from within the loop. Because this program is totally dedicated to the task of waveform generation, its output represents the minimum period which can be generated using the output compare function.
SYNLUP — A PROGRAM TO GENERATE A SQUARE WAVE

* OUTPUT USING THE OUTPUT COMPARE FUNCTION OF THE M6801 TIMER.

* THE PROGRAM DOES NOT USE THE OUTPUT COMPARE FLAG AND, INSTEAD, SYNCHRONIZES WITH THE TIMER IN A 15 CYCLE LOOP.

* THE OUTPUT IS A 30-CYCLE PERIOD SQUARE WAVE.

* THE PROGRAM IS INTENDED FOR INSTRUCTIONAL USE ONLY AND MAY NOT BE USED IN A "CYCLE STEALING" ENVIRONMENT WHICH MAY DESTROY THE TIMER SYNCHRONIZATION.

EQUATES

ORG $300
LDAA #2 SET P21 TO OUTPUT
STAA P2DDR PORT 2 BIT 1 SET
LDD $4 CLEAR A, 4→B
LDX TIMER GET CURRENT TIMER VALUE
ABX ADD 4 CYCLES
LDAB #15 B HAS CONSTANT LOOP VALUE OF 15 CY

* LEAD-IN NOW SYNCRONIZED TO TIMER

* MAIN LOOP

* THE FOLLOWING LOOP IS EXACTLY 15 CYCLES

* LONG, THE OUTPUT COMPARE IS MADE ON THE NEXT CYCLE AFTER THE WRITE TO THE LOW BYTE OF THE OUTPUT COMPARE REGISTER.

* EOR A #1 TOGGLE OLVL BIT

STAA TCSR OLVL NOW FIXED UP
ABX ADD 15 CYCLES TO OUTPUT COMPARE
STX OUTCMP OUTPUT COMPARE NOW SET
BRA LOOP DO IT ALL AGAIN
END START

TOTAL ERRORS 00000--00000

Figure 7-10. Synchronized Output Compare: SYNLUP
7.2.4 Echoing an Input Signal

The purpose of this example is to use the input capture function to detect an edge from an input signal and to use the output compare function to echo it at the output level pin, P21. Because the only way to generate a level change is by a successful output compare, the function of this program is three-fold:

1. sense an input level change using the Input Capture Flag (ICF),
2. upon detecting the level change, toggle the Output Level Bit (OLVL), and
3. force an immediate output compare to clock the new level to the output pin.

The level change on the input can be sensed by monitoring the Input Capture Flag (ICF). After the OLVL bit has been toggled, the remaining task is to clock it to the Output Level Register. This can only be accomplished by a successful compare with the Output Compare Register. The following three instructions can be used to generate an immediate output compare:

```
LDD $9   /GET COUNTER REGISTER VALUE
ADDD #10 /ADD AN OFFSET
STD $B   /UPDATE OUTPUT COMPARE
```

The diagram shown in Figure 7-11 illustrates the timing involved in the instruction sequence.

![Diagram of Immediate Output Compare Timing](image)

**Figure 7-11. Immediate Output Compare Timing**

If it is undesirable to disturb the A accumulator, the following four-instruction sequence can also be used:

```
LDAB #9   /COUNTER OFFSET TO ACCB
LDX $9    /READ COUNTER REGISTER
ABX       /ADD OFFSET
STX $B    /UPDATE OUTPUT COMPARE
```

The program ECHO, shown in Figure 7-13, illustrates the application of these techniques. The equipment arrangement shown in Figure 7-12 can be used to drive the program and monitor its output.
7.2.5 Generating an Input Capture Using LILbug (TM)*

This example is concerned with illustrating the effect of Reset on the Programmable Timer registers. Assume that the Programmable Timer output (OLVL) is connected to its input (input capture) and an external pullup resistor is connected to Port 2 bit 1. The MC6801 has been Reset and the following dialogue appears on the terminal:

```
LILBUG 1.0
!08 6061 < Set OLVL >
!01 FFFE < Make Port 2 bit 1 an Output >
!08 6160 < Clear OLVL >
!08 E0 < ICF is set >
```

What value should now be contained in the Input Capture Register?

During Reset all registers in the Programmable Timer were cleared to zero except the Output Compare Register which was set to $FFFF. The effect of this initialization is to cause an output compare whenever the Counter Register cycles through $FFFF. The Input Capture Unit is armed for a negative-going transition and the Output Level Bit is initialized to zero.

An input capture was generated by (1) setting OLVL in the TCSR, (2) configuring Port 2 bit 1 as an output, and (3) clearing OLVL. This sequence generates the negative-going transition which triggers the input capture function. The task which remains is to determine the value of the Counter Register which is transferred to the Input Capture Register.

Refer to the timing diagram in Figure 7-3. The output compare changes the output level on the positive half-cycle of the MPU E-cycle following a successful compare. The Counter Register transfer occurs on the second negative half-cycle following the edge. The Counter Register contained $FFFF when the compare was made but is incremented two more times before the transfer is accomplished. Therefore, if LILbug is used to read the Input Capture Register, it would find:

```
!0D 00 01
```

*LILbug (TM) is a monitor programmed in the ROM of the MC6801L1. See the LILbug Manual for details.
**SA:1 ECHO  **** TIMER INPUT CAPTURE & OUTPUT COMPARE FUNCTION**

```
00001      NAM  ECHO
00002      OPT  LL=80
00003      OPT  Z01
00004      TTL  **** TIMER INPUT CAPTURE & OUTPUT COMPARE FUNCTION
00005  **********************************************
00006  *
00007  *  ECHO --- THIS PROGRAM ECHOES A WAVEFORM FED INTO
00008  *  THE TIMER INPUT LEVEL (PORT 2 BIT 0) USING
00009  *  THE OUTPUT COMPARE FUNCTION (PORT2 BIT 1).
00010  *  THE INPUT LEVEL SHOULD BE REPEATED ON THE
00011  *  OUTPUT LEVEL OF THE TIMER WITH A SMALL DELAY
00012  *
00013  *  THE PERIOD OF THE INPUT SQUARE WAVE
00014  *  MUST BE AT LEAST 64 MPU CYCLES FOR THE
00015  *  PROGRAM TO HAVE SUFFICIENT TIME TO
00016  *  FUNCTION CORRECTLY.
00017  *
00018  *
00019  *  INPUT    OUTPUT
00020  *  +       -
00021  *  +       -
00022  *  +       -
00023  *  +       -
00024  *  +-------+
00025  *
00026  *  ----> 26 OR 29 CYCLES <-----
00027  *
00028  **********************************************
```

**Figure 7-13. Echoing an Input: ECHO**
PAGE 002 ECHO .SA:1 ECHO **** TIMER INPUT CAPTURE & OUTPUT COMPARE FUN

00042
00043
00044
00045
00046

00048 A 0300
00049 A 0300 86 02 A START LDAA #2 MAKE PORT 2 BIT 1 AN OUTPUT
00050 A 0302 97 01 A STAA P2DDR ALL SET NOW

00052 A 0304 7C 0008 A INC TCSR START WITH OLVL = "1"
00053 A 0307 96 08 A LDAA TCSR CLR OUT INPUT CAPTURE FLAG
00054 A 0309 96 0D A LDAA INPCAP READ STATUS/READ DATA CLRS FLAG

00056 A 030B 96 08 A LOOP LDAA TCSR CHECK FLAG
00057 A 030D 2A FC 030B BPL LOOP FLAG NOT SET

00059 A 030F 88 03 A EORA #3 CHANGE EDGE AND OUTPUT LEVEL
00060 A 0311 97 08 A STAA TCSR
00061 A 0313 96 0D A LDAA INPCAP CLEAR INPUT CAPTURE FLAG
00062 A 0315 DC 09 A LDD TIMER GET CURRENT TIMER VALUE
00063 A 0317 C3 000A A ADDD #10 ADD JUST ENOUGH OFFSET
00064 A 031A DD 0B A STD OUTCMP FOR AN IMMEDIATE COMPARE
00065 A 031C 20 ED 030B BRA LOOP

00067 A 0300 A END START
TOTAL ERRORS 00000--00000

Figure 7-13. Echoing an Input: ECHO (Continued)
CHAPTER 8
SELECTED APPLICATIONS

8.0 INTRODUCTION

While the preceding chapters were concerned with explaining the functional aspects of the MC6801, this chapter focuses on its application. The topics discussed in these applications are not presented in any particular order nor are they intended to be the solution to any specific problem. Instead, they are in keeping with the spirit of preceding chapters and are, therefore, tutorial in nature. They treat a wide range of topics which are considered to be of interest to many readers.

8.1 INTERFACE TO STATIC RAM (MODES 1, 2, 3, 6)

The MC6801 can be interfaced to a variety of devices using the expanded operating modes. The directly addressable external address space in the Expanded Non-Multiplexed Mode (Mode 5), however, is limited to $100 to $1FF due to the width of the address bus. In this mode, only eight address lines and an Input/Output Select are available which provides an external memory space of 256 bytes.

In the Expanded Multiplexed Modes, however, the MCU can access a 64K memory space. When interfacing to the expanded multiplexed bus, the following factors must be considered:

1. the MC6801 bus is synchronous, clocked by E (Enable), and cannot be easily "stretched" either for slow devices or direct memory access (DMA),
2. the low order address bits (A0-A7) are multiplexed with the data bus (D0-D7),
3. the Address Strobe (AS) signal must be used to control a latch to de-multiplex the two buses,
4. no external device can be enabled onto the data bus until the positive edge of Enable (E), and
5. all external devices must vacate the data bus by the positive edge of Address Strobe.

8.1.1. Expanded Multiplexed Bus Timing

Timing for the expanded multiplexed bus is shown in Figure 8-1 where numerical values for each of the symbols can be obtained from the MC6801 Data Sheet. From examination of this figure, it should be noted that

1. the negative edge of Address Strobe (AS) can be used to latch the eight least significant bits of the address bus,
2. although the address is valid for a short interval prior to the negative edge of the Address Strobe, devices cannot be enabled onto the data bus until E is high in order to avoid interference with the eight least significant lines of the address bus, and
3. the data bus must be vacated on the negative edge of E (Enable) to avoid interference with the address bus.
8.1.2 An MC6801 Interface with MCM2114 Static RAM

This discussion considers the interface of two MCM2114 static RAMs with the MC6801 expanded multiplexed bus. The MCM2114 is a 4096-bit static random access memory (RAM) organized as 1024 nibbles (1024 x 4). The address space for this block of memory is arbitrarily chosen for this example as $EC00 to $EFFF. Let us also arbitrarily require that the sixteen most significant locations of the RAM respond to the interrupt vector addresses, $FF00 to $FFFF. The latter requirement is only for purposes of illustration and probably would not be a typical requirement. Finally, the proposed design will require only simple modifications to fill the block of addresses ($EO00-$EBFF) with an additional six RAMS.

The operating mode of the MC6801 can be assumed to be 1, 2, 3, or 6. This design will not function correctly with Mode 0 because the memory address decoder will not respond correctly to the interrupt vectors. In modes 1, 2, or 3, external interrupt vectors are located in the sixteen most significant locations of the external RAM.

The reader could be wondering, at this point, just how these RAM locations are initialized. One approach is to Reset the system into Mode 6 and use the memory examine/change function of a monitor such as LILbug (TM). After initializing these locations, the system could be Reset again into Modes 1, 2, or 3.
It should be noted that this design does not include any buffering which implies that it is intended for use in a minimal single board system. A typical data sheet for the MCM2114 static RAM is shown in Figure 8-2. However, a current data sheet should be obtained for any actual design.

The MCM2114 can be interfaced to a system using ten address lines (A0-A9), a chip select (active low), and a write enable signal (active low for write). In response to a chip select, each MCM2114 will provide four bits of data which requires two of them to obtain all eight bits to the data bus.

The objective of the interface circuitry is to provide the address and R/W (Read/Write) signals and then enable the memory at the proper time. Details for the interface connections are shown in Figure 8-3.

The first step is to de-multiplex the Address and Data buses using a transparent 74LS373 latch. Note that AS (Address Strobe) can be used directly to control the latch.

The Data Bus, address lines A0 to A9, and R/W are connected directly to the memories. The only remaining required signal is a chip select, SEL. This signal is derived using combinatorial logic to decode either an address in the inclusive range of $E000 to $EFFF or $FFFF to $FFFF.

If the decoder senses either of these conditions, its output signal is then further qualified using the positive half-cycle of E (Enable). When all of these conditions are true, an active level is then presented to the Gate (G) input of the 2-to-4 line decoder which selects one of four possible pairs of MCM2114s based on the current values of address lines, A10 and A11. Because only one pair of memories is used in this example, only the output select line, SEL3, is shown as connected. Additional 1024-byte blocks of memory can be selected using the remaining three decoder outputs.

### 8.1.3 Final Remarks

Care should be taken when connecting any device to the MC6801 expanded multiplexed bus to ensure that it is not enabled to the data bus until after the positive edge of E (Enable). The reason for this precaution is that the data bus "does not exist" until $AHL$ (plus a brief MCU deselect time) after the negative edge of Address Strobe. Until that time, the lines are still half of the address bus.

Finally, one must consider the effect of the Reset state upon the circuit. During Reset, the Port 4 internal pullup resistors pull each line high. Port 3, however, is driven to a high impedance state and, without external pullup resistors, its output levels are undefined. The AS and R/W lines will be held high during Reset.

Note that this can result in an occasional "select" if the Port 3 lines (lower half of the address) should happen to "float" to $FX$ where "X" indicates "don't care." Due to the most significant byte of the address being held at $FF$, the RAM will select any time the Port 3 lines "float" to $FX$.

This will pose no difficulty in this example for two reasons; (1) the R/W line is held at a level "1" (read) and, (2) only one device can be selected. If more than one device can be enabled while Port 3 "floats," then (1) external pullup resistors can be utilized or (2) chip selects can be further qualified using low level AS signal.
4096-BIT STATIC RANDOM ACCESS MEMORY

The MCM2114 is a 4096-bit random access memory fabricated with high density, high reliability N-channel silicon-gate technology. For ease of use, the device operates from a single power supply, is directly compatible with TTL and DTL, and requires no clocks or refreshing because of fully static operation. Data access is particularly simple, since address setup times are not required. The output data has the same polarity as the input data.

The MCM2114 is designed for memory applications where simple interfacing is the design objective. The MCM2114 is assembled in 18-pin dual-in-line packages with the industry standard pin-out. A separate chip select (S) lead allows easy selection of an individual package when the three-state outputs are OR-tied.

The MCM2114 series has a maximum current of 100 mA. Low power versions (i.e., MCM21L14 series) are available with a maximum current of only 70 mA.

- 1024 Words by 4-Bit Organization
- Industry Standard 18-Pin Configuration
- Single +5 Volt Supply
- No Clock or Timing Strobe Required
- Fully Static: Cycle Time = Access Time
- Maximum Access Time
  - MCM2114-20/MCM21L14-20: 200 ns
  - MCM2114-25/MCM21L14-25: 250 ns
  - MCM2114-30/MCM21L14-30: 300 ns
  - MCM2114-45/MCM21L14-45: 450 ns
- Fully TTL Compatible
- Common Data Input and Output
- Three-State Outputs for OR-Ties
- Low Power Version Available

Figure 8-2. Typical Data for MCM2114 Static RAM
### ABSOLUTE MAXIMUM RATINGS (See Note)

<table>
<thead>
<tr>
<th>Rating</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperature Under Bias</td>
<td>-10 to +80</td>
<td>°C</td>
</tr>
<tr>
<td>Voltage on Any Pin With Respect to VSS</td>
<td>-0.5 to +7.0</td>
<td>V</td>
</tr>
<tr>
<td>DC Output Current</td>
<td>5.0</td>
<td>mA</td>
</tr>
<tr>
<td>Power Dissipation</td>
<td>1.0</td>
<td>Watt</td>
</tr>
<tr>
<td>Operating Temperature Range</td>
<td>0 to +70</td>
<td>°C</td>
</tr>
<tr>
<td>Storage Temperature Range</td>
<td>-66 to +150</td>
<td>°C</td>
</tr>
</tbody>
</table>

NOTE: Permanent device damage may occur if ABSOLUTE MAXIMUM RATINGS are exceeded. Functional operation should be restricted to RECOMMENDED OPERATING CONDITIONS: Exposure to higher than recommended voltages for extended periods of time could affect device reliability.

### DC OPERATING CONDITIONS AND CHARACTERISTICS

**DC CHARACTERISTICS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>MCM2114</th>
<th>MCM211L4</th>
<th>MCM2114-20</th>
<th>MCM211L14-20</th>
<th>MCM2114-25</th>
<th>MCM211L14-25</th>
<th>MCM2114-30</th>
<th>MCM211L14-30</th>
<th>MCM2114-45</th>
<th>MCM211L14-45</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Load Current (All Input Pins, Vln = 0 to 5.5 V)</td>
<td>IIL</td>
<td>-</td>
<td>10</td>
<td>-</td>
<td>10</td>
<td>-</td>
<td>10</td>
<td>-</td>
<td>10</td>
<td>-</td>
<td>10</td>
</tr>
<tr>
<td>I/O Leakage Current (V = 2.4 V, VDD = 0.4 V to VCC)</td>
<td>IOI</td>
<td>-</td>
<td>10</td>
<td>-</td>
<td>10</td>
<td>-</td>
<td>10</td>
<td>-</td>
<td>10</td>
<td>-</td>
<td>10</td>
</tr>
<tr>
<td>Power Supply Current (V = 5.5 V, IO = 0 mA, TA = 25°C)</td>
<td>ICC1</td>
<td>-</td>
<td>80</td>
<td>-</td>
<td>80</td>
<td>-</td>
<td>80</td>
<td>-</td>
<td>80</td>
<td>-</td>
<td>80</td>
</tr>
<tr>
<td>Power Supply Current (V = 5.5 V, IO = 0 mA, TA = 0°C)</td>
<td>ICC2</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>100</td>
</tr>
<tr>
<td>Output Low Current VCC = 0.4 V</td>
<td>IOL</td>
<td>2.1</td>
<td>6.0</td>
<td>2.1</td>
<td>6.0</td>
<td>2.1</td>
<td>6.0</td>
<td>2.1</td>
<td>6.0</td>
<td>2.1</td>
<td>6.0</td>
</tr>
<tr>
<td>Output High Current VOL = 2.4 V</td>
<td>IOH</td>
<td>-</td>
<td>1.4</td>
<td>-1.4</td>
<td>-1.4</td>
<td>-1.4</td>
<td>-1.4</td>
<td>-1.4</td>
<td>-1.4</td>
<td>-1.4</td>
<td>-1.4</td>
</tr>
</tbody>
</table>

### CAPACITANCE

For 1.0 MHz, TA = 25°C, periodically sampled rather than 100% tested.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Symbol</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Capacitance (V = 0)</td>
<td>CIN</td>
<td>5.0</td>
</tr>
<tr>
<td>Input/Output Capacitance (V = 0)</td>
<td>CO/O</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Capacitance measured with a Boonton Meter or effective capacitance calculated from the equation: C = |I/A V|.

### AC OPERATING CONDITIONS AND CHARACTERISTICS

<table>
<thead>
<tr>
<th>Input Pulse Levels</th>
<th>0.8 Volt to 2.4 Volt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Rise and Fall Times</td>
<td>10 ns</td>
</tr>
<tr>
<td>Input and Output Timing Levels</td>
<td>1.5 Volts</td>
</tr>
<tr>
<td>Output Load</td>
<td>1 TTL Gate and</td>
</tr>
</tbody>
</table>

### READ (NOTE 1), WRITE (NOTE 2) CYCLES

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>MCM2114-20</th>
<th>MCM211L14-20</th>
<th>MCM2114-25</th>
<th>MCM211L14-25</th>
<th>MCM2114-30</th>
<th>MCM211L14-30</th>
<th>MCM2114-45</th>
<th>MCM211L14-45</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address Valid to Address Don’t Care</td>
<td>!AVAX</td>
<td>200</td>
<td>250</td>
<td>300</td>
<td>450</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Address Valid to Output Valid</td>
<td>!AVOV</td>
<td>-</td>
<td>200</td>
<td>-</td>
<td>250</td>
<td>-</td>
<td>300</td>
<td>-</td>
<td>450</td>
</tr>
<tr>
<td>Chip Select Low to Data Valid</td>
<td>!SLQV</td>
<td>-</td>
<td>70</td>
<td>-</td>
<td>85</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>120</td>
</tr>
<tr>
<td>Chip Select Low to Output Don’t Care</td>
<td>!SLQX</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Chip Select High to Output High Z</td>
<td>!SHQZ</td>
<td>60</td>
<td>60</td>
<td>70</td>
<td>-</td>
<td>80</td>
<td>-</td>
<td>100</td>
<td>-</td>
</tr>
<tr>
<td>Address Don’t Care to Output High Z</td>
<td>!A XOZ</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Write Low to Write High</td>
<td>!WLWH</td>
<td>120</td>
<td>-</td>
<td>135</td>
<td>-</td>
<td>150</td>
<td>-</td>
<td>200</td>
<td>-</td>
</tr>
<tr>
<td>Write High to Address Don’t Care</td>
<td>!WHAX</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>-</td>
<td>20</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Write Low to Output High Z</td>
<td>!WLQZ</td>
<td>-</td>
<td>60</td>
<td>-</td>
<td>70</td>
<td>-</td>
<td>80</td>
<td>-</td>
<td>100</td>
</tr>
<tr>
<td>Data Valid to Write High</td>
<td>!DVWH</td>
<td>120</td>
<td>-</td>
<td>135</td>
<td>-</td>
<td>150</td>
<td>-</td>
<td>200</td>
<td>-</td>
</tr>
<tr>
<td>Write High to Data Don’t</td>
<td>!WHDX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**NOTES:**
1. A Read occurs during the overlap of a low ! and a high W.
2. A Write occurs during the overlap of a low ! and a low W.

**Figure 8-2.** Typical Data for MCM2114 Static RAM (Continued)
READ CYCLE TIMING (W HELD HIGH)

WRITE CYCLE TIMING (NOTE 3)

3. If the S low transition occurs simultaneously with the W low transition, the output buffers remain in a high-impedance state.

WAVEFORMS

Figure 8-2. Typical Data for MCM2114 Static RAM (Continued)
Figure 8-3. Expanded Multiplexed Bus Interface With MCM2114
8.2 PORT 3 PARALLEL INTERFACES (MODE 7)

There are many devices which utilize an 8-bit parallel interface. The required number, and type of control signals, varies depending upon the characteristics of the particular device. Some interfaces are passive and require no control signals. Other interfaces require a "Data Ready" line which signifies that valid data is present at the interface. Still others need an additional "Data Acknowledged" (or "Data Accepted") line which indicates that the device has accepted the last data and the interface can be given new data. The lines which are used to facilitate the dialogue between the control elements of two devices are called "handshake lines."

When operating in the Single Chip Mode, Port 3 functions as an 8-bit Input/Output port with two dedicated handshake control lines. It should also be noted that this capability exists only in Modes 7 and 4; in other modes, Port 3 is used as an address and/or data bus. Port 3 provides up to eight input or output lines as defined by its Data Direction Register. The two handshake lines, Input Strobe 3 (IS3) and Output Strobe 3 (OS3), provide the capability of implementing a simple synchronized interface. The two control lines function identically regardless of whether individual port bits are defined as inputs or outputs.

The Port 3 Control and Status Register provides state information and determines whether:
- input data presented to the Port 3 Data Register will be latched,
- OS3 will be generated by an MPU read or write to the Port 3 Data Register, and
- an IRQ1 interrupt will be enabled in response to an IS3 negative edge.

The Port 3 Control and Status Register contains a single status bit, IS3 FLAG. Its function is to indicate if an IS3 negative edge has been sensed. The bit can be polled or it can generate an IRQ1 interrupt if its interrupt enable bit is set. The IS3 FLAG bit is cleared by an MPU read of the Port 3 Control and Status Register (with IS3 FLAG set) followed by a read or write to its Data Register.

The following two applications illustrate hardware and software techniques involved in implementing a Port 3 interface in Single Chip Mode. The first example uses Port 3 to drive a line printer while the second illustrates driving the port from a keyboard.

8.2.1 Line Printer Interface to Port 3

A line printer is typical of many output devices which require both Data Ready and Data Accepted signals. To illustrate the details involved, this section discusses a parallel interface to a Model 306 Centronics line printer which uses two control lines. While this interface does not utilize all of the available features of the printer, it nevertheless illustrates a practical and realistic interface.

The interface, shown in Figure 8-4, provides eight data lines to the printer where the most significant bit (DATA8) is tied to ground. The printer, therefore, will receive 7-bit positive logic ASCII data with no parity (bit 7 is always a logic zero).
The most significant data bit in the Port 3 Data Register is configured as an input line and is used to determine the printer status. The motivation for mixing this single input with seven outputs is the result of an arbitrary decision to utilize only Port 3 resources. This interface does not, however, utilize all of the capabilities of the Model 306 line printer. The double character size cannot be used because DATA8 is tied to ground. The interface also does not provide more definitive fault conditions which are available from the printer.

A signal definition for the interface is shown in Figure 8-5. Assume that initialization has been completed and the interface is idle. Data is transferred across the interface as follows:

1. a byte is written to the Port 3 Data Register,
2. a strobe (i.e., pulse) from the MCU to the printer (Data Ready) is generated as a consequence of the MPU write to the Data Register,
3. after a data dependent delay, the printer responds with an Acknowledge pulse (or, Data Accepted), sets the Port 3 IS3 FLAG and indicates that the printer can accept more data, and
4. after the MCU clears IS3 FLAG, the interface is again idle.

The Output Strobe Select (OSS) bit in the Port 3 Control and Status Register is typically set during initialization if Port 3 is configured as an output data port. An output strobe (OS3) is then generated by an MPU write to the Port 3 Data Register.
The output current drive of Port 3 is not sufficient to directly drive the printer. An octal buffer (74LS244) is utilized for the seven output data lines and Output Strobe 3 (OS3) as shown in Figure 8-5 and provides the necessary drive capability.

The most significant data line of Port 3 (P37) is not used for data output. It is configured as an input and provides a printer status condition. A fault is indicated by checking bit 7 of the Port 3 Data Register which is normally high and goes low to indicate a printer fault. A printer fault can denote several conditions including "paper empty" and/or an "operator deselect."

A software driver routine for the interface is shown in Figure 8-6 and contains two entry points: one for initialization (PINZ) and another to write a byte to the printer (POUTC) from the A accumulator. In the initialization portion of the routine, all but bit 7 of the Port 3 Data Register are configured as outputs while bit 7 remains an input. Output strobe Select (OSS) is set which configures OS3 for a strobe-on-write.

The POUTC entry point is used to write a 7-bit ASCII character from the A accumulator where the value of bit 7 is of no consequence. To detect a printer fault, the calling routine can check the N-bit upon return, using either the BMI or BPL instruction. Polling is used in this example for illustrative purposes.

8.2.2 Keyboard Interface to Port 3

A keyboard with parallel output can be easily interfaced to Port 3 in Single Chip Mode. Port 3 functions as an Input/Output port in Single Chip Mode where each bit can be configured as either an input or an output. There are also two handshake control lines, IS3 and OS3, associated with it which can be used to simplify interface design. Output Strobe 3 (OS3) is controlled by the Output Strobe Select (OSS) bit and can be generated by either a read or write to the Port 3 Data Register. When configured as inputs, it is generally preferable for the strobe to be configured for a strobe-on-read (OSS = 0) of the Port 3 Data Register.

A typical keyboard provides parallel output data in ASCII format with a single control line which indicates when the data is valid. The keyboard output keeps the data latched for a given time period which is typically quite long with respect to MCU cycle times.
Figure 8-6. Line Printer Interface Driver: PINZ, POUTCH
This discussion illustrates interfacing a specific keyboard to Port 3: a Cherry Model B70-05AB. It is positive logic decoded and produces low outputs in the inactive state. When one of its keys is depressed, data is presented to the output lines in 7-bit ASCII format. After a data setup time of 25 microseconds, a 100 microsecond positive pulse is provided by the keyboard and indicates "Data Ready," as shown in Figure 8-7. Data is valid on both edges of Strobe and the negative edge is used in this example to latch the input data. The keyboard requires no reply and, therefore, does not use the Output Strobe (O3S) signal.

![Figure 8-7. Keyboard Interface Signals](image)

The Strobe output of the keyboard is connected to the Input Strobe 3 (IS3) as shown in Figure 8-8. During program initialization, the Port 3 LATCH ENABLE bit is set and the negative edge of the Strobe signal is used to both latch the keyboard data and set IS3 FLAG.

The IS3 FLAG bit is cleared by an MPU read of the Port 3 Control and Status Register followed by a read of its Data Register. This also makes the latch transparent again, but it should be noted that merely reading the Port 3 Data Register is sufficient to make the latch transparent.

A software driver routine for the interface is shown in Figure 8-9. The driver contains an entry point for initialization (KEYINZ) and for reading data from the port (KEYIN). Polling is used for illustrative purposes.

![Figure 8-8. Keyboard Interface Connection Diagram](image)
PAGE 001 KEYIN .SA:1 KEYIN *** KEYBOARD DRIVER ***

00001    NAM  KEYIN
00002    OPT  Z01,LLEN=80
00003    TTL  *** KEYBOARD DRIVER ***

00005

00006

00007    * K E Y I N -- KEYBOARD DRIVER ROUTINE

00008

00009    * THE ROUTINE SERVES AS A KEYBOARD

00010    * DRIVER FOR A PORT 3 INTERFACE TO

00011    * CHERRY KEYBOARD MODEL B70-05AB.

00012

00013    * THE DRIVER HAS TWO ENTRY POINTS:

00014

00015    * JSR KEYINZ NO ARGUMENTS; USES NO REGS

00016

00017    * JSR KEYIN RETURNS CHAR IN ACCA

00018

00019

00020

00021

00023    * E Q U A T E S

00025

00026

00027

00029A F980

00031

00032

00033

00035A F980 36

00036A F981 7F 0004 A

00037A F984 86 08 A

00038A F986 97 OF A

00039A F988 32

00040A F989 39

00042

00043

00044

00046A F98A 96 OF A

00047A F98C 2A FC F98A

00049A F98E 96 06 A

00050A F990 39

00052

TOTAL ERRORS 00000--00000

Figure 8-9. Keyboard Interface Drive: KEYINZ, KEYIN

8-13
In the initialization routine (KEYINZ), Port 3 is configured as an 8-bit input data port by clearing its Data Direction Register. The latching function is enabled by setting the Latch Enable bit in the Port 3 Control and Status Register.

In the data input routine (KEYIN), the subroutine waits until IS3 FLAG is set in the Port 3 Control and Status Register. Upon detecting the flag, the routine reads the Port 3 Data Register into the A accumulator and returns to the caller.

8.3 PRIORITIZED INTERRUPT VECTORS (MODES 1, 2, 3)

During processing of an IRQ1 interrupt, it is usually necessary at some point to ascertain which device of possibly several candidates is requesting service. While a routine which performs this task is usually trivial to implement (a "who-done-it" routine), the time to execute it adds directly to the interrupt response time. In some cases, this delay can result in an intolerably long response time.

It should be noted that the MC6801 IRQ2 Programmable Timer interrupts typically do not require a "who-done-it" routine because each interrupt is provided with a unique vector. The need for a "who-done-it" routine for IRQ1 interrupts can be deleted by adding external circuitry to provide for hardware prioritized interrupt vectors.

Before beginning a discussion of the methodology involved in implementing such a scheme, it is important to realize that: (a) this hardware is not necessary in many applications which can accomplish the task using software without any ill effects, and (b) the technique is applicable only to systems operating in the expanded multiplexed modes with external interrupt vectors (i.e., modes 1, 2, or 3).

8.3.1 General Considerations

The MC6801 uses memory mapped Input/Output and requires no specific Input/Output instructions. The communications technique embodied in this concept requires that the MPU put the "address" of the peripheral onto the Address Bus and the peripheral device (by listening to and decoding this "address") connects itself to the Data Bus when triggered by an appropriate "address." Note that a peripheral can be triggered by any number of "addresses" with this scheme if addresses are partially decoded.

The peripheral is then directed to "talk" or "listen" to the MPU as dictated by the level of the MPU R/W (Read/Write) line. It should also be noted that while any number of peripheral devices can "listen" to the MPU, only one can be permitted to "talk" at any given time.

While these points can appear to be irrelevant to the topic at hand, they are quite germane to the general concept behind the priority encoder circuitry. During an IRQ1 interrupt sequence, the MPU will generate the two addresses $FFFF8 and $FFFF9 on the address bus on two consecutive MPU E-cycles. The MPU will expect to receive a double byte address (high byte first) on the Data Bus as a response. The two byte address vector, supplied by the data bus, points to the location of the interrupt service routine. This vector can be supplied by any device capable of loading the Data Bus at the proper time with the appropriate two bytes.
The task of the priority encoder vector hardware, therefore, is to use the state of external IRQ1 inter-
terrupts as an input and, triggered by addresses $FFF8 and $FFF9 on the Address Bus, furnish two
bytes of the appropriate vector as a response to the Data Bus. The appropriate vector points to the
location of the service routine corresponding to the highest prioritized IRQ1 interrupt at the current
time.

8.3.2 8-Level Prioritizing Scheme

This section describes a scheme depicted in Figure 8-10 which provides eight IRQ1 hardware
prioritized interrupt vectors. The method is discussed as two tasks which accomplish (1) generating
the address of the appropriate vector and (2) providing the vector to the Data Bus.

8.3.2.1 GENERATING THE ADDRESS OF THE VECTOR. The IRQ1 interrupt request line
should be connected to the D-type flip-flop, as shown in Figure 8-10, with respect to priority: INT7
is the highest and INTO is the lowest priority interrupt. The flip-flop normally changes state on each
negative edge of E (Enable). The effect of this operation is to periodically latch external interrupts
which permits the priority encoder to perform its mapping function using stable inputs. The output
of the priority encoder is a 3-bit code which, after inverting, can be used as address lines A1-A3 of
the interrupt vector as shown in Figure 8-11. This process is repeated on each negative edge of E but
has no effect on system operation until the addresses $FFF8 or $FFF9 appear on the Address Bus.
Normally, this would occur only during the vector fetch cycles of the MPU IRQ1 interrupt se-
quenc. A timing diagram illustrating operation of the circuit is shown in Figure 8-12.

8.3.2.2 PROVIDING THE INTERRUPT VECTOR. The IRQ1 interrupt sequence begins after the
interrupt mask (I-bit) has been cleared and the level of any INT input to the priority encoder is low.
If this occurs, the GS output of the priority encoder is also forced low. Because this output is con-
nected to the MPU IRQ1 pin, an interrupt sequence will begin upon completion of the current in-
struction.

When address $FFF8 or $FFF9 is decoded on the Address Bus, the flip-flop is inhibited from chang-
ing state until after the vector fetch phase has been completed. The reason for this inhibit is to
preclude a higher priority interrupt from changing vectors between consecutive fetches of the two
bytes. This is accomplished by the OR-gate prior to the CLK input of the flip-flop as shown in
Figure 8-10.

The output of the IRQ1 address decoder is also connected to the SEL input of a data selector which
presents one of two sets of four inputs to the outputs. The set of inputs gated to the selector output
depends upon the level of the SEL input. In the absence of an IRQ1 interrupt, the SEL input is low
and the MC6801 normal Address Bus lines are passed through the data selector. When the address
decoder detects $FFF8 or $FFF9, however, the SEL input is high and the data selector connects the
priority encoder to the output. Having generated the address of the vector, all that remains is to
select the device containing the interrupt vector. In this example, the decoder output is also used as a
chip select for an MCM68708 EPROM which provides the two bytes of the interrupt vector.
Figure 8-10. 8-Level Priority Encoder
8.3.3 Final Remarks

Two concluding pertinent remarks are necessary before leaving this topic. First, it should be noted that the priority encoder circuitry will respond whenever $FFFF8 or $FFFF9 appears on the Address Bus. Normally, this will occur only during the interrupt sequence for an IRQ1 interrupt. It can also occur, however, during MPU reads or writes from location $FFFF8 or $FFFF9.

As an example, suppose that all eight external IRQ1 interrupt request lines are tied high. Further suppose that the LILbug monitor is used to read location $FFFF8. What location will be read during this access?

The priority encoder hardware will be activated when $FFFF8 appears on the Address Bus. When all external interrupts are high, the priority encoder will cause address $FFFF0 to be accessed. This same operation, however, will access a different location if any external interrupt (except INTO) is pulled low.

Finally, it should be noted that the last stage of logic prior to the EPROM in Figure 8-10 is an AND-gate which provides for selection of the device by an alternate decoding. The significance of this feature is that the interrupt vectors have two “addresses.” One address is the consequence of address decoding which results in activation of the SEL input and the other address is $FFFF8 or $FFFF9 mapped through the priority encoder. With this method of addressing, one can read the vectors using the address which generates SEL instead of $FFFF8 and $FFFF9. Furthermore, this dual addressing provides a method for using the entire EPROM. It is immaterial what address generates the SEL signal providing it does not conflict with addresses of other devices in the system.
Figure 8-12. Priority Encoder Timing
8.4 MEMORY AND ACIA INTERFACE (MODE 5)

The MCU Expanded Non-Multiplexed Mode provides a modest amount of external memory space while retaining significant on-chip resources. The expanded non-multiplexed bus is compatible with the M6800 family and this example illustrates typical techniques required to interface devices to it. The MC6801 non-multiplexed bus consists of the following MCU signals:

- an 8-bit bidirectional data bus (Port 3),
- up to eight address lines (Port 4),
- E (Enable),
- R/W (Read/Write), and
- IOS (Input/Output Select).

The expanded non-multiplexed bus is a synchronous bus clocked by E (Enable). IOS is active (low) whenever an address from $0100$ to $01FF$ appears on the internal address bus and indicates when an address in the 256-byte external memory space is being accessed. The eight address lines and IOS become valid during the interval when E is low and remain valid while E is high. The data bus becomes valid during the interval when E is high. The Read/Write line controls the direction of data bus transfers and is high during an MPU read. Timing for the bus is shown in Figure 8-13 and the symbols are quantitatively defined in the MC6801 Data Sheet.

An MCM6810 128-byte static RAM and an MC6850 Asynchronous Interface Adapter (ACIA) can be considered typical of devices which can be interfaced to the expanded non-multiplexed bus. In this example, they provide an additional 128 bytes of RAM and another full duplex serial port. The static RAM requires seven of the eight available address lines in order to access the entire RAM while the ACIA requires only a single address line to access its two internal locations. Address lines and IOS are also required to derive chip select signals.

![Expanded Non-Multiplexed Bus Timing](image)

Figure 8-13. Expanded Non-Multiplexed Bus Timing (Repeated)
A configuration which interfaces these two devices to the MCU expanded non-multiplexed bus is illustrated in Figure 8-14. The RAM responds to 128 addresses from $100 to $17F. Address line, A7, is used to select either the RAM (Low) or the ACIA (high). The RAM is enabled to the data bus only if all of the following conditions are true:

- E (Enable) is high,
- IOS (Input/Output Select) is low, and
- A7 is low.

The address is only partially decoded to generate a chip select for the ACIA and address line, A7, is used for this purpose. It will respond, therefore, to all addresses from $180 to $1FF. The ACIA is enabled to the bus when

- E (Enable) is high,
- IOS (Input/Output Select) is low, and
- A7 is high.

The ACIA transmitter and receiver require a bit rate clock and, in this example, the SCI provides this clock. Setting the CC1:CC0 field in the SCI Rate and Mode Control Register to 10 provides the SCI internal bit rate clock as an output at P22. The frequency is controlled by the SS1:SS0 field and the MCU input clock. The bit rate clock can be further divided (by 1, 16, or 64), by the ACIA and is controlled by the CR0 and CR1 bits in the ACIA Control Register.

All eight of the available address lines were used in this example. If fewer than eight are required, however, they can be used as additional input lines. Suppose, in this example, the RAM is replaced by a second ACIA. Two address lines and IOS would then be sufficient to decode the four locations and the remaining six Port 4 lines could be used as additional data input lines.

From Reset, the Port 4 Data Direction Register is cleared and the port is configured as an 8-bit parallel input data port. Any or all of its lines can be configured as address outputs (A0-A7) by setting the appropriate bits in the Port 4 Data Direction Register where bit 0 controls A0. No location in the external memory space can be accessed, however, until the Port 4 Data Direction Register has been configured.

**NOTE**

While 256 bytes of external read/write memory space are available in the Expanded Non-Multiplexed Mode, they cannot be accessed until the Port 4 Data Direction Register is configured by setting the desired bits.

### 8.4.1 Obtaining 256 Additional Bytes of Read-Only Memory

An additional 256 bytes of external read-only memory space can be obtained in the Expanded Non-Multiplexed mode by taking advantage of certain aspects of the MC6801 architecture. When the MCU is reading any address other than internal memory space, the Port 3 bus arbitrator directs the MPU to read from its external Data Bus. If the address being referenced is in the range of $100 to $1FF, Input/Output Select (IOS) becomes active to indicate an access in this range.
Figure 8-14. Memory and ACIA Interface in Expanded Non-Multiplexed Mode
If the MPU read is accessing a location not within $100$ to $1FF$ and also not an internal address, the MPU reads from its external Data Bus although this is not detectable external to the MCU. Figure 8-15 illustrates a method whereby the system designer can take advantage of this architecture in order to indirectly address an additional 256 bytes of Read-Only memory space. The address space for this area is (1) not in the address range $100$ to $1FF$ and (2) not an internal address. Note that $200$ to $2FF$, for example, fulfills these requirements.

In Figure 8-15, IOS is used as an additional address line although in other applications it could be used as a chip select qualifier. The PROM is selected when both (1) E (Enable) is high, and (2) Read/Write is high. The overall effect of this scheme is that the PROM will be selected on every bus cycle other than an MPU write cycle. When the MPU addresses its internal memory space, however, it will ignore its external data bus. If referencing other than internal memory space, it will read from the PROM.

8.5 PERIOD MEASUREMENT (ALL MODES)

The M6801 Programmable Timer can be used to measure the elapsed time between two edges of an input signal having TTL levels and a pulse width of at least two MPU E-cycles. There is no absolute limitation on the maximum time interval which can be measured but the minimum period is dependent upon the software response time in servicing the Input Capture Register. Specifically, the minimum period is dependent upon being able to read the Input Capture Register before it is overwritten by the next sequential capture.

The programming techniques involved in the measurement procedure depend upon whether the maximum interval can exceed the range of the free-running counter or 65,536 MPU E-cycles. If the elapsed time is of shorter duration, then the Counter Register overflow flag (TOF) need not be considered. If the input waveform is periodic, then the minimum period which can be reliably measured is 12 MPU E-cycles. If, however, the interval can exceed 65,535 MPU E-cycles, then the TOF bit must be used and the minimum period which can be measured is on the order of 30 MPU E-cycles.
8.5.1 Measuring Periods Less than 65,536 Cycles

Period measurement of intervals not exceeding the 16-bit range of the free-running counter can be obtained with a minimum of software effort. This discussion assumes reader familiarity with the following MC6801 subsystems:

1. IEDG bit in the Timer Control and Status Register (TCSR),
2. ICF flag bit in the TCSR, and
3. Input Capture Register of the Programmable Timer.

Each time the input capture edge detector senses a transition defined by the IEDG bit, the contents of the free-running counter are transferred to the Input Capture Register and the ICF bit is set. These two events occur regardless of whether the ICF bit has been cleared between successive transitions. The Input Capture Register, therefore, always contains the value of the free-running counter at the time of the last transition.

To measure the interval between two successive input captures, the initial Counter Register value (T0) must be obtained and then the next successive capture must be serviced before being overwritten by any following edge. If the direction of desired transition is not the same between the two successive edges then the IEDG bit must also be toggled between captures.

The general algorithm for a measurement between like transitions which do not exceed 65,535 MPU E-cycles is as follows:

1. Read the Timer Control and Status Register followed by the Input Capture Register. This obtains the initial counter value (T0) and also clears the Input Capture Flag (ICF).
2. When ICF is set for the next transition, read the Input Capture Register for the second counter value (T1).
3. Subtract the first capture value (T0) from the second capture value (T1). The resultant 16-bit unsigned value (T1 - T0) is the interval in E-cycles.

The minimum interval which can be measured using the input capture function is dependent upon the timing associated with a loop which polls the Input Capture Flag (ICF) and, when set, immediately loads the Input Capture Register. This interval is 12 MPU E-cycles, as shown in Figure 8-16 where the instruction sequence is shown across the top of the figure. The figure illustrates the worst case timing incurred by the polling loop and subsequent MPU read of the Input Capture Register. If waveforms with shorter intervals are presented to the program, the results will indicate an integer multiple of the actual period due to more than one input capture occurring prior to service completion.

While the period measurement routine can be interrupt-driven, the minimum period will be in excess of 12 MPU E-cycles due to the time taken to complete the interrupt sequence (see Chapter 5). It is left to the reader to determine this minimum period (i.e., the worst case response time) as it depends upon the exact method of implementation.
8.5.2 Measuring Periods Exceeding 65,535 Cycles

If the interval between transitions can exceed the 16-bit range of the free-running counter, then the overflow flag (TOF) must be taken into account in order to determine the period. In order to understand this procedure, the reader must be familiar with:

1. Timer Control and Status Register (TCSR),
2. IEDG bit in the TCSR,
3. ICF flag bit in the TCSR,
4. Input Capture Register, and
5. TOF flag bit in the TCSR.

The general equation used to obtain the time interval is:

\[ T = N \times 65,536 + (T_1 - T_0) \]

where

- \( T \) = time between input captures in MPU E-cycles,
- \( N \) = number of times the overflow flag is set between input captures,
- \( T_1 \) = value obtained from the second input capture, and
- \( T_0 \) = value obtained from the first input capture.

It is important to note that by maintaining an overflow counter in the most significant byte(s) of an n-byte value (where n exceeds 2) one effectively performs the multiplication: \( N \times 65,536 \).

The resultant period for this computation will typically exceed a double byte value. The following algorithm arbitrarily assumes a 24-bit period but is valid with larger values by simply extending the basic concept. The following algorithm is presented in its general form. However, there are two special cases which must be dealt with, and three assumptions which permit treatment of these two special cases. Both of these topics will be discussed after presenting the following general algorithm.

1. Allocate a 3-byte location for the final result. The most significant byte serves as the overflow counter and the remaining two bytes are used first as temporary storage for \( T_0 \) and finally as the least two significant bytes of the result.

Figure 8-16. Minimum Period Measurement Using MC6801 Timer
2. Clear the Input Capture (ICF) and Timer Overflow (TOF) flags.

3. Save the first input capture (T0) in the two low order bytes of the result area (temporary storage).

4. Increment the software overflow counter each time TOF is set between the first and second capture.

5. Subtract the first capture (T0) from the second capture (T1) and overwrite the temporary area with the result.

6. If the borrow bit is set, decrement the overflow counter. The overflow counter concatenated with the two byte temporary value is the 24-bit result in MPU E-cycles.

Problems with the above general algorithm result when the input capture occurs within the vicinity of Counter Register rollover. "Vicinity" conditions exist when both the ICF and TOF flag bits are found set when servicing the Timer Control and Status Register. For this condition, the problem is to determine whether or not the overflow should be included in the total overflow count. This condition can exist during capture of either T0 or T1 and leads to a pair of special cases.

The following three assumptions permit handling of these two special cases:

1. TOF is cleared before servicing the first input capture,
2. TOF is serviced while the Counter Register is between $0000$ and $7FFF$ or, equivalently, within 32,768 MPU E-cycles after TOF is set, and
3. the input capture is serviced before a Counter Register overflow if both are pending.

The two special cases are illustrated in Figure 8-17 where input captures for T0 and T1 are depicted which occur in the vicinity of Counter Register rollover. The arc depicts a segment of a closed circular unsigned integer number line with a range from $0000$ to $FFFF$ in a clockwise direction. The symbol, TOF, represents the integer $FFFF$. In both examples, reading the TCSR reveals that both the ICF and TOF flags are set and it is the task of the software to resolve whether or not to increment the software overflow counter. The most significant bit of the value in the Input Capture register can be used to resolve this difficulty if one complies with the three assumptions indicated above.

Figure 8-17(a) depicts two possible input captures of T0 in the vicinity of Counter Register rollover. If bit 15 of the captured value is a "1" (case 2), then the capture took place before the Counter Register rollover and the overflow should be counted in the total. If the most significant bit is a "0" (case 1), then the rollover occurred before the capture and the overflow should not be counted.

Figure 8-17(b) depicts two possible input captures of T1 in the vicinity of Counter Register rollover. A procedure analogous to the scheme illustrated above can be employed except the opposite conditions indicate whether or not to include the Counter Register rollover. If bit 15 of the captured value is a "0" (case 1), then the rollover occurred before the capture and the overflow should be included in the total count. If bit 15 of the captured value is a "1" (case 2), then the capture took place first, followed shortly by Counter Register rollover and the rollover should not be counted in this condition.

The PER24 and TSTAT subroutines of Figure 8-18 implement an interval measurement procedure using the considerations discussed in this section. The PER24 routine will return an interval from 30 to $16,777,215$ MPU E-cycles. This routine, in turn, calls a driver routine, TSTAT which services the Input Capture register and overflow counter. Both routines are non-reentrant and use a 3-byte location called "NUM" to store the result.
8.5.3 Period Measurement Sample Programs

Two complete sample programs are presented in this section which utilize the routines described in the previous two sections. The first program, TIM24, uses the PER24 and TSTAT routines to obtain a period from 30 to 16,777,215 MPU E-cycles (24-bits). The results are then formatted and displayed on a teleprinter using LILbug monitor output routines.

The program is presented in Figure 8-18 and can be exercised after connecting a signal generator (square wave, 0-5 V) to the MC6801 Input Capture input (pin 8). The program continuously measures and displays the period of the input waveform in an easily readable format on the teleprinter. The formatting routines, SHOW24, BUFFER, and FLUSH, transform the resultant 24-bit period into an 8-digit decimal number representing the period in units of MPU E-cycles.

An example of output from the program is shown in Figure 8-19 and indicates a period of 1000 (decimal) MPU E-cycles. The program is not functionally affected by the speed of the teleprinter.

The second program, TIM16, utilizes the same formatting routines as TIM24 and is shown in Figure 8-20. Only the main line and PER16 are shown. The formatting routines were presented in Figure 8-18. The TIM16 program can be exercised in the same manner as TIM24 except that the maximum period can not exceed 65,535 MPU E-cycles. The format of the output from the program is identical to that presented in Figure 8-19.
Figure 8-18. Period Measurement Sample Program: TIM24
**M6801 PERIOD MEASUREMENT ROUTINE**

**Calling Arguments** -- None

**Returns** -- A 24-BIT VALUE IN LOCATION "NUM" DEFINED AS:

NUM RMB 3 PERIOD & OVERFLOW COUNTER

**Assumptions** --

THE OVERFLOW FLAG WILL BE SERVICED WITHIN 32,768 CYCLES AFTER THE FLAG IS ASSERTED (WHILE THE TIMER VALUE IS POSITIVE)

THE OVERFLOW FLAG WILL BE SERVICED BEFORE THE FIRST INPUT CAPTURE IS SERVICED.

IF THE OVERFLAG FLAG AND INPUT CAPTURE FLAGS ARE ASSERTED SIMULTANEOUSLY, THE INPUT CAPTURE WILL BE SERVICED FIRST.

---

**Program Listing**

```assembly
0084A 111F DC 08 A PER24 LDD TCSR CLEAR OUT TOF AND ICF
0085A 1121 96 0D A LDAA INCAP CLEAR ICF
0087 * GET 1ST EDGE (TO)
0089A 1123 8D 24 1149 BSR TSTAT STATUS TIMER & WAIT
0090A 1125 7F 101E A CLR NUM CLEAR THE OVERFLOW COUNTER
0091A 1128 FF 101F A STX TO SAVE
0092A 112B 2B 02 112F BMI PER002 IT'S OK
0093A 112D 96 09 A LDAA TIMER SIGN IS +, CLEAR TOF
0095 * NOW GET 2ND EDGE (T1)
0097A 112F 8D 18 1149 PER002 BSR TSTAT WAIT FOR 2ND CAPTURE
0098A 1131 3C PSHX SAVE IT
0099A 1132 2B 07 113B BMI PER004 WAS T1 MINUS?
00A0A 1134 85 20 A BITA #$20 CHECK FOR OVERFLOW
00A1A 1136 27 03 113B BEQ PER004 IT'S OK
00A2A 1138 7C 101E A INC NUM ONE MORE UPDATE
00A3A 113B 32 PER004 PULA PULL OFF HIGH BYTE
00A4A 113C 33 PULB PULL OFF LOW BYTE
00A5A 113D 83 101F A SUBD TO COMPUTE PERIOD
00A6A 1140 FD 101F A STD NUM+1 LOWER TWO OK NOW
00A7A 1143 24 03 1148 BCC PER006 CHECK BORROW BIT
00A8A 1145 7A 101E A DEC NUM ALL 24-BITS OK NOW
00A9A 1148 39 PER006 RTS
```

---

Figure 8-18. Period Measurement Sample Program: TIM24 (Continued)
PAGE 003 TIM24 .SA:1 TIM24 ***** M6801 PERIOD MEASUREMENT ROUTINE ****

00111
00112 *
00113 * TSTAT -- ROUTINE TO STATUS THE M6801 TIMER
00114 *
00115 * CALLING ARGUMENTS -- NONE
00116 *
00117 * RETURNS -- VALUE OF INPUT CAPTURE REGISTER IN X AND
00118 * A--ACCUMULATOR CONTAINS LAST TIMER STATUS
00119 *
00120 * THE ROUTINE SERVICES THE OVERFLOW FLAG (TOF) AND USES
00121 * A 3-BYTE MEMORY AREA DEFINED AS
00122 *
00123 * NUM RMB 3
00124 *
00125 ***************************************************

00127A 1149 96 08 A TSTAT LDAA TCSR WAIT ON FLAGS
00128A 114B 2B 0B 1158 BMI TSTAT1 THAT'S IT
00129A 114D 85 20 A BITA #$20 MAYBE JUST OVERFLOW
00130A 114F 27 F8 1149 BEQ TSTAT NO, NOTHING

00132A 1151 96 09 A LDAA TIMER CLEAR TIMER OVERFLOW FLAG
00133A 1153 7C 101E A INC NUM BUMP COUNTER
00134A 1156 20 F1 1149 BRA TSTAT DO IT ALL AGAIN

00136A 1158 DE 0D A TSTAT1 LDX INCAP RETURN CAPTURE VALUE IN X
00137A 115A 39 RTS SPLIT BACK TO USER

Figure 8-18. Period Measurement Sample Program: TIM24 (Continued)
Figure 8-18. Period Measurement Sample Program: TIM24 (Continued)
TIM24 PERIOD MEASUREMENT ROUTINE

00197A 11B5 08 INX
00198A 11B6 08 INX
00199A 11B7 08 INX LOOK AT NEXT VALUE
00200A 11B8 8C 1171 A CPX #SHW10+21 MAYBE THAT’S ALL
00201A 11BB 26 C0 117D BNE SHOW02 NOT YET

00203A 11BD B6 1020 A LDA NUM+2 GET 1’S DIGIT
00204A 11C0 8B 30 A ADDA #$30 CONVERT TO ASCII
00205A 11C2 2D 11D7 A JSR BUFFER WRITE LAST ONE TO BUFFER
00206A 11C5 8D 1C 11E3 BSR FLUSH NOW FLUSH THE BUFFER

00208 * MAYBE ISSUE ANOTHER CR/LF

00210A 11C7 7A 102E A DEC SHWKT HOW MANY?
00211A 11CA 27 01 11CD BEQ SHOW06 TIME FOR ANOTHER CRLF
00212A 11CC 39 RTS AND RETURN

00214A 11CD B6 115B A SHOW06 LDAA SHWKT RE-INZ NUMKT
00215A 11DO B7 102E A STAA SHWKT
00216A 11D3 8D F818 A JSR PCLKF ISSUE CR/LF
00217A 11D6 39 RTS

Figure 8-18. Period Measurement Sample Program: TIM24 (Continued)
*** M6801 PERIOD MEASUREMENT ROUTINE ***

BUFFER PSHX SAVE X-REGISTER

BUFFER -- ROUTINE TO BUFFER 8 DIGITS BEFORE PRINTING

CALLING ARGUMENTS -- NONE

Routine uses an 11-byte area labeled as SHWBUF and a
2-byte pointer labeled as SHWPTR

ROUTINE USES AN II-BYTE AREA LABELED AS SHWBUF AND A 2-BYTE POINTER LABELED AS SHWPTR

*********************************************************
* BUFFER
* ROUTINE TO BUFFER 8 DIGITS BEFORE PRINTING
* CALLING ARGUMENTS -- NONE
* ROUTINE USES AN 11-BYTE AREA LABELED AS SHWBUF AND A 2-BYTE POINTER LABELED AS SHWPTR
*********************************************************

*********************************************************
* FLUSH - STRIP LEADING ZEROES AND PRINT
* CALLING ARGUMENTS -- NONE
* ROUTINE EXPECTS 3-BYTE ASCII DECIMAL VALUE STARTING AT "SHWBUF"
*********************************************************

FLUSH PSHX SAVE X-REGISTER

FLUSH -- ROUTINE TO BUFFER 8 DIGITS BEFORE PRINTING

CALLING ARGUMENTS -- NONE

ROUTINE USES AN 11-BYTE AREA LABELED AS SHWBUF AND A 2-BYTE POINTER LABELED AS SHWPTR

ROUTINE USES AN II-BYTE AREA LABELED AS SHWBUF AND A 2-BYTE POINTER LABELED AS SHWPTR

*********************************************************
* FLUSH -- ROUTINE TO BUFFER 8 DIGITS BEFORE PRINTING
* CALLING ARGUMENTS -- NONE
* ROUTINE USES AN 11-BYTE AREA LABELED AS SHWBUF AND A 2-BYTE POINTER LABELED AS SHWPTR
*********************************************************

*********************************************************
* FLUSH -- ROUTINE TO BUFFER 8 DIGITS BEFORE PRINTING
* CALLING ARGUMENTS -- NONE
* ROUTINE USES AN 11-BYTE AREA LABELED AS SHWBUF AND A 2-BYTE POINTER LABELED AS SHWPTR
*********************************************************

TOTAL ERRORS 00000--00000

Figure 8-18. Period Measurement Sample Program: TIM24 (Continued)
8.6 SCI PARALLEL INTERFACES (ALL MODES)

To derive the advantages of a particular MC6801 configuration, it may be necessary to decrease the number of required MCU Input/Output lines. If the SCI is not required by the application then perhaps some parallel data could be converted to serial format and interfaced with the SCI.

The two interfaces presented in this application illustrate techniques for converting 8-bit parallel data to serial format and interfacing it with the SCI. The first example represents details for implementing an 8-bit parallel-to-serial input interface whereas the second example discusses an 8-bit serial-to-parallel output interface. Both interfaces utilize the SCI output bit rate clock and NRZ format which are obtained by setting the CCI:CCO field of the Rate and Mode Control Register to 10. The bit rate is controlled by the SS1:SS0 field of the Rate and Control Register and the MCU input frequency.

8.6.1 SCI Parallel-to-Serial Input Interface

A keyboard with a parallel output is a typical device which can be interfaced with the SCI. Generally, its output consists of 8-bit parallel data and a Strobe to indicate when its output is valid. The interface must then capture the data by a specified length of time.

Two keyboard characteristics make this type of interface very attractive. First, the data rate is very slow compared to the bit rates achievable with the SCI. Finally, the required control signals are minimal: the keyboard merely uses its Strobe output to announce when it has valid data, holds it for a specified period, and needs no reply. The main timing restriction is that the keyboard repeat function must be adjusted in order not to generate data faster than the 10 bit times required by the interface to transmit a byte to the SCI.

Design considerations for a parallel-to-serial interface to the SCI include the following:

- both a Start bit and a Stop bit are required by the SCI and must be provided by the interface,
- the output of the shift register must be a logic high (mark) whenever the line is idle,
- data transfers through the interface must be synchronized with the SCI bit rate clock, and
- another MCU output line may be required if the device requires a reply from the interface such as Not Busy or Data Acknowledge signals. These signals are not readily available from the simple interface described in this example.
Figure 8-20. Period Measurement Sample Program: TIM16
**** M6801 PERIOD MEASUREMENT ROUTINE ****

**Calling Arguments** -- None

**Returns** -- A 16-BIT VALUE in LOCATION NUM+1:NUM+2

**Assumptions** -- Period will not exceed 65,535 cycles and service time will not exceed the period.

```
00059  A PER16  LDAA  TCSR  CLEAR OUT THE FLAG
00060  A PER001  LDAA  TCSR  WAIT ON #2
00061  A PER001  LDAA  TCSR  WAIT ON #2
00062  BPL  PER001  NOT YET
00063  A LDD  INCAP  GET THE VALUE
00064  A STX  TO  STASH FIRST VALUE
00065  A SUBD  TO
00066  A STD  NUM+1  ALL DONE
00067  RTS  RETURN TO CALLER
```

Figure 8-20. Period Measurement Sample Program: TIM16 (Continued)
A schematic diagram for a parallel-to-serial interface is presented in Figure 8-21. Key elements in the interface include three D-type flip-flops (U1, U2, and U4) and a parallel-load shift register (U3). The three flip-flops generate the Start and Stop bit in addition to providing synchronization with the SCI bit rate clock. The shift register is used to latch the parallel output from the keyboard and to convert it to SCI compatible serial format.

Data transfer is initiated by a positive edge of the Strobe signal. After synchronizing with the bit rate clock using D-type flip-flops, U1 and U2, the Start bit is transmitted (by clearing U4), and data is latched into the parallel-load shift register, U3. Data is clocked through the shift register on the positive edge (at mid-bit time) of the SCI bit rate clock until all eight data bits have been transmitted. As data is clocked out of the shift register, it is filled with 1’s. This provides the Stop bit and “marks” the line until the next byte is strobed into the interface.

8.6.2 SCI Serial-to-Parallel Output Interface

Some parallel output devices, such as displays, are candidates for interfacing with the SCI. The interface must convert the SCI serial format to parallel and, if necessary, provide a Strobe to indicate Data Ready. With respect to this type of interface, it is also beneficial if the output device cannot be overrun by data, which alleviates the interface from having to provide a Data Accepted signal. This signal is not difficult to generate but requires another MCU input line.

This discussion presents details for implementing an output serial-to-parallel interface which is illustrated in Figure 8-22. Key elements in the interface include two D-type flip-flops, U3 and U4, an 8-bit parallel output shift register, U1, and an octal D-type flip-flop, U2. The Rate and Mode Control Register must be configured at the desired rate and the CC1:CC0 field set to 10.

The RESET line is used to clear the shift register and force the Strobe line high. Pullup resistors on the output of P22 and P24 ensure that the lines remain high until software configures the SCI.

The quiescent state of the interface is a high for P24 and the Strobe, and all zeroes in the shift register. Note that the serial output is inverted before entering the shift register. This inversion is due to the shift register capability of being cleared (but not set) which provides a means to reinitialize the interface after each byte is received.

Data transfer begins when a Start bit appears at the output of P24. It is inverted to a “1”, shifted through U1, and appears at the input of U3 at the 8th mid-bit time. Note that data is clocked through the shift register by the positive edge of the bit rate clock (mid-bit time). On the ninth mid-bit time, the start bit is clocked into flip-flop, U3, and all eight bits of data are contained in the shift register. The Start bit is used to enable octal latch, U2, and on the next negative edge of the SCI bit rate clock (one-half bit time later), the output of shift register is captured by octal latch U2. A one bit time wide negative-going strobe is generated at the output of flip-flop U4 during the tenth bit time where either edge can be used as a Data Ready signal. The Strobe is also used to clear the shift register (MR is low) and the interface is again ready for data.

The interface can accept data at the maximum SCI rate. The bit rate should be chosen carefully, however, because the interface does not provide a Data Accepted line. Data must be captured from the output latch, U2, during the next ten bit times or it can be overwritten by the following byte.
Figure 8-21. SCI Parallel-to-Serial Interface
Figure 8-22. SCI Serial-to-Parallel Interface
8.7 DUAL PROCESSOR PARALLEL INTERFACES (MODE 7)

Applications which utilize distributed processing have become more common in recent years. As technology continues to make this concept more attractive, it is probable that microcomputer networks will be used to a greater extent. Communication between processors involves both data transfer and control elements. In such networks, this interaction can become troublesome to the designer. The MC6801 offers several alternatives with which to implement a processor-processor interface. The options available depend upon the MCU operating mode and include:

1. the Serial Communications Interface (SCI),
2. an MC6850 Asynchronous Communications Interface Adapter (ACIA),
3. an MC6820/1 Peripheral Interface Adapter (PIA),
4. an MC6846 ROM, I/O, Timer Unit, or
5. the Port 3 8-bit parallel handshaking data port.

The SCI can be used in all operating modes and has the advantages of an inherent control scheme and a noise and speed tolerant Bi-Phase format. All of the remaining alternatives, except when using Port 3, require an external bus for interfacing to another part. The ACIA and SCI provide a serial interface whereas all of the others employ a parallel interface.

The last alternative in the above list is available only in Single Chip mode and is the subject of this discussion. It utilizes Port 3 to implement a parallel interface between two MC6801s. Two associated control lines make this port better suited for this application than either Port 1 or Port 4.

Data width and manipulation of control signals must be considered when designing a dual-processor parallel interface. Following a discussion of control mechanisms, three interface schemes are presented where the last two differ only in the method of control:

1. 8-bit half duplex,
2. 4-bit full duplex using
   (a) Input Capture function, and
   (b) External Exclusive-OR function.

8.7.1 Interface Control Schemes

A major concern in the interface design is the avoidance of bus contention. The control system must ensure that only one of any pair of interconnected Port 3 lines can be configured as an output at any given time. Two Port 3 associated lines, IS3 and OS3, are adequate to control simple designs such as simplex (unidirectional) interfaces. A pre-defined sequence, or protocol, involving bidirectional transfers can also depend solely on IS3 and OS3 for control. In the latter case, the significance of either strobe depends upon the protocol.

In more general designs, however, additional control lines must be provided from available unused I/O port lines. If an interrupt-capable interface is desired, it must use specific MCU pins in order to generate the interrupt. Possible Candidates include IS3, IRQ1, NMI, and the Input Capture pins.

Furthermore, service for an interrupt-capable interface is simplified if each line generates an individual interrupt. If several input request lines can generate the same interrupt, a readable status bit is typically required for identification of the requestor.

8-39
An interrupt-driven interface has the advantage of allowing the MPU to perform other tasks while interface service is not required. Polling, however, provides the maximum data transfer rate across the interface.

Timing dependencies between two MCUs should be avoided. No software assumptions should be made as to the time required for the other MCU to service the interface. An MPU with Port 3 configured as outputs can be called a “talker” while its counterpart is configured as a “listener.” An output strobe from the “talker” indicates to the “listener” that its data is valid and the “talker” must then wait for receipt of an acknowledgement from the “listener” before writing more data.

There are several schemes which can be used to reverse the direction of 8-bit transfers through a Port 3 parallel interface. One such method allows an MCU to be the “talker” until the other MCU requests to talk whereupon it relinquishes control to the other MCU when convenient. This scheme is well suited for applications where the dialogue typically alternates between the two MCUs. The “talker” is necessarily the controller (or Master) of this type of interface because of having to avoid bus contention. The “talker” must change its output lines to inputs before the new “talker” can reconfigure its input lines to outputs. The 8-bit half duplex example illustrates an implementation of these concepts.

In some applications, data transfer is predominately unidirectional and it may be undesirable for an MCU to relinquish control for an extended period. In this case, the Master MCU can grant a Slave permission to transmit a fixed number of bytes after which it must relinquish control of the interface. An example of this interface scheme is not included in this section but it can be considered a hybrid design of the other schemes which are discussed.

Dedicating four Port 3 lines as outputs and four as inputs resolves all bus contention problems and saves time which would otherwise be spent reconfiguring the port. If the data is byte-oriented, however, additional processing is required to divide and rejoin each byte before and after transfer. This method of interface is well suited to those applications where both MCUs must converse at the same time and it is also intolerable to wait for control of the interface. Two examples are included which illustrate this 4-bit Full Duplex method.

Although there are numerous methods, the following discussion presents two different approaches in designing a dual processor interface using Port 3. The second approach is presented with two variations. Note that the maximum data transfer rate is very dependent upon the selection and use of a particular method.

### 8.7.2 8-Bit Half Duplex Interface

Alternating control and direction of the interface between two MCUs can be used to implement an 8-bit half duplex Port 3 interface. Upon initialization, one MCU is configured as a “talker” and the other as a “listener.” When the “listener” MCU desires to talk, it requests to become the “talker” while continuing to accept data until the request is granted. The “talker” can write to Port 3 while periodically checking for a request from the “listener.” If a request is present and the “talker” wishes to relinquish control of the interface, it reconfigures itself as a “listener” (i.e., as inputs) and then grants the request. After detecting the grant, the “listener” MCU reconfigures itself to a “talker” (i.e., outputs) and becomes the controller.
A typical configuration for this alternating controller scheme uses four interface control lines and is depicted in Figure 8-23. The meaning attached to the Port 3 Output Strobe 3 (OS3) signal depends upon the state of the two MCUs. Under various conditions, OS3 can indicate Data Ready, Data Acknowledged, or Interface Reconfigured and is connected to the opposite MCU Input Strobe.

![Figure 8-23. Half-Duplex Interface](image)

A data port line in each MCU is configured as an output and is used for an interface Request/Grant signal. The "talker" (or controller) must monitor the Request/Grant signal of the "listener." The relationship of the MCU "talker-listener" status and the level of the Request/Grant line is shown in Figure 8-24. Because both MCUs have identical configurations, both can use the same low-level software.

![Figure 8-24. Request/Grant Control Signals](image)

Software routines which implement this scheme are shown in Figure 8-25 where it is assumed that the calling routines utilize polling. The "no wait" feature in the IN3NW, OUT3NW, and CHKREQ subroutines ensures that control returns promptly to the caller regardless of the outcome of the operation and the carry bit can be tested to determine what action, if any, was taken. If the carry bit is set, the transaction was successfully completed; otherwise, another attempt can be made.
*** HALF DUPLEX INTERFACE ROUTINES***

NAM P3IO
OPT ZO1,LLEN=80
TTL *** HALF DUPLEX INTERFACE ROUTINES***

DUAL PROCESSOR PARALLEL HALF DUPLEX INTERFACE ROUTINES

TAKINZ -- INITIALIZE TALKER MCU TO TALKER STATE
NO CALLING ARGUMENTS.

LISINZ -- INITIALIZE LISTENER MCU TO LISTENER STATE. NO CALLING ARGUMENTS.

OUT3NW -- OUTPUT BYTE IN ACCA TO PORT 3 DATA REGISTER IF LAST TRANSMISSION ACKNOWLEDGED AND RETURN WITH C-BIT SET; ELSE RETURNS WITH C-BIT CLEAR. RESTORES ACCB.

NOTE: OUT3NW MAY BE ENTERED ONLY IF MCU IS IN TALKER STATE.

CHKREQ -- CHECK IF OTHER MCU IS REQUESTING TO TALK. IF NO REQUEST, IT RETURNS WITH THE C-BIT CLEAR; ELSE IT SETS UP CONTROL REGISTER AND PORT 3 FOR LISTENER STATE, STROBES OTHER MCU AND RETURNS WITH THE C-BIT SET. NO CALLING ARGUMENTS. RESTORES ACCB.

NOTE: CHKREQ MAY BE ENTERED ONLY IF MCU IS IN TALKER STATE.

IN3NW --- GET BYTE FROM PORT 3. IF A BYTE IS AVAILABLE, IN3NW RETURNS IT IN ACCA WITH THE C-BIT SET; OTHERWISE, THE C-BIT IS CLEAR. NO CALLING ARGUMENTS.

NOTE: IN3NW MAY BE ENTERED ONLY IF MCU IS IN THE LISTENER STATE.

REQ ------ RETURNS WITH MCU IN THE TALKER STATE.
WHILE WAITING FOR A GRANT, INPUT DATA FROM PORT 3 IS RECEIVED AND PROVIDED TO USER ROUTINE. NO CALLING ARGUMENTS.
USES ACCA AND ACCB.
REQ CALLS USER SUBROUTINE INBYTE WHICH TAKES A BYTE FROM ACCA AND RETURNS.

NOTE: REQ MAY BE ENTERED ONLY IF MCU IN LISTENER STATE.

********************************************************************

Figure 8-25. Half Duplex Routines
* EQUATES

00061 0005 A P4DDR EQU $05 PORT 4 DATA DIRECTION REG
00062 0007 A P4DATA EQU $07 PORT 4 DATA REG
00063 0004 A P3DDR EQU $04 PORT 3 DATA DIRECTION REG
00064 0006 A P3DATA EQU $06 PORT 3 DATA REG
00065 000F A P3CNTL EQU $0F PORT 3 CONTROL & STATUS REG

* TAKINZ -- INITIALIZE TALKER

00068

00069A F800 ORG $F800
00070A F800 86 10 A TAKINZ LDAA #$10
00071A F802 97 0F A STAA P3CNTL STROBE ON WRITE
00072A F804 86 FF A LDAA #$FF
00073A F806 97 04 A STAA P3DDR PORT 3 OUTPUT
00074A F808 86 02 A LDAA #$02
00075A F80A 97 07 A STAA P4DATA INITIALLY TALKER
00076A F80C 86 02 A LDAA #$02
00077A F80E 97 05 A STAA P4DDR BIT 0 HANDSHAKE FROM OTHER MCU
00078A F810 39 RTS BIT 1 HANDSHAKE TO OTHER MCU
00079

* LISINZ -- INITIALIZE LISTENER

00081

00082A F811 86 08 A LISINZ LDAA #$08
00083A F813 97 0F A STAA P3CNTL STROBE ON READ, LATCH DATA
00084A F815 4F CLR
00085A F816 97 04 A STAA P3DDR PORT 3 INPUT
00086A F818 97 07 A STAA P4DATA INITIALLY LISTENER
00087A F81A 86 02 A LDAA #$02
00088A F81C 97 05 A STAA P4DDR BIT 0 HANDSHAKE FROM OTHER MCU
00089

BIT 1 HANDSHAKE TO OTHER MCU

00090A F81E 96 0F A LDAA P3CNTL CLEAR ANY FALSE FLAG
00091A F820 96 06 A LDAA P3DATA STROBE TALKER MCU
00092A F822 39 RTS

* OUT3NW -- OUTPUT BYTE TO PORT 3

00095

00096A F823 37 OUT3NW PSHB
00097A F824 D6 0F A LDAB P3CNTL IS3 FLAG TO BE EXAMINED
00098A F826 58 ASLB
00099A F827 24 02 F82B BCC OUT3EX LAST TRANS NOT ACKNOWLEDGED YET
00100A F829 97 06 A STAA P3DATA OUTPUT TO PORT 3
00101A F82B 33 OUT3EX PULB
00102A F82C 39 RTS CARRY TO BE TESTED ON RETURN
00103

* CHKREQ -- CHECK IF LISTENER MCU IS REQUESTING

00105

* TO TALK.

00106

00107A F82D 37 CHKREQ PSHB SAVE ACCB
00108A F82E D6 07 A LDAB P4DATA
00109A F830 57 ASRB
00110A F831 24 11 F844 BCC CHKREX NO REQUEST
00111A F833 7F 0004 A CLR P3DDR PORT 3 INPUT
00112A F836 C6 08 A LDAB #$08
00113A F838 D7 0F A STAB P3CNTL STROBE ON READ, LATCH
00114A F83A D6 0F A CHKRO1 LDAB P3CNTL
00115A F83C 2A FC F83A BPL CHKRO1 WAIT TIL LAST TRANS ACKNOWLEDGED
00116A F83E 7F 0007 A CLR P4DATA GRANT REQUEST

Figure 8-25. Half Duplex Routines (Continued)
*** HALF DUPLEX INTERFACE ROUTINES***

00117A F841 D6 06 A LDAB P3DATA STROBE NEW TALKER
00118A F843 0D SEC MCU NOW IN LISTENER STATE
00119A F844 33 CHKREX PULB RESTORE ACCB
00120A F845 39 RTS

00121

* IN3NW -- INPUT BYTE FROM PORT 3

00123

00124A F846 96 0F A IN3NW LDA P3CNTL IS3 FLAG TO BE EXAMINED
00125A F848 48 ASLA
00126A F849 24 02 F84D BCC IN3EX NO INPUT YET
00127A F84B 96 06 A LDA P3DATA INPUT DATA & STROBE
00128A F84D 39 IN3EX RTS

00129

* REQ -- REQUEST TO TALK AND RECONFIGURE WHEN RE-

00130

* QUEST IS GRANTED.

00131

00132

00133A F84E C6 02 A REQ LDAB #$02
00134A F850 D7 07 A STAB P4DATA REQUEST TO BE TALKER
00135A F852 D6 0F A REQ001 LDAB P3CNTL
00136A F854 2A FC F852 BPL REQ001 NO INPUT, NO REQ GRANT YET
00137A F856 D6 07 A LDA P4DATA REQ/GRANT CONTROL WORD
00138A F858 57 ASRB
00139A F859 24 06 F861 BCC GRANTD REQUEST WAS GRANTED
00140A F85B 96 06 A LDA P3DATA INPUT DATA & STROBE TALKER
00141A F85D 8D 0B F86A BSR INBYTE DISPOSE OF INPUT
00142A F85F 20 F1 F852 BRA REQ001 STILL WAITING FOR GRANT
00143A F861 C6 FF A GRANTD LDAB #$FF TO BECOME TALKER
00144A F863 D7 04 A STAB P3DDR PORT 3 OUTPUT
00145A F865 C6 10 A LDAB #$10
00146A F867 D7 0F A STAB P3CNTL STROBE ON WRITE
00147A F869 39 RTS

00148

00149

* INBYTE -- USER WRITTEN SUBROUTINE TO ACCEPT

00150

* BYTE FROM ACCA AND RETURN.

00151

00152 A INBYTE EQU * USER SUPPLIED SUBROUTINE

00153A F86A 39 RTS
00154

TOTAL ERRORS 00000--00000

Figure 8-25. Half Duplex Routines (Continued)
The "talker" MCU can call the routines, OUT3NW, and CHKREQ while the "listener" MCU can call IN3NW or REQ. A successful execution of OUT3NW causes data in the A accumulator to be written to Port 3 and generates an OS3 (Data Ready) strobe. This latches data into the "listener" and sets IS3 FLAG in the Port 3 Control and Status Register.

On the "listener" side of the interface, a successful call to IN3NW reads Port 3 and generates an OS3 (Data Acknowledged) strobe. This sets the IS3 FLAG bit of the "talker." Unlike the routines with the "no wait" feature, when a "listener" calls the REQ subroutine, it returns only upon receipt of a grant to talk. A "listener" posts a request to talk by writing a "1" to its Request/Grant line. While waiting for the grant, this subroutine continues to accept data from Port 3 and provides it to an INBYTE routine which disposes of it.

A grant is recognized by the "listener" upon receipt of a strobe when the Request/Grant line of the "talker" changes to a low level. In response to the grant, the port is reconfigured to outputs, OS3 is configured for a strobe-on-write (OSS = 1) and the "listener" becomes the "talker" (controller).

Subroutine, CHKREQ, can be used to detect the presence of a request from the "listener" and, if present, relinquish control of the interface. This task is performed by first reconfiguring Port 3 as inputs, configuring OS3 for a strobe-on-read (OSS = 0), and setting LATCH ENABLE. Note that by the "listener" making use of the Port 3 input latch, it is possible for the "talker" to reconfigure the port without having to wait for acknowledgement of the last transmitted byte. However, a Data Acknowledge strobe from the "listener" must be received prior to issuing a grant signal.

A strobe is also generated by the "talker" using the CHKREQ subroutine when a "listener" request to talk is granted. This ensures that the new "talker" has an initial "Data Acknowledge" (IS3 FLAG is set) when it is ready to write. The OUT3NW routine will not write to the port until the "listener" has acknowledged receipt of the previous byte. The "listener" provides this signal in the IN3NW routine when reading the byte from the Port 3 Data Register.

Note that the "talker" can enter the CHKREQ subroutine regardless of whether the "listener" has acknowledged receipt of the last byte sent. If the "listener" has issued a request to talk, however, CHKREQ will wait until the last byte is acknowledged before granting the request.

When using these routines, precautions must be taken to ensure that the initial "talker" is out of Reset before the "listener" side of the interface is initialized. This ensures that the initial strobe from the "listener" sets the IS3 FLAG bit of the "talker." The "talker" cannot write data until receipt of this strobe. The maximum data transfer rates using this interface scheme are shown in Figure 8-26 which assumes the hardware configuration shown in Figure 8-23.

8.7.3 4-Bit Full Duplex Interface

If full duplex capability is a requirement, it can be implemented, at the expense of data transfer rate, by configuring some Port 3 lines as outputs and the remainder as inputs. This discussion utilizes four bits as outputs and four bits as inputs but other combinations can be used without modification of the control scheme. One desirable objective in designing the interface is to allow both MPUs to utilize identical controlling software which can be achieved by employing suitable symmetry. The principal advantage in using this interface scheme is avoidance of the software overhead required to
Example for Port 3 Byte Transmit:

```
LOOP1    LDAA DATA,X    READ DATA FROM MEMORY
LOOP2    LDAB P3CNTL    READ STATUS/CONTROL BYTE
        BPL LOOP2     WAIT FOR STROBE
        STAA P3DATA   OUTPUT DATA AND STROBE
        DEX           DECREMENT BYTE COUNTER
        BNE LOOP1     LOOP UNTIL DONE
```

Execution time for $f_0 = 1.0 \text{ MHz}$ ($1 \text{ E-cycle} = 1 \text{ microsecond}$)
Maximum data rate = 52.6 k bytes/sec

Example for Port 3 Listener with No Request to Become a Talker:

```
LOOP    LDAB P3CNTL    FETCH STATUS/CONTROL BYTE
        BPL LOOP      WAIT FOR INPUT DATA
        LDAA P3DATA   READ INPUT DATA AND STROBE
        DEX DATA,X    STORE INPUT
        DEX           DECREMENT BYTE COUNTER
        BNE LOOP      LOOP UNTIL DONE
```

Execution time for $f_0 = 1.0 \text{ MHz}$ ($1 \text{ E-cycle} = 1 \text{ microsecond}$)
Maximum data rate = 52.6 k bytes/sec

Port 3 Input While Requesting to Become a Talker:

```
LOPR    LDAB P3CNTL    READ STATUS/CONTROL BYTE
        BPL LOPR      WAIT FOR INPUT DATA STROBE
        LDAB P4DATA   READ REQ/GRA NT CONTROL
        ASRB          \text{Req is granted}
        BBC GRAN TD   \text{Req is granted}
        LDAA P3DATA   INPUT DATA AND STROBE
        STAA DATA,X   STORE INPUT IN MEMORY
        DEX           DECREMENT BYTE COUNTER
        BNE LO PR     LOOP UNTIL DONE
```

Execution time for $f_0 = 1.0 \text{ MHz}$ ($1 \text{ E-cycle} = 1 \text{ microsecond}$)
Maximum data rate = 37.0 k bytes/sec

Figure 8-26. Half Duplex Data Transfer Rate

reverse the data transfer direction. The OSS bit in the Port 3 Control and Status Register is set for an OS3 strobe-on-write. With OS3 of each MPU connected to IS3 of the other, the IS3 FLAG bit can be interpreted as a “Data Ready” software signal and is cleared by the input routine when it reads the Port 3 Control and Status Register followed by its Data Register.

The remaining necessary ingredients for a two-way interface is a signal to acknowledge receipt of the data. This signal must be generated by the input routine and be resettable by the output routine.

8.7.3.1 FULL DUPLEX WITH INPUT CAPTURE FUNCTION. One method suitable for controlling a 4-bit full duplex interface is to use a data port output line and the edge detector of the input capture function to control the data acknowledgement. From the viewpoint of the sender, writing data to the Port 3 Data Register generates an OS3 strobe to indicate “Data Ready.” The state of the sender ICF (Input Capture Flag) generates an OS3 strobe to indicate “Data Ready.” The state of the sender ICF (Input Capture Flag) indicates whether the last nibble has been received (acknowledged). An interrupt can be generated upon acknowledgement from the receiver by setting the EICI bit in the TCSR.
From the viewpoint of the receiver, the IS3 FLAG bit indicates “Data Ready” when set. Reading the data clears the bit and toggling a data port output line acknowledges receipt of the data. Note that the IS3 IRQ1 ENABLE bit can be set to provide an interrupt as a response to the “Data Ready” signal.

The following discussion considers the implementation of this type of interface and assumes reader familiarity with the Input Capture function of the Programmable Timer and the Timer Control and Status Register. A connection diagram for a full duplex interface, which uses the input capture function, is shown in Figure 8-27. An interrupt driven procedure for this interface is depicted in the flowchart of Figure 8-28. Both MCUs must initialize Port 3 before either can write to the port. The initialization sequence should:

- write $7F to the Port 3 Control and Status Register,
- set EICI in the TCSR, and
- configure the ACK bit as an output using any available data line.

After initialization is complete, interrupts are enabled by clearing the I-bit using the CLI instruction. To initiate output, the second nibble to be transmitted is stored in a buffer specified by the user-supplied subroutine which prepares the next output nibble. The first nibble is then written to the Port 3 Data Register and an interrupt driven dialogue proceeds until the data is exhausted. Note that the data to be transmitted must reside in the four least significant bits of the byte in the buffer.

Figure 8-27. Interface Using Input Capture Function
The output procedure shown in Figure 8-28 is entered in response to an IRQ2/Input Capture interrupt where the ACK signal (Data Acknowledged) from the other MCU produced this interrupt. In order to clear this interrupt, the TCSR and Capture Flag (ICF) could simply be monitored if polling is desired.

After clearing the interrupt, the output procedure reads the nibble to be transmitted from memory and writes it to the Port 3 Data Register. This generates an OS3 (Data Ready) strobe for the other MCU. Before returning from the interrupt, the next nibble to be transmitted is obtained from the output buffer and saved for the next interrupt. If there is no more output data, the Input Capture interrupt can be disabled to avoid an interrupt when the other MCU acknowledges receipt of the last byte.

The input procedure is invoked after an IRQ1 interrupt generated by IS3. The other MCU produces this interrupt by writing to Port 3 which generates an OS3 strobe. If other interrupt sources are also tied to the IRQ1 pin, then a “who-done-it” routine (see Chapter 5) must be employed to determine which device is requesting input service. The IS3 FLAG bit is set by an input strobe and can be used to determine if Port 3 is requesting service.
The input procedure consists of clearing the IS3 interrupt by reading the Port 3 Control and Status Register followed by reading its Data Register. Toggling the ACK line which is connected to P20 then triggers the input capture function of the other MCU. The procedure returns from the interrupt after disposing of the input nibble by calling a user-supplied subroutine.

The IRQI and the Input Capture interrupts have individual prioritized interrupt vectors. When both interrupts are pending, however, the IRQI interrupt is serviced first. It is possible that if an IS3 strobe occurs while processing a previous IS3 interrupt, lower priority interrupts will remain unserviced until all IS3 interrupts are serviced.

8.7.3.2 FULL DUPLEX WITH EXCLUSIVE-OR FUNCTION. If the Input Capture function is not available, a few (3 or 4) unused port lines can be utilized to control a 4-bit full duplex interface. This scheme uses an external exclusive-OR gate to generate a Data Acknowledged (ACK) signal as shown in Figure 8-29.

The ACK signal is generated using P12 and is cleared when the other MCU responds by toggling its P10 output line. Figure 8-29 indicates how P13 can be used to locally mask an ACK interrupt. Although not incorporated into this example, this masking feature can be useful to inhibit interrupts when there is no more output data.

![Figure 8-29. Interface Using Exclusive-OR Function](image-url)
A low-level driver for this interface and its associated flow chart are depicted in Figures 8-30 and 8-31, respectively. To prevent writing to an MCU that has not been initialized, an ACK interrupt occurs as soon as both MCUs are initialized.

Both the IS3 and ACK interrupts share the same IRQ1 interrupt vector. The program shown in Figure 8-30 provides an interrupt handler which calls low level I/O routines IS3NW and OUT3NW. The IN3NW subroutine services the IS3 (Data Ready) interrupt while the OUT3NW subroutine services the IRQ1 (Data Acknowledged) interrupt. If polling is used, these two routines can be called individually. Both return promptly and the state of the C-bit reflects what action was taken.

If the information being transferred is byte-oriented, routines to prepare the input and output nibbles can use the state of control lines P12 and P10, respectively, to control byte packing and unpacking. The maximum data transfer rate using the configuration depicted in Figure 8-29 is shown in Figure 8-32.
PAGE 001 DUALP3 .SA:1 DUALP3 *** FULL DUPLEX PORT 3 INTERFACE USING XOR **

00001                       NAM    DUALP3
00002                       OPT    Z01,LLEN=80
00003                       TTL    *** FULL DUPLEX PORT 3 INTERFACE USING XOR
00004
00005                       *
00006                       *
00007                       *
00008                       *
00009                       * INZ ------ INITIALIZE PORT 3 INTERFACE (NO CALLING
00010                       * ARGUMENTS).
00011                       *
00012                       * IN3NW --- INPUT BYTE FROM PORT 3 IF AVAILABLE. IF
00013                       * AVAILABLE, THE BYTE IS RETURNED IN ACCA
00014                       * AND THE C-BIT IS SET; OTHERWISE THE C-BIT
00015                       * IS CLEAR.
00016                       *
00017                       * OUT3NW -- OUTPUT BYTE FROM ACCA TO PORT 3 IF LAST
00018                       * TRANSMISSION ACKNOWLEDGED AND RETURN WITH
00019                       * C-BIT SET; ELSE, RETURN WITH C-BIT CLEAR.
00020                       *
00021                       * INT ------ ENTRY POINT FOR IRQ1 INTERRUPT VECTOR.
00022                       * ROUTINE SERVICES EITHER RECEIVER OR
00023                       * TRANSMITTER. PRIORITY IS ALTERNATED BE-
00024                       * TWEEN THEM TO INSURE FAIR SERVICE.
00025                       *
00026                       * INT CALLS TWO USER ROUTINES: INNYB AND
00027                       * OUTNYB. INNYB TAKES BYTE FROM ACCA, DIS-
00028                       * POSES OF IT, AND RETURNS. OUTNYB PUTS
00029                       * NEXT BYTE TO BE TRANSMITTED INTO LOCATION
00030                       * "NEXOUT" AND RETURNS. IF THERE IS NO
00031                       * MORE DATA TO BE TRANSMITTED, OUTNYB MUST
00032                       * INHIBIT THE TRANSMITTER INTERRUPT.
00033                       *
00034                       ****************************************************
00035                       *
00036                       * E Q U A T E S
00037                       *
00038                       0000  A P1DDR  EQU  $00  PORT 1 DATA DIRECTION REG
00039                       0002  A P1DATA  EQU  $02  PORT 1 DATA REG
00040                       0004  A P3DDR  EQU  $04  PORT 3 DATA DIRECTION REG
00041                       0006  A P3DATA  EQU  $06  PORT 3 DATA REG
00042                       000F  A P3CNTL  EQU  $0F  PORT 3 CONTROL & STATUS REG
00043
00044                       * L O C A L  V A R I A B L E S
00045
00046A 0080  ORG  $80
00047A 0080  0001  A NEXOUT  RMB  1  TEMP STORAGE FOR OUTPUT DATA
00048A 0081  0001  A WHO1ST  BSZ  1  IF + SERVICE TX 1ST; ELSE RX FIRST
00049
00050                       * I N I T I A L I Z A T I O N
00051
00052A F800  ORG  $F800
00053A F800  C6  0F  A INZ  LDAB  #0F
00054A F802  D7  04  A STAB  P3DDR
00055A F804  C6  7F  A LDAB  #$7F  STROBE ON WRITE, LATCH ENABLED
00056A F806  D7  0F  A STAB  P3CNTL
00057A F808  7F  0002  A CLR  PIDATA
00058A F80B  C6  0D  A LDAB  #$0D

Figure 8-30. Full Duplex Routine Using Exclusive-OR

8-51
Figure 8-30. Full Duplex Routine Using Exclusive-OR (Continued)
Figure 8-31. Flowchart for Interface
Example for Port 3 Nibble Transmit:

```
LOOP1:  LDAA    DATA,X    READ DATA FROM MEMORY
LOOP2:  LDAB    P3DATA   READ CONTROL BYTE
        ASRB
        ADCB  #0
        ASRB
        BCS  LOOP2    WAIT FOR ACKNOWLEDGEMENT
        STAA  P3DATA  OUTPUT DATA AND STROBE
        LDAB  #1
        EORB  P3DATA  UPDATE CONTROL BYTE
        STAB  P3DATA  DECREMENT BYTE COUNTER
        DEX
        BNE  LOOP1    LOOP UNTIL DONE
```

Execution time for $f_0 = 1.0$ MHz (1 E-cycle = 1 microsecond)
Maximum Data rate = 30.3 k nibbles/sec

Example for Port 3 Nibble Receive:

```
LOOP:   LDAB    P3CNTL  READ STATUS/CNTRL REG
        BPL    LOOP     NO INPUT YET
        LDAA  P3DATA   INPUT DATA AND STROBE
        STAA  X
        STAB  #4
        EORB  P1DATA  SEND ACKNOWLEDGE (ACK)
        STAB  P1DATA  DECREMENT BYTE COUNT
        DEX
        BNE  LOOP     LOOP UNTIL DONE
```

Execution time for $f_0 = 1.0$ MHz (1 E-cycle = 1 microsecond)
Maximum data rate = 37.0 k nibbles/sec

**Figure 8-32. Full Duplex Data Transfer Rates (Exclusive-OR)**
APPENDIX A
DEFINITION OF THE EXECUTABLE INSTRUCTIONS

A.1 NOMENCLATURE

The following nomenclature is used in the subsequent definitions.

(a) Operators
   ( ) = contents of register shown inside parentheses
   ← = is transferred to
   ↓ = "is pulled from stack"
   ↑ = "is pushed onto stack"
   • = Boolean AND
   + = Arithmetic addition symbol except where used as inclusive OR symbol in Boolean Formulae
   ◦ = Exclusive OR
   → = Boolean NOT
   * = Multiply
   : = Concatenation
   − = Arithmetic subtraction symbol or Negative symbol (two’s complement)

(b) Registers in the MPU
   ACCA = Accumulator A
   ACCB = Accumulator B
   ACCX = Accumulator ACCA or ACCB
   ACCD = Double Accumulator. Accumulator A concatenated with Accumulator B where A is the most significant byte.
   CCR = Condition code register
   IX = Index register, 16 bits
   IXH = Index register, higher order 8 bits
   IXL = Index register, lower order 8 bits
   PC = Program counter, 16 bits
   PCH = Program counter, higher order (most significant) 8 bits
   PCL = Program counter, lower order (least significant) 8 bits
   SP = Stack pointer
   SPH = Stack pointer high
   SPL = Stack pointer low
(c) Memory and Addressing

\begin{align*}
M &= \text{A memory location (one byte)} \\
M + 1 &= \text{The byte of memory at 0001 plus the address of the memory location indicated by "M".} \\
\text{Rel} &= \text{Relative offset (i.e., the two's complement number stored in the second byte of machine code corresponding to a branch instruction).}
\end{align*}

(d) Bits 0 through 5 of the Condition Code Register

\begin{align*}
C &= \text{Carry — Borrow, bit 0} \\
V &= \text{Two's complement overflow indicator, bit 1} \\
Z &= \text{Zero indicator, bit 2} \\
N &= \text{Negative indicator, bit 3} \\
I &= \text{Interrupt mask, bit 4} \\
H &= \text{Half carry, bit 5}
\end{align*}

(e) Status of Individual Bits BEFORE Execution of an Instruction

\begin{align*}
A_n &= \text{Bit n of ACCA (n = 7, 6, 5, …, 0)} \\
B_n &= \text{Bit n of ACCB (n = 7, 6, 5, …, 0)} \\
D_n &= \text{Bit n of ACCD (n = 15, 14, 13, …, 0)} \\
\text{Where bits 8-15 and 0-7 refer to ACCA and ACCB, respectively.} \\
I_{XHn} &= \text{Bit n of IXH (n = 7, 6, 5, …, 0)} \\
I_{XLn} &= \text{Bit n of IXL (n = 7, 6, 5, …, 0)} \\
M_n &= \text{Bit n of M (n = 7, 6, 5, …, 0)} \\
S_{PHn} &= \text{Bit n of SPH (n = 7, 6, 5, …, 0)} \\
S_{PLn} &= \text{Bit n of SPL (n = 7, 6, 5, …, 0)} \\
X_n &= \text{Bit n of ACCX (n = 7, 6, 5, …, 0)}
\end{align*}

(f) Status of Individual Bits of the RESULT of Execution of an Instruction

(i) For 8-bit Results

\begin{align*}
R_n &= \text{Bit n of the result (n = 7, 6, 5, …, 0)} \\
\text{This applies to instructions which provide a result contained in a single byte of memory or in an 8-bit register.}
\end{align*}

(ii) For 16-bit Results

\begin{align*}
R_{Hn} &= \text{Bit n of the more significant byte of the result (n = 7, 6, 5, …, 0)} \\
R_{Ln} &= \text{Bit n of the less significant byte of the result (n = 7, 6, 5, …, 0)} \\
\text{This applies to instructions which provide a result contained in two consecutive bytes of memory or in a 16-bit register.}
\end{align*}

\begin{align*}
R_n &= \text{Bit n of the result (n = 15, 14, 13, …, 0)}
\end{align*}

A.2 EXECUTABLE INSTRUCTIONS

Detailed definitions of the 83 executable instructions of the source language are provided on the following pages. The format of these instructions is similar to those used with the MC6800. Where new instructions are introduced or where the MC6800 instruction is enhanced (CPX), the mnemonic heading is printed in italics.
Add Accumulator B to Accumulator A

**Operation:** \( \text{ACCA} \leftarrow (\text{ACCA}) + (\text{ACCB}) \)

**Description:** Adds the contents of ACCB to the contents of ACCA and places the result in ACCA.

**Condition Codes:**
- **H:** Set if there was a carry from bit 3; cleared otherwise.
- **I:** Not affected.
- **N:** Set if most significant bit of the result is set; cleared otherwise.
- **Z:** Set if all bits of the result are cleared; cleared otherwise.
- **V:** Set if there was two's complement overflow as a result of the operation; cleared otherwise.
- **C:** Set if there was a carry from the most significant bit of the result; cleared otherwise.

**Boolean Formulae for Condition Codes:**
- \( H = A_3 \cdot B_3 + B_3 \cdot R_3 + R_3 \cdot A_3 \)
- \( N = R_7 \)
- \( Z = R_7 \cdot R_6 \cdot R_5 \cdot R_4 \cdot R_3 \cdot R_2 \cdot R_1 \cdot R_0 \)
- \( V = A_7 \cdot B_7 + \overline{A_7} \cdot B_7 \cdot R_7 \)
- \( C = A_7 \cdot B_7 + B_7 \cdot \overline{R_7} + \overline{R_7} \cdot A_7 \)

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>2</td>
<td>1</td>
<td>Hex 01B, Octal 033, Decimal 027</td>
</tr>
</tbody>
</table>

A-3
**ABX**

Add Accumulator B to Index Register

Operation: \( \text{IX} \leftarrow (\text{IX}) + (\text{ACCB}) \)

Description: Adds the 8 bit unsigned contents of ACCB to the contents of IX taking into account the possible carry out of the low order byte of the Index Register, and places the result in the IX. ACCB is not changed.

Condition Codes: Not affected

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>3</td>
<td>1</td>
<td>3A 072 058</td>
</tr>
</tbody>
</table>
Operation: \[\text{ACCX} \rightarrow (\text{ACCX}) + (M) + (C)\]

Description: Adds the contents of the C bit to the sum of the contents of ACCX and M and places the result in ACCX.

Condition Codes:
- H: Set if there was a carry from bit 3; cleared otherwise.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if there was two’s complement overflow as a result of the operation; cleared otherwise.
- C: Set if there was a carry from the most significant bit of the result; cleared otherwise.

Boolean Formulae for Condition Codes:
- \[H = X_3 \cdot M_3 + M_3 \cdot R_3 + R_3 \cdot X_3\]
- \[N = R_7\]
- \[Z = R_7 \cdot R_6 \cdot R_5 \cdot R_4 \cdot R_3 \cdot R_2 \cdot R_1 \cdot R_0\]
- \[V = X_7 \cdot M_7 + X_7 \cdot M_7 \cdot R_7\]
- \[C = X_7 \cdot M_7 + M_7 \cdot R_7 + R_7 \cdot X_7\]

Addressing Formats: See Table A-1.

Addressing, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A IMM</td>
<td>2</td>
<td>2</td>
<td>89 211</td>
</tr>
<tr>
<td>A DIR</td>
<td>3</td>
<td>2</td>
<td>99 231</td>
</tr>
<tr>
<td>A EXT</td>
<td>4</td>
<td>3</td>
<td>B9 271</td>
</tr>
<tr>
<td>A IND</td>
<td>4</td>
<td>2</td>
<td>A9 251</td>
</tr>
<tr>
<td>B IMM</td>
<td>2</td>
<td>2</td>
<td>C9 311</td>
</tr>
<tr>
<td>B DIR</td>
<td>3</td>
<td>2</td>
<td>D9 331</td>
</tr>
<tr>
<td>B EXT</td>
<td>4</td>
<td>3</td>
<td>F9 371</td>
</tr>
<tr>
<td>B IND</td>
<td>4</td>
<td>2</td>
<td>E9 351</td>
</tr>
</tbody>
</table>

A-5
ADD

Add Without Carry

Operation: ACCX ← (ACCX) + (M)

Description: Adds the contents of M to the contents of ACCX and places the result in ACCX.

Condition Codes:

H: Set if there was a carry from bit 3; cleared otherwise.
I: Not affected.
N: Set if most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
V: Set if two’s complement overflow as a result of the operation; cleared otherwise.
C: Set if there was a carry from the most significant bit of the result; cleared otherwise.

Boolean Formulae for Condition Codes:

H = X3•M3 + M3•R3 + R3•X3
N = R7
Z = R7•R6•R5•R4•R3•R2•R1•R0
V = X7•M7•R7 + X7•M7•R7
C = X7•M7 + M7•R7 + R7•X7

Addressing Formats: See Table A-1.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A IMM</td>
<td>2</td>
<td>2</td>
<td>8B 213 139</td>
</tr>
<tr>
<td>A DIR</td>
<td>3</td>
<td>2</td>
<td>9B 233 155</td>
</tr>
<tr>
<td>A EXT</td>
<td>4</td>
<td>3</td>
<td>BB 273 187</td>
</tr>
<tr>
<td>A IND</td>
<td>4</td>
<td>2</td>
<td>AB 253 171</td>
</tr>
<tr>
<td>B IMM</td>
<td>2</td>
<td>2</td>
<td>CB 313 203</td>
</tr>
<tr>
<td>B DIR</td>
<td>3</td>
<td>2</td>
<td>DB 333 219</td>
</tr>
<tr>
<td>B EXT</td>
<td>4</td>
<td>3</td>
<td>FB 373 251</td>
</tr>
<tr>
<td>B IND</td>
<td>4</td>
<td>2</td>
<td>EB 355 235</td>
</tr>
</tbody>
</table>
**ADD D**

**Add Double Accumulator**

**Operation:**  
ACCD ← (ACCD) + (M: M + 1)

**Description:** Adds the contents of M concatenated with M + 1 to the contents of ACCD and places the results in ACCD.

**Condition Codes:**
- **H:** Not affected.
- **I:** Not affected.
- **N:** Set if most significant bit of result is set; cleared otherwise.
- **Z:** Set if all bits of the result are cleared; cleared otherwise.
- **V:** Set if there was two's complement overflow as a result of the operation; cleared otherwise.
- **C:** Set if there was a carry from the most significant bit of the result; cleared otherwise.

**Boolean Formulae for Condition Codes:**

\[
N = R15 \\
Z = R15 \cdot R14 \cdot R13 \cdot R12 \cdot R11 \cdot R10 \cdot R9 \cdot R8 \cdot R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \\
V = D15 \cdot M15 \cdot R15 + D15 \cdot M15 \cdot R15 \\
C = D15 \cdot M15 + M15 \cdot R15 + R15 \cdot D15
\]

**Addressing Modes, Execution Time and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM</td>
<td>4</td>
<td>3</td>
<td>C3 303 195</td>
</tr>
<tr>
<td>DIR</td>
<td>5</td>
<td>2</td>
<td>D3 323 211</td>
</tr>
<tr>
<td>EXT</td>
<td>6</td>
<td>3</td>
<td>F3 363 243</td>
</tr>
<tr>
<td>IND</td>
<td>6</td>
<td>2</td>
<td>E3 343 227</td>
</tr>
</tbody>
</table>

A-7
**AND Logical AND**

**Operation:** \( \text{ACCX} \leftarrow (\text{ACCX}) \cdot (M) \)

**Description:** Performs logical AND between the contents of ACCX and the contents of M and places the result in ACCX. (Each bit of ACCX after the operation will be the logical AND of the corresponding bits of M and of ACCX before the operation.)

**Condition Codes:**
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Cleared.
- C: Not affected.

**Boolean Formulae for Condition Codes:**
\[
\begin{align*}
N &= R7 \\
Z &= R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \\
V &= 0
\end{align*}
\]

**Addressing Formats:** See Table A-1.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Codes</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A IMM</td>
<td>2</td>
<td>2</td>
<td>84, 204, 132</td>
</tr>
<tr>
<td>A DIR</td>
<td>3</td>
<td>2</td>
<td>94, 224, 148</td>
</tr>
<tr>
<td>A EXT</td>
<td>4</td>
<td>2</td>
<td>B4, 264, 180</td>
</tr>
<tr>
<td>A IND</td>
<td>4</td>
<td>2</td>
<td>A4, 244, 164</td>
</tr>
<tr>
<td>B IMM</td>
<td>2</td>
<td>2</td>
<td>C4, 304, 196</td>
</tr>
<tr>
<td>B DIR</td>
<td>3</td>
<td>2</td>
<td>D4, 324, 212</td>
</tr>
<tr>
<td>B EXT</td>
<td>4</td>
<td>2</td>
<td>F4, 364, 244</td>
</tr>
<tr>
<td>B IND</td>
<td>4</td>
<td>2</td>
<td>E4, 344, 228</td>
</tr>
</tbody>
</table>
ASL

Arithmetic Shift Left

Operation:

```
   C
b7 b6 b5 b4 b3 b2 b1 b0
  0
```

Description: Shifts all bits of the ACCX or M one place to the left. Bit 0 is loaded with a zero. The C bit is loaded from the most significant bit of ACCX or M.

Condition Codes:

- **H**: Not affected.
- **I**: Not affected.
- **N**: Set if most significant bit of the result is set; cleared otherwise.
- **Z**: Set if all bits of the result are cleared; cleared otherwise.
- **V**: Set if, after the completion of the shift operation, (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
- **C**: Set if, before the operation, the most significant bit of the ACCX or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

- **N** = R7
- **Z** = R7&R6&R5&R4&R3&R2&R1&R0
- **V** = N ⊕ C = [N•C] + [N•¬C]
  (the foregoing formula assumes values of N and C after the shift operation)
- **C** = M7

Addressing Formats: See Table A-3.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>48 110 072</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>58 130 088</td>
</tr>
<tr>
<td>EXT</td>
<td>6</td>
<td>3</td>
<td>78 170 120</td>
</tr>
<tr>
<td>IND</td>
<td>6</td>
<td>2</td>
<td>68 150 104</td>
</tr>
</tbody>
</table>

A-9
**ASLD**

Arithmetic Shift Left Double Accumulator

**Operation:**

```
  b7  b6  b5  b4  b3  b2  b1  b0
   ACCA----------------------------------------0
   b7  b6  b5  b4  b3  b2  b1  b0
   ACCB----------------------------------------0
```

**Description:** Shifts all bits of ACCD one place to the left. Bit 0 is loaded with a zero. The C bit is loaded from the most significant bit of ACCD.

**Condition Codes:**

- **H:** Not affected.
- **I:** Not affected.
- **N:** Set if most significant bit of result is set; cleared otherwise.
- **Z:** Set if all bits of the result are cleared; cleared otherwise.
- **V:** Set if, after the completion of the shift operation, (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
- **C:** Set if before the operation the most significant bit of ACCD was set; cleared otherwise.

**Boolean Formulae for Condition Codes:**

- \( N = R_{15} \)
- \( Z = \overline{R_{15}} \cdot \overline{R_{14}} \cdot R_{13} \cdot R_{12} \cdot \overline{R_{11}} \cdot R_{10} \cdot R_9 \cdot R_8 \cdot R_7 \cdot \overline{R_6} \cdot \overline{R_5} \cdot R_4 \cdot \overline{R_3} \cdot R_2 \cdot \overline{R_1} \cdot R_0 \)
- \( V = N \cdot \overline{C} + [N \cdot \overline{C}] \)
- \( C = D_{15} \)

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>3</td>
<td>1</td>
<td>05 005 005</td>
</tr>
</tbody>
</table>

A-10
**Operation:**

Shifts all bits of ACCX or M one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C bit.

**Condition Codes:**
- **H:** Not affected.
- **I:** Not affected.
- **N:** Set if the most significant bit of the result is set; cleared otherwise.
- **Z:** Set if all bits of the result are cleared; cleared otherwise.
- **V:** Set if, after the completion of the shift operation, (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
- **C:** Set if, before the operation, the least significant bit of the ACCX or M was set; cleared otherwise.

**Boolean Formulae for Condition Codes:**
- \( N = R_7 \)
- \( Z = R_7 \cdot R_6 \cdot R_5 \cdot R_4 \cdot R_3 \cdot R_2 \cdot R_1 \cdot R_0 \)
- \( V = N \oplus C = [N \cdot C] + [N \cdot \overline{C}] \)

(the foregoing formula assumes values of N and C after the shift operation)
- \( C = M_0 \)

**Addressing Formats:**
See Table A-3.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>47 107 071</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>57 127 087</td>
</tr>
<tr>
<td>EXT</td>
<td>6</td>
<td>3</td>
<td>77 167 119</td>
</tr>
<tr>
<td>IND</td>
<td>6</td>
<td>2</td>
<td>67 147 103</td>
</tr>
</tbody>
</table>
Description: Tests the state of the C bit in the condition code register and causes a branch if it is clear.

Condition Codes: Not affected.

Addressing Formats: See Table A-8.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>24 Oct. 044</td>
</tr>
</tbody>
</table>
Operation: \( PC \leftarrow (PC) + 0002 + \text{Rel if } (C) = 1 \)

Description: Tests the state of the C bit in the condition code register and causes a branch if it is set.

See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Addressing Formats: See Table A-8.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>25</td>
</tr>
</tbody>
</table>
BEQ

Branch if Equal

Operation: \( PC - (PC) + 0002 + \text{Rel} \) if \((Z) = 1\)

Description: Tests the state of the Z bit in the condition code register and causes a branch if it is set.

See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Addressing Formats: See Table A-8.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>27 047 039</td>
</tr>
</tbody>
</table>
BGE  
Branch if Greater Than or Equal to Zero  

Operation:  
PC ← (PC) + 0002 + Rel if (N) Φ (V) = 0  
i.e., if (ACCX) ≥ (M)  
(Two's complement numbers)

Description:  
Causes a branch if (N is set and V is set) OR (N is clear and V is clear).  
If the BGE instruction is executed immediately after execution of any of the in-  
stuctions CBA, CMP, SBA, or SUB, the branch will occur if and only if the  
two's complement number represented by the minuend (i.e., ACCX) was greater  
than or equal to the two's complement number represented by the subtrahend  
(i.e., M).

See BRA instruction for details of branch.

Condition  
Codes:  
Not affected.

Addressing  
Formats:  
See Table A-8.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>2C 054 044</td>
</tr>
</tbody>
</table>
BGT  

Branch if Greater than Zero

**Operation:** 

\[ PC \rightarrow (PC) + 0002 + \text{Rel} \text{ if } (Z) + [(N) \& (V)] = 0 \]

i.e., if \((\text{ACCX}) > (M)\)

(two's complement numbers)

**Description:**

Causes a branch if \([Z \text{ is clear}] \text{ AND } [(N \text{ is set AND } V \text{ is set}) \text{ OR } (N \text{ is clear AND } V \text{ is clear})].\)

If the BGT instruction is executed immediately after execution of any of the instructions CBA, CMP, SBA, or SUB, the branch will occur if and only if the two's complement number represented by the minuend (i.e., ACCX) was greater than the two's complement number represented by the subtrahend (i.e., M).

See BRA instruction for details of the branch.

**Condition Codes:**

Not affected.

**Addressing Formats:**

See Table A-8.

**Addressing, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>2E 056 046</td>
</tr>
</tbody>
</table>
**BHI**

**Branch if Higher**

**Operation:**

\[ PC \leftarrow (PC) + 0002 + \text{Rel if } (C) + (Z) = 0 \]

i.e., if \((\text{ACCX}) > (M)\)

(\text{unsigned binary numbers})

**Description:**

Causes a branch if \((C\text{ is clear})\) AND \((Z\text{ is clear})\).

If the BHI instruction is executed immediately after execution of any of the instructions CBA, CMP, SBA, or SUB, the branch will occur if and only if the unsigned binary number represented by the minuend (i.e., \text{ACCX}) was greater than the unsigned binary number represented by the subtrahend (i.e., \text{M}).

See BRA instruction for details of the execution of the branch.

**Condition Codes:**

Not affected.

**Addressing Formats:**

See Table A-8.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>Hex. 0x22, Oct. 042, Dec. 034</td>
</tr>
</tbody>
</table>

A-17
**BHS**

**Branch if Higher or Same**

**Operation:** \( PC \leftarrow (PC) + 0002 + \text{Rel if (C) = 0} \)

**Description:** Tests the state of the C bit in the condition code register and causes a branch if it is clear.

See BRA instruction for further details of the execution of the branch.

**Condition Codes:** Not affected.

**Addressing Formats:** See Table A-8.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>24 044 036</td>
</tr>
</tbody>
</table>
Operation: \((\text{ACCX}) \land (\text{M})\)

Description: Performs the logical AND comparison of the contents of ACCX and the contents of M and modifies condition codes accordingly. Neither the contents of ACCX or M operands are affected. (Each bit of the result of the AND would be the logical AND of the corresponding bits of M and ACCX.)

Condition Codes:
- **H**: Not affected.
- **I**: Not affected.
- **N**: Set if the most significant bit of the result of the AND is set; cleared otherwise.
- **Z**: Set if all bits of the result of the AND are cleared; cleared otherwise.
- **V**: Cleared.
- **C**: Not affected.

Boolean Formulae for Condition Codes:

\[
\begin{align*}
N &= R7 \\
Z &= R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \\
V &= 0
\end{align*}
\]

Addressing Formats: See Table A-1.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A IMM</td>
<td>2</td>
<td>2</td>
<td>85, 205, 133</td>
</tr>
<tr>
<td>A DIR</td>
<td>3</td>
<td>2</td>
<td>95, 225, 149</td>
</tr>
<tr>
<td>A EXT</td>
<td>4</td>
<td>2</td>
<td>B5, 265, 181</td>
</tr>
<tr>
<td>A IND</td>
<td>4</td>
<td>2</td>
<td>A5, 245, 165</td>
</tr>
<tr>
<td>B IMM</td>
<td>2</td>
<td>2</td>
<td>C5, 305, 197</td>
</tr>
<tr>
<td>B DIR</td>
<td>3</td>
<td>2</td>
<td>D5, 325, 213</td>
</tr>
<tr>
<td>B EXT</td>
<td>4</td>
<td>2</td>
<td>F5, 365, 245</td>
</tr>
<tr>
<td>B IND</td>
<td>4</td>
<td>2</td>
<td>E5, 345, 229</td>
</tr>
</tbody>
</table>
**BLE**

Branch if Less than or Equal to Zero

**Operation:**

\[ PC \leftarrow (PC) + 0002 + \text{Rel} \text{ if } (Z) + [(N) \oplus (V)] = 1 \]

i.e., if \((\text{ACCX}) \leq (M)\)

(two's complement numbers)

**Description:**

Causes a branch if \([Z \text{ is set}] \text{ OR } [(N \text{ is set AND } V \text{ is clear}) \text{ OR } (N \text{ is clear AND } V \text{ is set})]\).

If the BLE instruction is executed immediately after execution of any of the instructions CBA, CMP, SBA, or SUB, the branch will occur if and only if the two's complement number represented by the minuend (i.e., ACCX) was less than or equal to the two's complement number represented by the subtrahend (i.e., M).

See BRA instruction for details of the branch.

**Condition Codes:**

Not affected.

**Addressing Formats:**

See Table A-8.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>2F 057 047</td>
</tr>
</tbody>
</table>

A-20
BLO
Branch If Lower

Operation: \[ PC - (PC) + 0002 + \text{Rel if } (C) = 1 \]
i.e., if \((ACCX) < (M)\)
(unsigned binary numbers)

Description: Tests the state of the C bit in the condition code register and causes a branch if it is set.
See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Addressing Formats: See Table A-8.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>25 045 037</td>
</tr>
</tbody>
</table>

A-21
**BLS**

**Branch if Lower or Same**

**Operation:**

\[ \text{PC} \leftarrow (\text{PC}) + 0002 + \text{Rel} \text{ if } (C) + (Z) = 1 \]

i.e., if (ACCX) ≤ (M)

(unsigned binary numbers)

**Description:**

Causes a branch if (C is set) OR (Z is set).

If the BLS instruction is executed immediately after execution of any of the instructions CBA, CMP, SBA, or SUB, the branch will occur if and only if the unsigned binary number represented by the minuend (i.e., ACCX) was less than or equal to the unsigned binary number represented by the subtrahend (i.e., M).

See BRA instruction for details of the execution of the branch.

**Condition Codes:**

Not affected.

**Addressing Formats:**

See Table A-8.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>Hex. 0x23, Oct. 043, Dec. 035</td>
</tr>
</tbody>
</table>
BLT  Branch if Less than Zero  BLT

Operation:  \[ PC \leftarrow (PC) + 0002 + \text{Rel} \text{ if } (N) \oplus (V) = 1 \]
\[ \text{i.e., if } (\text{ACCX}) < (M) \]
(two's complement numbers)

Description:  Causes a branch if (N is set AND V is clear) OR (N is clear AND V is set).
If the BLT instruction is executed immediately after execution of any of the instructions CBA, CMP, SBA, or SUB, the branch will occur if and only if the two's complement number represented by the minuend (i.e., ACCX) was less than the two's complement number represented by the subtrahend (i.e., M).
See BRA instruction for details of the branch.

Condition Codes:  Not affected.

Addressing Formats:  See Table A-8.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>2D 055 045</td>
</tr>
</tbody>
</table>
**BMI**

Branch if Minus

**Operation:** \( PC \leftarrow (PC) + 0002 + \text{Rel if (N) = 1} \)

**Description:** Tests the state of the N bit in the condition code register and causes a branch if it is set.

See BRA instruction for details of the execution of the branch.

**Condition Codes:** Not affected.

**Addressing Formats:** See Table A-8.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>Hex. 05 Oct. 2B Dec. 043</td>
</tr>
</tbody>
</table>

A-24
**BNE**  
*Branch if Not Equal*

**Operation:**  
$$PC \leftarrow (PC) + 0002 + \text{Rel} \text{ if } (Z) = 0$$

**Description:**  
Tests the state of the Z bit in the condition code register and causes a branch if it is clear.  
See BRA instruction for details of the execution of the branch.

**Condition Codes:**  
Not affected.

**Addressing Formats:**  
See Table A-8.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>Hex: 26, Oct: 046, Dec: 038</td>
</tr>
</tbody>
</table>
**BPL**

**Branch if Plus**

**Operation:** \( PC \leftarrow (PC) + 0002 + \text{Rel if } (N) = 0 \)

**Description:** Tests the state of the \( N \) bit in the condition code register and causes a branch if it is clear.

See BRA instruction for details of the execution of the branch.

**Condition Codes:** Not affected.

**Addressing Formats:** See Table A-8.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>Hex.: 2A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Oct.: 052</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Dec.: 042</td>
</tr>
</tbody>
</table>
BRA

Branch Always

Operation: \( PC \leftarrow (PC) + 0002 + \text{Rel} \)

Description: Unconditional branch to the address given by the foregoing formula, in which REL is the relative offset stored as a two's complement number in the second byte of machine code corresponding to the branch instruction.

Note: The source program specifies the destination of any branch instruction by its absolute address, either as a numerical value or as a symbol or expression which can be numerically evaluated by the assembler. The assembler obtains the relative address Rel from the absolute address and the current value of the location counter.

Condition Codes: Not affected.

Addressing Formats: See Table A-8.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>20    040 032</td>
</tr>
</tbody>
</table>
**BRN**

**Branch Never**

**Operation:**  
PC ← (PC) + 0002

**Description:**  
Never branches. In effect, this two byte instruction can be considered as a NOP (No operation) requiring three cycles for execution. Its inclusion in the MC6801 instruction set is to provide a complement for the BRA instruction. The instruction is useful during program debug to "negate" the effect of another branch instruction without disturbing its offset byte. Having a complement for BRA is useful in compiler implementations. All MC6801 branch instructions have a complement.

**Condition Codes:**  
Not affected.

**Addressing Formats:**  
See Table A-8.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>21    041     033</td>
</tr>
</tbody>
</table>
BSR  Branch to Subroutine  BSR

**Operation:**

\[ \text{PC} \leftarrow (\text{PC}) + 0002 \]

\[ \text{I}(\text{PCL}) \]

\[ \text{SP} \leftarrow (\text{SP}) - 0001 \]

\[ \text{I}(\text{PCH}) \]

\[ \text{SP} \leftarrow (\text{SP}) - 0001 \]

\[ \text{PC} \leftarrow (\text{PC}) + \text{Rel} \]

**Description:**
The program counter is incremented by 2. The less significant byte of the contents of the program counter is pushed into the stack. The stack pointer is then decremented (by 1). The more significant byte of the contents of the program counter is then pushed onto the stack. The stack pointer is again decremented (by 1). A branch then occurs to the location specified by the branch. See BRA instruction for details of the execution of the branch.

**Condition Codes:**
Not affected.

**Addressing Formats:**
See Table A-8.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>6</td>
<td>2</td>
<td>8D 215 141</td>
</tr>
</tbody>
</table>

**BRANCH TO SUBROUTINE EXAMPLE**

<table>
<thead>
<tr>
<th>Memory Location</th>
<th>Machine Code (Hex)</th>
<th>Label</th>
<th>Assembler Language Operator</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A. Before</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td>$1000</td>
<td>8D</td>
<td>BSR</td>
<td>CHARLI</td>
</tr>
<tr>
<td>$1001</td>
<td></td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td>$EFFF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>B. After</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td>$1052</td>
<td>**</td>
<td>CHARLI</td>
<td>***</td>
</tr>
<tr>
<td>SP</td>
<td>$EFFD</td>
<td></td>
<td></td>
<td>*****</td>
</tr>
<tr>
<td>$EFFE</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$EFFF</td>
<td>02</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A-29
BVC

Branch if Overflow Clear

Operation: \( \text{PC} \leftarrow (\text{PC}) + 0002 + \text{Rel if } (V) = 0 \)

Description: Tests the state of the \( V \) bit in the condition code register and causes a branch if it is clear.
See BRA instruction for details of the execution of the branch.

Condition Codes: Not affected.

Addressing Formats: See Table A-8.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>28                             050               040</td>
</tr>
</tbody>
</table>
**BVS**  
**Branch if Overflow Set**

**Operation:**  
PC ← (PC) + 0002 + Rel if (V) = 1

**Description:**  
Tests the state of the V bit in the condition code register and causes a branch if it is set.  
See BRA instruction for details of the execution of the branch.

**Condition Codes:**  
Not affected.

**Addressing Formats:**  
See Table A-8.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>3</td>
<td>2</td>
<td>29 051 041</td>
</tr>
</tbody>
</table>
Operation: \((ACCA) - (ACCB)\)

Description: Compares the contents of ACCA to the contents of ACCB and sets the condition codes, which may be used for arithmetic and logical conditional branches. Both operands are unaffected.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result of the subtraction is set; cleared otherwise.
- Z: Set if all bits of the result of the subtraction are cleared; cleared otherwise.
- V: Set if the subtraction results in two's complement overflow; cleared otherwise.
- C: Set if the subtraction requires a borrow in the most significant bit of the result; cleared otherwise.

Boolean Formulae for Condition Codes:
\[
N = R7 \\
Z = R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \\
V = A7 \cdot B7 \cdot R7 + A7 \cdot B7 \cdot R7 \\
C = A7 \cdot B7 + B7 \cdot R7 + R7 \cdot A7
\]

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>2</td>
<td>1</td>
<td>11 Oct. 021 Dec. 017</td>
</tr>
</tbody>
</table>

A-32
CLC
Clear Carry

Operation: C bit — 0

Description: Clears the C bit in the condition code register.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Not affected.
- Z: Not affected.
- V: Not affected.
- C: Cleared.

Boolean Formulae for Condition Codes:
C = 0

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>2</td>
<td>1</td>
<td>0C 014 012</td>
</tr>
</tbody>
</table>
CLI

Clear Interrupt Mask

Operation: I bit ← 0

Description: Clears the interrupt mask bit in the condition code register. When the I bit is clear, all interrupts are enabled.

Condition Codes:
- H: Not affected.
- I: Cleared.
- N: Not affected.
- Z: Not affected.
- V: Not affected.
- C: Not affected.

Boolean Formulae for Condition Codes:
I = 0

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>2</td>
<td>1</td>
<td>0E 016 014</td>
</tr>
</tbody>
</table>
CLR

Operation: ACCX ← 00
or: M ← 00

Description: The contents of ACCX or M are replaced with zeros.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Cleared.
- Z: Set.
- V: Cleared.
- C: Cleared.

Boolean Formulae for Condition Codes:
- \( N = 0 \)
- \( Z = 1 \)
- \( V = 0 \)
- \( C = 0 \)

Addressing Formats: See Table A-3.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>4F 117 079</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>5F 137 095</td>
</tr>
<tr>
<td>EXT</td>
<td>6</td>
<td>3</td>
<td>7F 177 127</td>
</tr>
<tr>
<td>IND</td>
<td>6</td>
<td>2</td>
<td>6F 157 111</td>
</tr>
</tbody>
</table>
Operation:  V bit ← 0

Description:  Clears the two's complement overflow bit in the condition code register.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Not affected.
- Z: Not affected.
- V: Cleared.
- C: Not affected.

Boolean Formulae for Condition Codes:

\[ V = 0 \]

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>2</td>
<td>1</td>
<td>0A</td>
</tr>
</tbody>
</table>

A-36
CMP

Operation: \((\text{ACCX}) - (\text{M})\)

Description: Compares the contents of ACCX to the contents of M and determines the condition codes, which may be used subsequently for controlling conditional branching. Both operands are unaffected.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result of the subtraction is set; cleared otherwise.
- Z: Set if all bits of the result of the subtraction are cleared; cleared otherwise.
- V: Set if the subtraction results in two's complement overflow; cleared otherwise.
- C: Set if the absolute value of the contents of memory is larger than the absolute value of the accumulator; cleared otherwise.

Boolean Formulae for Condition Codes:
\[
\begin{align*}
N &= R7 \\
Z &= \overline{R7} \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot \overline{R2} \cdot \overline{R1} \cdot R0 \\
V &= X7 \cdot M7 \cdot R7 + \overline{X7} \cdot M7 \cdot R7 \\
C &= X7 \cdot M7 + M7 \cdot R7 + R7 \cdot X7
\end{align*}
\]

Addressing Formats: See Table A-1.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A IMM</td>
<td>2</td>
<td>2</td>
<td>81 201</td>
</tr>
<tr>
<td>A DIR</td>
<td>3</td>
<td>2</td>
<td>91 221</td>
</tr>
<tr>
<td>A EXT</td>
<td>4</td>
<td>3</td>
<td>B1 261</td>
</tr>
<tr>
<td>A IND</td>
<td>4</td>
<td>2</td>
<td>A1 241</td>
</tr>
<tr>
<td>B IMM</td>
<td>2</td>
<td>2</td>
<td>C1 301</td>
</tr>
<tr>
<td>B DIR</td>
<td>3</td>
<td>2</td>
<td>D1 321</td>
</tr>
<tr>
<td>B EXT</td>
<td>4</td>
<td>3</td>
<td>F1 361</td>
</tr>
<tr>
<td>B IND</td>
<td>4</td>
<td>2</td>
<td>E1 341</td>
</tr>
</tbody>
</table>
Complement

Operation: \( \text{ACCX} \leftarrow \rightarrow (\text{ACCX}) = \text{FF} - (\text{ACCX}) \)

or:
\( \text{M} \leftarrow \rightarrow (\text{M}) = \text{FF} - (\text{M}) \)

Description: Replaces the contents of ACCX or M with its one’s complement. (Each bit of the contents of ACCX or M is replaced with the complement of that bit).

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Cleared.
- C: Set.

Boolean Formulae for Condition Codes:
- \( N = R7 \)
- \( Z = R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \)
- \( V = 0 \)
- \( C = 1 \)

Addressing Formats: See Table A-3.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>43 103 067</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>53 123 083</td>
</tr>
<tr>
<td>EXT</td>
<td>6</td>
<td>3</td>
<td>73 163 115</td>
</tr>
<tr>
<td>IND</td>
<td>6</td>
<td>2</td>
<td>63 143 099</td>
</tr>
</tbody>
</table>
**CPX**

**Operation:** \((IX) - (M:M + 1)\)

**Description:** Compares the contents of the index register with a 16 bit value at the address specified and sets the condition codes accordingly. The compare is accomplished internally by doing a 16 bit subtract of \((M:M + 1)\) from the index register without modifying either the index register or \((M:M + 1)\).

**Condition Codes:**
- **H:** Not affected.
- **I:** Not affected.
- **N:** Set if most significant bit of the result of the subtraction is set; cleared otherwise.
- **Z:** Set if all bits of the internal result are cleared; cleared otherwise.
- **V:** Set if the subtraction results in two's complement overflow; cleared otherwise.
- **C:** Set if the absolute value of the contents of memory is larger than the absolute value of the index register; cleared otherwise.

**Boolean Formulae for Condition Codes:**
\[
N = R15
\]
\[
Z = R15 \cdot R14 \cdot R13 \cdot R12 \cdot R11 \cdot R10 \cdot R9 \cdot R8 \cdot R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0
\]
\[
V = IX15 \cdot M15 \cdot R15 + IX15 \cdot M15 \cdot R15
\]
\[
C = IX15 \cdot M15 + M15 \cdot R15 + R15 \cdot IX15
\]

**Addressing Formats:** See Table A-5.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM</td>
<td>4</td>
<td>3</td>
<td>8C 214 140</td>
</tr>
<tr>
<td>DIR</td>
<td>5</td>
<td>2</td>
<td>9C 234 156</td>
</tr>
<tr>
<td>EXT</td>
<td>6</td>
<td>3</td>
<td>BC 274 188</td>
</tr>
<tr>
<td>IND</td>
<td>6</td>
<td>2</td>
<td>AC 254 172</td>
</tr>
</tbody>
</table>
DAA

Decimal Adjust ACCA

Operation: Adds hexadecimal numbers 00, 06, 60, or 66 to ACCA, and may also set the carry bit, as indicated in the following table:

<table>
<thead>
<tr>
<th>State of C-Bit Before DAA (Column 1)</th>
<th>Upper Half-Byte (Bits 4-7) (Column 2)</th>
<th>Initial Half-Carry H-Bit (Column 3)</th>
<th>Lower to ACCA After By DAA (Bits 0-3) (Column 4)</th>
<th>Number Added After DAA (Column 5)</th>
<th>State of C-Bit DAA (Column 6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0-9</td>
<td>0</td>
<td>0-9</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0-8</td>
<td>0</td>
<td>A-F</td>
<td>06</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0-9</td>
<td>1</td>
<td>0-3</td>
<td>06</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>A-F</td>
<td>0</td>
<td>0-9</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>9-F</td>
<td>0</td>
<td>A-F</td>
<td>66</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>A-F</td>
<td>1</td>
<td>0-3</td>
<td>66</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0-2</td>
<td>0</td>
<td>0-9</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0-2</td>
<td>0</td>
<td>A-F</td>
<td>66</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0-3</td>
<td>1</td>
<td>0-3</td>
<td>66</td>
<td>1</td>
</tr>
</tbody>
</table>

NOTE: Columns (1) through (4) of the above table represent all possible cases which can result from any of the operations ABA, ADD, or ADC, with initial carry either set or clear, applied to two binary-coded-decimal operands. The table shows hexadecimal values.

Description: If the contents of ACCA and the state of the carry-borrow bit C and the half-carry bit H are all the result of applying any of the operations ABA, ADD, or ADC to binary-coded-decimal operands, with or without an initial carry, the DAA operation will function as follows.

Subject to the above condition, the DAA operation will adjust the contents of ACCA and the C bit to represent the correct binary-coded-decimal sum and the correct state of the carry.

Condition Codes: H: Not affected.
I: Not affected.
N: Set if most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
V: Not defined.
C: Set or clear according to the same rule as if the DAA and an immediately preceding ABA, ADD, or ADC were replaced by a hypothetical binary-coded-decimal addition.

Boolean Formulae for Condition Codes:
N = R7
Z = R7•R6•R5•R4•R3•R2•R1•R0
C = See table above.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>2</td>
<td>1</td>
<td>19    031 025</td>
</tr>
</tbody>
</table>
**DEC**

**Decrement**

**Operation:**

\[ \text{ACCX} \rightarrow (\text{ACCX}) - 01 \]

or:

\[ M \rightarrow (M) - 01 \]

**Description:** Subtract one from the contents of ACCX or M.

The N, Z, and V condition codes are set or reset according to the results of the operation.

The C bit is not affected by the operation.

**Condition Codes:**

- **H:** Not affected.
- **I:** Not affected.
- **N:** Set if most significant bit of the result is set; cleared otherwise.
- **Z:** Set if all bits of the result are cleared; cleared otherwise.
- **V:** Set if there was two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow occurs if and only if (ACCX) or (M) was 80 before the operation.
- **C:** Not affected.

**Boolean Formulae for Condition Codes:**

\[ N = R7 \]

\[ Z = R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \]

\[ V = X7 \cdot X6 \cdot X5 \cdot X4 \cdot X3 \cdot X2 \cdot X1 \cdot X0 = R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \]

**Addressing Formats:** See Table A-3.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>4A 112</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>5A 132</td>
</tr>
<tr>
<td>EXT</td>
<td>6</td>
<td>3</td>
<td>7A 172</td>
</tr>
<tr>
<td>IND</td>
<td>6</td>
<td>2</td>
<td>6A 152</td>
</tr>
</tbody>
</table>
**Operation:** \( SP \leftarrow (SP) - 0001 \)

**Description:** Subtract one from the stack pointer.

**Condition Codes:** Not affected.

### Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>3</td>
<td>1</td>
<td>34 Oct. 064 Dec. 052</td>
</tr>
</tbody>
</table>
Operation: \( \text{IX} \rightarrow (\text{IX}) - 001 \)

Description: Subtract one from the index register. Only the Z bit is set or reset according to the result of this operation.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Not affected.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Not affected.
- C: Not affected.

Boolean Formulae for Condition Codes:
\[
Z = (RH7\cdot RH6\cdot RH5\cdot RH4\cdot RH3\cdot RH2\cdot RH1\cdot RH0)\cdot \\
(RL7\cdot RL6\cdot RL5\cdot RL4\cdot RL3\cdot RL2\cdot RL1\cdot RL0)
\]

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>3</td>
<td>1</td>
<td>Hex. 09 Oct. 011 Dec. 009</td>
</tr>
</tbody>
</table>
EOR

Exclusive OR

Operation: \[ ACCX \leftarrow (ACCX) \oplus (M) \]

Description: Perform logical EXCLUSIVE OR between the contents of ACCX and the contents of M, and places the result in ACCX. (Each bit of ACCX after the operation will be the logical EXCLUSIVE OR of the corresponding bit of M and ACCX before the operation.)

Condition Codes:

- **H**: Not affected.
- **I**: Not affected.
- **N**: Set if most significant bit of the result is set; cleared otherwise.
- **Z**: Set if all bits of the result are cleared; cleared otherwise.
- **V**: Cleared.
- **C**: Not affected.

Boolean Formulae for Condition Codes:

\[
N = R7 \\quad \quad Z = R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \quad \quad V = 0
\]

Addressing Formats: See Table A-1.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A IMM</td>
<td>2</td>
<td>2</td>
<td>88 210 136</td>
</tr>
<tr>
<td>A DIR</td>
<td>3</td>
<td>2</td>
<td>98 230 152</td>
</tr>
<tr>
<td>A EXT</td>
<td>4</td>
<td>3</td>
<td>B8 270 184</td>
</tr>
<tr>
<td>A IND</td>
<td>4</td>
<td>2</td>
<td>A8 250 168</td>
</tr>
<tr>
<td>B IMM</td>
<td>2</td>
<td>2</td>
<td>C8 310 200</td>
</tr>
<tr>
<td>B DIR</td>
<td>3</td>
<td>2</td>
<td>D8 330 216</td>
</tr>
<tr>
<td>B EXT</td>
<td>4</td>
<td>3</td>
<td>F8 370 248</td>
</tr>
<tr>
<td>B IND</td>
<td>4</td>
<td>2</td>
<td>E8 350 232</td>
</tr>
</tbody>
</table>
INC Increment

Operation: \( \text{ACCX} \leftarrow (\text{ACCX}) + 01 \)

or:
\( M \leftarrow (M) + 01 \)

Description: Add one to the contents of ACCX or M.
The N, Z, and V condition codes are set or reset according to the results of this operation.
The C bit is not affected by the operation.

Condition Codes:
- \( H \): Not affected.
- \( I \): Not affected.
- \( N \): Set if most significant bit of the result is set; cleared otherwise.
- \( Z \): Set if all bits of the result are cleared; cleared otherwise.
- \( V \): Set if there is a two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow will occur if and only if \((\text{ACCX})\) or \((M)\) was 7F before the operation.
- \( C \): Not affected.

Boolean Formulae for Condition Codes:
\[
N = R7
\]
\[
Z = R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0
\]
\[
V = X7 \cdot X6 \cdot X5 \cdot X4 \cdot X3 \cdot X2 \cdot X1 \cdot X0
\]

Addressing Formats: See Table A-3.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>4C 114 076</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>5C 134 092</td>
</tr>
<tr>
<td>EXT</td>
<td>6</td>
<td>1</td>
<td>7C 174 124</td>
</tr>
<tr>
<td>IND</td>
<td>6</td>
<td>2</td>
<td>6C 154 108</td>
</tr>
</tbody>
</table>
**Operation:**  \[ \text{SP} \rightarrow (\text{SP}) + 0001 \]

**Description:** Add one to the stack pointer.

**Condition Codes:** Not affected.

### Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>3</td>
<td>1</td>
<td>31 061 049</td>
</tr>
</tbody>
</table>
Increment Index Register

Operation: \( IX \leftarrow (IX) + 001 \)

Description: Add one to the index register. Only the Z bit is set or reset according to the result of this operation.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Not affected.
- Z: Set if all 16 bits of the result are cleared; cleared otherwise.
- V: Not affected.
- C: Not affected.

Boolean Formulae for Condition Codes:
\[
Z = (R_{H7} \cdot R_{H6} \cdot R_{H5} \cdot R_{H4} \cdot R_{H3} \cdot R_{H2} \cdot R_{H1} \cdot R_{H0}) \cdot (R_{L7} \cdot R_{L6} \cdot R_{L5} \cdot R_{L4} \cdot R_{L3} \cdot R_{L2} \cdot R_{L1} \cdot R_{L0})
\]

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>3</td>
<td>1</td>
<td>08 010 008</td>
</tr>
</tbody>
</table>
JMP

Operation: PC—effective address

Description: A jump occurs to the instruction stored at the effective address. The effective address is obtained according to the rules for EXTended or INDexed addressing.

Condition Codes: Not affected.

Addressing Formats: See Table A-7.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT</td>
<td>3</td>
<td>3</td>
<td>7E 176 126</td>
</tr>
<tr>
<td>IND</td>
<td>3</td>
<td>2</td>
<td>6E 156 110</td>
</tr>
</tbody>
</table>
Operation:  
PC ← (PC) + 0003 (for EXTended addressing)  
PC ← (PC) + 0002 (for INDexed addressing)  
1 (PCL)  
SP ← (SP) - 0001  
1 (PCH)  
SP ← (SP) - 0001  
PC ← effective address

Description:  
The program counter is incremented by 3 or by 2, depending on the addressing mode, and is then pushed onto the stack, eight bits at a time. The stack pointer points to the next empty location in the stack. A jump occurs to the instruction stored at the numerical address. The effective address is obtained according to the rules for EXTended or INDexed addressing.

Condition Codes:  
Not affected.

Addressing Formats:  
See Table A-7.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT</td>
<td>6</td>
<td>3</td>
<td>BD 275 189</td>
</tr>
<tr>
<td>IND</td>
<td>6</td>
<td>2</td>
<td>AD 255 173</td>
</tr>
<tr>
<td>DIR</td>
<td>5</td>
<td>2</td>
<td>9D 235 157</td>
</tr>
</tbody>
</table>

JUMP TO SUBROUTINE EXAMPLE (EXTENDED MODE)

<table>
<thead>
<tr>
<th>Memory Location</th>
<th>Machine Code (Hex)</th>
<th>Label</th>
<th>Assembler Language Operator</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Before:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td>$0FF</td>
<td>BD</td>
<td>JSR</td>
<td>CHARLI</td>
</tr>
<tr>
<td>$1000</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$1001</td>
<td>77</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td>$EFFF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B. After:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td>$2077</td>
<td>**</td>
<td>CHARLI</td>
<td>***</td>
</tr>
<tr>
<td>S</td>
<td>$EFFF</td>
<td></td>
<td>$EFFF</td>
<td>02</td>
</tr>
<tr>
<td></td>
<td>$EFFFF</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A-49
LDA
Load Accumulator

Operation: ACCX ← (M)

Description: Loads the contents of memory into the accumulator. The condition codes are set according to the data.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Cleared.
- C: Not affected.

Boolean Formulae for Condition Codes:
\[ N = R7 \]
\[ Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot R3 \cdot R2 \cdot R1 \cdot R0 \]
\[ V = 0 \]

Addressing Formats: See Table A-1.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A IMM</td>
<td>2</td>
<td>2</td>
<td>86</td>
</tr>
<tr>
<td>A DIR</td>
<td>3</td>
<td>2</td>
<td>96</td>
</tr>
<tr>
<td>A EXT</td>
<td>4</td>
<td>3</td>
<td>B6</td>
</tr>
<tr>
<td>A IND</td>
<td>4</td>
<td>2</td>
<td>A6</td>
</tr>
<tr>
<td>B IMM</td>
<td>2</td>
<td>2</td>
<td>C6</td>
</tr>
<tr>
<td>B DIR</td>
<td>3</td>
<td>2</td>
<td>D6</td>
</tr>
<tr>
<td>B EXT</td>
<td>4</td>
<td>3</td>
<td>F6</td>
</tr>
<tr>
<td>B IND</td>
<td>4</td>
<td>2</td>
<td>E6</td>
</tr>
</tbody>
</table>
**LDD**

Load Double Accumulator

**Operation:**

ACCD ← (M: M + 1)

**Description:** Loads the contents of memory locations M and M + 1 into the double accumulator D. The condition codes are set according to the data.

**Condition Codes:**

- **H:** Not affected.
- **I:** Not affected.
- **N:** Set if most significant bit of the result is set by the operation; cleared otherwise.
- **Z:** Set if all bits of the result are cleared by the operation; cleared otherwise.
- **V:** Cleared.
- **C:** Not affected.

**Boolean Formulae for Condition Codes:**

\[ N = R_{15} \]
\[ Z = R_{15}R_{14}R_{13}R_{12}R_{11}R_{10}R_{9}R_{8}R_{7}R_{6}R_{5}R_{4}R_{3}R_{2}R_{1}R_{0} \]
\[ V = 0 \]

**Addressing Formats:**

See Table A-5.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM</td>
<td>3</td>
<td>3</td>
<td>CC 314</td>
</tr>
<tr>
<td>DIR</td>
<td>4</td>
<td>2</td>
<td>DC 334</td>
</tr>
<tr>
<td>EXT</td>
<td>5</td>
<td>3</td>
<td>FC 374</td>
</tr>
<tr>
<td>IND</td>
<td>5</td>
<td>2</td>
<td>EC 354</td>
</tr>
</tbody>
</table>
LDS  Load Stack Pointer  LDS

Operation:  SPH — (M)
SPL ← (M + 1)

Description:  Loads the more significant byte of the stack pointer from the byte of memory at the address specified by the program, and loads the less significant byte of the stack pointer from the next byte of memory, at one plus the address specified by the program.

Condition Codes:

H:  Not affected.
I:  Not affected.
N:  Set if the most significant bit of the stack pointer is set by the operation; cleared otherwise.
Z:  Set if all bits of the stack pointer are cleared by the operation; cleared otherwise.
V:  Cleared.
C:  Not affected.

Boolean Formulae for Condition Codes:

N = RH7
Z = (RH7 • RH6 • RH5 • RH4 • RH3 • RH2 • RH1 • RH0) •
    (RL7 • RL6 • RLS • RL4 • RL3 • RL2 • RL1 • RL0)
V = 0

Addressing Formats:  See Table A-5.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM</td>
<td>3</td>
<td>3</td>
<td>8E 216 142</td>
</tr>
<tr>
<td>DIR</td>
<td>4</td>
<td>2</td>
<td>9E 236 158</td>
</tr>
<tr>
<td>EXT</td>
<td>5</td>
<td>3</td>
<td>BE 276 190</td>
</tr>
<tr>
<td>IND</td>
<td>5</td>
<td>2</td>
<td>AE 256 174</td>
</tr>
</tbody>
</table>
**LDX**

**Load Index Register**

**Operation:**
- IXH ← (M)
- IXL ← (M + 1)

**Description:** Loads the more significant byte of the index register from the byte of memory at the address specified by the program, and loads the less significant byte of the index register from the next byte of memory, at one plus the address specified by the program.

**Condition Codes:**
- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the index register is set by the operation; cleared otherwise.
- Z: Set if all bits of the index register are cleared by the operation; cleared otherwise.
- V: Cleared.
- C: Not affected.

**Boolean Formulae for Condition Codes:**

\[
N = RH7 \\
Z = (RH7 \cdot RH6 \cdot RH5 \cdot RH4 \cdot RH3 \cdot RH2 \cdot RH1 \cdot RH0) \cdot (RL7 \cdot RL6 \cdot RL5 \cdot RL4 \cdot RL3 \cdot RL2 \cdot RL1 \cdot RL0) \\
V = 0
\]

**Addressing Formats:** See Table A-5.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM</td>
<td>3</td>
<td>3</td>
<td>CE 316</td>
</tr>
<tr>
<td>DIR</td>
<td>4</td>
<td>2</td>
<td>DE 336</td>
</tr>
<tr>
<td>EXT</td>
<td>5</td>
<td>3</td>
<td>FE 376</td>
</tr>
<tr>
<td>IND</td>
<td>5</td>
<td>2</td>
<td>EE 356</td>
</tr>
</tbody>
</table>
**LSL**

**Logical Shift Left**

**Operation:**

```
       C
 b7 b6 b5 b4 b3 b2 b1 b0 0
```

**Description:** Shifts all bits of the ACCX or M one place to the left. Bit 0 is loaded with a zero. The C bit is loaded from the most significant bit of ACCX or M.

**Condition Codes:**

- **H:** Not affected.
- **I:** Not affected.
- **N:** Set if most significant bit of the result is set; cleared otherwise.
- **Z:** Set if all bits of the result are cleared; cleared otherwise.
- **V:** Set if, after the completion of the shift operation, (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
- **C:** Set if, before the operation, the most significant bit of the ACCX or M was set; cleared otherwise.

**Boolean Formulae for Condition Codes:**

\[
N = R7 \\
Z = R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \\
V = N \oplus C = [N \cdot C] + [N \cdot C]
\]

(the foregoing formula assumes values of N and C after the shift operation)

\[
C = M7
\]

**Addressing Formats:** See Table A-3.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Codes</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>48, 110, 072</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>58, 130, 088</td>
</tr>
<tr>
<td>EXT</td>
<td>6</td>
<td>3</td>
<td>78, 170, 120</td>
</tr>
<tr>
<td>IND</td>
<td>6</td>
<td>2</td>
<td>68, 150, 104</td>
</tr>
</tbody>
</table>
**LSLD**

**Logical Shift Left Double**

**Operation:**

```
                C
                b7  ACCA  b0
                ← →
```

```
                C
                b7  ACCB  b0
                ← →
```

**Description:** Shifts all bits of ACCD one place to the left. Bit 0 is loaded with a zero. The C bit is loaded from the most significant bit of ACCD.

**Condition Codes:**

- **H:** Not affected.
- **I:** Not affected.
- **N:** Set if most significant bit of result is set; cleared otherwise.
- **Z:** Set if all bits of the result are cleared; cleared otherwise.
- **V:** Set if, after the completion of the shift operation, (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise.
- **C:** Set if before the operation the most significant bit of ACCD was set; cleared otherwise.

**Boolean Formulae for Condition Codes:**

\[
N = R_{15} \\
Z = R_{15} \& R_{14} \& R_{13} \& R_{12} \& R_{11} \& R_{10} \& R_{9} \& R_{8} \& R_{7} \& R_{6} \& R_{5} \& R_{4} \& R_{3} \& R_{2} \& R_{1} \& R_{0} \\
V = N \oplus C = [N \& \neg C] + [\neg N \& C] \\
C = D_{15}
\]

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>3</td>
<td>1</td>
<td>05 005 005</td>
</tr>
</tbody>
</table>
LSR

Logical Shift Right

Operation:

\[ \begin{array}{cccc}
0 & & & \rightarrow \\
\text{b7} & \text{b6} & \text{b5} & \text{b4} & \text{b3} & \text{b2} & \text{b1} & \text{b0} & \rightarrow \\
\end{array} \]

Description: Shifts all bits of ACCX or M one place to the right. Bit 7 is loaded with a zero. The C bit is loaded from the least significant bit of ACCX or M.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Cleared.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if, after the completion of the shift operation, (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
- C: Set if, before the operation, the least significant bit of the ACCX or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

\[
\begin{align*}
N &= 0 \\
Z &= \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot R0 \\
V &= N \cdot C + \overline{N} \cdot \overline{C} \\
C &= M0 \\
\end{align*}
\]

Addressing Formats: See Table A-3.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>44 104 068</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>54 124 084</td>
</tr>
<tr>
<td>EXT</td>
<td>6</td>
<td>3</td>
<td>74 164 116</td>
</tr>
<tr>
<td>IND</td>
<td>6</td>
<td>2</td>
<td>64 144 100</td>
</tr>
</tbody>
</table>
LSRD

Logical Shift Right Double Accumulator

Operations:

\[
\begin{array}{cccc}
0 & \rightarrow & b7 & \text{ACCA} \\
& & & b0 \\
& \rightarrow & b7 & \text{ACCB} \\
& & & b0 \\
& \rightarrow & C
\end{array}
\]

Description: Shifts all bits of ACCD one place to the right. Bit 15 (MSB of ACCA) is loaded with zero. The C bit is loaded from the least significant bit of ACCD (LSB of ACCB).

Condition Codes:

- H: No affected.
- I: Not affected.
- N: Cleared.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if, after completion of the shift operation, (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
- C: Set if, before the shift, the least significant bit of ACCD was set; cleared otherwise.

Boolean Formulae for Condition Codes:

\[
\begin{align*}
N &= 0 \\
Z &= R_{15} \cdot R_{14} \cdot R_{13} \cdot R_{12} \cdot R_{11} \cdot R_{10} \cdot R_9 \cdot R_8 \cdot R_7 \cdot R_6 \cdot R_5 \cdot R_4 \cdot R_3 \cdot R_2 \cdot R_1 \cdot R_0 \\
V &= N \oplus C = (N \cdot C) + (\overline{N} \cdot C) \\
C &= D_0
\end{align*}
\]

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>3</td>
<td>1</td>
<td>04 004 004</td>
</tr>
</tbody>
</table>
MUL

Multiply Unsigned

Operation: \( \text{ACCD} \leftarrow \text{ACCA} \times \text{ACCB} \)

Description: Multiplies the 8 bits in Accumulator A by the 8 bits in Accumulator B to obtain a 16 bit unsigned number in the double accumulator, D.

Condition Codes:
- **H**: Not affected.
- **I**: Not affected.
- **N**: Not affected.
- **Z**: Not affected.
- **V**: Not affected.
- **C**: Set if bit 7 of result (ACCB B7) is set; cleared otherwise.

Boolean Formulae for Condition Codes:
\[ C = R7 \]

**NOTE:**
The C-bit can be used to round the 16-bit result to an 8-bit result as shown in the following sequence:

\[
\begin{align*}
\text{MUL} \\
\text{ADCA} \quad \#0
\end{align*}
\]

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>10</td>
<td>1</td>
<td>3D    075  061</td>
</tr>
</tbody>
</table>
Operation: ACCX ← \(-\text{ACCX}) = 00 - \text{(ACCX)}$
or: \(M ← -M = 0000 - M\)

Description: Replaces the contents of ACCX or M with its two’s complement. Note that 80 is left unchanged.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if there is two’s complement overflow as a result of the implied subtraction from zero; this will occur if and only if the contents of ACCX or M is 80.
- C: Set if there is a borrow in the implied subtraction from zero; the C bit will be set in all cases except when the contents of ACCX or M is 00.

Boolean Formulae for Condition Codes:
\[
N = R7 \\
Z = R7\cdot R6\cdot R5\cdot R4\cdot R3\cdot R2\cdot R1\cdot R0 \\
V = R7\cdot R6\cdot R5\cdot R4\cdot R3\cdot R2\cdot R1\cdot R0 \\
C = R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0
\]

Addressing Formats: See Table A-3.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>40    100  064</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>50    120  080</td>
</tr>
<tr>
<td>EXT</td>
<td>6</td>
<td>3</td>
<td>70    160  112</td>
</tr>
<tr>
<td>IND</td>
<td>6</td>
<td>2</td>
<td>60    140  096</td>
</tr>
</tbody>
</table>
**NOP**

No Operation

**Description:** This is a single byte instruction which causes only the program counter to be incremented. No other registers are affected.

**Condition Codes:** Not affected.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>2</td>
<td>1</td>
<td>01 001 001</td>
</tr>
</tbody>
</table>
Operation: \( \text{ACCX} \leftarrow (\text{ACX}) + (\text{M}) \)

Description: Performs logical OR between the contents of ACCX and the contents of M and places the result in ACCX. (Each bit of ACCX after the operation will be the logical OR of the corresponding bits of M and of ACCX before the operation.)

Condition Codes:
- **H**: Not affected.
- **I**: Not affected.
- **N**: Set if most significant bit of the result is set; cleared otherwise.
- **Z**: Set if all bits of the result are cleared; cleared otherwise.
- **V**: Cleared.
- **C**: Not affected.

Boolean Formulae for Condition Codes:
- \( N = R7 \)
- \( Z = R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \)
- \( V = 0 \)

Addressing Formats: See Table A-1.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Bytes of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A IMM</td>
<td>2</td>
<td>2</td>
<td>8A 212 138</td>
</tr>
<tr>
<td>A DIR</td>
<td>3</td>
<td>2</td>
<td>9A 232 154</td>
</tr>
<tr>
<td>A EXT</td>
<td>4</td>
<td>3</td>
<td>BA 272 186</td>
</tr>
<tr>
<td>A IND</td>
<td>4</td>
<td>2</td>
<td>AA 252 170</td>
</tr>
<tr>
<td>B IMM</td>
<td>2</td>
<td>2</td>
<td>CA 312 202</td>
</tr>
<tr>
<td>B DIR</td>
<td>3</td>
<td>2</td>
<td>DA 332 218</td>
</tr>
<tr>
<td>B EXT</td>
<td>4</td>
<td>3</td>
<td>FA 372 250</td>
</tr>
<tr>
<td>B IND</td>
<td>4</td>
<td>2</td>
<td>EA 352 234</td>
</tr>
</tbody>
</table>
**PSH**  
**Push Data Onto Stack**

**Operation:**

```
1(ACCX)
SP ← (SP) – 0001
```

**Description:**
The contents of ACCX is stored in the stack at the address contained in the stack pointer. The stack pointer is then decremented.

**Condition Codes:**
Not affected.

**Addressing Formats:**
See Table A-4.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>1</td>
<td>36 066 054</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>1</td>
<td>37 067 055</td>
</tr>
</tbody>
</table>
PSHX

Operation:  
\( l(IXL), SP \leftarrow (SP) - 0001 \)
\( l(IXH), SP \leftarrow (SP) - 0001 \)

Description: The contents of the index register is pushed onto the stack at the address contained in the stack pointer. The stack pointer is decremented by 2.

Condition Codes: Not affected.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>4</td>
<td>1</td>
<td>3C 074 060</td>
</tr>
</tbody>
</table>
Pull Data from Stack

Operation: SP ← (SP) + 0001
        1ACCX

Description: The stack pointer is incremented. The ACCX is then loaded from the stack, from the address which is contained in the stack pointer.

Condition Codes: Not affected.

Addressing Formats: See Table A-4.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>1</td>
<td>33</td>
</tr>
</tbody>
</table>
**PULX**

Pull X-Register From Stack

**Operation:**

\[
\text{SP} \leftarrow (\text{SP}) + 1; \text{IXH} \\
\text{SP} \leftarrow (\text{SP}) + 1; \text{IXL}
\]

**Description:**

The index register is pulled from the stack beginning at the current address contained in the stack pointer +1. The stack pointer is incremented by 2 in total.

**Condition Codes:**

Not affected.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>5</td>
<td>1</td>
<td>38 070 056</td>
</tr>
</tbody>
</table>
ROL

Rotate Left

Operation:

\[ \begin{array}{c}
\text{C} \\
\text{b7} \\
\text{b0} \\
\text{C}
\end{array} \]

Description: Shifts all bits of ACCX or M one place to the left. Bit 0 is loaded from the C bit. The C bit is loaded from the most significant bit of ACCX or M.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if, after the completion of the operation, (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
- C: Set if, before the operation, the most significant bit of the ACCX or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

\[ \begin{align*}
N &= R7 \\
Z &= \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0} \\
V &= N \cdot C = [N \cdot \overline{C}] + [N \cdot C] \\
C &= M7
\end{align*} \]

Addressing Formats: See Table A-3.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>49</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>59</td>
</tr>
<tr>
<td>EXT</td>
<td>6</td>
<td>3</td>
<td>79</td>
</tr>
<tr>
<td>IND</td>
<td>6</td>
<td>2</td>
<td>69</td>
</tr>
</tbody>
</table>
ROR

Rotate Right

Operation:

C

b7

b0

C

Description: Shifts all bits of ACCX or M one place to the right. Bit 7 is loaded from the C bit. The C bit is loaded from the least significant bit of ACCX or M.

Condition Codes:

H: Not affected.
I: Not affected.
N: Set if most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
V: Set if, after the completion of the operation, (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
C: Set if, before the operation, the least significant bit of the ACCX or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

\[ N = R7 \]

\[ Z = R7 \bar{R}6 \bar{R}5 \bar{R}4 \bar{R}3 \bar{R}2 \bar{R}1 \bar{R}0 \]

\[ V = N \oplus C = [N \cdot \overline{C}] + [\overline{N} \cdot C] \]

(the foregoing formula assumes values of N and C after the rotation)

\[ C = M0 \]

Addressing Formats: See Table A-3.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>46 106 070</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>56 126 086</td>
</tr>
<tr>
<td>EXT</td>
<td>6</td>
<td>3</td>
<td>76 166 118</td>
</tr>
<tr>
<td>IND</td>
<td>6</td>
<td>2</td>
<td>66 146 102</td>
</tr>
</tbody>
</table>
RTI

Return from Interrupt

Operation:

\[
\begin{align*}
SP & \leftarrow (SP) + 0001, 1 \text{CC} \\
SP & \leftarrow (SP) + 0001, 1 \text{ACCB} \\
SP & \leftarrow (SP) + 0001, 1 \text{ACCA} \\
SP & \leftarrow (SP) + 0001, 1 \text{IXH} \\
SP & \leftarrow (SP) + 0001, 1 \text{IXL} \\
SP & \leftarrow (SP) + 0001, 1 \text{PCH} \\
SP & \leftarrow (SP) + 0001, 1 \text{PCL}
\end{align*}
\]

Description: The condition codes, Accumulators B and A, the index register, and the program counter, will be restored to a state pulled from the stack. Note that the interrupt mask bit will be reset if and only if the corresponding bit stored in the stack is zero.

Condition Codes: Restored to the states pulled from the stack.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>10</td>
<td>1</td>
<td>3B 073 059</td>
</tr>
</tbody>
</table>

Return from Interrupt

Example

<table>
<thead>
<tr>
<th>Memory Location</th>
<th>Machine Code (Hex)</th>
<th>Label</th>
<th>Assembler Language Operator</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Before</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td>$D066</td>
<td>3B</td>
<td>RTI</td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td>SEFF8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEFF9</td>
<td>11HINZVC (Binary)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEFFA</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEFFB</td>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEFFC</td>
<td>56</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEFFD</td>
<td>78</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEFFE</td>
<td>55</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEFFF</td>
<td>67</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B. After</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td>$5567</td>
<td>**</td>
<td></td>
<td>***</td>
</tr>
<tr>
<td></td>
<td>SEFF8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEFF9</td>
<td>11HINZVC (Binary)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEFFA</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEFFB</td>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEFFC</td>
<td>56</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEFFD</td>
<td>78</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEFFE</td>
<td>55</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEFFF</td>
<td>67</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CC = HINZVC (Binary)
ACCB = 12 (Hex)
ACCA = 34 (Hex)
IXH = 56 (Hex)
IXL = 78 (Hex)
**RTS**  

**Return from Subroutine**

**Operation:**  
SP ← (SP) + 0001  
PCH  
SP ← (SP) + 0001  
PCL

**Description:**  
The stack pointer is incremented (by 1). The contents of the byte of memory, at the address now contained in the stack pointer, are loaded into the 8 bits of highest significance in the program counter. The stack pointer is again incremented (by 1). The contents of the byte of memory, at the address now contained in the stack pointer, are loaded into the 8 bits of lowest significance in the program counter.

**Condition Codes:**  
Not affected.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>5</td>
<td>1</td>
<td>39 071 057</td>
</tr>
</tbody>
</table>

**Return from Subroutine**

<table>
<thead>
<tr>
<th>Memory Location</th>
<th>Machine Code (Hex)</th>
<th>Label</th>
<th>Assembler Language Operator</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Before</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td>$30A2</td>
<td>39</td>
<td>RTS</td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td>$EFFD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$EFFE</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$EFFF</td>
<td>02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B. After</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td>$1002</td>
<td>**</td>
<td>***</td>
<td>******</td>
</tr>
<tr>
<td></td>
<td>$EFFD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$EFFE</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td>$EFFF</td>
<td>02</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Operation: \( \text{ACCA} \leftarrow (\text{ACCA}) - (\text{ACCB}) \)

Description: Subtracts the contents of ACCB from the contents of ACCA and places the result in ACCA. The contents of ACCB are not affected.

Condition Codes:

- **H**: Not affected.
- **I**: Not affected.
- **N**: Set if most significant bit of the result is set; cleared otherwise.
- **Z**: Set if all bits of the result are cleared; cleared otherwise.
- **V**: Set if there is two's complement overflow as a result of the operation.
- **C**: Carry is set if the absolute value of Accumulator B is larger than the absolute value of Accumulator A; cleared otherwise.

Boolean Formulae for Condition Codes:

\[
\begin{align*}
N &= R7 \\
Z &= \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot R0 \\
V &= A7 \cdot B7 \cdot \overline{R7} + \overline{A7} \cdot B7 \cdot R7 \\
C &= \overline{A7} \cdot B7 + B7 \cdot \overline{R7} + R7 \cdot \overline{A7}
\end{align*}
\]

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>
**Operation:** ACCX ← (ACCX) − (M) − (C)

**Description:** Subtracts the contents of M and the contents of C from the contents of ACCX and places the result in ACCX.

**Condition Codes:**
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if there is two's complement overflow as a result of the operation; cleared otherwise.
- C: Carry is set if the absolute value of the contents of memory plus previous carry is larger than the absolute value of the accumulator; cleared otherwise.

**Boolean Formulae for Condition Codes:**

\[
\begin{align*}
N &= R7 \\
Z &= R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \\
V &= X7 \cdot M7 \cdot R7 + X7 \cdot M7 \cdot R7 \\
C &= X7 \cdot M7 + M7 \cdot R7 + R7 \cdot X7
\end{align*}
\]

**Addressing Formats:** See Table A-1.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Times (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A IMM</td>
<td>2</td>
<td>2</td>
<td>82 202 130</td>
</tr>
<tr>
<td>A DIR</td>
<td>3</td>
<td>2</td>
<td>92 222 146</td>
</tr>
<tr>
<td>A EXT</td>
<td>4</td>
<td>3</td>
<td>B2 262 178</td>
</tr>
<tr>
<td>A IND</td>
<td>4</td>
<td>2</td>
<td>A2 242 162</td>
</tr>
<tr>
<td>B IMM</td>
<td>2</td>
<td>2</td>
<td>C2 302 194</td>
</tr>
<tr>
<td>B DIR</td>
<td>3</td>
<td>2</td>
<td>D2 322 210</td>
</tr>
<tr>
<td>B EXT</td>
<td>4</td>
<td>3</td>
<td>F2 362 242</td>
</tr>
<tr>
<td>B IND</td>
<td>4</td>
<td>2</td>
<td>E2 342 226</td>
</tr>
</tbody>
</table>
Operation: C Bit ← 1

Description: Sets the C bit in the condition code register.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Not affected.
- Z: Not affected.
- V: Not affected.
- C: Set.

Boolean Formulae for Condition Codes:
\[ C = 1 \]

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>2</td>
<td>1</td>
<td>0D    015  013</td>
</tr>
</tbody>
</table>
SEI

Set Interrupt Mask

Operation: \( I \text{ Bit} \leftarrow 1 \)

Description: Sets the interrupt mask bit in the condition code register. When the I bit is set, all maskable interrupts are inhibited and the MPU will recognize only a Non-Maskable Interrupt (NMI) Request.

Condition Codes:

- \( H \): Not affected.
- \( I \): Set.
- \( N \): Not affected.
- \( Z \): Not affected.
- \( V \): Not affected.
- \( C \): Not affected.

Boolean Formulae for Condition Codes:

\[ I = 1 \]

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>2</td>
<td>1</td>
<td>0F Oct. 017 Dec. 015</td>
</tr>
</tbody>
</table>
Set Two's Complement Overflow Bit

Operation: \( V \text{ Bit} - 1 \)

Description: Sets the two's complement overflow bit in the condition code register.

Condition Codes:
- \( H \): Not affected.
- \( I \): Not affected.
- \( N \): Not affected.
- \( Z \): Not affected.
- \( V \): Set.
- \( C \): Not affected.

Boolean Formulae for Condition Codes:
\[ V = 1 \]

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>2</td>
<td>1</td>
<td>0B 013 011</td>
</tr>
</tbody>
</table>

A-74
STA

Store Accumulator

Operation: \( M \leftarrow (\text{ACCX}) \)

Description: Stores the contents of ACCX in memory. The contents of ACCX remains unchanged.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the contents of ACCX is set; cleared otherwise.
- Z: Set if all bits of the contents of ACCX are cleared; cleared otherwise.
- V: Cleared.
- C: Not affected.

Boolean Formulae for Condition Codes:
- \( N = X_7 \)
- \( Z = \overline{X_7} \cdot \overline{X_6} \cdot \overline{X_5} \cdot \overline{X_4} \cdot \overline{X_3} \cdot \overline{X_2} \cdot \overline{X_1} \cdot X_0 \)
- \( V = 0 \)

Addressing Formats: See Table A-2.

Addressing Modes, Execution Times, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A DIR</td>
<td>3</td>
<td>2</td>
<td>97 227 151</td>
</tr>
<tr>
<td>A EXT</td>
<td>4</td>
<td>3</td>
<td>B7 267 183</td>
</tr>
<tr>
<td>A IND</td>
<td>4</td>
<td>2</td>
<td>A7 247 167</td>
</tr>
<tr>
<td>B DIR</td>
<td>3</td>
<td>2</td>
<td>D7 327 215</td>
</tr>
<tr>
<td>B EXT</td>
<td>4</td>
<td>3</td>
<td>F7 367 247</td>
</tr>
<tr>
<td>B IND</td>
<td>4</td>
<td>2</td>
<td>E7 347 231</td>
</tr>
</tbody>
</table>

A-75
Store Double Accumulator

**Operation:** \( M : M + 1 - (ACCD) \)

**Description:** Stores the contents of double Accumulator A:B in memory. The contents of ACCD remain unchanged.

**Condition Codes:**
- **H:** Not affected.
- **I:** Not affected.
- **N:** Set if the most significant bit of the contents of ACCD is set; cleared otherwise.
- **Z:** Set if all bits of the contents of ACCD are cleared; cleared otherwise.
- **V:** Cleared.
- **C:** Not affected.

**Boolean Formulae for Condition Codes:**
\[
N = D_{15} \\
Z = D_{15}D_{14}D_{13}D_{12}D_{11}D_{10}D_9D_8D_7D_6D_5D_4D_3D_2D_1D_0 \\
V = 0
\]

**Addressing Formats:** See Table A-6.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIR</td>
<td>4</td>
<td>2</td>
<td>DD 335 221</td>
</tr>
<tr>
<td>EXT</td>
<td>5</td>
<td>3</td>
<td>FD 375 253</td>
</tr>
<tr>
<td>IND</td>
<td>5</td>
<td>2</td>
<td>ED 355 237</td>
</tr>
</tbody>
</table>
Operation:  \[ M \leftarrow (\text{SPH}) \]
\[ M + 1 \leftarrow (\text{SPL}) \]

Description: Stores the more significant byte of the stack pointer in memory at the address specified by the program, and stores the less significant byte of the stack pointer at the next location in memory, at one plus the address specified by the program.

Condition Codes:

- **H**: Not affected.
- **I**: Not affected.
- **N**: Set if the most significant bit of the stack pointer is set; cleared otherwise.
- **Z**: Set if all bits of the stack pointer are cleared; cleared otherwise.
- **V**: Cleared.
- **C**: Not affected.

Boolean Formulae for Condition Codes:

\[
N = \text{SPH7} \\
Z = (\text{SPH7} \cdot \text{SPH6} \cdot \text{SPH5} \cdot \text{SPH4} \cdot \text{SPH3} \cdot \text{SPH2} \cdot \text{SPH1} \cdot \text{SPH0}) \\
(\text{SPL7} \cdot \text{SPL6} \cdot \text{SPL5} \cdot \text{SPL4} \cdot \text{SPL3} \cdot \text{SPL2} \cdot \text{SPL1} \cdot \text{SPL0}) \\
V = 0
\]

Addressing Formats: See Table A-6.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIR</td>
<td>4</td>
<td>2</td>
<td>9F</td>
</tr>
<tr>
<td>EXT</td>
<td>5</td>
<td>3</td>
<td>BF</td>
</tr>
<tr>
<td>IND</td>
<td>5</td>
<td>2</td>
<td>AF</td>
</tr>
</tbody>
</table>
**STX**  
*Store Index Register*

**Operation:**  
\[ M \leftarrow (IXH) \]  
\[ M + 1 \leftarrow (IXL) \]

**Description:** Stores the more significant byte of the index register in memory at the address specified by the program, and stores the less significant byte of the index register at the next location in memory, at one plus the address specified by the program.

**Condition Codes:**

- **H:** Not affected.
- **I:** Not affected.
- **N:** Set if the most significant bit of the index register is set; cleared otherwise.
- **Z:** Set if all bits of the index register are cleared; cleared otherwise.
- **V:** Cleared.
- **C:** Not affected.

**Boolean Formulae for Condition Codes:**

\[ N = IXH7 \]
\[ Z = (IXH7 \& IXH6 \& IXH5 \& IXH4 \& IXH3 \& IXH2 \& IXH1 \& IXH0) \]
\[ (IXL7 \& IXL6 \& IXL5 \& IXL4 \& IXL3 \& IXL2 \& IXL1 \& IXL0) \]
\[ V = 0 \]

**Addressing Formats:** See Table A-6.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIR</td>
<td>4</td>
<td>2</td>
<td>DF 337</td>
</tr>
<tr>
<td>EXT</td>
<td>5</td>
<td>3</td>
<td>FF 377</td>
</tr>
<tr>
<td>IND</td>
<td>5</td>
<td>2</td>
<td>EF 357</td>
</tr>
</tbody>
</table>

A-78
Operation: \( \text{ACCX} \leftarrow (\text{ACCX}) - (M) \)

Description: Subtracts the contents of M from the contents of ACCX and places the result in ACCX.

Condition Codes:
- **H**: Not affected.
- **I**: Not affected.
- **N**: Set if most significant bit of the result is set; cleared otherwise.
- **Z**: Set if all bits of the result are cleared; cleared otherwise.
- **V**: Set if there is a two's complement overflow as a result of the operation; cleared otherwise.
- **C**: Set if the absolute value of the contents of memory are larger than the absolute value of the accumulator; cleared otherwise.

Boolean Formulae for Condition Codes:
- \( N = R7 \)
- \( Z = \bar{R}7 \cdot R6 \cdot \bar{R}5 \cdot R4 \cdot \bar{R}3 \cdot R2 \cdot \bar{R}1 \cdot \bar{R}0 \)
- \( V = X7 \cdot M7 \cdot \bar{R}7 + \bar{X}7 \cdot M7 \cdot R7 \)
- \( C = \bar{X}7 \cdot M7 + M7 \cdot \bar{R}7 + \bar{R}7 \cdot \bar{X}7 \)

Addressing Formats: See Table A-1.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A IMM</td>
<td>2</td>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>A DIR</td>
<td>3</td>
<td>2</td>
<td>90</td>
</tr>
<tr>
<td>A EXT</td>
<td>4</td>
<td>3</td>
<td>B0</td>
</tr>
<tr>
<td>A IND</td>
<td>4</td>
<td>2</td>
<td>A0</td>
</tr>
<tr>
<td>B IMM</td>
<td>2</td>
<td>2</td>
<td>C0</td>
</tr>
<tr>
<td>B DIR</td>
<td>3</td>
<td>2</td>
<td>D0</td>
</tr>
<tr>
<td>B EXT</td>
<td>4</td>
<td>3</td>
<td>F0</td>
</tr>
<tr>
<td>B IND</td>
<td>4</td>
<td>2</td>
<td>E0</td>
</tr>
</tbody>
</table>
**SUBD**  Subtract Double Accumulator

**Operation:**  \( \text{ACCD} - (\text{ACCD}) - (\text{M:M + 1}) \)

**Description:**  Subtracts the contents of \( \text{M:M + 1} \) from the contents of double Accumulator D and places the result in ACCD.

**Condition Codes:**
- **H:**  Not affected.
- **I:**  Not affected.
- **N:**  Set if most significant bit of the result is set; cleared otherwise.
- **Z:**  Set if all bits of the result are cleared; cleared otherwise.
- **V:**  Set if there is a two's complement overflow as a result of the operation; cleared otherwise.
- **C:**  Set if absolute value of the contents of memory is larger than the absolute value of the accumulator; cleared otherwise.

**Boolean Formulae for Condition Codes:**
- \( N = R_{15} \)
- \( Z = R_{15} \cdot R_{14} \cdot R_{13} \cdot R_{12} \cdot R_{11} \cdot R_{10} \cdot R_{9} \cdot R_{8} \cdot R_{7} \cdot R_{6} \cdot R_{5} \cdot R_{4} \cdot R_{3} \cdot R_{2} \cdot R_{1} \cdot R_{0} \)
- \( V = D_{15} \cdot M_{15} + D_{15} \cdot M_{15} + R_{15} \)
- \( C = D_{15} \cdot M_{15} + M_{15} \cdot R_{15} + R_{15} \cdot D_{15} \)

**Addressing Formats:**  See Table A-5.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM</td>
<td>4</td>
<td>3</td>
<td>83 203 131</td>
</tr>
<tr>
<td>DIR</td>
<td>5</td>
<td>2</td>
<td>93 223 147</td>
</tr>
<tr>
<td>EXT</td>
<td>6</td>
<td>3</td>
<td>B3 263 179</td>
</tr>
<tr>
<td>IND</td>
<td>6</td>
<td>2</td>
<td>A3 243 163</td>
</tr>
</tbody>
</table>
SWI
Software Interrupt

Operation:
\[
\begin{align*}
PC & \leftarrow (PC) + 0001 \\
(PCL), SP & \leftarrow (SP) - 0001 \\
(PCH), SP & \leftarrow (SP) - 0001 \\
(I(XL)), SP & \leftarrow (SP) - 0001 \\
(I(XH)), SP & \leftarrow (SP) - 0001 \\
(ACC(A)), SP & \leftarrow (SP) - 0001 \\
(ACC(B)), SP & \leftarrow (SP) - 0001 \\
(CCR), SP & \leftarrow (SP) - 0001 \\
I & \leftarrow 1 \\
PCH & \leftarrow (n - 0005) \\
PCL & \leftarrow (n - 0004)
\end{align*}
\]

Description:
The program counter is incremented (by 1). The program counter, index register, and Accumulator A and B, are pushed onto the stack. The condition code register is then pushed onto the stack, with condition codes H, I, N, Z, V, C going respectively into bit positions 5 through 0, and the top two bits (in bit positions 7 and 6) are set (to the 1 state). The stack pointer is decremented (by 1) after each byte of data is stored on the stack.
The interrupt mask bit is then set. The program counter is loaded with the vector (address) located at $FFFA:FFFB and instruction execution resumes at this location.

NOTE:
This instruction is not affected by the I bit.

Condition Codes:
\[
\begin{align*}
H & : \text{Not affected.} \\
I & : \text{Set.} \\
N & : \text{Not affected.} \\
Z & : \text{Not affected.} \\
V & : \text{Not affected.} \\
C & : \text{Not affected.}
\end{align*}
\]

Boolean Formulae for Condition Codes:
I = 1

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>12</td>
<td>1</td>
<td>Hex: 3F, Oct: 077, Dec: 063</td>
</tr>
</tbody>
</table>
Software Interrupt
Example:

A. Before:
CC = HINZVC (binary)
ACCB = 12 (Hex)
ACCA = 34 (Hex)

<table>
<thead>
<tr>
<th>Memory Location</th>
<th>Machine Code (Hex)</th>
<th>Label</th>
<th>Assembler Language Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>$5566</td>
<td>3F</td>
<td>SWI</td>
</tr>
<tr>
<td>SP</td>
<td>$EFFF</td>
<td>D0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$FFFA</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$FFFFB</td>
<td>55</td>
<td></td>
</tr>
</tbody>
</table>

B. After:
PC = $D055
SP = $EFF8

<table>
<thead>
<tr>
<th>Memory Location</th>
<th>Machine Code (Hex)</th>
<th>Label</th>
<th>Assembler Language Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$EFF9</td>
<td>11HINZVC (binary)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$EFFA</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$EFFB</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$EFFC</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$EFFD</td>
<td>78</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$EFFE</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$EFFF</td>
<td>67</td>
<td></td>
</tr>
</tbody>
</table>
Transfer from Accumulator A to Accumulator B

Operation: ACCB ← (ACCA)

Description: Moves the contents of ACCA to ACCB. The former contents of ACCB are lost. The contents of ACCA are not affected.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the contents of the accumulator is set; cleared otherwise.
- Z: Set if all bits of the contents of the accumulator are cleared; cleared otherwise.
- V: Cleared.
- C: Not affected.

Boolean Formulae for Condition Codes:

\[
N = R_7
\]
\[
Z = R_7 \cdot \overline{R_6} \cdot \overline{R_5} \cdot \overline{R_4} \cdot \overline{R_3} \cdot \overline{R_2} \cdot \overline{R_1} \cdot \overline{R_0}
\]
\[
V = 0
\]

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>2</td>
<td>1</td>
<td>16 026 022</td>
</tr>
</tbody>
</table>
Operation:  CCR ← (ACCA)

Description: Transfers the contents of bit positions 0 through 5 of Accumulator A to the corresponding bit positions of the condition code register. The contents of Accumulator A remain unchanged.

Condition Codes: Set or reset according to the contents of the respective bits 0 through 5 of Accumulator A.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Byte of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>2</td>
<td>1</td>
<td>06 006 006</td>
</tr>
</tbody>
</table>
Transfer from Accumulator B to Accumulator A

Operation: \( \text{ACCA} \leftarrow (\text{ACCB}) \)

Description: Moves the contents of ACCB to ACCA. The former contents of ACCA are lost. The contents of ACCB are not affected.

Condition Codes:
- \( H \): Not affected.
- \( I \): Not affected.
- \( N \): Set if the most significant bit of the accumulator is set; cleared otherwise.
- \( Z \): Set if all bits of the accumulator are cleared; cleared otherwise.
- \( V \): Cleared.
- \( C \): Not affected.

Boolean Formulae for Condition Codes:
\[
N = R_7 \\
Z = R_7 \cdot R_6 \cdot R_5 \cdot R_4 \cdot R_3 \cdot R_2 \cdot R_1 \cdot R_0 \\
V = 0
\]

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>2</td>
<td>1</td>
<td>17, 027, 023</td>
</tr>
</tbody>
</table>

A-85
Operation: ACCA ← (CCR)

Description: Transfers the contents of the condition code register to corresponding bit positions 0 through 5 of Accumulator A. Bit positions 6 and 7 of Accumulator A are effectively read as 1's. The condition code register remains unchanged.

Condition Codes: Not affected.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>2</td>
<td>1</td>
<td>Hex. 07, Oct. 007, Dec. 007</td>
</tr>
</tbody>
</table>
TST

Operation: (ACCX) - 00
(M) - 00

Description: Set condition codes N and Z according to the contents of ACCX or M.

Condition Codes:

H: Not affected.
I: Not affected.
N: Set if most significant bit of the contents of ACCX or M is set; cleared otherwise.
Z: Set if all bits of the contents of ACCX or M are cleared; cleared otherwise.
V: Cleared.
C: Cleared.

Boolean Formulae for Condition Codes:

\[ N = M_7 \]
\[ Z = \overline{M_7} \cdot M_6 \cdot M_5 \cdot M_4 \cdot M_3 \cdot M_2 \cdot M_1 \cdot M_0 \]
\[ V = 0 \]
\[ C = 0 \]

Addressing Formats: See Table A-3.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>4D 115 077</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>5D 135 093</td>
</tr>
<tr>
<td>EXT</td>
<td>6</td>
<td>3</td>
<td>7D 175 125</td>
</tr>
<tr>
<td>IND</td>
<td>6</td>
<td>2</td>
<td>6D 155 109</td>
</tr>
</tbody>
</table>

A-87
**TSX**

Transfer from Stack Pointer to Index Register

**Operation:** IX ← (SP) + 0001

**Description:** Loads the index register with one plus the contents of the stack pointer. The contents of the stack pointer remain unchanged.

**Condition Codes:** Not affected.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>3</td>
<td>1</td>
<td>30 060 048</td>
</tr>
</tbody>
</table>
**TXS**

*Transfer from Index Register to Stack Pointer*

**Operation:**  $\text{SP} \leftarrow (\text{IX}) - 0001$

**Description:** Loads the stack pointer with the contents of the index register, minus one. The contents of the index register remain unchanged.

**Condition Codes:** Not affected.

**Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):**

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>INH</td>
<td>3</td>
<td>1</td>
<td>35 Oct. 065 Dec. 053</td>
</tr>
</tbody>
</table>
Operation:  
\[
\begin{align*}
&PC \leftarrow (PC) + 0001 \\
&(PCL), SP \leftarrow (SP) - 0001 \\
&(PCH), SP \leftarrow (SP) - 0001 \\
&(IXL), SP \leftarrow (SP) - 0001 \\
&(IXH), SP \leftarrow (SP) - 0001 \\
&(ACCA), SP \leftarrow (SP) - 0001 \\
&(ACCB), SP \leftarrow (SP) - 0001 \\
&(CCR), SP \leftarrow (SP) - 0001 \\
\end{align*}
\]

Description:  
The program counter is incremented (by 1). The program counter, index register, and Accumulators A and B, are pushed onto the stack. The condition code register is then pushed onto the stack, with condition codes H, I, N, Z, V, C going respectively into bit positions 5 through 0, and the top two bits (in bit positions 7 and 6) are set (to the 1 state). The stack pointer is decremented (by 1) after each byte of data is stored in the stack.

The MPU then enters a "Wait State." The MPU leaves the Wait State when it senses a Non-Maskable Interrupt (NMI) or, if the I-bit is clear, any Maskable Interrupt (IRQ1 or IRQ2).

Upon leaving the Wait State, the MPU sets the I bit, fetches the vector (address) corresponding to the interrupt sensed, and instruction execution is resumed at this location.

Condition Codes:  
- H: Not affected.
- I: Not affected until the 10th E-cycle of the interrupt sequence. It is set during this cycle.
- N: Not affected.
- Z: Not affected.
- V: Not affected.
- C: Not affected.

Addressing Modes, Execution Time, and Machine Code (Hexadecimal/Octal/Decimal):

<table>
<thead>
<tr>
<th>Addressing Modes</th>
<th>Execution Time (No. of Cycles)</th>
<th>Number of Bytes of Machine Code</th>
<th>Coding of First (or Only) Byte of Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>3E   076  062</td>
</tr>
</tbody>
</table>
Table A-1. Addressing Formats

<table>
<thead>
<tr>
<th>Addressing Mode of First Operand</th>
<th>Second Operation</th>
<th>Accumulator A</th>
<th>Accumulator B</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMMEDIATE</td>
<td>CCC A #number</td>
<td>CCC B #number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CCC A #symbol</td>
<td>CCC B #symbol</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CCC A #expression</td>
<td>CCC B #expression</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CCC A, #C</td>
<td>CCC B, #C</td>
<td></td>
</tr>
<tr>
<td>DIRECT or EXTENDED</td>
<td>CCC A number</td>
<td>CCC B number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CCC A symbol</td>
<td>CCC B symbol</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CCC A expression</td>
<td>CCC B expression</td>
<td></td>
</tr>
<tr>
<td>INDEXED</td>
<td>CCC A X</td>
<td>CCC B X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CCC Z, X</td>
<td>CCC B, X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CCC A number, X</td>
<td>CCC B number, X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CCC A symbol, X</td>
<td>CCC B symbol, X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CCC A expression, X</td>
<td>CCC B expression, X</td>
<td></td>
</tr>
</tbody>
</table>

Notes: 1. CCC = mnemonic operator of source instruction. 2. "symbol" may be the special symbol ".". 3. "expression" may contain the special symbol ".". 4. Space may be omitted before A or B.

Applicable to the following source instructions:
ADC  ADD  AND  BIT  CMP
EOR  LDA  ORA  SBC  SUB

*Special symbol indicating the location counter or, equivalently, the address of the opcode.

Table A-2. Addressing Formats

<table>
<thead>
<tr>
<th>Addressing Mode of First Operand</th>
<th>Second Operation</th>
<th>Accumulator A</th>
<th>Accumulator B</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIRECT or EXTENDED</td>
<td>STA A number</td>
<td>STA B number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STA A symbol</td>
<td>STA B symbol</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STA A expression</td>
<td>STA B expression</td>
<td></td>
</tr>
<tr>
<td>INDEXED</td>
<td>STA A X</td>
<td>STA B X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STA A, X</td>
<td>STA B, X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STA A number, X</td>
<td>STA B number, X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STA A symbol, X</td>
<td>STA B symbol, X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STA A expression, X</td>
<td>STA B expression, X</td>
<td></td>
</tr>
</tbody>
</table>

Notes: 1. "symbol" may be the special symbol ".". 2. "expression" may contain the special symbol ".". 3. Space may be omitted before A or B.

Applicable to the source instruction:
STA

*Special symbol indicating the location counter or, equivalently, the address of the opcode.
### Table A-3. Addressing Formats

<table>
<thead>
<tr>
<th>Operand or Addressing Mode</th>
<th>Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator A</td>
<td>CCC A</td>
</tr>
<tr>
<td>Accumulator B</td>
<td>CCC B</td>
</tr>
<tr>
<td>EXTended</td>
<td>CCC number</td>
</tr>
<tr>
<td></td>
<td>CCC symbol</td>
</tr>
<tr>
<td></td>
<td>CCC expression</td>
</tr>
<tr>
<td>INDExed</td>
<td>CCC X</td>
</tr>
<tr>
<td></td>
<td>CCC,X</td>
</tr>
<tr>
<td></td>
<td>CCC number,X</td>
</tr>
<tr>
<td></td>
<td>CCC symbol,X</td>
</tr>
<tr>
<td></td>
<td>CCC expression,X</td>
</tr>
</tbody>
</table>

**Notes:**
1. CCC = mnemonic operator of source instruction.
2. "expression" may contain the special symbol "*".
3. Space may be omitted before A or B.

Applicable to the following source instructions:

- ASL
- ASR
- CLR
- COM
- DEC
- INC
- LSL
- LSR
- NEG
- ROL
- ROR
- TST

### Table A-4. Addressing Formats

<table>
<thead>
<tr>
<th>Operand</th>
<th>Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator A</td>
<td>CCC A</td>
</tr>
<tr>
<td>Accumulator B</td>
<td>CCC B</td>
</tr>
</tbody>
</table>

**Notes:**
1. CCC = mnemonic operator of source instruction.
2. Space may be omitted before A or B.

Applicable to the following source instructions:

- PSH
- PUL

### Table A-5. Addressing Formats

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMMEDIATE</td>
<td>CCC #number</td>
</tr>
<tr>
<td></td>
<td>CCC #symbol</td>
</tr>
<tr>
<td></td>
<td>CCC #expression</td>
</tr>
<tr>
<td></td>
<td>CCC #C</td>
</tr>
<tr>
<td>DIRECT or EXTENDED</td>
<td>CCC number</td>
</tr>
<tr>
<td></td>
<td>CCC symbol</td>
</tr>
<tr>
<td></td>
<td>CCC expression</td>
</tr>
<tr>
<td>INDIRECT</td>
<td>CCC X</td>
</tr>
<tr>
<td></td>
<td>CCC.X</td>
</tr>
<tr>
<td></td>
<td>CCC number,X</td>
</tr>
<tr>
<td></td>
<td>CCC symbol,X</td>
</tr>
<tr>
<td></td>
<td>CCC expression,X</td>
</tr>
</tbody>
</table>

**Notes:**
1. CCC = mnemonic operator of source instruction.
2. "symbol" may be the special symbol "*".
3. "expression" may contain the special symbol "*".

Applicable to the following source instructions:

- ADDD
- CPX
- LDD
- LDS
- LDX
- SUBD

*Special symbol indicating the location counter or, equivalently, the address of the opcode.*
### Table A-6. Addressing Formats

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIRect or EXTended</td>
<td>CCC number</td>
</tr>
<tr>
<td></td>
<td>CCC symbol</td>
</tr>
<tr>
<td></td>
<td>CCC expression</td>
</tr>
<tr>
<td>INDEXed</td>
<td>CCC X</td>
</tr>
<tr>
<td></td>
<td>CCC ,X</td>
</tr>
<tr>
<td></td>
<td>CCC number,X</td>
</tr>
<tr>
<td></td>
<td>CCC symbol,X</td>
</tr>
<tr>
<td></td>
<td>CCC expression,X</td>
</tr>
</tbody>
</table>

Notes: 1. CCC = mnemonic operator of source instruction.
2. "expression" may contain the special symbol ".".

Applicable to the following source instructions:

JSR  STD  STS  STX

### Table A-7. Addressing Formats

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTended</td>
<td>CCC number</td>
</tr>
<tr>
<td></td>
<td>CCC symbol</td>
</tr>
<tr>
<td></td>
<td>CCC expression</td>
</tr>
<tr>
<td>INDEXed</td>
<td>CCC X</td>
</tr>
<tr>
<td></td>
<td>CCC ,X</td>
</tr>
<tr>
<td></td>
<td>CCC number,X</td>
</tr>
<tr>
<td></td>
<td>CCC symbol,X</td>
</tr>
<tr>
<td></td>
<td>CCC expression,X</td>
</tr>
</tbody>
</table>

Notes: 1. CCC = mnemonic operator of source instruction.
2. "symbol" may be the special symbol ".".
3. "expression" may contain the special symbol ".".

Applicable to the following source instructions:

JMP

*Special symbol indicating the location counter or, equivalently, the address of the opcode.

### Table A-8. Addressing Formats

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELative</td>
<td>CCC number</td>
</tr>
<tr>
<td></td>
<td>CCC symbol</td>
</tr>
<tr>
<td></td>
<td>CCC expression</td>
</tr>
</tbody>
</table>

Notes: 1. CCC = mnemonic operator of source instruction.
2. "symbol" may be the special symbol ".".
3. "expression" may contain the special symbol ".".

Applicable to the following source instructions:

BCC  BCS  BEQ  BGE  BGT  BHI  BHS  BLE  BLO  BLS  
BLT  BMI  BNE  BPL  BRA  BRN  BSR  BVC  BVS

*Special symbol indicating the location counter or, equivalently, the address of the opcode.
APPENDIX B
MC6801 OPERATION CODE MAP

<table>
<thead>
<tr>
<th>Op Code</th>
<th>ACC A</th>
<th>ACC B</th>
<th>IND</th>
<th>Ext</th>
<th>ACCA or SP</th>
<th>ACCB or X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hi</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
</tr>
<tr>
<td>Lo</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>0000</td>
<td>SBA</td>
<td>BRA</td>
<td>TSX</td>
<td>NEG</td>
<td>SUB</td>
</tr>
<tr>
<td>0001</td>
<td>NOP</td>
<td>CBA</td>
<td>BRN</td>
<td>INS</td>
<td></td>
<td>CMP</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>BHI</td>
<td>PULA</td>
<td></td>
<td>SBC</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>BLS</td>
<td>PULB</td>
<td>COM</td>
<td>SUBD</td>
<td>ADDD</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>LSRD</td>
<td>BCC</td>
<td>DES</td>
<td>LSR</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>ASLD</td>
<td>BCS</td>
<td>TXS</td>
<td></td>
<td>BIT</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>TAP</td>
<td>TAB</td>
<td>BNE</td>
<td>PSH A</td>
<td>ROR</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>TPA</td>
<td>TBA</td>
<td>BEQ</td>
<td>PSHB</td>
<td>ASR</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>INX</td>
<td>BVC</td>
<td>PULX</td>
<td>ASL</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>DEX</td>
<td>DAA</td>
<td>BVS</td>
<td>RTS</td>
<td>ROL</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
<td>CLV</td>
<td>BPL</td>
<td>ABX</td>
<td>DEC</td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
<td>SEV</td>
<td>ABA</td>
<td>BMI</td>
<td>RTI</td>
<td>ADD</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
<td>CLC</td>
<td>BGE</td>
<td>PSHX</td>
<td>INC</td>
<td>CPX</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
<td>SEC</td>
<td>BLT</td>
<td>MUL</td>
<td>TST</td>
<td>BSR</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
<td>CLI</td>
<td>BGT</td>
<td>WAi</td>
<td>JMP</td>
<td>LDS</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
<td>SEI</td>
<td>BLE</td>
<td>SWI</td>
<td>CLR</td>
<td>STS</td>
</tr>
</tbody>
</table>

Note: Certain opcodes provide duplicate instructions. These include: $05, ASLD and LSLD; $24, BCC and BHS; $25, BCS and BLO; and $48, $58, $68, and $78, ASL and LSL.
# APPENDIX C
## ASCII CONVERSION TABLE

<table>
<thead>
<tr>
<th>Bits 4 through 6</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 NUL DLE SP 0 @ P ' p</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 SOH DC1 ! 1 A Q a q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 STX DC2 &quot; 2 B R b r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 ETX DC3 # 3 C S c s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 EOT DC4 $ 4 D T d t</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 ENQ NAK % 5 E U e u</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 ACK SYN &amp; 6 F V f v</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 BEL ETB ' 7 G W g w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bits 0 through 3</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 BS CAN ( 8 H X h x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 HT EM ) 9 I Y i y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A LF SUB : 7 J Z j z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B VT ESC + 6 K [ k ]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C FF FS , &lt; L \ / l</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D CR GS - = M ] m ]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E SO RS . &gt; N \ n ~</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F SI US / ? O - o DEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### APPENDIX D

#### SELECTED POWERS OF 2 AND 16

<table>
<thead>
<tr>
<th>Powers of 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^n$</td>
<td>$n$</td>
</tr>
<tr>
<td>256</td>
<td>8</td>
</tr>
<tr>
<td>512</td>
<td>9</td>
</tr>
<tr>
<td>1 024</td>
<td>10</td>
</tr>
<tr>
<td>2 048</td>
<td>11</td>
</tr>
<tr>
<td>4 096</td>
<td>12</td>
</tr>
<tr>
<td>8 192</td>
<td>13</td>
</tr>
<tr>
<td>16 384</td>
<td>14</td>
</tr>
<tr>
<td>32 768</td>
<td>15</td>
</tr>
<tr>
<td>65 536</td>
<td>16</td>
</tr>
<tr>
<td>131 072</td>
<td>17</td>
</tr>
<tr>
<td>262 144</td>
<td>18</td>
</tr>
<tr>
<td>524 288</td>
<td>19</td>
</tr>
<tr>
<td>1 048 576</td>
<td>20</td>
</tr>
<tr>
<td>2 097 152</td>
<td>21</td>
</tr>
<tr>
<td>4 194 304</td>
<td>22</td>
</tr>
<tr>
<td>8 388 608</td>
<td>23</td>
</tr>
<tr>
<td>16 777 216</td>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Powers of 16</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$16^n$</td>
<td>$n$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>256</td>
<td>2</td>
</tr>
<tr>
<td>4 096</td>
<td>3</td>
</tr>
<tr>
<td>65 536</td>
<td>4</td>
</tr>
<tr>
<td>1 048 576</td>
<td>5</td>
</tr>
<tr>
<td>16 777 216</td>
<td>6</td>
</tr>
<tr>
<td>268 435 466</td>
<td>7</td>
</tr>
<tr>
<td>4 294 967 296</td>
<td>8</td>
</tr>
<tr>
<td>68 719 476 736</td>
<td>9</td>
</tr>
<tr>
<td>1 099 511 627 776</td>
<td>10</td>
</tr>
<tr>
<td>17 592 186 044 416</td>
<td>11</td>
</tr>
<tr>
<td>281 474 976 710 666</td>
<td>12</td>
</tr>
<tr>
<td>4 503 599 627 370 496</td>
<td>13</td>
</tr>
<tr>
<td>72 057 594 037 927 936</td>
<td>14</td>
</tr>
<tr>
<td>1 152 921 504 606 846 976</td>
<td>15</td>
</tr>
</tbody>
</table>
APPENDIX E
THE MC68701 MICROCOMPUTER UNIT

E.1 INTRODUCTION

The MC68701 Microcomputer Unit (MCU) is a monolithic computer that is nearly identical to the MC6801 MCU. The primary difference is that the read-only memory (ROM) in the MC68701 contains an ultraviolet (UV) Erasable Programmable Read-Only Memory (EPROM) instead of the masked ROM available in the MC6801.

The ability to program the EPROM allows the MC68701 to be used in a variety of applications. The MC68701 is particularly effective in:

- applications which are low in volume and do not warrant mask programming, and
- prototype equipment in which it is desired to debug the resident firmware before committing it to an MC6801 masked ROM.

Note that in the former case, compatibility with the MC6801 is inherently achieved. Should the product volume increase to economical levels, a mask programmed MC6801 could be substituted for the MC68701 with minimal changes to the system.

The MC68701 includes a clock oscillator, microprocessor unit (MPU), 2048 bytes of EPROM, 128 bytes of RAM, serial communications interface (SCI), programmable timer, and input/output pins. The resources, except for the EPROM, are identical to those of the MC6801. Because the MC68701 is so similar to the MC6801, this appendix will focus only on the differences between the two parts.

E.2 DIFFERENCES BETWEEN MC6801 AND MC68701 MCUs

The MC68701 contains 2048 bytes of Erasable Programmable Read-Only Memory (EPROM) which replaces the 2048 bytes of Read-Only Memory (ROM) contained in the MC6801. The differences between the two parts involve MC68701 features which support EPROM programming. The essential differences are: (1) the functional and electrical characteristics of the RESET pin, (2) the Mode 0 memory map, (3) mask options involving the ROM, and (4) the RAM Control Register ($14). Signal timing and other detailed information are presented in the MC68701 Data Sheet. Symbolic values used in this discussion — such as $V_{PP}$ and $t_{PP}$ — are defined quantitatively in the MC68701 Data Sheet.

E.2.1 MC68701 RESET/VPP Pin

The RESET/VPP pin for the MC68701 performs three functions: (1) it resets the microcomputer when the pin voltage falls below $V_{IL}$, (2) it is used as a control signal to capture the operating mode of the MCU, and (3) it provides an input for an EPROM programming voltage ($V_{PP}$) at a maximum current of $I_{PP}$.
It is possible that an external Reset circuit for the MC6801 may not function with the MC68701 due to the input current requirements. An external circuit designed for the MC68701, however, can be designed to be compatible with the MC6801. For low volume applications, one should consider implementing the reset circuit for the MC68701 and thus achieving dual compatibility.

Several Reset circuits can be designed for the MC68701 depending upon the objectives of the designer. For example, a circuit could be designed for (a) both EPROM programming and normal operation, (b) normal operation only, or (c) EPROM programming only. Three circuits designed for (a) through (c) are shown in Figures E-1 through E-3, respectively. Each of the circuits has its advantages and disadvantages.

A general purpose Reset circuit for the MC68701 is shown in Figure E-1. The circuit provides the capability of switch selecting the operating mode and enabling or inhibiting programming power. Therefore, this circuit can be used to both program the EPROM and execute instructions from it depending upon the position of S1. In designing the RESET circuit, the reader should note that the current specifications (Iin) for the MC68701 and MC6801 are significantly different with respect to each other and with respect to whether or not the MC68701 is programming the EPROM.

In Figure E-1, compatibility between both MCUs with respect to Iin is achieved by using a value of VCC for V. A voltage of VIH must be achieved at the RESET/Vpp input at operating current, Iin. Assuming a voltage drop of 0.7 volt across the diode, D2, the voltage drop across the pullup resistor, R2, must equal (V - VIH - 0.7) volts. With a normal load current of Iin, the value (ohms) of the resistance, R2, can be calculated from

\[ R2 = \frac{V - VIH - 0.7}{Iin} \]

The principal advantage of this circuit is that it requires only a single power supply (VCC) except when programming the EPROM. Its principal disadvantage is the high power consumption in both the programming and normal modes due to amount of the current which must be sunk by the RESET circuit.

When programming the EPROM is not a consideration, the circuit of Figure E-2 can be used. It uses a single power supply and does not require the switch and diode of Figure E-1. However, it also results in high power consumption due to the amount of the current which must be sunk by the RESET circuit. Note this is the same RESET circuit recommended for the MC6801.

The advantages of the circuit in Figure E-3 are that it consumes less power and supports both normal operation and EPROM programming. However, its disadvantage is that Vpp is also required in normal operation. When compared with the circuit of Figure E-1, one finds that the lower power consumption is obtained by using a higher pullup voltage (Vpp instead of V) in series with a higher resistance for R2. A clamping diode keeps the level at the RESET value in the “Normal” switch position and maintains compatibility with the MC6801 RESET leakage current. The value of the resistor, R2, is obtained by using Iin (Vin High) for the RESET input and 2 milliamperes for the clamping diode. This results in a (Vpp - VIH) voltage drop across the resistor with a current of (Iin + 0.002) amperes or

\[ R2 = \frac{Vpp - VIH}{Iin + 0.002} \text{ ohms} \]
Notes:
1. Mode 0 as shown.
2. $R_1 = 10k$ ohms (typical).
3. The RESET time constant is equal to $RC$ where $R$ is the equivalent parallel resistance of $R_2$ and the number of resistors ($R_1$) placed in the circuit by closed mode control switches.
4. $D_2 = 1N914, 1N4001$ (typical).
5. If $V = V_{CC}$, then $R_2 = (V_{CC} - V_{IH} - 0.7)/I_{IN}$ ohms to meet $V_{IH}$ for the RESET/Vpp pin. Setting $V$ equal to $V_{CC}$ is also compatible with the MC6801. The RESET time constant in this case is approximately $R_2 \times C$.
6. Switch $S_1$ allows selection of normal (RESET) or programming (Vpp) as the input to the RESET/Vpp pin. During switching, the input level is held at a value determined by a diode ($D_2$), resistor ($R_2$) and input voltage ($V$).
7. While $S_1$ is in the "Program" position, RESET should not be asserted.
8. See Data Sheet for typical diode for $D_1$.

Figure E-1. General Purpose MC68701 RESET Circuit
Notes:
1. Mode 1 as shown.
2. $R_1 = 10k$ ohms (typical).
3. The RESET time constant is equal to $RC$ where $R$ is the equivalent parallel resistance of $R_2$ and the number of resistors ($R_1$) placed in the circuit by closed mode control switches.
4. See Data Sheet for typical diode for $D$.
5. $R_2 = \frac{V/I - V_{IH}}{I_{IP}}$ ohms to meet $V_{IH}$ for the $RESET/V_{PP}$ pin. This is also compatible with the MC6801. The RESET time constant in this case is approximately $R_2 \times C$.

Figure E-2. MC68701 RESET Circuit for Normal Operation
Notes:
1. Mode 0 as shown.
2. R1 = 10k ohms (typical).
3. The RESET time constant is equal to RC where R is the equivalent parallel resistance of R2 and the number of resistors (R1) placed in the circuit by closed mode control switches.
4. D2 = 1N914, 1N4001 (typical).
5. R2 = V/I = (Vpp - VH)/Vpp + 0.002 ohms.
6. Switch S1 allows selection of normal (RESET) or programming (Vpp) as the input to the RESET/Vpp pin. During switching, the input level is held at a value determined by R2 and Vpp.
7. See Data Sheet for typical diode for D1.
8. The diode, D2, clamps the maximum RESET input voltage to (Vpp - VCC - 0.7) volts for compatibility with the MC6801.

Figure E-3. MC68701 RESET Circuit for EPROM Programming
E.2.2 MC68701 Mode 0 Memory Map

In Mode 0, the interrupt vector area is changed from $FFFF through $FFFF (in the MC6801) to $BFFF through $BFFF in the MC68701. Note that this is a static address assignment that does not depend upon timing with respect to the \text{RESET} signal as does the MC6801 in Mode 0. A Mode 0 memory map for the MC68701 is shown in Figure E-4. Table E-1 lists the MC68701 interrupt vector locations in Mode 0.

<table>
<thead>
<tr>
<th>Priority</th>
<th>Location</th>
<th>Interrupt Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest</td>
<td>$BFFE:BFFF</td>
<td>RESET</td>
</tr>
<tr>
<td></td>
<td>$BFFC:BFFD</td>
<td>Non-Maskable Interrupt</td>
</tr>
<tr>
<td></td>
<td>$BFFA:BFFB</td>
<td>Software Interrupt (SWI)</td>
</tr>
<tr>
<td></td>
<td>$BFF8:BFF9</td>
<td>IRQ1/Input Strobe 3</td>
</tr>
<tr>
<td></td>
<td>$BFF6:BFF7</td>
<td>IRQ2/Timer Input Capture</td>
</tr>
<tr>
<td></td>
<td>$BFF4:BFF5</td>
<td>IRQ2/Timer Output Compare</td>
</tr>
<tr>
<td></td>
<td>$BFF2:BFF3</td>
<td>IRQ2/Timer Counter Overflow</td>
</tr>
<tr>
<td>Lowest</td>
<td>$BFF0:BFF1</td>
<td>IRQ2/SCI Interrupt</td>
</tr>
</tbody>
</table>

E.2.3 MC6801 Mask Options

When specifying the ROM mask for the MC67801, a mask option may be selected which “relocates” the ROM to one of the following addresses: $C800, $D800, or $E800. Useful memory maps which result from this mask option include Modes 1R and 6R. Initial versions of the MC68701, however, do not support either of these modes. Therefore, the current Data Sheet should be referenced to determine availability.

\textbf{NOTE}

If attempting to emulate the MC6801 1R and 6R operating modes with an MC68701, a current MC68701 Data Sheet should be referenced to determine if these modes are supported.

E.2.4 MC68701 RAM/EPROM Control Register ($14)

The RAM/EPROM Control Register provides a function similar to the RAM Control Register in the MC6801. The register contains four bits: STBY PWR, RAME, PPC, and PLC. The STBY PWR and RAME bits are described in the discussion for the RAM Control Register (Section 3.1.5.1.3). In the MC68701, two additional control bits (PPC and PLC) are included in the RAM/EPROM Control Register to facilitate programming the EPROM. The PLC and PPC bits are readable in all modes but can be changed only in Mode 0. The PLC bit can be written without restriction in Mode 0 but operation of the PPC bit is controlled by the value of PLC. A description of this register follows.
Notes:
(1) Excludes the following addresses which can be used externally: $04, $05, $06, $07, and $0F.
(2) There must be no overlapping of internal and external memory spaces to avoid driving the data bus with more than one device.
(3) Assumes RAME (RAM Enable bit) is set.

Figure E-4. MC68701 Memory Map for Mode 0
Bit 0 PLC  The Programming Latch Control bit controls (a) a latch which captures the EPROM address to be programmed and (b) whether the PPC bit can be cleared. The latch is triggered by an MPU write to a location in the EPROM. This bit is set by Reset and can be cleared only in Mode 0. The PLC bit is defined as follows:

PLC = 0 EPROM address latch enabled; EPROM address is latched during MPU writes to the EPROM.
PLC = 1 EPROM address latch is transparent.

Bit 1 PPC  The Programming Power Control bit gates power from the RESET/Vpp pin to the EPROM programming circuit. PPC is set by Reset and whenever the PLC bit is set. It can be cleared only if (a) operating in Mode 0, and (b) if PLC has been previously cleared. The PPC bit is defined as follows:

PPC = 0 EPROM programming power (Vpp) enabled.
PPC = 1 EPROM programming power (Vpp) is not applied.

Bits 2-5  Unused.
Bit 6  RAME. RAM Enable bit. Refer to the RAM Control Register (Section 3.1.5.1.3).
Bit 7  STBY PWR. Standby Power bit. Refer to the RAM Control Register (Section 3.1.5.1.3).

Note that if PPC and PLC are set, they cannot be simultaneously cleared as the result of a single MPU write. The PLC bit must be cleared prior to attempting to clear PPC. If both PPC and PLC are clear, setting PLC will also set PPC. In addition, it is assumed that Vpp is applied to the RESET/Vpp pin whenever PPC is clear. If this is not the case, the results to the EPROM are undefined.

E.3 DESCRIPTION OF INTERNAL EPROM PROGRAMMING CIRCUITRY

A block diagram of the internal EPROM programming circuitry is shown in Figure E-5. The EPROM programming circuitry consists of (a) address and data latches (b) RAM/EPROM Control Register, (c) RESET/Vpp programming power, and (d) associated control logic. The output data buffer used for MPU reads of the EPROM is also shown for completeness.

Data associated with an MPU write to an EPROM address is always captured in an 8-bit data latch. The 11-bit EPROM address latch is transparent providing the PLC bit is set. When PLC is clear, however, it latches the address during MPU writes to the EPROM. When Vpp is subsequently applied to the EPROM by clearing the PPC bit, the “1’s” in the data latch are programmed into the EPROM location specified by the address latch.
Programming power, \( V_{\text{pp}} \), is used to program the EPROM and control some EPROM functions. Whenever \( V_{\text{pp}} \) is applied to the \( \text{RESET}/V_{\text{pp}} \) pin, it is always provided to the EPROM control circuits. This could adversely affect the result of an EPROM MPU read.

**NOTE**

While \( V_{\text{pp}} \) is applied to the \( \text{RESET}/V_{\text{pp}} \) pin, the result of an EPROM MPU read is undefined regardless of the operating mode or value of PPC.

**E.4 PROGRAMMING THE MC68701 EPROM**

Ultraviolet erasure will clear all bits of the EPROM to the "0" state. Note that this erased state differs from that of some other widely used EPROMs (such as the MCM68708) where the erased state is a "1". The MC68701 EPROM is programmed by erasing it to "0's" and entering "1's" into the desired bit locations.

When the MC68701 is released from Reset in Mode 0, a vector is fetched from location \$BFFE:\$BFFF. This provides a method for an external program to obtain control of the microcomputer with access to every location in the EPROM.

To program the EPROM, it is necessary to operate the MC68701 in Mode 0 under the control of a program resident in external memory which can facilitate loading and programming of the EPROM. After the pattern has been loaded into external memory, the EPROM can be programmed as follows:

a. Apply programming power (\( V_{\text{pp}} \)) to the \( \text{RESET}/V_{\text{pp}} \) pin.

b. Clear the PLC control bit and set the PPC bit by writing \$FE to the RAM/EPROM Control Register.

c. Write data to the next EPROM location to be programmed. Triggered by an MPU write to the EPROM, internal latches capture both the EPROM address and the data byte.

d. Clear the PPC bit for programming time, \( t_{\text{pp}} \), by writing \$FC to the RAM/EPROM Control Register and waiting for time, \( t_{\text{pp}} \). This step gates the programming power (\( V_{\text{pp}} \)) from the \( \text{RESET}/V_{\text{pp}} \) pin to the EPROM which programs the location.

e. Repeat steps b through d for each byte to be programmed.

f. Set PPC and PLC by writing \$FF to the RAM/EPROM Register.

g. Remove the programming power (\( V_{\text{pp}} \)) from the \( \text{RESET}/V_{\text{pp}} \) pin. The EPROM can now be read and verified.

Because the erased state of an EPROM byte is \$00, it is not necessary to program a location which is to contain \$00. Finally, it should be noted that the result of inadvertently programming a location more than once is the logical OR of the data patterns.

---

* A monitor called PRObug™ is available in a masked ROM which can be used to load a pattern and then program it into the EPROM. The monitor can be used with the MEX6801EVM Evaluation Module (see Appendix K). See PRObug manual for details. Motorola Microsystems, 3102 N. 56th St., Phoenix, Arizona 85018.
A routine which can be used to program the MC68701 EPROM is shown in Figure E-6. This non-reentrant routine requires four double byte variables named IMBEG, IMEND, PNTR, and WAIT to be initialized prior to entry to the routine. These variables indicate (a) the first and last memory locations which bound the data to be programmed into the EPROM, (b) the first EPROM location to be programmed, and (c) a quantity which can be used to generate the programming time delay. The last variable, WAIT, takes into account the MCU input crystal (or TTL-compatible clock) frequency to insure the programming time, tpp, is met. WAIT is defined as the number of MPU E-cycles that will occur in the real-time EPROM programming interval, tpp. For example, if tpp = 50 milliseconds and the MC68701 is being driven with a 4.00 MHz TTL-compatible clock:

\[
\text{WAIT (MPU E-cycles)} = \frac{tpp (\text{MCU INPUT FREQ})}{4 \times 10^6}
\]

\[
= \frac{(50000)(4 \times 10^6)}{4 \times 10^6}
\]

\[
= 50000
\]

E.5 ERASING THE MC68701 EPROM

The MC68701 EPROM can be erased by exposing it to high-intensity ultraviolet light of a particular wavelength. The erasure time is a function of the intensity of the light and the exposure time. The MC68701 Data Sheet should be referenced for the details involved in selection of vendor equipment.

NOTE

The MC68701 transparent lid should always be covered with an opaque material after erasing. This shields both the EPROM and dynamic light-sensitive nodes from unintentional exposure to ultraviolet light.
Fig. E-5. Block Diagram of MC68701 EPROM Programming Circuit
*** ROUTINE TO PROGRAM THE MC68701 EPROM ***

00001
00002
00003
00004
00005
00006
00007
00008
00009
00010
00011
00012
00013
00014
00015
00016
00017
00018
00019
00020
00021
00022
00023
00024
00025
00026
00027
00028
00029
00030
00031
00032
00033
00034
00035
00036
00037
00038
00039
00040
00041
00042
00043
00044
00045
00046
00047
00048
00049
00050
00051
00052
00053
00054
00055
00056
00057
00058

NAM EPROM
OPT ZO1, LLEN=80
*** ROUTINE TO PROGRAM THE MC68701 EPROM ***

*************************************************************
* EPROM -- A NON-REENTRANT ROUTINE TO PROGRAM THE MC68701 EPROM.
* THE ROUTINE PROGRAMS THE MC68701 EPROM
* STARTING AT ADDRESS "PNTR" FROM A BLOCK OF MEMORY STARTING AT "IMBEG"
* AND ENDING AT "IMEND".
* CALLING CONVENTION:
* JSR EPROM
* NOTES:
* 1. THE ROUTINE EXPECTS FOUR DOUBLE BYTE VALUES TO BE INITIALIZED PRIOR TO BEING CALLED.
  THESE VALUES ARE:
* IMBEG = A DOUBLE BYTE ADDRESS WHICH POINTS TO THE FIRST BYTE TO BE PROGRAMMED INTO THE EPROM.
* IMEND = A DOUBLE BYTE ADDRESS WHICH POINTS TO THE LAST BYTE TO BE PROGRAMMED INTO THE EPROM.
* PNTR = A DOUBLE BYTE ADDRESS WHICH POINTS TO THE FIRST BYTE IN THE EPROM TO BE PROGRAMMED.
* WAIT = A DOUBLE BYTE COUNTER VALUE WHICH IS A FUNCTION OF THE MCU INPUT FREQUENCY AND IS USED WITH THE OUTPUT COMPARISON FUNCTION TO GENERATE A 50 MSEC TIMEOUT. IT IS EQUIVALENT TO 50000 * (MCU INPUT FREQ) / 4 * 10**6 VALUES FOR TYPICAL INPUT FREQUENCIES ARE:

<table>
<thead>
<tr>
<th>WAIT</th>
<th>MCU INPUT FREQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>30615 (87797)</td>
<td>2.45 MHz</td>
</tr>
<tr>
<td>50000 (5C350)</td>
<td>4.00 MHz</td>
</tr>
<tr>
<td>61375 (5F5F)</td>
<td>4.91 MHz</td>
</tr>
</tbody>
</table>

* 2. IT IS ASSUMED THAT POWER (VPP) IS AVAILABLE TO THE RESET PIN FOR PROGRAMMING.
* 3. THIS ROUTINE PERFORMS NO ERROR CHECKING.

*************************************************************

Figure E-6. Programming the MC68701 EPROM: EPROM
00060
00061 * EQUATES
00062
00063 0008  A TCSR   EQU $08  TIMER CONTROL/STAT REGISTER
00064 0009  A TIMR   EQU $09  COUNTER REGISTER
00065 000B  A OUTCMP EQU $0B  OUTPUT COMPARE REGISTER
00066 0014  A EPMCNT EQU $14  RAM/EPROM CONTROL REGISTER
00067
00068 *
00069
00070 A 0080  ORG $80
00071 A 0080 0002  A IMBEG RMB 2  START OF MEMORY BLOCK
00072 A 0082 0002  A IMEND RMB 2  LAST BYTE OF MEMORY BLOCK
00073 A 0084 0002  A PNTR RMB 2  FIRST BYTE OF EPROM TO BE PGM'D
00074 A 0086 0002  A WAIT RMB 2  COUNTER VALUE
00075
00076 * EPROM STARTS HERE
00077
00078 A 3000  ORG $3000
00079 A 3000 DE 84  A EPROM  LDX PNTR  SAVE CALLING ARGUMENT
00080 A 3002 3C  PSHX  RESTORE WHEN DONE
00081 A 3003 DE 80  A  LDX IMBEG  USE STACK
00082
00083 A 3005 3C  EPR002 PSHX  SAVE POINTER ON STACK
00084 A 3006 86 FE  A  LDAA #$FE  REMOVE VPP, SET LATCH
00085 A 3008 97 14  A  LDAA EPMCNT  PPC=1, PLC=0
00086 A 300A 0000  A  LDAA X  MOVE DATA MEMORY-TO-LATCH
00087 A 300C DE 84  A  LDX PNTR  GET WHERE TO PUT IT
00088 A 3014 3005  A  STAA X  STASH AND LATCH
00089 A 3010 08  INX  NEXT ADDR
00090 A 3011 DF 84  A  STX PNTR  ALL SET FOR NEXT
00091 A 3013 86 FC  A  LDAA #$FC  ENABLE EPROM POWER (VPP)
00092 A 3015 97 14  A  STAA EPMCNT  PPC=0, PLC=0
00093
00094 * NOW WAIT FOR 50 MSEC TIMEOUT USING OUTPUT COMPARE.
00095
00096 A 3017 DC 86  A  LDD  WAIT  GET CYCLE COUNTER
00097 A 3019 D3 09  A  ADDDD  TIMER  BUMP CURRENT VALUE
00098 A 301B 7F 0008  A  CLR TCSR  CLEAR OCF
00099 A 301E DD 08  A  STD OUTCMP  SET OUTPUT COMPARE
00100 A 3020 86 40  A  LDAA #$40  NOW WAIT FOR OCF
00101
00102 A 3022 95 08  A  EPR004 BITA TCSR
00103 A 3024 27 FC 3022  BEQ EPR004  NOT YET
00104 A 3026 38  PULX  SETUP FOR NEXT ONE
00105 A 3027 08  INX  NEXT
00106 A 3028 9C 82  A  CPX IMEND  MAYBE DONE
00107 A 302A 23 D9 3005  BLS EPR002  NOT YET
00108 A 302C 86 FF  A  LDAA #$FF  REMOVE VPP, INHIBIT LATCH
00109 A 302E 97 14  A  STAA EPMCNT  EPROM CAN NOW BE READ
00110 A 3030 38  PULX  RESTORE PNTR
00111 A 3031 DF 84  A  STX PNTR
00112 A 3033 39  RTS  THAT’S ALL
00113
TOTAL ERRORS 00000--00000

Figure E-6. Programming the MC68701 EPROM: EPROM (Concluded)
APPENDIX F
CYCLE-BY-CYCLE BUS ACTIVITY

Table F-1 provides a detailed description of the information present on the Address Bus, Data Bus, and the Read/Write (R/W) line during each cycle of each instruction.

The information is useful in comparing actual with expected results during debug of both software and hardware as the program is executed. The information is categorized in groups according to addressing mode and number of cycles per instruction. In general, instructions with the same addressing mode and number of cycles execute in the same manner. Exceptions are indicated in the table.

Note that during MPU reads of internal locations, the resultant value will not appear on the external Data Bus except in Mode 0. "High order" byte refers to the most significant byte of a 16-bit value.

Table F-1. Cycle-By-Cycle Operation

<table>
<thead>
<tr>
<th>Address Mode &amp; Instructions</th>
<th>Cycles</th>
<th>Cycle #</th>
<th>Address Bus</th>
<th>R/W Line</th>
<th>Data Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMMEDIATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC EOR</td>
<td>2</td>
<td>1</td>
<td>Op Code Address</td>
<td>1</td>
<td>Op Code</td>
</tr>
<tr>
<td>ADD LDA</td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1</td>
<td>Operand Data</td>
</tr>
<tr>
<td>AND ORA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIT SBC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMP SUB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDS</td>
<td>3</td>
<td>1</td>
<td>Op Code Address</td>
<td>1</td>
<td>Op Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1</td>
<td>Operand Data (High Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Op Code Address + 2</td>
<td>1</td>
<td>Operand Data (Low Order Byte)</td>
</tr>
<tr>
<td>LDX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDS</td>
<td>4</td>
<td>1</td>
<td>Op Code Address</td>
<td>1</td>
<td>Op Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1</td>
<td>Operand Data (High Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Address Bus FFFF</td>
<td>1</td>
<td>Low Byte of Restart Vector</td>
</tr>
<tr>
<td>ADDD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| DIRECT                     |        |         |             |          |          |
| ADC EOR                    | 3      | 1       | Op Code Address | 1     | Op Code |
| ADD LDA                    |        | 2       | Op Code Address + 1 | 1 | Address of Operand |
| AND ORA                    |        | 3       | Address of Operand | 1 | Operand Data |
| BIT SBC                    |        |         |             |          |          |
| CMP SUB                    |        |         |             |          |          |
| STA                        | 3      | 1       | Op Code Address | 1     | Op Code |
|                            |        | 2       | Op Code Address + 1 | 1 | Destination Address |
|                            |        | 3       | Destination Address | 0 | Data from Accumulator |
| LDS                        | 4      | 1       | Op Code Address | 1     | Op Code |
|                            |        | 2       | Op Code Address + 1 | 1 | Address ofOperand |
|                            |        | 3       | Address of Operand | 1 | Operand Data (High Order Byte) |
|                            |        | 4       | Operand Address + 1 | 1 | Operand Data (Low Order Byte) |
| LDD                        |        |         |             |          |          |
| STS                        | 4      | 1       | Op Code Address | 1     | Op Code |
|                            |        | 2       | Op Code Address + 1 | 1 | Address ofOperand |
|                            |        | 3       | Address ofOperand | 0 | Register Data (High Order Byte) |
|                            |        | 4       | Address ofOperand + 1 | 0 | Register Data (Low Order Byte) |

| CPX                        | 5      | 1       | Op Code Address | 1     | Op Code |
|                            |        | 2       | Op Code Address + 1 | 1 | Address ofOperand |
|                            |        | 3       | Operand Address | 1     | Operand Data (High Order Byte) |
|                            |        | 4       | Operand Address + 1 | 1 | Operand Data (Low Order Byte) |
|                            |        | 5       | Address Bus FFFF | 1 | Low Byte of Restart Vector |
| ADDD                       |        |         |             |          |          |
| JSR                        | 5      | 1       | Op Code Address | 1     | Op Code |
|                            |        | 2       | Op Code Address + 1 | 1 | Irrelevant Data |
|                            |        | 3       | Subroutine Address | 1 | First Subroutine Op Code |
|                            |        | 4       | Stack Pointer | 0     | Return Address (Low Order Byte) |
|                            |        | 5       | Stack Pointer -1 | 0 | Return Address (High Order Byte) |
### Table F-1. Cycle-By-Cycle Operation (Continued)

<table>
<thead>
<tr>
<th>Address Mode &amp; Instructions</th>
<th>Cycles</th>
<th>Cycle #</th>
<th>Address Bus</th>
<th>R/W Line</th>
<th>Data Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EXTENDED</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>3</td>
<td>1</td>
<td>Op Code Address</td>
<td>1</td>
<td>Op Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1</td>
<td>Jump Address (High Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Op Code Address + 2</td>
<td>1</td>
<td>Jump Address (Low Order Byte)</td>
</tr>
<tr>
<td>ADC EOR</td>
<td>4</td>
<td>1</td>
<td>Op Code Address</td>
<td>1</td>
<td>Op Code</td>
</tr>
<tr>
<td>ADD LDA</td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1</td>
<td>Address of Operand</td>
</tr>
<tr>
<td>AND ORA</td>
<td></td>
<td>3</td>
<td>Op Code Address + 2</td>
<td>1</td>
<td>Address of Operand (Low Order Byte)</td>
</tr>
<tr>
<td>BIT SBC</td>
<td></td>
<td>4</td>
<td>Address ofOperand</td>
<td>1</td>
<td>Operand Data</td>
</tr>
<tr>
<td>CMP SUB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STA</td>
<td>4</td>
<td>1</td>
<td>Op Code Address</td>
<td>1</td>
<td>Op Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1</td>
<td>Destination Address (High Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Op Code Address + 2</td>
<td>1</td>
<td>Destination Address (Low Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Operand Destination Address</td>
<td>0</td>
<td>Data from Accumulator</td>
</tr>
<tr>
<td>LDS</td>
<td>5</td>
<td>1</td>
<td>Op Code Address</td>
<td>1</td>
<td>Op Code</td>
</tr>
<tr>
<td>LDX</td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1</td>
<td>Address of Operand</td>
</tr>
<tr>
<td>LDD</td>
<td></td>
<td>3</td>
<td>Op Code Address + 2</td>
<td>1</td>
<td>Address of Operand (Low Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Address ofOperand</td>
<td>1</td>
<td>Operand Data (High Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Address ofOperand + 1</td>
<td>1</td>
<td>Operand Data (Low Order Byte)</td>
</tr>
<tr>
<td>STS</td>
<td>5</td>
<td>1</td>
<td>Op Code Address</td>
<td>1</td>
<td>Op Code</td>
</tr>
<tr>
<td>STX</td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1</td>
<td>Address of Operand</td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td>3</td>
<td>Op Code Address + 2</td>
<td>1</td>
<td>Address of Operand (Low Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Address ofOperand</td>
<td>0</td>
<td>Operand Data (High Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Address ofOperand + 1</td>
<td>0</td>
<td>Operand Data (Low Order Byte)</td>
</tr>
<tr>
<td>ASL LSR</td>
<td>6</td>
<td>1</td>
<td>Op Code Address</td>
<td>1</td>
<td>Op Code</td>
</tr>
<tr>
<td>ASR NEG</td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1</td>
<td>Address of Operand</td>
</tr>
<tr>
<td>CLR ROL</td>
<td></td>
<td>3</td>
<td>Op Code Address + 2</td>
<td>1</td>
<td>Address of Operand (Low Order Byte)</td>
</tr>
<tr>
<td>COM ROR</td>
<td></td>
<td>4</td>
<td>Address ofOperand</td>
<td>1</td>
<td>Current Operand Data</td>
</tr>
<tr>
<td>DEC TST</td>
<td></td>
<td>5</td>
<td>Address Bus FFFF</td>
<td>1</td>
<td>Low Byte of Restart Vector</td>
</tr>
<tr>
<td>INC</td>
<td></td>
<td>6</td>
<td>Address ofOperand</td>
<td>0</td>
<td>New Operand Data</td>
</tr>
<tr>
<td>CPX</td>
<td>6</td>
<td>1</td>
<td>Op Code Address</td>
<td>1</td>
<td>Op Code</td>
</tr>
<tr>
<td>SUBD</td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1</td>
<td>Operand Address (High Order Byte)</td>
</tr>
<tr>
<td>ADDD</td>
<td></td>
<td>3</td>
<td>Op code Address + 2</td>
<td>1</td>
<td>Operand Address (Low Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Operand Address</td>
<td>1</td>
<td>Operand Data (High Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Operand Address + 1</td>
<td>1</td>
<td>Operand Data (Low Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Address Bus FFFF</td>
<td>1</td>
<td>Low Byte of Restart Vector</td>
</tr>
<tr>
<td>JSR</td>
<td>6</td>
<td>1</td>
<td>Op Code Address</td>
<td>1</td>
<td>Op Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1</td>
<td>Address of Subroutine (High Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Op Code Address + 2</td>
<td>1</td>
<td>Address of Subroutine (Low Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Subroutine Starting Address</td>
<td>1</td>
<td>Op Code of Next Instruction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Stack Pointer</td>
<td>0</td>
<td>Return Address (Low Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Stack Pointer - 1</td>
<td>0</td>
<td>Return Address (High Order Byte)</td>
</tr>
</tbody>
</table>
### Table F-1. Cycle-By-Cycle Operation (Continued)

<table>
<thead>
<tr>
<th>Address Mode &amp; Instructions</th>
<th>Cycles</th>
<th>Cycle #</th>
<th>Address Bus</th>
<th>R/W Line</th>
<th>Data Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEXED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>3</td>
<td>1</td>
<td>Op Code Address</td>
<td>1 Op Code</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1 Offset</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Address Bus FFFF</td>
<td>1 Low Byte of Restart Vector</td>
<td></td>
</tr>
<tr>
<td>ADC EOR</td>
<td>4</td>
<td>1</td>
<td>Op Code Address</td>
<td>1 Op Code</td>
<td></td>
</tr>
<tr>
<td>ADD LDA</td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1 Offset</td>
<td></td>
</tr>
<tr>
<td>AND ORA</td>
<td></td>
<td>3</td>
<td>Address Bus FFFF</td>
<td>1 Low Byte of Restart Vector</td>
<td></td>
</tr>
<tr>
<td>BIT SBC</td>
<td></td>
<td>4</td>
<td>Index Register Plus Offset</td>
<td>1 Operand Data</td>
<td></td>
</tr>
<tr>
<td>CMP SUB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STA</td>
<td>4</td>
<td>1</td>
<td>Op Code Address</td>
<td>1 Op Code</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1 Offset</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Address Bus FFFF</td>
<td>1 Low Byte of Restart Vector</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Index Register Plus Offset</td>
<td>0 Operand Data</td>
<td></td>
</tr>
<tr>
<td>LDS</td>
<td>5</td>
<td>1</td>
<td>Op Code Address</td>
<td>1 Op Code</td>
<td></td>
</tr>
<tr>
<td>LDX</td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1 Offset</td>
<td></td>
</tr>
<tr>
<td>LDD</td>
<td></td>
<td>3</td>
<td>Address Bus FFFF</td>
<td>1 Low Byte of Restart Vector</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Index Register Plus Offset</td>
<td>1 Operand Data (High Order Byte)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Index Register Plus Offset + 1</td>
<td>1 Operand Data (Low Order Byte)</td>
<td></td>
</tr>
<tr>
<td>STS</td>
<td>5</td>
<td>1</td>
<td>Op Code Address</td>
<td>1 Op Code</td>
<td></td>
</tr>
<tr>
<td>STX</td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1 Offset</td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td>3</td>
<td>Address Bus FFFF</td>
<td>1 Low Byte of Restart Vector</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Index Register Plus Offset</td>
<td>0 Operand Data (High Order Byte)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Index Register Plus Offset + 1</td>
<td>0 Operand Data (Low Order Byte)</td>
<td></td>
</tr>
<tr>
<td>ASL LSR</td>
<td>6</td>
<td>1</td>
<td>Op Code Address</td>
<td>1 Op Code</td>
<td></td>
</tr>
<tr>
<td>ASR NEG</td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1 Offset</td>
<td></td>
</tr>
<tr>
<td>CLR ROL</td>
<td></td>
<td>3</td>
<td>Address Bus FFFF</td>
<td>1 Low Byte of Restart Vector</td>
<td></td>
</tr>
<tr>
<td>COM ROR</td>
<td></td>
<td>4</td>
<td>Index Register Plus Offset</td>
<td>1 Current Operand Data</td>
<td></td>
</tr>
<tr>
<td>DEC TST (1)</td>
<td></td>
<td>5</td>
<td>Address Bus FFFF</td>
<td>1 Low Byte of Restart Vector</td>
<td></td>
</tr>
<tr>
<td>INC</td>
<td></td>
<td>6</td>
<td>Index Register Plus Offset</td>
<td>0 New Operand Data</td>
<td></td>
</tr>
<tr>
<td>CPX</td>
<td>6</td>
<td>1</td>
<td>Op Code Address</td>
<td>1 Op Code</td>
<td></td>
</tr>
<tr>
<td>SUBD</td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1 Offset</td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td></td>
<td>3</td>
<td>Address Bus FFFF</td>
<td>1 Low Byte of Restart Vector</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Index Register + Offset</td>
<td>1 Operand Data (High Order Byte)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Index Register + Offset + 1</td>
<td>1 Operand Data (Low Order Byte)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Address Bus FFFF</td>
<td>1 Low Byte of Restart Vector</td>
<td></td>
</tr>
<tr>
<td>JSR</td>
<td>6</td>
<td>1</td>
<td>Op Code Address</td>
<td>1 Op Code</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Op Code Address + 1</td>
<td>1 Offset</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Address Bus FFFF</td>
<td>1 Low Byte of Restart Vector</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Index Register + Offset</td>
<td>1 First Subroutine Op Code</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Stack Pointer</td>
<td>0 Return Address (Low Order Byte)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Stack Pointer - 1</td>
<td>0 Return Address (High Order Byte)</td>
<td></td>
</tr>
<tr>
<td>Address Mode &amp; Instructions</td>
<td>Cycles</td>
<td>Cycle #</td>
<td>Address Bus</td>
<td>R/W Line</td>
<td>Data Bus</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------</td>
<td>---------</td>
<td>-------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>INHERENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABA DAA SEC ASL DEC SEI ASR INC SEV CBA LSR TAB CLC NEG TAP CLI NOP TBA CLR ROL TPA CLV ROR TST COM SBA</td>
<td>2</td>
<td>1 2</td>
<td>Op Code Address Op Code Address +1</td>
<td>1 1</td>
<td>Op Code Op Code of Next Instruction</td>
</tr>
<tr>
<td>ABX</td>
<td>3</td>
<td>1 2 3</td>
<td>Op Code Address Op Code Address +1 Address Bus FFFF</td>
<td>1 1 1</td>
<td>Op Code Irrelevant Data Low Byte of Restart Vector</td>
</tr>
<tr>
<td>ASLD LSRD</td>
<td>3</td>
<td>1 2 3</td>
<td>Op Code Address Op Code Address +1 Address Bus FFFF</td>
<td>1 1 1</td>
<td>Op Code Irrelevant Data Low Byte of Restart Vector</td>
</tr>
<tr>
<td>DES INS</td>
<td>3</td>
<td>1 2 3</td>
<td>Op Code Address Op Code Address +1 Previous Register Contents</td>
<td>1 1 1</td>
<td>Op Code Op Code of Next Instruction Irrelevant Data</td>
</tr>
<tr>
<td>INX DEX</td>
<td>3</td>
<td>1 2 3</td>
<td>Op Code Address Op Code Address +1 Address Bus FFFF</td>
<td>1 1 1</td>
<td>Op Code Op Code of Next Instruction Low Byte of Restart Vector</td>
</tr>
<tr>
<td>PSHA PSHB</td>
<td>3</td>
<td>1 2 3</td>
<td>Op Code Address Op Code Address +1 Stack Pointer</td>
<td>1 1 0</td>
<td>Op Code Op Code of Next Instruction Accumulator Data</td>
</tr>
<tr>
<td>TSX</td>
<td>3</td>
<td>1 2 3</td>
<td>Op Code Address Op Code Address +1 Stack Pointer</td>
<td>1 1 1</td>
<td>Op Code Op Code of Next Instruction Irrelevant Data</td>
</tr>
<tr>
<td>TXS</td>
<td>3</td>
<td>1 2 3</td>
<td>Op Code Address Op Code Address +1 Stack Pointer</td>
<td>1 1 1</td>
<td>Op Code Op Code of Next Instruction Irrelevant Data</td>
</tr>
<tr>
<td>PULA PULB</td>
<td>4</td>
<td>1 2 3 4</td>
<td>Op Code Address Op Code Address +1 Stack Pointer Stack Pointer +1</td>
<td>1 1 1 1</td>
<td>Op Code Op Code of Next Instruction Irrelevant Data Operand Data from Stack</td>
</tr>
<tr>
<td>PSHX</td>
<td>4</td>
<td>1 2 3 4</td>
<td>Op Code Address Op Code Address +1 Stack Pointer Stack Pointer</td>
<td>1 1 0 0</td>
<td>Op Code Irrelevant Data Index Register (Low Order Byte) Index Register (High Order Byte)</td>
</tr>
<tr>
<td>PULX</td>
<td>5</td>
<td>1 2 3 4 5</td>
<td>Op Code Address Op Code Address +1 Stack Pointer Stack Pointer +1 Stack Pointer +2</td>
<td>1 1 1 1 1</td>
<td>Op Code Irrelevant Data Index Register (Low Order Byte) Index Register (High Order Byte)</td>
</tr>
<tr>
<td>Address Mode &amp; Instructions</td>
<td>Cycles</td>
<td>Cycle #</td>
<td>Address Bus</td>
<td>R/W Line</td>
<td>Data Bus</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------</td>
<td>---------</td>
<td>-------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>INHERENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTS</td>
<td>5</td>
<td>1</td>
<td>Op Code Address</td>
<td>1</td>
<td>Op Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Op Code Address +1</td>
<td>1</td>
<td>Irrelevant Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Stack Pointer</td>
<td>1</td>
<td>Irrelevant Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Stack Pointer +1</td>
<td>1</td>
<td>Address of Next Instruction (High Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Stack Pointer +2</td>
<td>1</td>
<td>Address of Next Instruction (Low Order Byte)</td>
</tr>
<tr>
<td>WAI</td>
<td>9</td>
<td>1</td>
<td>Op Code Address</td>
<td>1</td>
<td>Op Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Op Code Address +1</td>
<td>1</td>
<td>Op Code of Next Instruction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Stack Pointer</td>
<td>0</td>
<td>Return Address (Low Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Stack Pointer -1</td>
<td>0</td>
<td>Return Address (High Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Stack Pointer -2</td>
<td>0</td>
<td>Index Register (Low Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Stack Pointer -3</td>
<td>0</td>
<td>Index Register (High Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Stack Pointer -4</td>
<td>0</td>
<td>Contents of Accumulator A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Stack Pointer -5</td>
<td>0</td>
<td>Contents of Accumulator B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Stack Pointer -6</td>
<td>0</td>
<td>Contents of Cond. Code Register</td>
</tr>
<tr>
<td>MUL</td>
<td>10</td>
<td>1</td>
<td>Op Code Address</td>
<td>1</td>
<td>Op Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Op Code Address +1</td>
<td>1</td>
<td>Irrelevant Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Address Bus FFFF</td>
<td>1</td>
<td>Low Byte of Restart Vector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Address Bus FFFF</td>
<td>1</td>
<td>Low Byte of Restart Vector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Address Bus FFFF</td>
<td>1</td>
<td>Low Byte of Restart Vector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Address Bus FFFF</td>
<td>1</td>
<td>Low Byte of Restart Vector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Address Bus FFFF</td>
<td>1</td>
<td>Low Byte of Restart Vector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Address Bus FFFF</td>
<td>1</td>
<td>Low Byte of Restart Vector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Address Bus FFFF</td>
<td>1</td>
<td>Low Byte of Restart Vector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Address Bus FFFF</td>
<td>1</td>
<td>Low Byte of Restart Vector</td>
</tr>
<tr>
<td>RTI</td>
<td>10</td>
<td>1</td>
<td>Op Code Address</td>
<td>1</td>
<td>Op Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Op Code Address +1</td>
<td>1</td>
<td>Irrelevant Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Stack Pointer</td>
<td>1</td>
<td>Irrelevant Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Stack Pointer +1</td>
<td>1</td>
<td>Contents of Cond. Code Reg. from Stack</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Stack Pointer +2</td>
<td>1</td>
<td>Contents of Accumulator B from Stack</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Stack Pointer +3</td>
<td>1</td>
<td>Contents of Accumulator A from Stack</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Stack Pointer +4</td>
<td>1</td>
<td>Index Register from Stack (High Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Stack Pointer +5</td>
<td>1</td>
<td>Index Register from Stack (Low Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Stack Pointer +6</td>
<td>1</td>
<td>Next Instruction Address from Stack (High Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Stack Pointer +7</td>
<td>1</td>
<td>Next Instruction Address from Stack (Low Order Byte)</td>
</tr>
<tr>
<td>SWI</td>
<td>12</td>
<td>1</td>
<td>Op Code Address</td>
<td>1</td>
<td>Op Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Op Code Address +1</td>
<td>1</td>
<td>Irrelevant Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Stack Pointer</td>
<td>0</td>
<td>Return Address (Low Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Stack Pointer -1</td>
<td>0</td>
<td>Return Address (High Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Stack Pointer -2</td>
<td>0</td>
<td>Index Register (Low Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Stack Pointer -3</td>
<td>0</td>
<td>Index Register (High Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>Stack Pointer -4</td>
<td>0</td>
<td>Contents of Accumulator A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Stack Pointer -5</td>
<td>0</td>
<td>Contents of Accumulator B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>Stack Pointer -6</td>
<td>0</td>
<td>Contents of Cond. Code Register</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Stack Pointer -7</td>
<td>1</td>
<td>Irrelevant Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>Vector Address FFFA (Hex)</td>
<td>1</td>
<td>Address of Subroutine (High Order Byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>Vector Address FFFB (Hex)</td>
<td>1</td>
<td>Address of Subroutine (Low Order Byte)</td>
</tr>
</tbody>
</table>
Table F-1. Cycle-By-Cycle Operation
(Concluded)

<table>
<thead>
<tr>
<th>Address Mode &amp; Instructions</th>
<th>Cycles</th>
<th>Cycle #</th>
<th>Address Bus</th>
<th>R/W Line</th>
<th>Data Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELATIVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCC BHI BNE BLO</td>
<td>3</td>
<td>1</td>
<td>Op Code Address</td>
<td>1 Op Code</td>
<td></td>
</tr>
<tr>
<td>BCS BLE BPL BHS</td>
<td></td>
<td>2</td>
<td>Op Code Address +1</td>
<td>1 Branch Offset</td>
<td></td>
</tr>
<tr>
<td>BEQ BLS BRA BRN</td>
<td></td>
<td>3</td>
<td>Address Bus FFFF</td>
<td>1 Low Byte of Restart Vector</td>
<td></td>
</tr>
<tr>
<td>BGE BLT BVC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BGT BMT BVS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSR</td>
<td>6</td>
<td>1</td>
<td>Op Code Address</td>
<td>1 Op Code</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Op Code Address +1</td>
<td>1 Branch Offset</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Address Bus FFFF</td>
<td>1 Low Byte of Restart Vector</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Subroutine Starting Address</td>
<td>1 Op Code of Next Instruction</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Stack Pointer</td>
<td>0 Return Address (Low Order Byte)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Stack Pointer -1</td>
<td>0 Return Address (High Order Byte)</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX G
GLOSSARY

This glossary provides an explanation of terms found in the text and Data Sheet. While some terms are very general, others are peculiar to the MC6801/MC6803/MC6803NR/MC68701. All terms are equally applicable to all of these MPUs unless specifically excluded.

absolute address  The physical location of a byte as specified using either the direct or extended addressing modes.

access  A read or write of a particular address.

access time  During a MPU read bus cycle, it is the period of time from when address is valid on the address bus until data is required to be valid.

ACIA (MC6850 Asynchronous Communications Interface Adapter)  An M6800 Family device which can be used to interface the MPU with devices utilizing an asynchronous NRZ serial format.

ACCA (Accumulator A)  One of two MC6801 accumulators.

ACCB (Accumulator B)  One of two MC6801 accumulators.

accumulator  A register used to contain the result of arithmetic and logical operations. The MC6801 has two such 8-bit registers referred to as accumulator A and accumulator B. In addition, the A and B accumulators can be concatenated to form a single 16-bit accumulator referred to as the D accumulator.

address  A location defined by logic levels appearing on the address bus. The set of levels to which a device will respond is known as the “address(es)” of the device.

address/input port  An MC6801 Port 4 configuration (Modes 5 and 6) in which any or all Port 4 lines can be used as either (a) address outputs or (b) data inputs. The configuration of each bit is determined its the corresponding bit in the Port 4 Data Direction Register (1 = output, 0 = input).

Address Strobe  (See AS)

addressable  Capable of being directly accessed using an MPU instruction. Bytes and double bytes are addressable in M6800 Family devices. Nibbles, bits, and values larger than two bytes are not directly addressable.
**address bus** A set of conductors used to carry an address. With respect to the MC6801, the number of lines is dependent upon the operating mode and configuration.

**algorithm** A step-by-step procedure which produces a specified result.

**alphanumeric** The set of letters (A-Z) and numerals (0-9).

**AS (Address Strobe)** An MC6801 output signal on the SCI pin in the expanded multiplexed modes (0, 1, 2, 3 and 6). It can be used to control a transparent latch to de-multiplex the least significant eight address lines from the address/data bus.

**ASCII (American Standard Code for Information Interchange)** A code used in data communications to represent characters, numbers, and control characters. The code uses seven bits to define 128 possible characters.

**assembler** A computer program which converts assembly language programs into machine language. If the machine language is not compatible with the computer which executes the assembler, it is called a "cross-assembler".

**assert** To make a signal "active" independent of whether the voltage is high or low.

**background** In an interrupt-driven program, background is used to describe that portion of execution time during which the MPU is not responding to an interrupt. When responding to interrupts, the program is said to be executing in "foreground".

**backward reference** With respect to an assembly language program, it refers to the use of a label in the operand field which has already appeared in the label field of a prior statement.

**baud** A unit of data transmission speed. In this text, it is equivalent to "bits per second".

**BCD (Binary Coded Decimal)** A scheme for encoding the decimal digits, 0 to 9, in four bits which range from binary 0000 to 1001. If two BCD digits are encoded in a single byte, it is referred to as packed BCD. Unpacked BCD contains a single BCD digit per byte typically right justified.

**binary** A number system containing two digits: 0 and 1.

**binary coded decimal** (See BCD)

**Bi-Phase** One of two SCI serial communications format. In bi-phase format, the level toggles at the start of each bit and, if the data is a "1", at the half-bit time. Bi-phase format is notable for its relatively large tolerance of transmitter and receiver clocking mismatch.

**bit** A unit of information having only two states. It is derived from binary and digit.
**bit rate** A term used in serial communications to describe the data rate. It is commonly expressed as the number of bits per second.

**bit time** The reciprocal of the bit rate. The length of time devoted to a single bit in serial data.

**Booth's Algorithm** An algorithm for performing signed multiplication using a binary machine.

**branch** To cause execution of an instruction other than the next sequential one. This is implemented by loading the Program Counter with the address of the branch destination. In the MC6801, a jump instruction performs similar operation except that (1) branch instructions use the relative addressing mode whereas jump instructions use extended or direct addressing and (2) branch instructions can be made conditional whereas jump instructions can not.

**break condition** In asynchronous communications, a break condition is generated by holding the communications line low for ten or more consecutive bit times.

**buffer** Temporary storage that is used to compensate for differences in data flow rates.

**bug** The result of a design or implementation error in either hardware or software.

**bus** A set of two or more parallel conductors which carry data.

**bus arbitrator** A circuit employed to determine which of two or more requesting devices is to be given control of the bus.

**bus cycle** The bus activity which occurs between two successive negative edges of E (Enable).

**byte** A group of eight (typically) adjacent bits. A byte is the smallest quantity which can be directly accessed using an MC6801 instruction.

---

**Carry Bit** (See C-bit)

**C-bit (Carry Bit)** Bit 0 of the Condition Code Register which is set when a carry-out or borrow occurs during an arithmetic operation. The BCC and BCS instructions provide a branch based on the value of the C-bit.

**CC1:CC0 (Clock Control)** Bits 2 and 3 of the Rate and Mode Control Register which define the following SCI variables: (a) the source of the serial clock, (b) the serial data format, and (c) whether or not the internal bit rate clock will be supplied as an output.

**CCR (Condition Code Register)** A 6-bit register used to (a) control maskable interrupts, (b) control conditional branches, (c) reflect the outcome of certain operations, and (d) adjust the results in BCD additions.
character  An element of an alphabet typically encoded into five to eight bits. The ASCII character set is a 7-bit representation of characters.

chip  A single unpackaged integrated circuit.

chip select  An input signal for M6800 Family devices which indicates that the particular device is being addressed.

clear  To initialize a bit or several bits to "0" in the MC6801. Bytes can be cleared using the CLR instruction. Selected bits can be cleared by using the AND instruction with a suitable mask.

clock  A signal used for timing or a circuit which generates a timing signal.

CMOS (Complementary Metal Oxide Semiconductor)  An integrated circuit technology characterized by low power consumption, and high device densities.

code  In software, it is a term used to indicate instructions to a computer.

complement  A value obtained by toggling each bit to its opposite state. This result is also called the one's complement of the value.

concatenate  To join together.

Condition Code Register  (See CCR)

conditional branches  An operation which could result in execution of an instruction other than the next sequential one depending upon a bit or combination of bits in the Condition Code Register.

configuration  With respect to the MC6801, it refers to a particular functionality of its pins and registers. Some pin and register functionality is operating mode dependent.

coupler  A MOSFET used in MOS circuits as a bi-directional transfer device. Enhancement mode devices are used for couplers in the MC6801. If the gate input is high, the coupler is enabled ("on") and current flows through the device. If the gate input is low, it is not enabled ("off").

CPU (Central Processing Unit)  That part of the MC6801 which is responsible for processing instructions. This term is equivalent to MPU in MC6801 literature.

D accumulator  An MC6801 16-bit accumulator consisting of the A accumulator concatenated with the B accumulator. The A accumulator contains the most significant byte.

D flip-flop  A device which transfers the level at the input (D) to the output (Q) in response to a specified transition at its clock input (CLK).
Darlington drive current  The current available (I\text{OH}) at a given voltage (V_0) which is used to drive a Darlington transistor configuration.

data  Unprocessed information.

data bus  A collection of parallel lines used to transmit data between a processor and peripheral devices. The MC6801 data bus consists of eight lines (D0-D7) and is controlled by a timing signal (E) and a transfer direction signal (R/W).

data direction register (See DDR)

data port  The MC6801 hardware elements required to perform data transfers using the data register assigned to the port. When functioning as a data port, it is controlled by a data direction register and the programmer has direct access to the port pins using the assigned data register.

Data Register  A register used to perform data transfers with an external device. MC6801 registers have a single address but are implemented using separate input and output registers. The Read/Write line is used during an access to discriminate between the two registers.

DDR (Data Direction Register)  A register used to define the direction of data transfer of each bit in an associated data register. In all MC6801 Data Direction Registers, a "0" defines a bit (or pin) as an input whereas a "1" defines it as an output.

debug  To remove or correct bugs, or errors, in a system.

demultiplex  To separate two or more signals which are transmitted at different times using common lines. With respect to the MC6801, this term refers to separating the least significant eight lines of the address bus (A0-A7) from the data bus (D0-D7). A control line, AS, is provided for this purpose.

deployment mode transistor  A field effect transistor (FET) which has appreciable channel current when zero gate-source voltage is applied. Channel conduction can be increased or decreased by application of a gate-source voltage of the correct polarity. In NMOS, a negative voltage is required to decrease channel conduction to virtually zero.

direct addressing mode  An addressing scheme in which the least significant byte of the address is explicitly contained in the second byte of the instruction and the most significant byte is implicitly $00$.

double byte  Two contiguous bytes which, taken jointly, define a 16-bit value. Double bytes are stored in memory with the most significant byte having the lower address.

double byte reads/writes  Accesses which involve two consecutive read or write bus cycles. Instructions which perform double byte reads include LDX, LDD, SUBD, ADDD, PULX, and LDS. Instructions which perform double byte writes include STX, STS, PSHX, and STD.

duty cycle  With respect to a square waveform, it is the ratio of the high time to the period, expressed as a percentage. A perfect square wave has a duty cycle of 50\%. 

G-5
**dynamic offset** An offset value used in the indexed addressing mode which is defined during execution as opposed to during assembly of the program. The MC6801 instruction, ABX, can be used to obtain a dynamic offset.

---

**E (Enable)** A timing signal used to synchronize bus transfers. A bus cycle occurs between consecutive negative edges of E.

**EA (Effective Address)** An address obtained by evaluating the addressing mode specified by an instruction.

**E-Cycle (Enable Cycle)** The period of time between consecutive negative edges of E. All MC6801 timing data given as "cycles" is with respect to E unless otherwise specified.

**edge-sensitive** An attribute of an input circuit which provides a single response for a specified level transition. Each subsequent response requires an additional transition. NMI and IS3 are MC6801 inputs which have this property.

**effective address** (See EA)

**EICI (Enable Input Capture Interrupt)** Bit 4 of the Timer Control and Status Register (TCSR). EICI defines whetherIRQ2 is asserted in response to a proper level transition on P20. If EICI is set, IRQ2 is asserted while ICF is set. The I-bit must be clear to obtain an IRQ2 interrupt. Which level transition is "proper" is defined by the IEDG bit.

**Enable** (See E)

**enhancement mode transistor** A field-effect transistor which has virtually no channel conduction when a zero gate-source voltage is applied. Channel conduction can be increased by applying a gate-source voltage of appropriate polarity. In NMOS, a positive voltage increases channel conduction.

**EOCI (Enable Output Compare Interrupt)** Bit 3 of the Timer Control and Status Register (TCSR). EOCI defines whether IRQ2 is asserted in response to a match between the Counter and Output Compare Registers. If the bit is set, IRQ2 will be asserted while OCF is set. The I-bit must be clear in order to obtain an IRQ2 interrupt.

**EPROM (Erasable Programmable Read-Only Memory)** A read-only memory which is both erasable and programmable.

**ETOI (Enable Timer Overflow Interrupt)** Bit 2 of the Timer Control and Status Register (TCSR). ETOI defines whether IRQ2 is asserted when the Counter Register contains all ones (i.e., counter overflow). If the bit is set, IRQ2 is asserted while TOF is set. The I-bit must be clear in order to obtain an IRQ2 interrupt.
exclusive OR  A boolean operation in which the outcome is ‘‘0’’ if both inputs are identical and ‘‘1’’ if they are not. The EOR instruction implements this operation in the MC6801.

expanded multiplexed mode  One of three fundamental operating modes of the MC6801 consisting of Modes 0, 1, 2, 3, and 6. In this configuration, the MC6801 can access an expanded 64K address space using a multiplexed address/data bus consisting of A0-A7 time multiplexed with D0-D7. Port 4 provides A8-A15.

expanded non-multiplexed mode  One of three fundamental operating modes of the MC6801 consisting only of Mode 5. This mode can directly access an additional 256 bytes of external address space using separate address (A0-A7) and data (D0-D7) buses.

EXTAL2  An MC6801 input pin which can be used as (a) one of two quartz crystal resonator inputs with XTAL1 providing the other input, or (b) an input for a TTL-compatible clock with XTAL1 grounded.

extended addressing mode  An addressing mode in which the effective address of the instruction is explicitly specified in the second and third bytes of an instruction.

external address space  The set of addresses which are not defined as internal to the MC6801 for a given operating mode.

-ffff-

fall time  The time required for a voltage level change from $V_{IH}$ to $V_{IL}$ (an input) or $V_{OH}$ to $V_{OL}$ (an output).

falling edge  A high to low level transition.

firmware  Software that is implemented in hardware (e.g., EPROM or masked ROM).

$fo$ (See 4$fo$)

foreground  That portion of execution time in an interrupt-driven program which is used to service interrupts. When not responding to interrupts, the program is considered to be operating in “background.”

forward reference  With respect to an assembly language program, it refers to the use of a label in the operand field which has not yet appeared in the label field of a prior statement. When the assembler has a forward reference in the operand field and it must choose between direct and extended addressing, the extended addressing mode will always be chosen.

4$fo$ (Four times E-cycle Frequency from an External Clock)  The frequency of an external TTL compatible clock which is connected to EXTAL1. Note that $fo$ is equal to the frequency of E.
framing error In asynchronous communications, it is the absence of a stop bit ("1") in the tenth bit time. It can occur only during serial reception and the SCI will set the ORFE bit to indicate this condition. It can indicate either misframed data or a break condition.

full duplex The capability of transmitting and receiving data simultaneously.

fXTAL (Input Crystal Frequency) The frequency of a quartz crystal resonator used to drive the MC6801. Note that the frequency of E is equal to fXTAL/4.

global variable A variable which can be accessed by all subroutines within a program.

H-bit (Half-carry bit) Bit 5 of the Condition Code Register (CCR). It is set during certain instructions (ADD, ADC, ABA) to indicate a half carry from bit 3 and is used only in the DAA instruction to adjust the results of BCD addition.

half-carry bit (See H-bit)

half duplex The capability of transmitting and receiving but not simultaneously.

hardware vectored interrupts A mechanism for jumping directly to a device-dependent address (vector) in response to an interrupt. The programmable timer and the SCI provide hardware vectored interrupts using an internal priority encoder.

hexadecimal A number system which uses 16 as a base consisting of the digits 0 to 9 and the letters A to F. The decimal values 10-15 are represented by the letters A-F, respectively. The hexadecimal system is used as a convenience in representing binary values. Each group of four binary digits can be combined to form a single hexadecimal digit. Motorola assemblers and literature commonly use a dollar sign ("\$") prefix to indicate a hexadecimal value. For example, \$23 is equivalent to 35 decimal.

high-impedance state A configuration in which a device acts as a negligible load. Devices capable of being switched to this state are effectively disconnected from a bus. This is normally referred to as the third state of a three-state device.

HMOS (High density, short-channel MOS) An integrated circuit technology characterized by its high density and low power consumption. As a refinement of the NMOS technology, it can typically achieve twice the circuit density and four times the speed-power product.

hold time The interval of time following a clock transition during which the logic levels must remain constant in order to be reliably recognized.
I-bit (Interrupt Mask Bit)  Bit 4 of the Condition Code Register (CCR). It is used to enable (if clear) or mask (if set) IRQ1 and IRQ2 interrupts. It does not affect NMI interrupts or the SWI instruction and is set during Reset and all interrupt sequences.

ICF (Input Capture Flag)  Bit 7 of the Timer Control and Status Register which is used to indicate that a proper transition has occurred on the P20 pin. The Counter Register is also transferred to the Input Capture Register when this occurs. Which transition is proper is defined by the IEDG bit.

idle bus cycle  A bus cycle during which the MCU performs internal operations. Externally, most idle bus cycles appear as an MPU read of the LSB of the Reset Vector ($FFFF). Data obtained from an idle bus cycle is ignored.

IEDG (Input Edge)  Bit 1 of the Timer Control and Status Register. IEDG is used to define which edge of a signal present at P20 initiates the input capture function.

IIn (Input Load Current) (Input Leakage Current)  The maximum input or output current when configured as an input. The input load current applies to Port 4 pins and IS3 which have internal pullup resistors. The input leakage current applies to NMI, IRQ1 and RESET which are always configured as inputs.

immediate addressing mode  An MC6801 addressing mode in which the value of the operand — as opposed to the address — is explicitly contained in the byte(s) which follow the opcode. The size of the operand matches the size of the register specified as the destination.

inclusive OR  A Boolean function in which the result is a “1” if any of the inputs is a “1”; otherwise, the result is “0”. The ORAA and ORAB instructions implement this function in the MC6801.

index register  A register used primarily to contain the address of (i.e., point to) another location. In the MC6801, the X (or IX) Register is used for this purpose.

indexed addressing mode  An addressing mode which contains an unsigned offset byte in the second byte of the instruction. The most significant byte of the offset is zero. The effective address of an indexed instruction is computed by adding the offset to the current value of the index register. The index register remains unchanged during this operation.

indexing  An addressing technique which uses a pointer to locate a memory operand. The pointer address can be biased by an offset as an extension to this technique. See indexed addressing mode.

indirection  A referencing technique which uses a pointer address to locate either (a) the address of an operand or (b) another pointer address. Each pointer address in a path to the operand represents a “level of indirection.”

inherent addressing mode  An addressing mode which does not require a memory operand and, therefore, requires no bus access. Only internal register operands are specified by inherent addressing using single byte instructions. TSX, ABA, COMA, and NOP are examples of MC6801 inherent instructions.
**input capture function** A programmable timer function which transfers the current contents of the Counter Register to the Input Capture Register in response to a proper level transition at P20. Which transition is "proper" is controlled by the IEDG bit in the TCSR. The purpose of the input capture function is to facilitate measurement of elapsed time between external level transitions using software.

**input/output** Data entered (input) or data received (output) from a computer.

**instruction** An MC6801 machine instruction consists of one to three consecutive bytes which define a single operation. An instruction consists of an opcode byte and may be followed by one or two bytes which define a memory-resident or immediate operand.

**internal address space** The set of addresses to which MC6801 internal devices respond. Internal addresses are a function of the operating mode and, in some cases, the RAME bit in the RAM Control Register.

**interrupt** A control mechanism which suspends execution of a program, saves certain registers, and transfers control to an interrupt routine. The suspended program can be resumed by restoring the saved registers.

**interrupt-driven** A program control structure utilizing interrupts (as opposed to polling) to initiate service to a device.

**interrupt mask bit** (See I-bit)

**interrupt response time** The elapsed time from assertion of the interrupt until the device is serviced. The response time includes recognition of the interrupt and stacking the registers.

**interrupt vector** An address obtained during the eleventh and twelfth cycle of the MC6801 interrupt sequence which is subsequently transferred to the Program Counter. Hence, instruction execution is resumed at this address. Interrupt vectors are obtained from fixed addresses in either the internal or external memory space depending upon the operating mode.

**I/O** (See Input/Output)

**I/O Port** One of four MC6801 ports which is controlled by a Data Direction Register and provides direct access to the port pins using the Data Register. Ports 1 and 2 are always available as I/O ports. Ports 3 and 4 can be used as I/O ports in some operating modes.

**IOS (Input/Output Select)** The configuration of the SC1 pin in Mode 5. IOS is an active low output signal which is asserted whenever an address from $100 to $1FF (inclusive) appears on the internal address bus. It should be used as an input to a chip select circuit for devices residing in the Mode 5 external address space. Note that internal addresses will also appear on the address bus.

**IRQ1** One of two types of maskable interrupts in the MC6801. IRQ1 is an active low, level-sensitive interrupt.
**IRQ2** One of two types of MC6801 maskable interrupts. IRQ2 is an internal active low, level-sensitive interrupt which is asserted internally by a certain combination of active high flags and interrupt enable bits. This is indicated in the following Boolean equation where IRQ2 is active low:

\[
IRQ2 = [(TOF \cdot ETOI) + (OCF \cdot EOCI) + (ICF \cdot EICI)] + [(ORFE + RDRF) \cdot RIE + (TDRE \cdot TIE)]
\]

**ISBB (Standby Current)** The current consumption at the maximum supply voltage (VSB) for VCC Standby. The current will not exceed this value in the powerdown state (VCC Standby = VSBB).

**IS3 (Input Strobe 3)** The configuration of the SCI pin in the single chip modes (4 and 7). An IS3 negative edge always sets the IS3 FLAG bit as a response and can also be used to (a) latch data into Port 3 if the LATCH ENABLE bit is set and/or (b) assert IRQ1 if the IS3 IRQ1 ENABLE bit is set.

**IS3 FLAG (Input Strobe 3 Flag)** Bit 7 of the Port 3 Control and Status Register. IS3 FLAG is set in response to a negative edge on IS3. It is active only in Modes 4 and 7 and is cleared by a proper access of the Port 3 Data Register.

**IS3 IRQ1 ENABLE (Input Strobe 3 IRQ1 Interrupt Enable)** Bit 6 of the Port 3 Control and Status Register. If set, IRQ1 is asserted whenever IS3 FLAG is also set. It is active only in Modes 4 and 7.

**ITSI (Three-state Input Current)** The maximum allowable current (either as a source or a sink) when the pin is configured as a high impedance input. This parameter applies to Ports 1, 2 and 3.

**IX (See Index Register)**

**jump** To cause execution of an instruction other than the next sequential one. This is implemented by loading the Program Counter with the address of the jump destination. This is very similar to a branch operation. See branch for differences.

**K** When referring to memory sizes, K is an abbreviation for 1024 decimal. The number of bits in a 64K bit RAM, therefore, is 65,536 decimal bits. It is also used here as an abbreviation for “kilo” or 1000 decimal.

**label** In the MC6801 Macroassembler, a label is a symbol which appears in the label field of an assembly language statement. A label consists of one to six alphanumeric characters (upper case letters A-Z, digits 0-9) and the three special characters, period (.), dollar ($) and underscore (_). A label must begin with either a letter or period (.)..

G-11
**LATCH ENABLE** Bit 3 of the Port 3 Control and Status Register. It can be used to cause input data to be latched into the Port 3 Data Register in Modes 4 and 7 in response to an IS3 negative edge.

**level-sensitive** An attribute of an input circuit which results in a response to a level. If the level is not removed or masked, it can cause continuous responses. IRQ1 and RESET are level-sensitive signals in the MC6801.

**LILbug™** The name of the firmware monitor mask programmed into the ROM of the MC6801L1. Commands and a source listing are included in the LILbug Manual.*

**local variable** A variable which can be accessed only by a given subroutine.

**logic analyzer** A device capable of capturing and displaying the bus signals for a given processor.

**loop** A control mechanism which causes a sequence of instructions to be repeated.

**LSB (Least Significant Byte/Bit)** That portion of a multi-byte/bit value which contains the least arithmetic significance. Double byte values used in MC6801 instructions are situated in memory with the MSB having the lower address.

**LSI (Large Scale Integration)** A term used to describe integrated circuits having an arbitrarily large number of gates. LSI is the third category of four types of integrated circuits consisting of SSI (Small scale integration), MSI (Medium Scale integration), LSI, and VLSI (Very Large Scale Integration).

---

**machine code** A sequence of one or more bytes which defines a single operation. MC6801 machine codes consist of one to three bytes. Machine code is typically shown in hexadecimal format.

**machine state** A term used to describe the contents of the MPU registers at a specified time. With respect to the MC6801, the term usually excludes the stack pointer (SP) and consists of: CCR, ACCA, ACCB, X, and PC. These registers are stacked during an interrupt sequence to save the machine state. The stack pointer then points to where the machine state is saved. This procedure allows a program to be interrupted and resumed.

**macro** In software, it defines a sequence of characters which is substituted into the program each time it is invoked by a macroinstruction. The sequence can typically accept argument substitutions. The purpose of a macro is to reduce the amount of programming effort required to generate similar sequences of instructions.

---

*Motorola Semiconductor Products, Inc., Box 20912, Phoenix, Arizona 85036

G-12
**macroinstruction** A source language statement which causes a sequence of instructions defined by a macro to be substituted into a program.

**macroprocessor** A program which is typically a module of an assembler and can be used to expand macroinstructions into sequences of assembly language instructions. An assembler having this capability is usually called a macroassembler.

**mask** A sequence of bits used as an operand in a Boolean operation. Mask is also sometimes used as a synonym for "inhibit." When an IRQ1 interrupt is masked, for instance, it is inhibited. Mask is also used to identify a photographic plate used in the manufacture of integrated circuits.

**mask option** Certain configuration options which may be chosen during specification of the MC6801 masked ROM.

**MCU (Microcomputer Unit)** A microcomputer implemented as a monolithic chip. It typically consists of at least an MPU, ROM, RAM, and some form of I/O.

**MC6801L1** An MC6801 which has the LILbug monitor mask programmed into the ROM. See the LILbug manual for details.

**MC6803** A microcomputer equivalent to an MC6801 operating in Modes 2 or 3.

**MC6803NR** A microcomputer equivalent to an MC6801 operating in Mode 3.

**MC68701** A microcomputer which is nearly identical to the MC6801. The MC68701 includes a 2048-byte EPROM instead of a masked ROM which provides a method for debugging software.

**memory map** A graphical illustration of the available addressable areas including annotations which describe subareas of the space.

**memory space** The set of addresses which can be directly accessed by the address bus. This is sometimes also referred to as the "physical" memory space.

**Microcomputer Unit** (See MCU)

**Microprocessor** (See MPU)

**mnemonic** An abbreviation for an operation which, by its appearance, makes details of the operation easier to remember. LDAA, for example, is the assembler mnemonic for "load accumulator A."

**monitor** A collection of routines which typically provide a uniform method of input/output and certain debugging functions.

**MPU (Microprocessor Unit)** A Central Processing Unit (CPU) implemented with LSI technology.

**MPU Read** Any bus cycle during which the R/W line is high. MC6801 instructions LDAA and TST both perform MPU reads.
**MPU Write** Any bus cycle during which the R/W line is low. MC6801 instructions STAA and COM both perform MPU writes.

**MOS (Metal Oxide Semiconductors)** An integrated circuit technology characterized by its high density and low power consumption. Subcategories of MOS include CMOS, HMOS, NMOS, and PMOS.

**MSB (Most Significant Byte/Bit)** That portion of a multi-byte/bit value which contains the most arithmetic significance. The MSB of values used in MC6801 double byte instructions are situated in memory with the MSB having the lower address.

**multiplex** To use the same set of lines to carry more than one signal. In the MC6801 expanded multiplexed modes, eight Port 3 pins are used for both address (A0-A7) and data (D0-D7).

---

**N-bit** Bit 3 of the Condition Code Register. The N-bit is set according to the most significant bit of the result in certain operations. Operation of the BPL and BMI instructions is controlled by the N-bit.

**negate** To form the two’s complement of a value. The MC6801 NEG instruction performs this operation. Negate is also used to indicate the inactive state of a signal independent of whether the voltage is high or low.

**negative edge** A high-to-low level transition.

**nested** An operation which is repeated while performing a similar operation. For example, a nested interrupt is an interrupt taken while servicing a prior interrupt. A nested loop is a loop within a loop.

**nibble** A quantity consisting of a half-byte, or 4 bits.

**NMI (Non-Maskable Interrupt)** An MC6801 edge-sensitive non-maskable interrupt input.

**NMOS (N-channel Metal Oxide Semiconductor)** An integrated circuit technology noted for its high density.

**NRZ (Non-Return to Zero)** One of two MC6801 serial communications formats. In NRZ format, the level throughout each bit time represents the value of the data.

---

**object code** An output of an assembler or compiler. Typically, it contains information pertaining to where the program is to reside in memory along with some representation of the machine code.

**OCF (Output Compare Flag)** Bit 6 of the Timer Control and Status Register. OCF is set when the contents of the Output Compare Register matches the Counter Register.
offset byte  The second byte of machine code in all instructions which use either the indexed or relative addressing modes. The offset byte is added to the current value of the index register (indexed addressing mode) or program counter (relative addressing mode) to obtain the effective address of the instruction.

OLVL (Output Level)  Bit 0 of the Timer Control and Status Register. OLVL is used with the output compare function to control an output waveform.

"1"  In a binary system, a "1" refers to one of two possible states where "0" is the other state. In positive logic, it refers to a logic high or "true" condition.

operand  That which is operated on. In an assembly language statement, the operand field follows the operation field, and defines both the addressing mode and the effective address. With respect to machine language, it is those byte(s) which follow the opcode byte and specifies an immediate value, offset, or an address.

operating mode  One of eight possible MC6801 configurations which is defined by the levels on P20, P21, and P22 on the positive transition of RESET.

opcode (operation code)  The first byte of machine code in any MC6801 instruction. The opcode defines the type of operation, the register(s) involved, and the addressing mode.

ORFE (Overrun or Framing Error)  Bit 6 of the Transmit/Receive Control and Status Register. ORFE is set by the SCI receiver to indicate either a receiver overrun or framing error. The type of error can be distinguished by noting the RDRF bit: if RDRF and ORFE are set, it indicates an overrun. If ORFE is set and RDRF is not set, a framing error or break condition has occurred.

OS3 (Output Strobe 3)  SC2 is configured as OS3 in Modes 4 and 7. It is an active low output which is asserted by a proper access of the Port 3 Data Register. Which access is "proper" is defined by the OSS bit.

OSS (Output Strobe Select)  Bit 4 of the Port 3 Control and Status Register. OSS is active only in Modes 4 and 7 and is used to define whether an OS3 output signal is to be generated by an MPU write or an MPU read of the Port 3 Data Register.

Output Compare Register  A 16-bit read/write register which is compared with the Counter Register during each E-cycle. If a match is found between the two registers, the OCF bit is set. The Output Compare Register can be used (1) with the OLVL bit to generate an output waveform or (2) without regard to the OLVL bit to generate an interrupt after an elapsed time.

output level register  A D flip-flop in the MC6801 Programmable Timer for which the OLVL bit is an input (D) and the clock (CLK) is an output compare pulse. Providing P21 is configured as an output, the output of the flip-flop (Q) will appear at the pin. Note that the level of P21 reflects the value of the OLVL bit during the last successful output compare.

overrun error  An error which occurs due to a delayed response to the RDRF flag. An overrun error occurs whenever a byte is ready to be transferred from the Receive Shift Register to the Receive Data Register and RDRF is still set.
parasitic capacitance The capacitance inherent in a device by virtue of its fabrication.

PC (Program Counter) A 16-bit counter which always points to the first byte of the next instruction to be executed. The register is incremented once for each instruction byte fetched.

PC0-PC2 (Program Control bits) Bits 5-7 of the Port 2 Data Register. These three bits reflect the current operating mode of the MC6801 where PC0 is the value latched from P20 on the positive edge of RESET. The bits are read-only with one exception: Mode 5 can be irreversibly entered from Mode 4 without going through Reset by writing a “1” to PC0 in Mode 4.

PD (Power Dissipation) The maximum power required for powerup operation of the MC6801 including both VCC and VCC Standby at maximum supply voltage and lowest temperature.

PDB (Peripheral Data Bus) The internal segment of the MC6801 Data Bus.

Peripheral Device A device which can be interfaced with a computer to provide some function.

Pij (Port i, bit j) A notational convention whereby an MC6801 pin is referred to by its port number and bit position. P24, for example, indicates Port 2 bit 4.

PLC (Program Latch Control) Bit 0 of the MC68701 RAM/EPROM Control Register which is used to enable latching the EPROM address during programming.

calling A software technique involving a loop which checks (i.e., polls) the status bits of devices to determine if service is required.

Port 3 Control and Status Register An 8-bit register which is active only in Modes 4 and 7. The bits provide control and status information for Port 3 including operation of the IS3 and OS3 lines.

positive edge A low-to-high level transition.

PPC (Program Power Control) Bit 1 of the MC68701 RAM/EPROM Control Register which is used to enable power from the RESET/Vpp pin to the EPROM during programming.

precharge The act of charging the capacitance of an internal node to a logic high during a particular time interval. The output is then generated by discharge paths of those nodes whose values are to be “0”. This technique is used to improve the speed at high capacitance nodes and decrease power consumption.

priority encoder A device in which the output is based on a hierarchical order of inputs. The output is the identity of the active input with the highest priority. A 74LS147 is an example of a priority encoder.

PRObug™ The name of an MC68701 monitor available in a masked ROM which can be used for system development and EPROM programming. Consult the PRObug manual for details.*

*Motorola Microsystems, 3102 N. 56th St., Phoenix, Arizona 85018
program counter (See PC)

Programmable Timer One of the major modules of the MC6801 consisting of registers which implement the output compare function, input capture function, and the free running counter.

pullup resistor A resistor which connects a line to VCC. Its purpose is to provide a logic high when no other device is driving the line. The value of the resistor is determined by the current capacity of each device connected to the line. Values between 3K and 10K ohms are typically used in MC6801 systems. Wire-OR configurations always require a pullup resistor and the value of the resistor must be determined from the number and type of devices sharing the line.

push-pull driver A circuit which includes two transistors, a pullup resistor (which could have the value zero), and a connection to ground. If the output is a “1”, one of the two transistors conducts and connects the output through a pullup resistor to VCC. If the output is a “0”, the opposite transistor conducts and connects the output to VSS. Most of the MC6801 data port output drivers employ this type of device.

PWASH (Pulse Width Address Strobe High) Applies in Modes 0, 1, 2, 3 and 6. It is the minimum width of the logic high portion of the Address Strobe signal.

PWEH (Pulse Width E High) The minimum width of the logic high portion of E.

PWEL (Pulse Width E Low) The minimum width of the logic low portion of E.

PWRSTL (RESET Low Pulse Width) The minimum number of E-cycles for which the RESET line must be held low during a Reset when operating in a powerup state. During initial powerup (from a powerdown condition), however, RESET must be held low for tRC to provide time for the oscillator to stabilize.

RAM (Random Access Memory) A memory in which each location can be individually read or written in any order.

RAM Control Register An MC6801 register used for controlling and determining the status of the RAM. It includes the RAME and STBY PWR bits.

RAM/EPROM Control Register An MC68701 register used for controlling and determining the status of the RAM and for programming the EPROM. It includes the RAME, STBY PWR, PPC, and PLC bits.

RAME (RAM Enable bit) Bit 6 of the RAM Control Register. RAME defines whether the RAM is included in the internal memory space of the MC6801. When set (and not in Mode 3), the RAM is included in the internal memory space. When clear, the address space of the RAM is considered to be external. The purpose of the bit is to inhibit undesirable accesses to the RAM during transition to powerdown operation.
Rate and Mode Control Register (See RMCR)

RDRF (Receive Data Register Full) Bit 7 of the Transmit/Receive Control and Status Register. When set, RDRF indicates that a byte has been transferred from the receive shift register to the Receive Data Register.

RE (Receiver Enable) Bit 3 of the Transmit/Receive Control and Status Register. Setting RE enables operation of the receiver section of the SCI.

read-modify-write instruction An instruction which fetches data from a memory location or register, modifies it, and writes the modified version back. This type of instruction performs both an MPU read and write to the same location. LSL, COM, and NEG are examples of MC6801 read-modify-write instructions. These instructions have opcodes which range from $40 to $7F, inclusively, excepting JMP (see Appendix B).

read-only A property of a memory element which is to be read but not modified. Writes to the location are allowed but have no affect on the value.

Read/Write (See R/W)

Receive Data Register An 8-bit read-only register which is used by the MPU to obtain data from the SCI receiver after it has been converted to parallel format.

receive shift register An 8-bit shift register which is clocked by the bit rate clock and is used to convert a serial data stream to parallel. The receive shift register is not directly addressable.

recursion A software technique which uses a procedure that invokes itself. For example, the following instructions implement a recursive software delay loop:

```
LDAA #VALUE
JSR TIMEOUT
•
TIMOUT DECA
BEQ OUT
BSR TIMEOUT
OUT RTS
```

re-entrant routine A routine which is written in such a manner as to allow execution of it to be interrupted, executed again by the interrupt service routine, and then be resumed without ill effects. Re-entrant routines modify no local variables with absolute addressing.

relative addressing mode An addressing mode in which the effective address is obtained by adding an 8-bit signed offset byte to the current value of the Program Counter. This addressing mode is used only for branches in the MC6801.

Reset To return a device or system to a defined state.

RESET An MC6801 active low level-sensitive input which causes a Reset sequence to be executed.
**RESET/Vpp** An MC68701 active low level-sensitive input which (1) resets the microcomputer when the pin voltage falls below VIL, (2) is used as a control signal to capture the operating mode of the MCU, and (3) provides an input for an EPROM programming voltage (Vpp) at a maximum current of Ipp.

**Reset vector** A 16-bit address which is loaded into the Program Counter in response to addresses $FFFFE and $FFFFF in the first two E-cycles after RESET goes high. Hence, instruction execution begins at the address obtained from $FFFFE:FFFF.

**RIE (Receiver Interrupt Enable)** Bit 5 of the Transmit/Receive Control and Status Register. When set, IRQ2 is asserted whenever RDRF and/or ORFE are set.

**rise time** The time required for a voltage level change from VIL to VIH (an input) or from VOL to VOH (an output).

**rising edge** A low-to-high level transition.

**RMCR (Rate and Mode Control Register)** A 4-bit write-only register. It is used by the SCI to define (a) the source of the serial bit rate clock, (b) the serial data format, (c) whether the internal bit rate clock is to be provided as an output, and (d) if the internal clock is selected, the serial bit rate.

**ROM (Read-Only Memory)** A memory in which each location can be individually read in a random order but not altered. Note that a RAM can be temporarily converted to a ROM by tying the R/W input line to a logic high through a switch. Typically, a ROM will retain its data in the absence of power.

**RX (Receiver Data)** Serial data intended for the input to a receiver or the receiver, itself.

**R/W (Read/Write)** The configuration of SC2 in all operating modes except Modes 4 and 7. It is used in MC6801 systems to control the direction of data bus transfers. When the level is high, it indicates a memory-to-MPU transfer (MPU read) whereas a low level indicates an MPU-to-memory (MPU write) transfer.

**SCLK (Serial Clock)** A bi-directional line to the SCI (P22) which can be used to input an external clock at eight times the desired bit rate or provide the internal bit rate clock as an output. Both functions are controlled by the RMCR.

**SCI (Serial Communications Interface)** A major module of the MC6801 which provides the capability to transmit and receive serial data in one of two formats, using several bit rates.

**self-modifying instructions** An instruction sequence which modifies itself as it executes. Such a sequence cannot be stored in ROM. Its use is generally considered poor programming practice due to debugging and modification difficulties.

**Serial Communications Interface** (See SCI)
set To change the value of a bit or device to a “1” (logic high).

serial data Data presented one bit-at-a-time in sequence. The LSB is typically transmitted first.

setup time The minimum time that the correct logic level must be present prior to a clock transition in order for it to be correctly recognized.

signed A notational scheme which provides a numbering system that extends in both positive and negative directions from zero. MC6801 signed values uses two’s complement format. A negative value is indicated by a “1” in the most significant position whereas a positive value contains a “0” in this position. Note that a signed single byte can have a value from $-128$ to $+127$ while a signed double byte value has a range from $-32,768$ to $+32,767$.

single chip mode One of two operating modes (4 and 7) of the MC6801. Mode 4 is used for testing while Mode 7 is used in single chip applications. Note that no external bus is provided in this mode.

source code The input to an assembler or compiler consisting of statements in the language being assembled or compiled.

SP (Stack Pointer) A 16-bit register which always points to the next available location in a last-in-first-out queue. The initial value of the stack pointer must be specified by the programmer. The stack pointer is automatically decremented as data is pushed onto the stack and incremented as data is pulled from it.

SC1 (Strobe Control 1) An MC6801 pin which is configured as an output in all modes except modes 4 and 7. In modes 0, 1, 2, 3, and 6, SC1 provides Address Strobe (AS). In mode 5, it provides the IOS signal. In modes 4 and 7, however, it is an input signal configured as IS3.

SC2 (Strobe Control 2) An MC6801 pin which is configured as an output in all modes. In all modes except 4 and 7, it provides the Read/Write (R/W) signal. In modes 4 and 7, it is configured as IS3.

SR flip-flop (Set/Reset flip-flop) A device having two inputs and a single output where only logic high inputs affect the output. A logic high at the “set” input provides a logic high at the output whereas a logic high at the “reset” input provides a logic low at the output. A logic low at both inputs produces no change in the output. However, if a logic high is presented to both inputs simultaneously, the output state is undefined.

SS1:SS0 (Speed Select bits) Bits 1 and 0 of the SCI Rate and Mode Control Register. They are write-only bits which control the SCI bit rate only when using the internal bit rate clock.

stack A linear list in which all additions and deletions are made at one end of the list. This is sometimes referred to as a last-in-first-out (LIFO) discipline. A stack is managed in hardware by the MC6801 16-bit stack pointer (SP).

Stack Pointer (See SP)
standby RAM  That portion of the RAM which is addressed from $80$ through $BF$. During a powerdown state, data in this portion of the RAM can be kept valid by providing power to the VCC Standby pin.

start bit  With respect to asynchronous serial communications, it is the first negative transition following an idle (mark) period. Following a framing error, however, a start bit is recognized without the transition if the following bit is also a zero (space). The start bit is used to synchronize a serial receiver with the data and is, by convention, always a “0”. In SCI serial data transition, it is followed by eight data bits and a single stop bit (“1”).

STBY PWR (Standby Power)  Bit 7 of the RAM Control Register. The STBY PWR bit is used to determine whether the power source to the standby portion of the RAM was sufficient to maintain the data.

stop bit  The last element using serial asynchronous protocol. The stop bit occurs during the tenth bit time and is always a “1”. If the stop bit is not detected, a framing error or break condition exists.

subroutine linkage  A scheme used to (a) transfer control to a subroutine, (b) transmit arguments to the subroutine, (c) transmit the results to the caller, and (d) transfer control back to the caller. With respect to the MC6801, the JSR and BSR instructions push the return address onto the stack and pass control to a subroutine. The programmer must adopt a convention for argument passing in the absence of a specified convention. Control can be returned to the caller by using the RTS instruction to pull the return address from the stack into the Program Counter.

symbolic addressing  A programming technique which utilizes symbols or labels to define addresses as opposed to their actual value.

symbol  With respect to the MC6801 Macroassembler, a symbol is used as a substitute for a specific value. A symbol consists of from one to six alphanumeric characters (upper case letters A-Z, digits 0-9), and the special characters period (.), dollar sign ($), and underscore (_). A symbol must begin with a letter or period (.)

-tttt-

$\text{tACCM (Expanded Multiplexed Read Access Time)}$  During an MPU read, $t\text{ACCM}$ is the maximum time from when the address bus is valid (A0-A15) to when data is required to be valid.

$\text{tACCN (Expanded Non-Multiplexed Read Access Time)}$  During an MPU read, it is the maximum time from when the address bus is valid (A0-A7) to when data is required to be valid.

$\text{tAD (Address Delay Time)}$  The maximum time following the negative edge of $E$ until $R/W$ and either A8-A15 (Expanded Multiplexed) or A0-A7 (Expanded Non-multiplexed) are valid.

$\text{tAH (Address Hold Time)}$  The minimum time that the address bus remains valid after the negative edge of $E$. Because addresses are typically used to generate external system chip selects, these signals would also be affected at this time.
tAHL (Address Hold Time for Latch) The minimum hold time for A0-A7 after the negative edge of AS. This provides a hold time for latching A0-A7 in the expanded multiplexed modes.

tAS (Address, R/W Setup Time before E) The minimum setup time before the positive edge of E for R/W and either A8-A15 (Expanded Multiplexed) or A0-A7 (Expanded Non-multiplexed).

tASD (Address Strobe Delay Time) The time between the negative edge of E and the positive edge of AS. This value represents the amount of time a peripheral device has to vacate the multiplexed data bus before the MCU will begin to drive it with A0-A7. This parameter is affected by the duty cycle of the MCU input clock (4fo or fXTAL). (See Appendix I). The minimum value listed in the Data Sheet for tASD corresponds to a 60% duty cycle.

tASED (Address Strobe to Enable Delay Time) The time between a negative edge of AS and the positive edge of E. This parameter is affected by the duty cycle of the MCU input clock (4fo or fXTAL). (See Appendix I.) The minimum value given in the Data Sheet for tASED corresponds to a 40% duty cycle.

tASF (Address Strobe Fall Time) The time required for AS to change from a logic high to a logic low.

tASL (Address Setup Time for Latch) The minimum setup time for A0-A7 to be valid before the negative edge of AS. This provides a setup time for latching A0-A7.

tASM (A0-A7 Setup Time before E) With respect to the Multiplexed Bus, it is the minimum setup time before the positive edge of E that address lines A0-A7 are valid. Because A8-A15 are always valid before this time, this represents the minimum setup time for the entire bus.

tASR (Address Strobe Rise Time) The time required for AS to change from a logic low to a logic high.

tCMOS (Delay, E to Data Valid) Applies to Ports 1, 2, 3, and 4 Data Registers (except for P21) only when configured as output data ports. During MPU writes, tCMOS is the time from the negative edge of E to when the level reaches 70% of the CMOS supply voltage. It should also be noted that (a) Port 4 outputs cannot be pulled above the MC6801 supply voltage, and (b) a 10k ohm pullup resistor is required for Port 2 when interfacing with CMOS.

TCSR (Timer Control and Status Register) An 8-bit register used to control and determine the status of the MC6801 Programmable Timer.

tCyc (Bus Cycle Time) The period of an MCU bus cycle. It is equivalent to 4/fXTAL, 4/4fo, 1/fo or 1/E.

tDDW (Data Delay Write Time) The maximum time after the positive edge of E until data is valid during an MPU write cycle.

TDRE (Transmitter Data Register Empty) Bit 5 of the Transmit/Receive Control and Status Register. When set, it indicates that the SCI transmitter is ready for more data. TDRE is cleared by reading TDRE and writing a byte to the Transmit Data Register.
The minimum MCU data setup time for an MPU read that must be provided by a peripheral device before the negative edge of E.

Bit 1 of the Transmit/Receive Control and Status Register. When set, it initializes the SCI transmitter and enables operation of the transmitter.

The time required for E to change from a logic high to a logic low.

The time required for E to change from a logic low to a logic high.

During an MPU read, it is the minimum time following the negative edge of E during which a peripheral is expected to hold the data valid.

For an MPU write, it is the minimum time following the negative edge of E during which the MCU will keep the data valid.

Bit 3 of the Transmit Receive Control and Status Register. When set, it asserts IRQ2 whenever the TDRE bit is set; otherwise, the interrupt is inhibited.

Applies only in Modes 4 and 7 when port 3 is configured as a latched input data port (LATCH ENABLE bit is set). It is the minimum time prior to the negative edge of IS3 that data must be held valid to insure latching the correct logic levels into the Port 3 data register.

Timer overflow occurs when the 16-bit Counter Register contains $FFFF. The TOF bit is set to reflect this condition.

An input to P20 which is connected to the MCU input capture edge detector.

Applies only in Modes 4 and 7 when Port 3 is configured as a latched input data port (LATCH ENABLE bit is set). It is the minimum time after the negative edge of IS3 that data must remain valid to insure latching the correct logic levels into the Port 3 data register.

The minimum time after the positive edge of RESET that the mode programming levels must remain valid to insure correct recognition.

The minimum time before the positive edge of RESET that the mode programming levels must remain valid to insure correct recognition.

Bit 5 of the Timer Control and Status Register. TOF is set when the 16-bit Counter Register contains $FFFF.

To change the value of a bit or byte to its opposite state.

An output from the output level register to P21. The value of the OLVL bit is clocked to this unaddressable register during a successful output compare. It is provided as an MCU output if P21 is defined as an output by bit 1 of the Port 2 Data Direction Register.
**tPCS (Processor Control Setup Time)** The minimum time prior to the negative edge of E that interrupts and RESET must be valid to be recognized during that E-cycle.

**Transmit Data Register** An 8-bit write-only register used to provide data to the SCI transmitter.

**tRC (Oscillator Stabilization Time)** The minimum time which RESET must be held low from a powerdown state (initial startup). This provides time for the MC6801 internal oscillator to stabilize. After a powerup condition has been established, RESET must be held low for a minimum of PWRSTL cycles.

**tOSD1 (First Edge Delay Time, E to OS3)** A timing parameter associated with OS3 (Modes 4 and 7). It is the delay time between the first positive edge of E following a proper access of the Port 3 Data Register and the negative edge of OS3. Which access (read or write) is proper is defined by the OSS bit.

**tOSD2 (Second Edge Delay Time, E to OS3)** A timing parameter associated with OS3 (Modes 4 and 7). It is the delay time between the second positive edge of E following a proper access of the Port 3 Data Register and the positive edge of OS3. Which access (read or write) is proper is defined by the OSS bit.

**tPDH (Peripheral Data Hold Time)** Applies to Ports 1, 2, 3, and 4 only when configured as input data ports. With respect to Ports 1, 2, and 4, it is the required hold time after the positive edge of E during the MPU read of the port Data Register. With respect to Port 3, it applies only to unlatched operation in Modes 4 or 7 (LATCH ENABLE bit = 0) and is the required hold time after the negative edge of E with respect to the end of an MPU read cycle of the Port 3 Data Register.

**tPDSU (Peripheral Data Setup Time)** Applies to Ports 1, 2, 3, and 4 only when configured as input data ports. With respect to Ports 1, 2, and 4, it is the required setup time before the positive edge of E during an MPU read of the port Data Register. With respect to Port 3, it applies only to unlatched operation in Modes 4 or 7 (the LATCH ENABLE bit is clear) and is the required setup time prior to the negative edge of E at the end of an MPU read of the Port 3 Data Register.

**tPP (Programming Time)** The minimum interval during which the address, data, and programming power (Vpp) must remain constant when programming the MC68701 EPROM.

**tPWD (Delay Time, E to Data Valid)** Applies to Ports 1, 2, 3, and 4 when configured as output data ports. tPWD is specified as the delay from the negative edge of E at the end of the port MPU write cycle to when the data is valid at the output pins of the port.

**tPWIS (IS3 Pulse Width)** Applies only in Modes 4 and 7. It refers to the minimum width of an active low IS3 input pulse to insure recognition.

**transmit shift register** An 8-bit shift register which converts parallel data from the Transmit Data Register into serial format. This register is not addressable.

**transparent latch** A device in which the outputs are the same as the inputs (i.e., transparent) until the latching feature is activated. When this occurs, the outputs will remain fixed to the value of the inputs. The MC6882 and 74LS373 are representative of this type of latch.
**TRCSR (Transmit/Receive Control and Status Register)** An 8-bit register used to control and determine the status of the MC6801 Serial Communications Interface.

**two's complement** A scheme for which negative values are formed from their corresponding positive values by complementing every bit and then adding one to the result. MC6801 instructions use this form of representing numbers.

**TX (Transmit)** A transmitter or serial data which is the output of a transmitter.

**-uuuu-**

**unsigned** A numbering scheme which contains no negative value and extends from 0 to a positive number limited by the number of bits used to represent the number. Note a single byte can represent any unsigned integer from 0 to 255 while a double byte can range from 0 to 65,535.

**upward compatible** Compatibility which implements all features of its predecessor. The MC6801 is upward compatible with the MC6800 with respect to both source and object code. However, the MC6801 executes some instructions with fewer cycles which may require some MC6800 execution time-dependent sequences (e.g., software timing loops) to be modified.

**-vvvv-**

**VCC (MCU Supply Voltage)** One of two MC6801 power supply pins. The VCC pin provides all MC6801 power except to the standby portion of the RAM and RAM Control Register. In the MC680701 VCC does not furnish power to the EPROM control circuits or to the EPROM itself (this voltage is furnished by the RESET/VPP pin). VCC is a traditional term for the supply voltage which has been borrowed from TTL terminology. With respect to MC6801 documentation, it is synonymous with the MOS term, VDD.

**VCC Standby (Standby Power Supply Voltage)** Power to the MC6801 standby portion of the RAM and RAM Control Register is connected to the VCC Standby pin. (See VCC.)

**VDD (Supply Voltage)** An MOS term which specifies the supply voltage. In MC6801 documentation it can be considered as synonymous with the TTL term VCC.

**V- Bit (Overflow bit)** Bit 1 of the Condition Code Register. When set, it indicates that a two's complement overflow has occurred during a preceding operation.

**vector** An address stored in memory which is transferred to the Program Counter in response to either an interrupt or Reset. The vector is usually named after the condition which causes it to be loaded (e.g., a Reset vector).

**V1H (Input Voltage for a Logic High)** The minimum input voltage which will be recognized as a logic high. The maximum voltage should not exceed VCC to prevent potential damage to the MCU.
**VIL (Input Voltage for a Logic Low)** The maximum input voltage which will be recognized as a logic low. The minimum voltage should not be less than VSS to prevent potential damage to the MCU.

**VLSI (Very Large Scale Integration)** An integrated circuit containing a very large number of gates as opposed to a circuit using smaller scale technology (LSI, MSI, or SSI).

**VMPDD (Mode Programming Diode Differential)** Applies when programming the mode with diodes. VMPDD is the minimum voltage difference between a Port 2 logic low for mode programming (VMPL) and the reset voltage level at the point when the mode logic level is latched. The diode forward voltage cannot exceed VMPDD minimum.

**VMPL (Mode Programming Input Voltage Low)** The maximum input voltage to P20, P21, and/or P22 which will be recognized as a logic low when programming the MC6801 operating mode. Note that this value is not identical to VIL.

**VMPH (Mode Programming Input Voltage High)** The minimum input voltage to P20, P21, and P22 which will be recognized as a logic high when programming the MC6801 operating mode. Note that this value is not identical to VIH.

**VOH (Output Voltage for a Logic High)** The minimum output voltage provided for a logic high. This value is specified for a corresponding maximum output load current as shown in the Data Sheet.

**VOL (Output Voltage for a Logic Low)** The maximum output voltage provided for a logic low. This value is specified for a corresponding maximum input current as shown in the Data Sheet.

**VPP (Programming Power)** The power applied to the MC68701 RESET/VPP pin which is used in Mode 0 to provide power to the EPROM for programming.

**VSS** The return path for the power supply. The Ground reference.

**VSB (Powerup Standby Voltage)** The range of voltage required to insure reliable operation of the standby portion of the RAM in a powerup state.

**VSBP (Powerdown Standby Voltage)** The range of voltage required to insure adequate power to the standby portion of the RAM in a powerdown state.

-WWWW-

**Wake-up feature** A feature of the SCI which allows the receiver to ignore the remaining characters in the current message.

**“who-done-it” routine** Upon interrupt, it is a routine which polls the status flags of candidates in order to determine the interrupt source. Note this is a consequence of using a single vector for multiple interrupt sources.
wire-OR A configuration in which two or more open collector (TTL) or open drain (MOS) devices are connected to a common line which is tied to a logic high using an external pullup resistor. When an open collector device is in an inactive state, it has no effect on the common line. When the device is active, however, it sinks current causing a logic low on the line. Note that the line is pulled low unless all devices are inactive.

WU (Wake-up bit) Bit 0 of the Transmit/Receive Control and Status Register. WU can be set to inhibit the SCI receiver during data reception until a sequence of ten consecutive 1’s is sensed by the receiver. WU is cleared by the SCI when this occurs.

\[\text{xxxx}\]

X Register (See Index Register)

XTAL1 One of a pair of MC6801 pins which interface with either a quartz crystal resonator or a TTL-compatible clock. If a crystal is used, it is connected between XTAL1 and EXTAL2. For an external clock, XTAL1 is connected to ground.

\[\text{zzzz}\]

Z-Bit (Zero bit) Bit 2 of the Condition Code Register which is set when the result of certain operations is zero. A branch based on the value of this bit is obtained with the BEQ or BNE instructions.

"0" In a binary system, a "0" refers to one of two possible states where "1" is the other state. In positive logic, it refers to a logic low or "false" condition.
## APPENDIX H

### SUMMARY OF INSTRUCTION E-CYCLE COUNTS

<table>
<thead>
<tr>
<th>ADDRESSING MODE</th>
<th>Immediate</th>
<th>Direct</th>
<th>Extended</th>
<th>Indexed</th>
<th>Inherent</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABA</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>2</td>
<td>★</td>
</tr>
<tr>
<td>ABX</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>2</td>
<td>★</td>
</tr>
<tr>
<td>ADC</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>ADD</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>ADDD</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>AND</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>ASL</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>★</td>
</tr>
<tr>
<td>ASLD</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>ASR</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>BCC</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>BCS</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>BEQ</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>BGE</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>BGT</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>BHI</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>BHS</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>BIT</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>BLE</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>BLO</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>BLS</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>BLT</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>BMI</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>BNE</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>BPL</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>BRA</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>BRN</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>BSR</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>6</td>
<td>★</td>
</tr>
<tr>
<td>BVC</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>BVS</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>CBA</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>CLC</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>2</td>
<td>★</td>
</tr>
<tr>
<td>CLI</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>2</td>
<td>★</td>
</tr>
<tr>
<td>CLR</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>2</td>
<td>★</td>
</tr>
<tr>
<td>CLV</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>2</td>
<td>★</td>
</tr>
<tr>
<td>CMP</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>COM</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>CPX</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>DAA</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>DEC</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>DES</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>DEX</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>EOR</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>INC</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>6</td>
<td>★</td>
</tr>
<tr>
<td>INS</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>INX</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>JMP</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>JSR</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>LDA</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>LDD</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>LDS</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>LDX</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>LSL</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>LSLD</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>LSR</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>LSRD</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>10</td>
<td>★</td>
</tr>
<tr>
<td>NEG</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>NOP</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>ORA</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>ORA</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>PSH</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>4</td>
<td>★</td>
</tr>
<tr>
<td>PSHX</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>4</td>
<td>★</td>
</tr>
<tr>
<td>PUL</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>5</td>
<td>★</td>
</tr>
<tr>
<td>PULX</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>5</td>
<td>★</td>
</tr>
<tr>
<td>ROL</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>ROR</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>RTI</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>10</td>
<td>★</td>
</tr>
<tr>
<td>RTS</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>5</td>
<td>★</td>
</tr>
<tr>
<td>SBA</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>2</td>
<td>★</td>
</tr>
<tr>
<td>SBC</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>SEC</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>2</td>
<td>★</td>
</tr>
<tr>
<td>SEI</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>2</td>
<td>★</td>
</tr>
<tr>
<td>SEV</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>2</td>
<td>★</td>
</tr>
<tr>
<td>STA</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>STD</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>★</td>
<td>★</td>
<td></td>
</tr>
<tr>
<td>STS</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>5</td>
<td>★</td>
</tr>
<tr>
<td>STX</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>5</td>
<td>★</td>
</tr>
<tr>
<td>SUB</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>SUBD</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>SWI</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>12</td>
<td>★</td>
</tr>
<tr>
<td>TAB</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>2</td>
<td>★</td>
</tr>
<tr>
<td>TAP</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>2</td>
<td>★</td>
</tr>
<tr>
<td>TBA</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>2</td>
<td>★</td>
</tr>
<tr>
<td>TPA</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>2</td>
<td>★</td>
</tr>
<tr>
<td>TST</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>TXS</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>TXS</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>3</td>
<td>★</td>
</tr>
<tr>
<td>WAI</td>
<td></td>
<td>★</td>
<td></td>
<td>★</td>
<td>9</td>
<td>★</td>
</tr>
</tbody>
</table>

H-1/H-2
APPENDIX I
EXPANDED MULTIPLEXED BUS CLOCKING

The principal clocking source for the MC6801 is the MCU input clock. Either a TTL compatible clock or a crystal can be used to control the MCU internal clock generator, which includes a divide-by-four circuit in the output. In the expanded multiplexed modes, two clocks, which are derived from the MCU input clock, provide synchronization for all external bus activity. This appendix describes the relationship of these two clocks — E (Enable) and AS (Address Strobe) — to the clock from which both are derived — the MCU input clock.

E is generated from the MCU input clock by toggling the level on alternating negative clock edges. AS is generated by toggling the level on every positive clock edge provided that E is low.

Two properties of the MCU input clock affect E and AS: the period and duty cycle. The affect of these two parameters on E and AS is illustrated in Figure 1-1. The input clock period determines the period of E and the widths of the active and inactive portions of AS. Note that (1) E always has a 50% duty cycle while AS does not and (2) a slight skew (delay) exists between the MCU input clock and both E and AS.

While the duty cycle does not affect the widths of E or AS, it does determine the position of AS with respect to the negative half-cycle of E as indicated by the values specified for tASD and tASED. The effect of increasing tASD is to increase the amount of time a peripheral device has to vacate the multiplexed data bus before the MCU will begin to drive it with address (A0-A7). However, this increase is accompanied by a corresponding decrease in the address setup time. A duty cycle of 50%, therefore, is considered ideal although a tolerance of 10% is acceptable with respect to MCU operation.

The effects of duty cycle on E and AS are shown in Figure 1-1. While the duty cycle for the MCU input clock is specified not to exceed 50% (±10%), the duty cycle shown in the figure admittedly exceeds this value for illustrative purposes. Note that if the duty cycle is 50%, tASD and tASED are equal. However, if the input clock high time is increased, tASD decreases while tASED increases. The opposite is true if the input clock low time increases.

In Figure 1-1, it should be noted that a TTL compatible clock is shown as the MCU input clock. The output of a quartz crystal resonator, however, is not a square wave. The waveform produced by the clock generator for this sinusoidal input depends upon the particular threshold voltage of the clock generator level detector. Note that a 50% duty cycle will be produced only if the threshold is at the mid-point of the voltage range of the sinusoidal input. If it is at a lower voltage, a longer high time will result; if it is above the mid-voltage point, a longer low time will result.
The exact threshold voltage is affected by the manufacturing process and typically the resultant duty cycle is not precisely 50%. However, for a sinusoidal input, the output waveform of the clock generator will conform to the duty cycle specified in the Data Sheet. Because one cannot precisely predict the duty cycle produced by a crystal input, the values specified for \( t_{ASD} \) and \( t_{ASED} \) both represent minimum values. In practice, however, the two minimum values are mutually exclusive: achieving minimum values for \( t_{ASD} \) and \( t_{ASED} \) simultaneously is not physically possible. The values specified, therefore, represent worst case situations.
Figure I-1. E and AS versus Input Clock Duty Cycle
APPENDIX J
RESET VECTOR CHIP SELECT CIRCUIT FOR MODE 0

Mode 0 is an expanded multiplexed mode with two characteristics which are not present in other expanded multiplexed modes. These characteristics are incorporated to facilitate testing and include:

1. the external data bus is also driven during MPU reads of internal locations, and
2. the Reset vector is treated as external within two E-cycles after RESET goes high but subsequent MPU reads of the vector area are read from internal locations $FFFE:FFFF$.

A consequence of the first characteristic is that no memory map overlap can be allowed to occur, when using Mode 0, in order to avoid bus contention during MPU reads of internal locations. For example, if it is desired to include a monitor in a Mode 0 test system, the monitor cannot be located at any address which is defined as a Mode 0 internal address.

A consequence of the second characteristic is that the chip select circuit for the device which provides the Reset vector must be active when within two E-cycles after the last positive edge of RESET. This characteristic makes it possible to obtain control of the MC6801 with an external Reset vector yet retain the capability of reading the entire internal ROM including the vector area.

The circuit shown in Figure J-1 can be used to implement a Reset vector chip select circuit for Mode 0. Note that the asynchronous reset signal is first synchronized with E using a D-type flip-flop.

After \texttt{RESET} has remained low for two E-cycles, the chip select signal (SEL) for the Reset vector will be active for as long as \texttt{RESET} remains low. When \texttt{RESET} returns high, the chip select signal will remain asserted until the \texttt{RESET} signal is clocked to the last D-type flip-flop which requires two E-cycles. It is not necessary to decode the address as part of the vector select circuit because only two addresses are possible after \texttt{RESET} goes high: $FFFE$ followed by $FFFF$.

Note that the chip select circuit will not respond again until two E-cycles after \texttt{RESET} is asserted. If the Reset vector is read using an MPU instruction (such as \texttt{LDX} $FFFF$), the chip select will remain negated and only the contents of the internal location will be driven onto the data bus.

*It should be noted that this Reset vector chip select circuit is not applicable to the MC68701. The Mode 0 chip select circuit for the MC68701 should be designed to trap $BFFE:BFFF$ on the address bus.*
Figure J-1. Schematic for Mode 0 Reset Vector Chip Select Circuit
APPENDIX K
MC6801 SYSTEM DEVELOPMENT TOOLS
MEX6801EVM
MC6801 Microcomputer Evaluation Module

- Single Module — EXORciser-Bus Compatible
- Incorporates an MC6801 DEbug Monitor
- Provides RS-232C Interface
- Sockets Can Be Added to Incorporate Optional
  ACIA, PTM, 2K EPROM, and 4K Bytes
  of Static RAM
- Address Map Established by Programmable Gate
  Array to Permit Reconfiguration
- Additional Decoding Provided for 8K of
  Off-Board Memory
- Large Wirewrap Area
- Two Modes of Operation: Single-Chip or Expanded
- Low Cost

The MEX6801EVM Microcomputer Evaluation Module is a completely self-contained microcomputer on a single printed circuit card, providing the user with the means of evaluating the MC6801 microcomputer.

The system allows the user to easily evaluate the MC6801 microcomputer. As configured, the MC6801 may be evaluated in the Single-Chip mode by attaching an RS-232C-compatible terminal to the serial port of the module. Thus, the minimum functioning system consists of only the MC6801 and an MC 1488 and MC1489.

In the Expanded mode, the customer may add an ACIA, PTM, 4K bytes RAM or 2K EPROM and a programmable gate array for address configuration.

The Evaluation Module provides the user with the ability to evaluate the MC6801 microcomputer using the DEbug monitor via the serial I/O port and RS-232C interface. Refer to Table A for a description of the DEbug commands. A 4.9 MHz crystal is used to generate the standard baud rates of 300, 1200, and 9600 baud. Sufficient space remains within the on-chip RAM for the user to write a small I/O program to work in conjunction with the DEbug monitor. The DEbug monitor program (LILbug) uses a patch table established within RAM for all I/O. Thus, the DEbug program's I/O routines can be readily modified by the user for the purpose of evaluation. Since the DEbug monitor uses the timer output and the serial I/O port, these resources are not available to the user in the Single-Chip mode. However, by using the Expanded mode, the user has the choice of adding an ACIA or PTM, thereby freeing more of the on-chip resources. The Evaluation Module has provisions for adding 4K bytes of static RAM and a 2K byte EPROM. This permits the user to develop his programs if desired. In addition, the Expanded mode allocates 8K bytes of off-board program space for further programming flexibility.

A wirewrap area is provided to permit the user to interface other peripheral devices or special interface circuits to the MC6801.
<table>
<thead>
<tr>
<th>Command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>O &lt;ADDR1&gt; &lt;ADDR2&gt;</td>
<td>Calculate the relative offset of a branch instruction from ADDR1 to ADDR2.</td>
</tr>
<tr>
<td>B</td>
<td>Display all breakpoints.</td>
</tr>
<tr>
<td>B - &lt;ADDR&gt;</td>
<td>Enter a breakpoint at the specified address.</td>
</tr>
<tr>
<td>B - &lt;ADDR&gt;</td>
<td>Remove a breakpoint at the specified address.</td>
</tr>
<tr>
<td>G &lt;ADDR&gt;</td>
<td>Start program execution at the specified address.</td>
</tr>
<tr>
<td>G</td>
<td>Start program execution at the current program counter setting.</td>
</tr>
<tr>
<td>R</td>
<td>Display/change the contents of the program registers and counter.</td>
</tr>
<tr>
<td>&lt;DATA&gt;</td>
<td>Replace current register value with new DATA.</td>
</tr>
<tr>
<td>SP</td>
<td>Display current register value (unless changed by the &lt;DATA&gt; command).</td>
</tr>
<tr>
<td>T &lt;HEX NUMBER&gt;</td>
<td>Trace the number of instructions specified in hexadecimal.</td>
</tr>
<tr>
<td>C &lt;ADDR&gt;</td>
<td>Execute a user routine as a subroutine starting at the specified address.</td>
</tr>
<tr>
<td>C</td>
<td>Same as the C &lt;ADDR&gt; command except that the execution begins from the current address in the program counter.</td>
</tr>
<tr>
<td>LO</td>
<td>Set low speed — 30/15 characters per second for on-chip clock.</td>
</tr>
<tr>
<td>HI</td>
<td>Set high speed — 120/60 characters per second for on-chip clock.</td>
</tr>
</tbody>
</table>

Register contents are displayed using the following format:

P-XXXX X-XXXX A-XX B-XX C-XX S-XX

where:
P = Program Counter
X = Index Register
A = Accumulator A
B = Accumulator B
C = Condition Code Register
S = Stack Pointer
XXXX = Current 16-bit Value
XX = Current 8-bit Value

The R command terminates after examining/changing the stack pointer or whenever any character other than <DATA> or SP is entered (SP = SPace).
EXORciser II Development Systems

- Versatile and easily expandable development tool, used to develop, evaluate, and debug the user's system hardware and software
- Reduces system development time and cost
- Emulates final system architecture and performance through modular building block concept
- Permits debugging of final system design through built-in diagnostic firmware
- Facilitates program development using separately available Resident Software
- Dual Memory Map mode of operation
- Selectable clock speeds of 1.0, 1.5, and 2.0 MHz
- 8 selectable baud rates from 110 to 9600 baud
- A single RS-232C compatible serial communications interface
- A chassis containing a 14-card motherboard and the necessary +5 Vdc and ±12 Vdc power supplies

The EXORciser II Development System is the basic tool for designing and developing microprocessor-based systems using any of Motorola's families of microprocessors including the M6800 family or other microprogrammable parts. It is an extremely powerful and easy-to-use development system that has been designed to be highly user-oriented in order to reduce system development time and cost. The EXORciser is a modularized, expandable instrument that permits "instant breadboarding" and evaluation of any M6800 or M6809-based microcomputer system. It consists of a prewired, bus-oriented chassis and power supply, together with two basic modules - an MPU Module and a Debug Module. These provide the basic control and interface functions of a microcomputer, and house the system development and diagnostic programs. A number of separately available, optional memory modules and additional interface modules (up to twelve) may be added, simply by plugging them into existing prewired sockets, to convert the basic system into an exact prototype of a desired end system. Thus, the EXORciser, with its built-in EXbug Firmware, enables the designer to configure, evaluate and debug his final system hardware and software. The EXORciser II incorporates several advanced features, including Dual Memory Map mode of operation and the ability to develop high performance systems using the M6800 and M6809 parts (1.0 to 2.0 MHz), it also adapts for use with the single chip M6879, M6801, M6805 and M146805 devices.

DESCRIPTION AND OPERATION

The basic EXORciser II contains the common ingredients of a microcomputer, and offers the system designer a low-cost, versatile means of achieving unique final-system performance through the selective addition of separately available, optional modules. These separate assemblies plug directly into the EXORciser's bus so that system expansion becomes quick, easy, and essentially error-proof. With provisions for up to 12 add-on assemblies, a system of almost any complexity can rapidly be assembled.

Supplied with the basic EXORciser II are the MPU II Module and DEbug II Module. The DEbug II Module supplies eight selectable baud rates, and serves as a communication link between the terminal and the EXbug 2 Firmware on the DEbug II Module.

The MPU II Module provides the 1.0, 1.5, or 2.0 MHz clock timing for the microprocessor system under development, as well as for the rest of the EXORciser II. In addition, this module houses the 6850 or 6809 Microprocessing Unit, which imparts to the EXORciser its computation and control capabilities. Also included with the 6800 EXORciser are a Timer, MC6840, and Priority Interrupt Controller, MC6828.

The DEbug II module is a system development tool which provides the user with instant capability to communicate with his system, load programs, monitor the execution of his program in real time, and to isolate and analyze hardware and software problems. The DEbug II module places no restrictions upon the user's system design since all 64K bytes of memory space are available to the user.

These functional subsystems of the basic EXORciser are supplemented by a power supply and a bus-oriented distribution system. This bus system transfers the power supply voltage (as well as the data, address, and control signals) to the optional modules. Overall, the EXORciser can address up to 64K bytes of memory, and addresses the input/output modules (as well as the memory modules) as memory.

ISSUE B

BASIC-M, EXORciser®, EXbug, EXORdisa, EXORterm and Micromodule are trademarks of Motorola Inc.

M6800EXOR
M6800EXORU
M6809EXOR

MOTOROLA INC., 1980

K-4
In order to provide the user additional flexibility, the appropriate EXORciser II modules have a 20-pin connector available for implementation of such system capabilities as priority interrupts, multi-paged memory and I/O systems, parity error detection, and power down/restart features.

Furthermore, the modules will have a standard jumper-arrangement for assigning memory and peripherals to either map in the dual map mode or to any page-extended memory systems.

**MICROCOMPUTER DESIGN WITH EXORciser II**

A design normally begins by defining the functions to be performed by the proposed system. This is followed by design of both hardware and software and trade-off decisions between them.

Using the appropriate memory and input/output modules, such as Micromodules, the designer now emulates his proposed system in the EXORciser. Recognizing that some modules have a special interface circuitry and customized circuitry, provisions have been made on input/output modules for the designer to insert wirewrap sockets and construct the special interface circuitry. Also, the designer can construct any customized circuitry on the Wirewrap Module.

The resident software provides the designer with a powerful software development tool. Using the Resident CRT Editor, which is supplied with the floppy disk system, the designer enters a source program via the terminal keyboard. The user now can modify and change his source program as required to meet his proposed systems requirements. This includes:

- Printing out all or any part of the program for detailed examination
- Changing any characters or string of characters in the source program
- Deleting or adding instruction lines or characters anywhere in the program

At the end of the editing process, the Resident Editor will provide a source program that may be stored on diskette. This source program may be used in subsequent assembly or compilation operations on any of the compatibile Motorola assemblers or compilers. The Resident Macro Assembler which is supplied with the floppy disk system can be used to translate the source program to produce:

- A printed assembly listing of the source program
- An object program on diskette
- A machine file, consisting of the machine-coded program stored directly into the EXORciser II memory.

During the assembly process, the Macro Assembler allows the assignment of relocatable memory addresses which are assigned by the Linking Loader at load time, rather than fixed during the assembly operation.

Once the designer has configured the EXORciser II to emulate his hardware using Micromodules and memory or other I/O modules and has developed his software, he is ready to debug his system. The EXORciser II, with its EXBug II system development firmware, permits the user to debug both his system hardware and his system software, as required, until he has his system operating correctly.

The EXORciser with USE (User System Evaluator) option can test and evaluate equipment external to its chassis. By removing the microprocessing unit from the user's system and connecting the USE cable from the EXORciser into the MPU socket, the EXORciser with its EXBug Firmware can debug and troubleshoot microprocessor systems.

**Use not only extends all of the development system EXBug functions into the user's system, but also all of the optional Systems Analyzer functions. The USE-equipped development system provides the designer with the capability of configuring an emulation of his M6800-family microprocessor system in the EXORciser, external to the EXORciser, or a combination of EXORciser-mounted modules working with the user's external system.**

The designer can develop his system software and firmware on the development system and can use the USE-equipped development system to debug his system hardware and software. The USE EXORciser can also be used as a production tool in testing and evaluating the user's production systems.

**USE** consists of two assemblies: The USE Processor Module, and the USE Cable and Buffer Assembly which connects it into the user's system. A 40-pin connector on this assembly plugs into the MPU socket in the user's system permitting the USE Processor Module's MPU to control the operation of the user's system. In addition, the USE Cable and Buffer Assembly buffers the transferred signals. In this application, everything within the EXORciser appears to be within the MPU in the user's system.

**MODULES**

(Included with the EXORciser II)

**MPU II Module**

- M6800 or M6800-Based
- 1.0, 1.5, 2.0 MHz Clock Speeds
- Triple Programmable Timer (on M6800 Module only)
- Priority Interrupt Controller (on M6800 Module only)
- Refresh Control
- Go/Halt Control
- User-Controlled Three-State Logic
- Internal or External Clock Option
- Interrupt Control
- Generation of a1, a2 and Memory Clock

The MPU Module includes both the system clock and the Microprocessor Unit (MPU). The MPU Module also automatically initiates an EXORciser or user RESTART when power is first applied to the EXORciser.

The clock circuit generates 1.0, 1.5 or 2.0 MHz clock signals. The system may be operated with an external clock over the range 800 kHz to 2.0 MHz.

In addition to generating the basic EXORbus timing signals, the clock circuit provides the EXORciser with the capability of refreshing dynamic memories and working with slow memories. The dynamic memories are refreshed on a cycle-stealing basis. In working with slow memories, the MPU Module stretches the clock pulse to give the memory sufficient time to complete its assigned operation.

**Debug Module**

- EXBug II System Monitor Firmware (3K bytes)
- System Console Interface
- Dual Map Address Control
- STOP-ON-ADDRESS/SYNC ENABLE
- Power Up/Restart Control
- Load, Verify, Search Tape
- Display, Change Memory and MPU Registers
- Trace Instruction(s)
- Set Up to 8 Software Breakpoints
- Search Memory
- Line Printer Echo Option
- Parity Detect
The DEbug II Module, through its EXbug 2 firmware and associated hardware, provides the EXORciser II with its powerful hardware/software debug capability.

Using the EXbug routines provides the designer virtually unlimited freedom in examining and debugging his proposed system hardware and software. He can, for example, search the input medium for a file, load a file into EXORciser memory, verify the contents in the EXORciser memory, and record the memory contents on the selected medium. In between these input/output functions, the user can examine and, if required, change the memory contents. He can insert and remove one hardware breakpoint and up to eight software breakpoints. He also can run in real time or trace through the user's program or a selected portion of the user's program. While using these routines, the user modifies his hardware and software, as required, until he has his system operating to specifications.

The STOP-ON-ADDRESS/SYNC ENABLE switch on the DEbug Module is used to generate a sync pulse at a preselected address or to enable the hardware breakpoint function.

The DEbug II Module provides the EXORciser II with the ability to address two separate 64K memory maps (Dual Map mode). To accomplish this, the DEbug II Module takes the Valid Memory Address (VMA) signal from the MPU II Module and converts it to two other signals: Valid User Address (VUA) and Valid Executive Address (VXA). All EXORciser II hardware modules may be configured to respond to one of these enabling signals. As a result, two complete maps of 64K bytes are addressable for either random access data storage or for data I/O.

A serial I/O port allows interfacing with any RS-232C compatible terminal. Baud rates are selectable from 110 baud to 9600 baud. The module also interfaces to the EXORciser's front panel RESTART and ABORT switches. The RESTART and ABORT push-button switches allow manual termination of program execution.

### OPTIONAL RANDOM ACCESS MEMORY

**Dynamic RAM**
- 1.0, 1.5, 2.0 MHz Clock Speeds
- Individual Address and Enable for Each 16K Block
- System Cycle Stealing Refresh
- 16, 32, 48, 56/64K Single Board Versions
- Dual Map and Page Control
- Standard Parity

**Static RAM**
- Clock Speed Independent to 2.0 MHz
- RAM-ROM Mode Selection
- 8, 16K Single Board Versions
- Up to 64K-Byte per EXORciser
- Individual Address and Enable for Each 8K Block
- Dual Map and Page Control
- Standard Parity

**Hidden Refresh RAM**
- 1.0 MHz Clock Speed
- Individual Address and Enable for each 16K Block
- Memory Refresh Without Processor Interruption
- 16, 32, 48, 56/64K Single Board Version
- Dual Map and Page Control
- Standard Parity

### SPECIFICATIONS

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Requirements</td>
<td>95-135/205-250 Vac</td>
</tr>
<tr>
<td></td>
<td>47-420 Hz</td>
</tr>
<tr>
<td></td>
<td>250 W</td>
</tr>
<tr>
<td>Word Size</td>
<td>8-bits</td>
</tr>
<tr>
<td>Data:</td>
<td>16-bits</td>
</tr>
<tr>
<td>Address:</td>
<td>8, 16, and 24-bits</td>
</tr>
<tr>
<td>Instruction:</td>
<td></td>
</tr>
<tr>
<td>Memory Capability</td>
<td>65,536 bytes (maximum)</td>
</tr>
<tr>
<td>Instructions</td>
<td>72, variable length</td>
</tr>
<tr>
<td>Clock Signal</td>
<td>Crystal controlled with logic for generating 2-phase non-overlapping signal to MPU and system bus</td>
</tr>
<tr>
<td>Memory Speed</td>
<td>Jumper selectable 1.0, 1.5, or 2.0 MHz</td>
</tr>
<tr>
<td>Interrupt</td>
<td>Maskable and nonmaskable</td>
</tr>
<tr>
<td>Data Terminal Interface</td>
<td></td>
</tr>
<tr>
<td>Characteristics</td>
<td></td>
</tr>
<tr>
<td>Baud Rates (Jumper Selectable)</td>
<td>110, 150, 300, 600, 1200, 2400, 4800, and 9600</td>
</tr>
<tr>
<td>Signal Characteristic</td>
<td>EIA RS-232C compatible</td>
</tr>
<tr>
<td>Operating Temperatures</td>
<td>0 to 55°C</td>
</tr>
</tbody>
</table>
ORDERING INFORMATION

The following table identifies the options of the EXORciser II Development System. For further information, contact your local sales office.

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M6800EXOR</td>
<td>EXORciser II Development System with chassis, 6800 MPU II Module and DEbug II Module. 110 V, 60 Hz</td>
</tr>
<tr>
<td>M6809EXOR</td>
<td>EXORciser II Development System with chassis, 6809 MPU Module and DEbug II Module. 110 V, 60 Hz</td>
</tr>
<tr>
<td>M6800EXORU</td>
<td>EXORciser II Development System with 6800 USE Module and DEbug II Module. 110 V, 60 Hz</td>
</tr>
</tbody>
</table>

RECOMMENDED CONFIGURATION GUIDE

<table>
<thead>
<tr>
<th>6800-Based</th>
<th>6809-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. EXORciser</td>
<td>M6800EXOR</td>
</tr>
<tr>
<td>B. EXORterm 155 Terminal</td>
<td>M68sx10155</td>
</tr>
<tr>
<td>C. 48K RAM (2 MHz Dyn)</td>
<td>MEX6848-22</td>
</tr>
<tr>
<td>D. EXORdisk II</td>
<td>M68DSK2</td>
</tr>
<tr>
<td>E. 781 Printer</td>
<td>M68SP781C1A</td>
</tr>
</tbody>
</table>

Recommended Options:

- System Analyzer: MEX68SA2, M6809SA
- PROM Programmer: M68PP3, M68PP3
- USE: MEX68USEC (see Alternatives A), M6809USE
- M6801 USE: MEX6801, MEX6801
- M6805 USE: MEX6805, MEX6805
- M146805 USE: MEX146805, MEX146805
- M3870 USE: MEX3870M, N/A
- M68000 Development Module: MEX68KDM, MEX68KDM
- BASIC/BASIC-M: M68BASR010M, M6809BASICM
- FORTRAN: M68FTR012M, M6809FORTRN
- COBOL: M68COBOL010M, N/A
- PASCAL Interpreter*: N/A, M6809PASCLI
- MPL*: M68MPLR020M, M6809MPL
- M68000 Cross Macro Assembler*: M68K0XASMBL0, M68K0XASMBL1
- M68000 Cross PASCAL Compiler*: N/A, M68K0XPASCLI

Alternatives:

A. EXORciser with USE | M6800EXORU | N/A (see options)
C. 48K RAM 1 MHz HR | MEX6848-1HR | MEX6848-1HR |
56/64K RAM 1 MHz HR | MEX6864-1HR | MEX6864-1HR |
56/64K RAM 2 MHz Dyn | MEX6864-22 | MEX6864-22 |
16K Static RAM (2 MHz) | MEX6816-22S | MEX6816-22S |
8K Static RAM (2 MHz) | MEX6808-22 | MEX6808-22 |
D. EXORdisk III | M68DSK3 | M68DSK3 |
E. 702 Printer | M68SP702C10 | M68SP702C10 |
703 Printer | MPRINT703 | MPRINT703 |

*Note: Requires 56K bytes of RAM, minimum.

Motorola reserves the right to make changes to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others.
# TABLE A. MEX8801EVM Debug COMMANDS

<table>
<thead>
<tr>
<th>Command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>Load a program from tape.</td>
</tr>
<tr>
<td>L &lt;OFFSET&gt;</td>
<td>Load a program from tape with an offset.</td>
</tr>
<tr>
<td>V</td>
<td>Verify that a program was properly loaded.</td>
</tr>
<tr>
<td>V &lt;OFFSET&gt;</td>
<td>Verify that a program was properly loaded with an offset.</td>
</tr>
<tr>
<td>D &lt;ADDR1&gt;, &lt;ADDR2&gt;</td>
<td>Display contents of memory from &lt;ADDR1&gt; to &lt;ADDR2&gt;.</td>
</tr>
<tr>
<td>P &lt;ADDR1&gt;, &lt;ADDR2&gt;</td>
<td>Punch/record on tape the contents of memory from &lt;ADDR1&gt; to &lt;ADDR2&gt;.</td>
</tr>
<tr>
<td>M &lt;ADDR&gt;</td>
<td>Examine/modify the contents of the specified address location.</td>
</tr>
<tr>
<td>&lt;DATA&gt;</td>
<td>Enter one byte of data to replace the value at the current address location.</td>
</tr>
<tr>
<td>LF</td>
<td>Display the contents of the next sequential memory location on the next line and enable the contents of this location to be changed (LF = Line Feed).</td>
</tr>
<tr>
<td>SP</td>
<td>Display the contents of the next sequential memory location on the same line and enable the contents of this location to be changed (SP = Space).</td>
</tr>
<tr>
<td>,</td>
<td>Increment the memory location pointer, but do not display the address or data. The contents of this memory location may still be changed. This entry permits data to be entered at sequential memory locations without displaying the current address or data.</td>
</tr>
<tr>
<td>UA</td>
<td>Display the contents of the previous memory location on the next line and enable the contents of this location to be changed (UA = Up Arrow).</td>
</tr>
<tr>
<td>/</td>
<td>Display the current address of the memory location pointer and the contents of that location.</td>
</tr>
<tr>
<td>CR</td>
<td>Terminate the memory examine/modify command and accept next command (CR = Carriage Return).</td>
</tr>
<tr>
<td>&lt;ADDR&gt; /</td>
<td>Display the contents of the specified address location and enable the contents of this location to be changed.</td>
</tr>
<tr>
<td>/</td>
<td>Display the address and contents of the memory location last referenced by the memory examine/modify command.</td>
</tr>
</tbody>
</table>
MEX6801
MC6801 EXORciser Support System

- User System Evaluator (USE) Capability
- Real-Time Emulation of MC6801
  - Single-Chip Mode
  - Multiplexed Mode
  - Non-Multiplexed Mode
- MC68701 Program Capability
- Compatible with Current System Development Tools Including EXORciser/
  EXORterm and all support modules

Minimum System Requirements
- EXORciser/EXORterm with 6800 MPU
  and DEbug Modules
- EXORdisk II
- 24K Memory
- EXORterm 150 (EXORciser® system only)

The MC6801 Support System — MEX6801 — upgrades existing Motorola EXORciser development tools for development of MC6801-based systems. All three modes of MC6801 operation; single-chip, expanded multiplexed and expanded non-multiplexed, are supported by MEX6801.

MEX6801 is fully compatible with all current EXORciser/EXORterm M6800 supporting hardware and software and includes the USE (User System Evaluator) function to provide designers with the widest range of development capabilities. Used with an EXORciser I or II or an EXORterm 200 or 220, the support system fosters real-time emulation of the MC6801 application hardware and facilitates the debugging of software developed for use on the hardware.

The MC6801 Support System comprises the following: three printed circuit boards including the Intercept Module, the Control Module and the Buffer Board and housing; four cable assemblies and an MDOS diskette containing the MC6801 monitor/debug program — ONEbug, and a Macro Assembler.

MEX6801 utilizes a two processor technique in which the support system’s MC6801 is slaved to the EXORciser/EXORterm’s MC6800 with just one allowed control of the bus at any one time. The MC6800 executes its own instructions, controls disk operation and, in conjunction with EXbug 2, provides MC6800-dependent monitoring and debugging capabilities. Additional capabilities for debugging code calling for execution of MC6801 instructions are provided by the ONEbug monitor which controls MC6801 operation. ONEbug also controls the MEX6801 provision for programming the 2K bytes of EPROM contained in MC68701, the EPROM version of MC6801.

For system emulation, MEX6801 provides the user with various options for structuring the address map, selection of MC6801 operating mode and a choice of several system clocks. The MEX6801 also permits MC6801 system emulation with its own control module completely isolated from the EXORciser bus signals to provide final assurance of correct design prior to mask commitment.
MEX6801 DEbug Functions

- Load/verify object tape from/against memory
- Search/punch object tape
- Display/change terminal pad value
- Abort current entry
- Wait for character entry
- Initialize memory
- Move/display/change/print memory
- Calculate relative offset
- Display/change offset
- Display/change search mask and memory range
- Search memory and display location of match
- Display/change MPU register(s)
  - A accumulator
  - B accumulator
  - Index register
- Display/change program register(s)
- Display/change second level SWI vector
- Display/set/remove breakpoint(s)
- Display/enable/change halt-on-address or scope sync
- Display/enable/change trace to address
- Trace n instruction(s)
- Go to specified/restart program address
- Execute program
- Execute, pass breakpoint n times

Ordering Information

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEX6801</td>
<td>MC6801 Support System</td>
</tr>
</tbody>
</table>
## INDEX

<table>
<thead>
<tr>
<th>-aaaa-</th>
<th>-ccccc-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator, 1-3, 4-10</td>
<td>CC1:CC2 (bits), 1-7, 6-2, 6-4</td>
</tr>
<tr>
<td>Address formats, A-91, A-92, A-93</td>
<td>Comparisons</td>
</tr>
<tr>
<td>Addressing modes, 4-3</td>
<td>MC6800 bus, 2-21</td>
</tr>
<tr>
<td>direct, 4-5</td>
<td>MC6803, 2-21</td>
</tr>
<tr>
<td>extended, 4-5</td>
<td>MC68701, 2-21, E-1</td>
</tr>
<tr>
<td>inherent, 4-4</td>
<td>MC68120, 2-22</td>
</tr>
<tr>
<td>immediate, 4-4</td>
<td>Condition code register, 1-3, 4-10, 4-11</td>
</tr>
<tr>
<td>indexed, 4-7</td>
<td>Configurations, 1-4</td>
</tr>
<tr>
<td>relative, 4-6</td>
<td>Counter register, 1-8, 7-1, 7-2</td>
</tr>
<tr>
<td>Applications, 8-1</td>
<td>Cycle-by-cycle bus activity, F-1</td>
</tr>
<tr>
<td>Arithmetic instructions, 4-11, 4-12, 4-34</td>
<td>Crystal parameters, 3-5</td>
</tr>
<tr>
<td>AS (Address Strobe), 3-37, 3-42, 8-1, 8-2, I-1</td>
<td>-ddddd-</td>
</tr>
<tr>
<td>ASCII conversion table, C-1</td>
<td>Data direction register, 1-6, 3-16</td>
</tr>
<tr>
<td>Assembler source statements, 4-2</td>
<td>Data handling instructions, 4-14</td>
</tr>
<tr>
<td>Data handling instructions, 4-14, 4-34</td>
<td>Data test instructions, 4-13</td>
</tr>
<tr>
<td>Data test instructions, 4-13</td>
<td>Direct addressing mode, 4-5</td>
</tr>
<tr>
<td>Direct addressing mode, 4-5</td>
<td>Division, 4-53</td>
</tr>
<tr>
<td>Division, 4-53</td>
<td>DIV16B (Routine), 4-55</td>
</tr>
<tr>
<td>DIV16B (Routine), 4-55</td>
<td>Dual processor interface, 8-39</td>
</tr>
<tr>
<td>-bbbbb-</td>
<td>-eeeee-</td>
</tr>
<tr>
<td>Block diagram</td>
<td>ECHO (Routine), 7-14</td>
</tr>
<tr>
<td>MC6801, 1-2, 3-2</td>
<td>E (Enable), 3-6</td>
</tr>
<tr>
<td>MC68701 EPROM programming circuit, E-11</td>
<td>EICI (bit), 1-8, 7-1, 7-5</td>
</tr>
<tr>
<td>Programmable Timer, 1-8, 7-1, 7-2</td>
<td>Enable (E), 3-6 See also Timing</td>
</tr>
<tr>
<td>SCI, 1-7, 6-3</td>
<td>EOCI (bit), 1-8, 7-1, 7-5</td>
</tr>
<tr>
<td>Bus</td>
<td>EPROM (Routine), E-12</td>
</tr>
<tr>
<td>Expanded multiplexed, 2-7, 3-42</td>
<td>EPROM programming, E-9</td>
</tr>
<tr>
<td>Expanded non-multiplexed, 2-5, 3-33</td>
<td>ETOI (bit), 1-8, 7-1, 7-5</td>
</tr>
<tr>
<td>BCD1 (Routine), 4-28</td>
<td>Expanded non-multiplexed mode, See Mode</td>
</tr>
<tr>
<td>BLOCKC (Routine), 4-30</td>
<td>Expanded multiplexed mode bus clocking, 1-1</td>
</tr>
<tr>
<td>BLOCKM (Routine), 4-31</td>
<td>Expanded multiplexed mode, See Mode</td>
</tr>
<tr>
<td>BCDAD1 (Routine), 4-39</td>
<td>EXTAL2, 3-4</td>
</tr>
<tr>
<td>BCDAD2 (Routine), 4-40</td>
<td>Extended addressing mode, 4-5</td>
</tr>
<tr>
<td>BCDSB1 (Routine), 4-42</td>
<td>Executable instructions (See Instruction Set)</td>
</tr>
<tr>
<td>BCDSB2 (Routine), 4-43</td>
<td></td>
</tr>
</tbody>
</table>
Framing error, 6-6, 6-10

Glossary, G-1

ICF (bit), 1-8, 7-1
IEDG (bit), 1-8, 7-1
Immediate addressing mode, 4-4
Index register instructions, 4-16, 4-25, 5-25
Indexed addressing mode, 4-7
Inherent addressing mode, 4-4
Input capture register, 1-8, 7-1, 7-5
Instruction cycle count, H-1
Instruction set, 4-7, 4-8, A-1
Internal register area, 2-19
Interrupt
considerations, 5-1, 5-3
flowchart, 5-10
generation, 5-3
instructions, 5-18
response, 5-7
SCI, 6-12
service, 5-21
sequence, 5-14
sources, 5-1
vectors, 5-15, 5-17, E-6
window, 5-9
IOS (Input/Output Select), 1-5, 3-29, 3-33
3-34, 8-19, 8-20
IRQ1, 3-14, 5-6
IRQ2, 1-2, 5-6
IS3, 3-23
IS3 FLAG (bit) 3-27
IS3 IRQ1 ENABLE (bit) 3-27
Jump and branch instructions, 4-9

Keyboard interface, 8-10
KEYIN (Routine), 8-13

Labels, 4-3
Latch
Port 3, 3-24
De-multiplexing, 3-43
Line printer interface, 8-8
Logic diagram
Port 1, 3-16
Port 2 bit 0, 3-20
Port 2 bit 1, 3-20
Port 2 bit 2, 3-21
Port 2 bit 3, 3-22
Port 2 bit 4, 3-23
Port 3, 3-26, 3-31, 3-38
Port 4, 3-29, 3-33, 3-41
Logic instructions, 4-11
LATCH ENABLE (bit), 3-27, 8-12

Mask option, 2-20, E-6
MC6800, 2-21
MC6803, 2-21
MC68120, 2-22
MC68701, 2-21, E-1
Memory maps, 2-9, E-7
Memory interface, 8-1
Mode
Characteristics, 2-2
Expanded multiplexed, 1-1, 1-2, 1-5, 2-2, 2-6, 2-7, 3-36, 3-37, 3-42, 8-14
Expanded non-multiplexed, 1-1, 1-2, 1-5, 2-2, 2-4, 2-6, 3-29, 3-33, 8-19
Fundamental modes, 2-3
Levels and timing, 3-9
Programming, 2-20, 3-8
Select pins, 3-19
Single chip, 1-1, 1-2, 1-4, 2-2, 2-4, 2-18
3-23, 3-24, 8-8
Test modes, 2-2, 2-8, 2-9, 2-14, 3-39, J-1

Index-2
MUL16 (Routine), 5-26
MUL16A (Routine), 4-46
MUL16B (Routine), 4-48
MULT16 (Routine), 4-52
Multi-byte addition and subtraction, 4-36
Multiplication 4-38
Using MUL, 4-45
Booth’s Algorithm, 4-50

NMI, 3-7, 3-13, 5-3
Number systems, 4-29

OCF (bit), 1-8, 1-9, 7-1, 7-5
OKBAD (Routine), 6-20
OLVL (Bit), 1-8, 7-1, 7-5
Opcode map, B-1
Operating mode (See mode)
ORFE (bit), 1-7, 6-2, 6-5
OS3, 3-24
OSS (bit), 3-27
Output compare register, 1-8
Overrun (SCI), 6-6

P10-P17 (Port 1), 3-14
P20-P24 (Port 2), 3-18
P30-P37 (Port 3), 3-25, 3-30, 3-37, 3-38
P40-P47 (Port 4), 3-27, 3-32, 3-40
PC0-2 (bits), 2-8, 2-14, 2-15, 2-20, 3-19
Period measurement, 8-22
Pin description, 3-4
Pin diagram, 3-1
PLC (bit), E-8
Polling, 5-2
Port 2 Data Register, 3-19
Port 3 Control and Status Register, 3-27
POUTCH (Routine), 8-11
Powers of two, D-1
PPC (bit), E-8
Prioritized interrupts, 5-4, 5-15, 8-14
Program control instructions, 4-16
Program restartability, 5-23

Programmable timer,
Block diagram, 1-8, 7-1
Counter register, 1-8, 7-1, 7-2
Description, 1-8, 7-1
Examples, 7-6
Input capture register, 1-8, 7-1, 7-4
Interface pins, 3-19
Output compare register, 1-8, 7-1
Output level register, 7-2, 7-3
Timer control and status register (TCSR), 1-8, 1-9, 7-1, 7-5
Programming model, 1-3, 4-1

RAM Control Register, 3-13
RAM/EPROM Control Register, E-8
RAME (bit), 3-11
Rate and Mode Control Register (See SCI)
RDRF (bit), 1-7, 6-2, 6-5
RE (bit), 1-7, 6-2, 6-5
R/\ (Read/Write), 3-9, 3-30, 3-34, 3-42.
5-14, 5-20, 8-2, 8-19
Re-entrant programming, 5-23
Relative addressing mode, 4-6
RESET, 3-7, E-3, J-1
RESET/Vpp, E-1
RIE (bit), 1-7, 6-2, 6-5
RMCR (See SCI)

SC1
I53, 3-23
I5S, 3-29
AS, 3-37

SC2
OS3, 3-24
R/\, 3-30, 3-37
AS, 3-37

SCI (Serial Communications Interface)
Baud Rates, 6-4
Block diagram, 6-3
Clocking options, 6-4, 6-6
Data formats, 6-4, 6-8
Examples, 6-13
Framing error, 6-9
Interface pins, 3-21

Index-3