SUBJECT:

Description and Use of the Multics Processor.

SOFTWARE SUPPORTED:

All Multics Software Releases

DATE:

October, 1975

ORDER NUMBER:

AL39, Rev. 0
This document describes the Processor used in the Multics system. It is assumed that the reader is familiar with the overall modular organization of the Multics system and with the philosophy of asynchronous operation. In addition, this manual presents a thorough discussion of virtual memory addressing concepts including segmentation and paging.

The manual is intended for use by system programmers responsible for writing software to interface with the special virtual memory hardware and with the fault and interrupt portions of the hardware. It should also prove valuable to programmers who must use machine instructions (particularly language translator implementors) and to those persons responsible for analyzing crash conditions in System Dumps.

© 1974 Honeywell Information Systems, Inc.

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975
# CONTENTS

## Section I

<table>
<thead>
<tr>
<th>Introduction To Processor</th>
<th>1-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features of the Multics Processor</td>
<td>1-1</td>
</tr>
<tr>
<td>Segmentation and Paging</td>
<td>1-1</td>
</tr>
<tr>
<td>Address Modification and Address Appending</td>
<td>1-2</td>
</tr>
<tr>
<td>Faults and Interrupts</td>
<td>1-2</td>
</tr>
<tr>
<td>Summary of Processor Features</td>
<td>1-2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Processor Modes of Operation</th>
<th>1-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Modes</td>
<td>1-3</td>
</tr>
<tr>
<td>Normal Mode</td>
<td>1-3</td>
</tr>
<tr>
<td>Privileged Mode</td>
<td>1-3</td>
</tr>
<tr>
<td>Addressing Modes</td>
<td>1-4</td>
</tr>
<tr>
<td>Absolute Mode</td>
<td>1-4</td>
</tr>
<tr>
<td>Append Mode</td>
<td>1-4</td>
</tr>
<tr>
<td>Bar Mode</td>
<td>1-4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Processor Unit Functions</th>
<th>1-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appending Unit</td>
<td>1-5</td>
</tr>
<tr>
<td>Associative Memory Assemblies</td>
<td>1-5</td>
</tr>
<tr>
<td>Control Unit</td>
<td>1-6</td>
</tr>
<tr>
<td>Operation Unit</td>
<td>1-6</td>
</tr>
<tr>
<td>Decimal Unit</td>
<td>1-6</td>
</tr>
</tbody>
</table>

## Section II

<table>
<thead>
<tr>
<th>Machine Instructions</th>
<th>2-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Repertoire</td>
<td>2-1</td>
</tr>
<tr>
<td>Basic Operations</td>
<td>2-1</td>
</tr>
<tr>
<td>Extended Instruction Set (EIS) Operations</td>
<td>2-1</td>
</tr>
<tr>
<td>EIS Single-Word Operations</td>
<td>2-1</td>
</tr>
<tr>
<td>EIS Multi-Word Operations</td>
<td>2-2</td>
</tr>
<tr>
<td>Format of Instruction Description</td>
<td>2-2</td>
</tr>
<tr>
<td>Definitions of Notation and Symbols</td>
<td>2-4</td>
</tr>
<tr>
<td>Main Store Addresses</td>
<td>2-4</td>
</tr>
<tr>
<td>Index Values</td>
<td>2-4</td>
</tr>
<tr>
<td>Abbreviations and Symbols</td>
<td>2-4</td>
</tr>
<tr>
<td>Register Positions and Contents</td>
<td>2-5</td>
</tr>
<tr>
<td>Other Symbols</td>
<td>2-6</td>
</tr>
<tr>
<td>Arrangement of Instructions</td>
<td>2-6</td>
</tr>
<tr>
<td>Common Attributes of Instructions</td>
<td>2-6</td>
</tr>
<tr>
<td>Illegal Modification</td>
<td>2-6</td>
</tr>
<tr>
<td>Parity Indicator</td>
<td>2-6</td>
</tr>
<tr>
<td>Instruction Word Formats</td>
<td>2-7</td>
</tr>
<tr>
<td>Basic and EIS Single-Word Instructions</td>
<td>2-7</td>
</tr>
<tr>
<td>Indirect Words</td>
<td>2-8</td>
</tr>
</tbody>
</table>

<p>| EIS Multi-Word Instructions | 2-9 |
| EIS Modification Fields (MF) | 2-9 |
| EIS Operand Descriptors and Indirect Pointers | 2-11 |
| Operand Descriptor Indirect Pointer Format | 2-11 |
| Alphanumeric Operand Descriptor Format | 2-12 |
| Numeric Operand Descriptor Format | 2-14 |
| Bit String Operand Descriptor Format | 2-15 |
| Fixed Point Arithmetic Instructions | 2-16 |
| Data Movement Load Instructions | 2-16 |
| Data Movement Store Instructions | 2-25 |</p>
<table>
<thead>
<tr>
<th>CONTENTS (Cont)</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Movement Shift Instructions</td>
<td>2-34</td>
</tr>
<tr>
<td>Addition Instructions</td>
<td>2-40</td>
</tr>
<tr>
<td>Subtraction Instructions</td>
<td>2-40</td>
</tr>
<tr>
<td>Multiplication Instructions</td>
<td>2-55</td>
</tr>
<tr>
<td>Division Instructions</td>
<td>2-57</td>
</tr>
<tr>
<td>Negate Instructions</td>
<td>2-60</td>
</tr>
<tr>
<td>Comparison Instructions</td>
<td>2-61</td>
</tr>
<tr>
<td>Miscellaneous Instructions</td>
<td>2-66</td>
</tr>
<tr>
<td>Boolean Operation Instructions</td>
<td>2-67</td>
</tr>
<tr>
<td>AND Instructions</td>
<td>2-67</td>
</tr>
<tr>
<td>OR Instructions</td>
<td>2-70</td>
</tr>
<tr>
<td>Exclusive OR Instructions</td>
<td>2-73</td>
</tr>
<tr>
<td>Comparative AND Instructions</td>
<td>2-76</td>
</tr>
<tr>
<td>Comparative NOT Instructions</td>
<td>2-78</td>
</tr>
<tr>
<td>Floating Point Arithmetic Instructions</td>
<td>2-80</td>
</tr>
<tr>
<td>Data Movement Load Instructions</td>
<td>2-80</td>
</tr>
<tr>
<td>Data Movement Store Instructions</td>
<td>2-81</td>
</tr>
<tr>
<td>Addition Instructions</td>
<td>2-84</td>
</tr>
<tr>
<td>Subtraction Instructions</td>
<td>2-87</td>
</tr>
<tr>
<td>Multiplication Instructions</td>
<td>2-90</td>
</tr>
<tr>
<td>Division Instructions</td>
<td>2-93</td>
</tr>
<tr>
<td>Negate Instructions</td>
<td>2-97</td>
</tr>
<tr>
<td>Normalize Instructions</td>
<td>2-98</td>
</tr>
<tr>
<td>Round Instructions</td>
<td>2-99</td>
</tr>
<tr>
<td>Compare Instructions</td>
<td>2-101</td>
</tr>
<tr>
<td>Miscellaneous Instructions</td>
<td>2-103</td>
</tr>
<tr>
<td>Transfer Instructions</td>
<td>2-105</td>
</tr>
<tr>
<td>Pointer Register Instructions</td>
<td>2-117</td>
</tr>
<tr>
<td>Data Movement Load Instructions</td>
<td>2-117</td>
</tr>
<tr>
<td>Data Movement Store Instructions</td>
<td>2-121</td>
</tr>
<tr>
<td>Address Arithmetic Instructions</td>
<td>2-124</td>
</tr>
<tr>
<td>Miscellaneous Instructions</td>
<td>2-125</td>
</tr>
<tr>
<td>Miscellaneous Instructions</td>
<td>2-126</td>
</tr>
<tr>
<td>Calendar Clock Instruction</td>
<td>2-126</td>
</tr>
<tr>
<td>Deral Instruction</td>
<td>2-127</td>
</tr>
<tr>
<td>Execute Instructions</td>
<td>2-128</td>
</tr>
<tr>
<td>Master Mode Entry Instructions</td>
<td>2-130</td>
</tr>
<tr>
<td>No Operation Instructions</td>
<td>2-133</td>
</tr>
<tr>
<td>Repeat Instructions</td>
<td>2-135</td>
</tr>
<tr>
<td>Ring Alarm Register Instruction</td>
<td>2-144</td>
</tr>
<tr>
<td>Store Base Address Register Instruction</td>
<td>2-145</td>
</tr>
<tr>
<td>Translation Instructions</td>
<td>2-146</td>
</tr>
<tr>
<td>Privileged Instructions</td>
<td>2-148</td>
</tr>
<tr>
<td>Register Load Instructions</td>
<td>2-148</td>
</tr>
<tr>
<td>Register Store Instructions</td>
<td>2-155</td>
</tr>
<tr>
<td>Clear Associative Memory Instructions</td>
<td>2-162</td>
</tr>
<tr>
<td>Configuration and Status Instructions</td>
<td>2-164</td>
</tr>
<tr>
<td>System Control Instructions</td>
<td>2-167</td>
</tr>
<tr>
<td>Miscellaneous Instructions</td>
<td>2-171</td>
</tr>
</tbody>
</table>

Extended Instruction Set (EIS) | 2-172 |
| Address Register Load Instructions | 2-172 |
| Address Register Store Instructions | 2-175 |
| Address Register Special Arithmetic Instructions | 2-178 |
| Alphanumeric Compare Instructions | 2-186 |
| Alphanumeric Move Instructions | 2-197 |
| Numeric Compare Instruction | 2-205 |
| Numeric Move Instructions | 2-208 |
| Bit String Combine Instructions | 2-213 |
| Bit String Compare Instructions | 2-217 |
## Section III

**Data Representation**

- Information Organization .................................. 3-1
- Position Numbering ............................................ 3-1
- Number System ................................................. 3-1
- Information Formats .......................................... 3-2
- Data Parity ..................................................... 3-4
- Representation of Data ......................................... 3-5

**Numeric Data** .................................................. 3-5

- Fixed Point Binary Data ........................................ 3-5
- Fixed Point Binary Integers ................................... 3-5
- Fixed Point Binary Fractions ................................ 3-6
- Floating Point Binary Data .................................... 3-8
- Overtlength Registers ......................................... 3-9
- Normalized Numbers .......................................... 3-9

**Decimal Data** .................................................. 3-11

- Decimal Data Values .......................................... 3-13
- Decimal Zero ................................................... 3-14

**Alphanumeric Data** ........................................... 3-14

- Character String Data ......................................... 3-14
- Bit String Data ................................................. 3-15

## Section IV

**Program Accessible Registers** ................................ 4-1

- Accumulator Register (A) ...................................... 4-2
- Quotient Register (Q) ........................................... 4-2
- Accumulator-Quotient Register (AQ) ........................ 4-3
- Exponent Register (E) ......................................... 4-4
- Exponent-Accumulator-Quotient Register (EAQ) ............ 4-4
- Index Registers (Xn) ........................................... 4-5
- Indicator Register (IR) ......................................... 4-5
- Base Address Register (BAR) .................................... 4-9

- Timer Register (TR) ............................................ 4-9
- Ring Alarm Register (RALR) .................................... 4-10
- Pointer Registers (PRn) ........................................ 4-11
- Procedure Pointer Register (PPR) ............................ 4-13
- Temporary Pointer Register (TPR) ............................ 4-15
- Descriptor Segment Base Register (DSBR, DBR) ............. 4-16
- Segment Descriptor Word Associative Memory (SDWAM) .... 4-18
- Page Table Word Associative Memory (PTWAM) ................ 4-21
- Fault Register .................................................. 4-23
- Mode Register (MR) ............................................. 4-25

**REVIEw DRAFT**

SUBJECT TO CHANGE

October, 1975
<table>
<thead>
<tr>
<th>Section</th>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section V</td>
<td>Addressing -- Segmentation and Paging</td>
<td>5-1</td>
</tr>
<tr>
<td></td>
<td>Addressing Modes</td>
<td>5-1</td>
</tr>
<tr>
<td></td>
<td>Absolute Mode</td>
<td>5-1</td>
</tr>
<tr>
<td></td>
<td>Append Mode</td>
<td>5-2</td>
</tr>
<tr>
<td></td>
<td>Segmentation</td>
<td>5-2</td>
</tr>
<tr>
<td></td>
<td>Paging</td>
<td>5-3</td>
</tr>
<tr>
<td></td>
<td>Changing Addressing Modes</td>
<td>5-6</td>
</tr>
<tr>
<td></td>
<td>Address Appending</td>
<td>5-7</td>
</tr>
<tr>
<td></td>
<td>Address Appending Sequences</td>
<td>5-7</td>
</tr>
<tr>
<td></td>
<td>Appending Unit Data Word Formats</td>
<td>5-10</td>
</tr>
<tr>
<td></td>
<td>Segment Descriptor Word (SDW) Format</td>
<td>5-10</td>
</tr>
<tr>
<td></td>
<td>Page Table Word (PTW) Format</td>
<td>5-11</td>
</tr>
<tr>
<td>Section VI</td>
<td>Effective Address Formation</td>
<td>6-1</td>
</tr>
<tr>
<td></td>
<td>Definition of Effective Address</td>
<td>6-1</td>
</tr>
<tr>
<td></td>
<td>Types of Effective Address Formation</td>
<td>6-1</td>
</tr>
<tr>
<td></td>
<td>Effective Address Formation Description</td>
<td>6-2</td>
</tr>
<tr>
<td></td>
<td>Effective Address Formation Involving Offset Only</td>
<td>6-2</td>
</tr>
<tr>
<td></td>
<td>The Address Modifier (TAG) Field</td>
<td>6-2</td>
</tr>
<tr>
<td></td>
<td>General Types of Offset Modification</td>
<td>6-2</td>
</tr>
<tr>
<td></td>
<td>Effective Address Formation Flowcharts</td>
<td>6-4</td>
</tr>
<tr>
<td></td>
<td>Register (R) Modification</td>
<td>6-4</td>
</tr>
<tr>
<td></td>
<td>Register Then Indirect (RI) Modification</td>
<td>6-6</td>
</tr>
<tr>
<td></td>
<td>Indirect Then Register (IR) Modification</td>
<td>6-8</td>
</tr>
<tr>
<td></td>
<td>Indirect Then Tally (IT) Modification</td>
<td>6-9</td>
</tr>
<tr>
<td></td>
<td>Effective Address Formation Involving Both</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Segment Number and Offset</td>
<td>6-15</td>
</tr>
<tr>
<td></td>
<td>The Use of Bit 29 of the Instruction Word</td>
<td>6-16</td>
</tr>
<tr>
<td></td>
<td>Special Modifiers</td>
<td>6-16</td>
</tr>
<tr>
<td></td>
<td>Indirect to Pointer (ltp) Modification</td>
<td>6-17</td>
</tr>
<tr>
<td></td>
<td>Indirect to Segment (lts) Modification</td>
<td>6-18</td>
</tr>
<tr>
<td></td>
<td>Effective Segment Number Generation</td>
<td>6-19</td>
</tr>
<tr>
<td></td>
<td>Effective Address Formation for Extended</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Instruction Set</td>
<td>6-21</td>
</tr>
<tr>
<td></td>
<td>Character- and Bit-String Addressing</td>
<td>6-23</td>
</tr>
<tr>
<td></td>
<td>Character- and Bit-String Address Arithmetic Algorithms</td>
<td>6-23</td>
</tr>
<tr>
<td></td>
<td>9-Bit Character String Address Arithmetic</td>
<td>6-24</td>
</tr>
</tbody>
</table>

Section VII

Faults and Interrupts                                               | 7-1  |
Fault Cycle Sequence                                                | 7-1  |
Fault Priority                                                      | 7-2  |
Fault Recognition                                                   | 7-3  |
Fault Descriptions                                                  | 7-4  |

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975
CONTENTS (Cont)

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td>Faults</td>
<td>7-4</td>
</tr>
<tr>
<td>Group 2</td>
<td>Faults</td>
<td>7-4</td>
</tr>
<tr>
<td>Group 3</td>
<td>Faults</td>
<td>7-5</td>
</tr>
<tr>
<td>Group 4</td>
<td>Faults</td>
<td>7-5</td>
</tr>
<tr>
<td>Group 5</td>
<td>Faults</td>
<td>7-6</td>
</tr>
<tr>
<td>Group 6</td>
<td>Faults</td>
<td>7-7</td>
</tr>
<tr>
<td>Group 7</td>
<td>Faults</td>
<td>7-7</td>
</tr>
<tr>
<td></td>
<td>Program Interrupts and External Faults</td>
<td>7-8</td>
</tr>
<tr>
<td></td>
<td>Execute Interrupt Sampling</td>
<td>7-8</td>
</tr>
<tr>
<td></td>
<td>Execute Interrupt Cycle Sequence</td>
<td>7-9</td>
</tr>
<tr>
<td>Section VIII</td>
<td>Hardware Ring Implementation</td>
<td>8-1</td>
</tr>
<tr>
<td></td>
<td>Ring Protection Philosophy</td>
<td>8-1</td>
</tr>
<tr>
<td></td>
<td>Ring Protection in Multics</td>
<td>8-1</td>
</tr>
<tr>
<td></td>
<td>Ring Protection in the Multics Processor</td>
<td>8-2</td>
</tr>
<tr>
<td></td>
<td>Appending Unit Operation with Ring Mechanism</td>
<td>8-4</td>
</tr>
<tr>
<td>Section IX</td>
<td>Cache Store Operation</td>
<td>9-1</td>
</tr>
<tr>
<td></td>
<td>Philosophy of Cache Store</td>
<td>9-1</td>
</tr>
<tr>
<td></td>
<td>Cache Store Organization</td>
<td>9-1</td>
</tr>
<tr>
<td></td>
<td>Cache Store/Main Store Mapping</td>
<td>9-2</td>
</tr>
<tr>
<td></td>
<td>Cache Store Addressing</td>
<td>9-4</td>
</tr>
<tr>
<td></td>
<td>Cache Store Control</td>
<td>9-4</td>
</tr>
<tr>
<td></td>
<td>Enabling and Disabling Cache Store</td>
<td>9-4</td>
</tr>
<tr>
<td></td>
<td>Cache Store Control in Segment Descriptor Words</td>
<td>9-5</td>
</tr>
<tr>
<td></td>
<td>Loading the Cache Store</td>
<td>9-6</td>
</tr>
<tr>
<td></td>
<td>General Clear</td>
<td>9-6</td>
</tr>
<tr>
<td></td>
<td>Selective Clear</td>
<td>9-7</td>
</tr>
<tr>
<td></td>
<td>Dumping the Cache Store</td>
<td>9-7</td>
</tr>
</tbody>
</table>
# CONTENTS (Cont)

<table>
<thead>
<tr>
<th>Appendix A</th>
<th>Operation Code Map</th>
<th>A-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix B</td>
<td>Alphabetic Operation Code List</td>
<td>B-1</td>
</tr>
<tr>
<td>Appendix C</td>
<td>Address Modifiers</td>
<td>C-1</td>
</tr>
</tbody>
</table>

## ILLUSTRATIONS

| Figure 2-1 | Basic and EIS Single-Word Instruction Format | 2-7 |
| Figure 2-2 | Indirect Word Format | 2-8 |
| Figure 2-3 | EIS Multi-Word Format | 2-9 |
| Figure 2-4 | EIS Modification Field (MF) Format | 2-10 |
| Figure 2-5 | Operand Descriptor Indirect Pointer Format | 2-12 |
| Figure 2-6 | Alphanumeric Operand Descriptor Format | 2-14 |
| Figure 2-7 | Numeric Operand Descriptor Format | 2-14 |
| Figure 2-8 | Bit String Operand Descriptor Format | 2-15 |
| Figure 2-9 | Repeat Double (RPO) Instruction Word Format | 2-135 |
| Figure 2-10 | Repeat Link (RPL) Instruction Word Format | 2-138 |
| Figure 2-11 | Repeat (RPT) Instruction Word Format | 2-141 |
| Figure 2-12 | EIS Address Register Special Arithmetic Instruction Format | 2-178 |
| Figure 2-13 | Compare Alphanumeric Strings (CMPC) EIS Multi-Word Instruction Format | 2-186 |
| Figure 2-14 | Scan Characters Double (SCD) EIS Multi-Word Instruction Format | 2-188 |
| Figure 2-15 | Scan with Mask (SCM) EIS Multi-Word Instruction Format | 2-191 |
| Figure 2-16 | Test Character and Translate (TCT) EIS Multi-Word Instruction Format | 2-194 |
| Figure 2-17 | Move Alphanumeric Left to Right (MLR) EIS Multi-Word Instruction Format | 2-197 |
| Figure 2-18 | Move Alphanumeric Edited (MVE) EIS Multi-Word Instruction Format | 2-200 |
| Figure 2-19 | Move Alphanumeric with Translation (MVTE) EIS Multi-Word Instruction Format | 2-202 |
| Figure 2-20 | Compare Numeric (CMNP) EIS Multi-Word Instruction Format | 2-205 |
| Figure 2-21 | Move Numeric (MN) EIS Multi-Word Instruction Format | 2-208 |
| Figure 2-22 | Move Numeric Edited (MVNE) EIS Multi-Word Instruction Format | 2-211 |
| Figure 2-23 | Combine Bit Strings Left (CSL) EIS Multi-Word Instruction Format | 2-213 |
| Figure 2-24 | Compare Bit Strings (CMPB) EIS Multi-Word Instruction Format | 2-217 |
| Figure 2-25 | Binary to Decimal Convert (BTD) EIS Multi-Word Instruction Format | 2-221 |
| Figure 2-26 | Decimal to Binary Convert (DTB) EIS Multi-Word Instruction Format | 2-223 |
| Figure 2-27 | Add Using 2 Decimal Operands (ADZD) EIS Multi-Word Instruction Format | 2-225 |
| Figure 2-28 | Add Using 3 Decimal Operands (ADZD) EIS Multi-Word Instruction Format | 2-228 |
| Figure 2-29 | Micro Operation (MOP) Character Format | 2-240 |
| Figure 3-1 | Unstructured Machine Word Format | 3-2 |
| Figure 3-2 | Unstructured Word Pair Format | 3-3 |
| Figure 3-3 | Unstructured 4-bit Character Format | 3-3 |
| Figure 3-4 | Unstructured 8-bit Character Format | 3-3 |
### CONTENTS (Cont)

<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-4</td>
<td>Unstructured 18-bit Half Word Format</td>
</tr>
<tr>
<td>3-4</td>
<td>Upper 18-bit Half Word Floating Point Binary Operand Format</td>
</tr>
<tr>
<td>3-8</td>
<td>Lower 18-bit Half Word Floating Point Binary Operand Format</td>
</tr>
<tr>
<td>3-9</td>
<td>Single Precision Floating Point Binary Operand Format</td>
</tr>
<tr>
<td>3-9</td>
<td>Double Precision Floating Point Binary Operand Format</td>
</tr>
<tr>
<td>4-2</td>
<td>Accumulator Register (A) Format</td>
</tr>
<tr>
<td>4-2</td>
<td>Quotient Register (Q) Format</td>
</tr>
<tr>
<td>4-3</td>
<td>Accumulator-Quotient Register (AQ) Format</td>
</tr>
<tr>
<td>4-4</td>
<td>Exponent Register (E) Format</td>
</tr>
<tr>
<td>4-4</td>
<td>Exponent-Accumulator-Quotient Register (EAQ) Format</td>
</tr>
<tr>
<td>4-5</td>
<td>Index Register (Xn) Format</td>
</tr>
<tr>
<td>4-5</td>
<td>Indicator Register (IR) Format</td>
</tr>
<tr>
<td>4-9</td>
<td>Base Address Register (BAR) Format</td>
</tr>
<tr>
<td>4-9</td>
<td>Timer Register (TR) Format</td>
</tr>
<tr>
<td>4-10</td>
<td>Ring Alarm Register (RALR) Format</td>
</tr>
<tr>
<td>4-11</td>
<td>Pointer Register (PRn) Format</td>
</tr>
<tr>
<td>4-13</td>
<td>Procedure Pointer Register (PPR) Format</td>
</tr>
<tr>
<td>4-15</td>
<td>Temporary Pointer Register (TPR) Format</td>
</tr>
<tr>
<td>4-16</td>
<td>Descriptor Segment Base Register (DSBR, DBR) Format</td>
</tr>
<tr>
<td>4-18</td>
<td>Segment Descriptor Word Associative Memory (SDWAM) Format</td>
</tr>
<tr>
<td>4-21</td>
<td>Page Table Word Associative Memory (PTWAM) Format</td>
</tr>
<tr>
<td>4-23</td>
<td>Fault Register Format</td>
</tr>
<tr>
<td>4-25</td>
<td>Mode Register (MR) Format</td>
</tr>
<tr>
<td>4-28</td>
<td>Cache Mode Register (CMR) Format</td>
</tr>
<tr>
<td>4-32</td>
<td>Control Unit (CU) History Register Format</td>
</tr>
<tr>
<td>4-37</td>
<td>Operations Unit (OU) History Register Format</td>
</tr>
<tr>
<td>4-39</td>
<td>Appending Unit (AU) History Register Format</td>
</tr>
<tr>
<td>4-51</td>
<td>Configuration Switch Data Format</td>
</tr>
<tr>
<td>4-56</td>
<td>Control Unit Data Format</td>
</tr>
<tr>
<td>5-3</td>
<td>Decimal Unit Data Format</td>
</tr>
<tr>
<td>5-6</td>
<td>Examples of Page Number Formation</td>
</tr>
<tr>
<td>5-6</td>
<td>Final Address Generation for an Unpaged Segment</td>
</tr>
<tr>
<td>5-9</td>
<td>Paged Segment Generation for an Operation Flowchart</td>
</tr>
<tr>
<td>5-10</td>
<td>Segment Descriptor Word (SDW) Format</td>
</tr>
<tr>
<td>5-11</td>
<td>Page Table Word (PTW) Format</td>
</tr>
<tr>
<td>5-2</td>
<td>Address Modifier (TAG) Field Format</td>
</tr>
<tr>
<td>6-2</td>
<td>Common Effective Address Formation Flowchart</td>
</tr>
<tr>
<td>6-4</td>
<td>Register Modification Flowchart</td>
</tr>
<tr>
<td>6-7</td>
<td>Register Then Indirect Modification Flowchart</td>
</tr>
<tr>
<td>6-9</td>
<td>Indirect Then Register Modification Flowchart</td>
</tr>
<tr>
<td>6-15</td>
<td>Indirect Then Tally Modification Flowchart</td>
</tr>
<tr>
<td>6-16</td>
<td>Format of Instruction Word ADDRESS When Bit 29 = 1</td>
</tr>
<tr>
<td>6-18</td>
<td>ITP Pointer Pair Format</td>
</tr>
<tr>
<td>6-19</td>
<td>ITS Pointer Pair Format</td>
</tr>
</tbody>
</table>

**Figure 6-10** Effective Segment Number Generation Flowchart

**Figure 6-11** EIS Effective Address Formation Flowchart

**Figure 8-1** Complete Appending Unit Operation Flowchart

**Figure 9-1** Cache Store/Main Store Mapping
### CONTENTS (Cont)

#### TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2-1</td>
<td>R-type Modifiers for REG Fields.</td>
<td>2-10</td>
</tr>
<tr>
<td>Table 2-2</td>
<td>Alphanumeric Character Number (CN) Codes.</td>
<td>2-13</td>
</tr>
<tr>
<td>Table 2-3</td>
<td>Alphanumeric Data Type (TA) Codes.</td>
<td>2-13</td>
</tr>
<tr>
<td>Table 2-4</td>
<td>Sign and Decimal Type (S) Codes.</td>
<td>2-14</td>
</tr>
<tr>
<td>Table 2-5</td>
<td>Relation Between Data Bits and Indicators.</td>
<td>2-21</td>
</tr>
<tr>
<td>Table 2-6</td>
<td>Control Relations for Store Character Instructions (Nine Bit).</td>
<td>2-28</td>
</tr>
<tr>
<td>Table 2-7</td>
<td>Control Relations for Store Character Instructions (Six Bit).</td>
<td>2-30</td>
</tr>
<tr>
<td>Table 2-8</td>
<td>Default Edit Insertion Table Characters.</td>
<td>2-241</td>
</tr>
<tr>
<td>Table 2-9</td>
<td>Micro Operation Code Assignment Map.</td>
<td>2-251</td>
</tr>
<tr>
<td>Table 3-1</td>
<td>Fixed Point Binary Integer Values.</td>
<td>3-6</td>
</tr>
<tr>
<td>Table 3-2</td>
<td>Fixed Point Binary Fraction Values.</td>
<td>3-7</td>
</tr>
<tr>
<td>Table 3-3</td>
<td>Floating Point Binary Operand Values.</td>
<td>3-10</td>
</tr>
<tr>
<td>Table 3-4</td>
<td>Decimal Sign Character Interpretation.</td>
<td>3-12</td>
</tr>
<tr>
<td>Table 3-5</td>
<td>Decimal Data Values.</td>
<td>3-13</td>
</tr>
<tr>
<td>Table 3-6</td>
<td>Character String Data Length Limits.</td>
<td>3-14</td>
</tr>
<tr>
<td>Table 4-1</td>
<td>Processor Registers.</td>
<td>4-1</td>
</tr>
<tr>
<td>Table 4-2</td>
<td>System Controller Illegal Action Codes.</td>
<td>4-25</td>
</tr>
<tr>
<td>Table 5-1</td>
<td>Appending Unit Cycle Definitions.</td>
<td>5-8</td>
</tr>
<tr>
<td>Table 6-1</td>
<td>General Offset Modification Types.</td>
<td>6-3</td>
</tr>
<tr>
<td>Table 6-2</td>
<td>Register Modification Decode.</td>
<td>6-5</td>
</tr>
<tr>
<td>Table 6-3</td>
<td>Variations of Indirect Then Tally Modification.</td>
<td>6-10</td>
</tr>
<tr>
<td>Table 6-4</td>
<td>Special Append Mode Address Modifiers.</td>
<td>6-17</td>
</tr>
<tr>
<td>Table 7-1</td>
<td>List of Faults.</td>
<td>7-3</td>
</tr>
</tbody>
</table>
SECTION I

INTRODUCTION TO PROCESSOR

The Processor described in this reference manual is a hardware module designed for use with the MUltiplied Information and Computing Service (Multics). The many distinctive features and functions of Multics are enhanced by the powerful hardware features of the Processor. The addressing features, in particular, are designed to permit the Multics software to compute relative and absolute addresses, locate data and programs in different devices, and retrieve such data and program as necessary.

MULTICS "PROCESSOR" FEATURES

The Multics Processor contains the following general features:

1. Storage protection to place access restrictions on specified segments.

2. Capability to interrupt a process in execution in response to an external signal (e.g., I/O termination) at the end of any even/odd instruction pair (mid-instruction interrupts are permitted for some instructions), to save Processor status, and to restore the status at a later time without loss of continuity of the process.

3. Capability to fetch instruction pairs and to buffer two instructions (up to four instructions, depending on certain main store overlap conditions) including the one currently in execution.

4. Overlapping "instruction-execution", address preparation, and instruction fetch. While an instruction is being executed, address preparation for the next operand (or even the operand following it) or the next instruction pair is taking place. The operations unit can be executing instruction N; instruction N+1 can be buffered in the operations unit (with its operand buffered in a main store port); and the control unit can be executing instructions N+2 or N+3 (if such execution does not involve the main store port or registers of instructions N or N+1), or preparing the address to fetch instructions N+4 and N+5.

5. Capability to detect main store instructions that alter the contents of buffered instructions. Ability to delay preprocessing of an address using register modification if the instruction currently in execution changes the register to be used in that modification.

6. Interlacing capability to direct main store accesses to the proper system controller module.

7. Intermediate storage of address and control information in high-speed registers addressable by content (Associative Memory).
8. Intermediate storage of base address and control information in pointer registers which are loaded by the executing program.

9. Absolute address computation at execution time.

**SPECIAL CAPABILITIES**

The Processor also includes several unique capabilities, such as hardware implemented segmentation and paging, address modification, address appending, and detection of faults and external interrupts. These features are summarized in this section and described in detail respectively in Sections V, VI, and VII.

"Segmentation" and "Paging"

A segment is a collection of data or instructions that is assigned a symbolic name and addressed symbolically by the user. Paging is at the discretion of the software; the user need not be aware of the existence of pages. User visible address preparation is concerned with the calculation of a segment effective address relative to the origin of the segment; the Processor hardware completes address preparation by translating the final segment effective address into an absolute main store address. The user may view each of his segments as residing in an independent main store unit. Each segment has its own origin which can be addressed as location zero. The size of each segment varies without affecting the addressing of the other segments. Each segment can be addressed like a conventional main store image starting at location zero. Maximum "segment-size" is 262,144 words.

When viewed from the Processor, main store consists of blocks or pages, each of which is defined as "page-size" words in length. (The page size used by Multics is 1024 words.) Each page begins at an absolute address which is zero modulo the page size. Any page of a segment can be placed in any available main store block. These pages may be addressed as if they were contiguous even though they are in widely scattered absolute locations. Only currently referenced pages need be in main store. If a segment is not paged, the complete segment is located in contiguous blocks of main store. In the current Multics implementation, all user segments are paged.

**Address Modification and Address Appending**

Prior to each main store access, two major phases of address preparation take place:

1. "Address-modification" by Register or Indirect Word content, if specified by the Instruction Word or Indirect Word.

2. "Address-appending", in which a segment effective address is translated into an absolute address to access main store.

Although the above two types of modification are combined in most operations, they are described separately in Sections V and VI. The address modification procedure can go on indefinitely, with one type of modification leading to repetitions of the same type or to other types of modification prior
to a main store access for an operand. However, to simplify the descriptions in this manual, each type of address modification is described as if it were the first (and usually the only) modification prior to a main store access.

**Faults and Interrupts**

The Processor detects certain illegal procedures: faulty communication with the main store; programmed faults; certain external events; and arithmetic faults. Many of the Processor fault conditions are deliberately or inadvertently caused by the software and do not necessarily involve error conditions.

Similarly, the Processor communicates with the other system modules by setting and answering external interrupts. When a fault or interrupt is recognized, a trap results. This causes the forced execution of a pair of instructions in a main store location, unique to the fault or interrupt vector. The first of the forced instructions may cause safe storage of the Processor status. The second instruction in a fault vector should be a transfer, or the faulting program will be resumed without the fault having been processed. "Faults" and "interrupts" are described in Section VII.

Interrupts and certain low priority faults are recognized only at specific times during the execution of an instruction pair. If, at these times, the Processor detects the presence of bit 28 in the Instruction Word, the trap is inhibited and program execution continues. The interrupt or fault signal is saved for future recognition and is reset only when the trap occurs.

**PROCESSOR MISTES OF OPERATION**

There are three "modes of main store addressing" (Absolute Mode, Append Mode, and BAR Mode), and two modes of instruction execution (Normal Mode and Privileged Mode). These modes of operation and the functions performed are summarized in Table 1-1.

**Instruction Modes**

**NORMAL MODE**

Most instructions can be executed in the "Normal" Mode of operation. Certain instructions, classed as privileged, cannot be executed in Normal Mode. These are identified in the individual instruction descriptions. An attempt to execute privileged instructions while in the Normal Mode results in an Illegal Procedure Fault. In the Normal Mode, various restrictions are indicated in Segment Descriptor Words and Page Table Words, which are explained in Section V. Address Preparation uses the appending phase. The Processor executes in Normal Mode when the access bits of the Segment Descriptor Word specify a nonprivileged procedure.
PRIVILEGED MODE

In Privileged Mode, all instructions can be executed. Address Preparation uses the appending phase. The Processor executes in "Privileged" Mode when the access bits of the Segment Descriptor Word specify a privileged procedure and the execution ring is equal to zero. Refer to Sections V and VIII for more detailed information.

Addressing Modes

ABSOLUTE MODE

All instructions can be executed in the "Absolute" Mode and unrestricted access is permitted to privileged hardware features. Address Preparation for instruction fetches does not use the appending phase. During instruction fetches, the Procedure Pointer Register is ignored.

The Processor enters Absolute Mode immediately after a fault or interrupt and remains in Absolute Mode until it executes a transfer instruction whose operand is obtained via explicit use of the appending mechanism, that is, via explicit reference to one of the Pointer Register by the use of bit 29 of the Instruction Word (See Append Mode below).

APPEND MODE

The "Append" Mode is the most commonly used main store addressing mode. In this mode, the final effective segment address is either added to the Procedure Pointer Register, or it is added to one of the eight Pointer Registers. If bit 29 of the Instruction Word contains a 0, then the Procedure Pointer Register is selected; otherwise, the Pointer Register given by bits 0-2 of the instruction word is selected.

BAR MODE

In "BAR" (Base-Address-Register) Mode, the 18-bit BAR is used. The BAR contains a 0 modulo 512 address bound in bit positions 9-17 and a 0 modulo 512 base address in bit positions 0-8. All addresses are relocated by adding the effective segment address to the base address in bits 0-8. The relocated address then becomes the final segment effective address as in Append Mode and is added to the Procedure Pointer Register. A process is kept within certain main store limits by subtracting the unrelocated effective address from the address bound in bits 9-17. If the result is zero or negative, the relocated address would be out of range, and a Store Fault occurs.
Table 1-1. Modes of Operation

<table>
<thead>
<tr>
<th>FUNCTIONS</th>
<th>NORMAL</th>
<th>PRIVILEGED</th>
<th>ABSOLUTE</th>
<th>BAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute privileged instructions.</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Main store address for instruction fetch.</td>
<td>Append</td>
<td>Append</td>
<td>Absolute</td>
<td>Procedure Pointer Register plus BAR base address.</td>
</tr>
<tr>
<td>Main store address for operand fetch.</td>
<td>Append</td>
<td>Append</td>
<td>Append if bit 29 = 1, else Absolute.</td>
<td>Procedure Pointer Register plus BAR base address.</td>
</tr>
<tr>
<td>Restriction of access to other segments.</td>
<td>Some</td>
<td>Some</td>
<td>None</td>
<td>Total</td>
</tr>
</tbody>
</table>

PROCESSOR UNIT FUNCTIONS

Major functions of each principal logic element are listed below and are described in subsequent sections of this manual.

"Appending" Unit

Controls data input/output to main store.
Performs main store selection and interface.
Does address appending.
Controls fault recognition.

"Associative" Memory Assembly

This assembly consists of sixteen 72-bit Page Table Word Associative Memory (PTWAM)-registers and sixteen 108-bit Segment Descriptor Word Associative Memory (SDWAM)-registers. These registers are used to hold pointers to most recently used segments (SDWs) and pages (PTWs). This unit oviates the need for possible multiple main store accesses before obtaining an absolute main store address of an operand.
"Control Unit"

Performs all Processor control functions.
Performs address modification.
Controls mode of operation (Privileged, Normal, etc.).
Performs interrupt recognition.
Decodes Instruction Words and Indirect Words.
Performs Timer Register loading and decrementing.

"Operation Unit"

Does fixed and floating binary arithmetic.
Does shifting and Boolean operations.

"Decimal Unit"

Does decimal arithmetic.
Does character- and bit-string operations.
SECTION II

MACHINE INSTRUCTIONS

This section describes the comprehensive set of "machine-instructions" for the Multics Processor. The presentation assumes that the reader is familiar with the general structure of the Processor, the representation of information, the data formats, and the method of address preparation. Additional information on these subjects appears near the beginning of this section and in Sections III through VI.

INSTRUCTION REPERTOIRE

The Processor interprets a 10 bit field of the Instruction Word as the Operation Code. This field size yields an instruction universe of 1024 of which 547 are implemented. The instruction population is divided into 456 Basic Operations and 91 Extended Instruction Set (EIS) Operations.

Arrangement of Instructions

Instructions in this section are presented alphabetically by their mnemonic codes within functional categories. However, an overall alphabetic listing of instruction codes and their names appears in Appendix 3 to aid the user in locating specific instructions via that code.

Basic Operations

The 456 "basic" operations in the Processor all require exactly one 36-bit machine word and are further subdivided into the following types:

- Fixed Point Binary Arithmetic
- Boolean Operations
- Floating Point Binary Arithmetic
- Transfer of Control
- Pointer Register
- Miscellaneous
- Privileged

Extended Instruction Set (EIS) Operations

The 91 "Extended Instruction Set" (EIS) Operations are further subdivided into 62 EIS Single-Word Instructions and 29 EIS Multi-Word Instructions.
EIS SINGLE-WORD OPERATIONS

The 62 "EIS-Single-Word" Instructions load, store, and perform special arithmetic on the Address Registers (ARn) used to access bit- and character-string operands, and safe-store Decimal Unit (DU) control information required to service a Processor fault. Like the Basic Operations, EIS Single-Word Instructions require exactly one 36-bit Machine Word.

EIS MULTI-WORD OPERATIONS

The 29 "EIS-Multi-Word" Instructions perform Decimal Arithmetic and bit- and character-string operations. They require 3 or 4 36-bit Machine Words depending on individual Operand Descriptor requirements.

"FORMAT" OF INSTRUCTION DESCRIPTION

Each instruction in the repertoire is described in the following pages of this section. The descriptions are presented in the format shown below.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>INSTRUCTION NAME</th>
<th>OP CODE (OCTAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;FORMAT&quot;</td>
<td>Figure or Figure reference</td>
<td></td>
</tr>
<tr>
<td>SUMMARY</td>
<td>Text and/or bit transfer equations</td>
<td></td>
</tr>
<tr>
<td>MODIFICATIONS</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>INDICATORS</td>
<td>Text and/or logic statements</td>
<td></td>
</tr>
<tr>
<td>NOTES</td>
<td>Text</td>
<td></td>
</tr>
</tbody>
</table>

Line 1: MNEMONIC, INSTRUCTION NAME, OP CODE (OCTAL)

This line has three parts that contain the following:

1. Mnemonic -- The "mnemonic-coded" for the Operation field of the assembler statement. The Multics assembler, ALH, recognizes this value and maps it into the appropriate binary pattern when generating the actual object code.

2. Instruction Name -- The name of the machine instruction from which the Mnemonic was derived.

3. Op Code (Octal) -- The octal value of the operation code for the instruction. A zero or a one in parentheses following an octal code indicates whether bit 27 (Op Code extension bit) of the instruction
word is OFF or ON.

Line 2: FORMAT

The layout and definition of the subfields of the instruction word or words is given here either as a Figure or as a reference to a Figure.

Line 3: SUMMARY

The change in the state of the processor affected by the execution of the instruction is described in a short and generally symbolic form. If reference is made to the state of an indicator in the SUMMARY, it is the state of the indicator before the instruction is executed.

Line 4: MODIFICATIONS

Those modifiers that cannot be used with the instruction are listed explicitly as exceptions either because they are not permitted or because their effect cannot be predicted from the general address modification procedure. (See "Effective Address Formation" in Section VI.)

Line 5: INDICATORS

Only those indicators are listed whose state can be changed by the execution of the instruction. In most cases, a condition for setting ON as well as one for setting OFF is stated. If only one of the two is stated, then the indicator remains unchanged if the condition is not met. Unless stated otherwise, the conditions refer to the contents of registers existing after instruction execution. Refer also to "Common Attributes of Instructions", later in this section.

Line 6: NOTES

This part of the description exists only in those cases where the SUMMARY is not sufficient for in depth understanding of the operation.
DEFINITIONS OF "NOTATION" AND "SYMBOLS"

"Main-Store Addresses"

Y = the 18 low order bits of the final 24 bit main store address of the instruction operand after all address preparation is complete.

Y-pair = a symbol denoting that Y designates a pair of main store locations with successive addresses, the smaller address being even. When the main store address is even, it designates the pair Y(even), Y+1; and when it is odd, the pair Y-1, Y+odd. The main store location with the smaller (even) address contains the most significant part of a double-word operand or the first of a pair of instructions.

Y-blockn = a symbol denoting that Y designates a block of main store locations of 4-, 8-, or 16-word extent. For a block of n-word extent, the Processor assumes that Y-blockn is a 0 modulo n address and performs address incrementing through the block accordingly, stopping when the address next reaches a value 0 modulo n. Note the difference between Y-block addressing and Y-pair addressing that forces the address to be 0 modulo 2.

Y-charnk = a symbol denoting that Y designates a character or string of characters in main store of character size n bits as described by the kth Operand Descriptor. n is specified by the data type field of Operand Descriptor k and may have values 4, 8, or 9. See Section VI, Effective Address Formation, for details of Operand Descriptors.

Y-bitk = a symbol denoting that Y designates a bit or string of bits in main store as described by the kth Operand Descriptor. See Section VI, Effective Address Formation, for details of Operand Descriptors.

"Index Values"

When reference is made to the elements of a string of characters or bits in main store, the notation shown in Register Position and Contents below is used. The index used to show traversing a string of extent n may take any of the values in the interval (1, 0) unless noted otherwise. The elements of a main store block are traversed explicitly by using the index as an addend to the given block address, e.g., Y-block8+m and Y-block4#2m+1.

"Abbreviations and Symbols"

A Accumulator Register
ARN Address Register n (n = 0, 1, 2, ..., 7)
\( \text{consists of: PRn.WORDNO1:PRn.CHARI:PRn.BITNO} \)
Aq Combined Accumulator-Quotient Register
BAR Base Address Register

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975 2-4 AL39
Register Positions and Contents

("R" standing for any of the registers listed above as well as for main store words, word-pairs, word-blocks, and character strings.)

Ri the ith bit position of R
R(i) the ith register of a set of n registers, R
Ri, j the bit positions i through j of R
C(R) the contents of the full register R

C(R)i the contents of the ith bit or character of R
C(R)i, j the contents of the bits or characters i through j of R
xx...x a string of binary bits (0's or 1's) of any necessary length
When the description of an instruction specifies a change for a part of a register or main store location, it is understood that the part of the register or main store location not mentioned remains unchanged.

**Other Symbols**

- `->` replaces
- `!!` compare with
- `&` the Boolean connective AND
- `|` the Boolean connective OR
- `@` the Boolean connective NON-EQUIVALENCE (or EXCLUSIVE OR)
- `-` the Boolean unary NOT operator
- `#` not equal

`n**m` indicates exponentiation (n and m are integers); for example, the fifth power of 2 is represented as `2**5`.

`x` multiplication; for example, `C(Y)` times `C(Q)` is represented as `C(Y) * C(Q)`.

`/` division; for example, `C(Y)` divided by `C(A)` is represented as `C(Y) / C(A)`.

`""` concatenation; for example, `string1 : string2`.

`1...1` the absolute value of the value between vertical bars (no algebraic sign). For example, the absolute value of `C(A)` plus `C(Y)` is represented as `abs(C(A) + C(Y))`.

**COMMON ATTRIBUTES OF INSTRUCTIONS**

**Illegal Modification**

If an "illegal-modifier" is used with any instruction, an Illegal Procedure Fault with a subcode class of Illegal Modifier occurs.

**Parity Indicator**

The Parity Indicator is turned ON at the end of a main store access which has incorrect parity.
"INSTRUCTION-WORD-FORMATS"

Basic and EIS Single-Word Instructions

The "Basic" Instructions and "EIS-Single-Word" Instructions require exactly one 36 bit Machine Word and are interpreted according to the format shown in Figure 2-1 below.

The given address of the Operand or Indirect Word. This address may be:

- An 18 bit main store address if $A = 0$ (Absolute Mode only)
- An 18 bit offset to the Base Address Register if $A = 0$ (BAR Mode only)
- An 18 bit offset relative to the base of the current procedure segment if $A = 0$ (Appending Mode only)
- A 3 bit Pointer Register number ($p$) and a 15 bit offset to $C(PRN, WORDND)$ if $A = 1$ (Absolute and Appending Modes only)
- A 3 bit Address Register number ($q$) and a 15 bit offset to $C(ARN)$ if $A = 1$ (All modes depending on instruction type)
- An 18 bit literal signed or unsigned constant (All modes depending on instruction type and Modifier)
- An 8 bit Shift Operation count (All modes)
- An 18 offset to the current value of the Instruction Counter $C(PRR, IC)$ (All modes)

Instruction operation code.

Program Interrupt inhibit bit. When this bit is set, the Processor will ignore all external Program Interrupt signals. See Section VII, Faults and Interrupts, for details.

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>OPCODE</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 2 2 2 3 3 7 8</td>
<td>II A I TAG I</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

Figure 2-1 Basic and EIS Single-Word Instruction Format
Indirect Words

Certain of the Basic and EIS Single-Word Instructions permit indirection to be specified as part of Address Modification. When such indirection is specified, CY is interpreted as an "Indirect-Word" according to the format shown in Figure 2-2 below.

0 1 1 2 3 3
0 7 8 9 0 5
1 1 ADDRESS 1 TALLY 1 TAG 1
1 18 12 6

Figure 2-2 Indirect Word Format

ADDRESS The given address of the Operand or next Indirect Word. This address may be:

An 18 bit main store address if A = 0 in the Instruction Word (Absolute Mode only)

An 18 bit offset relative to the Base Address Register (BAR) if A = 0 in the Instruction Word (BAR Mode only)

An 18 bit offset relative to the base of the current procedure segment if A = 0 (Appending Mode only)

An 18 bit offset relative to the origin of the segment described by PR if A = 1 in the Instruction Word and PR is selected by the Instruction Word (Absolute and Appending Modes only)

TALLY A count field for use by those Address Modifiers that involve tallying.

TAG Next address modifier.
EIS Multi-Word Instructions

The "EIS" Multi-Word Instructions require 3 or 4 Machine Words depending on the Operand Descriptor requirements of the individual instructions. The words are interpreted according to the format shown in Figure 2-3 below.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>V A R I A B L E</td>
<td>O P C O D E</td>
<td>I I</td>
<td>M F 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operand Descriptor or Indirect Pointer for Operand 1
Operand Descriptor or Indirect Pointer for Operand 2
Operand Descriptor or Indirect Pointer for Operand 3

Figure 2-3 EIS Multi-Word Instruction Format

"VARIABLE" This field is interpreted variously according to the requirements of the individual EIS Instructions. Its interpretation is given under FORMAT for each EIS Instruction. The Modification Fields MF2 and MF3 are contained in this field if they are required.

OPCODE Instruction operation code as for Basic and EIS Single-Word Instructions.

I Program Interrupt inhibit bit as for Basic and EIS Single-Word Instructions.

MF1 Modification Field for Operand Descriptor 1. See EIS Modification Fields (MF) below for details.

EIS Modification Fields (MF)

Each of the Operand Descriptors following an EIS Multi-Word Instruction Word has a "Modification" Field in the Instruction Word. The Modification Field controls the interpretation of the Operand Descriptor. The Modification Field is interpreted according to the format shown in Figure 2-4.
Figure 2-4 EIS Modification Field (MF) Format

**key**

a AR
Address Register flag. This flag controls interpretation of the ADDRESS field of the Operand Descriptor just as the "A" flag controls interpretation of the ADDRESS field of the Basic and EIS Single-Word Instructions.

b RL
Register length control. If RL = 0, then the Length (N) field of the Operand Descriptor contains the length of the operand. If RL = 1, then the Length (N) field of the Operand Descriptor contains a selector value specifying a register holding the operand length.

c ID
Indirect descriptor control. If ID = 1 for MFk, then the kth word following the Instruction Word is an Indirect Pointer to the Operand Descriptor for the kth operand; otherwise, that word is the Operand Descriptor.

REG
The register number for R-type modification (if any) of ADDRESS of the Operand Descriptor. These modifications are similar to R-type modifications for Basic Instructions and are summarized in Table 2-1 below. Illegal modifiers have the entry "IPR" and cause an Illegal Procedure Fault.

---

**Table 2-1 R-type Modifiers for REG Fields**

<table>
<thead>
<tr>
<th>Octal Code</th>
<th>R-type</th>
<th>MF,REG</th>
<th>Indirect Operand</th>
<th>C(Operand Descriptor)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>IPR</td>
</tr>
<tr>
<td>01</td>
<td>AU</td>
<td>AU</td>
<td>AU</td>
<td>IPR</td>
</tr>
<tr>
<td>02</td>
<td>QU</td>
<td>QU</td>
<td>QU</td>
<td>IPR</td>
</tr>
<tr>
<td>03</td>
<td>DU</td>
<td>IPR(a)</td>
<td>IPR</td>
<td>IPR</td>
</tr>
<tr>
<td>04</td>
<td>IC</td>
<td>IC(b)</td>
<td>IC(b)</td>
<td>IPR</td>
</tr>
<tr>
<td>05</td>
<td>AL</td>
<td>A(c)</td>
<td>AL</td>
<td>A(c)</td>
</tr>
<tr>
<td>06</td>
<td>QL</td>
<td>Q(c)</td>
<td>QL</td>
<td>Q(c)</td>
</tr>
<tr>
<td>07</td>
<td>DL</td>
<td>IPR</td>
<td>IPR</td>
<td>IPR</td>
</tr>
</tbody>
</table>

| 10         | X0     | X0     | X0                | X0                    |
| 11         | X1     | X1     | X1                | X1                    |
| 12         | X2     | X2     | X2                | X2                    |
| 13         | X3     | X3     | X3                | X3                    |
| 14         | X4     | X4     | X4                | X4                    |
| 15         | X5     | X5     | X5                | X5                    |
| 16         | X6     | X6     | X6                | X6                    |
| 17         | X7     | X7     | X7                | X7                    |
(a) The DU modifier is permitted only in the second Operand Descriptor of the SCD, SCDR, SCM, and SCM instructions to specify that the test character(s) reside(s) in bits 0-18 of the Operand Descriptor.

(b) The IC modifier is permitted only in the REG field of Indirect Pointers and in HF3.REG for the SCD, SCDR, SCM, SCM, HVT, TCI, and TCTR instructions, that is, the instructions that store summary results of a scan operation. C(IC) is always interpreted as a word offset.

(c) The limit of addressing extent of the processor is $2^{18} - 1$ words; that is, given any main store address, Y, a modifier may be employed to access a main store word anywhere in the range (Y - $2^{18} + 1$, Y + $2^{18} - 1$), provided other address range constraints are not violated. Since it is desirable to address this same extent as words, characters, and bits it is necessary to provide a register with range greater than the 12 bits of N or the 18 bits of normal R-type modifiers. This is done by extending the range of the A and Q modifiers as follows...

<table>
<thead>
<tr>
<th>Mode</th>
<th>Range</th>
<th>A+Q bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>9-bit</td>
<td>20</td>
<td>16,35</td>
</tr>
<tr>
<td>6-bit</td>
<td>21</td>
<td>15,35</td>
</tr>
<tr>
<td>4-bit</td>
<td>21</td>
<td>15,35</td>
</tr>
<tr>
<td>bit</td>
<td>24</td>
<td>12,35</td>
</tr>
</tbody>
</table>

The unused high order bits are ignored.

---

**EIS Operand Descriptors and Indirect Pointers**

The words following an EIS Multi-Word Instruction Word are either descriptions of the operands or "Indirect" Pointers to the operand descriptions. The interpretation of the words is performed according to the settings of the control bits in the associated Modification Field (MF). The kth Word following the Instruction Word is interpreted according to the contents of MFk. See EIS Modifications Fields (MF) above for meaning of the various control bits.

See Section III, Data Representation, and Section VI, Effective Address Formation, for further details.

---

**"OPERAND"DESCRIPTOR"INDIRECT"POINTER"FORMAT**

If MFk.ID = 1, then the kth word following an EIS multi-word Instruction Word is not an Operand Descriptor, but is an Indirect Pointer to an Operand Descriptor and is interpreted as shown in Figure 2-5.
ADDRESS  The given address of the Operand Descriptor. This address may be:

- An 18 bit main store address if \( A = 0 \) (Absolute Mode only)
- An 18 bit offset relative to the Base Address Register (BAR) if \( A = 0 \) (BAR Mode only)
- An 18 bit offset relative to the base of the current procedure segment if \( A = 0 \) (Appending Mode only)
- A 3 bit Pointer Register number \( p \) and a 15 bit offset relative to \( C(\text{PRn.WORDNO}) \) if \( A = 1 \) (All modes)

\( \text{A} \)

Indirect via Pointer Register flag. This flag controls interpretation of the ADDRESS field of the Indirect Pointer just as the "A" flag controls interpretation of the ADDRESS field of the Basic and EIS Single-Word Instructions.

\( \text{REG} \)

Address modifier for ADDRESS. All Register Modifiers except DU and DL may be used. If IC is used, then ADDRESS is an 18 bit offset to value of the Instruction Counter for the Instruction Word. \( C(\text{REG}) \) is always interpreted as a word offset to ADDRESS.

"ALPHANUMERIC-OPERAND-DESCRIPTOR-FORMAT"

For any operand of an EIS Multi-word Instruction that requires Alphanumeric Data, the Operand Descriptor is interpreted as shown in Figure 2-6 below.

ADDRESS  The given address of the operand. This address may be (for the \( k \)th operand):

- An 18 bit main store address if \( \text{MF} \cdot \text{AR} = 0 \) (Absolute Mode only)
An 18 bit offset to the Base Address Register if MFk,AR = 0 (BAR Mode only)

An 18 bit offset relative to the base of the current procedure segment if MFk,AR = 0 (Appending Mode only)

A 3 bit Address Register number (g) and a 15 bit word offset to C(ARn.) if MFk,AR = 1 (All modes)

Character Number. This field gives the character position within the word at ADDRESS of the first operand character. Its interpretation depends on the Data Type (see TA below) of the operand. Table 2-2 below shows the interpretation of the field. A digit in the table indicates the corresponding character position (See Section III, Data Representation, for data formats) and an "x" indicates an invalid code for the Data Type. Invalid codes cause Illegal Procedure Faults.

Table 2-2 Alphanumeric Character Number (CN) Codes

<table>
<thead>
<tr>
<th>CN(CN)</th>
<th>4-bit</th>
<th>5-bit</th>
<th>9-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
<td>3</td>
<td>x</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
<td>5</td>
<td>x</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>111</td>
<td>7</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Type Alphanumeric. This is the Data Type code for the operand. The interpretation of the field is shown in Table 2-3 below. The code shown as Invalid causes an Illegal Procedure Fault.

Table 2-3 Alphanumeric Data Type (TA) Codes

<table>
<thead>
<tr>
<th>CN(TA)</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>9-bit</td>
</tr>
<tr>
<td>01</td>
<td>6-bit</td>
</tr>
<tr>
<td>10</td>
<td>4-bit</td>
</tr>
<tr>
<td>11</td>
<td>Invalid</td>
</tr>
</tbody>
</table>

Operand length. If MFk,RL = 0, this field contains the string length of the operand. If MFk,RL = 1, this field contains the code for a register holding the operand string length. See Table 2-1 and EIS Modification Fields (MF) above for a discussion of register codes.
For any operand of an EIS Multi-word Instruction that requires Numeric Data, the Operand Descriptor is interpreted as shown in Figure 2-7 below.

Figure 2-7 Numeric Operand Descriptor Format

**Key**

ADDRESS: The given address of the operand. This address may be for the kth operand:

- An 18 bit main store address if $MFA=0$ (Absolute Mode only)
- An 18 bit offset to the Base Address Register if $MFA=0$ (BAR Mode only)
- An 18 bit offset relative to the base of the current procedure segment if $MFA=0$ (Appending Mode only)

CN: Character Number. This field gives the character position within the word at ADDRESS of the first operand character. Its interpretation depends on the Data Type (see Table 2 below) of the operand. Table 2-2 above shows the interpretation of the field.

a TN: Type Numeric. This is the Data Type code for the operand. The codes are:

<table>
<thead>
<tr>
<th>Octal Code</th>
<th>Sign and Decimal Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Floating point, leading sign</td>
</tr>
<tr>
<td>01</td>
<td>Scaled fixed point, leading sign</td>
</tr>
<tr>
<td>10</td>
<td>Scaled fixed point, trailing sign</td>
</tr>
<tr>
<td>11</td>
<td>Scaled fixed point, unsigned</td>
</tr>
</tbody>
</table>

S: Sign and decimal type of data. The interpretation of the field is shown in Table 2-4 below.

Table 2-4 Sign and Decimal Type (S) Codes

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975
SF Scaling factor. This field contains the two's complement value of the base 10 scaling factor; that is, the value of \( m \) for numbers represented as \( n \times 10^m \). The decimal point is assumed to the right of the least significant digit of \( n \). Negative values move the decimal point to the left; positive values, to the right. The range of \( m \) is \((-32,31)\).

\( N \) Operand length. If MFk\_RL = 0, this field contains the operand length in digits. If MFk\_RL = 1, it contains the REG code for the register holding the operand length and C(REG) is treated as a 0 modulo 64 number.

**BIT-STRING-OPERAND-DESCRIPTOR-FORMAT**

For any operand of an EIS Multi-word Instruction that requires Bit-string Data, the Operand Descriptor is interpreted as shown in Figure 2-8 below.

```
ADDRESS
0 1 1 2 2 2 3 3
0 7 8 9 0 3 4 5
1 1 1 1 1 1
1 ADDRESS 1 1 1 1 1
1 1 1 1 1
18 2 4 12
```

**Figure 2-8 Bit String Operand Descriptor Format**

**ADDRESS** The given address of the operand. This address may be (for the \( i \)th operand):
- An 18 bit main store address if MFk\_AR = 0 (Absolute Mode only)
- An 18 bit offset to the Base Address Register if MFk\_AR = 0 (BAR Mode only)
- An 18 bit offset relative to the base of the current procedure segment if MFk\_AR = 0 (Appending Mode only)
- A 3 bit Address Register number (C) and a 15 bit word offset to C(ARn) if MFk\_AR = 1 (All modes)

**C** The character number of the 9-bit character within ADDRESS containing the first bit of the operand.

**B** The bit number within the 9-bit character, C, of the first bit of the operand.
**FIXED POINT DATA MOVEMENT LOAD**

**"FIXED-POINT" ARITHMETIC INSTRUCTIONS**

**"Fixed-Point" Data Movement Load**

<table>
<thead>
<tr>
<th>EAA</th>
<th>Effective Address to A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>635 (I)</td>
</tr>
</tbody>
</table>

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
- \( Y \rightarrow C(A)0,17 \)
- \( 00...0 \rightarrow C(A)18,35 \)

**MODIFICATIONS:** All except DU, DL

**INDICATORS:** (Indicators not listed are not affected)
- **Zero**
  - If \( C(A) = 0 \), then ON; otherwise OFF
- **Negative**
  - If \( C(A) \) bit 0 = 1, then ON; otherwise OFF

**NOTES:**
- The EAA instruction, and the instructions EAQ and EAXn, facilitate interregister data movements; the data source is specified by the address modification, and the data destination by the operation code of the instruction.
- Attempted repetition with RPL causes an Illegal Procedure Fault.

---

<table>
<thead>
<tr>
<th>EAQ</th>
<th>Effective Address to Q</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>636 (0)</td>
</tr>
</tbody>
</table>

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
- \( Y \rightarrow C(Q)0,17 \)
- \( 00...0 \rightarrow C(Q)18,35 \)

**MODIFICATIONS:** All except DU, DL

**INDICATORS:** (Indicators not listed are not affected)
- **Zero**
  - If \( C(Q) = 0 \), then ON; otherwise OFF
- **Negative**
  - If \( C(Q) \) 0 = 1, then ON; otherwise OFF

**NOTES:**
- Attempted repetition with RPL causes an Illegal Procedure Fault.
FIXED POINT DATA MOVEMENT LOAD

EAXn Effective Address to Xn 62n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code
Y -> C(Xn)

MODIFICATIONS: All except DU, DL

INDICATORS: (Indicators not listed are not affected)

Zero If C(Xn) = 0, then ON; otherwise OFF
Negative If C(Xn)0 = 1, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

LCA Load Complement A 335 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If C(Y) 0, then -C(Y) -> C(A)
otherwise, 00...0 -> C(A)

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If C(A) = 0, then ON; otherwise OFF
Negative If C(A)0 = 1, then ON; otherwise OFF
Overflow If range of A is exceeded, then ON; otherwise OFF

NOTES: The LCA instruction changes the number to its negative (if
≠ 0) while moving it from Y to A. The operation is
executed by forming the two's complement of the string of
36 bits.
FIXED POINT DATA MOVEMENT LOAD

LCAQ  Load Complement AQ  337 (0)

**FORMAT:**
Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
If C(Y-pair) ≠ 0, then -C(Y-pair) -> C(AQ)
otherwise, 00...0 -> C(AQ)

**MODIFICATIONS:**
All except DU, DL, CI, SC, SCR

**INDICATORS:**
(Indicators not listed are not affected)
- **Zero:** If C(AQ) = 0, then ON; otherwise OFF
- **Negative:** If C(AQ) = 1, then ON; otherwise OFF
- **Overflow:** If range of AQ is exceeded, then ON; otherwise OFF

**NOTES:**
The LCAQ instruction changes the number to its negative (if ≠ 0) while moving it from Y-pair to AQ. The operation is executed by forming the two's complement of the string of 72 bits.

LCQ  Load Complement Q  336 (0)

**FORMAT:**
Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
If C(Y) ≠ 0, then -C(Y) -> C(Q)
otherwise, 00...0 -> C(Q)

**MODIFICATIONS:**
All

**INDICATORS:**
(Indicators not listed are not affected)
- **Zero:** If C(Q) = 0, then ON; otherwise OFF
- **Negative:** If C(Q) = 1, then ON; otherwise OFF
- **Overflow:** If range of Q is exceeded, then ON; otherwise OFF

**NOTES:**
The LCQ instruction changes the number to its negative (if ≠ 0) while moving it from Y to Q. The operation is executed by forming the two's complement of the string of 36 bits.
**LCXn**

Load Complement Xn

**FORMAT:**

Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

For n = 0, 1, ..., or 7 as determined by operation code

- If C(Y)0,17 ≠ 0, then -C(Y)0,17 -> C(Xn)
- otherwise, 00...0 -> C(Xn)

**MODIFICATIONS:**

All except CI, SC, SCR

**INDICATORS:**

(Indicators not listed are not affected)

- **Zero**
  - If C(Xn) = 0, then ON; otherwise OFF
- **Negative**
  - If C(Xn)0 = 1, then ON; otherwise OFF
- **Overflow**
  - If range of Xn is exceeded, then ON; otherwise OFF

**NOTES:**

The LCXn instruction changes the number to its negative (if ≠ 0) while moving it from Y0,17 to Xn. The operation is executed by forming the two's complement of the string of 18 bits.

Attempted repetition with RPL and with the same register given as target and modifier causes an Illegal Procedure Fault.

---

**LDA**

Load A

**FORMAT:**

Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

C(Y) -> C(A)

**MODIFICATIONS:**

All

**INDICATORS:**

(Indicators not listed are not affected)

- **Zero**
  - If C(A) = 0, then ON; otherwise OFF
- **Negative**
  - If C(A)0 = 1, then ON; otherwise OFF
### LDAC

**Load A and Clear**  
**Format:** Basic Instruction Format (See Figure 2-1).  
**Summary:** 
- \( C(Y) \rightarrow C(A) \)  
- \( 00\ldots0 \rightarrow C(Y) \)  
**Modifications:** All except DU, DL, CI, SC, SCR  
**Indicators:** (Indicators not listed are not affected)  
- **Zero**: If \( C(A) = 0 \), then ON; otherwise OFF  
- **Negative**: If \( C(A)0 = 1 \), then ON; otherwise OFF  

**Notes:** The LDAC instruction causes a special main store reference that performs the load and clear in one cycle. Thus, this instruction can be used in locking data.

### LDAQ

**Load AQ**  
**Format:** Basic Instruction Format (See Figure 2-1).  
**Summary:** \( C(Y\text{-pair}) \rightarrow C(AQ) \)  
**Modifications:** All except DU, DL, CI, SC, SCR  
**Indicators:** (Indicators not listed are not affected)  
- **Zero**: If \( C(AQ) = 0 \), then ON; otherwise OFF  
- **Negative**: If \( C(AQ)0 = 1 \), then ON; otherwise OFF  

### LDI

**Load Indicator Register**  
**Format:** Basic Instruction Format (See Figure 2-1).  
**Summary:** \( C(Y)18,31 \rightarrow C(IR) \)  
**Modifications:** All except CI, SC, SCR
FIXED POINT DATA MOVEMENT LOAD

INDICATORS:

(Indicators not listed are not affected)

Parity
If C(Y)27 = 1, and the Processor is in Absolute or
Privileged Mode, then ON; otherwise OFF. This indicator
is not affected in the Normal or BAR modes.

Mask
Not BAR
Mode

Hask
Cannot be changed by the LDI instruction

Not BAR
Mode

Multiword
If C(Y)30 = 1, and the Processor is in Absolute or
Instruction Privileged mode, then ON; otherwise OFF. This indicator
Fault
is not affected in Normal or BAR modes.

Absolute
Mode.

All Other
Indicators

If corresponding bit in C(Y) is 1, then ON; otherwise, OFF

NOTES:
The relation between C(Y)18,31 and the indicators is given
in Table 2-5 below.

The Tally Runout indicator reflects C(Y)25 regardless of
what address modification is performed on the LDI
Instruction for tally operations.

Attempted repetition with RPT, RPD, or RPL causes an
Illegal Procedure Fault.

Table 2-5. Relation Between Data Bits and Indicators

<table>
<thead>
<tr>
<th>Bit Position C(Y)</th>
<th>Indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>Zero</td>
</tr>
<tr>
<td>19</td>
<td>Negative</td>
</tr>
<tr>
<td>20</td>
<td>Carry</td>
</tr>
<tr>
<td>21</td>
<td>Overflow</td>
</tr>
<tr>
<td>22</td>
<td>Exponent Overflow</td>
</tr>
<tr>
<td>23</td>
<td>Exponent UndeFlow</td>
</tr>
<tr>
<td>24</td>
<td>Overflow Mask</td>
</tr>
<tr>
<td>25</td>
<td>Tally Runout</td>
</tr>
<tr>
<td>26</td>
<td>Parity Error</td>
</tr>
<tr>
<td>27</td>
<td>Parity Mask</td>
</tr>
<tr>
<td>28</td>
<td>Not BAR Mode</td>
</tr>
<tr>
<td>29</td>
<td>Truncation</td>
</tr>
<tr>
<td>30</td>
<td>Multiword Instruction Fault (MIF)</td>
</tr>
<tr>
<td>31</td>
<td>Absolute Mode</td>
</tr>
</tbody>
</table>
### Fixed Point Data Movement Load

**LDQ**  
Load Q  

**FORMAT:** Basic Instruction Format (See Figure 2-1).  

**SUMMARY:**  
C(Y) -> C(Q)  

**MODIFICATIONS:** All  

**INDICATORS:** (Indicators not listed are not affected)  
- **Zero:** If C(Q) = 0, then ON; otherwise OFF  
- **Negative:** If C(Q)0 = 1, then ON; otherwise OFF  

**LDQC**  
Load Q and Clear  

**FORMAT:** Basic Instruction Format (See Figure 2-1).  

**SUMMARY:**  
C(Y) -> C(Q)  
00...0 -> C(Y)  

**MODIFICATIONS:** All except DU, DL, CI, SC, SCR  

**INDICATORS:** (Indicators not listed are not affected)  
- **Zero:** If C(Y) = 0, then ON; otherwise OFF  
- **Negative:** If C(Y)0 = 1, then ON, otherwise OFF  

**NOTES:** The LDQC instruction causes a special main store reference that performs the load and clear in one cycle. Thus, this instruction can be used in locking data.

**LDXn**  
Load Index Register Xn  

**FORMAT:** Basic Instruction Format (See Figure 2-1).  

**SUMMARY:** For n = 0, 1, ..., or 7 as determined by operation code  
C(Y)0,17 -> C(Xn)  

**MODIFICATIONS:** All except CI, SC, SCR
FIXED POINT DATA MOVEMENT LOAD

**INDICATORS:**
- **Zero:** If \( C(X_n) = 0 \), then ON; otherwise OFF
- **Negative:** If \( C(X_n) = 1 \), then ON; otherwise OFF

**NOTES:** Attempted repetition with RPL and with the same register given as target and modifier causes an Illegal Procedure Fault.

**LREG**
Load Registers

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
- \( C(Y+0,17) \rightarrow C(X_0) \)
- \( C(Y+1,17) \rightarrow C(X_1) \)
- \( C(Y+2,17) \rightarrow C(X_2) \)
- \( C(Y+3,17) \rightarrow C(X_3) \)
- \( C(Y+4) \rightarrow C(A) \)
- \( C(Y+5) \rightarrow C(Q) \)
- \( C(Y+6,0,7) \rightarrow C(E) \)

Where \( Y \) must be \( 0 \) modulo 8; otherwise, the next smaller such address is used.

**MODIFICATIONS:** All except DU, DL, CI, SC, SCR

**INDICATORS:** None affected

**NOTES:** Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

**LXLn**
Load \( X_n \) from Lower

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** For \( n = 0, 1, \ldots, \), or 7 as determined by operation code
- \( C(Y+18,35) \rightarrow C(X_n) \)

**MODIFICATIONS:** All except CI, SC, SCR
FIXED POINT DATA MOVEMENT LOAD

INDICATORS: (Indicators not listed are not affected)

Zero  If \( C(Xn) = 0 \), then ON; otherwise OFF
Negative  If \( C(Xn) \neq 1 \), then ON; otherwise OFF

NOTES: Attempted repetition with RPL and with the same register given as target and modifier causes an Illegal Procedure Fault.
**FIXED POINT DATA MOVEMENT STORE**

**"Fixed-Point-Data-Movement"Store**

**SBAR**
Store Base Address Register

**FORMAT**
Basic Instruction Format (See Figure 2-1).

**SUMMARY**
C(BAR) -> C(Y)0,17

**MODIFICATIONS**
All except DU, DL, CI, SC, SCR

**INDICATORS**
None affected

**NOTES**
Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

**SREG**
Store Registers

**FORMAT**
Basic Instruction Format (See Figure 2-1).

**SUMMARY**
C(X0) -> C(Y)0,17
C(X2) -> C(Y+1)0,17
C(X4) -> C(Y+2)0,17
C(X6) -> C(Y+3)0,17
C(A) -> C(Y+4)
C(E) -> C(Y+6)0,7
C(TR) -> C(Y+7)0,26
C(RALR) -> C(Y+7)33,35

where Y must be a 0 modulo 8 address; otherwise the next lower such address is used.

**MODIFICATIONS**
All except DU, DL, CI, SC, SCR

**INDICATORS**
None affected

**NOTES**
Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

**REVIEW DRAFT**
**SUBJECT TO CHANGE**
October, 1975

2-25 AL39
## FIXED POINT DATA MOVEMENT STORE

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STA</strong> Store A 755 (0)</td>
<td></td>
</tr>
<tr>
<td><strong>FORMAT</strong>: Basic Instruction Format (See Figure 2-1).</td>
<td></td>
</tr>
<tr>
<td><strong>SUMMARY</strong>: C(A) -&gt; C(Y)</td>
<td></td>
</tr>
<tr>
<td><strong>MODIFICATIONS</strong>: All except DU, DL</td>
<td></td>
</tr>
<tr>
<td><strong>INDICATORS</strong>: None affected</td>
<td></td>
</tr>
<tr>
<td><strong>NOTES</strong>: Attempted repetition with RPL causes an Illegal Procedure Fault.</td>
<td></td>
</tr>
</tbody>
</table>

| **STAC** Store A Conditional C(Y) = 0 354 (0) |
| **FORMAT**: Basic Instruction Format (See Figure 2-1). |
| **SUMMARY**: If C(Y) = 0, then C(A) -> C(Y) |
| **MODIFICATIONS**: All except DU, DL, CI, SC, SCR |
| **INDICATORS**: (Indicators not listed are not affected) |
| **Zero** If initial C(Y) = 0, then ON; otherwise OFF |
| **NOTES**: If the initial C(Y) is nonzero, then C(Y) is not changed by the STAC instruction. Attempted repetition with RPL causes an Illegal Procedure Fault. |

| **STACQ** Store A Conditional C(Y) = C(Q) 654 (0) |
| **FORMAT**: Basic Instruction Format (See Figure 2-1). |
| **SUMMARY**: If C(Y) = C(Q), then C(A) -> C(Y) |

| **MODIFICATIONS**: All except DU, DL, CI, SC, SCR |
| **INDICATORS**: (Indicators not listed are not affected) |
| **Zero** If initial C(Y) = C(Q), then ON; otherwise OFF |
STACQ

Store AQ

757 (0)

If the initial C(Y) is \( \neq C(Q) \), then C(Y) is not changed by the STACQ instruction.

Attempted repetition with RPL causes an Illegal Procedure Fault.

STBA

Store Character of A (Nine Bit)

551 (0)

Basic Instruction Format (See Figure 2-1).

Characters of C(A) \( \rightarrow \) Corresponding Characters of C(Y), the character positions affected being specified in the tag field.

None

None affected

Binary ones in the tag field of this instruction specify the character positions of A and Y that are affected. The control relations are shown in Table 2-6.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
The text appears to be from a technical document discussing fixed point data movement store, with specific tables for control relations for store character instructions (nine bit). Here are the key points:

### Table 2-6: Control Relations for Store Character Instructions (Nine Bit)

<table>
<thead>
<tr>
<th>Bit Position within Tag Field</th>
<th>Bit of Instruction</th>
<th>Structure of A and Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>30</td>
<td>Char 0 (bits 0-8)</td>
</tr>
<tr>
<td>1</td>
<td>31</td>
<td>Char 1 (bits 9-17)</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>Char 2 (bits 18-26)</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
<td>Char 3 (bits 27-35)</td>
</tr>
</tbody>
</table>

**STBQ**

- Store Character of Q (Nine Bit)
  - 552 (0)

**FORMAT:**

- Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

- Characters of C(Q) -> Corresponding Characters of C(Y), the character positions affected being specified in the tag field.

**MODIFICATIONS:**

- None

**INDICATORS:**

- None affected

**NOTES:**

- Binary ones in the tag field of this instruction specify the character positions of Q and Y that are affected. The control relations are shown in Table 2-6 above.

- Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

**STC1**

- Store Instruction Counter Plus 1
  - 554 (0)

**FORMAT:**

- Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

- C(PPR.IC) + 1 -> C(Y)0,17
- C(IR) -> C(Y)18,31
- 00...0 -> C(Y)32,35

**MODIFICATIONS:**

- All except DU, DL, CI, SC, SCR

---

**REVIEW DRAFT**
**SUBJECT TO CHANGE**
**October, 1975**

2-28 AL39
FIXED POINT DATA MOVEMENT STORE

INDICATORS:
None affected

NOTES:
The contents of the Instruction Counter and the Indicator Register after address preparation are stored in C(Y)0,17 and C(Y)18,31, respectively. C(Y)25 reflects the state of the Tally Runout indicator prior to modification. The relationship between the C(Y)10,31 and the indicators are given in Table 2-5.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

STC2
Store Instruction Counter Plus 2

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
C(PPR.IC) + 2 → C(Y)0,17

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
None affected

NOTES:
Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

STCA
Store Character of A (Six Bit)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
Characters of C(A) → Corresponding Characters of C(Y), the character positions affected being specified in the tag field.

MODIFICATIONS:
None

INDICATORS:
None affected

NOTES:
Binary ones in the tag field of this instruction specify character positions of A and Y that are affected. The control relations are shown in Table 2-7.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
### Table 2-7. Control Relations for Store Character Instructions (Six Bit)

<table>
<thead>
<tr>
<th>Bit Position Within Tag Field</th>
<th>Bit of Instruction</th>
<th>Structure of A and Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>30</td>
<td>Char 0 (bits 0-5)</td>
</tr>
<tr>
<td>1</td>
<td>31</td>
<td>Char 1 (bits 6-11)</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>Char 2 (bits 12-17)</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
<td>Char 3 (bits 18-23)</td>
</tr>
<tr>
<td>4</td>
<td>34</td>
<td>Char 4 (bits 24-29)</td>
</tr>
<tr>
<td>5</td>
<td>35</td>
<td>Char 5 (bits 30-35)</td>
</tr>
</tbody>
</table>

STCQ

**Store Character of Q (Six Bit)**

752 (0)

**FORMAT:**

Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

Characters of C(Q) -> Corresponding Characters of C(Y), the character positions affected being specified by the tag field.

**MODIFICATIONS:**

None

**INDICATORS:**

None affected

**NOTES:**

Binary ones in the tag field of this instruction specify the character positions of Q and Y that are affected. The control relations are shown in Table 2-7 above.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
**STCO**

Store Control Double

**SUMMARY**

00...0 → C(Y-pair)0,2
C(PR,PSR) → C(Y-pair)3,17
C(PR,PRR) → C(Y-pair)18,20
00...0 → C(Y-pair)21,29
43 (octal) → C(Y-pair)30,35
C(PR,IC)+2 → C(Y-pair)36,53
00...0 → C(Y-pair)54,71

**MODIFICATIONS**

All except DU, DL, CI, SC, SCR

**INDICATORS**

None affected

**NOTES**

The hardware assumes Y17 = 0; no check is made.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

---

**STI**

Store Indicator Register

**SUMMARY**

C(IR) → C(Y)18,31
00...0 → C(Y)32,35

**MODIFICATIONS**

All except DU, DL, CI, SC, SCR

**INDICATORS**

None affected

**NOTES**

The contents of the Indicator Register after address preparation are stored in C(Y)18,31. C(Y)25 reflects the state of the Tally Runout indicator prior to address preparation. The relation between C(Y)18,31 and the Indicators is given in Table 2-5.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
FIXED POINT DATA MOVEMENT STORE

STQ

<table>
<thead>
<tr>
<th>Store Q</th>
</tr>
</thead>
</table>

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** C(Q) -> C(Y)

**MODIFICATIONS:** All except DU, DL

**INDICATORS:** None affected

**NOTES:** Attempted repetition with RPL causes an Illegal Procedure Fault.

STT

<table>
<thead>
<tr>
<th>Store Timer Register</th>
</tr>
</thead>
</table>

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

- C(TR) -> C(Y)0,26
- 00...0 -> C(Y)27,35

**MODIFICATIONS:** All except DU, DL, CI, SC, SCR

**INDICATORS:** None affected

**NOTES:** Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

STXn

<table>
<thead>
<tr>
<th>Store Xn in Upper</th>
</tr>
</thead>
</table>

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

- For n = 0, 1, ..., or 7 as determined by operation code
- C(Xn) -> C(Y)0,17

**MODIFICATIONS:** All except DU, DL, CI, SC, SCR

**INDICATORS:** None affected

**NOTES:** Attempted repetition with RPL causes an Illegal Procedure Fault.
**FIXED POINT DATA MOVEMENT STORE**

<table>
<thead>
<tr>
<th>SIZ</th>
<th>Store Zero</th>
<th>450 (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FORMAT:</strong></td>
<td>Basic Instruction Format (See Figure 2-1).</td>
<td></td>
</tr>
<tr>
<td><strong>SUMMARY:</strong></td>
<td>DD...0 -&gt; C(Y)</td>
<td></td>
</tr>
<tr>
<td><strong>MODIFICATIONS:</strong></td>
<td>All except DU, DL</td>
<td></td>
</tr>
<tr>
<td><strong>INDICATORS:</strong></td>
<td>None affected</td>
<td></td>
</tr>
<tr>
<td><strong>NOTES:</strong></td>
<td>Attempted repetition with RPL causes an Illegal Procedure Fault.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SXLn</th>
<th>Store Xn in Lower</th>
<th>44n (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FORMAT:</strong></td>
<td>Basic Instruction Format (See Figure 2-1).</td>
<td></td>
</tr>
<tr>
<td><strong>SUMMARY:</strong></td>
<td>For ( n = 0, 1, \ldots, 7 ) as determined by operation code ( C(Xn) ) -&gt; ( C(Y)18,35 )</td>
<td></td>
</tr>
<tr>
<td><strong>MODIFICATIONS:</strong></td>
<td>All except DU, DL, CI, SC, SCR</td>
<td></td>
</tr>
<tr>
<td><strong>INDICATORS:</strong></td>
<td>None affected</td>
<td></td>
</tr>
<tr>
<td><strong>NOTES:</strong></td>
<td>Attempted repetition with RPL causes an Illegal Procedure Fault.</td>
<td></td>
</tr>
</tbody>
</table>
**Fixed Point Data Movement Shift**

**ALR**

- **FORMAT**: Basic Instruction Format (See Figure 2-1).
- **SUMMARY**: Shift C(A) left the number of positions specified by Y11,17; enter each bit leaving A0 into A35.
- **MODIFICATIONS**: All except DU, DL, CI, SC, SCR
- **INDICATORS**: (Indicators not listed are not affected)
  - **Zero**: If C(A) = 0, then ON; otherwise OFF
  - **Negative**: If C(A)· = 1, then ON; otherwise OFF
- **NOTES**: Attempted repetition with RPL causes an Illegal Procedure Fault.

**ALS**

- **FORMAT**: Basic Instruction Format (See Figure 2-1).
- **SUMMARY**: Shift C(A) left the number of positions specified by Y11,17; fill vacated positions with zeros.
- **MODIFICATIONS**: All except DU, DL, CI, SC, SCR
- **INDICATORS**: (Indicators not listed are not affected)
  - **Zero**: If C(A) = 0, then ON; otherwise OFF
  - **Negative**: If C(A)· = 1, then ON; otherwise OFF
  - **Carry**: If C(A)· changes during the shift, then ON; otherwise OFF
- **NOTES**: Attempted repetition with RPL causes an Illegal Procedure Fault.
ARL

A Right Logic

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
Shift C(A) right the number of positions specified by Y
fill vacated positions with zeros.

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

Zero
If C(A) = 0, then ON; otherwise OFF

Negative
If C(A) = 1, then ON; otherwise OFF

NOTES:
Attempted repetition with RPL causes an Illegal Procedure Fault.

ARS

A Right Shift

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
Shift C(A) right the number of positions specified by Y
fill vacated positions with C(A)0.

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

Zero
If C(A) = 0, then ON; otherwise OFF

Negative
If C(A)0 = 1, then ON; otherwise OFF

NOTES:
Attempted repetition with RPL causes an Illegal Procedure Fault.

LLR

Long Left Rotate

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
Shift C(AQ) left by the number of positions specified by Y
enter each bit leaving AQ0 into AQ71.
FIXED POINT DATA MOVEMENT SHIFT

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF
Negative If C(AQ) = 1, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

LLS Long Left Shift

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Shift C(AQ) left the number of positions specified by Y11,17; fill vacated positions with zeros.

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF
Negative If C(AQ) = 1, then ON; otherwise OFF
Carry If C(AQ) changes during the shift, then ON; otherwise OFF

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

LRL Long Right Logic

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Shift C(AQ) right the number of positions specified by Y11,17; fill vacated positions with zeros.

MODIFICATIONS: All except DU, DL, CI, SC, SCR
INDICATORS:

(Indicators not listed are not affected)

Zero
If C(AQ) = 0, then ON; otherwise OFF

Negative
If C(AQ) = 1, then ON; otherwise OFF

NOTES:
Attempted repetition with RPL causes an Illegal Procedure Fault.

LRS
Long Right Shift 733 (0)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
Shift C(AQ) right the number of positions specified by Y11,17; fill vacated positions with C(A)0.

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

Zero
If C(AQ) = 0, then ON; otherwise OFF

Negative
If C(AQ) = 1, then ON; otherwise OFF

NOTES:
Attempted repetition with RPL causes an Illegal Procedure Fault.

QLR
Q Left Rotate 776 (0)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
Shift C(Q) the number of positions specified by Y11,17; enter each bit leaving Q0 into Q35.

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

Zero
If C(Q) = 0, then ON; otherwise OFF

Negative
If C(Q) = 1, then ON; otherwise OFF

NOTES:
Attempted repetition with RPL causes an Illegal Procedure Fault.
FIXED POINT DATA MOVEMENT SHIFT

QLS  
Q Left Shift  

FORMAT:  
Basic Instruction Format (See Figure 2-1).

SUMMARY:  
Shift C(Q) left the number of positions specified by Y10,17; fill vacated positions with zeros.

MODIFICATIONS:  
All except DU, DL, CI, SC, SCR

INDICATORS:  
(Indicators not listed are not affected)

Zero  
If C(Q) = 0, then ON; otherwise OFF

Negative  
If C(Q)0 = 1, then ON; otherwise OFF

Carry  
If C(Q)0 changes during the shift, then ON; otherwise OFF

NOTES:  
Attempted repetition with RPL causes an Illegal Procedure Fault.

QRL  
Q Right Logic  

FORMAT:  
Basic Instruction Format (See Figure 2-1).

SUMMARY:  
Shift C(Q) right the number of positions specified by Y10,17; fill vacated positions with zeros.

MODIFICATIONS:  
All except DU, DL, CI, SC, SCR

INDICATORS:  
(Indicators not listed are not affected)

Zero  
If C(Q) = 0, then ON; otherwise OFF

Negative  
If C(Q)0 = 1, then ON; otherwise OFF

NOTES:  
Attempted repetition with RPL causes an Illegal Procedure Fault.
<table>
<thead>
<tr>
<th>QRS</th>
<th>Q Right Shift</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FORMAT:</strong></td>
<td>Basic Instruction Format (See Figure 2-1).</td>
</tr>
<tr>
<td><strong>SUMMARY:</strong></td>
<td>Shift C(Q) right the number of positions specified by Y11,17; fill vacated positions with C(Q)0.</td>
</tr>
<tr>
<td><strong>MODIFICATIONS:</strong></td>
<td>All except DU, DL, CI, SC, SCR</td>
</tr>
<tr>
<td><strong>INDICATORS:</strong></td>
<td>(Indicators not listed are not affected)</td>
</tr>
<tr>
<td><strong>Zero</strong></td>
<td>If C(Q) = 0, then ON; otherwise OFF</td>
</tr>
<tr>
<td><strong>Negative</strong></td>
<td>If C(Q)0 = 1, then ON; otherwise OFF</td>
</tr>
<tr>
<td><strong>NOTES:</strong></td>
<td>Attempted repetition with RPL causes an Illegal Procedure Fault.</td>
</tr>
</tbody>
</table>
### ADD

**ADA**

**ADD to A**

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** \( C(A) + C(Y) \rightarrow C(A) \)

**MODIFICATIONS:** All

**INDICATORS:**
- Zero: If \( C(A) = 0 \), then ON; otherwise OFF
- Negative: If \( C(A) = 1 \), then ON; otherwise OFF
- Overflow: If range of A is exceeded, then ON; otherwise OFF
- Carry: If a carry out of A is generated, then ON; otherwise OFF

---

### ADAQ

**ADAQ**

**ADD to AQ**

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** \( C(AQ) + C(Y\text{-pair}) \rightarrow C(AQ) \)

**MODIFICATIONS:** All except DU, DL, CI, SC, SCR

**INDICATORS:**
- Zero: If \( C(AQ) = 0 \), then ON; otherwise OFF
- Negative: If \( C(AQ) = 1 \), then ON; otherwise OFF
- Overflow: If range of AQ is exceeded, then ON; otherwise OFF
- Carry: If a carry out of AQ is generated, then ON; otherwise OFF

---

### ADL

**ADL**

**Add Low to AQ**

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** \( C(AQ) + C(Y) \) sign extended \( \rightarrow C(AQ) \)
### FIXED POINT ADDITION

#### MODIFICATIONS:
All except CI, SC, SCR

#### INDICATORS:
- (Indicators not listed are not affected)
  - **Zero**: If \( C(AQ) = 0 \), then ON; otherwise OFF
  - **Negative**: If \( C(AQ)0 = 1 \), then ON; otherwise OFF
  - **Overflow**: If range of AQ is exceeded, then ON; otherwise OFF
  - **Carry**: If a carry out of AQ is generated, then ON; otherwise OFF

#### NOTES:
- A 72-bit number is formed from \( C(Y) \) in the following manner:
  - The lower 36 bits \((36,71)\) is identical to \( C(Y) \). Each of the upper 36 bits \((0,35)\) is identical to \( C(Y)0 \).
  - This 72-bit number is added to the contents of the combined AQ-register.

#### ADLA
- **Add Logical to A**

#### FORMAT:
- Basic Instruction Format (See Figure 2-1).

#### SUMMARY:
- \( C(A) + C(Y) \rightarrow C(A) \)

#### MODIFICATIONS:
- All

#### INDICATORS:
- (Indicators not listed are not affected)
  - **Zero**: If \( C(A) = 0 \), then ON; otherwise OFF
  - **Negative**: If \( C(A)0 = 1 \), then ON; otherwise OFF
  - **Carry**: If a carry out of AO is generated, then ON; otherwise OFF

#### NOTES:
- The ADLA instruction is identical to the ADA instruction with the exception that the Overflow indicator is not affected by the ADLA instruction, nor does an Overflow Fault occur. Operands and results are treated as unsigned, positive binary integers.
**FIXED POINT ADDITION**

### ADL AQ

**Add Logical to AQ** 037 (0)

<table>
<thead>
<tr>
<th>FORMAT:</th>
<th>Basic Instruction Format (See Figure 2-1).</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUMMARY:</td>
<td>$C(AQ) + CY$ -&gt; $C(AQ)$</td>
</tr>
<tr>
<td>MODIFICATIONS:</td>
<td>All except DU, DL, CI, SC, SCR</td>
</tr>
<tr>
<td>INDICATORS:</td>
<td>(Indicators not listed are not affected)</td>
</tr>
</tbody>
</table>

- **Zero**: If $C(AQ) = 0$, then ON; otherwise OFF
- **Negative**: If $C(AQ)0 = 1$, then ON; otherwise OFF
- **Carry**: If a carry out of AQ0 is generated, then ON; otherwise OFF

**NOTES**: The ADL AQ instruction is identical to the ADAQ instruction with the exception that the Overflow indicator is not affected by the ADL AQ instruction, nor does an Overflow Fault occur. Operands and results are treated as unsigned, positive binary integers.

### ADL Q

**Add Logical to Q** 036 (0)

<table>
<thead>
<tr>
<th>FORMAT:</th>
<th>Basic Instruction Format (See Figure 2-1).</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUMMARY:</td>
<td>$C(Q) + C(Y)$ -&gt; $C(Q)$</td>
</tr>
<tr>
<td>MODIFICATIONS:</td>
<td>All</td>
</tr>
<tr>
<td>INDICATORS:</td>
<td>(Indicators not listed are not affected)</td>
</tr>
</tbody>
</table>

- **Zero**: If $C(Q) = 0$, then ON; otherwise OFF
- **Negative**: If $C(Q)0 = 1$, then ON; otherwise OFF
- **Carry**: If a carry out of Q0 is generated, then ON; otherwise OFF

**NOTES**: The ADL Q instruction is identical to the ADQ instruction with the exception that the Overflow indicator is not affected by the ADL Q instruction, nor does an Overflow Fault occur. Operands and results are treated as unsigned, positive binary integers.
**ADLXn**

Add Logical to \( X_n \)

02n (0)

**FORMAT**: Basic Instruction Format (See Figure 2-1).

**SUMMARY**: For \( n = 0, 1, \ldots, \) or 7 as determined by operation code

\[
C(X_n) + C(Y)_{0,17} \rightarrow C(X_n)
\]

**MODIFICATIONS**: All except CI, SC, SCR

**INDICATORS**: (Indicators not listed are not affected)

- **Zero**: If \( C(X_n) = 0 \), then ON; otherwise OFF
- **Negative**: If \( C(X_n) = 1 \), then ON; otherwise OFF
- **Carry**: If a carry out of \( X_n0 \) is generated, then ON; otherwise OFF

**NOTES**: The ADLXn instruction is identical to the ADXn instruction with the exception that the Overflow indicator is not affected by the ADLXn instruction, nor does an Overflow Fault occur. Operands and results are treated as unsigned, positive binary integers.

**ADQ**

Add to Q

076 (0)

**FORMAT**: Basic Instruction Format (See Figure 2-1).

**SUMMARY**: \( C(Q) + C(Y) \rightarrow C(Q) \)

**MODIFICATIONS**: All

**INDICATORS**: (Indicators not listed are not affected)

- **Zero**: If \( C(Q) = 0 \), then ON; otherwise OFF
- **Negative**: If \( C(Q) = 1 \), then ON; otherwise OFF
- **Overflow**: If range of \( Q \) is exceeded, then ON; otherwise OFF

- **Carry**: If a carry out of \( Q0 \) is generated, then ON; otherwise OFF
### FIXED POINT ADDITION

**ADXn**

- **Add to Xn**
- **06n (0)**

**FORMAT:**

Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

For \( n = 0, 1, \ldots, \) or 7 as determined by operation code

\[
C(Xn) + C(Y)_{0,17} \rightarrow C(Xn)
\]

**MODIFICATIONS:**

All except CI, SC, SCR

**INDICATORS:**

(Indicators not listed are not affected)

- **Zero**
  - If \( C(Xn) = 0 \), then ON; otherwise OFF
- **Negative**
  - If \( C(Xn)0 = 1 \), then ON; otherwise OFF
- **Overflow**
  - If range of \( Xn \) is exceeded, then ON; otherwise OFF
- **Carry**
  - If a carry out of \( Xn0 \) is generated, then ON; otherwise OFF

**AoS**

- **Add One to Storage**
- **054 (0)**

**FORMAT:**

Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

\( C(Y) + 1 \rightarrow C(Y) \)

**MODIFICATIONS:**

All except DU, DL, CI, SC, SCR

**INDICATORS:**

(Indicators not listed are not affected)

- **Zero**
  - If \( C(Y) = 0 \), then ON; otherwise OFF
- **Negative**
  - If \( C(Y)0 = 1 \), then ON; otherwise OFF
- **Overflow**
  - If range of \( Y \) is exceeded, then ON; otherwise OFF
- **Carry**
  - If a carry out of \( Y0 \) is generated, then ON; otherwise OFF

**NOTES:**

Attempted repetition with RPL causes an Illegal Procedure Fault.

---

**REVIEW DRAFT**

**SUBJECT TO CHANGE**

October, 1975

2-44

AL39
ASA

Add Stored to A

ASSUMPTIONS: Basic Instruction Format (See Figure 2-1).

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero
Negative
Overflow
Carry

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.

ASQ

Add Stored to Q

ASSUMPTIONS: Basic Instruction Format (See Figure 2-1).

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero
Negative
Overflow
Carry

NOTES: Attempted repetition with RPL causes an Illegal Procedure Fault.
FIXED POINT ADDITION

ASXn
Add Stored to Xn
04n (0)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
For n = 0, 1, ..., or 7 as determined by operation code
C(Xn) + C(Y)0,17 -> C(Y)0,17

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

Zero
If C(Y)0,17 = 0, then ON; otherwise OFF
Negative
If C(Y)0 = 1, then ON; otherwise OFF
Overflow
If range of Y0,17 is exceeded, then ON; otherwise OFF
Carry
If a carry out of Y0 is generated, then ON; otherwise OFF

NOTES:
Attempted repetition with RPL causes an Illegal Procedure Fault.

AWCA
Add with Carry to A
071 (0)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
If Carry indicator OFF, then C(A) + C(Y) -> C(A)
If Carry indicator ON, then C(A) + C(Y) + 1 -> C(A)

MODIFICATIONS:
All

INDICATORS:
(Indicators not listed are not affected)

Zero
If C(A) = 0, then ON; otherwise OFF
Negative
If C(A)0 = 1, then ON; otherwise OFF
Overflow
If range of A is exceeded, then ON; otherwise OFF

Carry
If a carry out of A0 is generated, then ON; otherwise OFF

NOTES:
The AWCA instruction is identical to the ADA instruction
with the exception that when the Carry indicator is ON at
the beginning of the instruction, 1 is added to the sum of
C(A) and C(Y).

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975

2-46

AL39
AWCQ

Add with Carry to Q

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
If Carry indicator OFF, then C(Q) + C(Y) -> C(Q)
If Carry indicator ON, then C(Q) + C(Y) + 1 -> C(Q)

MODIFICATIONS:
All

INDICATORS:
(Indicators not listed are not affected)
Zero
If C(Q) = 0, then ON; otherwise OFF
Negative
If C(Q)0 = 1, then ON; otherwise OFF
Overflow
If range of Q is exceeded, then ON; otherwise OFF
Carry
If a carry out of Q0 is generated, then ON; otherwise OFF

NOTES:
The AWCQ instruction is identical to the ADQ instruction
with the exception that when the Carry indicator is ON at
the beginning of the instruction, 1 is added to the sum of
C(Q) and C(Y).
**FIXED POINT SUBTRACTION**

"Fixed-Point Subtraction"

**SBA**

- **SUBTRACTION**
  - **FORMAT:** Basic Instruction Format (See Figure 2-1).
  - **SUMMARY:** \( C(A) - C(Y) \rightarrow C(A) \)
  - **MODIFICATIONS:** All
  - **INDICATORS:**
    - Zero: If \( C(A) = 0 \), then ON; otherwise OFF
    - Negative: If \( C(A) = 1 \), then ON; otherwise OFF
    - Overflow: If range of \( A \) is exceeded, then ON; otherwise OFF
    - Carry: If a carry out of \( A0 \) is generated, then ON; otherwise OFF

**SBAQ**

- **SUBTRACTION**
  - **FORMAT:** Basic Instruction Format (See Figure 2-1).
  - **SUMMARY:** \( C(AQ) - C(Y) \rightarrow C(AQ) \)
  - **MODIFICATIONS:** All except DU, DL, CI, SC, SCR
  - **INDICATORS:**
    - Zero: If \( C(AQ) = 0 \), then ON; otherwise OFF
    - Negative: If \( C(AQ) = 1 \), then ON; otherwise OFF
    - Overflow: If range of \( AQ \) is exceeded, then ON; otherwise OFF
    - Carry: If a carry out of \( AQ0 \) is generated, then ON; otherwise OFF

**SBLA**

- **SUBTRACTION**
  - **FORMAT:** Basic Instruction Format (See Figure 2-1).
  - **SUMMARY:** \( C(A) - C(Y) \rightarrow C(A) \)
FIXED POINT SUBTRACTION

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero: If C(A) = 0, then ON; otherwise OFF
Negative: If C(A)0 = 1, then ON; otherwise OFF
Carry: If a carry out of A0 is generated, then ON; otherwise OFF

NOTES: The SBLA instruction is identical to the SBA instruction with the exception that the Overflow indicator is not affected by the SBLA instruction, nor does an Overflow Fault occur. Operands and results are treated as unsigned, positive binary integers.

SBLAQ

Subtract Logical from AQ

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(AQ) - C(Y-pair) -> C(AQ)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero: If C(AQ) = 0, then ON; otherwise OFF
Negative: If C(AQ)0 = 1, then ON; otherwise OFF
Carry: If a carry out of AQ is generated, then ON; otherwise OFF

NOTES: The SBLAQ instruction is identical to the SBAQ instruction with the exception that the Overflow indicator is not affected by the SBLAQ instruction. Operands and results are treated as unsigned, positive binary integers.

SBLQ

Subtract Logical from Q

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Q) - C(Y) -> C(Q)

MODIFICATIONS: All
FIXED POINT SUBTRACTION

INDICATORS:

Zero
If C(Q) = 0, then ON; otherwise OFF

Negative
If C(Q) = 1, then ON; otherwise OFF

Carry
If a carry out of Q0 is generated, then ON; otherwise OFF

NOTES:
The SBLQ instruction is identical to the SBQ instruction with the exception that the Overflow indicator is not affected by the SBLQ instruction, nor does an Overflow Fault occur. Operands and results are treated as unsigned, positive binary integers.

SBLXn
Subtract Logical from Xn

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
For n = 0, 1, ..., or 7 as determined by operation code
C(Xn) - C(Y)0,17 -> C(Xn)

MODIFICATIONS:
All except CI, SC, SCR

INDICATORS:

Zero
If C(Xn) = 0, then ON; otherwise OFF

Negative
If C(Xn) = 1, then ON; otherwise OFF

Carry
If a carry out of Xn0 is generated, then ON; otherwise OFF

NOTES:
The SBLXn instruction is identical to the SBXn instruction with the exception that the Overflow indicator is not affected by the SBLXn instruction, nor does an Overflow Fault occur. Operands and results are treated as unsigned, positive binary integers.

SBQ
Subtract from Q

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
C(Q) - C(Y) -> C(Q)

MODIFICATIONS:
All
FIXED POINT SUBTRACTION

INDICATORS: (Indicators not listed are not affected)

- Zero
  If \( C(Q) = 0 \), then ON; otherwise OFF
- Negative
  If \( C(Q) = 1 \), then ON; otherwise OFF
- Overflow
  If range of \( Q \) is exceeded, then ON; otherwise OFF
- Carry
  If a carry out of \( Q0 \) is generated, then ON; otherwise OFF

S\(8\)x\(n\)
Subtract from \( Xn \)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
For \( n = 0, 1, \ldots, 7 \) as determined by operation code
\[ C(Xn) - C(Y)0,17 \rightarrow C(Xn) \]

MODIFICATIONS:
All except CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

- Zero
  If \( C(Xn) = 0 \), then ON; otherwise OFF
- Negative
  If \( C(Xn) = 1 \), then ON; otherwise OFF
- Overflow
  If range of \( Xn \) is exceeded, then ON; otherwise OFF
- Carry
  If a carry out of \( Xn0 \) is generated, then ON; otherwise OFF

SSA
Subtract Stored from \( A \)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
\[ C(A) - C(Y) \rightarrow C(Y) \]

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

- Zero
  If \( C(Y) = 0 \), then ON; otherwise OFF
- Negative
  If \( C(Y) = 1 \), then ON; otherwise OFF
- Overflow
  If range of \( Y \) is exceeded, then ON; otherwise OFF
- Carry
  If a carry out of \( Y0 \) is generated, then ON; otherwise OFF

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975

2-51

AL39
### FIXED POINT SUBTRACTION

**NOTES:** Attempted repetition with RPL causes an Illegal Procedure Fault.

<table>
<thead>
<tr>
<th>SSQ</th>
<th>Subtract Stored from Q</th>
<th>156 (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FORMAT:</strong></td>
<td>Basic Instruction Format (See Figure 2-1).</td>
<td></td>
</tr>
<tr>
<td><strong>SUMMARY:</strong></td>
<td>C(Q) - C(Y) -&gt; C(Y)</td>
<td></td>
</tr>
<tr>
<td><strong>MODIFICATIONS:</strong></td>
<td>All except DU, DL, CI, SC, SCR</td>
<td></td>
</tr>
<tr>
<td><strong>INDICATORS:</strong></td>
<td>(Indicators not listed are not affected)</td>
<td></td>
</tr>
<tr>
<td><strong>Zero</strong></td>
<td>If C(Y) = 0, then ON; otherwise OFF</td>
<td></td>
</tr>
<tr>
<td><strong>Negative</strong></td>
<td>If C(Y)0 = 1, then ON; otherwise OFF</td>
<td></td>
</tr>
<tr>
<td><strong>Overflow</strong></td>
<td>If range of Y is exceeded, then ON; otherwise OFF</td>
<td></td>
</tr>
<tr>
<td><strong>Carry</strong></td>
<td>If a carry out of Y0 is generated, then ON; otherwise OFF</td>
<td></td>
</tr>
</tbody>
</table>

**NOTES:** Attempted repetition with RPL causes an Illegal Procedure Fault.

<table>
<thead>
<tr>
<th>SSXn</th>
<th>Subtract Stored from Xn</th>
<th>14n (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FORMAT:</strong></td>
<td>Basic Instruction Format (See Figure 2-1).</td>
<td></td>
</tr>
<tr>
<td><strong>SUMMARY:</strong></td>
<td>For n = 0, 1, ..., or 7 as determined by operation code C(Xn) - C(Y)0,17 -&gt; C(Y)0,17</td>
<td></td>
</tr>
<tr>
<td><strong>MODIFICATIONS:</strong></td>
<td>All except DU, DL, CI, SC, SCR</td>
<td></td>
</tr>
<tr>
<td><strong>INDICATORS:</strong></td>
<td>(Indicators not listed are not affected)</td>
<td></td>
</tr>
<tr>
<td><strong>Zero</strong></td>
<td>If C(Y)0,17 = 0, then ON; otherwise OFF</td>
<td></td>
</tr>
<tr>
<td><strong>Negative</strong></td>
<td>If C(Y)0 = 1, then ON; otherwise OFF</td>
<td></td>
</tr>
<tr>
<td><strong>Overflow</strong></td>
<td>If range of Y0,17 exceeded, then ON; otherwise OFF</td>
<td></td>
</tr>
<tr>
<td><strong>Carry</strong></td>
<td>If a carry out of Y0 is generated, then ON; otherwise OFF</td>
<td></td>
</tr>
</tbody>
</table>

**NOTES:** Attempted repetition with RPL causes an Illegal Procedure Fault.
FIXED POINT SUBTRACTION

**SWCA**

Subtract with Carry from A

| FORMAT: | Basic Instruction Format (See Figure 2-1). |
| SUMMARY: | If Carry indicator ON, then C(A) - C(Y) -> C(A)  
If Carry indicator OFF, then C(A) - C(Y) - 1 -> C(A) |
| MODIFICATIONS: | All |
| INDICATORS: | (Indicators not listed are not affected)  
Zero: If C(A) = 0, then ON; otherwise OFF  
Negative: If C(A)0 = 1, then ON; otherwise OFF  
Overflow: If range of A is exceeded, then ON; otherwise OFF  
Carry: If a carry out of A0 is generated, then ON; otherwise OFF |
| NOTES: | The SWCA instruction is identical to the SBA instruction with the exception that when the Carry indicator is OFF at the beginning of the instruction, +1 is subtracted from the difference of C(A) minus C(Y). The SWCA instruction treats the Carry indicator as the complement of a borrow indicator due to the implementation of negative numbers in two's complement form. |

**SWCQ**

Subtract with Carry from Q

| FORMAT: | Basic Instruction Format (See Figure 2-1). |
| SUMMARY: | If Carry indicator ON, then C(Q) - C(Y) -> C(Q)  
If Carry indicator OFF, then C(Q) - C(Y) - 1 -> C(Q) |
| MODIFICATIONS: | All |
| INDICATORS: | (Indicators not listed are not affected)  
Zero: If C(Q) = 0, then ON; otherwise OFF  
Negative: If C(Q)0 = 1, then ON; otherwise OFF  
Overflow: If range of Q is exceeded, then ON; otherwise OFF  
Carry: If a carry out of Q0 is generated, then ON; otherwise OFF |

REVIEW DRAFT  
SUBJECT TO CHANGE  
October, 1975
The SWCQ instruction is identical to the SBQ instruction with the exception that when the Carry indicator is OFF at the beginning of the instruction, +1 is subtracted from the difference of C(Q) minus C(Y). The SWCQ instruction treats the Carry indicator as the complement of a borrow indicator; due to the implementation of negative numbers in two's complement form.
**Fixed Point Multiplication**

**HPF**

Multiply Fraction

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** 

\[ C(A) \times C(Y) \rightarrow C(AQ), \text{left adjusted} \]

**MODIFICATIONS:** All except CI, SC, SCR

**INDICATORS:** (Indicators not listed are not affected)

- **Zero**
  If \( C(AQ) = 0 \), then ON; otherwise OFF

- **Negative**
  If \( C(AQ) = 1 \), then ON; otherwise OFF

- **Overflow**
  If range of AQ is exceeded, then ON; otherwise OFF

**NOTES:**

Two 36-bit fractional factors (including sign) are multiplied to form a 71-bit fractional product (including sign), which is stored left-adjusted in the AQ-register. AQ71 contains a zero. Overflow can occur only in the case of \( A \) and \( Y \) containing all ones and the result exceeding the combined AQ-register.

\[
\begin{array}{c|c|c}
  0 & 0 & 0 \\
  1 & 1 & 1 \\
\end{array}
\times
\begin{array}{c|c|c}
  0 & 0 & 0 \\
  1 & 1 & 1 \\
\end{array}
\]

A Register

Main Store Location Y

yielding

\[
\begin{array}{c|c|c}
  0 & 0 & 7 \\
  1 & 1 & 7 \\
\end{array}
\]

Combined AQ Register

**HPY**

Multiply Integer

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** 

\[ C(Q) \times C(Y) \rightarrow C(AQ), \text{right adjusted} \]


**MODIFICATIONS:** All except CI, SC, SCR

**INDICATORS:** (Indicators not listed are not affected)

- **Zero:** If C(AQ) = 0, then ON; otherwise OFF
- **Negative:** If C(AQ) = 1, then ON; otherwise OFF

**NOTES:**

Two 36-bit integer factors (including sign) are multiplied to form a 71-bit integer product (including sign), which is stored in AQ, right-adjusted. AQ0 is filled with an extended sign bit.

<table>
<thead>
<tr>
<th>0 0</th>
<th>3 0 0</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1</td>
<td>5 0 1</td>
<td>5</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1 1</td>
<td>1 1 1</td>
</tr>
</tbody>
</table>

Q Register

Main Store Location Y

yielding

<table>
<thead>
<tr>
<th>0 0 0</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2</td>
<td>1</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 1</td>
</tr>
</tbody>
</table>

Combined AQ Register

In the case of (-2**35) x (-2**35) = +2**70, AQ1 is used to represent the product rather than the sign. No overflow can occur.
"Fixed-Point Division"

DIV
Divide Integer

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
C(Q) / C(Y) integer quotient → C(Q)
integer remainder → C(A)

MODIFICATIONS:
All

INDICATORS:
(Indicators not listed are not affected)

If division takes place
Zero
If C(Q) = 0, then ON;
otherwise OFF

Negative
If C(Q) = 1, then ON;
otherwise OFF

If no division takes place
Zero
If divisor = 0, then ON;
otherwise OFF

Negative
If dividend < 0, then ON;
otherwise OFF

NOTES:
A 36-bit integer dividend (including sign) is divided by a
36-bit integer divisor (including sign) to form a 36-bit
integer quotient (including sign) and a 36-bit integer
remainder (including sign). The remainder sign is equal
to the dividend sign unless the remainder is zero.

0 0 3
0 1 5
1 1

|sl|<----dividend-------->1
1 1

Q Register
yielding

0 0 3
0 1 5
1 1

|sl|<----remainder-------->1
1 1

A-Register

If the dividend = -2**35 and the divisor = -1, or if the
divisor = 0, then division does not take place. Instead,
a Divide Check fault occurs. C(Q) contains the dividend
magnitude, and the Negative indicator reflects the
dividend sign.
FIXED POINT DIVISION

DVF

Divide Fraction

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
C(AQ) / C(Y) fractional quotient -> C(A)
fractional remainder -> C(Q)

MODIFICATIONS:
All

INDICATORS:
(Indicators not listed are not affected)

If division takes place If no division takes place
Zero
If C(A) = 0, then ON; If divisor = 0, then ON;
otherwise OFF otherwise OFF

Negative
If C(AQ) = 1, then ON; If dividend < 0, then ON;
otherwise OFF otherwise OFF

NOTES:
A 71-bit fractional dividend (including sign) is divided
by a 36-bit fractional divisor yielding a 36-bit
fractional quotient (including sign) and a 36-bit
fractional remainder (including sign). C(AQ)71 is
ignored; bit position 35 of the remainder corresponds
to bit position 70 of the dividend. The remainder sign
is equal to the dividend sign unless the remainder is zero.

0 0
1 1
\[1|1\]
\[\text{Left }<\text{- dividend}\]
\[\text{Main Store Location } Y\]
yielding

Combined AQ-Register
0 0 3
0 1 5
1 1
\[\text{Left }<\text{- divisor}\]

A-Register
0 0
0 1
1 1
\[\text{Left }<\text{- quotient}\]

Q-Register
0 0
0 1
1 1
\[\text{Left }<\text{- remainder}\]
If \( |\text{dividend}| \geq |\text{divisor}| \) or if the divisor = 0, division does not take place. Instead, a Divide Check Fault occurs, C(AQ) contains the dividend magnitude in absolute, and the Negative indicator reflects the dividend sign.
FIXED POINT NEGATE

NEG

Negate A

 FORMAT: Basic Instruction Format (See Figure 2-1).

 SUMMARY: \(-C(A) \rightarrow C(A) \text{ if } C(A) \neq 0\)

 MODIFICATIONS: All, but none affect instruction execution.

 INDICATORS: (Indicators not listed are not affected)

 Zero If \(C(A) = 0\), then ON; otherwise OFF

 Negative If \(C(A) = 1\), then ON; otherwise OFF

 Overflow If range of \(A\) is exceeded, then ON; otherwise OFF

 NOTES: The NEG instruction changes the number in \(A\) to its negative (if \(\neq 0\)). The operation is performed by forming the two's complement of the string of 36 bits.

 Attempted repetition with RPL causes an Illegal Procedure Fault.

 NEGL

 Negate Long

 FORMAT: Basic Instruction Format (See Figure 2-1).

 SUMMARY: \(-C(AQ) \rightarrow C(AQ) \text{ if } C(AQ) \neq 0\)

 MODIFICATIONS: All, but none affect instruction execution.

 INDICATORS: (Indicators not listed are not affected)

 Zero If \(C(AQ) = 0\), then ON; otherwise OFF

 Negative If \(C(AQ) = 1\), then ON; otherwise OFF

 Overflow If range of \(AQ\) is exceeded, then ON; otherwise OFF

 NOTES: The NEGL instruction changes the number in \(AQ\) to its negative (if \(\neq 0\)). The operation is performed by forming the two's complement of the string of 72 bits.

 Attempted repetition with RPL causes an Illegal Procedure Fault.
"Fixed-Point" Comparison

CMG

Compare Magnitude

405 (0)

FORMAT:

Basic Instruction Format (See Figure 2-1).

SUMMARY:

IC(A) ≥ IC(Y)

MODIFICATIONS:

All

INDICATORS:

Zero: If IC(A) ≥ IC(Y), then ON; otherwise OFF

Negative: If IC(A) ≤ IC(Y), then ON; otherwise OFF

CMK

Compare Masked

211 (0)

FORMAT:

Basic Instruction Format (See Figure 2-1).

SUMMARY:

For i = 0, 1, ..., 35

C(Z)i = ¬C(Q)j & (C(A)i ⊕ C(Y)i)

MODIFICATIONS:

All

INDICATORS:

Zero: If C(Z) = 0, then ON; otherwise OFF

Negative: If C(Z)j = 1, then ON; otherwise OFF

NOTES:

The CMK instruction compares the contents of bit positions of A and Y for identity that are not masked by a 1 in the corresponding bit position of Q.

The Zero indicator is set ON if the comparison is successful for all bit positions; i.e., if for all i = 0, 1, ..., 35 there is either C(A)i = C(Y)i (the identical case) or C(Q)i = 1 (the masked case); otherwise, Zero indicator is set OFF.

The Negative Indicator is set ON if the comparison is unsuccessful for bit position 0; i.e., if C(A)0 ⊕ C(Y)0 (they are nonidentical) as well as C(Q)0 = 0 (they are unmasked); otherwise, Negative indicator is set OFF.
FIXED POINT COMPARISON

CMPA

Compare with A

115 (0)

FORMAT:
Base Instruction Format (See Figure 2-1).

SUMMARY:
C(A) \equiv C(Y)

MODIFICATIONS:
All

INDICATORS:
(Indicators not listed are not affected)

The Zero (Z), Negative (N), and Carry (C) indicators are set as follows.

Algebraic Comparison (Signed Binary Operands)

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>Relation</th>
<th>Sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>C(A) &gt; C(Y)</td>
<td>C(A)0 = 0, C(Y)0 = 1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>C(A) &gt; C(Y)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>C(A) = C(Y)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>C(A) &lt; C(Y)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>C(A) &lt; C(Y)</td>
<td></td>
</tr>
</tbody>
</table>

Logical Comparison (Unsigned Positive Binary Operands)

<table>
<thead>
<tr>
<th>Z</th>
<th>C</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>C(A) &lt; C(Y)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>C(A) = C(Y)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>C(A) &gt; C(Y)</td>
</tr>
</tbody>
</table>

CMPAQ

Compare with AQ

117 (0)

FORMAT:
Base Instruction Format (See Figure 2-1).

SUMMARY:
C(AQ) \equiv C(Y-pair)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR
**INDICATORS**

(Indicators not listed are not affected)

The Zero (Z), Negative (N), and Carry (C) indicators are set as follows.

**Algebraic Comparison (Signed Binary Operands)**

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>Relation</th>
<th>Sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>C(AQ) &gt; C(Y-pair)</td>
<td>C(AQ)0 = 0, C(Y-pair)0 = 1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>C(AQ) &gt; C(Y-pair)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>C(AQ) = C(Y-pair)</td>
<td>C(AQ)0 = C(Y-pair)0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>C(AQ) &lt; C(Y-pair)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>C(AQ) &lt; C(Y-pair)</td>
<td>C(AQ)0 = 1, C(Y-pair)0 = 0</td>
</tr>
</tbody>
</table>

**Logical Comparison (Unsigned Positive Binary Operands)**

<table>
<thead>
<tr>
<th>Z</th>
<th>C</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>C(AQ) &lt; C(Y-pair)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>C(AQ) = C(Y-pair)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>C(AQ) &gt; C(Y-pair)</td>
</tr>
</tbody>
</table>

**CMPO**

Compare with Q

**FORMAT**

Basic Instruction Format (See Figure 2-1).

**SUMMARY**

C(Q) :: C(Y)

**MODIFICATIONS**

All

**INDICATORS**

(Indicators not listed are not affected)

The Zero (Z), Negative (N), and Carry (C) indicators are set as follows.
## FIXED POINT COMPARISON

### Algebraic Comparison (Signed Binary Operands)

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>Relation</th>
<th>Sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>C(Q) &gt; C(Y)</td>
<td>C(Q)0 = 0, C(Y)0 = 1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>C(Q) &gt; C(Y)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>C(Q) = C(Y)</td>
<td>C(Q)0 = C(Y)0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>C(Q) &lt; C(Y)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>C(Q) &lt; C(Y)</td>
<td>C(Q)0 = 1, C(Y)0 = 0</td>
</tr>
</tbody>
</table>

### Logical Comparison (Unsigned Positive Binary Operands)

<table>
<thead>
<tr>
<th>Z</th>
<th>C</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>C(Q) &lt; C(Y)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>C(Q) = C(Y)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>C(Q) &gt; C(Y)</td>
</tr>
</tbody>
</table>

### CMPXn

Compare with Xn

**FORMAT:**

Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

For n = 0, 1, ..., or 7 as determined by operation code

\[ C(Xn) \equiv C(Y)0,17 \]

**MODIFICATIONS:**

All except CI, SC, SCR

**INDICATORS:**

(Indicators not listed are not affected)

The Zero (Z), Negative (N), and Carry (C) indicators are set as follows.

### Algebraic Comparison (Signed Binary Operands)

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>Relation</th>
<th>Sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>C(Xn) &gt; C(Y)0,17</td>
<td>C(Xn)0 = 0, C(Y)0 = 1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>C(Xn) &gt; C(Y)0,17</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>C(Xn) = C(Y)0,17</td>
<td>C(Xn)0 = C(Y)0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>C(Xn) &lt; C(Y)0,17</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>C(Xn) &lt; C(Y)0,17</td>
<td>C(Xn)0 = 1, C(Y)0 = 0</td>
</tr>
</tbody>
</table>
**FIXED POINT COMPARISON**

**Logical Comparison (Unsigned Positive Binary Operands)**

<table>
<thead>
<tr>
<th>Z</th>
<th>Relation</th>
<th>C(Xn) &lt; C(Y)0,17</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>C(Xn) = C(Y)0,17</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>C(Xn) &gt; C(Y)0,17</td>
</tr>
</tbody>
</table>

**CWL**

Compare with Limits

**FORMAT**

Basic Instruction Format (See Figure 2-1).

**SUMMARY**

- C(Y) := closed interval [C(A);C(Q)]
- C(Y) := C(Q)

**MODIFICATIONS**

All

**INDICATORS**

(Indicators not listed are not affected)

- **Zero**
  - If C(A) <= C(Y) <= C(Q) or C(A) >= C(Y) >= C(Q), then ON; otherwise OFF.

The Negative (N) and Carry (C) indicators are set as follows.

<table>
<thead>
<tr>
<th>N</th>
<th>Relation</th>
<th>Sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>C(Q)0 = 0, C(Y)0 = 1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>C(Q)0 = C(Y)0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>C(Q) &lt; C(Y)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>C(Q) &lt; C(Y)</td>
</tr>
</tbody>
</table>

**NOTES**

The CWL instruction tests the value of C(Y) to determine if it is within the range of values set by C(A) and C(Q). The comparison of C(Y) with C(Q) locates C(Y) with respect to the interval if C(Y) is not contained within the interval.
FIXED POINT MISCELLANEOUS

Fixed Point Miscellaneous

<table>
<thead>
<tr>
<th>SZN</th>
<th>Set Zero and Negative Indicators</th>
<th>234 (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMAT:</td>
<td>Basic Instruction Format (See Figure 2-1).</td>
<td></td>
</tr>
<tr>
<td>SUMMARY:</td>
<td>Set indicators according to C(Y)</td>
<td></td>
</tr>
<tr>
<td>MODIFICATIONS:</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>INDICATORS:</td>
<td>(Indicators not listed are not affected)</td>
<td></td>
</tr>
<tr>
<td>Zero</td>
<td>If C(Y) = 0, then ON; otherwise OFF</td>
<td></td>
</tr>
<tr>
<td>Negative</td>
<td>If C(Y) = 1, then ON; otherwise OFF</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SZNC</th>
<th>Set Zero and Negative Indicators and Clear</th>
<th>214 (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMAT:</td>
<td>Basic Instruction Format (See Figure 2-1).</td>
<td></td>
</tr>
<tr>
<td>SUMMARY:</td>
<td>Set indicators according to C(Y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>00...0 =&gt; C(Y)</td>
<td></td>
</tr>
<tr>
<td>MODIFICATIONS:</td>
<td>All except DU, DL, CI, SC, SCR</td>
<td></td>
</tr>
<tr>
<td>INDICATORS:</td>
<td>(Indicators not listed are not affected)</td>
<td></td>
</tr>
<tr>
<td>Zero</td>
<td>If C(Y) = 0, then ON, otherwise OFF</td>
<td></td>
</tr>
<tr>
<td>Negative</td>
<td>If C(Y) = 1, then ON; otherwise OFF</td>
<td></td>
</tr>
</tbody>
</table>
"BOOLEAN-OPERATION INSTRUCTIONS"

"Boolean-ANA"

ANA AND to A

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(A)i & C(Y)i -> C(A)i for i = (0, 1, ..., 35)

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero If C(A) = 0, then ON; otherwise OFF

Negative If C(A) = 1, then ON; otherwise OFF

ANAQ AND to AQ

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(AQ)i & C(Y-pair)i -> C(AQ)i for i = (0, 1, ..., 71)

MODIFICATIONS: All except OI, OL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF

Negative If C(AQ) = 1, then ON; otherwise OFF

ANQ AND to Q

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Q)i & C(Y)i -> C(Q)i for i = (0, 1, ..., 35)

MODIFICATIONS: All
BOOLEAN AND

INDICATORS:
(Indicators not listed are not affected)

Zero
If \( C(Q) = 0 \), then ON; otherwise OFF

Negative
If \( C(Q) = 1 \), then ON; otherwise OFF

ANS A

AND to Storage A

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
\( C(A)i \cdot C(Y)i \rightarrow C(Y)i \) for \( i = (0, 1, \ldots, 35) \)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

Zero
If \( C(Y) = 0 \), then ON; otherwise OFF

Negative
If \( C(Y) = 1 \), then ON; otherwise OFF

NOTES:
Attempted repetition with RPL causes an Illegal Procedure Fault.

ANSQ

AND to Storage Q

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
\( C(Q)i \cdot C(Y)i \rightarrow C(Y)i \) for \( i = (0, 1, \ldots, 35) \)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

Zero
If \( C(Y) = 0 \), then ON; otherwise OFF

Negative
If \( C(Y) = 1 \), then ON; otherwise OFF

NOTES:
Attempted repetition with RPL causes an Illegal Procedure Fault.
ANSXn

ANXn

AND to Storage Xn

AND to Xn

FORMAT:
Basic Instruction Format (See Figure 2-1).

Basic Instruction Format (See Figure 2-1).

SUMMARY:
For n = 0, 1, ..., or 7 as determined by operation code

C(Xn)i & C(Y)i → C(Y)i for i = (0, 1, ..., 17)

C(Xn)i & C(Y)i → C(Xn)i for i = (0, 1, ..., 17)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

All except CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

(Indicators not listed are not affected)

Zero
If C(Y)0,17 = 0, then ON; otherwise OFF

Zero
If C(Xn) = 0, then ON; otherwise OFF

Negative
If C(Y)0 = 1, then ON; otherwise OFF

Negative
If C(Xn)0 = 1, then ON; otherwise OFF

NOTES:
Attempted repetition with RPL causes an Illegal Procedure Fault.

NOTES:

attempted repetition with rpl causes an illegal procedure fault.
### ORA

**Format:**
Basic Instruction Format (See Figure 2-1).

**Summary:**
\[ C(A)_i \land C(Y)_i \rightarrow C(A)_i \text{ for } i = (0, 1, ..., 35) \]

**Modifications:**
All

**Indicators:**
(Indicators not listed are not affected)

- **Zero:** If \( C(A) = 0 \), then ON; otherwise OFF
- **Negative:** If \( C(A)_0 = 1 \), then ON; otherwise OFF

### ORAQ

**Format:**
Basic Instruction Format (See Figure 2-1).

**Summary:**
\[ C(A)_{i} \land C(Y_{-pair})_{i} \rightarrow C(AQ)_{i} \text{ for } i = (0, 1, ..., 71) \]

**Modifications:**
All except DU, DL, CI, SC, SCR

**Indicators:**
(Indicators not listed are not affected)

- **Zero:** If \( C(AQ) = 0 \), then ON; otherwise OFF
- **Negative:** If \( C(AQ)_0 = 1 \), then ON; otherwise OFF

### ORQ

**Format:**
Basic Instruction Format (See Figure 2-1).

**Summary:**
\[ C(Q)_i \land C(Y)_i \rightarrow C(Q)_i \text{ for } i = (0, 1, ..., 35) \]

**Modifications:**
All

**Indicators:**
(Indicators not listed are not affected)

- **Zero:** If \( C(Q) = 0 \), then ON; otherwise OFF
Negative
If \( C(Q) = 1 \), then ON; otherwise OFF

**ORSA**
OR to Storage A

**FORMAT**
Basic Instruction Format (See Figure 2-1).

**SUMMARY**
\( C(A) i \mid C(Y) i \to C(Y) i \) for \( i = \{0, 1, \ldots, 35\} \)

**MODIFICATIONS**
All except DU, DL, CI, SC, SCR

**INDICATORS**
(Indicators not listed are not affected)

**Zero**
If \( C(Y) = 0 \), then ON; otherwise OFF

**Negative**
If \( C(Y) = 1 \), then ON; otherwise OFF

**NOTES**
Attempted repetition with RPL causes an Illegal Procedure Fault.

**ORSQ**
OR to Storage Q

**FORMAT**
Basic Instruction Format (See Figure 2-1).

**SUMMARY**
\( C(Q) i \mid C(Y) i \to C(Y) i \) for \( i = \{0, 1, \ldots, 35\} \)

**MODIFICATIONS**
All except DU, DL, CI, SC, SCR

**INDICATORS**
(Indicators not listed are not affected)

**Zero**
If \( C(Y) = 0 \), then ON; otherwise OFF

**Negative**
If \( C(Y) = 1 \), then ON; otherwise OFF

**NOTES**
Attempted repetition with RPL causes an Illegal Procedure Fault.
BOOLEAN OR

ORXn
OR to Storage Xn

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
For \( n = 0, 1, \ldots, \) or 7 as determined by operation code
\( C(Xn)_i \mid C(Y)_i \to C(Y)_i \) for \( i = (0, 1, \ldots, 17) \)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)
- Zero: If \( C(Y)_0,17 = 0 \), then ON; otherwise OFF
- Negative: If \( C(Y)_0 = 1 \), then ON; otherwise OFF

NOTES:
Attempted repetition with RPL causes an Illegal Procedure Fault.

ORXn
OR to Xn

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
For \( n = 0, 1, \ldots, \) or 7 as determined by operation code
\( C(Xn)_i \mid C(Y)_i \to C(Xn)_i \) for \( i = (0, 1, \ldots, 17) \)

MODIFICATIONS:
All except CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)
- Zero: If \( C(Xn) = 0 \), then ON; otherwise OFF
- Negative: If \( C(Xn)_0 = 1 \), then ON; otherwise OFF
### Boolean EXCLUSIVE OR

#### ERA

**FORMAT:**
Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
\[ C(A)_i \oplus C(Y)_i \rightarrow C(A)_i \text{ for } i = (0, 1, \ldots, 35) \]

**MODIFICATIONS:**
All

**INDICATORS:**
(Indicators not listed are not affected)

- **Zero:** If \( C(A) = 0 \), then ON; otherwise OFF
- **Negative:** If \( C(A) = 1 \), then ON; otherwise OFF

#### ERAQ

**FORMAT:**
Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
\[ C(AQ)_i \oplus C(Y\text{-pair})_i \rightarrow C(AQ)_i \text{ for } i = (0, 1, \ldots, 71) \]

**MODIFICATIONS:**
All except DU, DL, CI, SC, SCR

**INDICATORS:**
(Indicators not listed are not affected)

- **Zero:** If \( C(AQ) = 0 \), then ON; otherwise OFF
- **Negative:** If \( C(AQ) = 1 \), then ON; otherwise OFF

#### ERQ

**FORMAT:**
Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
\[ C(Q)_i \oplus C(Y)_i \rightarrow C(Q)_i \text{ for } i = (0, 1, \ldots, 35) \]

**MODIFICATIONS:**
All

**INDICATORS:**
(Indicators not listed are not affected)

- **Zero:** If \( C(Q) = 0 \), then ON; otherwise OFF
BOOLEAN EXCLUSIVE OR

ERSA

EXCLUSIVE OR to Storage A

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
C(A)i ⊕ C(Y)i → C(Y)i for i = (0, 1, ..., 35)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

Zero
If C(Y) = 0, then ON; otherwise OFF

Negative
If C(Y) = 1, then ON; otherwise OFF

NOTES:
Attempted repetition with RPL causes an Illegal Procedure Fault.

ERSQ

EXCLUSIVE OR to Storage Q

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
C(Q)i ⊕ C(Y)i → C(Y)i for i = (0, 1, ..., 35)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

Zero
If C(Y) = 0, then ON; otherwise OFF

Negative
If C(Y) = 1, then ON; otherwise OFF

NOTES:
Attempted repetition with RPL causes an Illegal Procedure Fault.
<table>
<thead>
<tr>
<th>ERSXn</th>
<th>EXCLUSIVE OR to Storage Xn</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64n (0)</td>
</tr>
</tbody>
</table>

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** For \( n = 0, 1, \ldots, 7 \) as determined by operation code
\[
C(Xn)i \oplus C(Y)i \rightarrow C(Y)i \quad \text{for} \quad i = (0, 1, \ldots, 17)
\]

**MODIFICATIONS:** All except DU, DL, CI, SC, SCR

**INDICATORS:** (Indicators not listed are not affected)

| Zero | If \( C(Y)0,17 = 0 \), then ON; otherwise OFF |
|      |                                            |

**NOTES:** Attempted repetition with RPL causes an Illegal Procedure Fault.

<table>
<thead>
<tr>
<th>ERXn</th>
<th>EXCLUSIVE OR to Xn</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>66n (0)</td>
</tr>
</tbody>
</table>

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** For \( n = 0, 1, \ldots, 7 \) as determined by operation code
\[
C(Xn)i \oplus C(Y)i \rightarrow C(Xn)i \quad \text{for} \quad i = (0, 1, \ldots, 17)
\]

**MODIFICATIONS:** All except CI, SC, SCR

**INDICATORS:** (Indicators not listed are not affected)

| Zero | If \( C(Xn) = 0 \), then ON; otherwise OFF |
|      |                                            |
| Negative | If \( C(Xn)0 = 1 \), then ON; otherwise OFF |
**Boolean Comparative AND**

**CANA**
Comparative AND with A

**FORMAT:**
Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
\[ C(Z)i = C(A)i \land C(Y)i \text{ for } i = (0, 1, \ldots, 35) \]

**MODIFICATIONS:**
All

**INDICATORS:**
(Indicators not listed are not affected)

- **ZERO:** If \( C(Z) = 0 \), then ON; otherwise OFF
- **Negative:** If \( C(Z)0 = 1 \), then ON; otherwise OFF

**CANAQ**
Comparative AND with AQ

**FORMAT:**
Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
\[ C(Z)i = C(AQ)i \land C(Y\text{-pair})i \text{ for } i = (0, 1, \ldots, 71) \]

**MODIFICATIONS:**
All except DU, DL, CI, SC, SCR

**INDICATORS:**
(Indicators not listed are not affected)

- **ZERO:** If \( C(Z) = 0 \), then ON; otherwise OFF
- **Negative:** If \( C(Z)0 = 1 \), then ON; otherwise OFF

**CANQ**
Comparative AND with Q

**FORMAT:**
Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
\[ C(Z)i = C(Q)i \land C(Y)i \text{ for } i = (0, 1, \ldots, 35) \]

**MODIFICATIONS:**
All

**INDICATORS:**
(Indicators not listed are not affected)

- **ZERO:** If \( C(Z) = 0 \), then ON; otherwise OFF
**BOOLEAN COMPARATIVE AND**

Negative: If C(Z) = 1, then ON; otherwise OFF

**CANXn**

Comparative AND with Xn

**FORMAT:**
Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
For \( n = 0, 1, \ldots, 7 \) as determined by operation code

\[ C(Z)_i = C(Xn)_i \land C(Y)_i \text{ for } i = (0, 1, \ldots, 17) \]

**MODIFICATIONS:**
All except CI, SC, SCR

**INDICATORS:**
(Indicators not listed are not affected)

Zero: If C(Z) = 0, then ON; otherwise OFF

Negative: If C(Z) = 1, then ON; otherwise OFF
BOOLEAN COMPARATIVE NOT

CNAA

Comparative NOT with A

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
\[ C(Z)i = C(A)i \land \neg C(Y)i \text{ for } i = \{0, 1, \ldots, 35\} \]

MODIFICATIONS:
All

INDICATORS:
(Indicators not listed are not affected)

Zero
If \( C(Z) = 0 \), then ON; otherwise OFF

Negative
If \( C(Z) = 1 \), then ON; otherwise OFF

CNAAQ

Comparative NOT with AQ

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
\[ C(Z)i = C(AQ)i \land \neg C(Y-pair)i \text{ for } i = \{0, 1, \ldots, 71\} \]

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

Zero
If \( C(Z) = 0 \), then ON; otherwise OFF

Negative
If \( C(Z) = 1 \), then ON; otherwise OFF

CNAQ

Comparative NOT with Q

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
\[ C(Z)i = C(Q)i \land \neg C(Y)i \text{ for } i = \{0, 1, \ldots, 35\} \]

MODIFICATIONS:
All

INDICATORS:
(Indicators not listed are not affected)

Zero
If \( C(Z) = 0 \), then ON; otherwise OFF

Negative
If \( C(Z) = 1 \), then ON; otherwise OFF
BOOLEAN COMPARATIVE NOT

\[ \text{CNAX}_n \quad \text{Comparative NOT with } X_n \quad 20n(0) \]

**FORMAT:**
Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
For \( n = 0, 1, \ldots, \) or 7 as determined by operation code
\[ C(Z)i = C(Xn)i \land \neg C(Y)i \quad \text{for } i = (0, 1, \ldots, 17) \]

**MODIFICATIONS:**
All except CI, SC, SCR

**INDICATORS:**
(Indicators not listed are not affected)

- **Zero**
  If \( C(Z) = 0 \), then ON; otherwise OFF

- **Negative**
  If \( C(Z)0 = 1 \), then ON; otherwise OFF
FLOATING POINT DATA MOVEMENT LOAD

"FLOATING-POINT-ARITHMETIC INSTRUCTIONS"

"Floating-Point Data Movement Load"

DFLD

<table>
<thead>
<tr>
<th>Description</th>
<th>Instruction</th>
<th>Hex Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double Precision Floating Load</td>
<td>FLD</td>
<td>433 (0)</td>
</tr>
</tbody>
</table>

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
- CY, pair 0, 7 -> C(E)
- CY, pair 8, 71 -> C(AQ) 0, 63
- 00...0 -> C(AQ) 64, 71

**MODIFICATIONS:** All except DU, DL, CI, SC, SCR

**INDICATORS:** (Indicators not listed are not affected)
- Zero: If C(AQ) = 0, then ON; otherwise OFF
- Negative: If C(AQ) = 1, then ON; otherwise OFF

FLD

<table>
<thead>
<tr>
<th>Description</th>
<th>Instruction</th>
<th>Hex Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating Load</td>
<td>FLD</td>
<td>431 (0)</td>
</tr>
</tbody>
</table>

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
- CY, pair 0, 7 -> C(E)
- CY, pair 8, 35 -> C(AQ) 0, 27
- 00...0 -> C(AQ) 30, 71

**MODIFICATIONS:** All except CI, SC, SCR

**INDICATORS:** (Indicators not listed are not affected)
- Zero: If C(AQ) = 0, then ON; otherwise OFF
- Negative: If C(AQ) = 1, then ON; otherwise OFF
"Floating-Point Data Movement Store"

DFST

Double Precision Floating Store

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
C(E) → C(Y-pair)0,7
C(AQ)0,63 → C(Y-pair)8,71

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
None affected

NOTES:
Attempted repetition with RPL causes an Illegal Procedure Fault.

DFSTR

Double Precision Floating Store Rounded

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
C(EAQ) rounded → C(Y-pair)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

Zero
If C(Y-pair) = floating point 0, then ON; otherwise OFF

Negative
If C(Y-pair)8 = 1, then ON; otherwise OFF

Exponent Overflow
If exponent is greater than +127, then ON; otherwise OFF

Exponent Underflow
If exponent is less than -128, then ON; otherwise OFF

NOTES:
The DFSTR instruction performs a double precision true round and normalization on C(EAQ) as it is stored.

The definition of true round is located under the description of the Floating Round (FRD) instruction.

The definition of normalization is located under the description of the Floating Normalize (FNO) instruction.

Except for the precision of the stored result, the DFSTR instruction is identical to the FSTR instruction.
FLOATING POINT DATA MOVEMENT STORE

**FST**  
Floating Store  455 (0)

**FORMAT:**  
Basic Instruction Format (See Figure 2-1).

**SUMMARY:**  
C(E) -> C(Y)0,7  
C(A)0,27 -> C(Y)8,35

**MODIFICATIONS:**  
All except DU, DL, CI, SC, SCR

**INDICATORS:**  
None affected

**NOTES:**  
Attempted repetition with RPL causes an Illegal Procedure Fault.

**FSTR**  
Floating Store Rounded  470 (0)

**FORMAT:**  
Basic Instruction Format (See Figure 2-1).

**SUMMARY:**  
C(EAQ) rounded -> C(Y)

**MODIFICATIONS:**  
All except DU, DL, CI, SC, SCR

**INDICATORS:**  
(Indicators not listed are not affected)

- **Zero:**  
  If C(Y) = floating point 0, then ON; otherwise OFF
- **Negative:**  
  If C(Y)8 = 1, then ON; otherwise OFF
- **Exponent Overflow:**  
  If exponent is greater than +127, then ON; otherwise OFF
- **Exponent Underflow:**  
  If exponent is less than -128, then ON; otherwise OFF

**NOTES:**  
The FSTR instruction performs a true round and normalization on C(EAQ) as it is stored.

The definition of true round is located under the description of the Floating Round (FRO) instruction.

The definition of normalization is located under the description of the Floating Normalize (FNO) instruction.
Steps in the execution may be thought of as follows:

- Execute FN0
- Execute FST
- Restore C(EAQ) to original values.

Attempted repetition with RPL causes an Illegal Procedure Fault.
FLOATING POINT ADDITION

**Floating Point Addition**

**DFAD**

*Double Precision Floating Add*

**FORMAT:**

Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

\((C(\text{EAQ}) + C(\text{Y\text{-}pair}))\text{ normalized} \rightarrow C(\text{EAQ})\)

**MODIFICATIONS:**

All except DU, DL, CI, SC, SCR

**INDICATORS:**

(Indicators not listed are not affected)

- **Zero**
  
  If \(C(\text{AQ}) = 0\), then ON; otherwise OFF

- **Negative**
  
  If \(C(\text{AQ}) = 1\), then ON; otherwise OFF

- **Exponent Overflow**
  
  If exponent is greater than +127, then ON; otherwise OFF

- **Exponent Underflow**
  
  If exponent is less than -128, then ON; otherwise OFF

- **Carry**
  
  If a carry out of AQ0 is generated, then ON; otherwise OFF

**NOTES:**

The DFAD instruction may be thought of as a Double Precision Unnormalized Floating Add (DUFA) instruction followed by a Floating Normalize (FNO) instruction.

The definition of normalization is located under the description of the Floating Normalize (FNO) instruction.

**DUFA**

*Double Precision Unnormalized Floating Add*

**FORMAT:**

Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

\(C(\text{EAQ}) + C(\text{Y\text{-}pair}) \rightarrow C(\text{EAQ})\)

**MODIFICATIONS:**

All except DU, DL, CI, SC, SCR
FLOATING POINT ADDITION

INDICATORS: (Indicators not listed are not affected)

Zero
Negative
Exponent Overflow
Exponent Underflow
Carry

NOTES: Except for the precision of the mantissa of the operand from main store, the DUFA instruction is identical to the UFA instruction.

FAD

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: (C(EAQ) + C(Y)) normalized -> C(EAQ)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero
Negative
Exponent Overflow
Exponent Underflow
Carry

NOTES: The FAD instruction may be thought of as an Unnormalized Floating Add (UFA) instruction followed by a Floating Normalize (FNO) instruction.

The definition of normalization is located under the description of the Floating Normalize (FNO) instruction.

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975
**FLOATING POINT ADDITION**

**UFA**

Unnormalized Floating Add

**FORMAT:**

Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

C(EAQ) + C(Y) -> C(EAQ)

**MODIFICATIONS:**

All except CI, SC, SCR

**INDICATORS:**

(Indicators not listed are not affected)

- **Zero**
  - If $(E_AQ) = 0$, then ON; otherwise OFF

- **Negative**
  - If $(E_AQ)0 = 1$, then ON; otherwise OFF

- **Exponent Overflow**
  - If exponent is greater than +127, then ON; otherwise OFF

- **Exponent Underflow**
  - If exponent is less than -128, then ON; otherwise OFF

- **Carry**
  - If a carry out of $AQO$ is generated, then ON; otherwise OFF

**NOTES:**

The UFA instruction is executed as follows:

The mantissas are aligned by shifting the mantissa of the operand having the algebraically smaller exponent to the right the number of places equal to the absolute value of the difference in the two exponents. Bits shifted beyond the bit position equivalent to $AQ71$ are lost.

The algebraically larger exponent replaces $C(E)$.

The sum of the mantissas replaces $C(AQ)$.

If an overflow occurs during addition, then:

- $C(AQ)$ are shifted one place to the right.
- $C(AQ)0$ is inverted to restore the sign.
- $C(E)$ is increased by one.
DOUBLE PRECISION FLOATING SUBTRACT

**DFSBS**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FORMAT</strong></td>
<td>Basic Instruction Format (See Figure 2-1).</td>
</tr>
<tr>
<td><strong>SUMMARY</strong></td>
<td>((C(AQ) - C(Y-pair))) normalized (\rightarrow C(EAQ))</td>
</tr>
<tr>
<td><strong>MODIFICATIONS</strong></td>
<td>All except DU, DL, CI, SC, SCR</td>
</tr>
<tr>
<td><strong>INDICATORS</strong></td>
<td>(Indicators not listed are not affected)</td>
</tr>
</tbody>
</table>

- **Zero**: If \(C(AQ) = 0\), then ON; otherwise OFF
- **Negative**: If \(C(AQ) = 1\), then ON; otherwise OFF
- **Exponent Overflow**: If exponent is greater than +127, then ON; otherwise OFF
- **Exponent Underflow**: If exponent is less than -128, then ON; otherwise OFF
- **Carry**: If a carry out of AQ0 is generated, then ON; otherwise OFF

**NOTES**: The DFSBS instruction is identical to the Double Precision Floating Add (DFAD) instruction with the exception that the 2's complement of the mantissa of the operand from main store is used.

DOUBLE PRECISION UNNORMALIZED FLOATING SUBTRACT

**DUFS**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FORMAT</strong></td>
<td>Basic Instruction Format (See Figure 2-1).</td>
</tr>
<tr>
<td><strong>SUMMARY</strong></td>
<td>((C(EAQ) - C(Y-pair) \rightarrow C(EAQ))</td>
</tr>
<tr>
<td><strong>MODIFICATIONS</strong></td>
<td>All except DU, DL, CI, SC, SCR</td>
</tr>
<tr>
<td><strong>INDICATORS</strong></td>
<td>(Indicators not listed are not affected)</td>
</tr>
</tbody>
</table>

- **Zero**: If \(C(AQ) = 0\), then ON; otherwise OFF
- **Negative**: If \(C(AQ) = 1\), then ON; otherwise OFF
- **Exponent Overflow**: If exponent is greater than +127, then ON; otherwise OFF
- **Exponent Underflow**: If exponent is less than -128, then ON; otherwise OFF
FLOATING POINT SUBTRACTION

**FSB**
Floating Subtract

**FORMAT:**
Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
\((C(\text{EAQ}) - C(Y))\) normalized \(\rightarrow C(\text{EAQ})\)

**MODIFICATIONS:**
All except CI, SC, SCR

**INDICATORS:**
(Indicators not listed are not affected)

- **Zero:** If \(C(\text{AQ}) = 0\), then ON; otherwise OFF
- **Negative:** If \(C(\text{AQ})0 = 1\), then ON; otherwise OFF
- **Exponent Overflow:** If exponent is greater than +127, then ON; otherwise OFF
- **Exponent Underflow:** If exponent is less than -128, then ON; otherwise OFF
- **Carry:** If a carry out of AQ0 is generated, then ON; otherwise OFF

**NOTES:**
The FSB instruction may be thought of as an Unnormalized Floating Subtract (UFS) instruction followed by a Floating Normalize (FNO) instruction.

The definition of normalization is located under the description of the Floating Normalize (FNO) instruction.

**UFS**
Unnormalized Floating Subtract

**FORMAT:**
Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
\((C(\text{EQ}) - C(Y)) \rightarrow C(\text{EQ})\)

**MODIFICATIONS:**
All except CI, SC, SCR
### Floating Point Subtraction

**INDICATORS:** (Indicators not listed are not affected)

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>If C(AQ) = 0, then ON; otherwise OFF</td>
</tr>
<tr>
<td>Negative</td>
<td>If C(AQ)0 = 1, then ON; otherwise OFF</td>
</tr>
<tr>
<td>Exponent</td>
<td>If exponent is greater than +127, then ON; otherwise OFF</td>
</tr>
<tr>
<td>Overflow</td>
<td></td>
</tr>
<tr>
<td>Underflow</td>
<td>If exponent is less than -128, then ON; otherwise OFF</td>
</tr>
<tr>
<td>Carry</td>
<td>If a carry out of AQ0 is generated, then ON; otherwise OFF</td>
</tr>
</tbody>
</table>

**NOTES:**

The UFS instruction is identical to the Unnormalized Floating Add (UFA) instruction with the exception that the 2's complement of the mantissa of the operand from main store is used.
FLOATING POINT MULTIPLICATION

"Floating-Point Multiplication"

DFMP

Double Precision Floating Multiply

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
\( (C\text{EAQ}) \times (C\text{Y-pair}) \rightarrow C\text{EAQ} \)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

- **Zero**: If \( C\text{AQ} = 0 \), then ON; otherwise OFF
- **Negative**: If \( C\text{AQ} = 1 \), then ON; otherwise OFF
- **Exponent Overflow**: If exponent is greater than +127, then ON; otherwise OFF
- **Exponent Underflow**: If exponent is less than -128, then ON; otherwise OFF

NOTES:
The DFMP instruction may be thought of as a Double Precision Unnormalized Floating Multiply (DUFM) instruction followed by a Floating Normalize (FNO) instruction.

The definition of normalization is located under the description of the Floating Normalize (FNO) instruction.

DUFM

Double Precision Unnormalized Floating Multiply

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
\( C\text{EAQ} \times C\text{Y-pair} \rightarrow C\text{EAQ} \)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

- **Zero**: If \( C\text{AQ} = 0 \), then ON; otherwise OFF
- **Negative**: If \( C\text{AQ} = 1 \), then ON; otherwise OFF
- **Exponent Overflow**: If exponent is greater than +127, then ON; otherwise OFF
- **Exponent Underflow**: If exponent is less than -128, then ON; otherwise OFF
FLOATING POINT MULTIPLICATION

NOTES: Except for the precision of the mantissa of the operand from main store, the DUFM instruction is identical to the Unnormalized Floating Multiply (UFM) instruction.

FMP
Floating Multiply

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: \( (C(EAQ) \times C(Y)) \text{ normalized } \rightarrow C(EAQ) \)

MODIFICATIONS: All except CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero: If \( C(AQ) = 0 \), then ON; otherwise OFF

Negative: If \( C(AQ) = 1 \), then ON; otherwise OFF

Exponent Overflow: If exponent is greater than +127, then ON; otherwise OFF

Exponent Underflow: If exponent is less than -128, then ON; otherwise OFF

NOTES: The FMP instruction may be thought of as an Unnormalized Floating Multiply (UFM) instruction followed by a Floating Normalize (FNO) instruction.

The definition of normalization is located under the description of the Floating Normalize (FNO) instruction.

UFM
Unnormalized Floating Multiply

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: \( C(EAQ) \times C(Y) \rightarrow C(EAQ) \)

MODIFICATIONS: All except CI, SC, SCR
FLOATING POINT MULTIPLICATION

INDICATORS:

(Indicators not listed are not affected)

Zero
If $C(AQ) = 0$, then ON; otherwise OFF

Negative
If $C(AQ) = 1$, then ON; otherwise OFF

Exponent Overflow
If exponent is greater than +127, then ON; otherwise OFF

Exponent Underflow
If exponent is less than -128, then ON; otherwise OFF

NOTES:

The UFM instruction is executed as follows:

$C(E) + C(Y)0,7 -> C(E)$

$(C(AQ) \times C(Y)8,35)0,71 -> C(AQ)$

A normalization is performed only in the case of both factor mantissas being $100...0$ which is the 2's complement approximation to the decimal value $-1.0$.

The definition of normalization is located under the description of the Floating Normalize (FNO) instruction.
Floating Point Division

DFDI

Double Precision Floating Divide Inverted 527 (0)

FORMAT:

Basic Instruction Format (See Figure 2-1).

SUMMARY:

C(Y-pair) / C(EAQ) -> C(EAQ)

MODIFICATIONS:

All except DU, DL, CI, SC, SCR

INDICATORS:

(Indicators not listed are not affected)

If division takes place

Zero

If C(AQ) = 0, then ON; otherwise OFF

Negative

If C(AQ) = 1, then ON; otherwise OFF

Exponent Overflow

If exponent is greater than +127, then ON; otherwise OFF

Exponent Underflow

If exponent is less than -128, then ON; otherwise OFF

If no division takes place

If divisor mantissa = 0, then ON; otherwise OFF

If dividend < 0, then ON; otherwise OFF

If exponent is greater than +127, then ON; otherwise OFF

If exponent is less than -128, then ON; otherwise OFF

NOTES:

Except for the interchange of the roles of the operands, the execution of the DFDI instruction is identical to the execution of the Double Precision Floating Divide (DFDV) instruction.

If the divisor mantissa C(AQ) is zero, the division does not take place. Instead, a Divide Check Fault occurs and all registers remain unchanged.

DFDV

Double Precision Floating Divide 567 (0)

FORMAT:

Basic Instruction Format (See Figure 2-1).

SUMMARY:

C(EAQ) / C(Y-pair) -> C(EAQ)

MODIFICATIONS:

All except DU, DL, CI, SC, SCR
FLOATING POINT DIVISION

INDICATORS:

<table>
<thead>
<tr>
<th>Indicator</th>
<th>If division takes place</th>
<th>If no division takes place</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>If C(AQ) = 0, then ON;</td>
<td>If divisor mantissa = 0,</td>
</tr>
<tr>
<td></td>
<td>otherwise OFF</td>
<td>then ON; otherwise OFF</td>
</tr>
<tr>
<td>Negative</td>
<td>If C(AQ)0 = 1, then ON;</td>
<td>If dividend &lt; 0, then ON;</td>
</tr>
<tr>
<td></td>
<td>otherwise OFF</td>
<td>otherwise OFF</td>
</tr>
<tr>
<td>Exponent</td>
<td>If exponent is greater</td>
<td>If exponent is less than</td>
</tr>
<tr>
<td>Overflow</td>
<td>than +127, then ON;</td>
<td>-128, then ON; otherwise</td>
</tr>
<tr>
<td>Underflow</td>
<td>otherwise OFF</td>
<td>OFF</td>
</tr>
</tbody>
</table>

NOTES:
The DFDV instruction is executed as follows:
The dividend mantissa C(AQ) is shifted right and the dividend exponent C(E) increased accordingly until IC(AQ)0,631 < IC(Y-pair)8,711.

\[
C(E) - C(Y\text{-pair})0,7 \rightarrow C(E) \\
C(AQ) / C(Y\text{-pair})8,71 \rightarrow C(AQ)0,63 \\
00...0 \rightarrow C(Q)64,71
\]

If the divisor mantissa C(Y-pair)8,71 is zero, the division does not take place. Instead, a Divide Check fault occurs, C(AQ) contains the dividend magnitude, and the Negative indicator reflects the dividend sign.

FDI Floating Divide Inverted 525 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(Y) / C(EAQ) \rightarrow C(EA)

MODIFICATIONS: All except CI, SC, SCR
FLOATING POINT DIVISION

INDICATORS:

<table>
<thead>
<tr>
<th>Indicator</th>
<th>If division takes place</th>
<th>If no division takes place</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>If ( C(A) = 0 ), then ON; otherwise OFF</td>
<td>If divisor mantissa = 0, then ON; otherwise OFF</td>
</tr>
<tr>
<td>Negative</td>
<td>If ( C(A) = 0 ), then ON; otherwise OFF</td>
<td>If dividend &lt; 0, then ON; otherwise OFF</td>
</tr>
<tr>
<td>Exponent Overflow</td>
<td>If exponent is greater than +127, then ON; otherwise OFF</td>
<td></td>
</tr>
<tr>
<td>Exponent Underflow</td>
<td>If exponent is less than -128, then ON; otherwise OFF</td>
<td></td>
</tr>
</tbody>
</table>

NOTES:

Except for the interchange of roles of the operands, the execution of the \( FDV \) instruction is identical to the execution of the Floating Divide (FDV) instruction.

If the divisor mantissa \( C(AQ) \) is zero, the division does not take place. Instead, a Divide-Check fault occurs and all the registers remain unchanged.

FDV Floating Divide 565 (0)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY
\[ C(EAQ) / C(Y) \rightarrow C(EA) \]
\[ 00...0 \rightarrow C(Q) \]

MODIFICATIONS:
All except CI, SC, SCR

INDICATORS:

<table>
<thead>
<tr>
<th>Indicator</th>
<th>If division takes place</th>
<th>If no division takes place</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>If ( C(A) = 0 ), then ON; otherwise OFF</td>
<td>If divisor mantissa = 0, then ON; otherwise OFF</td>
</tr>
<tr>
<td>Negative</td>
<td>If ( C(A) = 1 ), then ON; otherwise OFF</td>
<td>If dividend &lt; 0, then ON;</td>
</tr>
<tr>
<td>Exponent Overflow</td>
<td>If exponent is greater than +127, then ON; otherwise OFF</td>
<td>otherwise OFF</td>
</tr>
<tr>
<td>Exponent Underflow</td>
<td>If exponent is less than -128, then ON; otherwise OFF</td>
<td></td>
</tr>
</tbody>
</table>

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975 2-95 AL39
The FDV instruction is executed as follows:

The dividend mantissa C(AQ) is shifted right and the
dividend exponent C(E) increased accordingly until
IC(AQ)0,271 < IC(Y)8,351.

C(E) - C(Y)0,7 -> C(E)
C(AQ) / C(Y)8,35 -> C(A)
00...0 -> C(Q)

If the divisor mantissa C(Y)8,35 is zero, the division
does not take place. Instead, a Divide Check fault
occurs. C(AQ) contains the dividend magnitude, and the
Negative indicator reflects the dividend sign.
**FLOATING POINT NEGATE**

**"Floating-Point Negate"**

<table>
<thead>
<tr>
<th>FNEG</th>
<th>Floating Negate</th>
<th>513 (0)</th>
</tr>
</thead>
</table>

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** \(-C(AQ)\) normalized \(\rightarrow\) \(C(AQ)\)

**MODIFICATIONS:** All, but none affect instruction execution.

**INDICATORS:** (Indicators not listed are not affected)

- **Zero**
  - If \(C(AQ) = 0\), then ON; otherwise OFF

- **Negative**
  - If \(C(AQ) = 1\), then ON; otherwise OFF

- **Exponent Overflow**
  - If exponent is greater than \(+127\), then ON; otherwise OFF

- **Exponent Underflow**
  - If exponent is less than \(-128\), then ON; otherwise OFF

**NOTES:**

This instruction changes the number in \(C(EAQ)\) to its normalized negative (if \(C(AQ) \neq 0\)). The operation is executed by first forming the two's complement of \(C(AQ)\), and then normalizing \(C(EAQ)\).

Even if originally \(C(EAQ)\) were normalized, an exponent overflow can still occur, namely when \(C(E) = +127\) and \(C(AQ) = 100\ldots0\) which is the 2's complement approximation for the decimal value \(-1.0\).

The definition of normalization may be found under the description of the Floating Normalize (FNO) instruction.

Attempted repetition with RPL causes an Illegal Procedure Fault.
FLOATING POINT NORMALIZE

"Floating Point Normalize"

**FNO**

**FORMAT**: Basic Instruction Format (See Figure 2-1).

**SUMMARY**: C(EAQ) normalized -> C(EAQ)

**MODIFICATIONS**: All, but none affect instruction execution.

**INDICATORS**: (Indicators not listed are not affected)

- **Zero**: If C(EAQ) = floating point 0, then ON; otherwise OFF
- **Negative**: If C(AQ)0 = 1, then ON; otherwise OFF
- **Exponent Overflow**: If exponent is greater than +127, then ON; otherwise OFF
- **Exponent Underflow**: If exponent is less than -128, then ON otherwise OFF
- **Overflow**: Set OFF

**NOTES**: The FNO instruction normalizes the number in C(EAQ) if C(AQ)0 ≠ 0 and the Overflow indicator is OFF.

A normalized floating number is defined as one whose mantissa lies in the interval [0.5, 1.0] such that

\[ 0.5 \leq |C(EAQ)| < 1.0 \]

which, in turn, requires that C(AQ)0 ≠ C(AQ)1.

If the Overflow indicator is ON, then C(AQ) is shifted one place to the right, C(AQ)0 is inverted to reconstitute the actual sign, and the Overflow indicator is set OFF.

Normalization is performed by shifting C(AQ)1,71 one place to the left and reducing C(E) by 1, repeatedly, until the conditions for C(AQ)0 and C(AQ)1 are met. Bits shifted out of AQ1 are lost.

If C(AQ) = 0, then C(E) is set to -128 and the Zero indicator is set ON.

The FNO instruction can be used to correct overflows that occur with fixed point numbers.

Attempted repetition with RPL causes an Illegal Procedure Fault.
# Floating-Point Round

<table>
<thead>
<tr>
<th>DFRD</th>
<th>Double Precision Floating Round</th>
<th>473 (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FORMAT:</strong></td>
<td>Basic Instruction Format (See Figure 2-1).</td>
<td></td>
</tr>
<tr>
<td><strong>SUMMARY:</strong></td>
<td>C(EAQ) rounded to 64 bits → C(EAQ)</td>
<td></td>
</tr>
<tr>
<td><strong>MODIFICATIONS:</strong></td>
<td>All, but none affect instruction execution.</td>
<td></td>
</tr>
<tr>
<td><strong>INDICATORS:</strong></td>
<td>(Indicators not listed are not affected)</td>
<td></td>
</tr>
<tr>
<td>Zero</td>
<td>If C(EAQ) = floating point 0, then ON; otherwise OFF</td>
<td></td>
</tr>
<tr>
<td>Negative</td>
<td>If C(AQ)0 = 1, then ON; otherwise OFF</td>
<td></td>
</tr>
<tr>
<td>Exponent Overflow</td>
<td>If exponent is greater than +127, then ON; otherwise OFF</td>
<td></td>
</tr>
<tr>
<td>Exponent Underflow</td>
<td>If exponent is less than -128, then ON; otherwise OFF</td>
<td></td>
</tr>
</tbody>
</table>

**NOTES:**
The DFRD instruction is identical to the Floating Round (FRD) instruction except that the rounding constant used is \((11\ldots1)65,71\) instead of \((11\ldots1)29,71\).

Attempted repetition with RPL causes an Illegal Procedure Fault.

<table>
<thead>
<tr>
<th>FRD</th>
<th>Floating Round</th>
<th>471 (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FORMAT:</strong></td>
<td>Basic Instruction Format (See Figure 2-1).</td>
<td></td>
</tr>
<tr>
<td><strong>SUMMARY:</strong></td>
<td>C(EAQ) rounded to 28 bits → C(EAQ)</td>
<td></td>
</tr>
<tr>
<td><strong>MODIFICATIONS:</strong></td>
<td>All, but none affect instruction execution.</td>
<td></td>
</tr>
<tr>
<td><strong>INDICATORS:</strong></td>
<td>(Indicators not listed are not affected)</td>
<td></td>
</tr>
<tr>
<td>Zero</td>
<td>If C(EAQ) = floating point 0, then ON; otherwise OFF</td>
<td></td>
</tr>
<tr>
<td>Negative</td>
<td>If C(AQ)0 = 1 then ON; otherwise OFF</td>
<td></td>
</tr>
<tr>
<td>Exponent Overflow</td>
<td>If exponent is greater than +127, then ON; otherwise OFF</td>
<td></td>
</tr>
<tr>
<td>Exponent Underflow</td>
<td>If exponent is less than -128, then ON; otherwise OFF</td>
<td></td>
</tr>
</tbody>
</table>
FLOATING POINT ROUND

NOTES:
If \( C(AQ) \neq 0 \), the FRD instruction performs a true round to a precision of 28 bits and a normalization on \( C(EAQ) \).

A true round is a rounding operation such that the sum of the result of applying the operation to two numbers of equal magnitude but opposite sign is exactly zero.

The FRD instruction is executed as follows:
\[
C(AQ) + (11...1)29,71 \rightarrow C(AQ)
\]
If \( C(AQ) = 0 \), then a carry is added at \( AQ71 \)
If overflow occurs, \( C(AQ) \) is shifted one place to the right and \( C(E) \) is increased by 1.
If overflow does not occur, \( C(EAQ) \) is normalized.
If \( C(AQ) = 0 \), \( C(E) \) is set to -128 and the Zero indicator is set ON.
Attempted repetition with RPL causes an Illegal Procedure Fault.
DFCMG

Double Precision Floating Compare Magnitude

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
IC(E,AQO,63) || IC(Y-pair)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

Zero
If IC(E,AQO,63) = IC(Y-pair), then ON; otherwise OFF

Negative
If IC(E,AQO,63) < IC(Y-pair), then ON; otherwise OFF

NOTES:
The DFCMG instruction is identical to the Double Precision Floating Compare (DFCMP) instruction except that the magnitudes of the mantissas are compared instead of the algebraic values.

DFCMP

Double Precision Floating Compare

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
C(E,AQO,63) || C(Y-pair)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

Zero
If C(E,AQO,63) = C(Y-pair), then ON; otherwise OFF

Negative
If C(E,AQO,63) < C(Y-pair), then ON; otherwise OFF

NOTES:
The DFCMP instruction is identical to the Floating Compare (FCHP) instruction except for the precision of the mantissas actually compared.
FLOATING POINT COMPARE

FCMG  Floating Compare Magnitude

FORMAT: Basic Instruction Format (See Figure 2-1).
SUMMARY: IC(E, AQO, 27) I I IC(Y) I
MODIFICATIONS: All except CI, SC, SCR
INDICATORS: (Indicators not listed are not affected)

Zero  If IC(E, AQO, 27) I = IC(Y) I, then ON; otherwise OFF
Negative  If IC(E, AQO, 27) I < IC(Y) I, then ON; otherwise OFF

NOTES: The FCMG instruction is identical to the Floating Compare (FCMP) instruction except that the magnitudes of the mantissas are compared instead of the algebraic values.

FCMP  Floating Compare

FORMAT: Basic Instruction Format (See Figure 2-1).
SUMMARY: IC(E, AQO, 27) I I IC(Y) I
MODIFICATIONS: All except CI, SC, SCR
INDICATORS: (Indicators not listed are not affected)

Zero  If IC(E, AQO, 27) I = IC(Y) I, then ON; otherwise OFF
Negative  If IC(E, AQO, 27) I < IC(Y) I, then ON; otherwise OFF

NOTES: The FCMP instruction is executed as follows:
The mantissas are aligned by shifting the mantissa of the operand with the algebraically smaller exponent to the right the number of places equal to the difference in the two exponents.

The aligned mantissas are compared and the indicators set accordingly.
## Floating-Point Miscellaneous

### ADE
Add to Exponent

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** 
\[ C(E) + C(Y)0.7 \rightarrow C(E) \]

**MODIFICATIONS:** All except CI, SC, SCR

**INDICATORS:** (Indicators not listed are not affected)

- **Zero:** Set OFF
- **Negative:** Set OFF
- **Exponent Overflow:** If exponent is greater than +127, then ON; otherwise OFF
- **Exponent Underflow:** If exponent is less than -128, then ON; otherwise OFF

### FSZN
Floating Set Zero and Negative Indicators

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** Set indicators according to C(Y)

**MODIFICATIONS:** All except CI, SC, SCR

**INDICATORS:** (Indicators not listed are not affected)

- **Zero:** If \( C(Y)8,35 = 0 \), then ON; otherwise OFF
- **Negative:** If \( C(Y)8 = 1 \), then ON; otherwise OFF

### LDE
Load Exponent

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** 
\[ C(Y)0.7 \rightarrow C(E) \]

**MODIFICATIONS:** All except CI, SC, SCR

---

*REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975*
FLOATING POINT MISCELLANEOUS

INDICATORS: (Indicators not listed are not affected)

Zero Set OFF
Negative Set OFF

STE Store Exponent

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(E) -> C(Y)0,7
00...0 -> C(Y)8,17

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

October, 1975

REVIEW DRAFT
SUBJECT TO CHANGE
2-104
AL39
TRANSFER INSTRUCTIONS

CALL6 Call (Using PR6 and PR7)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If C(TPR.TR) < C(PPR.PRR) then
C(USSR.STACK) = C(TPR.TR) -> C(PR7.SNR)

If C(TPR.TR) = C(PPR.PRR) then
C(PR6.SNR) -> C(PR7.SNR)

C(TPR.TR) -> C(PR7.RNR)

If C(TPR.TR) = 0 then
C(SDH.P) -> C(PPR.P);
otherwise 0 -> C(PPR.P)

00...0 -> C(PR7.HORDNO)

00...0 -> C(PR7.BITNO)

C(TPR.TR) -> C(PPR.PRR)

C(TPR.TSR) -> C(PPR.PSR)

C(TPR.CA) -> C(PPR.IC)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: If C(TPR.TR) > C(PPR.PRR), an Access Violation Fault, Outward Call, occurs and the CALL6 instruction is not executed.

If the CALL6 instruction is executed with the Processor in Absolute Mode with bit 29 of the instruction word equal to zero and without indirection through an ITP or ITS pair, then...

the Appending Mode is entered for the address preparation of the CALL6 operand address and is retained if the instruction executes successfully, and...

the Effective Segment Number generated for the SDW fetch and subsequent loading into C(TPR.TSR) is equal to C(PPR.PSR) and may be undefined in Absolute Mode, and...

the Effective Ring Number loaded into C(TPR.TR) prior to the SDW fetch is equal to C(PPR.PRR) (which is 0 in Absolute Mode) implying that the Access Violation checks for Outward Call and Bad Outward Call are ineffective and that an Access Violation, Out of Call Brackets will occur if C(SDW.R1) ≠ 0.
TRANSFER

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

RET

Return

630 (0)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
C(Y)0,17 -> C(PPR.IC)
C(Y)18,31 -> C(IR)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)
Parity Mask
If C(Y)27 = 1, and the Processor is in Absolute or Privileged Mode, then ON; otherwise JFF. This indicator is not affected in the Normal or BAR modes.

Not BAR Mode
Cannot be changed by the RET instruction

Multimword Instruction Fault
If C(Y)30 = 1, and the Processor is in Absolute or Privileged mode, then ON; otherwise JFF. This indicator is not affected in Normal or BAR modes.

Absolute Mode
Cannot be changed by the RET instruction

All Other Indicators
If corresponding bit in C(Y) is 1, then ON; otherwise, OFF

NOTES:
The relation between C(Y)18,31 and the indicators is given in Table 2-5.

The Tally Runout indicator reflects C(Y)25 regardless of what address modification is performed on the RET instruction for tally operations.

The RET instruction may be thought of as a Load Indicators (LDI) instruction followed by a transfer to location C(Y)0,17.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
RTCD

Return Control Double

610 (0)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
C(Y-pair)3,17 -> C(PPR,PSR)

Maximum of
C(Y-pair)18,20; C(TPR,TRR); C(SDW,RI) -> C(PPR,PRR)

C(Y-pair)36,53 -> C(PPR,IC)

If C(PPR,PRR) = 0 then C(SDW,P) -> C(PPR,P); otherwise 0 -> C(PPR,P)

C(PPR,PRR) -> C(PRn,RNR) for n = (0, 1, ..., 7)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
None affected

NOTES:
The hardware assumes that C(Y)17 = 0; no check is made.

If an access violation occurs when fetching the SDW for location Y, the C(PPR,PSR) and C(PPR,PRR) are not altered.

If the RTCD instruction is executed with the Processor in Absolute Mode with bit 29 of the instruction word equal to 0 and without indirection through an ITP or ITS pair, then...

the Appending Mode is entered for the address preparation of the RTCD operand address and is retained if the instruction executes successfully, and...

the Effective Segment Number generated for the SDW fetch and subsequent loading into C(TPR,SR) is equal to C(PPR,PSR) and may be undefined in Absolute Mode, and...

the Effective Ring Number loaded into C(TPR,RR) prior to the SDW fetch is equal to C(PPR,PRR) (which is 0 in Absolute Mode) implying that control is always transferred into Ring 0.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
TRANSFER

TEO

Transfer On Exponent Overflow

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
If Exponent Overflow indicator ON then
C(TPR.CA) \rightarrow C(PPR.IC)
C(TPR.TSR) \rightarrow C(PPR.PSR)
otherwise, no change to C(PPR)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

Exponent
Overflow
Set OFF

NOTES:
Attempted repetition with RPT, RPD, or RPL causes an
Illegal Procedure Fault.

TEU

Transfer on Exponent Underflow

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
If Exponent Underflow indicator ON then
C(TPR.CA) \rightarrow C(PPR.IC)
C(TPR.TSR) \rightarrow C(PPR.PSR)
otherwise, no change to C(PPR)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
(Indicators not listed are not affected)

Exponent
Underflow
Set OFF

NOTES:
Attempted repetition with RPT, RPD, or RPL causes an
Illegal Procedure Fault.
TRANSFER

TMI
Transfer on Minus
604 (0)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
If Negative indicator ON then
C(TPR,CA) -> C(PPR,IC)
C(TPR,TSR) -> C(PPR,PSR)
otherwise, no change to C(PPR)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
None affected

NOTES:
Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TMOZ
Transfer On Minus or Zero
604 (1)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
If Negative or Zero indicator ON then
C(TPR,CA) -> C(PPR,IC)
C(TPR,TSR) -> C(PPR,PSR)
otherwise, no change to C(PPR)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
None affected

NOTES:
Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TNC
Transfer on No Carry
602 (0)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
If Carry Indicator OFF then
C(TPR,CA) -> C(PPR,IC)
TRANSFER

C(TPR.TSR) -> C(PPR.PSR)
otherwise, no change to C(PPR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TNZ

Transfer On Not Zero 601 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If Zero indicator OFF then
C(TPR.CA) -> C(PPR.IC)
C(TPR.TSR) -> C(PPR.PSR)
otherwise, no change to C(PPR)

MODIFICATIONS: All except, DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TOV

Transfer On Overflow 617 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If Overflow indicator ON then
C(TPR.CA) C(PPR.IC)
C(TPR.TSR) -> C(PPR.PSR)

otherwise, no change to C(PPR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)
OVERFLOW
Set OFF

NOTES:
Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TPL
Transfer on Plus 605 (0)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
If Negative indicator OFF, then
C(TPR.CA) -> C(PPR.IC)
C(TPR.TSR) -> C(PPR.PSR)
otherwise, no change to C(PPR)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
None affected

NOTES:
Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TPNZ
Transfer on Plus and Nonzero 605 (1)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
If Negative and Zero indicators are OFF then
C(TPR.CA) -> C(PPR.IC)
C(TPR.TSR) -> C(PPR.PSR)
otherwise, no change to C(PPR)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
None affected

NOTES:
Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
TRANSFER

TRA        Transfer Unconditionally  710 (0)

FORMAT:  Basic Instruction Format (See Figure 2-1).

SUMMARY:  C(TPR.CA) --> C(PPR.IC)
          C(TPR.TSR) --> C(PPR.PSR)

MODIFICATIONS:  All except DU, DL, CI, SC, SCR

INDICATORS:  None affected

NOTES:  Attempted repetition with RPT, RPO, or RPL causes an
         Illegal Procedure Fault.

TRC        Transfer on Carry  603 (0)

FORMAT:  Basic Instruction Format (See Figure 2-1).

SUMMARY:  If Carry indicator ON then
          C(TPR.CA) C(PPR.IC)
          C(TPR.TSR) --> C(PPR.PSR)
          otherwise, no change to C(PPR)

MODIFICATIONS:  All except DU, DL, CI, SC, SCR

INDICATORS:  None affected

NOTES:  Attempted repetition with RPT, RPO, or RPL causes an
         Illegal Procedure Fault.

TRIF       Transfer on Truncation Indicator OFF  601 (1)

FORMAT:  Basic Instruction Format (See Figure 2-1).

SUMMARY:  If Truncation Indicator OFF then
          C(TPR.CA) --> C(PPR.IC)
          C(TPR.TSR) --> C(PPR.PSR)
          otherwise, no change to C(PPR)
TRANSFER

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TRTN
Transfer on Truncation Indicator ON 600 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If Truncation Indicator ON then
C(TPR.CA) -> C(PPR.IC)
C(TPR.TSR) -> C(PPR.PSR)
otherwise, no change to C(PPR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Truncation Set OFF

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TSP0  Transfer and Set PR0 270 (0)
TSP1  Transfer and Set PR1 271 (0)
TSP2  Transfer and Set PR2 272 (0)
TSP3  Transfer and Set PR3 273 (0)
TSP4  Transfer and Set PR4 670 (0)
TSP5  Transfer and Set PR5 671 (0)
TSP6  Transfer and Set PR6 672 (0)
TSP7  Transfer and Set PR7 673 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code
C(PPR.PRR) -> C(PRn.RNR)
C(PPR.PSR) -> C(PRn.SNR)
C(PPR.IC) +1 -> C(PRn.WORDNO)
00...0 -> C(PRn.BITNO)
TRANSFER

C(TPR.CA) -> C(PPR.IC)
C(TPR.TSR) -> C(PPR.PSR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TSS

Transfer and Set Slave

715 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: C(TPR.CA) -> C(PPR.IC)
C(TPR.TSR) -> C(PPR.PSR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected (except as noted below)

NOTES: If the TSS instruction is executed with the Processor not in BAR mode, the Absolute indicator is set OFF, and the Not BAR Mode indicator is set OFF to signal that subsequent addressing is to be done in the BAR Mode. The Base Address Register (BAR) is used in the address preparation of the transfer, and the BAR will be used in address preparation for all subsequent instructions until a fault or interrupt occurs.

If the TSS instruction is executed with the Not BAR Mode Indicator already OFF, it functions as a Transfer (TR/) instruction and no indicators are changed.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TSXn

Transfer and Set Index Register Xn

70n (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code

C(PPR.IC) + 1 -> C(Xn)
C(TPR.CA) -> C(PPR.IC)
MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TTF
Transfer on Tally Runout Indicator OFF 607 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If Tally Runout Indicator OFF then
C(TPR.CA) -> C(PPR.IC)
C(TPR.TSR) -> C(PPR.PSR)
otherwise, no change to C(PPR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TTN
Transfer on Tally Runout Indicator ON 606 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: If Tally Runout Indicator ON then
C(TPR.CA) -> C(PPR.IC)
C(TPR.TSR) -> C(PPR.PSR)
otherwise, no change to C(PPR)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
TRANSFER

TZE  Transfer On Zero  600 (0)

FORMAT:  Basic Instruction Format (See Figure 2-1).

SUMMARY:  If Zero indicator ON then
            C(TPR.CA) -> C(PPR.IC)
            C(TPR.ISR) -> C(PPR.PSR)
            otherwise, no change to C(PPR)

MODIFICATIONS:  All except DU, DL, CI, SC, SCR

INDICATORS:  None affected

NOTES:  Attempted repetition with RPT, RPD, or RPL causes an
        Illegal Procedure Fault.
**POINTER REGISTER INSTRUCTIONS**

**Summary:**
For \( n = 0, 1, \ldots, \) or 7 as determined by operation code

\[
C(TPR,CA) \to C(PRn.SNR)
\]

**Modifications:**
All except DU, DL, CI, SC, SCR

**Indicators:**
None affected

**Notes:**
- Attempted execution in BAR mode causes an Illegal Procedure Fault.
- Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

### Effective Address to Segment Number

<table>
<thead>
<tr>
<th>Pointer</th>
<th>Effective Address to Segment Number of PR</th>
<th>Format</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>EASP0</td>
<td>Effective Address to Segment Number of PR0</td>
<td>311 (0)</td>
<td>For ( n = 0, 1, \ldots, ) or 7 as determined by operation code ( C(TPR,CA) \to C(PRn.SNR) )</td>
</tr>
<tr>
<td>EASP1</td>
<td>Effective Address to Segment Number of PR1</td>
<td>310 (1)</td>
<td></td>
</tr>
<tr>
<td>EASP2</td>
<td>Effective Address to Segment Number of PR2</td>
<td>313 (0)</td>
<td></td>
</tr>
<tr>
<td>EASP3</td>
<td>Effective Address to Segment Number of PR3</td>
<td>312 (1)</td>
<td></td>
</tr>
<tr>
<td>EASP4</td>
<td>Effective Address to Segment Number of PR4</td>
<td>331 (0)</td>
<td></td>
</tr>
<tr>
<td>EASP5</td>
<td>Effective Address to Segment Number of PR5</td>
<td>330 (1)</td>
<td></td>
</tr>
<tr>
<td>EASP6</td>
<td>Effective Address to Segment Number of PR6</td>
<td>333 (0)</td>
<td></td>
</tr>
<tr>
<td>EASP7</td>
<td>Effective Address to Segment Number of PR7</td>
<td>332 (1)</td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**

- Attempted execution in BAR mode causes an Illegal Procedure Fault.
- Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

### Effective Address to Word/Bit Number

<table>
<thead>
<tr>
<th>Pointer</th>
<th>Effective Address to Word/Bit Number of PR</th>
<th>Format</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAWP0</td>
<td>Effective Address to Word/Bit Number of PR0</td>
<td>310 (0)</td>
<td>For ( n = 0, 1, \ldots, ) or 7 as determined by operation code ( C(TPR,CA) \to C(PRn.WORDNO) )</td>
</tr>
<tr>
<td>EAWP1</td>
<td>Effective Address to Word/Bit Number of PR1</td>
<td>311 (1)</td>
<td></td>
</tr>
<tr>
<td>EAWP2</td>
<td>Effective Address to Word/Bit Number of PR2</td>
<td>312 (0)</td>
<td></td>
</tr>
<tr>
<td>EAWP3</td>
<td>Effective Address to Word/Bit Number of PR3</td>
<td>313 (1)</td>
<td></td>
</tr>
<tr>
<td>EAWP4</td>
<td>Effective Address to Word/Bit Number of PR4</td>
<td>330 (0)</td>
<td></td>
</tr>
<tr>
<td>EAWP5</td>
<td>Effective Address to Word/Bit Number of PR5</td>
<td>331 (1)</td>
<td></td>
</tr>
<tr>
<td>EAWP6</td>
<td>Effective Address to Word/Bit Number of PR6</td>
<td>332 (0)</td>
<td></td>
</tr>
<tr>
<td>EAWP7</td>
<td>Effective Address to Word/Bit Number of PR7</td>
<td>333 (1)</td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**

- Attempted execution in BAR mode causes an Illegal Procedure Fault.
- Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
POINTER REGISTER DATA MOVEMENT LOAD

INDICATORS: None affected

NOTES:
- Attempted execution in BAR Mode causes an Illegal Procedure Fault.
- Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EPBP0  Effective Pointer at Base to PRO  350 (1)
EPBP1  Effective Pointer at Base to PRI  351 (0)
EPBP2  Effective Pointer at Base to PR2  352 (1)
EPBP3  Effective Pointer at Base to PR3  353 (1)
EPBP4  Effective Pointer at Base to PR4  370 (1)
EPBP5  Effective Pointer at Base to PR5  371 (0)
EPBP6  Effective Pointer at Base to PR6  372 (1)
EPBP7  Effective Pointer at Base to PR7  373 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code

C(TPR.TRR) -> C(PRn.RNR)
C(TPR.TSR) -> C(PRn.SNR)
00...0 -> C(PRn.WORDNO)
00 -> C(PRn.CHAR)
0000 -> C(PRn.BITNO)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES:
- Attempted execution in BAR Mode causes an Illegal Procedure Fault.
- Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

EPP0  Effective Pointer to PRO  350 (0)
EPP1  Effective Pointer to PRI  351 (1)
EPP2  Effective Pointer to PR2  352 (0)
EPP3  Effective Pointer to PR3  353 (1)
EPP4  Effective Pointer to PR4  370 (0)
EPP5  Effective Pointer to PR5  371 (1)
EPP6  Effective Pointer to PR6  372 (0)
EPP7  Effective Pointer to PR7  373 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975 2-118
POINTER REGISTER DATA MOVEMENT LOAD

SUMMARY: For \( n = 0, 1, \ldots, 7 \) as determined by operation code
\[
\begin{align*}
C(TPR,TRR) & \rightarrow C(PRn.RNR) \\
C(TPR,TSR) & \rightarrow C(PRn.SNR) \\
C(TPR,CA) & \rightarrow C(PRn.WORDNO) \\
C(TPR,TBR) / 9 & \rightarrow C(PRn.CHAR) \\
C(TPR,TBR) \text{ modulo } 9 & \rightarrow C(PRn.BITNO)
\end{align*}
\]

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted execution in BAR Mode causes an Illegal Procedure Fault.

Attemted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LPRI

Load Pointer Registers from ITS Pairs

-FORMAT- Basic Instruction Format (See Figure 2-1).

SUMMARY: For \( n = 0, 1, \ldots, 7 \)
\[
\begin{align*}
\text{Maximum of } & \quad \text{C}(Y+2n\text{-pair})18,20; \text{C}(SDW.RI); \text{C}(TPR,TRR) \rightarrow C(PRn.RNR) \\
& \quad \text{C}(Y+2n\text{-pair})3,17 \rightarrow C(PRn.SNR) \\
& \quad \text{C}(Y+2n\text{-pair})36,53 \rightarrow C(PRn.WORDNO) \\
& \quad \text{C}(Y+2n\text{-pair})57,62 / 9 \rightarrow C(PRn.CHAR) \\
& \quad \text{C}(Y+2n\text{-pair})57,62 \text{ modulo } 9 \rightarrow C(PRn.BITNO)
\end{align*}
\]

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None Affected

NOTES: Starting at location \( Y \), the contents of eight word pairs (in ITS pair format) replace the contents of Pointer Registers 0 through 7 as shown. The hardware assumes that \( Y_{14,17} = 0000 \) and addressing is incremented accordingly; no check is made.

Since \( C(TPR,TRR) \) and \( C(SDW,R1) \) are both equal to zero in Absolute mode, \( C(Y+2n\text{-pair})18,20 \) are loaded into \( PRn.RNR \) in Absolute mode.

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975

2-119

AL39
POINTER REGISTER DATA MOVEMENT LOAD

Attempted execution in BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LPRPn
Load PRn Packed

76n (0)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
For n = 0, 1, ..., or 7 as determined by operation code

\[ C(TPR,TRR) \rightarrow C(PRn,RNR) \]

If \( C(Y) \neq 11 \), then

\[ C(Y) \mod 9 \rightarrow C(PRn,BITNO); \]

otherwise, generate Command Fault

If \( C(Y) = 11....1 \), then 111 \( \rightarrow C(PRn,SNR)0,2 \)

Otherwise, 000 \( \rightarrow C(PRn,SNR)0,2 \)

\[ C(Y)18,35 \rightarrow C(PRn,WORDNO) \]

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
None affected

NOTES:
Binary "1"s in \( C(Y) \mod 9 \) correspond to an illegal BITNO, that is, a bit position beyond the extent of \( C(Y) \). Detection of these bits causes a Command Fault.

Attempted execution in BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
## Pointer Register Data Movement Store

<table>
<thead>
<tr>
<th>SPBPQ</th>
<th>Store Segment Base Pointer of PR0</th>
<th>250 (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPBP1</td>
<td>Store Segment Base Pointer of PR1</td>
<td>251 (0)</td>
</tr>
<tr>
<td>SPBP2</td>
<td>Store Segment Base Pointer of PR2</td>
<td>252 (1)</td>
</tr>
<tr>
<td>SPBP3</td>
<td>Store Segment Base Pointer of PR3</td>
<td>253 (0)</td>
</tr>
<tr>
<td>SPBP4</td>
<td>Store Segment Base Pointer of PR4</td>
<td>650 (1)</td>
</tr>
<tr>
<td>SPBP5</td>
<td>Store Segment Base Pointer of PR5</td>
<td>651 (0)</td>
</tr>
<tr>
<td>SPBP6</td>
<td>Store Segment Base Pointer of PR6</td>
<td>652 (1)</td>
</tr>
<tr>
<td>SPBP7</td>
<td>Store Segment Base Pointer of PR7</td>
<td>653 (0)</td>
</tr>
</tbody>
</table>

### Format

Basic Instruction Format (See Figure 2-1).

### Summary

For \( n = 0, 1, \ldots, 7 \) as determined by operation code:

- \( C(PRn.SNR) \rightarrow C(Y\text{-pair})3,17 \)
- \( C(PRn.RNR) \rightarrow C(Y\text{-pair})18,20 \)
- \( 000 \rightarrow C(Y\text{-pair})0,2 \)
- \( 000...0 \rightarrow C(Y\text{-pair})21,29 \)
- \( 43 \text{ (octal)} \rightarrow C(Y\text{-pair})30,35 \)
- \( 000...0 \rightarrow C(Y\text{-pair})36,71 \)

### Modifications

All except DU, DL, CI, SC, SCR

### Indicators

None affected

### Notes

The hardware assumes \( Y \) bit 17 = 0; no check is made.

Attempted execution in BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

### SPRI

Store Pointer Registers as ITS Pairs

Basic Instruction Format (See Figure 2-1).

### Summary

For \( n = 0, 1, \ldots, 7 \):

- \( 000 \rightarrow C(Y+2n\text{-pair})0,2 \)
- \( C(PRn.SNR) \rightarrow C(Y+2n\text{-pair})3,17 \)
- \( C(PRn.RNR) \rightarrow C(Y+2n\text{-pair})18,20 \)
- \( 000...0 \rightarrow C(Y+2n\text{-pair})21,29 \)
POINTER REGISTER DATA MOVEMENT STORE

43 (octal) \(\rightarrow\) C(Y+2n-pair)30,35
C(PRn.WORDNO) \(\rightarrow\) C(Y+2n-pair)36,53
000 \(\rightarrow\) C(Y+2n-pair)54,56
9 * C(PRn.CHAR) + C(PRn.BITNO) \(\rightarrow\) C(Y+2n-pair)57,62
00...0 \(\rightarrow\) C(Y+2n-pair)63,71

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES:
Starting at location Y, the contents of Pointer Registers 0 through 7 replace the contents of eight word pairs (in ITS pair format). The hardware assumes Y bits 14 to 17 = 0000 and addressing is incremented accordingly; no check is made.

Attempted execution in BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SPRI0
SPRI1
SPRI2
SPRI3
SPRI4
SPRI5
SPRI6
SPRI7

Store PRO as ITS Pair 250 (0)
Store PR1 as ITS Pair 251 (1)
Store PR2 as ITS Pair 252 (0)
Store PR3 as ITS Pair 253 (1)
Store PR4 as ITS Pair 650 (0)
Store PR5 as ITS Pair 651 (1)
Store PR6 as ITS Pair 652 (0)
Store PR7 as ITS Pair 653 (1)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY:
For \(n = 0, 1, \ldots, \) or 7 as determined by operation code
000 \(\rightarrow\) C(Y-pair)0,2
C(PRn.SNR) \(\rightarrow\) C(Y-pair)3,17
C(PRn.RNR) \(\rightarrow\) C(Y-pair)18,20
00...0 \(\rightarrow\) C(Y-pair)21,29

43 (octal) \(\rightarrow\) C(Y-pair)30,35
C(PRn.WORDNO) \(\rightarrow\) C(Y-pair)36,53
000 \(\rightarrow\) C(Y-pair)54,56
9 * C(PRn.CHAR) + C(PRn.BITNO) \(\rightarrow\) C(Y-pair)57,62
MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: The hardware assumes Y bit 17 = 0; no check is made.

- Attempted execution in BAR Mode causes an Illegal Procedure Fault.
- Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SPRn

- Store PRn Packed

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code

\[
9 \cdot C(PRn.CHAR) + C(PRn.BITN) \rightarrow C(Y)8,5
\]

\[
C(PRn.SNR)3,14 \rightarrow C(Y)6,17
\]

\[
C(PRn.WORDN) \rightarrow C(Y)18,35
\]

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES:

- If C(PRn.SNR)0,2 are nonzero, and C(PRn.SNR) \neq 11...1, then a Store Fault, Illegal Pointer, will occur and C(Y) will not be changed.

- Attempted execution in BAR Mode causes an Illegal Procedure Fault.

- Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
### Pointer Register Address Arithmetic

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADWP0</td>
<td>Add to Word Register of PR0</td>
<td>050 (0)</td>
</tr>
<tr>
<td>ADWP1</td>
<td>Add to Word Register of PR1</td>
<td>051 (0)</td>
</tr>
<tr>
<td>ADWP2</td>
<td>Add to Word Register of PR2</td>
<td>052 (0)</td>
</tr>
<tr>
<td>ADWP3</td>
<td>Add to Word Register of PR3</td>
<td>053 (0)</td>
</tr>
<tr>
<td>ADWP4</td>
<td>Add to Word Register of PR4</td>
<td>150 (0)</td>
</tr>
<tr>
<td>ADWP5</td>
<td>Add to Word Register of PR5</td>
<td>151 (0)</td>
</tr>
<tr>
<td>ADWP6</td>
<td>Add to Word Register of PR6</td>
<td>152 (0)</td>
</tr>
<tr>
<td>ADWP7</td>
<td>Add to Word Register of PR7</td>
<td>153 (0)</td>
</tr>
</tbody>
</table>

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** For \( n = 0, 1, \ldots, 7 \) as determined by operation code:

\[
\begin{align*}
C(Y)0,17 + C(PRn.WORDNO) &\rightarrow C(PRn.WORDNO) \\
00 &\rightarrow C(PRn.CHAR) \\
0000 &\rightarrow C(PRn.BITNO)
\end{align*}
\]

**MODIFICATIONS:** All except DL, CI, SC, SCR

**INDICATORS:** None affected

**NOTES:** Attempted execution in BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
**Effective Pointer to AQ Register**

**EPAQ**

**Format:**
Basic Instruction Format (See Figure 2-1).

**Summary:**

<table>
<thead>
<tr>
<th>Expression</th>
<th>AQ Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>00...0</td>
<td>C(AQ)0,2</td>
</tr>
<tr>
<td>C(TPR.TSR)</td>
<td>C(AQ)3,17</td>
</tr>
<tr>
<td>00...0</td>
<td>C(AQ)18,32</td>
</tr>
<tr>
<td>C(TPR.TRR)</td>
<td>C(AQ)33,35</td>
</tr>
<tr>
<td>C(TPR.CA)</td>
<td>C(AQ)36,53</td>
</tr>
<tr>
<td>00...0</td>
<td>C(AQ)54,65</td>
</tr>
<tr>
<td>C(TPR.TBR)</td>
<td>C(AQ)66,71</td>
</tr>
</tbody>
</table>

**Modifications:**
All except DU, DL, CI, SC, SCR

**Indicators:**
(Indicators not listed are not affected)

- **Zero**
  - If C(AQ) = 0, then ON; otherwise OFF

**Notes:**

- Attempted execution in BAR Mode causes an Illegal Procedure Fault.
- Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

---

**Review Draft**
Subject to Change
October, 1975

---

2-125 AL39
CALENDAR CLOCK

"MISCELLANEOUS INSTRUCTIONS"

"Calendar Clock"

<table>
<thead>
<tr>
<th>RCCL</th>
<th>Read Calendar Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>633 (0)</td>
</tr>
</tbody>
</table>

**FORMAT**: Basic Instruction Format (See Figure 2-1).

**SUMMARY**:
- `00...0 -> C(AQ)0,19`
- `C(Calendar Clock) -> C(AQ)20,71`

**MODIFICATIONS**: All except DU, DL, CI, SC, SCR

**INDICATORS**: None affected

**NOTES**: C(TPR,CA)0,2 specify which Processor port (i.e., which System Controller) is to be used. The contents of the clock in the designated System Controller replace the contents of the AQ-register as shown.

- Attempted execution in BAR Mode causes an Illegal Procedure Fault.
- Attempted repetition with PRT, RPD, or RPL causes an Illegal Procedure Fault.
<table>
<thead>
<tr>
<th>DRL</th>
<th>Derail</th>
<th>002 (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMAT:</td>
<td>Basic Instruction Format (See Figure 2-1).</td>
<td></td>
</tr>
<tr>
<td>SUMMARY:</td>
<td>Causes a fault which fetches and executes, in Absolute Mode, the instruction pair at main store location C+14(octal). The value of C is obtained from the FAULT VECTOR switches on the Processor Configuration Panel.</td>
<td></td>
</tr>
<tr>
<td>MODIFICATIONS:</td>
<td>All, but none affect instruction execution</td>
<td></td>
</tr>
<tr>
<td>INDICATORS:</td>
<td>None affected</td>
<td></td>
</tr>
<tr>
<td>NOTES:</td>
<td>Except for the different constant used for fetching the instruction pair from main store, the DRL Instruction is identical to the Master Mode Entry (MME) instruction. Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.</td>
<td></td>
</tr>
</tbody>
</table>
**EXECUTE**

**EXECUTE**

**XEC**

**Execute**

716 (0)

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** Fetch and execute the instruction in C(Y).

**MODIFICATIONS:** All except DU, DL, CI, SC, SCR

**INDICATORS:** None affected

**NOTES:**

The XEC instruction itself does not affect any indicator. However, the execution of the instruction from C(Y) may affect indicators.

If the execution of the instruction from C(Y) modifies C(PPR.IC), then a transfer of control occurs; otherwise, the next instruction to be executed is fetched from C(PPR.IC)+1.

To execute a Repeat Double (RPD) instruction, the XEC instruction must be in an odd location. The instruction pair repeated is that instruction pair at C(PPR.IC)+1, that is, the instruction pair immediately following the XEC instruction. C(PPR.IC) is adjusted during the execution of the repeated instruction pair so that the next instruction fetched for execution is from the first word following the repeated instruction pair.

EIS Multiword instructions may be executed but the required Data Descriptors must be located immediately after the XEC instruction, that is, starting at C(PPR.IC)+1. C(PPR.IC) is adjusted during execution of the EIS Multiword instruction so that the next instruction fetched for execution is from the first word following the EIS Data Descriptors.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
EXECUTE

XED

Execute Double

717 (0)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
Fetch and execute the instruction pair at CY-pair.

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
None affected

NOTES:
The XED instruction itself does not affect any indicator. However, the execution of the instruction pair from CY-pair may affect indicators.

The even instruction from CY-pair must not alter CY-pair)36,71, and must not be another XED instruction.

If the execution of the instruction pair from CY-pair alters C(PPR.IC), then a transfer of control occurs; otherwise, the next instruction to be executed is fetched from C(PPR.IC)+1. If the even instruction from CY-pair alters C(PPR.IC), then the transfer of control is effective immediately and the odd instruction is not executed.

To execute an instruction pair having a Repeat Double (RPD) instruction as the odd instruction, the XED must be located at an odd address. The instruction pair repeated is that instruction pair at C(PPR.IC) + 1, that is, the instruction pair immediately following the XED instruction. C(PPR.IC) is adjusted during the execution of the repeated instruction pair so the the next instruction fetched for execution is from the first word following the repeated instruction pair.

An attempt to execute an EIS Multiword instruction will cause an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an illegal Procedure Fault.
MASTER MODE ENTRY

"Master" Mode Entry

MME Master Mode Entry 001 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: Causes a fault that fetches and executes, in Absolute Mode, the instruction pair at main store location C+4 (octal). The value of C is obtained from the FAULT VECTOR switches on the Processor Configuration Panel.

MODIFICATIONS: All, but none affect instruction execution

INDICATORS: None affected

NOTES: Execution of the MME instruction implies the following conditions:

During the execution of the MME instruction and the two instructions fetched, the Processor is temporarily in Absolute Mode independent of the value of the Absolute Mode indicator. The Processor stays in Absolute Mode if the Absolute Mode indicator is ON after the execution of the instructions.

The instruction at C+4 must not alter the contents of main store location C+5, and must not be an XED instruction.

If the contents of the instruction counter (PPR.IC) are changed during execution of the instruction pair at C+4, the next instruction is fetched from the modified C(PPR.IC); otherwise, the next instruction is fetched from C(PPR.IC)+1.

If the instruction at C+4 alters C(PPR.IC), then this transfer of control is effective immediately, and the instruction at C+5 is not executed.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
HME2

Master Mode Entry 2

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
Causes a fault that fetches and executes, in Absolute Mode, the instruction pair at main store location C+52(octal). The value of C is obtained from the FAULT VECTOR switches on the Processor Configuration Panel.

MODIFICATIONS:
All, but none affect instruction execution.

INDICATORS:
None affected.

NOTES:
Attempted execution in BAR mode causes an Illegal Procedure, Illegal Opcode Fault.

Except for the different constant used for fetching the instruction pair from main store, the HME2 instruction is identical to the Master Mode Entry (MME) instruction.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

MME3

Master Mode Entry 3

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
Causes a fault that fetches and executes, in Absolute Mode, the instruction pair at main store location C+54(octal). The value of C is obtained from the FAULT VECTOR switches on the Processor Configuration Panel.

MODIFICATIONS:
All, but none affect instruction execution.

INDICATORS:
None affected.

NOTES:
Attempted execution in BAR mode causes an Illegal Procedure, Illegal Opcode Fault.

Except for the different constant used for fetching the instruction pair from main store, the MME3 instruction is identical to the Master Mode Entry (MME) instruction.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
Master Mode Entry 4

Format:
Basic Instruction Format (See Figure 2-1).

Summary:
Causes a fault that fetches and executes, in Absolute Mode, the instruction pair at main store location C+56 (octal). The value of C is obtained from the FAULT VECTOR switches on the Processor Configuration Panel.

Modifications:
All, but none affect instruction execution.

Indicators:
None affected.

Notes:
Attempted execution in BAR mode causes an Illegal Procedure, Illegal Opcode Fault.

Except for the different constant used for fetching the instruction pair from main store, the MME4 instruction is identical to the Master Mode Entry (MME) instruction.

Attempted repetition with RPT, RPO, or RPL causes an Illegal Procedure Fault.
**No Operation**

<table>
<thead>
<tr>
<th>NOP</th>
<th>No Operation</th>
<th>011 (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMAT</td>
<td>Basic Instruction Format (See Figure 2-1).</td>
<td></td>
</tr>
<tr>
<td>SUMMARY</td>
<td>No operation takes place</td>
<td></td>
</tr>
<tr>
<td>MODIFICATIONS</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>INDICATORS</td>
<td>None affected (except as noted below)</td>
<td></td>
</tr>
<tr>
<td>NOTES</td>
<td>No operation takes place but address preparation is performed according to the specified modifier, if any. If modification other than DU or DL is used, the effective addresses generated may cause Store Fails. The use of Indirect and Tally modifiers causes changes in the address and tally fields of the referenced Indirect Words and the Tally Runout indicator may be set ON as a result. Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.</td>
<td></td>
</tr>
</tbody>
</table>

**Pulse One**

<table>
<thead>
<tr>
<th>PULS1</th>
<th>Pulse One</th>
<th>012 (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMAT</td>
<td>Basic Instruction Format (See Figure 2-1).</td>
<td></td>
</tr>
<tr>
<td>SUMMARY</td>
<td>No operation takes place</td>
<td></td>
</tr>
<tr>
<td>MODIFICATIONS</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>INDICATORS</td>
<td>None affected (except as noted below)</td>
<td></td>
</tr>
<tr>
<td>NOTES</td>
<td>The PULS1 instruction is identical to the No Operation (NOP) instruction except that it causes certain unique synchronizing signals to appear in the Processor logic circuitry. Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.</td>
<td></td>
</tr>
</tbody>
</table>
PULS2 Pulse Two

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: No operation takes place

MODIFICATIONS: All

INDICATORS: None affected (except as noted below)

NOTES: The PULS2 instruction is identical to the No Operation (NOP) instruction except that it causes certain unique synchronizing signals to appear in the Processor logic circuitry.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
**Repeat Double (RPD)**

**Format:**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<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>TALLY</td>
<td>1A13IC</td>
<td>Term. Cond.</td>
<td>1</td>
<td>(560)B</td>
<td>101101</td>
<td>DELTA</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<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>

|   | 0 | 1 | 1 | 1 | 1 | 7 | 9 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Figure 2-9 Repeat Double (RPD) Instruction Word Format

**Summary:**

Execute the pair of instructions at C(PPR.IC)+1 either a specified number of times or until a specified termination condition is met.

**Modifications:**

None

**Indicators:**

(Indicators not listed are not affected)

- **Tally Runout**
  - If C(XO)0,7 = 0 at termination, then ON; otherwise, OFF
  - All other indicators None affected. However, the execution of the repeated instructions may affect indicators.

**Notes:**

The RPD instruction must be stored in an odd main store location except when accessed via the XEC or XED instructions, in which case the XEC or XED instruction must itself be in an odd main store location.

Both repeated instructions must use R or RI modifiers and only X1, X2, ..., X7 are permitted. For the purposes of this description, the even repeated instruction shall use X-even and the odd repeated instruction shall use X-odd. X-even and X-odd may be the same register.

If C = 1, then C(RPD instruction word)0,17 -> C(XO); otherwise, C(XO) unchanged prior to execution.

The termination condition and tally fields of C(XO) control the repetition of the instruction pair. An initial tally of zero is interpreted as 256.

The repetition cycle consists of the following steps:

a. Execute the pair of repeated instructions

b. C(XO)0,7 - 1 -> C(XO)0,7
   - Modify C(X-even) and C(X-odd) as described below.

C. If C(XO)0,7 = 0, then set Tally Runout indicator ON and terminate.
d. If a terminate condition has been met, then set Tally Runout indicator OFF and terminate.

e. Go to step a.

If a Processor Fault occurs during the execution of the instruction pair, the repetition loop is terminated and control passes to the Fault Trap according to the conditions for the Processor Fault. C(X0), C(X-even), and C(X-odd) are not updated for the repetition cycle in which the fault occurs. Note: in particular that certain Processor Faults occurring during execution of the even instruction preclude the execution of the odd instruction for the faulting repetition cycle.

EIS Multiword instructions cannot be repeated. All other instructions may be repeated except as noted for individual instructions or those that...

Explicitly alter C(X0)

The effective address, Y, of the operand (in the case of R modification) or indirect word (in the case of RI modification) is determined as follows:

For the first execution of the repeated instruction pair...

\[ C(C(PPR,IC)+1)0,17 + C(X-even) \to Y; Y-even \to C(X-even) \]
\[ C(C(PPR,IC)+2)0,17 + C(X-odd) \to Y-odd; Y-odd \to C(X-odd) \]

For all successive executions of the repeated instruction pair...

if C(X0)8 = 1, then C(X-even) + Delta \to Y-even, Y-even \to C(X-even); otherwise, C(X-even) \to Y-even

if C(X0)9 = 1, then C(X-odd) + Delta \to Y-odd, Y-odd \to C(X-odd); otherwise, C(X-odd) \to Y-odd

C(X0)8,9 correspond to Control Bits A and B, respectively, of the RPD instruction.

In the case of RI modification, only one indirect reference is made per repeated execution. The tag field of the indirect word is not interpreted. The indirect word is treated as though it had R modification with R = N.

The bit configuration in C(X0)11,17 defines the conditions for which the repetition loop is terminated. The terminate conditions are examined at the completion of execution of the odd instruction. If more than one condition is specified, the repeat terminates if any of the specified conditions are met.

Bit 17 = 0 Ignore all overflows. Do not set Overflow Indicator; inhibit Overflow Fault.
Bit 17 = 1  If Overflow Mask indicator is ON, then set Overflow indicator and terminate; otherwise, cause an Overflow Fault.

Bit 16 = 1  Terminate if Carry indicator OFF.

Bit 15 = 1  Terminate if Carry indicator ON.

Bit 14 = 1  Terminate if Negative indicator OFF.

Bit 13 = 1  Terminate if Negative indicator ON.

Bit 12 = 1  Terminate if Zero indicator OFF.

Bit 11 = 1  Terminate if Zero indicator ON.

At the time of termination:

C(X)0,7 contain the Tally Residue; that is, the number of repeats remaining until a Tally Runout would have occurred.

If the RPO instruction is interrupted (by any fault) before termination, the Tally Runout indicator is OFF.

C(X-even) and C(X-odd) contain the effective addresses of the next operands or indirect words that would have been used had the repetition loop not terminated.

Attempted repetition with RPI, RPD, or RPL causes an Illegal Procedure Fault.
Executive the instruction at \( C(P_P + 1) \) either a specified number of times or until a specified termination condition is met.

**MODIFICATIONS:** None

**INDICATORS:**
- **Tally:** If \( C(X_0) = 0 \) or link address \( C(Y) = 0 \) at termination, then ON; otherwise OFF
- **Runout:** None affected. However, the execution of the repeated instruction may affect indicators.

**NOTES:**
- The repeated instruction must use an \( R \) modifier and only \( X_1, X_2, \ldots, X_7 \) are permitted. For the purposes of this description, the repeated instruction shall use \( X_n \).
- If \( C = 1 \), then \( C(RPL \text{ instruction word}) = C(X_0) \); otherwise, \( C(X_0) \) unchanged prior to execution.
- The termination condition and tally fields of \( C(X_0) \) control the repetition of the instruction. An initial tally of zero is interpreted as 256.
- The repetition cycle consists of the following steps:
  a. Execute the repeated instruction
  b. \( C(X_0) = 1 \rightarrow C(X_0) = C(X_0) \)
  c. Modify \( C(X_n) \) as described below.
  d. If \( C(X_0) = 0 \) or \( C(Y) = 0 \), then set Tally Runout indicator ON and terminate.
  e. If a terminate condition has been met, then set Tally Runout indicator OFF and terminate.
  f. Go to step a.
If a Processor Fault occurs during the execution of the instruction, the repetition loop is terminated and control passes to the Fault Trap according to the conditions for the Processor Fault. C(X0) and C(Xn) are not updated for the repetition cycle in which the fault occurs.

EIS Multword instructions cannot be repeated. All other instructions may be repeated except as noted for individual instructions or those that...

Explicitly alter C(X0)

Explicitly alter the link address, C(Y)0,17

The effective address, Y, of the operand is determined as follows:

For the first execution of the repeated instruction ...

C(C(PPRCIC)+1)0,17 + C(Xn) -> Y; Y -> C(Xn)

For all successive executions of the repeated instruction ...

C(Xn) -> Y
if C(Y)0,17 ≠ 0, then C(Y)0,17 -> C(Xn); otherwise, no change to C(Xn)

C(Y)0,17 is known as the link address and is the effective address of the next entry in a threaded list of operands to be referenced by the repeated instruction.

The operand data is formed as

\[ \{00...0\}0,17 \| \{1\} \{C(Y)18\}p \]

where p is 35 for single precision operands and 71 for double precision operands.

The bit configuration in C(X0)11,17 and the link address, C(Y)0,17, define the conditions for which the repetition loop is terminated. The terminate conditions are examined at the completion of execution of the instruction. If more than one condition is specified, the repeat terminates if any of the specified conditions are met.

C(Y)0,17 = 0 Set Tally Runout indicator ON and terminate.

Bit 17 = 0 Ignore all overflows. Do not set Overflow Indicator; inhibit Overflow Fault.

Bit 17 = 1 If Overflow Mask indicator is ON, then set

\[ \text{Overflow indicator and terminate;} \]
\[ \text{otherwise, cause an Overflow Fault.} \]

Bit 16 = 1 Terminate if Carry indicator OFF.

Bit 15 = 1 Terminate if Carry indicator ON.

Bit 14 = 1 Terminate if Negative indicator OFF.
Bit 13 = 1  Terminate if Negative indicator ON.
Bit 12 = 1  Terminate if Zero indicator OFF.
Bit 11 = 1  Terminate if Zero indicator ON.

At the time of termination:

C(XO)0,7 contain the Tally Residue; that is, the number of repeats remaining until a Tally Runout would have occurred.

If the RPL instruction is interrupted (by any fault) before termination, the Tally Runout indicator is OFF.

C(Xn) contain the last link address, that is, the effective address of the list word containing the last operand data and the NEXT link address.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
REPEAT

RPT Repeat

520 (0)

FORMAT:

<table>
<thead>
<tr>
<th></th>
<th>0 0 0 1 1</th>
<th>1 1</th>
<th>2 2 2 3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7 8 9 0 1</td>
<td>1 1</td>
<td>1 1 1 1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>TALLY</td>
<td>1 0</td>
<td>IC1 Term. Cond.</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1 1</td>
<td>1 1 1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 2-11 Repeat (RPT) Instruction Word Format

SUMMARY:
Execute the instruction at C(P)RR,IC)+1 either a specified number of times or until a specified termination condition is met.

MODIFICATIONS:
None

INDICATORS:
(Indicators not listed are not affected)
- Tally
- Runout

If C(X)(0, 7 = 0 at termination, then ON; otherwise, OFF

All other Indicators None affected. However, the execution of the repeated instruction may affect indicators.

NOTES:
The repeated instruction must use an R or RI modifier and only X1, X2, ..., X7 are permitted. For the purposes of this description, the repeated instruction shall use Xn.

If C = 1, then C(RPT instruction word)0,17 -> C(X)(0); otherwise, C(X) unchanged prior to execution.

The termination condition and tally fields of C(X)(0) control the repetition of the instruction pair. An initial tally of zero is interpreted as 256.

The repetition cycle consists of the following steps:

a. Execute the repeated instruction
b. C(X)(0,7 - 1 -> C(X)0,7
   Modify C(Xn) as described below
   c. If C(X)(0,7 = 0, then set Tally Runout indicator ON and terminate
   d. If a terminate condition has been met, then set Tally Runout indicator OFF and terminate
   e. Go to step a
If a Processor Fault occurs during the execution of the instruction, the repetition loop is terminated and control passes to the Fault Trap according to the conditions for the Processor Fault. C(X0) and C(Xn) are not updated for the repetition cycle in which the fault occurs.

EIS Multiword instructions cannot be repeated. All other instructions may be repeated except as noted for individual instructions or those that...

Explicitly alter C(X0)

Explicitly alter C(PPR.IC)+2

The effective address, Y, of the operand (in the case of R modification) or indirect word (in the case of RI modification) is determined as follows:

For the first execution of the repeated instruction...

\[ \text{C}(\text{C}(\text{PPR.IC})+1)_{0,17} + \text{C}(\text{Xn}) \rightarrow Y; \ Y \rightarrow \text{C}(\text{Xn}) \]

For all successive executions of the repeated instruction...

\[ \text{if } \text{C}(\text{XO})_{18} = 1, \text{ then } \text{C}(\text{Xn}) + \text{Delta} \rightarrow Y, \ Y \rightarrow \text{C}(\text{Xn}); \text{ otherwise, } \text{C}(\text{Xn}) \rightarrow Y \]

C(X0)18 corresponds to Control Bit A of the RPD instruction.

In the case of RI modification, only one indirect reference is made per repeated execution. The tag field of the indirect word is not interpreted. The indirect word is treated as though it had R modification with R = N.

The bit configuration in C(X0)11,17 defines the conditions for which the repetition loop is terminated. The terminate conditions are examined at the completion of execution of the instruction. If more than one condition is specified, the repeat terminates if any of the specified conditions are met.

Bit 17 = 0 Ignore all overflows. Do not set Overflow Indicator; inhibit Overflow Fault.

Bit 17 = 1 If Overflow Mask Indicator is ON, then set Overflow indicator and terminate; otherwise, cause an Overflow Fault.

Bit 16 = 1 Terminate if Carry indicator OFF.

Bit 15 = 1 Terminate if Carry indicator ON.

Bit 14 = 1 Terminate if Negative indicator OFF.

Bit 13 = 1 Terminate if Negative indicator ON.

Bit 12 = 1 Terminate if Zero indicator OFF.

Bit 11 = 1 Terminate if Zero indicator ON.
At the time of termination:

\[ C(X0) \] contains the Tally Residue; that is, the number of repeats remaining until a Tally Runout would have occurred.

If the RPT instruction is interrupted (by any fault) before termination, the Tally Runout indicator is OFF.

\[ C(Xn) \] contains the effective address of the next operand or indirect word that would have been used had the repetition loop not terminated.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
RING ALARM REGISTER

- Ring Alarm Register -

SRA: Store Ring Alarm

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: 00...0 -> C(Y)0,32

C(RALR) -> C(Y)33,35

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: Attempted execution in BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
### Store Base Address Register

<table>
<thead>
<tr>
<th>SBAR</th>
<th>Store Base Address Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMAT:</td>
<td>Basic Instruction Format (See Figure 2-1).</td>
</tr>
<tr>
<td>SUMMARY:</td>
<td>C(BAR) -&gt; C(Y)0,17</td>
</tr>
<tr>
<td>MODIFICATIONS:</td>
<td>All except DU, DL, CI, SC, and SCR</td>
</tr>
<tr>
<td>INDICATORS:</td>
<td>None affected</td>
</tr>
</tbody>
</table>
**BCD**

**FORMAT:**

Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

Shift CA left six positions

\[ \text{CA} / \text{CY} \rightarrow 4\text{-bit quotient plus remainder} \]

Shift CO left six positions

4-bit quotient \( \rightarrow \text{CQ} \) 32, 35

remainder \( \rightarrow \text{CA} \)

**MODIFICATIONS:**

All except CI, SC, SCR

**INDICATORS:**

(Indicators not listed are not affected)

Zero

If CA = 0, then ON

Negative

If CA 0 = 1 before execution, then ON; otherwise OFF

**NOTES:**

The BCD instruction carries out one step in an algorithm for the conversion of a binary number to a string of Binary-Coded-Decimal (BCD) digits. The algorithm requires the repeated short division of the binary number or last remainder by a set of constants \( C(i) = 8^{*}i \times 10^{*(n-1)} \) for \( i = 1, 2, \text{...}, n \) with \( n \) being defined by:

\[ 10^{*(n-1)} \leq 1<\text{binary number}>1 \leq 10^{*}n - 1. \]

The values in the table that follows are the conversion constants to be used with the BCD instruction. Each vertical column represents the set of constants to be used depending on the initial value of the binary number to be converted. The instruction is executed once per digit while traversing the appropriate column from top to bottom.

An alternate use of the table for conversion involves the use of the constants in the row corresponding to conversion step 1. If, after each execution, the contents of the accumulator are shifted right 3 positions, the constants in the first row, starting at the appropriate column, may be used while traversing the row from left to right.

Because there is a limit on range, a full 36-bit word cannot be converted. The largest binary number that may be converted correctly is \( 2^{*}33 -1 \) yielding ten decimal digits.

Attempted repetition with RPL causes an Illegal Procedure Fault.
For $10^n(n-1) <= IC(AR) <= 10^{n-1}$ and $n = \ldots$

<table>
<thead>
<tr>
<th>Digit</th>
<th>10</th>
<th>2</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000000000</td>
<td>800000000</td>
<td>80000000</td>
<td>80000000</td>
<td>80000000</td>
<td>80000000</td>
</tr>
<tr>
<td>2</td>
<td>6400000000</td>
<td>6400000000</td>
<td>6400000000</td>
<td>6400000000</td>
<td>6400000000</td>
<td>6400000000</td>
</tr>
<tr>
<td>3</td>
<td>5120000000</td>
<td>5120000000</td>
<td>5120000000</td>
<td>5120000000</td>
<td>5120000000</td>
<td>5120000000</td>
</tr>
<tr>
<td>4</td>
<td>4096000000</td>
<td>4096000000</td>
<td>4096000000</td>
<td>4096000000</td>
<td>4096000000</td>
<td>4096000000</td>
</tr>
<tr>
<td>5</td>
<td>3276800000</td>
<td>3276800000</td>
<td>3276800000</td>
<td>3276800000</td>
<td>3276800000</td>
<td>3276800000</td>
</tr>
<tr>
<td>6</td>
<td>2621440000</td>
<td>2621440000</td>
<td>2621440000</td>
<td>2621440000</td>
<td>2621440000</td>
<td>2621440000</td>
</tr>
<tr>
<td>7</td>
<td>2097152000</td>
<td>2097152000</td>
<td>2097152000</td>
<td>2097152000</td>
<td>2097152000</td>
<td>2097152000</td>
</tr>
<tr>
<td>8</td>
<td>1677721600</td>
<td>1677721600</td>
<td>1677721600</td>
<td>1677721600</td>
<td>1677721600</td>
<td>1677721600</td>
</tr>
<tr>
<td>9</td>
<td>1342177280</td>
<td>1342177280</td>
<td>1342177280</td>
<td>1342177280</td>
<td>1342177280</td>
<td>1342177280</td>
</tr>
<tr>
<td>10</td>
<td>1073741824</td>
<td>1073741824</td>
<td>1073741824</td>
<td>1073741824</td>
<td>1073741824</td>
<td>1073741824</td>
</tr>
</tbody>
</table>

GTB

Gray to Binary

FORMAT:

Basic Instruction Format (See Figure 2-1).

SUMMARY:

C(A) converted from Gray Code to a 36 bit binary number

MODIFICATIONS:

None

INDICATORS:

(Indicators not listed are not affected)

Zero

If C(A) = 0, then ON; otherwise OFF

Negative

If C(A) = 1, then ON; otherwise OFF

NOTES:

This conversion is defined by the following algorithm:

\[
C(A)_0 \rightarrow C(A)_0
\]

\[
C(A)_{(i)} \oplus C(A)_{(i-1)} \rightarrow C(A)_{(i)} \quad \text{for} \quad i = 1, 2, \ldots, 35
\]

Attempted repetition with RPL causes an Illegal Procedure Fault.
PRIVILEGED - REGISTER LOAD

PRIVILEGED INSTRUCTIONS

-Privileged- "Register"Load-

**LBAR**

Load Base Address Register 230 (0)

**FORMAT:**

Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

C(Y)0,17 -> C(BAR)

**MODIFICATIONS:**

All except CI, SC, SCR

**INDICATORS:**

None affected

**NOTES:**

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

Attempted execution in BAR Mode causes a Illegal Procedure Fault.

**LCPR**

Load Central Processor Register 674 (0)

**FORMAT:**

Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

Load selected register as noted

**MODIFICATIONS:**

None. The instruction TAG field is used for register selection as follows.

<table>
<thead>
<tr>
<th>C(TAG)</th>
<th>Data and Register(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>C(Y) -&gt; C(Cache Mode Register)0,35</td>
</tr>
<tr>
<td>04</td>
<td>C(Y) -&gt; C(Mode Register)0,35</td>
</tr>
<tr>
<td>03</td>
<td>00...0 -&gt; C(CU, OU, DU, and APU History Register)0,71</td>
</tr>
<tr>
<td>07</td>
<td>11...1 -&gt; C(CU, OU, DU, and APU History Register)0,71</td>
</tr>
</tbody>
</table>

**INDICATORS:**

None affected

**NOTES:**

See Section IV, Program Accessible Registers, for descriptions and use of the various registers.

For TAG values 03 and 07, the History Register loaded is selected by the current value of a Cyclic Counter for each unit. All four Cyclic Counters are advanced by one count.
for each execution of the instruction.

Use of TAG values other than those defined above causes an Illegal Procedure Fault.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LDBR

Load Descriptor Segment Base Register

232 (0)

FORMAT

Basic Instruction Format (See Figure 2-1).

SUMMARY

If SDWAM is enabled, then

0 -> C(SDWAM(i).FULL) for i = 0, 1, ..., 15

(i) -> C(SDWAM(i).USE) for i = 0, 1, ..., 15

If PTWAM is enabled, then

0 -> C(PTWAM(i).FULL) for i = 0, 1, ..., 15

(i) -> C(PTWAM(i).USE) for i = 0, 1, ..., 15

C(Y-pair)0,23 -> C(OSBR.ADDR)

C(Y-pair)37,50 -> C(OSBR.BOUND)

C(Y-pair)55 -> C(OSBR.U)

C(Y-pair)60,71 -> C(OSBR.STACK)

MODIFICATIONS:

All except DU, DL, CI, SC, and SCR

INDICATORS:

None affected

NOTES

The hardware assumes Y17 = 0 if no check is made.

The Associative Memories are cleared (FULL indicators reset) if they are enabled.

See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging, for description

and use, respectively, of the SDWAM, PTWAM, and DSBR.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
PRIVILEGED - REGISTER LOAD

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDT</td>
<td>Load Timer Register</td>
<td>637 (0)</td>
</tr>
<tr>
<td>FORMAT</td>
<td>Basic Instruction Format (See Figure 2-1).</td>
<td></td>
</tr>
<tr>
<td>SUMMARY</td>
<td>C(Y)0,26 -&gt; C(TRY)</td>
<td></td>
</tr>
<tr>
<td>MODIFICATIONS</td>
<td>All except CI, SC, SCR</td>
<td></td>
</tr>
<tr>
<td>INDICATORS</td>
<td>None Affected</td>
<td></td>
</tr>
<tr>
<td>NOTES</td>
<td>Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.</td>
<td></td>
</tr>
<tr>
<td>LPTP</td>
<td>Load Page Table Pointers</td>
<td>257 (1)</td>
</tr>
<tr>
<td>FORMAT</td>
<td>Basic Instruction Format (See Figure 2-1).</td>
<td></td>
</tr>
<tr>
<td>SUMMARY</td>
<td>For i = 0, 1, ..., 15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m = C(PTWAM(i).USE)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C(Y+m)0,14 -&gt; C(PTWAM(m).POINTER)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C(Y+m)15,26 -&gt; C(PTWAM(m).PAGE)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C(Y+m)27 -&gt; C(PTWAM(m).F)</td>
<td></td>
</tr>
<tr>
<td>MODIFICATIONS</td>
<td>All except DU, DL, CI, SC, SCR</td>
<td></td>
</tr>
<tr>
<td>INDICATORS</td>
<td>None Affected</td>
<td></td>
</tr>
<tr>
<td>NOTES</td>
<td>The hardware assumes Y14,17 = 00000; no check is made.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Associative Memory is ignored (forced to &quot;no match&quot;) during Address Preparation.</td>
<td></td>
</tr>
</tbody>
</table>

See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging, for description and use, respectively, of the PTWAM.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975
LPTR

Load Page Table Registers

173 (1)

FORMAT:

Basic Instruction Format (See Figure 2-1).

SUMMARY:

For \( l = 0, 1, \ldots, 15 \)

\[ m = C(PTWAM(l) \cdot USE) \]

\[ C(Y+m)0_{17} \rightarrow C(PTWAM(m) \cdot ADDR) \]

\[ C(Y+m)29 \rightarrow C(PTWAM(m) \cdot M) \]

MODIFICATIONS:

All except DU, DL, CI, SC, SCR

INDICATORS:

None affected

NOTES:

The hardware assumes \( Y14,17 = 0000 \); no check is made.

The Associative Memory is ignored (forced to "no match") during Address Preparation.

See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging, for description and use, respectively, of the PTWAM.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LRA

Load Ring Alarm Register

774 (1)

FORMAT:

Basic Instruction Format (See Figure 2-1).

SUMMARY:

\[ C(Y)33,35 \rightarrow C(RALR) \]

MODIFICATIONS:

All except DU, DL, CI, SC, SCR

INDICATORS:

None affected

NOTES:

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
PRIVILEGED - REGISTER LOAD

**LSDP**

Load Segment Descriptor Pointers

**FORMAT:**

Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

For \( i = 0, 1, \ldots, 15 \)

\[ m = C(SDWAM(i)\_USE) \]

\[ C(Y+m)0,14 \rightarrow C(SDWAM(m)\_POINTER) \]

\[ C(Y+m)17 \rightarrow C(SDWAM(m)\_P) \]

**MODIFICATIONS:**

All except DU, DL, CI, SC, SCR

**INDICATORS:**

None affected

**NOTES:**

The hardware assumes \( Y14,17 = 0000 \); no check is made.

The Associative Memory is ignored (forced to "no match") during Address Preparation.

See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging, for description and use, respectively, of the SDWAM.

Attempted execution in Normal or 3AR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
LSDR

Load Segment Descriptor Registers

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
For \( i = 0, 1, \ldots, 15 \)
\[
m = C(SDWAH(l).USE)
\]
\[
C(Y+2m)0,23 \rightarrow C(SDWAH(m).ADR)
\]
\[
C(Y+2m)24,32 \rightarrow C(SDWAH(m).R1, R2, R3)
\]
\[
C(Y+2m)37,50 \rightarrow C(SDWAH(m).BOUND)
\]
\[
C(Y+2m)52,57 \rightarrow C(SDWAH(m).R, E, W, P, U, G, C)
\]
\[
C(Y+2m)58,71 \rightarrow C(SDWAH(m).CL)
\]

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
None Affected

NOTES:
The hardware assumes \( Y_{14,17} = 0000 \); no check is made.
The Associative Memory is ignored (forced to "no-match") during Address Preparation.

See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging for description and use, respectively, of the SDWAH.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPO, or RPL causes an Illegal Procedure Fault.

RCU

Restore Control Unit

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
\( C(Y-\text{block8}) \) words 0 to 7 \( \rightarrow C(\text{Control Unit Data}) \)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
None affected
NOTES

See Section IV, Program Accessible Registers, for description and use of Control Unit Data.

The hardware assumes Y15,17 = 000 and addressing is incremented accordingly; no check is made.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
SCPR

Store Central Processor Register

452 (0)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
Store selected register as noted

MODIFICATIONS:
None. The instruction TAG field is used for register selection as follows.

<table>
<thead>
<tr>
<th>C[TAG]</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>C(APU History Register) -&gt; C(Y-pair)</td>
</tr>
<tr>
<td>01</td>
<td>C(Fault Register) -&gt; C(Y-pair)0,35</td>
</tr>
<tr>
<td></td>
<td>00...0 -&gt; C(Y-pair)36,71</td>
</tr>
<tr>
<td>06</td>
<td>C(Mode Register) -&gt; C(Y-pair)0,35</td>
</tr>
<tr>
<td></td>
<td>C(Cache Mode Register) -&gt; C(Y-pair)36,71</td>
</tr>
<tr>
<td>20</td>
<td>C(CU History Register) -&gt; C(Y-pair)</td>
</tr>
<tr>
<td>40</td>
<td>C(OU History Register) -&gt; C(Y-pair)</td>
</tr>
<tr>
<td>60</td>
<td>C(OU History Register) -&gt; C(Y-pair)</td>
</tr>
</tbody>
</table>

INDICATORS:
None affected

NOTES:
See Section IV, Program Accessible Registers, for description and use of the various registers.

For TAG values 00, 20, 40, and 60, the History Register stored is selected by the current value of a Cyclic Counter for each Unit. The individual Cyclic Counters are advanced by one count for each execution of the instruction.

The use of TAG values other than those defined above causes an Illegal Procedure Fault.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
**SCU**

**Store Control Unit**

**FORMAT:**
Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
C(Control Unit Data) -> C(Y-block8) words 0 to 7

**MODIFICATIONS:**
All except DU, DL, CI, SC, SCR

**INDICATORS:**
None affected

**NOTES:**
See Section IV, Program Accessible Registers, for description and use of Control Unit Data.

The SCU instruction safe-stores control information required to service a Processor fault. The Control Unit Data is not, in general, valid at any time except when safe-stored by the first instruction of a fault/interrupt vector.

The hardware assumes Y15,17 = 000 and addressing is incremented accordingly; no check is made.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
<table>
<thead>
<tr>
<th>SDBR</th>
<th>Store Descriptor Segment Base Register</th>
<th>154 (0)</th>
</tr>
</thead>
</table>

**FORMAT:**
Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
- $C(OSBR.\text{ADDR})$ -> $C(Y\text{-pair})0,23$
- $0000.0$ -> $C(Y\text{-pair})24,36$
- $C(OSBR.\text{BOUND})$ -> $C(Y\text{-pair})37,50$
- $0000$ -> $C(Y\text{-pair})51,54$
- $C(OSBR.u)$ -> $C(Y\text{-pair})55$
- $000$ -> $C(Y\text{-pair})56,59$
- $C(OSBR.\text{STACK})$ -> $C(Y\text{-pair})60,71$

**MODIFICATIONS:**
All except DU, DL, CI, SC, SCR

**INDICATORS:**
None affected

**NOTES:**
The hardware assumes $Y_{17} = 0$; no check is made.
$C(OSBR)$ is unchanged.

See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging for description and use, respectively, of the OBR.

Attempted execution in Normal or B&R Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
PRIVILEGED - REGISTER STORE

SPTP

Store Page Table Pointers

557 (1)

FORMAT:

Basic Instruction Format (See Figure 2-1).

SUMMARY:

For \( i = 0, 1, ..., 15 \)

\[
\begin{align*}
C(PTWAM(i)\cdot POINTER) & \rightarrow C(Y+i)0,14 \\
C(PTWAM(i)\cdot PAGE) & \rightarrow C(Y+i)15,26 \\
C(PTWAM(i)\cdot F) & \rightarrow C(Y+i)27 \\
0000 & \rightarrow C(Y+i)28,31 \\
C(PTWAM(i)\cdot USE) & \rightarrow C(Y+i)32,35
\end{align*}
\]

MODIFICATIONS:

All except DU, DL, CI, SC, SCR

INDICATORS:

None affected

NOTES:

The hardware assumes that \( Y_{14,17} = 0000 \), and addressing is incremented accordingly; no check is made.

The contents of PTWAM(m) remain unchanged.

The Associative Memory is ignored (forced to a "no match") during Address Preparation.

See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging for description and use, respectively, of the PTWAM.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPO, or RPL causes an Illegal Procedure Fault.
### SPTR

**Store Page Table Registers**

<table>
<thead>
<tr>
<th>FORMAT</th>
<th>Basic Instruction Format (See Figure 2-1).</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUMMARY</td>
<td>For $i = 0, 1, \ldots, 15$</td>
</tr>
<tr>
<td></td>
<td>$C(PTWAM(i).ADDR) \rightarrow C(Y+i)0,17$</td>
</tr>
<tr>
<td></td>
<td>$00\cdots0 \rightarrow C(Y+i)18,28$</td>
</tr>
<tr>
<td></td>
<td>$C(PTWAM(i).H) \rightarrow C(Y+i)29$</td>
</tr>
<tr>
<td></td>
<td>$00\cdots0 \rightarrow C(Y+i)30,35$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MODIFICATIONS</th>
<th>All except DU, DL, CI, SC, SCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDICATORS</td>
<td>None affected</td>
</tr>
</tbody>
</table>

**NOTES**

- The hardware assumes that $Y14,17 = 0000$, and addressing will be incremented accordingly; no check is made.
- The contents of $PTWAM(m)$ are unchanged.
- The Associative Memory is ignored (forced to a "no match") during Address Preparation.
- See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging for description and use, respectively, of the PTWAM.
- Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.
- Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
Store Segment Descriptor Pointers

Basic Instruction Format (See Figure 2-1).

For \( i = 0, 1, \ldots, 15 \)

\[
\begin{align*}
C(SDHAH(i) . POINTER) & \rightarrow C(Y+i)10,14 \\
00000 & \rightarrow C(Y+i)15,26 \\
C(SDHAH(i) . F) & \rightarrow C(Y+i)27 \\
0000 & \rightarrow C(Y+i)28,31 \\
C(SDHAH(i) . USE) & \rightarrow C(Y+i)32,35
\end{align*}
\]

All except DU, DL, CI, SC, SCR

None affected

The hardware assumes \( Y14,17 = 0000 \), and addressing is incremented accordingly; no check is made.

The contents of SDHAM(i) are unchanged.

The Associative Memory is ignored (forced to a "no match") during Address Preparation.

See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging for description and use, respectively, of the SDHAM.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
Store Segment Descriptor Registers 254 (1)

**FORMAT**: Basic Instruction Format (See Figure 2-1).

**SUMMARY**: For i = 0, 1, ..., 15

- \( C(SDWAH(i).ADDR) \rightarrow C(Y+2i\text{-pair})0, 23 \)
- \( C(SDWAH(i).R1, R2, R3) \rightarrow C(Y+2i\text{-pair})24, 32 \)
- \( 0000 \rightarrow C(Y+2i\text{-pair})33, 36 \)
- \( C(SDWAH(i).BOUND) \rightarrow C(Y+2i\text{-pair})37, 50 \)
- \( C(SDWAH(i).R, E, P, U, G, C) \rightarrow C(Y+2i\text{-pair})51, 57 \)
- \( C(SDWAH(i).CL) \rightarrow C(Y+2i\text{-pair})58, 71 \)

**MODIFICATIONS**: All except DU, DL, CI, SC, SCR

**INDICATORS**: None affected

**NOTES**: The hardware assumes \( Y13,17 = 00000 \), and addressing is incremented accordingly; no check is made.

The contents of SDWAH(i) are unchanged.

The Associative Memory is ignored (forced to a "no match") during Address Preparation.

See Section IV. Program Accessible Registers, and Section V. Addressing -- Segmentation and Paging for description and use, respectively, of the SDWAH.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
Clear Associative Memory Paged

Basic Instruction Format (See Figure 2-1).

For $i = 0, 1, \ldots, 15$

$0 \rightarrow C(PTWAM(i).F)$

$(1) \rightarrow C(PTWAM(i).USE)$

All except DU, DL, CI, SC, SCR

None affected

The Full/Empty bit of each PTWAM Register is set to 0, and the usage counters (PTWAM USE) are set to their pre-assigned values of 0 through 15. The remainder of of $C(PTWAM(i))$ is unchanged.

The execution of this instruction enables the PTWAM if it is disabled and $C(TPR.CA)_{16,17} = 01$.

The execution of this instruction disables the PTWAM if $C(TPR.CA)_{16,17} = 10$.

If $C(TPR.CA)_{15} = 1$, a selective clear of cache is executed. Any cache block for which the upper 14 bits of the directory entry equal $C(TPR.CA)_{0,13}$ will have its Full/Empty bit set to Empty.

See Section IV, Program Accessible Registers and Section V, Addressing -- Segmentation and Paging for description and use, respectively, of the PTWAM.

Attempted execution in Normal or B&H Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure fault.
CAHS FORHATa SUMHARya

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: The Full/Empty bit of each SDWAM Register is set to zero, and the usage counters (SDWAM USE) are initialized to their pre-assigned values of 0 through 15. The remainder of C(SDWAM(i)) are unchanged.

The execution of this instruction enables the SDWAM if it is previously disabled and if C(TPR,CA)16,17 = 01.

The execution of this instruction disables the SDWAM if C(TPR,CA)16,17 = 10.

The execution of this instruction sets the Full/Empty bits of all cache blocks to Empty if C(TPR,CA)15 = 1.

See Section IV, Program Accessible Registers, and Section V, Addressing -- Segmentation and Paging for description and use, respectively, of the SDWAM.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
PRIVILEGED - CONFIGURATION AND STATUS

"Privileged - Configuration and Status"

RMCM Read Memory Controller Mask Register 233 (0)

FORMAT: Basic Instruction Format (See Figure 2-1).

SUMMARY: For the selected System Controller:

If the Processor has a Mask Register assigned, then
C(MR)0,15 -> C(AQ)0,15
00...0 -> C(AQ)16,31
C(MR)32,35 -> C(AQ)32,35
C(MR)36,51 -> C(AQ)36,51
00...0 -> C(AQ)52,67
C(MR)68,71 -> C(AQ)68,71
otherwise, 00...0 -> C(AQ)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: (Indicators not listed are not affected)

Zero If C(AQ) = 0, then ON; otherwise OFF
Negative If C(AQ) = 1, then ON; otherwise OFF

NOTES: The contents of the Mask Register remain unchanged.

C(TPR.CA)0,2 specify which Processor Port (i.e., which System Controller) is used.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975 2-164 AL39
**RSCR**

**Read System Controller Register**

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** The effective address, Y, is used to select a system controller (SCU) and the function to be performed as follows:

<table>
<thead>
<tr>
<th>Effective Address</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>y0000x</td>
<td>C(SCU Mode Register) -&gt; C(AQ)</td>
</tr>
<tr>
<td>y0001x</td>
<td>C(SCU Configuration Switches) -&gt; C(AQ)</td>
</tr>
<tr>
<td>y0002x</td>
<td>C(Interrupt Mask Port 0) -&gt; C(AQ)</td>
</tr>
<tr>
<td>y0012x</td>
<td>C(Interrupt Mask Port 1) -&gt; C(AQ)</td>
</tr>
<tr>
<td>y0022x</td>
<td>C(Interrupt Mask Port 2) -&gt; C(AQ)</td>
</tr>
<tr>
<td>y0032x</td>
<td>C(Interrupt Mask Port 3) -&gt; C(AQ)</td>
</tr>
<tr>
<td>y0042x</td>
<td>C(Interrupt Mask Port 4) -&gt; C(AQ)</td>
</tr>
<tr>
<td>y0052x</td>
<td>C(Interrupt Mask Port 5) -&gt; C(AQ)</td>
</tr>
<tr>
<td>y0062x</td>
<td>C(Interrupt Mask Port 6) -&gt; C(AQ)</td>
</tr>
<tr>
<td>y0072x</td>
<td>C(Interrupt Mask Port 7) -&gt; C(AQ)</td>
</tr>
<tr>
<td>y0003x</td>
<td>C(Interrupt Cells) -&gt; C(AQ)</td>
</tr>
<tr>
<td>y0004x or y0005x</td>
<td>C(System Clock) -&gt; C(AQ)</td>
</tr>
<tr>
<td>y0006x or y0007x</td>
<td>C(Store Unit Mode Register) -&gt; C(AQ)</td>
</tr>
</tbody>
</table>

where: \( y = \text{octal value of } y_0,2 \text{ as used to select SCU} \\
\( x = \text{any octal digit} \)

**MODIFICATIONS:** All except DU, DL, CI, SC, SCR

**INDICATORS:** None affected

**NOTES:** See Section IV, Program Accessible Registers, for description and use of the various registers.

For effective addresses y0006x and y0007x, Store Unit selection is done by the normal address decoding function of the System Controller.

---

*Privileged - Configuration and Status*

**REVIEW DRAFT**

**SUBJECT TO CHANGE**

**October, 1975**
**PRIVILEGED - CONFIGURATION AND STATUS**

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPL causes an Illegal Procedure Fault.

**RSW**

Read Switches

231 (0)

**FORMAT**

Basic Instruction Format (See Figure 2-1).

**SUMMARY**

The effective address, Y, is used to select certain Processor switches whose settings are read into C(A).

The switches selected are as follows:

**Effective Address Function**

<table>
<thead>
<tr>
<th>Address</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxxx0</td>
<td>C(Data Switches) -&gt; C(A)</td>
</tr>
<tr>
<td>xxxxx1</td>
<td>C(Config. Switches, ports A, B, C, D) -&gt; C(A)</td>
</tr>
<tr>
<td>xxxxx2</td>
<td>00...0 -&gt; C(A)0,5</td>
</tr>
<tr>
<td></td>
<td>C(Fault Base Switches) -&gt; C(A)6,12</td>
</tr>
<tr>
<td></td>
<td>00...0 -&gt; C(A)13,26</td>
</tr>
<tr>
<td></td>
<td>C(Processor ID) -&gt; C(A)27,33</td>
</tr>
<tr>
<td></td>
<td>C(Processor Number Switches) -&gt; C(A)34,35</td>
</tr>
<tr>
<td>xxxxx3</td>
<td>C(Config. Switches, ports E, F, G, H) -&gt; C(A)</td>
</tr>
<tr>
<td>xxxxx4</td>
<td>00...0 -&gt; C(A)0,12</td>
</tr>
<tr>
<td></td>
<td>C(Port Interface and Size Switches) -&gt; C(A)13,28</td>
</tr>
<tr>
<td></td>
<td>00...0 -&gt; C(A)29,35</td>
</tr>
</tbody>
</table>

**MODIFICATIONS**

All, but none affect instruction execution

**INDICATORS**

(Indicators not listed are not affected)

- **Zero**
  - If C(A) = 0, then ON; otherwise OFF

- **Negative**
  - If C(A)0 = 1, then ON; otherwise OFF

**NOTES**

See Section IV, Program Accessible Registers for description and use of the switch data.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
"Privileged-System-Control"

**CIOC**

**Connect I/O Channel**

**FORMAT:**

Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

The System Controller addressed by Y (i.e., contains the word at Y) sends a connect pulse to the port specified by C(Y)33,35.

**MODIFICATIONS:**

All except DU, DL, CI, SC, SCR

**INDICATORS:**

None affected

**NOTES:**

- Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.
- Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

**SMCM**

**Set Memory Controller Mask Register**

**FORMAT:**

Basic Instruction Format (See Figure 2-1).

**SUMMARY:**

For the selected System Controller:

If the Processor has a Mask Register assigned, then

- C(AQ)0,15 -> C(MR)0,15
- C(AQ)32,35 -> C(MR)32,35
- C(AQ)36,51 -> C(MR)36,51
- C(AQ)68,71 -> C(MR)68,71

otherwise, a Store Fault, Not Control, occurs.

**MODIFICATIONS:**

All except DU, DL, CI, SC, SCR

**INDICATORS:**

None affected

**NOTES:**

C(AQ) are unchanged.

C(TPR.CA)0,2 specify which Processor Port (i.e., which System Controller) is used.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.
PRIVILEGED - SYSTEM CONTROL

Attempted repetition with RPL causes an Illegal Procedure Fault.

SMIC

Set Memory Controller Interrupt Cells 451 (D)

FORMAT:
Basic Instruction Format (See Figure 2-1).

SUMMARY:
For i = 0, 1, ..., 15 and C(A)35 = 0:
if C(A)i = 1, then set Interrupt Cell i ON
For i = 0, 1, ..., 15 and C(A)35 = 1:
if C(A)i = 1, then set Interrupt Cell 16+i ON

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
None affected

NOTES:
C(IPT,CA)0,2 specify which Processor Port (i.e., which System Controller) is used.

Attempted execution in Normal or BAR Mode causes an Illegal Procedure Fault.
### PRIVILEGED - SYSTEM CONTROL

<table>
<thead>
<tr>
<th>SSCR</th>
<th>Set System Controller Register</th>
<th>057 (0)</th>
</tr>
</thead>
</table>

### FORMAT:
Basic Instruction Format (See Figure 2-1).

### SUMMARY:
The effective address, Y, is used to select a System Controller (SCU) and the function to be performed as follows:

<table>
<thead>
<tr>
<th>Effective Address</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y0000x</td>
<td>C(AQ) → C(SCU Mode Register)</td>
</tr>
<tr>
<td>Y0001x</td>
<td>Reserved</td>
</tr>
<tr>
<td>Y0002x</td>
<td>C(AQ) → C(Interrupt Mask Port 0)</td>
</tr>
<tr>
<td>Y0012x</td>
<td>C(AQ) → C(Interrupt Mask Port 1)</td>
</tr>
<tr>
<td>Y0022x</td>
<td>C(AQ) → C(Interrupt Mask Port 2)</td>
</tr>
<tr>
<td>Y0032x</td>
<td>C(AQ) → C(Interrupt Mask Port 3)</td>
</tr>
<tr>
<td>Y0042x</td>
<td>C(AQ) → C(Interrupt Mask Port 4)</td>
</tr>
<tr>
<td>Y0052x</td>
<td>C(AQ) → C(Interrupt Mask Port 5)</td>
</tr>
<tr>
<td>Y0062x</td>
<td>C(AQ) → C(Interrupt Mask Port 6)</td>
</tr>
<tr>
<td>Y0072x</td>
<td>C(AQ) → C(Interrupt Mask Port 7)</td>
</tr>
<tr>
<td>Y0003x or Y0006x</td>
<td>C(AQ) → C(Interrupt Cells) (0,15)</td>
</tr>
<tr>
<td></td>
<td>C(AQ)36,31 → C(Interrupt Cells) (16,31)</td>
</tr>
<tr>
<td></td>
<td>C(AQ) → C(Store Unit Mode Register)</td>
</tr>
<tr>
<td>Where: y = octal value of Y0,2 as used to select SCU</td>
<td></td>
</tr>
<tr>
<td>x = any octal digit</td>
<td></td>
</tr>
</tbody>
</table>

### MODIFICATIONS:
All except DU, DL, CI, SC, SCR

### INDICATORS:
None affected

### NOTES:
If the Processor does not have a Mask Register assigned in the selected System Controller, a Store Fault, Not Control, will occur.

For effective addresses Y0006x and Y0007x, Store Unit selection is done by the normal address decoding function of the System Controller.
See Section IV, Program Accessible Registers, for description and use of the various registers.

Attempted execution on Normal or B&R Mode causes an Illegal Procedure Fault.
### ABSA

**Absolute Address to Accumulator**

212 (0)

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:**
- Final Main Store Address -> CIA0, 23
- 00..0 -> CIA24, 35

**MODIFICATIONS:** All except DU, DL, CI, SC, SCR

**INDICATORS:** (Indicators not listed are not affected)
- **Zero**: If CIA = 0, then ON; otherwise OFF
- **Negative**: If CIA = 1, then ON; otherwise OFF

**NOTES:**
- If the ABSA instruction is executed in Absolute mode, CIA will be undefined.
- Attempted execution in Normal or BAR modes causes an Illegal Procedure Fault.
- Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

### DIS

**Delay Until Interrupt Signal**

616 (0)

**FORMAT:** Basic Instruction Format (See Figure 2-1).

**SUMMARY:** No operation takes place, and the Processor does not continue with the next instruction; it waits for an interrupt signal.

**MODIFICATIONS:** All, but none affect instruction execution

**INDICATORS:** None affected

**NOTES:** Attempted execution in Normal or BAR mode causes an Illegal Procedure Fault.
- Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
EIS - ADDRESS REGISTER LOAD

"EXTENDED" INSTRUCTION - SET 1 ("EIS")

"EIS - Address Register Load"

AARn | Alphanumeric Descriptor to ARn 56n (1)
--|---

**FORMAT:**

EIS Single-Word Instruction (See Figure 2-1).

**SUMMARY:**

For \( n = 0, 1, \ldots, \) or 7 as determined by operation code

\[
C(Y)0,17 \rightarrow C(PRn.WORDNO)
\]

If \( C(Y)21,22 = 00 \) (TA code = 0), then

\[
C(Y)18,19 \rightarrow C(PRn.CHAR)
\]

0000 \( \rightarrow C(PRn.BITNO) \)

If \( C(Y)21,22 = 01 \) (TA code = 1), then

\[
(6 \cdot C(Y)18,20) / 9 \rightarrow C(PRn.CHAR)
\]

\[
(6 \cdot C(Y)18,20) \text{ modulo } 9 \rightarrow C(PRn.BITNO)
\]

If \( C(Y)21,22 = 10 \) (TA code = 2), then

\[
C(Y)18,20 / 2 \rightarrow C(PRn.CHAR)
\]

\[
4 \cdot (C(Y)18,20) \text{ modulo } 2 + 1 \rightarrow C(PRn.BITNO)
\]

**MODIFICATIONS:**

All except DU, DL, CI, SC, SCR

**INDICATORS:**

None affected.

**NOTES:**

An alphanumeric descriptor is fetched from \( Y \) and \( C(Y)21,22 \) (TA field) is examined to determine the data type described.

If TA = 0 (9-bit data), \( C(Y)18,19 \) goes to \( C(PRn.CHAR) \) and zeros fill \( C(PRn.BITNO) \).

If TA = 1 (6-bit data) or TA = 2 (4-bit data), \( C(Y)18,20 \) is appropriately translated into an equivalent character and bit position that goes to \( C(PRn.CHAR) \) and \( C(PRn.BITNO) \).

If \( C(Y)21,22 = 11 \) (TA code = 3) an Illegal Procedure Fault occurs.

If \( C(Y)23 = 1 \) an Illegal Procedure Fault occurs.

If \( C(Y)21,22 = 00 \) (TA code = 8) and \( C(Y)20 = 1 \) an Illegal Procedure Fault occurs.

If \( C(Y)21,22 = 01 \) (TA code = 1) and \( C(Y)18,20 = 110 \) or 111 an Illegal Procedure Fault occurs.
EIS - ADDRESS REGISTER LOAD

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LARn

Load Address Register n

76n (1)

FORMAT:
EIS Single-Word Instruction (See Figure 2-1).

SUMMARY:
For \( n = 0, 1, \ldots, 7 \) as determined by operation code
\( C(Y)$0,23 \rightarrow C(ARn) \)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
None affected

NOTES:
Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LAREG

Load Address Registers

463 (1)

FORMAT:
EIS Single-Word Instruction (See Figure 2-1).

SUMMARY:
For \( n = 0, 1, \ldots, 7 \)
\( C(Y+n)$0,23 \rightarrow C(ARn) \)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
None affected

NOTES:
The hardware assumes \( Y15,17 = 000 \) and addressing is incremented accordingly; no check is made.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

LPL

Load Pointers and Lengths

467 (1)

FORMAT:
EIS Single-Word Instruction (See Figure 2-1).

SUMMARY:
\( C(Y$-block$) \rightarrow C(Decimal Unit Control Data) \)

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

- REVIEW DRAFT
- SUBJECT TO CHANGE
- October, 1975
- 2-173
EIS - ADDRESS REGISTER LOAD

INDICATORS: None affected

NOTES: See Section IV, Program Accessible Registers, for description and use of Decimal Unit Control Data.

The hardware assumes Y15,17 = 000 and addressing is incremented accordingly; no check is made.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

NARn Numeric Descriptor to ARn

FORMAT: EIS Single-Word Instruction (See Figure 2-1).

SUMMARY: For n = 0, 1, ..., or 7 as determined by operation code

C(Y)0,17 -> C(PRn.WORDNO)

If C(Y)21 = 0 (TN code = 0), then

C(Y)18,20 -> C(PRn.CHAR)

0000 -> C(PRn.BITNO)

C(Y)21 = 1 (TN code = 1), then

(C(Y)18,20) / 2 -> C(PRn.CHAR)

4 * (C(Y)18,20 modulo 2) + 1 -> C(PRn.BITNO)

MODIFICATIONS: All except DU, DL, CI, SC, SCR

INDICATORS: None affected

NOTES: A numeric descriptor is fetched from Y and C(Y)21 (TN bit) is examined.

If TN = 0 (9-bit data), then C(Y)18,19 go to C(PRn.CHAR) and zeros fill C(PRn.BITNO).

If TN = 1 (4-bit data), C(Y)18,20 is appropriately translated to an equivalent character and bit position that goes to C(PRn.CHAR) and C(PRn.BITNO).

If C(Y)21 = 0 (TN code = 0) and C(Y)20 = 1 an Illegal Procedure Fault occurs.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975

2-174

AL39
**EIS - Address Register Store**

- **EIS - Address Register Store**

<table>
<thead>
<tr>
<th>ARAn</th>
<th>ARn to Alphanumeric Descriptor</th>
<th>56n (1)</th>
</tr>
</thead>
</table>

**FORMAT:**

EIS Single-Word Instruction (See Figure 2-1).

**SUMMARY:**

For \( n = 0, 1, \ldots, \) or 7 as determined by operation code

\[
C(PRn.WORDNO) \rightarrow C(Y)0,17
\]

If \( C(Y)21,22 = 00 \) (TA code = 0), then

\[
C(PRn.CHAR) \rightarrow C(Y)18,19
\]

0 \( \rightarrow C(Y)20 \)

If \( C(Y)21,22 = 01 \) (TA code = 1), then

\[
(9 \times C(PRn.CHAR) + C(PRn.BITNO) \div 6 \rightarrow C(Y)18,20
\]

If \( C(Y)21,22 = 10 \) (TA code = 2), then

\[
(9 \times C(PRn.CHAR) + C(PRn.BITNO) - 1) \div 4 \rightarrow C(Y)18,20
\]

**MODIFICATIONS:**

All except DU, DL, CI, SC, SCR

**INDICATORS:**

None affected

**NOTES:**

This instruction is the inverse of AARn.

The alphanumeric descriptor is fetched from Y and C(Y)21,22 (TA field) is examined to determine the data type described.

If TA = 0 (9-bit data), C(PRn.CHAR) goes to C(Y)18,19.

If TA = 1 (6-bit data) or TA = 2 (4-bit data), C(PRn.CHAR) and C(PRn.BITNO) are translated to an equivalent character position that goes to C(Y)18,20.

If C(Y)21,22 = 11 (TA code = 3) or C(Y)23 = 1 (unused bit), an Illegal Procedure Fault occurs.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

---

**REVIEW DRAFT**

**SUBJECT TO CHANGE**

**October, 1975**

2-175

AL39
EIS - ADDRESS REGISTER STORE

ARNₙ

<table>
<thead>
<tr>
<th>ARₙ to Numeric Descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>6ₙn (1)</td>
</tr>
</tbody>
</table>

**FORMAT:**
EIS Single-Word Instruction (See Figure 2-1).

**SUMMARY:**
For n = 0, 1, ..., or 7 as determined by operation code

- C(PRₙ.WORDNO) -> C(Y)₀,₁₇
- If C(Y)₂₁ = 0 (TN code = 0), then
  - C(PRₙ.CHAR) -> C(Y)₁₈,₁₉
  - 0 -> C(Y)₂₀
- If C(Y)₂₁ = 1 (TN code = 1), then
  - (9 * C(PRₙ.CHAR) + C(PRₙ.BITNO) - 1) / 4 -> C(Y)₁₈,₂₀

**MODIFICATIONS:**
All except DU, DL, CI, SC, SCR

**INDICATORS:**
None affected

**NOTES:**
This instruction is the inverse of NARₙ.

The numeric descriptor is fetched from Y and C(Y)₂₁ (TN bit) is examined.

- If TN = 0 (9-bit data), then C(PRₙ.CHAR) goes to C(Y)₁₈,₁₉.
- If TN = 1 (4-bit data), then C(PRₙ.CHAR) and C(PRₙ.BITNO) are translated to an equivalent character position that goes to C(Y)₁₈,₂₀.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SARₙ

<table>
<thead>
<tr>
<th>Store Address Register n</th>
</tr>
</thead>
<tbody>
<tr>
<td>7ₙn (1)</td>
</tr>
</tbody>
</table>

**FORMAT:**
EIS Single-Word Instruction (See Figure 2-1).

**SUMMARY:**
For n = 0, 1, ..., or 7 as determined by operation code

- C(PRₙ.AR) -> C(Y)₀,₂₃
- 00...₀ -> C(Y)₂₄,₃₅

**MODIFICATIONS:**
All except DU, DL, CI, SC, SCR

**INDICATORS:**
None affected

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975
2-176
EIS - ADDRESS REGISTER STORE

NOTES:
Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SAREG
Store Address Registers

FORMAT:
EIS Single-Word Instruction (See Figure 2-1).

SUMMARY:
For \( n = 0, 1, \ldots, 7 \)
\[
\text{C(ARn)} \rightarrow \text{C(Y+n)0,23}
\]
\[
00000 \rightarrow \text{C(Y+n)24,35}
\]

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
None affected

NOTE:
The hardware assumes \( Y15,17 = 000 \) and addressing is incremented accordingly; no check is made.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SPL
Store Pointers and Lengths

FORMAT:
EIS Single-Word Instruction (See Figure 2-1).

SUMMARY:
\[
\text{C(Decimal Unit Control Data)} \rightarrow \text{C(Y-block8)}
\]

MODIFICATIONS:
All except DU, DL, CI, SC, SCR

INDICATORS:
None affected

NOTE:
The hardware assumes \( Y15,17 = 000 \) and addressing is incremented accordingly; no check is made.

See Section IV, Program Accessible Registers, for description and use of Decimal Unit Control Data.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
EIS - ADDRESS REGISTER SPECIAL ARITHMETIC

Figure 2-12 EIS Address Register Special Arithmetic Instruction Format

ARN Number of Address Register selected
ADDRESS Literal word displacement value
OPCODE Instruction operation code
I Program Interrupt inhibit bit
A Use Address Register contents flag
REG Any Register Modifier except DU, DL, and IC

SUMMARY: If A = 0, then
ADDRESS + C(REG) / 4 -> C(PRn.WORDN)
C(REG) modulo 4 -> C(PRn.CHAR)
4 * (C(REG) modulo 2) + 1 -> C(PRn.BITNO)

If A = 1, then
C(PRn.WORDN) + ADDRESS + (9 * C(PRn.CHAR) + 4 * C(REG) + C(PRn.BITNO)) / 36 -> C(PRn.WORDN)
((9 * C(PRn.CHAR) + 4 * C(REG) + C(PRn.BITNO)) modulo 36) / 9 -> C(PRn.CHAR)
4 * (C(PRn.CHAR) + 2 * C(REG) + C(PRn.BITNO) / 4) modulo 2 + 1 -> C(PRn.BITNO)

MODIFICATIONS: None except AU, QU, AL, QL, or Xn

INDICATORS: None affected

NOTES: The steps described in SUMMARY define special 4-bit addition arithmetic for ADDRESS, C(REG), C(PRn.WORDN), C(PRn.CHAR), and C(PRn.BITNO).
The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal procedure Fault.

Add 6-Bit Displacement to Address Register

If A = 0, then

ADDRESS + C(REG) / 6 -> C(PRn.WORDNO)

((6 * C(REG)) modulo 36) / 9 -> C(PRn.CHAR)

If A = 1, then

C(PRn.WORDNO) + ADDRESS + (9 * C(PRn.CHAR) + 6 * C(REG) + C(PRn.BITNO)) / 36 -> C(PRn.WORDNO)

((9 * C(PRn.CHAR) + 6 * C(REG) + C(PRn.BITNO)) modulo 36) / 9 -> C(PRn.CHAR)

None except AU, QU, AL, QL, and Xn

None Affected

The steps described in SUMMARY define special 6-bit addition arithmetic for ADDRESS, C(REG), C(PRn.WORDNO), C(PRn.CHAR), and C(PRn.BITNO).

The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
### A98D

**Add 9-Bit Displacement to Address Register**

500 (1)

#### FORMAT:

EIS Address Register Special Arithmetic Instruction

(See Figure 2-12).

#### SUMMARY:

If $A = 0$, then

- $ADDRESS + C(REG)/4 \rightarrow C(PRn.WORDNO)\)

- $C(REG) \mod 4 \rightarrow C(PRn.CHAR)$

If $A = 1$, then

- 

- 

- $C(PRn.WORDNO) + ADDRESS + (C(REG) + C(PRn.CHAR))/4 \rightarrow C(PRn.WORDNO)$

- $(C(PRn.CHAR) + C(REG)) \mod 4 \rightarrow C(PRn.CHAR)$

$0000 \rightarrow C(PRn.BITNO)$

#### MODIFICATIONS:

None except AU, QU, AL, QL, and Xn

#### INDICATORS:

None affected

#### NOTES:

- The steps described in SUMMARY define special 9-bit addition arithmetic for ADDRESS, C(REG), C(PRn.WORDNO), and C(PRn.CHAR).

- The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

- Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

### ABD

**Add Bit Displacement to Address Register**

503 (1)

#### FORMAT:

EIS Address Register Special Arithmetic Instruction

(See Figure 2-12).

#### SUMMARY:

If $A = 0$, then

- $ADDRESS + C(REG)/36 \rightarrow C(PRn.WORDNO)$

- $(C(REG) \mod 36)/9 \rightarrow C(PRn.CHAR)$

- $C(REG) \mod 9 \rightarrow C(PRn.BITNO)$

If $A = 1$, then

- 

- 

- $C(PRn.WORDNO) + ADDRESS + (9 * C(PRn.CHAR) + 36 * C(REG) + C(PRn.BITNO))/36 \rightarrow C(PRn.WORDNO)$

**REVIEW DRAFT**

**SUBJECT TO CHANGE**

**October, 1975**

2-180

**AL39**
The steps described in SUMMARY define special bit addition arithmetic for ADDRESS, C(REG), C(PRn.WORDNO), C(PRn.CHAR), and C(PRn.BITNO).

The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

**MODIFICATIONS:** None except AU, QU, AL, QL, or XN

**INDICATORS:** None affected

**NOTES:**
EIS - ADDRESS REGISTER SPECIAL ARITHMETIC

**S48D**
Subtract 4-bit Displacement from Address Register 522 (1)

**FORMAT**
EIS Address Register Special Arithmetic Instruction
(See Figure 2-12).

**SUMMARY**
If \( A = 0 \), then
- \((\text{ADDRESS} + \text{C(\text{REG})}) / 4) \rightarrow \text{C(PRn.WORDNO)}\)
- \(\text{C(\text{REG})} \mod 4 \rightarrow \text{C(PRn.CHAR)}\)
- \(4 \times (\text{C(\text{REG})} \mod 2) + 1 \rightarrow \text{C(PRn.BITNO)}\)

If \( A = 1 \), then
\[
\begin{align*}
\text{C(PRn.WORDNO) - ADDRESS} &+ (9 \times \text{C(PRn.CHAR)} - 4 \times \text{C(REG)} \\
&+ \text{C(PRn.BITNO))} / 36 \rightarrow \text{C(PRn.WORDNO)}
\end{align*}
\]
\[
\begin{align*}
((9 \times \text{C(PRn.CHAR)} - 4 \times \text{C(REG)} \\
&+ \text{C(PRn.BITNO))} \mod 36) / 9 \rightarrow \text{C(PRn.CHAR)}
\end{align*}
\]
\[
\begin{align*}
4 \times (\text{C(PRn.CHAR)} - 2 \times \text{C(REG)} \\
&+ \text{C(PRn.BITNO)} / 4) \mod 2 + 1 \rightarrow \text{C(PRn.BITNO)}
\end{align*}
\]

**MODIFICATIONS**
None except AU, QU, AL, QL, or Xn

**INDICATORS**
None affected

**NOTES**
The steps described in SUMMARY define special 4-bit subtraction arithmetic for ADDRESS, C(REG), C(PRn.WORDNO), C(PRn.CHAR), and C(PRn.BITNO).

The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal procedure Fault.

**S68D**
Subtract 6-Bit Displacement from Address Register 521(1)

**FORMAT**
EIS Address Register Special Arithmetic Instruction
(See Figure 2-12).

**SUMMARY**
If \( A = 0 \), then
- \((\text{ADDRESS} + \text{C(\text{REG})}) / 6) \rightarrow \text{C(PRn.WORDNO)}\)
- \((6 \times \text{C(\text{REG})}) \mod 36) / 9 \rightarrow \text{C(PRn.CHAR)}\)
- \((6 \times \text{C(\text{REG})}) \mod 9 \rightarrow \text{C(PRn.BITNO)}\)
EIS - ADDRESS REGISTER SPECIAL ARITHMETIC

If A = 1, then
\[ C(PRn.WORDNO) - \text{ADDRESS} + (9 \times C(PRn.CHAR) - 6 \times C(REG) + C(PRn.BITNO))/36 \rightarrow C(PRn.WORDNO) \]
\[ ((9 \times C(PRn.CHAR) - 6 \times C(REG) + C(PRn.BITNO)) \text{ modulo } 36)/9 \rightarrow C(PRn.CHAR) \]
\[ (9 \times C(PRn.CHAR) - 6 \times C(REG) + C(PRn.BITNO)) \text{ modulo } 9 \rightarrow C(PRn.BITNO) \]

MODIFICATIONS: None except AU, QU, AL, QU, and Xn

INDICATORS: None Affected

NOTES: The steps described in SUMMARY define special 6-bit subtraction arithmetic for ADDRESS, C(REG), C(PRn.WORDNO), C(PRn.CHAR), and C(PRn.BITNO).

The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

S9BD
Subtract 9-Bit Displacement from Address Register 520 (1)

FORMAT: EIS Address Register Special Arithmetic Instruction (See Figure 2-12).

SUMMARY: If A = 0, then
\[ - (\text{ADDRESS} + C(REG))/4 \rightarrow C(PRn.WORDNO) \]
\[ - C(REG) \text{ modulo } 4 \rightarrow C(PRn.CHAR) \]
If A = 1, then
\[ C(PRn.WORDNO) - \text{ADDRESS} + (C(PRn.CHAR) - C(REG))/4 \rightarrow C(PRn.CHAR) \]
\[ (C(PRn.CHAR) - C(REG)) \text{ modulo } 4 \rightarrow C(PRn.CHAR) \]
0000 -> C(PRn.BITNO)

MODIFICATIONS: None except AU, QU, AL, QU, or Xn

INDICATORS: None affected

NOTES: The steps described in SUMMARY define special 9-bit subtraction arithmetic for ADDRESS, C(REG), C(PRn.WORDNO), and C(PRn.CHAR).

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975 2-183  AL39
EIS - ADDRESS REGISTER SPECIAL ARITHMETIC

The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

SBD

Subtract Bit Displacement from Address Register 523 (1)

FORMAT:

EIS Address Register Special Arithmetic Instruction (See Figure 2-12).

SUMMARY:

If $A = 0$, then
- $(ADDRESS + C(REG) / 36) \rightarrow C(PRn.WORDNO)$
- $(C(REG) \mod 36) / 9 \rightarrow C(PRn.CHAR)$
- $C(REG) \mod 9 \rightarrow C(PRn.BITNO)$

If $A = 1$, then
$$C(PRn.WORDNO) = ADDRESS + \left(9 \ast C(PRn.CHAR) - 36 \ast C(REG) + C(PRn.BITNO)\right) / 36 \rightarrow C(PRn.WORDNO)$$

$$\left(\left(9 \ast C(PRn.CHAR) - 36 \ast C(REG) + C(PRn.BITNO)\right) \mod 36\right) / 9 \rightarrow C(PRn.CHAR)$$

$$\left(9 \ast C(PRn.CHAR) - 36 \ast C(REG) + C(PRn.BITNO)\right) \mod 9 \rightarrow C(PRn.BITNO)$$

MODIFICATIONS:

None except AU, QU, AL, QL, or Xn

INDICATORS:

None affected

NOTES:

The steps described in SUMMARY define special bit subtraction arithmetic for ADDRESS, C(REG), C(PRn.WORDNO), C(PRn.CHAR), and C(PRn.BITNO).

The use of an Address Register is inherent; the value of bit 29 affects Address Preparation but not instruction decoding.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
SMD

Subtract Word Displacement from Address Register 527 (1)

FORMAT:

EIS Address Register Special Arithmetic Instruction
(See Figure 2-12).

SUMMARY:

If \( A = 0 \), then
\[
- (\text{ADDRESS} + \text{C(REG)}) \to \text{C(PRn.WORDN)}
\]

If \( A = 1 \), then
\[
\text{C(PRn.WORDNO)} - (\text{ADDRESS} + \text{C(REG)}) \to \text{C(PRn.WORDNO)}
\]

\( 00 \to \text{C(PRn.CHAR)} \)

\( 0000 \to \text{C(PRn.BITNO)} \)

MODIFICATIONS:
None except \( \text{AU}, \text{QU}, \text{AL}, \text{QL}, \text{or Xn} \)

INDICATORS:
None Affected

NOTES:
The use of an Address Register is inherent; the value of
bit 29 affects Address Preparation but not instruction
decoding.

Attempted repetition with RPT, RPD, or RPL causes an
Illegal Procedure Fault.
**EIS - ALPHANUMERIC COMPARE**

**CMPC**

Compare Alphanumeric Character Strings

106 (1)

**FORMAT:**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0 0 1 1</th>
<th>1 1 2 2 2 2</th>
<th>2 2 2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1 1 1</td>
<td>7 8 0 1 2 3 4</td>
<td>7 8 2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1 1 1</td>
<td>1 1 1 1 1 1 1</td>
<td>1 1 1</td>
<td>1</td>
</tr>
<tr>
<td>FILL</td>
<td>10 0 1</td>
<td>MF2</td>
<td>1 106 (1)</td>
<td>II</td>
<td>MF1</td>
</tr>
<tr>
<td></td>
<td>1 1 1 1 1 1</td>
<td>1 1 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 9 2</td>
<td>7 1 1 1 1</td>
<td>10 1 1 1 1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Y-char1</td>
<td>1 CN1 1TA1101</td>
<td>N1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 1 1 1 1 1</td>
<td>1 1 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y-char2</td>
<td>1 CN2 10 0 0 1</td>
<td>N2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 1 1 1 1 1</td>
<td>1 1 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 1 1 1 1 1</td>
<td>1 1 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2-13** Compare Alphanumeric Strings (CMPC) EIS Multi-Word Instruction Format

**FILL**

Fill character for string extension

**MF1**

Modification Field for Operand Descriptor 1

**MF2**

Modification Field for Operand Descriptor 2

**I**

Program Interrupt inhibit bit

**Y-char1**

Address of "left-hand" string

**CN1**

First character position of "left-hand" string

**TA1**

Data type of "left-hand" string

**N1**

Length of "left-hand" string

**Y-char2**

Address of "right-hand" string

**CN2**

First character position of "right-hand" string

**N2**

Length of "right-hand" string

**ALH Coding Format:**

```
cmpc (MF1),(MF2),[fill(octalexpression)]
descna Y-char1[(CN1)],N1 n = 4, 6, or 9 (TA1 = 2, 1, or 0)
descna Y-char2[(CN2)],N2 n = 4, 6, or 9 (TA2 is ignored)
```

**SUMMARY:**

For \( i = 1, 2, \ldots \), minimum \((N1,N2)\)

\[ \text{C}((\text{Y-char1}i)1-i \equiv \text{C}((\text{Y-char2}2)i-1} \]

**REVIEW DRAFT**

**SUBJECT TO CHANGE**

**October, 1975**

2-186

AL39
EIS - ALPHANUMERIC COMPARE

If \( N_1 < N_2 \), then for \( i = N_1+1, N_1+2, \ldots, N_2 \)

\[ C(\text{FILL}) \equiv C(Y-\text{char}_2)i-1 \]

If \( N_2 < N_1 \), then for \( i = N_2+1, N_2+2, \ldots, N_1 \)

\[ C(Y-\text{char}_1)i-1 \equiv C(\text{FILL}) \]

**MODIFICATIONS:** None except AU, QU, AL, QL, or Xn for MF1 and MF2

**INDICATORS:** (Indicators not listed are not affected)

- Zero
  
  If \( C(Y-\text{char}_1)i = C(Y-\text{char}_2)i \) for all \( i \), then ON; otherwise, OFF

- Carry
  
  If \( C(Y-\text{char}_1)i < C(Y-\text{char}_2)i \) for any \( i \), then OFF; otherwise ON

**NOTES:**

Both strings are treated as the data type given for the "left-hand" string, TA1. The data type given for the "right-hand" string, TA2, is ignored.

Comparison is made on full 9-bit fields. If the given data type is not 9-bit (TA1 ≠ 0), then characters from \( C(Y-\text{char}_1)i \) and \( C(Y-\text{char}_2)i \) are high-order zero filled. All 9 bits of \( C(\text{FILL}) \) are used.

Instruction execution proceeds until an inequality is found or the larger string length count is exhausted.

If \( MF_k.\text{RL} = 1 \), then \( N_k \) does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If \( MF_k.\text{ID} = 1 \), then the 4th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
Figure 2-14 Scan Characters Double (SCD) EIS Multi-Word Instruction Format

- **MF1**: Modification Field for Operand Descriptor 1
- **MF2**: Modification Field for Operand Descriptor 2
- **I**: Program Interrupt inhibit bit
- **Y-char1**: Address of string
- **CN1**: First character position of string
- **TA1**: Data type of string
- **N1**: Length of string
- **Y-char2**: Address of test character pair
- **CN2**: First character position of test character pair
- **Y3**: Address of compare count word
- **A**: Indirect via Pointer Register flag for Y3
- **REG**: Register modifier for Y3

**ALM Coding Format**

- `scd` = `(MF1),(MF2)`
- `descna` = `Y-char1(CN1),N1`
- `descna` = `Y-char2(CN2)`
- `arg` = `Y3[,tag]`

- \( D = 4, 6, \text{ or } 9 \) (TA1 = 2, 1, or 0)
- \( D = 4, 6, \text{ or } 9 \) (TA2 is ignored)
**SUMMARY**

For \( i = 1, 2, \ldots, N_1 - 1 \)

\[ c(y_{-char1})_{i-1}, i \equiv c(y_{-char2})_{0}, i \]

On instruction completion

\[ 00 \ldots 0 \to c(y_{3})_{0}, i \]

\( i \to c(y_{3})_{12}, 35 \)

**MODIFICATIONS**

None except AU, QU, AL, QL, or Xn for MF1 and REG
None except DU, AU, QU, AL, QL, or Xn for MF2

**INDICATORS**

(Indicators not listed are not affected)

Tally Runout

If the string length count is exhausted without a match, or if \( N_1 = 1 \), then ON; otherwise OFF

**NOTES**

Both the string and the test character pair are treated as the data type given for the string, \( TA_1 \). The data type given for the test character pair, \( TA_2 \), is ignored.

Instruction execution proceeds until a character pair match is found or the string length count is exhausted.

If \( MF_1.RL = 1 \), then \( N_k \) does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If \( MF_1.ID = 1 \), then the \( k \)th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

If \( MF_2.ID = 0 \) and \( MF_2.REG = DU \), then the second word following the Instruction Word does not contain an Operand Descriptor for the test character pair; instead, it contains the test character pair as a Direct Upper operand in bits 0-17.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

**SCDR**

Scan Characters Double in Reverse 121 (1)

**FORMAT**

Same as Scan Characters Double (SCD) (See Figure 2-14).

**SUMMARY**

For \( i = 1, 2, \ldots, N_1 - 1 \)

\[ c(y_{-char1})_{N_1-1}, N_1-i \equiv c(y_{-char2})_{0}, i \]
On instruction completion

\[ 00...0 \rightarrow C(Y3)0,11 \]
\[ i \rightarrow C(Y3)12,35 \]

MODIFICATIONS:
None except AU, QU, AL, QL, or Xn for MF1 and REG
None except DU, AU, QU, AL, QL, or Xn for MF2

INDICATORS:
(Indicators not listed are not affected)

Tally
If the string length count is exhausted without a match,
or if \( N1 = 1 \), then ON; otherwise OFF

Runout

NOTES:
Both the string and the test character pair are treated as
the data type given for the string, TA1. The data type
given for the test character pair, TA2, is ignored.

Instruction execution proceeds until a character pair
match is found or the string length count is exhausted.

If \( HFk.\text{RL} = 1 \), then \( Nk \) does not contain the operand
length; instead, it contains a register code for a
register holding the operand length.

If \( HFk.\text{ID} = 1 \), then the \( k \)-th word following the Instruction
Word does not contain an Operand Descriptor; instead, it
contains an Indirect Pointer to the Operand Descriptor.

If \( HF2.\text{ID} = 0 \) and \( HF2.\text{REG} = DU \), then the second word
following the Instruction Word does not contain an Operand
Descriptor for the test character pair; instead, it
contains the test character pair as a Direct Upper operand
in bits 0,17.

Attempted execution with XED causes an Illegal Procedure
Fault.

Attempted repetition with RPT, RPD, or RPL causes an
Illegal Procedure Fault.
### SCM Scan with Mask

#### FORMAT

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

#### Figure 2-15 Scan with Mask (SCM) EIS Multi-Word Instruction Format

- **MASK**: Comparison bit mask
- **MF1**: Modification Field for Operand Descriptor 1
- **MF2**: Modification Field for Operand Descriptor 2
- **I**: Program Interrupt inhibit bit
- **Y-charn1**: Address of string
- **CN1**: First character position of string
- **TAI**: Data type of string
- **N1**: Length of string
- **Y-charn2**: Address of test character
- **CN2**: First character position of test character
- **Y3**: Address of compare count word
- **A**: Indirect via Pointer Register flag for Y3
- **REG**: Register modifier for Y3

### ALM Coding Format

- `scm (MF1, MF2), mask=octalexpression`
- `descn Y-charn1(CN1), N1`  \( \Delta = 4, 6, \text{ or } 9 \) (TAI = 2, 1, or 0)
- `descn Y-charn2(CN2)`  \( \Delta = 4, 6, \text{ or } 9 \) (TAI2 is ignored)
- `arg Y3(tag)`

**REVIEW DRAFT**
**SUBJECT TO CHANGE**
**October, 1975**  
2-191  
AL39
EIS - ALPHANUMERIC COMPARE

SUMMARY
For characters \( i = 1, 2, \ldots, N_1 \)
For bits \( j = 0, 1, \ldots, 8 \)

\[
C(Z) = C(MASK) \land \\
(C(Y-char11)i-1) \land (C(Y-char2)1))
\]

If \( C(Z)0,8 = 00...0 \), then

\[
00...0 \rightarrow C(Y)0,11 \\
1 \rightarrow C(Y)12,35
\]
otherwise, continue scan of \( C(Y-char11) \)

MODIFICATIONS:
None except AU, QU, AL, QL, or Xn for MF1 and REG
None except DU, AU, QU, AL, QL, or Xn for MF2

INDICATORS:
(Indicators not listed are not affected)

Tally Runout
If the string length count exhausts, then ON;
otherwise, OFF

NOTES:
Both the string and the test character are treated as the
data type given for the string, TA1. The data type given
for the test character, TA2, is ignored.

Instruction execution proceeds until a masked character
match is found or the string length count is exhausted.

Masking and comparison is done on full 9-bit fields. If
the given data type is not 9-bit (TA1 \( \neq 0 \)), then
characters from \( C(Y-char11) \) and \( C(Y-char2) \) are high-order
zero filled. All 9 bits of \( C(MASK) \) are used.

If \( MF1.RL = 1 \), then \( N_1 \) does not contain the operand
length; instead, it contains a register code for a
register holding the operand length.

If \( MF1.ID = 1 \), then the \( k \)th word following the Instruction
Word does not contain an Operand Descriptor; instead, it
contains an Indirect Pointer to the Operand Descriptor.

If \( MF2.ID = 0 \) and \( MF2.REG = DU \), then the second word
following the Instruction Word does not contain an Operand
Descriptor for the test character; instead, it contains the
test character as a Direct Upper operand in bits 0,8.

Attempted execution with XED causes an Illegal Procedure

Fault.

Attempted repetition with RPT, RPD, or RPL causes an
Illegal Procedure Fault.

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975
2-192
AL39
EIS - ALPHANUMERIC COMPARE

SCNR

Scan with Mask in Reverse

FORMAT:
Same as Scan with Mask (SCM) (See Figure 2-15).

SUMMARY:
For characters i = 1, 2, ..., N1
For bits j = 0, 1, ..., 8

C(Z) = C(MASK) &
(C(Y-char1)N1-1) & (C(Y-char2)1))

If C(Z)0,8 = 00...0, then
00...0 -> C(Y)0,11
1 -> C(Y)12,35
otherwise, continue scan of C(Y-char1)

MODIFICATIONS:
None except AU, QU, AL, QL, or Xn for MF1 and REG
None except DU, AU, QU, AL, QL, or Xn for MF2

INDICATORS:
(Indicators not listed are not affected)

Tally
If the string length count exhausts, then ON;
otherwise, OFF

Runout

NOTES:
Both the string and the test character are treated as the
data type given for the string, TAI. The data type given
for the test character, TAZ, is ignored.

Instruction execution proceeds until a masked character
match is found or the string length count is exhausted.

Masking and comparison is done on full 9-bit fields. If
the given data type is not 9-bit (TAI # 0), then
characters from C(Y-char1) and C(Y-char2) are high-order
zero filled. All 9 bits of C(MASK) are used.

If MF1.RL = 1, then N1 does not contain the operand
length; instead, it contains a register code for a
register holding the operand length.

If MF1.ID = 1, then the kth word following the Instruction
Word does not contain an Operand Descriptor; instead, it
contains an Indirect Pointer to the Operand Descriptor.

If MF2.ID = 0 and MF2.REG = DU, then the second word
following the Instruction Word does not contain an Operand
Descriptor for the test character; instead, it contains
the test character as a Direct Upper operand in bits 0,8.

Attempted execution with XED causes an Illegal Procedure
Fault.

Attempted repetition with RPT, RPD, or RPL causes an
Illegal Procedure Fault.
TCT  Test Character and Translate  164 (1)

**FORMAT:**

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</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>10</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>164 (1)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</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>1</td>
<td>Y-char1</td>
<td>CN1</td>
<td>TA1</td>
<td>N1</td>
<td>10</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>151</td>
</tr>
<tr>
<td>1</td>
<td>Y-char92</td>
<td>10</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></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Y3</td>
<td>10</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></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>11</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2-16 Test Character and Translate (TCT) EIS Multi-Word Instruction Format**

- **MF1**: Modification Field for Operand Descriptor 1
- **I**: Program Interrupt inhibit bit
- **Y-char1**: Address of string
- **CN1**: First character position of string
- **TA1**: Data type of string
- **N1**: Length of string
- **Y-char92**: Address of character translation table
- **Y3**: Address of result word
- **A**: Indirect via Pointer Register flag for Y2 and Y3
- **REG**: Register modifier for Y2 and Y3

**ALM Coding Format:**

- **tct**
- **descga**
- **arg**
- **arg**

**SUMMARY:**

For \( i = 1, 2, \ldots, N1 \)

\[
 m = C(Y\text{-char1})i-1
\]

If \( C(Y\text{-char92})m-1 \neq 00\ldots0 \), then

\[
 C(Y\text{-char92})m-1 \rightarrow C(Y3)0,8
\]
EIS - ALPHANUMERIC COMPARE

000 -> C(Y3)9,11
i -> C(Y3)12,35
otherwise, continue scan of C(y-char91)

MODIFICATIONS:
None except AU, QU, AL, QL, or Xn for MF1 and REG

INDICATORS:
(Indicators not listed are not affected)

Tally
Runout
If the string length count exhausts, then ON;
otherwise, OFF

NOTES:
If the data type of the string to be scanned is not 9-bit (IA1 ≠ 0), then characters from C(Y-char91)1 are high-order zero filled in forming the table index, m.

Instruction execution proceeds until a non-zero table entry is found or the string length count is exhausted.

If MF1.RL = 1, then N1 does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MF1.ID = 1, then the first word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

TCTR
Test Character and Translate in Reverse 165 (1)

FORMAT:
Same as Test Character and Translate (TCT)
(See Figure 2-16).

SUMMARY:
For i = 1, 2, ..., N1

m = C(Y-char91)N1-i

If C(Y-char92)m-1 ≠ 00...0, then

C(Y-char92)m-1 -> C(Y3)0,8
000 -> C(Y3)9,11
i -> C(Y3)12,35
otherwise, continue scan of C(y-char91)
MODIFICATIONS: None except AU, QU, AL, QL, or Xn for 9F1 and REG

INDICATORS:

Tally Runout

If the string length count exhausts, then ON; otherwise, OFF

NOTES:

If the data type of the string to be scanned is not 9-bit (TA1 ≠ 0), then characters from CY-chars1 are high-order zero filled in forming the table index, m.

Instruction execution proceeds until a non-zero table entry is found or the string length count is exhausted.

If MF1.RO = 1, then N1 does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MF1.IO = 1, then the first word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

 Attempted execution with XED causes an Illegal Procedure Fault.

 Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
**EIS - Alphanumeric Move**

**MLR**
Move Alphanumeric Left to Right

---

**FORMAT**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>1</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>FILL</td>
<td>TF101</td>
<td>MF2</td>
<td>1</td>
<td>100 (1)</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>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>71</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>71</td>
<td>1</td>
<td>1</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>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</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>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 2-17** Move Alphanumeric Left to Right (MLR) EIS Multi-Word Instruction Format

**FILL**
Fill character for string extension

**T**
Truncation Fault enable bit

**MF1**
Modification Field for Operand Descriptor 1

**MF2**
Modification Field for Operand Descriptor 2

**Y-charg1**
Address of sending string

**CN1**
First character position of sending string

**TA1**
Data type of sending string

**N1**
Length of sending string

**Y-charg2**
Address of receiving string

**CN2**
First character position of receiving string

**TA2**
Data type of receiving string

**N2**
Length of receiving string

**ALH Coding Format**

```plaintext
mlr = (MF1)(MF2)(TF101)(fill(octal expression))(enabelfault)
descna = Y-charg1(CN1),N1
        Y-charg2(CN2),N2
        N = 4, 6, or 9 (TA1 = 2, 1, or 0)
```

**SUMMARY**
For i = 1, 2, ..., minimum (N1,N2)

\[ C(Y-charg1)i-1 \rightarrow C(Y-charg2)i-1 \]
EIS - ALPHANUMERIC MOVE

If \( N_1 < N_2 \), then for \( i = N_1+1, N_1+2, \ldots, N_2 \)

\[
C(\text{FILL}) \rightarrow C(Y\text{-char}2)i-1
\]

**MODIFICATIONS:** None except AU, QU, AL, QL, or Xn for \( \text{MF1} \) and \( \text{MF2} \)

**INDICATORS:** (Indicators not listed are not affected)

- **Truncation**
  
  If \( N_1 > N_2 \) then ON; otherwise OFF

**NOTES:**

If data types are dissimilar (TA1 \( \neq \) TA2), each character is high-order truncated or zero filled, as appropriate, as it is moved. No character conversion takes place.

If \( N_1 > N_2 \), then \( (N_1-N_2) \) trailing characters of \( C(Y\text{-char}1) \) are not moved and the Truncation indicator is set ON.

If \( N_1 < N_2 \) and TA2 = 2 (4-bit data) or 1 (6-bit data), then fill characters are high-order truncated as they are moved to \( C(Y\text{-char}2) \). No character conversion takes place.

If \( N_1 < N_2 \), \( C(\text{FILL})0 = 1 \), TA1 = 1, and TA2 = 2, then \( C(Y\text{-char}1)N1 \) is examined for a GBCD overpunch sign. If a negative overpunch sign is found, then the minus sign character is placed in \( C(Y\text{-char}2)N2 \); otherwise, a plus sign character is placed in \( C(Y\text{-char}2)N2 \).

If \( \text{MFk}.RL = 1 \), then \( Nk \) does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If \( \text{MFk}.ID = 1 \), then the \( k \)th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

\( C(Y\text{-char}1) \) and \( C(Y\text{-char}2) \) may be overlapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string \( C(Y\text{-char}1) \) data is not inadvertently destroyed.

The user of string replication or overlaying is warned that the Decimal Unit addresses the main store in unaligned (not on 0 modulo 8 boundary) units of Y-block8 words and that the overlayed string \( (C(Y\text{-char}2)) \) is not returned to main store until the unit of Y-block8 words is filled or the instruction completes.

If \( T = 1 \) and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Attempted execution with XED causes an Illegal Procedure Fault.
Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

**HRL**

Move Alphanumeric Right to Left

101 (1)

**FORMAT**

Same as Move Alphanumeric Left to Right (MLR)

(See Figure 2-17).

**SUMMARY**

For \( i = 1, 2, \ldots, \text{minimum} \ (N1,N2) \)

\[
C(Y-	ext{char}1)N1-i \to C(Y-	ext{char}2)N2-i
\]

If \( N1 < N2 \), then for \( i = N1+1, N2+1, \ldots, N2 \)

\[
C(\text{FILL}) \to C(Y-	ext{char}2)N2-i
\]

**MODIFICATIONS**

None except AU, QU, AL, QL, or Xn for MF1 and MF2

**INDICATORS**

(Indicators not listed are not affected)

- **Truncation**
  - If \( N1 > N2 \) then ON; otherwise OFF

**NOTES**

If data types are dissimilar \((TA1 \neq TA2)\), each character is high-order truncated or zero filled, as appropriate, as it is moved. No character conversion takes place.

If \( N1 > N2 \), then \((N1-N2)\) leading characters of \( C(Y-	ext{char}1) \) are not moved and the Truncation indicator is set ON.

If \( N1 < N2 \) and \( TA2 = 2 \) (4-bit data) or 1 (6-bit data), then FILL characters are high-order truncated as they are moved to \( C(Y-	ext{char}2) \). No character conversion takes place.

If \( MF3.RL = 1 \), then \( Nk \) does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If \( MF3.ID = 1 \), then the \( k \)th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

\( C(Y-	ext{char}1) \) and \( C(Y-	ext{char}2) \) may be overlapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string \( C(Y-	ext{char}1) \) data is not inadvertently destroyed.

The user of string replication or overlaying is warned that the Decimal Unit addresses the main store in unaligned (not on 0 modulo 8 boundary) units of Y-blocks and that the overlayed string \( C(Y-	ext{char}2) \) is not returned to main store until the unit of Y-blocks is filled or the instruction completes.
If \( I = 1 \) and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

**MVE**

Move Alphanumeric Edited

020 (1)

**Format:**

```
0 0 0 0 0 1 1 1 1 2 2 2 2 2 2 2 3
```

| 0 1 2 | 0 9 0 1 | 7 0 0 1 2 3 4 | 7 8 9 | 5 |
|-------|---------|---------------|-------|
| 1 1   | MF3     | MF2           | 0 2 0 | 1 1 1 1 | MF1 |
| 1 1   | 1 2     | 7 2           | 1 1 1 | 1 1 1 1 | 10 1 | 7 1 |
| Y-char1| CN1     | TA1           | N1    |       |
|       | 1 1     | 1 1           |       |
| Y-char92| CN2     | 0 0 1 0 1     | N2    |       |
|       | 1 1     | 1 1           |       |
| Y-char93| CN3     | TA3           | N3    |       |
|       | 1 1     | 1 1           |       |
|       | 1 8 1   | 3 2 1         | 1 1 1 | 1 2 1 |
```

**Figure 2-18** Move Alphanumeric Edited (MVE) EIS Multi-Word Instruction Format

- **MF1**: Modification Field for Operand Descriptor 1
- **MF2**: Modification Field for Operand Descriptor 2
- **MF3**: Modification Field for Operand Descriptor 3
- **I**: Program interrupt inhibit bit
- **Y-char1**: Address of sending string
- **CN1**: First character position of sending string
- **TA1**: Data type of sending string
- **N1**: Length of sending string
- **Y-char92**: Address of MOP control string
- **CN2**: First character position of MOP control string
- **N2**: Length of MOP control string
- **Y-char93**: Address of receiving string
EIS - ALPHANUMERIC MOVE

CN3
First character position of receiving string

TA3
Data type of receiving string

N3
Length of receiving string

ALM Coding Format:

\[
\text{mve} \quad (\text{MF1}), (\text{MF2}), (\text{MF3})
\]

\[
\text{desc}\alpha \quad Y\text{-char}1(CN1), N1
\]

\[
\text{desc9a} \quad Y\text{-char}2(CN2), N2
\]

\[
\text{desc\alpha} \quad Y\text{-char}3(CN3), N3
\]

The maximum string length is 63. The count fields N1, N2, and N3 are treated as modulo 64 numbers.

The instruction completes normally only if N3 = minimum (N1, N2, N3), that is, if the receiving string is the first to exhaust; otherwise, an Illegal Procedure Fault occurs.

If MF\textsubscript{K}.RL = 1, then N\textsubscript{K} does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MF\textsubscript{K}.ID = 1, then the kth word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

C(Y-char\textsubscript{1}) and C(Y-char\textsubscript{3}) may be overlapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string C(Y-char\textsubscript{1}) data is not inadvertently destroyed.

The user of string replication or overlaying is warned that the Decimal Unit addresses the main store in unaligned (not on 0 modulo 8 boundary) units of Y-block\textsubscript{8} words and that the overlayed string C(Y-char\textsubscript{3}) is not returned to main store until the unit of Y-block\textsubscript{8} words is

SUMMARY:
C(Y-char\textsubscript{1}) \rightarrow C(Y-char\textsubscript{3}) under C(Y-char\textsubscript{2}) MOP control

See "Micro Operations for Edit Instructions" later in this section for details of editing under MOP control.

MODIFICATIONS:
None except AU, QU, AL, QL, or Xn for YF1, MF2, and MF3

INDICATORS:
None affected

NOTES:
If data types are dissimilar (TA1 $\neq$ TA3), each character of C(Y-char\textsubscript{1}) is high-order truncated or zero filled, as appropriate, as it is moved. No character conversion takes place.

If the data type of the receiving string is not 9-bit (TA3 $\neq$ 0), then Insertion Characters are high-order truncated as they are inserted.

The maximum string length is 63. The count fields N1, N2, and N3 are treated as modulo 64 numbers.

The instruction completes normally only if N3 = minimum (N1, N2, N3), that is, if the receiving string is the first to exhaust; otherwise, an Illegal Procedure Fault occurs.

If MF\textsubscript{K}.RL = 1, then N\textsubscript{K} does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MF\textsubscript{K}.ID = 1, then the kth word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

C(Y-char\textsubscript{1}) and C(Y-char\textsubscript{3}) may be overlapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string C(Y-char\textsubscript{1}) data is not inadvertently destroyed.

The user of string replication or overlaying is warned that the Decimal Unit addresses the main store in unaligned (not on 0 modulo 8 boundary) units of Y-block\textsubscript{8} words and that the overlayed string C(Y-char\textsubscript{3}) is not returned to main store until the unit of Y-block\textsubscript{8} words is
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILL</td>
<td>Fill character for string extension</td>
</tr>
<tr>
<td>I</td>
<td>Truncation Fault enable bit</td>
</tr>
<tr>
<td>MF1</td>
<td>Modification Field for Operand Descriptor 1</td>
</tr>
<tr>
<td>MF2</td>
<td>Modification Field for Operand Descriptor 2</td>
</tr>
<tr>
<td>Y-char1</td>
<td>Address of sending string</td>
</tr>
<tr>
<td>CN1</td>
<td>First character position of sending string</td>
</tr>
<tr>
<td>TA1</td>
<td>Data type of sending string</td>
</tr>
<tr>
<td>N1</td>
<td>Length of sending string</td>
</tr>
<tr>
<td>Y-char2</td>
<td>Address of receiving string</td>
</tr>
<tr>
<td>CN2</td>
<td>First character position of receiving string</td>
</tr>
<tr>
<td>TA2</td>
<td>Data type of receiving string</td>
</tr>
<tr>
<td>N2</td>
<td>Length of receiving string</td>
</tr>
<tr>
<td>Y-char93</td>
<td>Address of character translation table</td>
</tr>
</tbody>
</table>

Figure 2-19 Move Alphanumeric with Translation (MVT) EIS Multi-Word Instruction Format
A

Indirect via Pointer Register flag for Y-char93

REG

Register modifier for Y-char93

ALM Coding Format:

mut

{MF1},{MF2},{fill{octal expression}},{enable fault}

descpa

Y-char91{CN1},N1

D = 4, 6, or 9 (TA1 = 2, 1, or 0)

descqja

Y-char92{CN2},N2

D = 4, 6, or 9 (TA2 = 2, 1, or 0)

arg

Y-char93{,tag}

SUMMARY:

For i = 1, 2, ..., minimum (N1,N2)

m = C(Y-char91)i-1

C(Y-char93)m-1 -> C(Y-char92)i-1

If N1 < N2, then for i = N1+1, N1+2, ..., N2

m = C(FILL)

C(Y-char93)m-1 -> C(Y-char92)i-1

MODIFICATIONS:

None except AU, QU, AL, QL, and Xn for MF1, MF2, and REG

INDICATORS:

(Indicators not listed are not affected)

Truncation If N1 > N2 then ON; otherwise OFF

NOTES:

If the data type of the receiving field is not 9-bit (TA2 ≠ 0), then characters from C(Y-char93) are high-order truncated, as appropriate, as they are removed.

If the data type of the sending field is not 9-bit (TA1 ≠ 0), then characters from C(Y-char91) are high-order zero filled when forming the table index.

If N1 > N2, then (N1-N2) leading characters of C(Y-char91) are not moved and the Truncation indicator is set ON.

If MFk*RL = 1, then Nk does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MFk*ID = 1, then the 5th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

C(Y-char91) and C(Y-char92) may be overlapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string (C(Y-char91)) data is not inadvertently destroyed.

The user of string replication or overlaying is warned that the Decimal Unit addresses the main store in
unaligned (not on 0 modulo 8 boundary) units of Y-block8 words and that the overlayed string (C(Y-charg2)) is not returned to main store until the unit of Y-block8 words is filled or the instruction completes.

If T = 1 and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
**EIS - NUMERIC COMPARE**

**CMPN**: Compare Numeric

<table>
<thead>
<tr>
<th>Format</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>MF2</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>(1)</td>
<td>III</td>
<td>MF1</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y-charg1</td>
<td>1</td>
<td>CN1</td>
<td>I</td>
<td>S1</td>
<td>SF1</td>
<td>N1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y-charg2</td>
<td>1</td>
<td>CN2</td>
<td>I</td>
<td>S2</td>
<td>SF2</td>
<td>N2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<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>

Figure 2-20 Compare Numeric (CMPN) EIS Multi-Word Instruction Format

**Key**

- **MF1**: Modification Field for Operand Descriptor 1
- **MF2**: Modification Field for Operand Descriptor 2
- **I**: Program Interrupt inhibit bit
- **Y-charg1**: Address of "left-hand" number
- **CN1**: First character position of "left-hand" number
- **TN1**: Data type of "left-hand" number
- **S1**: Sign and decimal type of "left-hand" number
- **SF1**: Scaling factor of "left-hand" number
- **N1**: Length of "left-hand" number
- **Y-charg2**: Address of "right-hand" number
- **CN2**: First character position of "right-hand" number
- **TN2**: Data type of "right-hand" number
- **S2**: Sign and decimal type of "right-hand" number
- **SF2**: Scaling factor of "right-hand" number
- **N2**: Length of "right-hand" string
EIS - NUMERIC COMPARE

ALM Coding Format:

<table>
<thead>
<tr>
<th>cmpn</th>
<th>descn(f1,l1,n1,s1,t1)</th>
<th>Y-charn1(CN1),N1,SF1</th>
<th>0 = 4 or 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>descn(f1,l1,n1,s1,t1)</td>
<td>Y-charn2(CN2),N2,SF2</td>
<td></td>
<td>0 = 4 or 9</td>
</tr>
</tbody>
</table>

SUMMARY: C(Y-charn1) :: C(Y-charn2) as numeric values

MODIFICATIONS: None except AU, QU, AL, QI, or Xn for MF1 and MF2

INDICATORS:

- Zero
  - If C(Y-charn1) = C(Y-charn2), then ON; otherwise OFF
- Negative
  - If C(Y-charn1) > C(Y-charn2), then ON; otherwise OFF
- Carry
  - If IC(Y-charn1) > IC(Y-charn2), then OFF, otherwise ON

NOTES:

Comparison is made on 4-bit numeric values contained in each character of C(Y-charn). If either given data type is 9-bit (TNk = 0), characters from C(Y-charn9) are high-order truncated to 4 bits before comparison.

Sign characters are located according to information in CNk, SK, and Nk and interpreted as 4-bit fields; 9-bit sign characters are high-order truncated before interpretation. The sign character 15 (octal) is interpreted as a minus sign; all other legal sign characters are interpreted as plus signs.

The position of the decimal point in C(Y-charn) is determined from information in CNk, SK, SFk, and Nk.

Comparison begins at the decimal position corresponding to the first digit of the operand with the larger number of integer digits and ends with the last digit of the operand with the larger number of fraction digits.

Four-bit numeric zeros are used to represent digits to the left of the first given digit of the operand with the smaller number of integer digits.

Four-bit numeric zeros are used to represent digits to the right of the last given digit of the operand with the smaller number of fraction digits.

Instruction execution proceeds until an inequality is found or the larger string length count is exhausted.

If MFk,RL = 1, then Nk does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MFk,ID = 1, then the kth word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.
Detection of a character outside the range [0,11] (octal) in a digit position or a character outside the range [12,17] (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPO, or RPL causes an Illegal Procedure Fault.
EIS - NUMERIC MOVE

Figure 2-21 Move Numeric (MVN) EIS Multi-Word Instruction Format

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>4-bit data sign character control</td>
</tr>
<tr>
<td>T</td>
<td>Truncation Fault enable bit</td>
</tr>
<tr>
<td>R</td>
<td>Rounding flag</td>
</tr>
<tr>
<td>MF1</td>
<td>Modification Field for Operand Descriptor 1</td>
</tr>
<tr>
<td>MF2</td>
<td>Modification Field for Operand Descriptor 2</td>
</tr>
<tr>
<td>I</td>
<td>Program Interrupt inhibit bit</td>
</tr>
<tr>
<td>Y-char_1</td>
<td>Address of sending number</td>
</tr>
<tr>
<td>CN1</td>
<td>First character position of sending number</td>
</tr>
<tr>
<td>a</td>
<td>Data type of sending number</td>
</tr>
<tr>
<td>S1</td>
<td>Sign and decimal type of sending number</td>
</tr>
<tr>
<td>SF1</td>
<td>Scaling factor of sending number</td>
</tr>
<tr>
<td>N1</td>
<td>Length of sending number</td>
</tr>
<tr>
<td>Y-char_2</td>
<td>Address of receiving number</td>
</tr>
<tr>
<td>b</td>
<td>Data type of receiving number</td>
</tr>
<tr>
<td>S2</td>
<td>Sign and decimal type of receiving number</td>
</tr>
<tr>
<td>SF2</td>
<td>Scaling factor of receiving number</td>
</tr>
<tr>
<td>N2</td>
<td>Length of receiving string</td>
</tr>
<tr>
<td>CN2</td>
<td>First character position of receiving number</td>
</tr>
</tbody>
</table>

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975
ALM Coding Format:

\[
\begin{align*}
\text{mwn} & \quad (\text{MF1}),(\text{MF2}),\text{enablefault},\text{round} \\
\text{descn}(f1,l1,s1,t1) & \quad Y\text{-charn1}(\text{CN1}),N1,\text{SF1} \quad n = 4 \text{ or } 9 \\
\text{descn}(f1,l1,s1,t1) & \quad Y\text{-charn2}(\text{CN2}),N2,\text{SF2} \quad n = 4 \text{ or } 9
\end{align*}
\]

**Summary:**

\(Y\text{-charn1}) \) converted and/or rescaled \( \rightarrow Y\text{-charn2})

**Modifications:**

None except \( \text{AU, QU, AL, QL, or Xn for MF1 and MF2} \)

**Indicators:**

(Indicators not listed are not affected)

- **Zero**
  - If \( Y\text{-charn2}) = \text{decimal 0} \), then \( \text{ON} \); otherwise \( \text{OFF} \)

- **Negative**
  - If a minus sign character is moved to \( Y\text{-charn2}) \), then \( \text{ON} \); otherwise \( \text{OFF} \)

- **Truncation**
  - If low-order digit truncation occurs without rounding, then \( \text{ON} \); otherwise \( \text{OFF} \)

- **Overflow**
  - If fixed point integer overflow occurs, then \( \text{ON} \); otherwise unchanged. (See NOTES)

- **Exponent Overflow**
  - If exponent of floating point result exceeds +127, then \( \text{ON} \); otherwise unchanged.

- **Exponent Underflow**
  - If exponent of floating point result is less than -128, then \( \text{ON} \); otherwise unchanged.

**Notes:**

If data types are dissimilar (\( \text{TN1 } \neq \text{TN2} \)), each character is high-order truncated or filled, as appropriate, as it is moved. The fill data used is "00011"b for digit characters and "00010"b for sign characters.

If \( \text{TN2} \) and \( \text{SN2} \) specify a 4-bit signed number and \( \text{SN2} \) specify a 4-bit signed number and \( P = 1 \), then a legal plus sign character in \( Y\text{-charn1}) \) is converted to \( 13 \) (octal) as it is moved.

If \( \text{N2} \) is not large enough to hold the integer part of \( Y\text{-charn1}) \) as rescaled by \( \text{SF2} \), an overflow condition exists; the overflow indicator is set ON and an Overflow Fault occurs. This implies that an unsigned fixed point receiving field has a minimum length of 1 character; a signed fixed point field, 2 characters; and a floating point field, 3 characters.

If \( \text{N2} \) is not large enough to hold all the given digits of

\( Y\text{-charn1}) \) as rescaled by \( \text{SF2} \) and \( R = 0 \), then a truncation condition exists; data movement stops when \( Y\text{-charn2}) \) is filled and the Truncation indicator is set ON. If \( R = 1 \), then the last digit moved is rounded according to the absolute value of the remaining digits of \( Y\text{-charn1}) \) and the instruction completes normally.

If \( \text{MFk, RL} = 1 \), then \( \text{Nk} \) does not contain the operand length; instead, it contains a register code for a register holding the operand length.
If $MFk.IO = 1$, then the $k$th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

$C(Y-char1)$ and $C(Y-char2)$ may be overlapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string ($C(Y-char1)$) data is not inadvertently destroyed. Difficulties may be encountered because of scaling factors and the special treatment of sign characters and floating point exponents.

The user of string replication or overlaying is warned that the Decimal Unit addresses the main store in unaligned (not on 0 modulo 8 boundary) units of $Y$-block8 words and that the overlayed string ($C(Y-char2)$) is not returned to main store until the unit of $Y$-block8 words is filled or the instruction completes.

If $T = 1$ and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range [0,1] (octal) in a digit position or a character outside the range [12,17] (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
Figure 2-22 Move Numeric Edited (MVNE) EIS Multi-Word Instruction Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MF1</td>
<td>Modification Field for Operand Descriptor 1</td>
</tr>
<tr>
<td>MF2</td>
<td>Modification Field for Operand Descriptor 2</td>
</tr>
<tr>
<td>MF3</td>
<td>Modification Field for Operand Descriptor 3</td>
</tr>
<tr>
<td>I</td>
<td>Program Interrupt inhibit bit</td>
</tr>
<tr>
<td>Y-char1</td>
<td>Address of sending string</td>
</tr>
<tr>
<td>CN1</td>
<td>First character position of sending string</td>
</tr>
<tr>
<td>TN1</td>
<td>Data type of sending string</td>
</tr>
<tr>
<td>S1</td>
<td>Sign and decimal type of sending string</td>
</tr>
<tr>
<td>N1</td>
<td>Length of sending string</td>
</tr>
<tr>
<td>Y-char2</td>
<td>Address of MOP control string</td>
</tr>
<tr>
<td>CN2</td>
<td>First character position of MOP control string</td>
</tr>
<tr>
<td>N2</td>
<td>Length of MOP control string</td>
</tr>
<tr>
<td>Y-char3</td>
<td>Address of receiving string</td>
</tr>
<tr>
<td>CN3</td>
<td>First character position of receiving string</td>
</tr>
<tr>
<td>TA3</td>
<td>Data type of receiving string</td>
</tr>
<tr>
<td>N3</td>
<td>Length of receiving string</td>
</tr>
</tbody>
</table>
EIS - NUMERIC MOVE

ALM Coding Format:

<table>
<thead>
<tr>
<th>mvne</th>
<th>(MF1), (MF2), (MF3)</th>
<th>q = 4 or 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>desc0{fl,ls,ns,ts}</td>
<td>Y-char{11(CN1)},N1</td>
<td></td>
</tr>
<tr>
<td>desc9a</td>
<td>Y-char{22(CN2)},N2</td>
<td></td>
</tr>
<tr>
<td>desc9a</td>
<td>Y-char{33(CN3)},N3</td>
<td>q = 4, 6, or 9</td>
</tr>
</tbody>
</table>

SUMMARY:

C(Y-char{1}) -> C(Y-char{3}) under C(Y_char{2}) MOP control

See "Micro Operations for Edit Instructions" later in this section for details of editing under MOP control.

MODIFICATIONS:

None except AU, QU, AL, QL, or Xn for MF1, MF2, and MF3

INDICATORS:

None affected

NOTES:

If data types are dissimilar (TA1 ≠ TA3), each character of C(Y-char{1}) is high-order truncated or zero filled, as appropriate, as it is moved. No character conversion takes place.

If the data type of the receiving string is not 9-bit (TA3 ≠ 0), then Insertion Characters are high-order truncated as they are inserted.

The maximum string length is 63. The count fields N1, N2, and N3 are treated as modulo 64 numbers.

The instruction completes normally only if N3 = minimum (N1, N2, N3), that is, if the receiving string is the first to exhaust; otherwise, an Illegal Procedure Fault occurs.

If MF{k,RL} = 1, then Nk does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MF{k,ID} = 1, then the kth word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

C(Y-char{1}) and C(Y-char{3}) may be overlapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string (C(Y-char{1})) data is not inadvertently destroyed.

The user of string replication or overlaying is warned that the Decimal Unit addresses the main store in unaligned (not on 0 modulo 8 boundary) units of Y-block8 words and that the overlayed string (C(Y-char{3})) is not returned to main store until the unit of Y-block8 words is filled or the instruction completes.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975
2-212
**EIS - Bit String Combine**

**CSL** Combine Bit Strings Left

060 (1)

**FORMATS**

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

**Figure 2-23** Combine Bit Strings Left (CSL) EIS Multi-Word Instruction Format

- **F** Fill bit for string extension
- **BOLR** Boolean result control field
- **-T** Truncation Fault enable bit
- **MF1** Modification Field for Operand Descriptor 1
- **MF2** Modification Field for Operand Descriptor 2
- **I** Program Interrupt inhibit bit
- **Y-bit1** Address of "sending" string
- **C1** First character position of "sending" string
- **B1** First bit position of "sending" string
- **N1** Length of "sending" string
- **Y-bit2** Address of "receiving" string
- **C2** First character position of "receiving" string
- **B2** First bit position of "receiving" string
- **N2** Length of "receiving" string

**ALM Coding Format:**

\[
\text{cs1} \quad \text{(MF1), (MF2), enablefault, bool(Octalexpression), fill(011)}
\]

\[
\text{descb} \quad \text{Y-bit1([BITNO1]), N1}
\]

\[
\text{descb} \quad \text{Y-bit2([BITNO2]), N2}
\]
EIS - BIT STRING COMBINE

SUMMARY:
For \( i = \text{bits } 1, 2, \ldots, \text{minimum } (N_1, N_2) \)
\[ m = C(Y\text{-bit}1)i-1 \| C(Y\text{-bit}2)i-1 \]
\[ C(BOLR)m \rightarrow C(Y\text{-bit}2)i-1 \]
If \( N_1 < N_2 \), then for \( i = N_1+1, N_1+2, \ldots, N_2 \)
\[ m = C(F)i \| C(Y\text{-bit}2)i-1 \]
\[ C(BOLR)m \rightarrow C(Y\text{-bit}2)i-1 \]

MODIFICATIONS:
None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS:
(Indicators not listed are not affected)

Zero
If \( C(Y\text{-bit}2) = 00\ldots0 \), then ON; otherwise OFF

Truncation
If \( N_1 > N_2 \), then ON; otherwise OFF

NOTES:
If \( N_1 > N_2 \), the low order \( (N_1 - N_2) \) bits of \( C(Y\text{-bit}1) \) are not processed and the Truncation indicator is set ON.

The bit pattern in \( C(BOLR) \) defines the Boolean operation to be performed. Any of the sixteen possible Boolean operations may be used. Some common Boolean operations and their BOLR fields are shown below.

<table>
<thead>
<tr>
<th>Operation</th>
<th>C(BOLR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE</td>
<td>0011</td>
</tr>
<tr>
<td>AND</td>
<td>0001</td>
</tr>
<tr>
<td>OR</td>
<td>0111</td>
</tr>
<tr>
<td>NAND</td>
<td>1110</td>
</tr>
<tr>
<td>Exclusive OR</td>
<td>0110</td>
</tr>
<tr>
<td>Clear</td>
<td>0000</td>
</tr>
<tr>
<td>Invert</td>
<td>1100</td>
</tr>
</tbody>
</table>

If \( MF_{XR} = 1 \), then \( N_k \) does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If \( MF_{ID} = 1 \), then the 4th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

\( C(Y\text{-bit}1) \) and \( C(Y\text{-bit}2) \) may be overlapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string \( C(Y\text{-bit}1) \) data is not
inadvertently destroyed.

The user of string replication or overlaying is warned that the Decimal Unit addresses the main store in unaligned (not on 0 modulo 8 boundary) units of Y-block8 words and that the overlayed string (C(Y-bit2)) is not returned to main store until the unit of Y-block8 words is filled or the instruction completes.

If I = 1 and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

**CSR**

Combine Bit Strings Right 061 (1)

**FORMAT**

Same as Combine Strings Left (CSL) (See Figure 2-23).

**SUMMARY**

For \( i = \) bits 1, 2, ..., minimum \( N1,N2 \)

\[
m = C(Y-bit1)N1-i \cup C(Y-bit2)N2-i \]

\[
C(BOLR)m \rightarrow C(Y-bit2)N2-i \]

If \( N1 < N2 \), then for \( i = N1+1, N1+2, ..., N2 \)

\[
m = C(F) I I C(Y-bit2)N2-i \]

\[
C(BOLR)m \rightarrow C(Y-bit2)N2-i \]

**MODIFICATIONS**

None except AU, QU, AL, QL, or Xn for H-1 and HF2

**INDICATORS**

(Indicators not listed are not affected)

Zero If \( C(Y-bit2) = 00...0 \), then ON; otherwise OFF

Truncation If \( N1 > N2 \), then ON; otherwise OFF

**NOTES**

If \( N1 > N2 \), the high order \( (N1-N2) \) bits of \( C(Y-bit1) \) are not processed and the Truncation indicator is set ON.

The bit pattern in \( C(BOLR) \) defines the Boolean operation to be performed. Any of the sixteen possible Boolean operations may be used. See NOTES under Combine Strings Left (CSL) instruction for examples of BOLR.

If MFk, RL = 1, then \( Nk \) does not contain the operand length; instead, it contains a register code for a register holding the operand length.
If \( MFR.ID = 1 \), then the \( k \)th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

\( C(Y\text{-bit1}) \) and \( C(Y\text{-bit2}) \) may be overlapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string \( (C(Y\text{-bit1})) \) data is not inadvertently destroyed.

\( C(Y\text{-bit1}) \) and \( C(Y\text{-bit2}) \) may be overlapping strings; no check is made. This feature is useful for replication of substrings within a larger string, but care must be exercised in the construction of the Operand Descriptors so that sending string \( (C(Y\text{-bit1})) \) data is not inadvertently destroyed.

The user of string replication or overlaying is warned that the Decimal Unit addresses the main store in unaligned (not on 0 modulo 8 boundary) units of Y-block8 words and that the overlaid string \( (C(Y\text{-bit2})) \) is not returned to main store until the unit of Y-block8 words is filled or the instruction completes.

If \( T = 1 \) and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
**EIS - Bit String Compare**

**CMPB**  
Compare Bit Strings  
066 (1)

**FORMAT:**

<table>
<thead>
<tr>
<th></th>
<th>0 0</th>
<th>0 0 1 1</th>
<th>1 1 1 2</th>
<th>2 2 2 2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1</td>
<td>0.9 0.1</td>
<td>7.8 9.0</td>
<td>3.4</td>
<td>7.8 9.0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IF1</td>
<td>0 0 0 0 0 0 0 0 0 0 1 1</td>
<td>MF2</td>
<td>1</td>
<td>066 (1)</td>
<td>I I I</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>I1</td>
<td>8 1 1</td>
<td>7 1</td>
<td>1</td>
<td>10 1</td>
<td>7 1</td>
</tr>
<tr>
<td>I</td>
<td>Y-bit1</td>
<td>1</td>
<td>C1 1</td>
<td>B1 1</td>
<td>N1 1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>Y-bit2</td>
<td>1</td>
<td>C2 1</td>
<td>B2 1</td>
<td>N2 1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 2-24** Compare Bit Strings (CMPB) EIS Multi-Word Instruction Format

- **F**: Fill bit for string extension
- **T**: Truncation Fault enable bit
- **MF1**: Modification Field for Operand Descriptor 1
- **MF2**: Modification Field for Operand Descriptor 2
- **I**: Program Interrupt inhibit bit
- **Y-bit1**: Address of "left hand" string
- **C1**: First character position of "left hand" string
- **B1**: First bit position of "left hand" string
- **N1**: Length of "left hand" string
- **Y-bit2**: Address of "right hand" string
- **C2**: First character position of "right hand" string
- **B2**: First bit position of "right hand" string
- **N2**: Length of "right hand" string

**ALM Coding Format:**

- cmpb  (MF1),(MF2)[,.enablefault][,.fill(011)]
- descb  Y-bit1((BITNO1)),N1
- descb  Y-bit2((BITNO2)),N2

**SUMMARY:** For i = 1, 2, ..., minimum (N1,N2)

---

**REVIEW DRAFT**  
**SUBJECT TO CHANGE**  
October, 1975  
2-217  
AII39
EIS - BIT STRING COMPARE

\[ C(Y-blt1)i-1 \equiv C(Y-blt2)i-1 \]

If \( N1 < N2 \), then for \( i = N1+1, N1+2, \ldots, N2 \)

\[ C(FILL) \equiv C(Y-blt2)i-1 \]

If \( N2 < N2 \), then for \( i = N2+1, N2+2, \ldots, N1 \)

\[ C(Y-bit1)i-1 \equiv C(FILL) \]

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for \#F1 and MF2

INDICATORS:

- **Zero**: If \( C(Y-bit1)i = C(Y-bit2)i \) for all \( i \), then ON; otherwise, OFF
- **Carry**: If \( C(Y-bit1)i < C(Y-bit2)i \) for any \( i \), then OFF; otherwise ON

NOTES:

Instruction execution proceeds until an inequality is found or the larger string length count is exhausted.

If \( MFk,RL = 1 \), then \( Nk \) does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If \( MFk,ID = 1 \), then the \( k \)th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

Attempted execution with XEO causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
EIS - BIT STRING SET INDICATORS

SET ZERO AND TRUNCATION INDICATORS WITH BIT STRINGS LEFT

**FORMAT**: Same as Combine Strings Left (CSL) (See Figure 2-23).*

**SUMMARY**: For \( i = \text{bits } 1, 2, \ldots, \text{minimum } (N_1, N_2) \)

\[
m = C(Y\text{-bit}_1)i-1 \oplus C(Y\text{-bit}_2)i-1
\]

If \( C(\text{BOLR})m \neq 0 \), then terminate

If \( N_1 < N_2 \), then for \( i = N_1+1, N_1+2, \ldots, N_2 \)

\[
m = C(F) \oplus C(Y\text{-bit}_2)i-1
\]

If \( C(\text{BOLR})m \neq 0 \), then terminate

**MODIFICATIONS**: None except **AU**, **QU**, **AL**, **QL**, or **Xn** for **MF1** and **MF2**

**INDICATORS**: (Indicators not listed are not affected)

- **Zero**: If \( C(\text{BOLR})m = 0 \) for all \( i \), then ON; otherwise OFF
- **Truncation**: If \( N_1 > N_2 \), then ON; otherwise OFF

**NOTES**: If \( N_1 > N_2 \), the low order \((N_1-N_2)\) bits of \( C(Y\text{-bit}_1) \) are not processed and the Truncation indicator is set ON.

The execution of this instruction is identical to Combine Strings Left (CSL) except that \( C(\text{BOLR})m \) is not placed into \( C(Y\text{-bit}_2)i-1 \).

The bit pattern in \( C(\text{BOLR}) \) defines the Boolean operation to be performed. Any of the sixteen possible Boolean operations may be used. See NOTES under Combine Strings Left (CSL) instruction for examples of \( \text{BOLR} \).

If \( \text{MF}_k(\text{RL}) = 1 \), then \( N_k \) does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If \( \text{MF}_k(\text{IO}) = 1 \), then the \( k \)th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

If \( T = 1 \) and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Attempted execution with **XED** causes an Illegal Procedure Fault.

Attempted repetition with **RPT**, **RPD**, or **RPL** causes an Illegal Procedure Fault.
EIS - BIT STRING SET INDICATORS

SZTR

Set Zero and Truncation Indicators with Bit Strings Right

FORMAT:

Same as Combine Strings Left (CSL) (See Figure 2-23).

SUMMARY:

For \( i = \text{bits } 1, 2, \ldots, \text{minimum } (N_1, N_2) \)

\[
m = C(Y-\text{bit}1)N_1-i \text{ II } C(Y-\text{bit}2)N_2-i
\]

If \( C(BOLR)m \neq 0 \), then terminate

If \( N_1 < N_2 \), then for \( i = N_1+1, N_1+2, \ldots, N_2 \)

\[
m = C(F) \text{ II } C(Y-\text{bit}2)N_2-i
\]

If \( C(BOLR)m \neq 0 \), then terminate

MODIFICATIONS:

None except AU, QU, AL, QL, or \( Xn \) for HF1 and HF2

INDICATORS:

(Indicators not listed are not affected)

Zero

If \( C(BOLR)m = 0 \) for all \( i \), then ON; otherwise OFF

Truncation

If \( N_1 > N_2 \), then ON; otherwise OFF

NOTES:

If \( N_1 > N_2 \), the low order \((N_1-N_2)\) bits of \( C(Y-\text{bit}1) \) are not processed and the Truncation indicator is set ON.

The execution of this instruction is identical to Combine Strings Right (CSR) except that \( C(BOLR)m \) is not placed into \( C(Y-\text{bit}2)N_2-i \).

The bit pattern in \( C(BOLR) \) defines the Boolean operation to be performed. Any of the sixteen possible Boolean operations may be used. See NOTES under Combine Strings Left (CSL) instruction for examples of BOLR.

If \( MFk=RL = 1 \), then \( N_k \) does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If \( MFk=ID = 1 \), then the \( k \)th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

If \( T = 1 \) and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
EIS - DATA CONVERSION

"EIS - Data Conversion"

BTD     Binary to Decimal Convert      301 (1)

FORMAT:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td>8</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Y-char91</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 2-25 Binary to Decimal Convert (BTD) EIS Multi-Word Instruction Format

Key

P     4-bit data sign character control
MF1   Modification Field for Operand Descriptor 1
MF2   Modification Field for Operand Descriptor 2
I     Program Interrupt inhibit bit
Y-char91 Address of binary number
CN1   First character position of binary number
N1    Length of binary number in characters
Y-charn2 Address of decimal number
CN2   First character position of decimal number
S2    Data type of decimal number
N2    Length of decimal number

ALM Coding Format:

btd    (MF1),(MF2)
desc9ns Y-char91((CN1)],N1
descn[ls,ns,ts] Y-charn2((CN2)],N2  n = 4 or 9

SUMMARY: C(Y-char91) converted to decimal -> C(Y-charn2)

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975  2-221  AL39
EIS - DATA CONVERSION

MODIFICATIONS: None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS:

Indicators not listed are not affected

Zero
If C(Y-char2) = decimal 0, then ON; otherwise OFF

Negative
If a minus sign character is moved to C(Y-char2), then ON; otherwise OFF

Overflow
If fixed point integer overflow occurs, then ON; otherwise unchanged (See NOTES)

NOTES:

C(Y-char91) contains a two's complement binary integer aligned on 9-bit character boundaries with length 0 < N1 <= 8.

If TN2 and S2 specify a 4-bit signed number and P = 1, then if C(Y-char91) is positive (bit 0 of C(Y-char91) = 0), then the 13 (octal) plus sign character is moved to C(Y-char2) as appropriate.

The scaling factor of C(Y-char2), SF2, must be 0.

If N2 is not large enough to hold the digits of C(Y-char91) an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault occurs. This implies that an unsigned fixed point receiving field has a minimum length of 1 character and a signed fixed point field, 2 characters.

If MFh.QL = 1, then N9 does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MFh.ID = 1, then the kth word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

C(Y-char91) and C(Y-char2) may be overlapping strings; no check is made.

Attempted conversion to a floating point number (S2 = 0) or attempted use of a scaling factor (SF2 ≠ 0) causes an Illegal Procedure Fault.

If N1 = 0 or N1 > 8 an Illegal Procedure Fault occurs.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
Decimal to Binary Convert

**Format:**

<table>
<thead>
<tr>
<th>Decimals</th>
<th>Binary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 2 2 2 2 2 2</td>
<td>1 1 1 0 1 2 3 4 7 8</td>
<td>Conversion from decimal to binary</td>
</tr>
</tbody>
</table>

### Figure 2-26

**Decimal to Binary Convert (DTB) EIS Multi-Word Instruction Format**

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MF1</td>
<td>Modification Field for Operand Descriptor 1</td>
</tr>
<tr>
<td>MF2</td>
<td>Modification Field for Operand Descriptor 2</td>
</tr>
<tr>
<td>I</td>
<td>Program Interrupt inhibit bit</td>
</tr>
<tr>
<td>Y-char1</td>
<td>Address of decimal number</td>
</tr>
<tr>
<td>CN1</td>
<td>First character position of decimal number</td>
</tr>
<tr>
<td>S1</td>
<td>Data type of decimal number</td>
</tr>
<tr>
<td>N1</td>
<td>Length of decimal number</td>
</tr>
<tr>
<td>Y-char2</td>
<td>Address of binary number</td>
</tr>
<tr>
<td>CN2</td>
<td>First character position of binary number</td>
</tr>
<tr>
<td>N2</td>
<td>Length of binary number in characters</td>
</tr>
</tbody>
</table>

### ALH Coding Format:

- `db` `descn[ls,ns,ts]` `desc9ns` `(MF1),(MF2)` `Y-char1(CN1),N1` `Y-char2(CN2),N2` $a = 4$ or $9$

### Summary:

- C(Y-char1) converted to binary -> C(Y-char2)

### Modifications:

- None except AU, QU, AL, QL, or Xn for MF1 ad MF2

### Indicators:

- (Indicators not listed are not affected)

---

**Review Draft**

**Subject to Change**

October, 1975

--

2-223 AL39
EIS - DATA CONVERSION

Zero
If $C(Y\text{-char92}) = 0$, then ON; otherwise OFF

Negative
If a minus sign character is found in $C(Y\text{-char91})$, then ON; otherwise OFF

Overflow
If fixed point integer overflow occurs, then ON; otherwise unchanged (See NOTES)

NOTES:
$C(Y\text{-char92})$ will contain a two's complement binary integer aligned on 9-bit character boundaries with length $0 < N2 \leq 8$.

The scaling factor of $C(Y\text{-char91})$, SF1, must be 0.

If $N2$ is not large enough to hold the converted value of $C(Y\text{-char91})$, an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault occurs.

If $MFk.\text{RL} = 1$, then $Nk$ does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $MFk.\text{ID} = 1$, then the $k$th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

$C(Y\text{-char91})$ and $C(Y\text{-char92})$ may be overlapping strings; no check is made.

Attempted conversion of a floating point number ($S1 = 0$) or attempted use of a scaling factor ($SF1 \neq 0$) causes an Illegal Procedure Fault.

If $N2 = 0$ or $N2 > 8$ an Illegal Procedure Fault occurs.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
**EIS - Decimal Addition**

**AD20**  Add Using 2 Decimal Operands  202 (11)

**FORMAT**

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

**Figure 2-27** Add Using 2 Decimal Operands (AD20) EIS Multi-Word Instruction Format

**Key**

- **P** - 4-bit data sign character control
- **T** - Truncation Fault enable bit
- **R** - Rounding flag
- **MF1** - Modification Field for Operand Descriptor 1
- **MF2** - Modification Field for Operand Descriptor 2
- **I** - Program Interrupt inhibit bit
- **Y-charn1** - Address of augend (AD20), minuend (SB20), multiplicand (MP20), or divisor (DV20)
- **CN1** - First character position of augend (AD20), minuend (SB20), multiplicand (MP20), or divisor (DV20)
- **a TN1** - Data type of augend (AD20), minuend (SB20), multiplicand (MP20), or divisor (DV20)
- **S1** - Sign and decimal type of augend (AD20), minuend (SB20), multiplicand (MP20), or divisor (DV20)
- **SF1** - Scaling factor of augend (AD20), minuend (SB20), multiplicand (MP20), or divisor (DV20)
- **N1** - Length of augend (AD20), minuend (SB20), multiplicand (MP20), or divisor (DV20)
- **Y-charn2** - Address of addend and sum (AD20), subtrahend and difference (SB20), multiplier and product (MP20), or dividend and quotient (DV20)
EIS - DECIMAL ADDITION

CN2
First character position of addend and sum (AD2D), subtrahend and difference (SB2D), multiplier and product (MP2D), or dividend and quotient (DV2D)

b TN2
Data type of addend and sum (AD2D), subtrahend and difference (SB2D), multiplier and product (MP2D), or dividend and quotient (DV2D)

S2
Sign and decimal type of addend and sum (AD2D), subtrahend and difference (SB2D), multiplier and product (MP2D), or dividend and quotient (DV2D)

SF2
Scaling factor of addend and sum (AD2D), subtrahend and difference (SB2D), multiplier and product (MP2D), or dividend and quotient (DV2D)

N2
Length of addend and sum (AD2D), subtrahend and difference (SB2D), multiplier and product (MP2D), or dividend and quotient (DV2D)

ALH Coding Format:

\[
\text{add2d} \quad \{\text{MF1},\{\text{MF2}\},\{\text{enablefault}\},\{\text{round}\}
\]

\[
desc\{f\},\{ls\},\{ns\},\{ts\} = Y-charg1\{CN1\},N1,SF1 \quad n = 4 \text{ or } 9
\]

\[
desc\{f\},\{ls\},\{ns\},\{ts\} = Y-charg2\{CN2\},N2,SF2 \quad n = 4 \text{ or } 9
\]

SUMMARY:
\[
C(Y-charg1) + C(Y-charg2) -> C(Y-charg2)
\]

MODIFICATIONS:
None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS:
(Indicators not listed are not affected)

- **Zero**
  If \(C(Y-charg2) = \text{decimal 0}\), then ON; otherwise OFF

- **Negative**
  If \(C(Y-charg2)\) is negative, then ON; otherwise OFF

- **Truncation**
  If the truncation condition exists without rounding, then ON; otherwise OFF (See NOTES)

- **Overflow**
  If the overflow condition exists, then ON; otherwise unchanged (See NOTES)

- **Exponent**
  If exponent of floating point result exceeds 127 then

- **Overflow**
  ON; otherwise unchanged.

- **Exponent**
  If exponent of floating point result is less than -128

Underflow
then ON; otherwise unchanged

NOTES:
If TN2 and S2 specify a 4-bit signed number and \(P = 1\), then the 13 (octal) plus sign character is placed appropriately if the result of the operation is positive.

If N2 is not large enough to hold the integer part of the result as scaled by SF2, an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975

2-226

AL39
This implies that an unsigned fixed point receiving field has a minimum length of 1 character; a signed fixed point field, 2 characters; and a floating point field, 3 characters.

If \( N2 \) is not large enough to hold all the digits of the result as scaled by \( SF2 \) and \( R = 0 \), then a truncation condition exists; data movement stops when \( C(Y\text{-char}2) \) is filled and the Truncation indicator is set ON. If \( R = 1 \), then the last digit moved is rounded according to the absolute value of the remaining digits of the result and the instruction completes normally.

If \( MFh\text{-RL} = 1 \), then \( Nh \) does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If \( MFh\text{-ID} = 1 \), then the \( k \)th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

\( C(Y\text{-char}1) \) and \( C(Y\text{-char}2) \) may be overlapping strings; no check is made.

If \( T = 1 \) and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range \([0,11]\) (octal) in a digit position or a character outside the range \([12,17]\) (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RP0, or RPL causes an Illegal Procedure Fault.
EIS - DECIMAL ADDITION

Add Using 3 Decimal Operands

**FORMAT**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 1</td>
<td>1 1 2</td>
<td>0 0 1</td>
<td>1 1 1</td>
</tr>
<tr>
<td>0 1 2</td>
<td>0 0 1</td>
<td>0 1 2</td>
<td>0 1 2</td>
<td>1 1 1</td>
</tr>
<tr>
<td>I P I</td>
<td>1 F 1</td>
<td>I T R</td>
<td>M F 2</td>
<td>1 1 1</td>
</tr>
<tr>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>1 1 1</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>1 1 1</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>1 1 1</td>
</tr>
</tbody>
</table>

**Figure 2-28** Add Using 3 Decimal Operands (AD30) EIS Multi-Word Instruction Format

**Key**

- **P**: 4-bit data sign character control
- **-T**: Truncation Fault enable bit
- **R**: Rounding flag
- **MF1**: Modification Field for Operand Descriptor 1
- **MF2**: Modification Field for Operand Descriptor 2
- **MF3**: Modification Field for Operand Descriptor 3
- **I**: Program Interrupt inhibit bit
- **Y-char1**: Address of augend (AD30), minuend (SB30), multiplicand (MP30), or divisor (OV30)
- **CN1**: First character position of augend (AD30), minuend (SB30), multiplicand (MP30), or divisor (OV30)
- **a TN1**: Data type of augend (AD30), minuend (SB30), multiplicand (MP30), or divisor (OV30)
- **S1**: Sign and decimal type of augend (AD30), minuend (SB30), multiplicand (MP30), or divisor (OV30)
- **SF1**: Scaling factor of augend (AD30), minuend (SB30), multiplicand (MP30), or divisor (OV30)
- **N1**: Length of augend (AD30), minuend (SB30), multiplicand (MP30), or divisor (OV30)
- **Y-char2**: Address of addend (AD30), subtrahend (SB30), multiplier (MP30), or dividend (OV30)
CN2  First character position of addend (AO3D), subtrahend (SB3D), multiplier (MP3D), or dividend (OV3D)

b TN2  Data type of addend (AO3D), subtrahend (SB3D), multiplier (MP3D), or dividend (OV3D)

S2  Sign and decimal type of addend (AO3D), subtrahend (SB3D), multiplier (MP3D), or dividend (OV3D)

SF2  Scaling factor of addend (AO3D), subtrahend (SB3D), multiplier (MP3D), or dividend (OV3D)

N2  Length of addend (AO3D), subtrahend (SB3D), multiplier (MP3D), or dividend (OV3D)

Y-char\3  Address of sum (AO3D), difference (SB3D), product (MP3D), or quotient (OV3D)

CN3  First character position of sum (AO3D), difference (SB3D), product (MP3D), or quotient (OV3D)

a TN3  Data type of sum (AO3D), difference (SB3D), product (MP3D), or quotient (OV3D)

S3  Sign and decimal type of sum (AO3D), difference (SB3D), product (MP3D), or quotient (OV3D)

SF3  Scaling factor of sum (AO3D), difference (SB3D), product (MP3D), or quotient (OV3D)

N3  Length of sum (AO3D), difference (SB3D), product (MP3D), or quotient (OV3D)

ALN Coding Format:

\[ \text{ad3d} \quad \text{(MF1), (MF2), (MF3), (enablefault), (round)} \]
\[ \text{desc}[[1, ls, ns, ts)] \quad \text{Y-char\1} \quad \text{(CN1)}, N1, SF1 \quad n = 4 \text{ or } 9 \]
\[ \text{desc}[[1, ls, ns, ts)] \quad \text{Y-char\2} \quad \text{(CN2)}, N2, SF2 \quad n = 4 \text{ or } 9 \]
\[ \text{desc}[[1, ls, ns, ts)] \quad \text{Y-char\3} \quad \text{(CN3)}, N3, SF3 \quad n = 4 \text{ or } 9 \]

SUMMARY:  \( C(\text{Y-char\1}) + C(\text{Y-char\2}) \rightarrow C(\text{Y-char\3}) \)

MODIFICATIONS:  None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS:  (Indicators not listed are not affected)

Zero  If \( C(\text{Y-char\3}) = \text{decimal } 0 \), then ON; otherwise OFF

Negative  If \( C(\text{Y-char\3}) \) is negative, then ON; otherwise OFF

Truncation  If the truncation condition exists without rounding, then ON; otherwise OFF (See NOTES)

Overflow  If the overflow condition exists, then ON; otherwise unchanged (See NOTES)

Exponent Overflow  If exponent of floating point result exceeds 127 then

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975  2-229  AL39
EIS - DECIMAL ADDITION

NOTES:

If exponent of floating point result is less than -128 then ON; otherwise unchanged.

If IN3 and S3 specify a 4-bit signed number and P = 1, then the 13 (octal) plus sign character is placed appropriately if the result of the operation is positive.

If S3 specifies fixed point and N3 is not large enough to hold the integer part of the result as scaled by SF3, an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault occurs. This implies that an unsigned fixed point receiving field has a minimum length of 1 character; a signed fixed point field, 2 characters; and a floating point field, 3 characters.

If N3 is not large enough to hold all the digits of the result as scaled by SF3 and R = 0, then a truncation condition exists; data movement stops when C(Y-charn3) is filled and the Truncation indicator is set ON. If R = 1, then the last digit moved is rounded according to the absolute value of the remaining digits of the result and the instruction completes normally.

If MFk,RL = 1, then Nk does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MFk,IO = 1, then the kth word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

C(Y-charn1), C(Y-charn2), and C(Y-charn3) may be overlapping strings; no check is made.

If T = 1 and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range [0,11] (octal) in a digit position or a character outside the range [12,17] (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
EIS - DECIMAL SUBTRACTION

SB2D

Subtract Using 2 Decimal Operands

203 (1)

FORMAT:
Same as Add Using 2 Decimal Operands (A020)
(See Figure 2-27).

SUMMARY:
C(Y-charn1) - C(Y-charn2) = C(Y-charn2)

MODIFICATIONS:
None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS:
(Indicators not listed are not affected)

Zero
If C(Y-charn2) = decimal 0, then ON; otherwise OFF

Negative
If C(Y-charn2) is negative, then ON; otherwise OFF

Truncation
If the truncation condition exists without rounding, then
ON; otherwise OFF (See NOTES)

Overflow
If the overflow condition exists, then ON; otherwise
unchanged (See NOTES)

Exponent
If exponent of floating point result exceeds 127 then
ON; otherwise unchanged.

Exponent
If exponent of floating point result is less than -128
then ON; otherwise unchanged.

NOTES:
If TN2 and S2 specify a 4-bit signed number and P = 1,
then the 13 (octal) plus sign character is placed
appropriately if the result of the operation is positive.

If N2 is not large enough to hold the integer part of the
result as scaled by SF2, an overflow condition exists;
the Overflow indicator is set ON and an Overflow Fault
occurs. This implies that an unsigned fixed point
receiving field has a minimum length of 1 character; a
signed fixed point field, 2 characters; and a floating
point field, 3 characters.

If N2 is not large enough to hold all the digits of the
result as scaled by SF2 and R = 0, then a truncation
condition exists; data movement stops when C(Y-charn2) is
filled and the Truncation indicator is set ON. If R = 1,
then the last digit moved is rounded according to the

absolute value of the remaining digits of the result and
the instruction completes normally.

If MF1+RL = 1, then Nn does not contain the operand
length; instead, it contains a register code for a
register holding the operand length.

If MF1+ID = 1, then the 5th word following the Instruction
Word does not contain an Operand Descriptor; instead, it
contains an Indirect Pointer to the Operand Descriptor.
C(Y-charn1) and C(Y-charn2) may be overlapping strings; no check is made.

If T = 1 and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range [0, 11] (octal) in a digit position or a character outside the range [12, 17] (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

S83D
Subtract Using 3 Decimal Operands

FORMAT:
Same as Add Using 3 Decimal Operands (A03D)
(See Figure 2-28).

SUMMARY:
C(Y-charn1) - C(Y-charn2) \rightarrow C(Y-charn3)

MODIFICATIONS:
None except AU, QU, AL,QL, or Xn for HF1 and HF2

INDICATORS:
(Indicators not listed are not affected)

Zero
If C(Y-charn3) = decimal 0, then ON; otherwise OFF

Negative
If C(Y-charn3) is negative, then ON; otherwise OFF

Truncation
If the truncation condition exists without rounding, then ON; otherwise OFF (See NOTES)

Overflow
If the overflow condition exists, then ON; otherwise unchanged (See NOTES)

Exponent
Overflow
If exponent of floating point result exceeds 127 then ON; otherwise unchanged.

Exponent
Underflow
If exponent of floating point result is less than -128 then ON; otherwise unchanged

NOTES:
If TN3 and S3 specify a 4-bit signed number and P = 1, then the 13 (octal) plus sign character is placed appropriately if the result of the operation is positive.

If S3 specifies fixed point and N3 is not large enough to hold the integer part of the result as scaled by SF3, an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault occurs. This implies that an unsigned fixed point receiving field has a minimum length of 1 character; a signed fixed point field, 2 characters;
and a floating point field, 3 characters.

If \( N_3 \) is not large enough to hold all the digits of the result as scaled by \( SF_3 \) and \( R = 0 \), then a truncation condition exists; data movement stops when \( C(Y-charn3) \) is filled, and the Truncation indicator is set ON. If \( R = 1 \), then the last digit moved is rounded according to the absolute value of the remaining digits of the result and the instruction completes normally.

If \( MF \cdot RL = 1 \), then \( N_k \) does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If \( MF \cdot ID = 1 \), then the \( k \)th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

\( C(Y-charn1), C(Y-charn2), \) and \( C(Y-charn3) \) may be overlapping strings; no check is made.

If \( T = 1 \) and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range \([0,11]\) (octal) in a digit position or a character outside the range \([12,17]\) (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
DECIMAL MULTIPLICATION

**EIS - Decimal Multiplication**

**MP2D**

Multiply Using 2 Decimal Operands 206 (1)

**FORMAT:**

Same as Add Using 2 Decimal Operands (AD2D)

(See Figure 2-27).

**SUMMARY:**

C(Y-charn1) x C(Y-charn2) -> C(Y-charn2)

**MODIFICATIONS:**

None except AU, QU, AL, QL, or Xn for 4F1 and MF2

**INDICATORS:**

(Indicators not listed are not affected)

- **Zero**
  - If C(Y-charn2) = decimal 0, then ON; otherwise OFF

- **Negative**
  - If C(Y-charn2) is negative, then ON; otherwise OFF

- **Truncation**
  - If the truncation condition exists without rounding, then ON; otherwise OFF (See NOTES)

- **Overflow**
  - If the overflow condition exists, then ON; otherwise unchanged (See NOTES)

- **Exponent Overflow**
  - If exponent of floating point result exceeds 127 then ON; otherwise unchanged.

- **Exponent Underflow**
  - If exponent of floating point result is less than -128 then ON; otherwise unchanged.

**NOTES:**

If TN2 and S2 specify a 4-bit signed number and P = 1, then the 13 (octal) plus sign character is placed appropriately if the result of the operation is positive.

If N2 is not large enough to hold the integer part of the result as scaled by SF2, an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault occurs. This implies that an unsigned fixed point receiving field has a minimum length of 1 character; a signed fixed point field, 2 characters; and a floating point field, 3 characters.

If N2 is not large enough to hold all the digits of the result as scaled by SF2 and R = 0, then a truncation condition exists; data movement stops when C(Y-charn2) is filled and the Truncation indicator is set ON. If R = 1, then the last digit moved is rounded according to the absolute value of the remaining digits of the result and the instruction completes normally.

If MF1 or RL = 1, then Nk does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MF1 or ID = 1, then the kth word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.
C(Y-charn) and C(Y-charo2) may be overlapping strings; no check is made.

If T = 1 and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range [0,11] (octal) in a digit position or a character outside the range [12,17] (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.

Multiply Using 3 Decimal Operands

FORMAT:
Same as Add Using 3 Decimal Operands (A030).
(See Figure 2-28).

SUMMARY:
C(Y-charn) x C(Y-charo2) -> C(Y-charn3)

MODIFICATIONS:
None except AU, QU, AL, QL, or Xn for MF1 and MF2

INDICATORS:
(Indicators not listed are not affected)

Zero
If C(Y-charn3) = decimal 0, then ON; otherwise OFF

Negative
If C(Y-charn3) is negative, then ON; otherwise OFF

Truncation
If the truncation condition exists without rounding, then ON; otherwise OFF (See NOTES)

Overflow
If the overflow condition exists, then ON; otherwise unchanged (See NOTES)

Exponent Overflow
ON; otherwise unchanged.

Exponent Underflow
If exponent of floating point result is less than -128 then ON; otherwise unchanged

NOTES:
If TN3 and S3 specify a 4-bit signed number and P = 1, then the 13 (octal) plus sign character is placed appropriately if the result of the operation is positive.

If S3 specifies fixed point and N3 is not large enough to hold the integer part of the result as scaled by SF3, an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault occurs. This implies that an unsigned fixed point receiving field has a minimum length of 1 character; a signed fixed point field, 2 characters;
and a floating point field, 3 characters.

If N3 is not large enough to hold all the digits of the result as scaled by SF3 and R = 0, then a truncation condition exists; data movement stops when C(Y-char3) is filled and the Truncation indicator is set ON. If R = 1, then the last digit moved is rounded according to the absolute value of the remaining digits of the result and the instruction completes normally.

If MF<sub>RL</sub> = 1, then N<sub>k</sub> does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If MF<sub>ID</sub> = 1, then the kth word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

C(Y-char1), C(Y-char2), and C(Y-char3) may be overlapping strings; no check is made.

If T = 1 and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range [0,11] (octal) in a digit position or a character outside the range [12,17] (octal) in a sign position causes an Illegal Procedure Fault.

At tempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
**EIS - DECIMAL DIVISION**

**EIS - Decimal Division**

**DV2D**
Divide Using 2 Decimal Operands 227 (1)

**FORMAT:**
Same as Add Using 2 Decimal Operands (AD20) (See Figure 2-27).

**SUMMARY:**
C(Y-charn2) / C(Y-charn1) -> C(Y-charn2)

**MODIFICATIONS:**
None except AU, QU, AL, QL, or Xn for MF1 and MF2

**INDICATORS:**
(Indicators not listed are not affected)

- **Zero**
  - If C(Y-charn2) = decimal 0, then ON; otherwise OFF

- **Negative**
  - If C(Y-charn2) is negative, then ON; otherwise OFF

- **Truncation**
  - If the truncation condition exists without rounding, then ON; otherwise OFF (See NOTES)

- **Overflow**
  - If the overflow condition exists, then ON; otherwise unchanged (See NOTES)

- **Exponent Overflow**
  - If exponent of floating point result exceeds 127 then ON; otherwise unchanged.

- **Exponent Underflow**
  - If exponent of floating point result is less than -128 then ON; otherwise unchanged.

**NOTES:**
This instruction performs continued long division on the operands until it has produced enough output digits to satisfy the requirements of the quotient field. The number of required quotient digits, NQ, is determined before division begins as follows ...

1) **Floating point quotient**

   \[ NQ = N2 \]
   
   but if the divisor is greater than the dividend after operand alignment, the leading zero digit produced is counted and the effective precision of the result is reduced by one.

2) **Fixed point quotient**

   \[ NQ = (N2-LZ2+1) - (N1-LZ1) + (E2-E1-SF2) \]

   where:
   - \( NQ \) = given operand field length
   - \( LZn \) = leading zero count for operand \( n \)
   - \( Eq \) = exponent of operand \( n \)
   - \( SF2 \) = scaling factor of quotient

3) **Rounding**

   If rounding is specified \( R = 1 \), then one extra quotient digit is produced.
If \( C(Y\text{-char1}) = \text{decimal 0 or } NQ > 63 \), then division does not take place, \( C(Y\text{-char2}) \) are unchanged, and a Divide Check Fault occurs.

If \( T N2 \) and \( S2 \) specify a 4-bit signed number and \( P = 1 \), then the 13 (octal) plus sign character is placed appropriately if the result of the operation is positive.

If \( N2 \) is not large enough to hold the integer part of the result as scaled by \( SF2 \), an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault occurs. This implies that an unsigned fixed point receiving field has a minimum length of 1 character; a signed fixed point field, 2 characters; and a floating point field, 3 characters.

If \( N2 \) is not large enough to hold all the digits of the result as scaled by \( SF2 \) and \( R = 0 \), then a truncation condition exists; data movement stops when \( C(Y\text{-char2}) \) is filled and the Truncation indicator is set ON. If \( R = 1 \), then the last digit moved is rounded according to the absolute value of the extra quotient digit and the instruction completes normally.

If \( HF_{RL} = 1 \), then \( Nk \) does not contain length; instead, it contains a register code for a register holding the operand length.

If \( HF_{ID} = 1 \), then the \( k \)th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

\( C(Y\text{-char1}) \) and \( C(Y\text{-char2}) \) may be overlapping strings; no check is made.

If \( T = 1 \) and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range \([0,11]\) (octal) in a digit position or a character outside the range \([12,17]\) (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with \( XED \) causes an Illegal Procedure Fault.

Attempted repetition with \( RPT \), \( RPD \), or \( RPL \) causes an Illegal Procedure Fault.

**DV3D**

Divide Using 3 Decimal Operands

**FORMAT**

Same as Add Using 3 Decimal Operands (ADD3D)

(See Figure 2-28).

**SUMMARY**

\[ C(Y\text{-char2}) / C(Y\text{-char1}) \rightarrow C(Y\text{-char3}) \]
MODIFICATIONS:
None except AU, QU, AL, QL, or Xn for HF1 and MF2

INDICATORS:
(Indicators not listed are not affected)

Zero
If C(Y-charn3) = decimal 0, then ON; otherwise OFF

Negative
If C(Y-charn3) is negative, then ON; otherwise OFF

Truncation
If the truncation condition exists without rounding, then ON; otherwise OFF (See NOTES)

Overflow
If the overflow condition exists, then ON; otherwise unchanged (See NOTES)

Exponent
If exponent of floating point result exceeds 127 then ON; otherwise unchanged.

Exponent Overflow
If exponent of floating point result is less than -128 then ON; otherwise unchanged

NOTES:
This instruction performs continued long division on the operands until it has produced enough output digits to satisfy the requirements of the quotient field. The number of required quotient digits, NQ, is determined before division begins as follows...

1) Floating point quotient

NQ = N3, but if the divisor is greater than the dividend after operand alignment, the leading zero digit produced is counted and the effective precision of the result is reduced by one.

2) Fixed point quotient

NQ = (N2-LZ2+1) - (N1-LZ1) + (E2-E1-SF3)

where:
Np = given operand field length
LZp = leading zero count for operand p
Ep = exponent of operand p
SF3 = scaling factor of quotient

3) Rounding

If rounding is specified (R = 1), then one extra quotient digit is produced.

If C(Y-charn1) = decimal 0 or NQ > 63, then division does not take place, C(Y-charn3) are unchanged, and a Divide Check Fault occurs.

If TN3 and S3 specify a 4-bit signed number and P = 1, then the 13 (octal) plus sign character is placed appropriately if the result of the operation is positive.

If S3 specifies fixed point and N3 is not large enough to hold the integer part of the result as scaled by SF3, an overflow condition exists; the Overflow indicator is set ON and an Overflow Fault occurs. This implies that an unsigned fixed point receiving field has a minimum length...
of 1 character; a signed fixed point field, 2 characters; and a floating point field, 3 characters.

If $N3$ is not large enough to hold all the digits of the result as scaled by $SF3$ and $R = 0$, then a truncation condition exists; data movement stops when $C(Y\text{-char}\_3)$ is filled and the Truncation indicator is set ON. If $R = 1$, then the last digit moved is rounded according to the absolute value of the extra quotient digit and the instruction completes normally.

If $MFK\_RL = 1$, then $Nk$ does not contain the operand length; instead, it contains a register code for a register holding the operand length.

If $MFK\_IO = 1$, then the $k$th word following the Instruction Word does not contain an Operand Descriptor; instead, it contains an Indirect Pointer to the Operand Descriptor.

$C(Y\text{-char}\_1)$, $C(Y\text{-char}\_2)$, and $C(Y\text{-char}\_3)$ may be overlapping strings; no check is made.

If $T = 1$ and the Truncation indicator is set ON by execution of the instruction, then a Truncation (Overflow) Fault occurs.

Detection of a character outside the range $[0,11]$ (octal) in a digit position or a character outside the range $[12,17]$ (octal) in a sign position causes an Illegal Procedure Fault.

Attempted execution with XED causes an Illegal Procedure Fault.

Attempted repetition with RPT, RPD, or RPL causes an Illegal Procedure Fault.
The Move Alphanumeric Edited (MVE) and Move Numeric Edited (MVNE) instructions require micro operations to perform the editing functions in an efficient manner. The sequence of micro operation steps to be executed is contained in storage and is referenced by the second operand descriptor of the MVE or MVNE instructions. Some of the micro operations require special characters for insertion into the string of characters being edited. These special characters are shown in the "Edit Insertion Table" discussion below.

**Micro Operation Sequence**

The micro operation string operand descriptor points to a string of 9-bit characters that specify the micro operations to be performed during an edited move. Each of the 9-bit characters defines a micro operation and has the following format:

```
0 0 0 0
0 4 5 6
1 1 1 1
1 1 1 1
```

![Figure 2-29 Micro Operation (MOP) Character Format](image)

- **MOP** 5 bit code specifying Micro Operation to be performed.
- **IF** Information Field containing one of the following:
  1. A sending string character count. A value of 0 is interpreted as 16.
  2. The index of an entry in the edit insertion table to be used. Permissible values are 1 through 8.
  3. An interpretation of the "blank-when-zero" operation

**Edit Insertion Table**

While executing an edit instruction, the Processor provides a register of eight 9-bit characters to hold insertion information. This register, called the "Edit Insertion Table", is not maintained after execution of an edit instruction. At the start of each edit instruction, the Processor hardware initializes the table to the values given in Table 2-8, where each symbol refers to the corresponding standard ASCII character.
Table 2-8 Default Edit Insertion Table Characters

<table>
<thead>
<tr>
<th>Table Entry Number</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>blank</td>
</tr>
<tr>
<td>2</td>
<td>*</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>$</td>
</tr>
<tr>
<td>5</td>
<td>*</td>
</tr>
<tr>
<td>6</td>
<td>*</td>
</tr>
<tr>
<td>7</td>
<td>0 (zero)</td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

One or all of the table entries can be changed by the Load Table Entry or the Change Table micro operations to provide different insertion characters.

**Edit Flags**

The hardware provides the following four "edit flags" for use by the micro operations.

- **ES** End Suppression Flag; initially OFF and set ON by a micro operation when zero suppression ends.
- **SN** Sign Flag; initially set OFF if the sending string is alphanumeric or unsigned numeric. If the sending string is signed numeric, the sending string sign character is tested and SN is set OFF if positive, and ON if negative.
- **Z** Zero Flag; initially set ON. It is set OFF whenever a sending string character that is not decimal zero is moved into the receiving string.
- **BZ** Blank-When-Zero Flag; initially set OFF and is set ON by either the ENF or SES micro operation. If, at the completion of a move, both the Z and BZ are ON, the receiving string is filled with character 1 of the Edit Insertion Table.

**Terminating Micro Operations**

The micro operations are terminated normally when the receive string length becomes exhausted. The micro operations are terminated abnormally (with an Illegal Procedure Fault) if a move from an exhausted sending string or the use of an exhausted MOP string is attempted.
"HVNE and HVE Differences"

The hardware executes HVNE in a slightly different manner than it executes HVE. This is due to the inherent differences in which numeric and alphanumeric data is handled. The following are brief descriptions of the hardware operations for HVNE and HVE.

"NUMERIC EDIT"

1. Load the entire sending string number (maximum length 63 characters) into the Decimal Unit Input Buffer as 4-bit digits (high-order truncating 9-bit data). Strip the sign and exponent characters (if any), put them aside into special holding registers and decrease the Input Buffer count accordingly.

2. Test sign and, if required, set the SN flag.

3. Execute micro operation string, starting with first (4-bit) digit.

4. If an Edit Insertion Table entry or MDP insertion character is to be stored, "ANDed", or "ORed" into a receiving string of 4- or 6-bit characters, high-order truncate the character accordingly.

5. If the receiving string is 9-bit characters, high-order fill the (4-bit) digits from the Input Buffer with bits 0-4 of character 8 of the Edit Insertion Table. If the receiving string is 6-bit characters, high-order fill the digits with "00"b.

"ALPHANUMERIC EDIT"

1. Load Decimal Unit Input Buffer with sending string characters. Data is read from main store in unaligned units (not 0 modulo 8 boundary) of 8-block words. The number of characters read is the minimum of the remaining sending string count, the remaining receiving string count, and 64.

2. Execute micro operation string, starting with the first receiving string character.

3. If an Edit Insertion Table entry or MDP insertion character is to be stored, "ANDed", or "ORed" into a receiving string of 4- or 6-bit characters, high-order truncate the character accordingly.

Micro Operators

A description of the 17 micro operations (MOPs) follows. The mnemonic, name, octal value, and the function performed is given for each MOP in a format similar to that for Processor Instructions. These micro operations are included in the alphabetical list of instructions in Appendix D, identified by the code MOP.

Checks for termination are made during and after each micro operation. All MOPs that make a zero test of a sending string character test only the four least significant bits of the character.
The following additional abbreviations and symbols are used in the descriptions of the HOPs.

**EIT**  
Edit Insertion Table

**pin**  
current position in the sending string

**pmop**  
current position in the micro operation string

**pout**  
current position in the receiving string

- **CHT**  
Change Table

**SUMMARY#**  
For \( i = 1, 2, \ldots, 8 \)
\[
C(Y\text{-char}92)pmop+i \rightarrow C(EIT)i
\]

**FLAGS#**  
None affected

**NOTES#**  
C(IF) is not interpreted for this operation.

- **ENF**  
End Floating Suppression

**SUMMARY#**  
If C(IF)0 = 0, then
  - If ES is OFF, then
    - If SN is OFF, then \( C(EIT)3 \rightarrow C(Y\text{-char}3)pout+1 \)
    - If SN is ON, then \( C(EIT)4 \rightarrow C(Y\text{-char}3)pout+1 \)
    - \( pout = pout + 1 \)
    - ES set ON
  - If ES is ON, then no action

If C(IF)0 = 1, then
  - If ES is OFF, then
    - \( C(EIT)5 \rightarrow C(Y\text{-char}3)pout+1 \)
    - \( pout = pout + 1 \)
  - ES set ON
  - If ES is ON, then no action
  - If C(IF)1 = 1, then BZ set ON; otherwise no action

**FLAGS#**  
(Flags not listed are not affected)
ES
If OFF, then set ON

BZ
If C(IF) = 1, then set ON; otherwise no change

IGN
Ignore Source Character

SUMMARY:
C(IF) \pm \text{pin} \rightarrow \text{pin}

FLAGS:
None affected

INSA
Insert Asterisk on Suppression

SUMMARY:
If ES is OFF, then
C(EIT)_2 \rightarrow C(Y\text{-char}_3)pout+1
If C(IF) = 0, then pmop = pmop + 1
If ES is ON, then
If C(IF) \neq 0, then
m = C(IF)
C(EIT)_m \rightarrow C(Y\text{-char}_3)pout+1
If C(IF) = 0, then
C(Y\text{-char}_2)pmop+1 \rightarrow C(Y\text{-char}_3)pout+1

pmop = pmop + 1

FLAGS:
None affected

NOTES:
If C(IF) > 8 an illegal procedure Fault occurs.

INSB
Insert Blank on Suppression

SUMMARY:
If ES is OFF, then
C(EIT)_1 \rightarrow C(Y\text{-char}_3)pout+1

If C(IF) = 0, then pmop = pmop + 1
If ES is ON, then
If C(IF) \neq 0, then
m = C(IF)
C(EIT)_m \rightarrow C(Y\text{-char}_3)pout+1
If $C(IF) = 0$, then

\[ C(Y-char_92)p_{mop+1} \rightarrow C(Y-char_92)p_{out+1} \]

\[ p_{mop} = p_{mop} + 1 \]

**FLAGS:** None affected

**NOTES:** If $C(IF) > 8$ an illegal Procedure Fault occurs.

**INSN**

Insert Table Entry 1 Multiple

| SUMMARY | For $i = 1, 2, \ldots, C(IF)$
|---------|-----------------
| $C(EIT)_i \rightarrow C(Y-char_92)p_{out+1}$ |

**FLAGS:** None affected

**INSN**

Insert on Negative

| SUMMARY | If SN is OFF, then
|---------|-----------------
| $C(EIT)_i \rightarrow C(Y-char_92)p_{out+1}$ |
| If $C(IF) = 0$, then $p_{mop} = p_{mop} + 1$ |

If SN is ON, then

If $C(IF) \neq 0$, then

\[ m = C(IF) \]

\[ C(EIT)_m \rightarrow C(Y-char_92)p_{out+1} \]

If $C(IF) = 0$, then

\[ C(Y-char_92)p_{mop+1} \rightarrow C(Y-char_92)p_{out+1} \]

\[ p_{mop} = p_{mop} + 1 \]

**FLAGS:** None affected

**NOTES:** If $C(IF) > 8$ an illegal procedure Fault occurs.
Insert on Positive

**SUMMARY:**

If SN is ON, then

\[ C(EIT)_{1} \rightarrow C(Y-char_{3})_{pout+1} \]

If \( C(IF) = 0 \), then \( pmop = pmop + 1 \)

If SN is OFF, then

If \( C(IF) \neq 0 \), then

\[ m = C(IF) \]

\[ C(EIT)m \rightarrow C(Y-char_{3})_{pout+1} \]

If \( C(IF) = 0 \), then

\[ C(Y-char_{92})_{pmop+1} \rightarrow C(Y-char_{3})_{pout+1} \]

\[ pmop = pmop + 1 \]

**FLAGS:**

None affected

**NOTES:**

If \( C(IF) > 8 \) an Illegal Procedure Fault occurs.

**LTE**

Load Table Entry

\[ m = C(IF) \]

\[ C(Y-char_{92})_{pmop+1} \rightarrow C(EIT)m \]

\[ pmop = pmop + 1 \]

**FLAGS:**

None affected

**NOTES:**

If \( C(IF) = 0 \) or \( C(IF) > 8 \) an Illegal Procedure Fault occurs.

**MFLC**

Move with Float Currency Symbol Insertion

**SUMMARY:**

For \( i = 1, 2, \ldots, C(IF) \)

If ES is ON, then \( C(Y-char_{3})_{pout+1} \rightarrow C(Y-char_{3})_{pout+1} \)

If ES is OFF and \( C(Y-char_{3})_{pout+1} = \text{decimal 0} \), then

\[ C(EIT)_{1} \rightarrow C(Y-char_{3})_{pout+1} \]

If ES is OFF and \( C(Y-char_{3})_{pout+1} \neq \text{decimal 0} \), then

\[ C(EIT)_{5} \rightarrow C(Y-char_{3})_{pout+1} \]
C(Y-charg1)pin+i -> C(Y-charq3)pout+i+1
pout = pout + 1
ES set ON

FLAGS:
(Flags not listed are not affected)

ES
If OFF and any of C(Y-charq1)pin+i ≠ decimal 0, then ON; otherwise unchanged

NOTES:
If N1 or N2 exhausts before N3, an Illegal Procedure Fault occurs.

The number of characters moved to the receiving string is data dependent. If the entire C(Y-charq1) is decimal 0's, C(IF) characters are moved to C(Y-charq3). However, if the receiving string contains a non-zero character, then C(IF)+1 characters are moved to C(Y-charq3); the insertion character plus C(Y-charq1). The user is advised that a possible Illegal Procedure Fault due to this condition may be avoided by assuring that the Z and BZ flags are ON.

MFLS
Move with Float Sign Insertion

SUMMARY:
For i = 1, 2, ..., C(IF)
If ES is ON, then C(Y-charq1)pin+i -> C(Y-charq3)pout+i
If ES is OFF and C(Y-charq1)pin+i = decimal 0, then
C(EIF)1 -> C(Y-charq3)pout+i
If ES is OFF and C(Y-charq1)pin+i ≠ decimal 0, then
If SN is OFF, then C(EIF)3 -> C(Y-charq3)pout+i
If SN is ON, then C(EIF)4 -> C(Y-charq3)pout+i
C(Y-charq1)pin+i -> C(Y-charq3)pout+i+1
pout = pout + 1
ES set ON

FLAGS:
(Flags not listed are not affected)

ES
If OFF and any of C(Y-charq1)pin+i ≠ decimal 0, then ON; otherwise unchanged

NOTES:
If N1 or N2 exhausts before N3, an Illegal Procedure Fault occurs.

The number of characters moved to the receiving string is data dependent. If the entire C(Y-charq1) is decimal 0's,
C(IF) characters are moved to C(Y-char3). However, if the receiving string contains a non-zero character, then C(IF)+1 characters are moved to C(Y-char3); the insertion character plus C(Y-char1). The user is advised that a possible Illegal Procedure Fault due to this condition may be avoided by assuring that the Z and 3Z flags are ON.

SUMMARY:

Move and OR Sign

For i = 1, 2, ..., C(IF)

If SN is OFF, then

C(Y-char1)pin+i & C(EIT)3 -> C(Y-char3)pout+i

If SN is ON, then

C(Y-char1)pin+i & C(EIT)4 -> C(Y-char3)pout+i

FLAGS:

None affected

SUMMARY:

Move and Set Sign

For MVNE

For i = 1, 2, ..., C(IF)

C(Y-char1)pin+i -> C(Y-char3)pout+i

For MV

For i = 1, 2, ..., C(IF)

C(Y-char1)pin+i -> C(Y-char3)pout+i

C(Z) = C(Y-char1)pin+i & C(EIT)3

If C(Z) ≠ 0, then for j = i+1, i+2, ..., C(IF)

C(Y-char1)pin+j -> C(Y-char3)pout+j

If C(Z) = 0, then

C(Z) = C(Y-char1)pin+i & C(EIT)4

If C(Z) ≠ 0, then

SN set ON

For j = i+1, i+2, ..., C(IF)

C(Y-char1)pin+j -> C(Y-char3)pout+j

FLAGS:

(Flags not listed are not affected)
<table>
<thead>
<tr>
<th>SN</th>
<th>Move Source Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVC</td>
<td>Move with Zero Suppression and Asterisk Replacement</td>
</tr>
<tr>
<td>SUMMARY:</td>
<td>For ( i = 1, 2, \ldots, C(\text{IF}) )</td>
</tr>
<tr>
<td></td>
<td>( C(Y\text{-char1})<em>{\text{pin}+1} \rightarrow C(Y\text{-char3})</em>{\text{pout}+1} )</td>
</tr>
<tr>
<td>FLAGS:</td>
<td>None affected</td>
</tr>
<tr>
<td>MVZB</td>
<td>Move with Zero Suppression and Blank Replacement</td>
</tr>
<tr>
<td>SUMMARY:</td>
<td>For ( i = 1, 2, \ldots, C(\text{IF}) )</td>
</tr>
<tr>
<td></td>
<td>If ES is ON, then ( C(Y\text{-char1})<em>{\text{pin}+1} \rightarrow C(Y\text{-char3})</em>{\text{pout}+1} )</td>
</tr>
<tr>
<td></td>
<td>If ES is OFF and ( C(Y\text{-char1})<em>{\text{pin}+1} = \text{decimal 0} ), then ( C(\text{EIT})</em>{\text{pout}+1} \rightarrow C(Y\text{-char3})_{\text{pout}+1} )</td>
</tr>
<tr>
<td></td>
<td>If ES is OFF and ( C(Y\text{-char1})<em>{\text{pin}+1} \neq \text{decimal 0} ), then ( C(Y\text{-char1})</em>{\text{pin}+1} \rightarrow C(Y\text{-char3})_{\text{pout}+1} )</td>
</tr>
<tr>
<td>FLAGS:</td>
<td>(Flags not listed are not affected)</td>
</tr>
<tr>
<td>ES</td>
<td>If OFF and any of ( C(Y\text{-char1})_{\text{pin}+1} \neq \text{decimal 0} ), then ON; otherwise unchanged</td>
</tr>
<tr>
<td>NOTES:</td>
<td>If ( N1 ) or ( N2 ) exhausts before ( N3 ), an Illegal Procedure Fault occurs.</td>
</tr>
<tr>
<td>REVIEW DRAFT</td>
<td>SUBJET TO CHANGE</td>
</tr>
<tr>
<td>October, 1975</td>
<td>2-250</td>
</tr>
</tbody>
</table>
FLAGS:

(Flags not listed are not affected)

ES

If OFF and any of (Y-char)pin ≠ decimal 0, then ON; otherwise unchanged

NOTES:

If N1 or N2 exhausts before N3, an Illegal Procedure Fault occurs.

SES

Set End Suppression

SUMMARY:

If C(IF)0 = 0, then ES set OFF
If C(IF)0 = 1, then ES set ON
If C(IF)1 = 1, then BZ set ON; otherwise no action

FLAGS:

(Flags not listed are not affected)

ES

Set by this micro operation

BZ

If C(IF)1 = 1, then ON; otherwise no change

Micro Operation Code Assignment Map

Operation code assignments for the micro operations are shown in Table 2-9 below. (---) indicates an unassigned code. All unassigned codes cause an Illegal Procedure Fault.

Table 2-9 "Micro Operation Code Assignment Map"

<table>
<thead>
<tr>
<th></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>00</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
</tr>
<tr>
<td>10</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
</tr>
<tr>
<td>20</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
</tr>
<tr>
<td>30</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
</tr>
</tbody>
</table>
SECTION III
DATA REPRESENTATION

INFORMATION ORGANIZATION

The Processor, like the rest of the Multics system, is organized to deal with information in basic units of 36-bit "words". Other units of 4-, 6-, 9-bit "characters" or "bytes", 18-bit "half words", and 72-bit "word pairs" can be manipulated within the Processor by use of the instruction set. These bit groupings are used by the hardware and software to represent a variety of forms of coded data. Certain Processor functions appear to manipulate larger units of 144, 288, 576, and 1152 bits, but functions are performed by means of repeated use of 72-bit word pairs. All information is represented as strings of binary bits.

POSITION NUMBERING

The numbering of bit positions, character positions, and words increases in the direction of conventional reading and writing, from the most-significant to the least-significant digit of a number, and from left to right in conventional alphanumeric text.

Graphic presentations in this manual show registers and data with position numbers increasing from left to right.

NUMBER SYSTEM

The arithmetic functions of the Processor are implemented in the two's complement, binary number system. One of the primary properties of this number system is that a field (or register) having width p bits may be interpreted in two different ways; the "logical" case and the "arithmetic" or "algebraic" case.

In the logical case, the number is unsigned, positive, and lies in the range \(0, 2^{*p} - 1\). The results of arithmetic operations on numbers for this case are interpreted as \(0\) modulo \(p\) numbers. Overflow is not defined for this case since the range of the field or register cannot be exceeded. The numbers "0" and "2^{*p} - 1" are consecutive (not separated) in the set of numbers defined for the field or register.
In the arithmetic case, the number is signed and lies in the range \((-2^{(n-1)}, 2^{(n-1)} - 1\). Overflow is defined for this case since since the range can be exceeded in either direction (positive or negative). The left-hand-most bit of the field or register (bit 0) serves as the sign bit and does not contribute to the value of the number.

The main advantage of this implementation is that the hardware arithmetic algorithms for the two cases are identical; the only distinction lying in the interpretation of the results by the user. Instruction set features are provided for performing binary arithmetic with overflow disabled (the so-called logical instructions) and for comparing numbers in either sense.

Subtraction is performed by adding the two's complement of the subtrahend to the minuend. (Note that when the subtrahend is zero the algorithm for forming the two's complement is still carried out, but, since the two's complement of zero is zero, the result is correct.)

Another important feature of the two's complement number system (with respect to comparison of numeric values) is that the "no borrow" condition in true subtraction is identical to the "carry" condition in true addition and vice versa.

A statement on the assumed location of the binary point has significance only for multiplication and division. These two operations are implemented for the arithmetic case in both integer and fraction modes. "Integer" means that the position of the binary point is assumed to the right of the least-significant bit position (that is, to the right of the right-hand-most bit of the field or register) and "fraction" means that the position of the binary point is assumed to the left of the most-significant bit position (that is, between bit 0 and bit 1 of the field or register; recall that bit 0 is the sign bit).

INFORMATION FORMATS

The Figures below show the unstructured formats (templates) for the various information units defined for the Processor. Data transfer between the Processor and main store is word oriented; a 36-bit machine word is transferred for single-precision operands and sub-fields of machine words, and a 72-bit word pair is transferred for all other cases (multi-word operands, instruction fetches, bit- and character-str.

The information unit to be used and the data transfer mode is determined by the Processor according to the function to be performed.

The 36-bit unstructured machine word shown in Figure 3-1 below is the minimum addressable information unit in main store. Its location is uniquely determined by its main store address, Y. All other information units are defined relative to the 36-bit machine word.
Two consecutive machine words as shown in Figure 3-2 below, the first having an even main store address, form a 72-bit word pair. In 72-bit word pair data transfer mode, the word pair is uniquely located by the main store address of either of its constituent 36-bit machine words. Thus, if \( Y \) is even, the word pair at \((Y,Y+1)\) is selected. If \( Y \) is odd, the word pair at \((Y-1,Y)\) is selected. The term "Y-pair" is used for such a word pair address.

4-bit characters are mapped onto 36-bit machine words as shown in Figure 3-3 below. The "0" bits at bit positions 0, 9, 18, and 27 are forced to be 0 by the Processor on data transfers to main store and are ignored on data transfers from main store.

6-bit characters are mapped onto 36-bit machine words as shown in Figure 3-4 below.
Figure 3-4 Unstructured 6-bit Character Format

9-bit characters are mapped onto 36-bit machine words as shown in Figure 3-5 below.

Figure 3-5 Unstructured 9-bit Character Format

18-bit half words are mapped onto 38-bit machine words as shown in Figure 3-6 below.

Figure 3-6 Unstructured 18-bit Half Word Format

DATA PARITY

Odd parity on each 36-bit machine word transferred to main store is generated as it leaves the Processor, is verified at several points along the transmission path, and is held in main store as an "extra" bit. If an incorrect parity is detected at any of the various parity "check points", the main store returns an Illegal Action signal and a code appropriate to the check point.

On data transfers from main store, the parity bit is retrieved and transmitted with the data bits. The same verification checks are made and Illegal Action signalled for errors. The Processor makes a final parity check as the data enters the Processor.
Any detected parity error causes the Processor Parity indicator to set ON and (if enabled) a Parity Fault.

**REPRESENTATION OF DATA**

Data is defined by imposing an operand structure on the information units described above. Data is represented in two forms: numeric or alphanumeric. The form is determined by the Processor according to function to be performed.

**Numeric Data**

Numeric data is represented in three modes: fixed point binary, floating point binary, and decimal. The mode is determined by the Processor according to the function being performed and any Address Modification invoked for the instruction being executed.

**FIXED POINT BINARY DATA**

**Fixed Point Binary Integers**

Fixed point binary integer data is defined by imposing either of the bit position value structures shown below on an information unit of n bits.

**Logical value:**

\[ a(0)x2^{n} + a(1)x2^{(n-1)} + \ldots + a(n-1) \]

\[ \top \]

**Arithmetic value:**

\[ (\neg a(0)) \cdot (a(1)a(0))x2^{n} + (a(2)a(0))x2^{(n-1)} + \ldots + (a(n-1)a(0)) \]

where:

\[ a(i) \text{ is the value of the bit in the } i\text{th bit position} \]

\[ \neg \text{ indicates the Boolean Exclusive OR function} \]

\[ \Top \text{ indicates the position of the binary point} \]

\[ (\neg a(0)) \text{ selects the proper sign according the value of } a(0) \]

The following fixed point binary integer data items are defined:
Note that 4-bit Byte Operands are not defined. This data item is defined only for Decimal Data. (See Decimal Data below.)

The proper operand and its position within a 36-bit machine word is determined by the Processor during preparation of the main store address for the operand. If the data width of the operand selected is smaller than the register involved, the operand is high-order and/or low-order zero filled as necessary.

Table 3-1 Fixed Point Binary Integer Values

<table>
<thead>
<tr>
<th>Operand</th>
<th>6-bit Byte</th>
<th>9-bit Byte</th>
<th>18-bit Half Word</th>
<th>36-bit Single Precision</th>
<th>72-bit Double Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical range</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>(2**6)-1</td>
<td>(2**9)-1</td>
<td>(2**18)-1</td>
<td>(2**36)-1</td>
<td>(2**72)-1</td>
</tr>
<tr>
<td>Resolution</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Arithmetic range</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>-(2**5)</td>
<td>-(2**8)</td>
<td>-(2**17)</td>
<td>-(2**35)</td>
<td>-(2**71)</td>
</tr>
<tr>
<td>Pos.</td>
<td>(2**5)-1</td>
<td>(2**8)-1</td>
<td>(2**17)-1</td>
<td>(2**35)-1</td>
<td>(2**71)-1</td>
</tr>
<tr>
<td>Resolution</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Fixed Point Binary Fractions

Fixed point binary fraction data is defined by imposing the bit position value structure below on an information unit of \( n \) bits.

Arithmetic value:

\[
(\text{not} a(0)) \times 2^{-n} a(1) a(2) \cdots a(n-1) x 2^{-n} + a(0) x 2^{-n} + (n-1)
\]

Note that logical values are not defined for fixed point binary fraction data.

The following fixed point binary fraction data items are defined:
Note that 4-bit Byte Operands and 72-bit Double Precision Operands are not defined. 4-bit Byte Operands are defined only for Decimal Data. (See Decimal Data below.) If the instruction being executed is Divide Fraction (DVF), the contents of the combined Accumulator and Quotient Registers are treated as a 72-bit fixed point binary fraction value but are not addressable as an operand.

The proper operand and its position within a 36-bit machine word is determined by the Processor during preparation of the main store address for the operand. If the data width of the operand selected is smaller than the register involved, the operand is high-order or low-order zero filled as necessary.

Table 3-2 Fixed Point Binary Fraction Values

<table>
<thead>
<tr>
<th>Operand</th>
<th>6-bit Byte</th>
<th>9-bit Byte</th>
<th>Lower 18-bit Half Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic range</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Maxima</td>
<td>---(1)</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Neg.</td>
<td>((2^{15})-1) x 2^{35}</td>
<td>((2^{8})-1) x 2^{35}</td>
<td>((2^{17})-1) x 2^{35}</td>
</tr>
<tr>
<td>Pos.</td>
<td>2^35</td>
<td>2^35</td>
<td>2^35</td>
</tr>
<tr>
<td>Resolution</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operand</th>
<th>Upper 18-bit Half Word</th>
<th>36-bit Single Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic range</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Maxima</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neg.</td>
<td>-1.0</td>
<td>-1.0</td>
</tr>
<tr>
<td>Pos.</td>
<td>1.0 - 2^{17}</td>
<td>1.0 - 2^{35}</td>
</tr>
<tr>
<td>Resolution</td>
<td>2^{17}</td>
<td>2^{35}</td>
</tr>
</tbody>
</table>

(1) No Negative maximum is shown for 6-bit Byte, 9-bit Byte, and Lower 18-bit Half Word operands since the high-order zero fill during operand alignment forces the sign bit to zero.

All operands are legal for the Divide Fraction (DVF) instruction but only the 18-bit Half Word and 36-bit Single Precision operands are legal for the Multiply Fraction (MPF) instruction.

Fixed point binary fraction operands are illegal for all other instructions.
FLOATING POINT BINARY DATA

A floating point binary number is expressed as

\[ Z = M \times 2^{E} \]

where:

- \( M \) is an arithmetic fixed point binary fraction; the mantissa
- \( E \) is an arithmetic fixed point integer; the exponent

A floating point binary number is defined by imposing the bit position value structure below on an information unit of \( n \) bits.

**Exponent value:**

\[ \left[ \ldots (a(0)a(0)) \times 2^6 + (a(2)a(0)) \times 2^5 + \ldots + (a(7)a(0)) \right] \]

**Mantissa value:**

\[ \left[ \ldots (a(8)) \times 2^8 - 1 + (a(10)a(8)) \times 2^7 - 2 + \ldots + (a(n-1)a(8)) \times 2^{(7-n)} \right] \]

where the symbols and notation are the same as for fixed point binary data above.

The following floating point data items are defined.

<table>
<thead>
<tr>
<th>( D )</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>Half Word Operand</td>
</tr>
<tr>
<td>36</td>
<td>Single Precision Operand</td>
</tr>
<tr>
<td>72</td>
<td>Double Precision Operand</td>
</tr>
</tbody>
</table>

For clarity, the formats of these operands are shown in Figures 3-7 through 3-10 below.

![Figure 3-7](image-url)

**Figure 3-7** Upper 18-bit Half Word Floating Point Binary Operand Format
The proper operand is selected by the Processor during preparation of the main store address for the operand. If the data width of the operand is smaller than the register involved, the operand is high-order or low-order zero filled as necessary.

Overlength Registers

The combined AQ register is used to hold the mantissa of all floating point binary numbers. The AQ register is said to be overlength with respect to the operands since it has more bits than are provided by the operands. Operands are low-order zero filled when loaded and low-order truncated (or rounded, depending on the instruction) when stored. Thus, the result of all floating point instructions has more bits of precision in the AQ than may be stored.

Users are cautioned that algorithms involving floating point operands may suffer from propagation of truncation errors unless the algorithms are designed to hold mantissas in the AQ register as long as possible. It is possible to retain full AQ precision of results if they are saved with the Store AQ (STAQ) and Store Exponent (STE) instructions but such saved data are not usable as a floating point operand.

Normalized Numbers
A floating point number is said to be normalized if the relation
\[ \frac{0.5}{1} < |m| < 1.0 \]
is satisfied. The presence of unnormalized numbers in any finite mantissa
arithmetic can only degrade the accuracy of results. For example, in an
arithmetic allowing only two digits in the mantissa, the number 0.005 × 10**2
has the value zero instead the value one half.

Normalization is a process of shifting the mantissa and adjusting the
exponent until the relation above is satisfied. Normalization may be used to
recover some or all of the extra bits of the overlength AQ register after a
floating point operation.

There are cases where the limits of the registers force the use of
unnormalized numbers. For example, in an arithmetic allowing three digits of
mantissa and one digit of exponent, the calculation 0.3 × 10**-10 - 0.1 × 10**-11
(the normalized case) may not be made, but 0.03 × 10**-9 - 0.001 × 10**-9 = 0.029 × 10**-9 (the unnormalized case) is a valid result.

Some examples of normalized and unnormalized numbers are:

- Unnormalized positive binary: 0.00011010 x (2**7)
- Same number normalized: 0.11010000 x (2**4)
- Unnormalized negative binary: 1.11010111 x (2**-4)
- Same number normalized: 1.01011100 x (2**-6)

The minimum normalized non-zero floating point binary number is 2**-126 in
all cases.

<table>
<thead>
<tr>
<th>Table 3-3 Floating Point Binary Operand Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operand</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Unnormalized range</strong></td>
</tr>
<tr>
<td>Minimum:</td>
</tr>
<tr>
<td>Maximum:</td>
</tr>
<tr>
<td>Neg.</td>
</tr>
<tr>
<td>Pos.</td>
</tr>
<tr>
<td>Resolution</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Operand</strong></th>
<th><strong>Double Precision</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unnormalized range</strong></td>
<td></td>
</tr>
<tr>
<td>Minimum:</td>
<td>0</td>
</tr>
<tr>
<td>Maximum:</td>
<td>-1.0 x 2**127</td>
</tr>
<tr>
<td>Neg.</td>
<td>(1 - 2**-63) x 2**127</td>
</tr>
<tr>
<td>Pos.</td>
<td></td>
</tr>
<tr>
<td>Resolution</td>
<td>1163</td>
</tr>
</tbody>
</table>
(1) There is no unique representation for the value zero in floating point binary numbers; any number with mantissa zero has the value zero. However, the Processor treats a zero mantissa as a special case in order to preserve precision in later calculations with a zero intermediate result. Whenever the Processor detects a zero mantissa as the result of a floating binary operation, the AQ register is cleared to zeros and the E register is set to -128. This representation is known as a floating normalized zero. The unnormalized zero (any zero mantissa) will be handled correctly if encountered in an operand but precision may be lost. For example, A x 10**-14 + 0 x 10**-85 will not produce desired results since all the precision of A will be lost when it is aligned to match the 10**-85 exponent of the 0.

(2) No Negative maximum is shown for Lower 18-bit Half Word operands since the high-order zero fill during operand alignment forces the sign bit to zero.

(3) A value cannot be given for Resolution in these cases since such a value depends on the value of the exponent, E. The notation used (1/m) indicates resolution to 1 bit in a field of m. Thus, the following general statement on resolution may be made:

The resolution of a floating point binary operand with mantissa length m and exponent value E is 2**(E-m).

DECIMAL DATA

Decimal numbers are expressed in one of the following forms:

Fixed point, no sign

Fixed point, leading sign

Fixed point, trailing sign

Floating point

The form is specified by control information in the Operand Descriptor for the operand as used by the Extended Instruction Set (EIS). (See Section II, Machine Instructions.)

A decimal number is defined by imposing any of the character position value structures below on a 4-bit Character or 9-bit Character Information unit of length n characters.

Fixed point, no sign:

\[ c(0) \times 10^{*(n-1)} + c(1) \times 10^{*(n-2)} + \ldots + c(n-1) \]

Fixed point, leading sign:

\[ (\text{sign}=c(0)) \times c(1) \times 10^{*(n-2)} + c(2) \times 10^{*(n-3)} + \ldots + c(n-1) \]
Fixed point, trailing sign:
\[ c(0) \times 10^{**(n-2)} + c(1) \times 10^{**(n-3)} + \cdots + c(n-1) \text{ [sign=c(0)]} \]
\[ \downarrow \]

Floating point:
\[ [\text{sign}=c(0)] \ c(1) \times 10^{**(n-3)} + c(2) \times 10^{**(n-4)} + \cdots + c(n-2) \text{ [exponent=8 bits]} \]
\[ \downarrow \]

where:
- \( c(i) \) is the decimal value of the character in the \( i \)th character position.
- "\( \downarrow \)" indicates the position of the decimal point.
- \( [\text{sign}=c(i)] \) indicates that \( c(i) \) is interpreted as a sign character.
- \( [\text{exponent}=8 \text{ bits}] \) indicates that the exponent value is taken from the last 8 bits of the character string. If the data is in 9-bit Characters, the exponent is bits 1-8 of \( c(n) \). If the data is in 4-bit Characters, the exponent is the concatenated value of \( c(n-1) \) and \( c(n) \).

The decimal number as described above is the only decimal data item defined. It may begin on any legal character boundary (without regard to word boundaries) and has a maximum extent of 63 characters.

The Processor handles decimal data as 4-bit bytes internally. Thus, 9-bit characters are high-order truncated as they are transferred from main store and high-order filled as they are transferred to main store. The fill pattern is "00100"\( b \) for digit characters and "00100"\( b \) for sign characters. The floating point exponent is a special case and is treated as a two's complement binary integer.

The Processor performs validity checking on decimal data. Only the byte values (0,11) octal are legal in digit positions and only the byte values (12,17) octal are legal in sign positions. Detection of an illegal byte value causes an Illegal Procedure Fault. The interpretation of decimal sign characters is shown in Table 3-4 below.

Table 3-4 Decimal Sign Character Interpretation

<table>
<thead>
<tr>
<th>9-bit</th>
<th>4-bit</th>
<th>Character Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>12</td>
<td>+</td>
</tr>
<tr>
<td>53(1)</td>
<td>13(2)</td>
<td>+</td>
</tr>
<tr>
<td>54</td>
<td>14(1)</td>
<td>+</td>
</tr>
<tr>
<td>55(1)</td>
<td>15(1)</td>
<td>-</td>
</tr>
<tr>
<td>56</td>
<td>16</td>
<td>+</td>
</tr>
<tr>
<td>57</td>
<td>17</td>
<td>+</td>
</tr>
</tbody>
</table>
(1) This character is used as the default sign character for storage of results. The presence of other characters will yield correct results according to the interpretation.

(2) An optional control bit in the EIS Decimal Arithmetic Instructions (See Section II, Machine Instructions) allows the selection of (13) octal for the plus sign character for storage of results in 4-bit data mode.

Decimal Data Values

The Operand Descriptors for decimal data operands have a 6-bit two's complement binary field for invocation of a Scaling Factor (SF). This Scaling Factor has the same effect as the value of E in floating point decimal operands; a negative value moves the assumed decimal point to the left; a positive value, to the right. The use of the Scaling Factor extended the range and resolution of decimal data operands. The range of the Scaling Factor is (-32,31).

Table 3-5 Decimal Data Values

<table>
<thead>
<tr>
<th>Operand</th>
<th>Fixed Point</th>
<th>Fixed Point</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Sign</td>
<td>Leading or Trailing Sign</td>
</tr>
<tr>
<td>Arithmetic range</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum:</td>
<td>0(1)</td>
<td>0</td>
</tr>
<tr>
<td>Maximum:</td>
<td>(10<strong>64 - 1) x 10</strong>31</td>
<td>(10<strong>63 - 1) x 10</strong>31</td>
</tr>
<tr>
<td>Resolution:</td>
<td>11SF(2)</td>
<td>11SF</td>
</tr>
<tr>
<td>Operand</td>
<td>9-bit Floating Point</td>
<td>4-bit Floating Point</td>
</tr>
<tr>
<td>Arithmetic range</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Maximum:</td>
<td>(10<strong>62 - 1) x 10</strong>158</td>
<td>(10<strong>61 - 1) x 10</strong>158</td>
</tr>
<tr>
<td>Resolution:</td>
<td>11SF+E</td>
<td>11SF+E</td>
</tr>
</tbody>
</table>

(1) See Decimal Zero below.

(2) A value cannot be given for Resolution in these cases since such a value depends on the value of the Scaling Factor, SF, and/or the exponent, E. The notation used (11SF+E) indicates resolution to 1 part in 10**(SF+E). Thus, the following general statement on resolution may be made:

The resolution of a fixed point decimal operand with Scaling Factor SF is 10**SF and the resolution of a floating point decimal operand with Scaling Factor SF and exponent E is 10**(SF+E).
Decimal Zero

As in floating point binary arithmetic, there is no unique representation of the value zero except in the case of fixed point, no sign data. Therefore, the processor detects a zero result and forces a value of +0. for fixed point, leading or trailing sign and +0. x 10**127 for floating point data. Again, as in floating binary arithmetic, other representations of the value zero will be handled correctly except for possible loss of precision during operand alignment.

Alphanumeric Data

Alphanumeric data is represented in two modes: character string and bit string. The mode is determined by the processor according to the function being performed.

CHARACTER STRING DATA

Character string data is defined by imposing the character position structure below on a 4-bit, 6-bit, or 9-bit character information unit of length n characters.

\[ c(0) \| c(1) \| \ldots \| c(n-1) \]

where:

- \( c(i) \) is the character in the \( i \)th character position.
- \( \| \) indicates the concatenation operation.

The character string described above is the only character string data item defined. It may begin on any legal character boundary (without regard to word boundaries) and has a maximum extent as shown in Table 3-6 below.

Table 3-6 Character String Data Length Limits

<table>
<thead>
<tr>
<th>Character Size</th>
<th>Length Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>9-bit</td>
<td>1048576</td>
</tr>
<tr>
<td>6-bit</td>
<td>1572864</td>
</tr>
<tr>
<td>4-bit</td>
<td>2097152</td>
</tr>
</tbody>
</table>

No interpretation of the characters is made except as specified for the instruction being executed. (See Section II, Machine Instructions.)
BIT STRING DATA

Bit string data is defined by imposing the bit position structure below on a machine word information unit of length \(n\) bits.

\[b(0) || b(1) || ... || b(n-1)\]

where:

- \(b(l)\) is the value of the bit in the \(l\)th position.
- \(\|\) indicates the concatenation operation.

The bit string described above is the only bit string data item defined. It may begin at any bit position (without regard to character or word boundaries) and has a maximum extent of \(9437184000\) bits.
A processor register is a hardware assembly that holds information for use in some specified way. An accessible register is a register whose contents are available to the user for his purposes. Some accessible registers are explicitly referenced by particular instructions, some are implicitly referenced during the course of execution of instructions, and some are used in both ways. The accessible registers are listed in the table below. See Section II, Machine Instructions, for a discussion of each instruction to determine the way in which the registers are used.

Table 4-1 Processor Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>Mnemonic</th>
<th>Bit Length</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator Register</td>
<td>A</td>
<td>36</td>
<td>1</td>
</tr>
<tr>
<td>Quotient Register</td>
<td>Q</td>
<td>36</td>
<td>1</td>
</tr>
<tr>
<td>Accumulator-Quotient Register(1)</td>
<td>AQ</td>
<td>72</td>
<td>1</td>
</tr>
<tr>
<td>Exponent Register</td>
<td>E</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>Exponent-Accumulator-Quotient Register(1)</td>
<td>EAQ</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td>Index Registers</td>
<td>Xn</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>Indicator Register</td>
<td>IR</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>Base Address Register</td>
<td>BAR</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>Timer Register</td>
<td>TR</td>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td>Ring Alarm Register</td>
<td>RALR</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Pointer Registers</td>
<td>PRn</td>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td>Procedure Pointer Register</td>
<td>PPR</td>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td>Temporary Pointer Register</td>
<td>TPR</td>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td>Descriptor Segment Base Register</td>
<td>DSBR, (OBR)</td>
<td>51</td>
<td>1</td>
</tr>
<tr>
<td>Segment Descriptor Word Associative Memory</td>
<td>SDWAM</td>
<td>51</td>
<td>16</td>
</tr>
<tr>
<td>Page Table Word Associative Memory</td>
<td>PTWAM</td>
<td>51</td>
<td>16</td>
</tr>
<tr>
<td>Fault Register</td>
<td></td>
<td>35</td>
<td>1</td>
</tr>
<tr>
<td>Mode Register</td>
<td></td>
<td>33</td>
<td>1</td>
</tr>
<tr>
<td>Cache Mode Register</td>
<td></td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>Control Unit (CU) History Register</td>
<td></td>
<td>72</td>
<td>16</td>
</tr>
<tr>
<td>Operations Unit (OU) History Register</td>
<td></td>
<td>72</td>
<td>16</td>
</tr>
<tr>
<td>Decimal Unit (DU) History Register</td>
<td></td>
<td>72</td>
<td>16</td>
</tr>
<tr>
<td>Appending Unit (AU) History Register</td>
<td></td>
<td>72</td>
<td>16</td>
</tr>
<tr>
<td>Configuration Switch Data</td>
<td></td>
<td>36</td>
<td>5</td>
</tr>
<tr>
<td>Control Unit Data</td>
<td></td>
<td>576</td>
<td>1</td>
</tr>
<tr>
<td>Decimal unit data</td>
<td></td>
<td>288</td>
<td>1</td>
</tr>
</tbody>
</table>

(1) These registers are not separate physical assemblies but are logical combinations of their constituent registers.

In the descriptions that follow, the diagrams given for register formats do not imply that a physical assembly possessing the pictured bit pattern exists.
The diagram is a graphic representation of the form of the register data as it appears in main store when the register contents are stored or how data bits must be assembled for loading into the register.

**ACCUMULATOR REGISTER (A)**

*Format* - 36 bits

```
<table>
<thead>
<tr>
<th></th>
<th>11</th>
<th></th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7 8</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>A-Upper</td>
<td>A-Lower</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>
```

**Figure 4-1 Accumulator Register (A) Format**

**Description**

A 36 bit physical register located in the Operations Unit.

**Function**

In fixed point binary operations, holds operands and results.

In floating point binary operations, holds the most significant part of the mantissa.

In shifting operations, holds original data and shifted results.

In address preparation, may hold two logically independent word offsets, A-Upper and A-Lower, or an extended range bit or character offset.

**QUOTIENT REGISTER (Q)**

*Format* - 36 bits

```
<table>
<thead>
<tr>
<th></th>
<th>11</th>
<th></th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7 8</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Q-Upper</td>
<td>Q-Lower</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td>18</td>
</tr>
</tbody>
</table>
```

**Figure 4-2 Quotient Register (Q) Format**
Description

A 36 bit physical register located in the Operations Unit.

Function

In fixed point binary operations, holds operands and results.
In floating point binary operations, holds the least significant part of the mantissa.
In shifting operations, holds original data and shifted results.
In access preparation, may hold two logically independent word offsets, Q-Upper and Q-Lower, or an extended range bit or character offset.

ACCUMULATOR-QUOTIENT REGISTER (AQ)

Format - 72 bits

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

Figure 4-3 Accumulator-Quotient Register (AQ) Format

Description

A logical combination of the Accumulator (A) and Quotient (Q) registers.

Function

In fixed point binary operations, holds double precision operands and results.
In floating point binary operations, holds the mantissa.
In shifting operations, holds original data and shifted results.
EXponent REGISTER (E)

Format - 8 bits

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

Figure 4-4 Exponent Register (E) Format

Description

An 8 bit physical register located in the Operations Unit. Bits pictured as "x" are "don't care" bits, that is, are irrelevant to the register or its use.

Function

In floating point binary operations, holds the exponent.

EXponent-ACCumulator-Quotient REGISTER (EAQ)

Format - 80 bits

<table>
<thead>
<tr>
<th>0 0 0</th>
<th>7 6</th>
<th>2 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>exponent</td>
<td>1</td>
<td>mantissa</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>

Figure 4-5 Exponent-Accumulator-Quotient Register (EAQ) Format

Description

A logical combination of the Exponent (E), Accumulator (A), and Quotient (Q) registers. Although the register has a total of 80 bits, only 72 are involved in transfers to and from main store. The low order 8 bits are truncated on store and zero filled on load.

Function

In floating point binary operations, holds operands and results.
INDEX REGISTERS (Xn)

Format: - 18 bits each

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
</tr>
</tbody>
</table>

Figure 4-6 Index Register (Xn) Format

Description:

Eight 18 bit physical registers in the Operations Unit numbered 0 through 7. Index Register data may occupy the position of either an Upper or Lower 18-bit Half Word operand in a main store machine word.

Function:

In fixed point binary operations, hold half word operands and results.

In address preparation, hold word offsets or extended range bit or character offsets.

INDICATOR REGISTER (IR)

Format: - 14 bits

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

<table>
<thead>
<tr>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
</tr>
</thead>
<tbody>
<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>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</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>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 4-7 Indicator Register (IR) Format

Description:

A logical assemblage of 14 indicator flags from various units of the Processor. The data occupies the position of a Lower 18-bit Half Word operand. Bits pictured as "x" are "don't care" bits and are irrelevant to the register or its use. Bits pictured as "0" are reserved and must have value 0. When interpreted as data, a bit value of 1 corresponds to the ON state of the indicator, a bit value of 0 corresponds to the OFF state.
The functions of the individual indicator bits are given below. An "x" in the column headed "L" indicates that the state of the indicator is not affected by instructions that load the IR.

<table>
<thead>
<tr>
<th>Key</th>
<th>Indicator Name</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Zero</td>
<td>This indicator is set ON whenever the output of the main binary adder consists entirely of zero bits for binary or shifting operations or the output of the decimal adder consists of zero digits for decimal operations; otherwise, it is set OFF.</td>
</tr>
<tr>
<td>b</td>
<td>Negative</td>
<td>This indicator is set ON whenever the output of bit 0 of the main binary adder has value 1 for binary or shifting operations or the sign character of the result of a decimal operation is the negative sign character; otherwise, it is set OFF.</td>
</tr>
<tr>
<td>c</td>
<td>Carry</td>
<td>This indicator is set ON for any of the following conditions; otherwise, it is set OFF.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1) If a bit propagates leftward out of bit 0 of the main binary adder for any binary or shifting operation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2) If (</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3) If (</td>
</tr>
<tr>
<td>d</td>
<td>Overflow</td>
<td>This indicator is set ON if the arithmetic range of a register is exceeded in a fixed point binary operation or if the target string of a decimal numeric operation is too small to hold the integer part of the result. It remains ON until reset by the Transfer on Overflow (TOV) instruction or is reset by some other instruction that loads the IR. The event that sets this indicator ON may also cause an Overflow Fault. (See Overflow Mask indicator below.)</td>
</tr>
<tr>
<td>e</td>
<td>Exponent Overflow</td>
<td>This indicator is set ON if the exponent of the result of a floating point binary or decimal numeric operation is greater than +127. It remains ON until reset by the Transfer on Exponent Overflow (TED) instruction or is reset by some other instruction that loads the IR. The event that sets this indicator ON may also cause an Overflow Fault. (See Overflow Mask indicator below.)</td>
</tr>
<tr>
<td>f</td>
<td>Exponent Underflow</td>
<td>This indicator is set ON if the exponent of the result of a floating point binary or decimal numeric operation is less than -128.</td>
</tr>
</tbody>
</table>
g  Overflow Mask

This indicator is set ON or OFF only by the instructions that load the IR. When set ON, it inhibits the generation of the fault for those events that normally cause an Overflow Fault. If the Overflow Mask indicator is set OFF after occurrence of an Overflow event, an Overflow Fault will not occur even though the indicator for that event is still set ON. The state of the Overflow Mask indicator does not affect the setting, testing, or storing of any other indicator.

h  Tally Runout

This indicator is set OFF at initialization of any tallying operation, that is, any repeat instruction or any Indirect Then Tally Address Modification. It is then set ON for any of the following conditions:

1. If a repeat instruction terminates because of tally exhaust.

2. If a Repeat Link (RPL) instruction terminates because of a zero link address.

3. If a tally exhaust is detected for an Indirect Then Tally modifier. The instruction will be executed whether or not tally exhaust occurs.

i  Parity Error

This indicator is set ON whenever the main store signals Illegal Address with a parity error code or the Processor detects an internal parity error condition. The indicator is set OFF only by instructions that load the IR.

j  Parity Mask

This indicator is set ON or OFF only by the instructions that load the IR. When it is set ON, it inhibits the generation of the Parity Fault for all events that set the Parity Error indicator. If the Parity Mask indicator is set OFF after the occurrence of a Parity Error event, a Parity Fault will not occur even though the Parity Error indicator may still be set ON. The state of the Parity Mask indicator does not affect the loading, testing, or storing of any other indicator, generated from previously set parity error indicators. The status of the parity mask indicator does not affect the setting, testing, or storing of the parity error indicator.
This indicator is set OFF only by execution of the Transfer and Set Slave (TSS) instruction that places the Processor in BAR Mode. It is set ON (taking the Processor out of BAR Mode) by the execution of any transfer class instruction other than TSS during a Fault or Interrupt Trap. However, if the Fault or Interrupt Trap occurs while in BAR Mode, and the transfer class instruction is Return (RET), Return Control Double (RTCD), or Restore Control Unit (RCU) and bit 28 of the saved IR data is 0, the Processor will remain in BAR Mode.

This indicator is set ON whenever the target string of a decimal numeric operation is too small to hold all the fraction digits of the result or the target string of an alphanumeric operation is too small to hold all the bits or characters to be stored. Also see the Overflow indicator condition for decimal numeric operations. The event that sets this indicator ON may also cause an Overflow Fault. (See Overflow Mask indicator above.)

This indicator is set ON whenever the current instruction is interrupted by an external event. The indicator has meaning only when determining the proper restart sequence for the interrupted instruction. The indicator is set OFF at normal termination of every instruction. The events that set this indicator are:

1. An Access Violation Fault during Address Preparation for any operand.
2. Detection of the arrival of a Program Interrupt signal during execution of those EIS instructions that allow very long operand strings.

This indicator is set ON only by execution an absolute (non-appended) transfer class instruction during a Fault or Interrupt Trap and is set OFF by any execution of an appended transfer class instruction. However, if the Processor is not in Absolute Mode when the Fault or Interrupt occurs and the transfer class instruction is Return (RET), Return Control Double (RTCD), or Restore Control Unit (RCU) and the appropriate mode bit is properly set in the IR data, the Processor will remain in its current mode.
BASE ADDRESS REGISTER (BAR)

**Format**: 18 bits

\[
\begin{array}{cccccc}
0 & 0 & 0 & 1 & 1 & 3 \\
0 & 0 & 8 & 9 & 7 & 8 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & BASE & 1 & BOUND & 1 & 1 \\
1 & 9 & 9 & 9 & 18 & 18 \\
\end{array}
\]

**Figure 4-8 Base Address Register (BAR) Format**

**Description**

An 18 bit physical register in the Control Unit. The data is pictured in its normal operand position as stored by the Store Base Address Register (SBAR) instruction. Bits pictured as "x" are "don't care" bits and are irrelevant to the register or its use.

**Function**

The Base Address Register provides automatic hardware address relocation and address range limitation when the Processor is in BAR Mode.

- **BAR.BASE** Contains the high-order nine bits of an 18-bit address relocation constant. The low order bits are generated as zeros.
- **BAR.BOUND** Contains the un relocated address limit stated as a number of 512 word blocks. An attempt to access main store beyond this limit causes a Store Fault, Out of Bounds.

TIMER REGISTER (TR)

**Format**: 27 bits

\[
\begin{array}{cccccc}
0 & 0 & 2 & 2 & 3 & 3 \\
0 & 0 & 6 & 7 & 6 & 7 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & Timer Value & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

**Figure 4-9 Timer Register (TR) Format**
Description:

A 27 bit setable, free running clock in the Control Unit. The value decrements at a rate of 512 kHz. Its range is 1.953125 microseconds to approximately 4.37 minutes. Bits pictured as "x" are "don't care" bits and are irrelevant to the register and its use.

Function:

The TR may be loaded with any convenient value with the privileged Load Timer (LDT) instruction. When the value next passes through zero, a Timer Runout Fault will be signalled. If the Processor is in Normal or BAR Mode with Program Interrupts not inhibited, the Fault will occur immediately. If the Processor is in Absolute or Privileged Mode or has Program Interrupts inhibited, the Fault will be delayed until the Processor returns to uninhibited Normal or BAR Mode.

RING ALARM REGISTER (RALR)

Format: 3 bits

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 4-10 Ring Alarm Register (RALR) Format

Description:

A 3 bit physical register in the Appending Unit. The bits pictured as "x" are "don't care" bits and are irrelevant to the register or its use. The bits may have meaning with regard to other data structures.

Function:

If the Effective Ring Number (See TPR,TRR below) is greater than the contents of RALR an Access Violation, Ring Alarm, Fault will occur. The Multics supervisor uses this mechanism to assure the proper handling of User Ring events (such as QUITs) that occur while executing in the supervisor.
POINTER REGISTERS (PRn)

**Format** - 42 bits each

**Even Word of ITS Pointer Pair**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>0001</td>
</tr>
<tr>
<td>4-7</td>
<td>0123</td>
</tr>
<tr>
<td>8-11</td>
<td>1111</td>
</tr>
<tr>
<td>12-15</td>
<td>1000</td>
</tr>
<tr>
<td>16-19</td>
<td>0000</td>
</tr>
<tr>
<td>20-23</td>
<td>0000</td>
</tr>
<tr>
<td>24-27</td>
<td>x x x x</td>
</tr>
<tr>
<td>28-31</td>
<td>x x x x</td>
</tr>
</tbody>
</table>

**Odd Word of ITS Pointer Pair**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>1112</td>
</tr>
<tr>
<td>4-7</td>
<td>2222</td>
</tr>
<tr>
<td>8-11</td>
<td>1111</td>
</tr>
<tr>
<td>12-15</td>
<td>1000</td>
</tr>
<tr>
<td>16-19</td>
<td>0000</td>
</tr>
<tr>
<td>20-23</td>
<td>0000</td>
</tr>
<tr>
<td>24-27</td>
<td>x x x x</td>
</tr>
<tr>
<td>28-31</td>
<td>x x x x</td>
</tr>
</tbody>
</table>

Figure 4-11 Pointer Register (PRn) Format

**Description**

Eight logical combinations of physical registers from the Appending Unit and Control Unit numbered 0 through 7. PRn.RNR and PRn.SNR are located in the Appending Unit and PRn.WORDNO, PRn.CHAR, and PRn.BITNO are located in the Decimal Unit. Bits pictured as "x" are "don't care" bits and are irrelevant to the register and its use. Bits pictured as "0" are reserved and must have value 0. The format above shows the data from the register when stored in ITS Pointer Pair format. The "x" bits do have meaning in the ITS Pointer Pair format. Certain of the register data may also be stored in Packed Pointer format.

The reader's attention is directed to the double definition of bits 21-26 of the Odd Word and to the Note under the discussion of PRn.CHAR.

**Function**

The Pointer Registers hold information relative to the location in main store of "external" data items, that is, data items external to the segment containing the procedure being executed. The functions of the individual constituent registers are:

**Key Register**

<table>
<thead>
<tr>
<th>Register</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRn.RNR</td>
<td>The Ring Number Register contains the maximum privilege level (smallest ring number) that may be assigned to a process attempting to access the data item described by the Pointer Register. For example, if PRn.RNR is</td>
</tr>
</tbody>
</table>

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975
4-11
AL39
greater (less privileged) than the current validation level of the process (as contained in PPR.PRR described below) then the Effective Ring Number for the access is PRn.RNR. The value of PRn.RNR is determined from directory entry information for the segment when the pointer data is constructed.

The Segment Number Register contains the segment number of the segment containing the data item described by the Pointer Register. The segment number is determined when the Segment Descriptor Word (SDW) is constructed from directory entry information for the segment.

The Word Number register contains the offset in machine words from the base or origin of the segment to the data item. The value is determined when the pointer data is constructed from the data item description in the procedure.

The Character register contains the number of the 9-bit character within the machine word at PRn.WORDNO containing the data item. The value is determined when the pointer data is constructed from the data item description in the procedure. Word boundary aligned data items will always have the value 0. Unaligned data items may have any value.

NOTE: The reader's attention is directed to the double definition of bits 18-26 of the Odd Word in the format above. Because the Multics Processor was implemented as an enhancement to an existing design, certain apparent anomalies appear. One of these is the difference in the handling of unaligned data items by the Appending Unit and Decimal Unit. The preexisting Decimal Unit handles all unaligned data items with a 9-bit character number plus bit offset with conversion from the description given in the EIS Operand Descriptor done automatically by the hardware. The Appending Unit maintains compatibility with the earlier generation Multics Processor by handling all unaligned data items with a bit offset from the prior word boundary; again with any necessary conversion done automatically by the hardware. Thus, a Pointer Register may be loaded from an ITS Pointer Pair having a pure bit offset and modified by one of the EIS Address Register instructions (A4B, S98D, etc.) using character displacement counts. When the results of such a modification are stored as an ITS Pointer Pair with SPRIn (or as a Packed Pointer with SPRP), the BITNO field as indicated in the upper line of the format (bits 21-26) will contain a pure bit count. When the results are stored as an Address Register with SARIn the CHAR and BITNO fields as indicated in the lower line of the format (bits 18-23) will contain the character number plus bit offset.

WARNING: The Decimal Unit has built-in hardware checks for illegal bit offset values but the Appending Unit does not except for a single case for packed pointers. See NOTES for Load Packed Pointers (LPBP) in Section II, Machine Instructions.
**Key Register**

Pn.BITNO  

The Bit Number register contains the number of the bit within Pn.CHAR of the word at Pn.WORD40 containing the data item. The value is determined when the pointer data is constructed from the data item description in the procedure. Word and character boundary aligned data items will always have the value 0. Unaligned data items may have any value in the range (0,10) octal. See NOTE under Pn.CHAR above.

**PROCEDURE POINTER REGISTER (PPR)**

**Format** - 37 bits

Word 0 of Control Unit Data

<table>
<thead>
<tr>
<th>0 0 0</th>
<th>1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>7 8 9</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>PRR</td>
<td>PSR</td>
</tr>
<tr>
<td>3</td>
<td>15 1</td>
</tr>
</tbody>
</table>

Figure 4-12 Procedure Pointer Register (PPR) Format

**Description**

A logical combination of physical registers from the Appending Unit and the Control Unit. PPR, PRR, PPR, PSR, and PPR, P are located in the Appending Unit and PPR, IC is located in the Control Unit. The data is pictured as it appears in main store in Words 0 and 4 of Control Unit Data. Bits pictured as "x" are "don't care" bits and are irrelevant to the register or its use. The bits do have meaning with regard to Control Unit Data. (See Control Unit Data below.)

**Function**

The Procedure Pointer Register holds information relative to the location in main store of the procedure segment in execution and the location of the current instruction within that segment. The functions of the individual constituent registers are:
<table>
<thead>
<tr>
<th>Register</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPR.PRR</td>
<td>The Procedure Ring Register contains the number of the ring (validation level) in which the process is executing. It is set to the Effective Ring Number of the procedure segment when control is transferred to the procedure.</td>
</tr>
<tr>
<td>PPR.PSR</td>
<td>The Procedure Segment Register contains the segment number of the procedure being executed. Its value changes every time control is transferred to a new procedure.</td>
</tr>
<tr>
<td>PPR.P</td>
<td>The Privileged bit register is a flag controlling execution of privileged instructions. Its value is &quot;1&quot;b (permitting privileged instructions) if PPR.PRR is 0 and the privileged bit in the Segment Descriptor word (SDW.P) for the procedure is &quot;1&quot;b. Its value is &quot;0&quot;b if SDW.P is 0 or PPR.PRR is greater than 0. Its value is set every time a new procedure is entered.</td>
</tr>
<tr>
<td>PPR.IC</td>
<td>The Instruction Counter register contains the word offset from the origin of the procedure segment to the current instruction.</td>
</tr>
</tbody>
</table>
TEMPORARY POINTER REGISTER (TPR)

Format - 42 bits

Word 2 of Control Unit Data

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>TRR</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>TSR</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>15</td>
<td>18</td>
<td></td>
</tr>
</tbody>
</table>

Word 3 of Control Unit Data

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>TBR</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

Word 5 of Control Unit Data

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>CA</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18</td>
</tr>
</tbody>
</table>

Figure 4-13 Temporary Pointer Register (TPR) Format

Description

A logical combination of physical registers from the Appending Unit and the Control Unit. TPR, TRR, TPR.TSR, and TPR.TBR are located in the Appending Unit and TPR.CA is located in the Control Unit. The data is pictured as it appears in main store in Words 2, 3, and 5 of Control Unit Data. Bits pictured as "x" are "don't care" bits and are irrelevant to the register or its use. The bits do have meaning with regard to Control Unit Data. (See Control Unit Data below.)

Function

The Temporary Pointer Register holds information relative to the location in main store of indirect words and pointers (during address preparation) and operands (during instruction execution). At the completion of address preparation, the contents of the TPR is presented to the Appending Unit Associative Memory Assemblies for translation into the final 24-bit main store address. The functions of the individual constituent registers are:
Register | Function
--- | ---
TPR.TRR | The Temporary Ring Register contains the Effective Ring Number for the data access. If the access is to the procedure segment, TPR.TRR is set to PPR.PRR; if the access invokes a Pointer Register, TPR.TRR is set to the larger of PRn.RNR and PPR.PRR.
TPR.TSR | The Temporary Segment Register contains the segment number of the segment to be accessed.
TPR.TBR | The Temporary Bit Register holds the bit offset for indirect words or pointers (during address preparation) or operands (during instruction execution). Its value is calculated during address preparation from the contents of PRn.CHAR and PRn.BITNO and other information provided by the Address Modification specified for the instruction. See PRn.CHAR and PRn.BITNO above for further detail.
TPR.CA | The Computed Address register contains the word offset of indirect words or pointers (during address preparation) or operands (during instruction execution).

**DESCRIPTOR SEGMENT BASE REGISTER (DSBR, DBR)**

**Format** - 51 bits

Even Word of Y-pair

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2 2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>ADDR</td>
<td>10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Odd Word of Y-pair

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>1 1</td>
<td>2 2</td>
</tr>
<tr>
<td>0 1</td>
<td>4 5</td>
<td>8 9</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>BND</td>
<td>10 0 0 0 0 0 1 0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>STACK</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>4 1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>12</td>
</tr>
</tbody>
</table>

Figure 4-14 Descriptor Segment Base Register (DSBR, DBR) Format

Description

A logical combination of various Appending Unit registers. The data is pictured in the format expected by the Load Descriptor Base Register (LDBR) and Store Descriptor Base Register (SDBR) instructions. Bits pictured as
"0" are reserved and must have the value 0.

**Function**

The Descriptor Segment Base Register contains information concerning the Descriptor Segment for a process. The Descriptor Segment holds the Segment Descriptor Words (SDWs) for all segments accessible by the process. The functions of its individual constituent registers are:

<table>
<thead>
<tr>
<th>Register</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSBR.ADDR</td>
<td>The interpretation of the ADDRess register depends on the value of DSBR.U.</td>
</tr>
<tr>
<td></td>
<td><strong>For DSBR.ADDR contains</strong></td>
</tr>
<tr>
<td></td>
<td>U=0 The 24-bit main store address of the Page Table for the Descriptor Segment.</td>
</tr>
<tr>
<td></td>
<td>U=1 The 24-bit main store address of the Descriptor Segment.</td>
</tr>
<tr>
<td>DSBR.BND</td>
<td>The <strong>BOUND</strong> register contains 14 most significant bits of the highest 16 word block of the Descriptor Segment that can be addressed without causing an Access Violation.</td>
</tr>
<tr>
<td>DSBR.U</td>
<td>The <strong>U</strong> register is a flag specifying whether the descriptor segment is unpaged (U=1) or paged (U=0).</td>
</tr>
<tr>
<td>DSBR.STACK</td>
<td>The <strong>STACK</strong> register contains the upper 12 bits of the 15-bit stack base segment number. It is used only during the execution of the CALL6 instruction. (The Segment Number of the Stack Segment for a running process is given by 6 * DSBR.STACK + PPR.PRR.)</td>
</tr>
</tbody>
</table>
SEGMENT DESCRIPTOR WORD ASSOCIATIVE MEMORY (SDWAM)

Format: 85 bits each

Even Word of Y-pairs as stored by Store Segment Descriptor Registers (SSDR)

<table>
<thead>
<tr>
<th>Bit</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>ADDR</td>
<td>1</td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Odd Word of Y-pairs as stored by Store Segment Descriptor Registers (SSDR)

<table>
<thead>
<tr>
<th>Bit</th>
<th>3</th>
<th>3</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>101</td>
<td>BOUND</td>
<td>1</td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Data as stored by Store Segment Descriptor Pointers (SSDP)

<table>
<thead>
<tr>
<th>Bit</th>
<th>11</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<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>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<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>

Figure 4-15 Segment Descriptor Word Associative Memory (SDWAM) Format

Description:

Sixteen logical combinations of registers and flags from the Appending Unit comprising the Segment Descriptor Word Associative Memory Assembly. The registers are numbered from 0 through 15 but are not directly addressable by number. Bits pictured as "0" are reserved and must have the value 0.

Function:

Hardware segmentation in the Multics Processor is implemented by the Appending Unit (See Section V, Address -- Segmentation and Paging for details). In order to permit addressing by Segment Number and offset as prepared in the Temporary Pointer Register (described above), a table containing the location and status of each accessible segment must be kept. This table is the Descriptor Segment and is unique to the process. The Descriptor Segment for a running process is located by information held in the Descriptor Segment Base Register (DSBR) described above.
Every time an Effective Segment Number (TPR.TSR) is prepared, it is used as an index into the Descriptor Segment to retrieve the Segment Descriptor Word (SDW) for the target segment. To reduce the number of main store references required for segment addressing, the SDWAM provides a content addressable store to hold the sixteen most recently referenced SDWs.

Whenever a reference to the SDW for a segment is required, the Effective Segment Number (TPR.TSR) is matched associatively against all 16 SDWAM.POINTER registers (described below). If the SDWAM match logic circuitry indicates a "hit", all usage counts (SDWAM.USE) greater than the usage count of the "hit" register are decremented by one, the usage count of the "hit" register is set to 15, and the contents of the "hit" register are read out into the address preparation circuitry as necessary. If the SDWAM match logic does not indicate a "hit", the SDW is fetched from main store and loaded into the SDWAM register with usage count 0 (the "oldest"), all usage counts are decremented by one with the newly loaded register rolling over from 0 to 15, and the newly loaded register is read out into the address preparation circuitry as necessary. Faulted SDWs are loaded into the SDWAM.

The functions of the constituent registers and flags of each SDWAM register are:

<table>
<thead>
<tr>
<th>Register</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDWAM.ADDR</td>
<td>The interpretation of the ADDRESS register depends on the value of SDWAM.U.</td>
</tr>
<tr>
<td></td>
<td>For SDWAM.ADDR contains</td>
</tr>
<tr>
<td></td>
<td>U=0  The 24-bit main store address of the Page Table</td>
</tr>
<tr>
<td></td>
<td>for the target segment.</td>
</tr>
<tr>
<td></td>
<td>U=1  The 24-bit main store address of the target segment.</td>
</tr>
<tr>
<td>SDWAM.R1</td>
<td>Upper limit of read/write Ring Bracket. (See Section VIII, Hardware Ring Implementation)</td>
</tr>
<tr>
<td>SDWAM.R2</td>
<td>Upper limit of read/execute Ring Bracket. (See Section VIII, Hardware Ring Implementation)</td>
</tr>
<tr>
<td>SDWAM.R3</td>
<td>Upper limit of call Ring Bracket. (See Section VIII, Hardware Ring Implementation)</td>
</tr>
<tr>
<td>SDWAM.BOUND</td>
<td>The upper limit of segment addresses stated as a number of 16 word blocks. A segment address (TPR.CA) with a block address larger than this value will cause an Access Violation, Out of Segment Bounds, Fault.</td>
</tr>
<tr>
<td>SDWAM.R</td>
<td>Read permission bit. If this bit is set ON, read access requests may be honored.</td>
</tr>
<tr>
<td>SDWAM.E</td>
<td>Execute permission bit. If this bit is set ON, the SDW may be loaded into the Procedure Pointer Register (PPR) and control transferred to the segment for execution.</td>
</tr>
<tr>
<td>SDWAM.W</td>
<td>Write permission bit. If this bit is set ON, write access requests may be honored.</td>
</tr>
<tr>
<td>SDWAM.P</td>
<td>Privileged flag bit. If this bit is set ON, privileged instructions from the segment may be executed if PPR.PPE is 0.</td>
</tr>
<tr>
<td>Register</td>
<td>Function</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SDWAM.U</td>
<td>Unpaged flag bit. If this bit is set ON, the segment is unpaged and SDWAM.ADDR is the 24-bit main store address of the base of the segment. If this bit is set OFF, the segment is paged and SDWAM.ADDR is the 24-bit address the array of Page Table Words (PTWs) for the segment.</td>
</tr>
<tr>
<td>SDWAM.G</td>
<td>Gate control bit. If this bit is set ON, calls into the segment must be to an offset no greater than the value of SDWAM.CL as described below.</td>
</tr>
<tr>
<td>SDWAM.C</td>
<td>Cache control bit. If this bit is set ON, data from the segment may be placed in the cache store.</td>
</tr>
<tr>
<td>SDWAM.CL</td>
<td>Call Limiter value. If the segment is gated (SDWAM.G set ON), transfers of control into the segment must be to segment addresses no greater than this value.</td>
</tr>
<tr>
<td>SDWAM.POINTER</td>
<td>The Effective Segment Number used to fetch this SDW from main store.</td>
</tr>
<tr>
<td>SDWAM.F</td>
<td>Full/empty bit. If this bit is set ON, the SDW in the register is valid. If this bit is set OFF, a &quot;hit&quot; is not possible. All SDWAM.F bits are set OFF by the instructions that clear the SDWAM.</td>
</tr>
<tr>
<td>SDWAM.USE</td>
<td>USE count for the register. The SDWAM.USE field is used to maintain a strict FIFO queue order among the SDWs. When an SDW is matched its USE value is set to 15 (&quot;newest&quot;) and the queue is reordered. SDWs newly fetched from main store replace the SDW with USE value 0 (&quot;oldest&quot;) and the queue is reordered. SDWAM.USE is set the internal (and invisible) SDWAM register number by instructions that clear the SDWAM.</td>
</tr>
</tbody>
</table>
PAGE TABLE WORD ASSOCIATIVE MEMORY (PTWAM)

Format - 51 bits each

Data as stored by Store Page Table Registers (SPTR)

```
<table>
<thead>
<tr>
<th>Number</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>ADDR</td>
</tr>
<tr>
<td>2</td>
<td>2 3</td>
</tr>
<tr>
<td>3</td>
<td>4 0 5</td>
</tr>
</tbody>
</table>
```

Data as stored by Store Page Table Pointers (SPTP)

```
<table>
<thead>
<tr>
<th>Number</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>POINTER</td>
</tr>
<tr>
<td>2</td>
<td>2 2 3 3</td>
</tr>
<tr>
<td>3</td>
<td>4 5 6</td>
</tr>
</tbody>
</table>
```

Figure 4-16 Page Table Word Associative Memory (PTWAM) Format

Description

Sixteen logical combinations of registers and flags from the Appending Unit comprising the Page Table Word Associative Memory Assembly. The registers are numbered from 0 through 15 but are not directly addressable by number. Bits pictured as "0" are reserved and must have the value 0.

Function

Hardware paging in the Multics Processor is implemented by the Appending Unit (See Section V, Address — Segmentation and Paging for details). In order to permit segment addressing by Page Number and page offset as derived from the Effective Address prepared in the Temporary Pointer Register (TPR.CA described above), a table containing the location and status of each page of an accessible segment must be kept. This table is the Page Table Word Array (PTWA) for the segment that is located in the System Segment Table (SST) (a supervisory ring 0 data base) and is sharable by all processes. The PTWA for an accessible paged segment is located by information held in the Segment Descriptor Word (SDW) for the segment.

Every time an Effective Address (TPR.CA) for a paged segment is prepared, it is separated into a Page Number and a page offset. The Page Number is used as an index into the Page Table Word Array to retrieve the Page Table Word (PTW) for the target page. To reduce the number of main store references required for paging, the PTWAM provides a content addressable store to hold the sixteen most recently referenced PTWs.

Whenever a reference to the PTW for a page of a paged segment is required, the Page Number (as derived from TPR.CA) is matched associatively against all 16 PTWAM.PAGENO registers (described below) and, simultaneously,
TPR.TSR is matched against PTWAH.POINTER (described below). If the PTWAH
match logic circuitry indicates a "hit", all usage counts (PTWAH.USE) greater than the usage count of the "hit" register are decremented by one,
the usage count of the "hit" register is set to 15, and the contents of the
"hit" register are read out into the address preparation circuitry as necessary. If the PTWAH match logic does not indicate a "hit", the PTW is
fetched from main store and loaded into the PTWAH register with usage count 
( the "oldest"), all usage counts are decremented by one with the newly
loaded register rolling over from 0 to 15, and the newly loaded register is
read out into the address preparation circuitry as necessary. Faulted PTWs
are not loaded into the PTWAH.

The functions of the constituent registers and flags of each PTWAH register are:

<table>
<thead>
<tr>
<th>Register</th>
<th>Function</th>
</tr>
</thead>
</table>
| PTWAH.ADDR   | The ADDRESS register holds the 18 most significant bits of the 24-bit main store address of the page. The hardware ignores low order bits of the page address according to page size based on the following ...

<table>
<thead>
<tr>
<th>Page Size in words</th>
<th>ADDR bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>none</td>
</tr>
<tr>
<td>128</td>
<td>17</td>
</tr>
<tr>
<td>256</td>
<td>16-17</td>
</tr>
<tr>
<td>512</td>
<td>15-17</td>
</tr>
<tr>
<td>1024</td>
<td>14-17</td>
</tr>
<tr>
<td>2048</td>
<td>13-17</td>
</tr>
<tr>
<td>4096</td>
<td>12-17</td>
</tr>
</tbody>
</table>

| PTWAH.M         | Page Modified flag bit. This bit is set ON whenever the PTW is used for a store type instruction. When the bit changes value from 0 to 1, a special extra cycle is generated to write it back into the PTW in the PTWA. |

| PTWAH.POINTER   | The Effective Segment Number used to fetch this PTW from main store. |

| PTWAH.PAGENO    | The 12 most significant bits of the 18-bit Effective Address (TPR.CA) used to fetch this PTW from main store. Low order bits are forced to zero by the hardware and not used as part of the PTWA index according to page size based on the following ...

<table>
<thead>
<tr>
<th>Page Size in words</th>
<th>PAGENO bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>none</td>
</tr>
<tr>
<td>128</td>
<td>11</td>
</tr>
<tr>
<td>256</td>
<td>10-11</td>
</tr>
<tr>
<td>512</td>
<td>09-11</td>
</tr>
<tr>
<td>1024</td>
<td>08-11</td>
</tr>
<tr>
<td>2048</td>
<td>07-11</td>
</tr>
<tr>
<td>4096</td>
<td>06-11</td>
</tr>
</tbody>
</table>

| PTWAH.F          | Full/empty bit. If this bit is set ON, the PTW in the register is valid. If this bit is set OFF, a "hit" is not possible. All PTWAH.F bits are set OFF by the instructions that clear the PTWA. |

| PTWAH.USE        | Usage count for the register. The PTWAH.USE field is used to maintain a strict FIFO queue order among the
PTWs. When an PTW is matched its USE value is set to 15 ("newest") and the queue is reordered. PTWs newly fetched from main store replace the PTW with USE value 0 ("oldest") and the queue is reordered. PTWAM.USE is set the internal (and invisible) PTWAM register number by instructions that clear the PTWAM.

**FAULT REGISTER**

**Format**: 35 bits

Data as stored by Store Central Processor Register (SCPR), TAG = 01, instruction

```
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 9 0 3 4 7 8 1 2 3 4 5
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Figure 4-17 Fault Register Format

**Description**

A logical combination of flags and registers all located in the Control Unit. The register is stored and cleared by the SCPR (tag 01) command. Note that the data is stored into the word pair at location Y and that the contents of Y+1 are cleared. The Fault Register cannot be loaded.

**Function**

The Fault Register contains the conditions in the Processor for several of the hardware faults. Data is strobed into the Fault Register during a fault sequence. Once a bit or field in the Fault Register has been set, it remains set until the register is cleared. The data will not be overwritten during subsequent fault events.

The reader's attention is directed to another apparent anomaly in the design of the Multics Processor as an enhancement to an existing design. It will be noted that the Fault Register records events from only ports A through D. These four ports are the limit of connectability of the existing design and, since all eight ports are reported in Control Unit Data (described below), no change was made in the Fault Register for the added ports. Data reported for ports A through D are valid in both locations.

The functions of the constituent flags and registers are...
Key Register | Function
---|---
a | ILL OP | An illegal operation code has been detected.
b | ILL MOD | An illegal Address Modifier has been detected.
c | ILL SLV | An illegal BAR Mode procedure has been encountered.
d | ILL PROC | An illegal procedure other than BAR Mode has been encountered.
e | NEM | A nonexistant main store address has been requested.
f | OOB | A BAR Mode boundary violation has occurred.
g | WRT INH | An illegal decimal digit has been detected by the Decimal Unit. (Flag name is obsolete)
h | PROC PARU | A parity error has been detected in the upper 36 bits of data.
i | PROC PARL | A parity error has been detected in the lower 36 bits of data.
j | $CON A | A $CONNECT signal has been received through port A.
k | $CON B | A $CONNECT signal has been received through port B.
l | $CON C | A $CONNECT signal has been received through port C.
m | $CON D | A $CONNECT signal has been received through port D.
n | DA ERR1 | CPU/SCU interface sequence error 1 has been detected. ($DATA-AVAIL received with no prior $INTERRUPT sent.)
o | DA ERR2 | CPU/SCU interface sequence error 2 has been detected. (Multiple $DATA-AVAIL received or $DATA-AVAIL received out of order.)
I AA | Coded Illegal Action, Port A. (See Table 4-2 below)
I AB | Coded Illegal Action, Port B. (See Table 4-2 below)
I AC | Coded Illegal Action, Port C. (See Table 4-2 below)
I AD | Coded Illegal Action, Port D. (See Table 4-2 below)
p | CPAR DIR | A parity error has been detected in the cache store directory.
q | CPAR STR | A data parity error has been detected in the cache store.
r | CPAR IA | An Illegal Action has been received from an SCU during a store operation.
s | CPAR BLK | A cache parity error has occurred during a cache store data block load.
Table 4-2 System Controller Illegal Action Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Priority</th>
<th>Processor Fault</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td></td>
<td>No illegal action</td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>CMD</td>
<td>Unassigned</td>
</tr>
<tr>
<td>02</td>
<td>5</td>
<td>STR</td>
<td>Nonexistent address</td>
</tr>
<tr>
<td>03</td>
<td>1</td>
<td>CMD</td>
<td>Stop on condition</td>
</tr>
<tr>
<td>04</td>
<td></td>
<td>CMD</td>
<td>Unassigned</td>
</tr>
<tr>
<td>05</td>
<td>12</td>
<td>PAR</td>
<td>Data parity, store to SCU</td>
</tr>
<tr>
<td>06</td>
<td>11</td>
<td>PAR</td>
<td>Data parity in store</td>
</tr>
<tr>
<td>07</td>
<td>10</td>
<td>PAR</td>
<td>Data parity in store and store to SCU</td>
</tr>
<tr>
<td>08</td>
<td>4</td>
<td>CMD</td>
<td>Not control</td>
</tr>
<tr>
<td>09</td>
<td>13</td>
<td>CMD</td>
<td>Port not enabled</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>CMD</td>
<td>Illegal command</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>STR</td>
<td>Store not ready</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>PAR</td>
<td>ZAC parity, CPU to SCU</td>
</tr>
<tr>
<td>13</td>
<td>6</td>
<td>PAR</td>
<td>Data parity, CPU to SCU</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>PAR</td>
<td>ZAC parity, SCU to store</td>
</tr>
<tr>
<td>15</td>
<td>9</td>
<td>PAR</td>
<td>Data parity, SCU to store</td>
</tr>
</tbody>
</table>

**MODE REGISTER**

**Format**: 33 bits

Even word of Y-pair as stored by Store Central Processor Register (SCPR), TAG = 06, instruction

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Opcode</td>
</tr>
<tr>
<td>1</td>
<td>FFV</td>
</tr>
<tr>
<td>15</td>
<td>11111011011111111111</td>
</tr>
<tr>
<td>16</td>
<td>11111011011111111111</td>
</tr>
<tr>
<td>17</td>
<td>11111011011111111111</td>
</tr>
<tr>
<td>18</td>
<td>11111011011111111111</td>
</tr>
<tr>
<td>19</td>
<td>11111011011111111111</td>
</tr>
<tr>
<td>20</td>
<td>11111011011111111111</td>
</tr>
<tr>
<td>21</td>
<td>11111011011111111111</td>
</tr>
</tbody>
</table>

**Description**: A logical assemblage of flags and registers from the Control Unit. The Mode Register and the Cache Mode Register are both stored into the Y-pair by the SCPR, TAG = 06, instruction. The Mode Register is loaded with the Load Central Processor Register (LCPR), Tag = 04, instruction. Bits pictured as "0" are reserved and must have the value 0.

The functions of the constituent flags and registers are:
**key Register**

**Function**

**FFV**
A "floating fault vector" address. The 15 most significant bits of the Y-block address of four word pairs constituting a "floating fault vector". Traps to these floating faults are generated by other conditions setable by the mode register.

**a QC TRAP**
Trap on OPCODE match. If this bit is set ON and OPCODE matches the operation code of the instruction for which an address is being prepared (including indirect cycles), generate the second floating fault (XED FFV+2). (See NOTE below)

**b ADR TRAP**
Trap on ADDRESS match. If this bit is set ON and the Computed ADDRESS (TPR.CA) matches the setting of the Address Switches on the Processor Maintenance panel, generate the fourth floating fault (XED FFV+6). (See NOTE below)

**OPCODE**
The operation code on which to trap if OC TRAP (bit 16, key a) is set ON or for which to strobe all CU cycles into the CU History Registers if O.C$é (bit 29, key j) is set ON.

**OC**
Processor conditions codes as follows if OC TRAP (bit 16, key a) and O.C$é (bit 29, key j) are set OFF and & VOLTAGE (bit 32, key m) is set ON.

**key Condition**

c Set Control Unit Overlap Inhibit if set ON. The Control Unit shall wait for the Operations Unit to complete execution of the even instruction of the current instruction pair before it begins address preparation for the associated odd instruction. The Control Unit shall also wait for the Operations Unit to complete execution of the odd instruction before it fetches the next instruction pair.

d Set Store Overlap Inhibit if set ON. The Control Unit shall wait for completion of a current main store fetch (read cycles only) before requesting a main store access for another fetch.

e Set Store Incorrect Data Parity if set ON. The Control Unit shall cause incorrect data parity to be sent to the SCU for each data store instruction and then shall reset bit 20.

f Set Store Incorrect ZAC Parity if Set ON. The Control Unit shall cause incorrect Zone-Address-Command (ZAC) parity to be sent to the SCU for each main store cycle of the next data store instruction and shall reset bit 21 at the end of the instruction.

g Set Timing Margins if set ON. If & VOLT (bit 32, key m) is set ON and the Margin Control switch on the Processor Maintenance panel is in PROG position, set Processor timing margins as follows.
**Function**

<table>
<thead>
<tr>
<th>Key</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.33</td>
<td>margin</td>
</tr>
<tr>
<td>0.0</td>
<td>normal</td>
</tr>
<tr>
<td>0.1</td>
<td>slow</td>
</tr>
<tr>
<td>1.0</td>
<td>normal</td>
</tr>
<tr>
<td>1.1</td>
<td>fast</td>
</tr>
</tbody>
</table>

*Set +5 Voltage Margins if set ON. If & VOLT (bit 32, key m) is set ON and the Margin Control switch on the Processor Maintenance panel is in the PROG position, set +5 voltage margins as follows.*

<table>
<thead>
<tr>
<th>24.25</th>
<th>margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>normal</td>
</tr>
<tr>
<td>0.1</td>
<td>low</td>
</tr>
<tr>
<td>1.0</td>
<td>high</td>
</tr>
<tr>
<td>1.1</td>
<td>normal</td>
</tr>
</tbody>
</table>

*Trap on Control Unit History Register count overflow if set ON. If this bit and STROBE & (bit 30, key k) are set ON and the Control Unit History Register counter overflows, generate the third floating fault (XED FFV+4). Further, if FAULT RESET (bit 31, key l) is set, reset STROBE & (bit 30, key k), locking the history registers. An LCRP, TAG = 04, instruction setting bit 28 ON will reset the Control Unit History Register counter to zero. (See NOTE below)*

| 0.32 & | Strobe Control Unit History Registers on OPCODE match. If this bit and STROBE & (bit 30, key k) are set ON and the operation code of the current instruction matches OPCODE, strobe the Control Unit History Registers on all Control Unit cycles (including indirect cycles).*

<table>
<thead>
<tr>
<th>k</th>
<th>STROBE &amp;</th>
</tr>
</thead>
</table>
| Enable history registers. If this bit is set ON, all history registers are strobed at appropriate points in the various Processor cycles. If this bit is set OFF or MR ENABLE (bit 35, key n) is set OFF, all history registers are locked. This bit is set OFF with an LCRP, TAG = 04, instruction providing a "zero" bit, by an Op Not Complete Fault, and, conditionally, by other faults (See FAULT RESET (bit 31, key l) below). Once set OFF, this bit must be set ON with an LCRP, TAG = 04, instruction providing a "one" bit before the history registers again become active.*

<table>
<thead>
<tr>
<th>l</th>
<th>FAULT RESET</th>
</tr>
</thead>
<tbody>
<tr>
<td>History register lock control. If this bit is set ON, set STROBE &amp; (bit 30, key k) OFF, locking the history registers, for all faults including the floating faults. (See NOTE below)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>m</th>
<th>&amp; VOLT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test mode indicator. This bit is set ON whenever the Test/Normal switch on the Processor Maintenance panel is in Test position and is set OFF otherwise. It serves to enable the program control of Voltage and Timing Margins.*</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>MR ENABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable mode register. When this bit is set ON, all other bits and controls of the mode register are active. When this bit is set OFF, the mode register controls are disabled.*</td>
<td></td>
</tr>
</tbody>
</table>
NOTE

The traps described above (Address match, OPCODE match, Control Unit History Register counter overflow) occur after completion of the next odd instruction following their detection. They are handled as Group VII faults in regard to servicing and inhibition. The complete Group VII priority sequence is...

1 - con
2 - tro
3 - sdf
4 - OPCODE trap
5 - Control Unit History Register counter overflow
6 - Address match trap
7 - External interrupts

CACHE MODE REGISTER (CMR)

Format: 28 bits

Odd word of Y-pair as stored by Store Central Processor Register (SCPR), TAG = 06, instruction

<table>
<thead>
<tr>
<th>3</th>
<th>5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>CACHE DIR ADDRESS</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

Figure 4-19 Cache Mode Register (CMR) Format

Description:

A logical assemblage of flags and registers from the control unit. The Mode Register and Cache Mode Register are both stored into the Y-pair by the SCPR, TAG = 06, instruction.

The Cache Mode Register data stored is address dependent. The algorithm used to map main store into the cache store (See Section XX, Cache Store) is effective for the SCPR instruction. In general, the user may read out data from the directory entry for any cache block by proper selection of certain subfields of the final 24-bit main store address. In particular, the user may read out the directory entry for the cache block involved in a suspected cache error by assuring that the required 24-bit final address subfields are the same as those for the access which produced the suspected error.

WARNING: The user is warned that the fault handling procedure(s) should be unencachable (SDW.C = 0) and that the History Registers and cache should be disabled as quickly as possible in order that vital information concerning the suspected error not be lost.

The Cache Mode Register is loaded with the Load Central Processor Register (LCPR), TAG = 02, instruction. Those items with an "x" in the column headed L are not loaded with the LCPR instruction. Bits pictured as "0" are reserved and must have the value 0.
The functions of the constituent flags and registers are:

<table>
<thead>
<tr>
<th>Key &amp; Register</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>x CACHE DIR ADDRESS</td>
<td>15 most significant bits of the block address from the cache directory</td>
</tr>
<tr>
<td>a PAR BIT</td>
<td>Cache directory parity error on this read out</td>
</tr>
<tr>
<td>b LEV FUL</td>
<td>The selected column and level is loaded with active data</td>
</tr>
<tr>
<td>c CSH1 ON</td>
<td>Enable the upper 1024 words of the cache</td>
</tr>
<tr>
<td>d CSH2 ON</td>
<td>Enable the lower 1024 words of the cache</td>
</tr>
<tr>
<td>e OPND ON</td>
<td>Enable the cache for operands</td>
</tr>
<tr>
<td>f INST ON</td>
<td>Enable the cache for instructions</td>
</tr>
<tr>
<td>g CSH REG</td>
<td>Enable cache-to-register (dump) mode. When this bit is set ON, double precision Operations Unit operands (e.g., LDAQ operands) are read from the cache according to the mapping algorithm and without regard to matching of the full final address. All other operands address main store as though the cache were disabled. This bit is reset automatically by the hardware for any Fault or Program Interrupt.</td>
</tr>
<tr>
<td>h STR ASD</td>
<td>Enable store aside. When this bit is set ON, the Processor does not wait for main store cycle completion after a store operation but proceeds after the cache cycle is complete.</td>
</tr>
<tr>
<td>i COL FUL</td>
<td>Selected cache column is full</td>
</tr>
<tr>
<td>j RRO A,B</td>
<td>Cache round robin counter</td>
</tr>
<tr>
<td>k LUF MSB, LSB</td>
<td>Lockup timer setting. The Lockup Timer may set to four different values according to the setting of this field.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LUF Value</th>
<th>Lockup Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2 ms.</td>
</tr>
<tr>
<td>1</td>
<td>4 ms.</td>
</tr>
<tr>
<td>2</td>
<td>8 ms.</td>
</tr>
<tr>
<td>3</td>
<td>16 ms.</td>
</tr>
</tbody>
</table>

The Lockup Timer is set to 16 ms. when the Processor is initialized.

CONTROL UNIT (CU) HISTORY REGISTERS

Format - 72 bits each

Even word as stored by Store Central Processor Register (SCPR), Tag = 20, instruction.
Odd word as stored by Store Central Processor register (SCPR), TAG = 20, instruction

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
0 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
QPCODE & \text{TAG} & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

Figure 4-20 Control Unit (CU) History Register Format

**Description**

Sixteen logical combinations of flags and registers from the Control Unit. The sixteen registers are handled as a rotating queue controlled by the Control Unit History Register counter. The counter is always set to the number of the oldest entry and advances by one for each history register reference (data entry or SCPR). True multicycle instructions (such as "ipri, lreg, rcu, etc.) will have an entry for each of their cycles.

**Function**

A Control Unit History Register entry shows the conditions at the end of the Control Unit cycle to which it applies. The sixteen registers will hold the conditions for the last sixteen Control Unit cycles. Entries are made according to controls set in the Mode Register. (See Mode Register above)

The meanings of the constituent flags and registers are:

<table>
<thead>
<tr>
<th>Key</th>
<th>Flag Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>PIA</td>
<td>prepare instruction address</td>
</tr>
<tr>
<td>b</td>
<td>POA</td>
<td>prepare operand address</td>
</tr>
<tr>
<td>c</td>
<td>RIW</td>
<td>request indirect word</td>
</tr>
<tr>
<td>d</td>
<td>SIW</td>
<td>restore indirect word</td>
</tr>
<tr>
<td>e</td>
<td>POT</td>
<td>prepare operand tally (indirect tally chain)</td>
</tr>
<tr>
<td>f</td>
<td>PON</td>
<td>prepare operand notally (as for POT except no chain)</td>
</tr>
<tr>
<td>g</td>
<td>RAW</td>
<td>request read-alter-rewrite word</td>
</tr>
<tr>
<td>h</td>
<td>SAN</td>
<td>restore read-alter-rewrite word</td>
</tr>
<tr>
<td>Key</td>
<td>Flag_Name</td>
<td>Meaning</td>
</tr>
<tr>
<td>-----</td>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td>l</td>
<td>TRGO</td>
<td>transfer GO (conditions met)</td>
</tr>
<tr>
<td>j</td>
<td>XDE</td>
<td>execute XED from even IC</td>
</tr>
<tr>
<td>k</td>
<td>XDO</td>
<td>execute XED from odd IC</td>
</tr>
<tr>
<td>l</td>
<td>IC</td>
<td>execute odd instruction of the current pair</td>
</tr>
<tr>
<td>m</td>
<td>RPTS</td>
<td>execute a repeat operation</td>
</tr>
<tr>
<td>n</td>
<td>WI</td>
<td>wait for instruction fetch</td>
</tr>
<tr>
<td>o</td>
<td>AR F/E</td>
<td>( i = ) Computed Address (TPR.CA) has valid data</td>
</tr>
<tr>
<td>p</td>
<td>XIP</td>
<td>NOT prepare Program Interrupt address</td>
</tr>
<tr>
<td>q</td>
<td>FLT</td>
<td>NOT prepare Fault address</td>
</tr>
<tr>
<td>r</td>
<td>BASE</td>
<td>NOT BAR mode</td>
</tr>
<tr>
<td>0</td>
<td>OPCODE</td>
<td>current operation code</td>
</tr>
<tr>
<td>i</td>
<td>I</td>
<td>Program Interrupt inhibit bit</td>
</tr>
<tr>
<td>p</td>
<td>P</td>
<td>Pointer register flag bit</td>
</tr>
<tr>
<td>6</td>
<td>TAG</td>
<td>Current address modifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This modifier is replaced by the contents of the TAG fields of indirect words as they are fetched during indirect chains.</td>
</tr>
<tr>
<td>ADDRESS</td>
<td></td>
<td>Current Computed Address (TPR.CA)</td>
</tr>
<tr>
<td>CMD</td>
<td></td>
<td>SCU command</td>
</tr>
<tr>
<td>SEL</td>
<td></td>
<td>Port select bits. (Valid only if Port A through D is selected)</td>
</tr>
<tr>
<td>s</td>
<td>XEC-INT</td>
<td>A Program Interrupt is present</td>
</tr>
<tr>
<td>t</td>
<td>INS-FETCH</td>
<td>Perform an instruction fetch</td>
</tr>
<tr>
<td>u</td>
<td>CU-STORE</td>
<td>Control Unit store cycle</td>
</tr>
<tr>
<td>v</td>
<td>OU-STORE</td>
<td>Operations Unit store cycle</td>
</tr>
<tr>
<td>w</td>
<td>CU-LOAD</td>
<td>Control Unit load cycle</td>
</tr>
<tr>
<td>x</td>
<td>OU-LOAD</td>
<td>Operations Unit load cycle</td>
</tr>
<tr>
<td>y</td>
<td>DIRECT</td>
<td>direct cycle</td>
</tr>
<tr>
<td>z</td>
<td>PC-BUSY</td>
<td>Port control logic not busy</td>
</tr>
<tr>
<td>*</td>
<td>BUSY</td>
<td>Port interface busy</td>
</tr>
</tbody>
</table>
**OPERATIONS UNIT (OU) HISTORY REGISTERS**

*Format* - 72 bits each

**Even word as stored by Store central Processor Register (SCPR), TAG = 40:**

Instruction

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<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>RP REG</td>
<td>1</td>
<td>1</td>
<td>1</td>
<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>OP CODE</td>
<td>IA</td>
<td>IA1</td>
<td>1</td>
<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>

**Odd word as stored by Store Central Processor Register (SCPR), TAG = 40:**

Instruction

|       | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| INTLPQIR| IA1Q | IA1Q1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ICT TRACER| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

*Figure 4-21 Operations Unit (OU) History Register Format*

**Description**

Sixteen logical combinations of flags and registers from the Operations Unit and Control Unit. The sixteen registers are handled as a rotating queue controlled by the Operations Unit History Register counter. The counter is always set to the number of the oldest entry and advances by one for each history register reference (data entry or SCPR).

**Function**

An Operations Unit History Register entry shows the conditions at the end of the Operations Unit cycle to which it applies. The sixteen registers will hold the conditions for the last sixteen Operations Unit cycles. Entries are made according to controls set in the Mode Register. (See Mode Register above)

The meanings of the constituent flags and registers are:

<table>
<thead>
<tr>
<th>Key Flag Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP REG</td>
<td>Primary Operations Unit operation register</td>
</tr>
<tr>
<td></td>
<td>RP REG receives the instruction operation code and other data from the Control Unit during the Control Unit instruction cycle while the Operations Unit may be busy with a prior operation. RP REG is further sub-structured as ...</td>
</tr>
</tbody>
</table>
### Key Flag Name and Meaning

<table>
<thead>
<tr>
<th>Flag Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP CODE</td>
<td>The 9 most significant bits of the operation code for the instruction. Note that basic (non EIS) operations do not involve bit 27 hence the 9 bit field is sufficient to define the operation code.</td>
</tr>
</tbody>
</table>
| a 9 CHAR  | Character size for Indirect Then Tally modifiers  
0 = 6-bit  
1 = 9-bit |
| b TAG1,2,3| The 3 least significant bits of the modifier of the instruction. This field may contain a character position for an Indirect Then Tally character modifier. |
| c CR FLG  | Character operation flag |
| d DK FLG  | Direct operation flag |
| EAC       | Effective address counter for LREG/SREG instructions |
| RS REG    | Secondary Operations Unit operation register  
OP CODE is moved from RP REG to RS REG during the operand fetch and is held until completion of the instruction. |
| e RB1 FULL| OP CODE buffer full |
| f RP FULL | RP REG full |
| g RS FULL | RS REG full |
| h GIN     | First cycle for all Operations Unit operations |
| i GOS     | Second cycle for Operations Unit multi-ops |
| j GD1     | First divide cycle |
| k GD2     | Second divide cycle |
| l GOE     | Exponent compare cycle |
| m GOA     | Mantissa alignment cycle |
| n GOM     | General Operations Unit cycle |
| o GON     | Normalize cycle |
| p GOF     | Final Operations Unit cycle |
| q STR OP  | Operations Unit store data available |
| t DA-AV   | Data not available |

#### Register States

<table>
<thead>
<tr>
<th>Register</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A-REG</td>
</tr>
<tr>
<td>Q</td>
<td>Q-REG</td>
</tr>
<tr>
<td>0</td>
<td>X0-RG</td>
</tr>
<tr>
<td>1</td>
<td>X1-RG</td>
</tr>
<tr>
<td>2</td>
<td>X2-RG</td>
</tr>
</tbody>
</table>

*REVIEW DRAFT  
SUBJECT TO CHANGE  
October, 1975*
<table>
<thead>
<tr>
<th>Key Flag Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 X3-RG</td>
<td>X3 not in use</td>
</tr>
<tr>
<td>4 X4-RG</td>
<td>X4 not in use</td>
</tr>
<tr>
<td>5 X5-RG</td>
<td>X5 not in use</td>
</tr>
<tr>
<td>6 X6-RG</td>
<td>X6 not in use</td>
</tr>
<tr>
<td>7 X7-RG</td>
<td>X7 not in use</td>
</tr>
</tbody>
</table>

ICT TRACKER

The current value of the Instruction Counter (PPR, IC). Since the Control Unit and Operations Unit run asynchronously and overlap is usually enabled, the value of ICT TRACKER may not be the address of the Operations Unit instruction currently being executed.

DECIMAL UNIT (OU) HISTORY REGISTERS

Format - 72 bits each

Decimal Unit History Register data is stored with the Store Central Processor Register (SCPR), TAG = 60, Instruction. No Format diagram is given because the data is defined as individual bits.

Description

Sixteen logical combinations of flags from the Decimal Unit. The sixteen registers are handled as a rotating queue controlled by the Decimal Unit History Register counter. The counter is always set to the number of the oldest entry and advances by one for each history register reference (data entry or SCPR).

The Decimal Unit and the Control Unit run synchronously. There is a Control Unit History Register entry for every Decimal Unit History Register entry and vice versa. If the Processor is not executing a Decimal operation, the Decimal Unit History Register entry will show an idle condition.

Function

A Decimal Unit History Register entry shows the conditions in the Decimal Unit at the end of the Control Unit cycle to which it applies. The sixteen registers will hold the conditions for the last sixteen Control Unit cycles. Entries are made according to controls set in the Mode Register.

(See Mode Register above)

A minus (-) sign preceding the flag name indicates that the complement of the flag is shown. Unused bits are set ON.

The meanings of the constituent flags are:
<table>
<thead>
<tr>
<th>bit</th>
<th>Flag Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-FPOL</td>
<td>Prepare operand length</td>
</tr>
<tr>
<td>1</td>
<td>-FPOP</td>
<td>Prepare operand pointer</td>
</tr>
<tr>
<td>2</td>
<td>-NEED-DESC</td>
<td>Need descriptor</td>
</tr>
<tr>
<td>3</td>
<td>-SEL-ADR</td>
<td>Select address register</td>
</tr>
<tr>
<td>4</td>
<td>-DLEN=DIRECT</td>
<td>Length equals direct</td>
</tr>
<tr>
<td>5</td>
<td>-DFRST</td>
<td>Descriptor processed for first time</td>
</tr>
<tr>
<td>6</td>
<td>-FEXR</td>
<td>Extended register modification</td>
</tr>
<tr>
<td>7</td>
<td>-DLAST-FRST</td>
<td>Last cycle of DFRST</td>
</tr>
<tr>
<td>8</td>
<td>-DDU-LOEA</td>
<td>Decimal Unit load</td>
</tr>
<tr>
<td>9</td>
<td>-DDU-STAE</td>
<td>Decimal Unit store</td>
</tr>
<tr>
<td>10</td>
<td>-DOREDO</td>
<td>Redo operation without pointer and length update</td>
</tr>
<tr>
<td>11</td>
<td>-OLVL&lt;WD-SZ</td>
<td>Load with count less than word size</td>
</tr>
<tr>
<td>12</td>
<td>-EXH</td>
<td>Exhaust</td>
</tr>
<tr>
<td>13</td>
<td>DEND-SEQ</td>
<td>End of sequence</td>
</tr>
<tr>
<td>14</td>
<td>DEND</td>
<td>End of instruction</td>
</tr>
<tr>
<td>15</td>
<td>-DU=RO+WRT</td>
<td>Decimal Unit write-back</td>
</tr>
<tr>
<td>16</td>
<td>-PTRA00</td>
<td>PR address bit 0</td>
</tr>
<tr>
<td>17</td>
<td>-PTRA01</td>
<td>PR address bit 1</td>
</tr>
<tr>
<td>18</td>
<td>FA/I1</td>
<td>Descriptor 1 active</td>
</tr>
<tr>
<td>19</td>
<td>FA/I2</td>
<td>Descriptor 2 active</td>
</tr>
<tr>
<td>20</td>
<td>FA/I3</td>
<td>Descriptor 3 active</td>
</tr>
<tr>
<td>21</td>
<td>-WRD</td>
<td>Word operation</td>
</tr>
<tr>
<td>22</td>
<td>-NINE</td>
<td>9-bit character operation</td>
</tr>
<tr>
<td>23</td>
<td>-SIX</td>
<td>6-bit character operation</td>
</tr>
<tr>
<td>24</td>
<td>-FOUR</td>
<td>4-bit character operation</td>
</tr>
<tr>
<td>25</td>
<td>-BIT</td>
<td>8-bit operation</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td>30</td>
<td>FSAMPL</td>
<td>Sample for mid-instruction interrupt</td>
</tr>
<tr>
<td>bit</td>
<td>Flag Name</td>
<td>Meaning</td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>31</td>
<td>-DFRST-CT</td>
<td>Specified first count of a sequence</td>
</tr>
<tr>
<td>32</td>
<td>-ADJ-LENGTH</td>
<td>Adjust length</td>
</tr>
<tr>
<td>33</td>
<td>-INTRPTD</td>
<td>Mid-instruction interrupt</td>
</tr>
<tr>
<td>34</td>
<td>-INHIB</td>
<td>Inhibit STC1 (force &quot;STC0&quot;)</td>
</tr>
<tr>
<td>35</td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td>36</td>
<td>DUD</td>
<td>Decimal Unit idle</td>
</tr>
<tr>
<td>37</td>
<td>-GOLDA</td>
<td>Descriptor load gate A</td>
</tr>
<tr>
<td>38</td>
<td>-GOLDB</td>
<td>Descriptor load gate B</td>
</tr>
<tr>
<td>39</td>
<td>-GOLOC</td>
<td>Descriptor load gate C</td>
</tr>
<tr>
<td>40</td>
<td>NLD1</td>
<td>Prepare alignment count for first numeric operand load</td>
</tr>
<tr>
<td>41</td>
<td>GLDP1</td>
<td>Numeric operand one load gate</td>
</tr>
<tr>
<td>42</td>
<td>NLD2</td>
<td>Prepare alignment count for second numeric operand load</td>
</tr>
<tr>
<td>43</td>
<td>GLDP2</td>
<td>Numeric operand two load gate</td>
</tr>
<tr>
<td>44</td>
<td>ANLD1</td>
<td>Alphanumeric operand one load gate</td>
</tr>
<tr>
<td>45</td>
<td>ANLD2</td>
<td>Alphanumeric operand two load gate</td>
</tr>
<tr>
<td>46</td>
<td>LDWRT1</td>
<td>Load rewrite register one gate</td>
</tr>
<tr>
<td>47</td>
<td>LDWRT2</td>
<td>Load rewrite register two gate</td>
</tr>
<tr>
<td>48</td>
<td>-DATA-AVLDU</td>
<td>Decimal Unit data available</td>
</tr>
<tr>
<td>49</td>
<td>WRT1</td>
<td>Rewrite register one loaded</td>
</tr>
<tr>
<td>50</td>
<td>GSTR</td>
<td>Numeric store gate</td>
</tr>
<tr>
<td>51</td>
<td>ANSTR</td>
<td>Alphanumeric store gate</td>
</tr>
<tr>
<td>52</td>
<td>FSTR-OP-AV</td>
<td>Operand available to be stored</td>
</tr>
<tr>
<td>53</td>
<td>-FEND-SEQ</td>
<td>End sequence flag</td>
</tr>
<tr>
<td>54</td>
<td>-FLEN&lt;128</td>
<td>Length less than 128</td>
</tr>
<tr>
<td>55</td>
<td>FGCH</td>
<td>Character operation gate</td>
</tr>
<tr>
<td>56</td>
<td>FANPK</td>
<td>Alphanumeric packing cycle gate</td>
</tr>
<tr>
<td>57</td>
<td>FEXMOP</td>
<td>Execute MOP gate</td>
</tr>
<tr>
<td>58</td>
<td>FBLNK</td>
<td>Blanking gate</td>
</tr>
<tr>
<td>59</td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td>60</td>
<td>DGBD</td>
<td>Binary to decimal execution gate</td>
</tr>
<tr>
<td>61</td>
<td>DGDB</td>
<td>Decimal to binary execution gate</td>
</tr>
</tbody>
</table>

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975 4-36
### Bit Flag Name Meaning

<table>
<thead>
<tr>
<th>Bit</th>
<th>Flag Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>DGSP</td>
<td>Shift procedure gate</td>
</tr>
<tr>
<td>63</td>
<td>FFLTG</td>
<td>Floating result flag</td>
</tr>
<tr>
<td>64</td>
<td>FRND</td>
<td>Rounding flag</td>
</tr>
<tr>
<td>65</td>
<td>DADD-GATE</td>
<td>Add/subtract execute gate</td>
</tr>
<tr>
<td>66</td>
<td>DMP+DV-GATE</td>
<td>Multiply/divide execution gate</td>
</tr>
<tr>
<td>67</td>
<td>DXPN-GATE</td>
<td>Exponent network execution gate</td>
</tr>
<tr>
<td>68</td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td>69</td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td>70</td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td>71</td>
<td></td>
<td>Unused</td>
</tr>
</tbody>
</table>

### Appendix: Appending Unit (AU) History Registers

**Format:** 72 bits each

#### Even word as stored by Store Central Processor Register (SCPR), TAG = 00, instruction

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>111111122222223333</td>
</tr>
<tr>
<td>1</td>
<td>456789012345690145</td>
</tr>
<tr>
<td>1</td>
<td>1111111111111111</td>
</tr>
<tr>
<td>1</td>
<td>1111111111111111</td>
</tr>
<tr>
<td>15</td>
<td>2111111111111111</td>
</tr>
</tbody>
</table>

#### Odd word as stored by Store Central Processor Register (SCPR), TAG = 00, instruction

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2222233333</td>
</tr>
<tr>
<td>1</td>
<td>3467901345</td>
</tr>
<tr>
<td>1</td>
<td>1111111111</td>
</tr>
<tr>
<td>1</td>
<td>1111111111</td>
</tr>
<tr>
<td>24</td>
<td>331311</td>
</tr>
</tbody>
</table>

**Figure 4-22 Appending Unit (AU) History Register Format**

**Description:**

Sixteen logical combinations of flags and registers from the Appending Unit. The sixteen registers are handled as a rotating queue controlled by the Appending Unit History Register counter. The counter is always set to the number of the oldest entry and advances by one for each history.
An Appending Unit History Register entry shows the conditions in the Appending Unit at the end of an address preparation cycle in Appending Mode. The sixteen registers will hold the conditions for the last sixteen such address preparation cycles. Entries are made according to controls set in the Mode Register. (See Mode Register above)

The meanings of the constituent flags and registers are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Flag Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>BSY</td>
<td>Data source for ESN</td>
</tr>
<tr>
<td>b</td>
<td>FOSTPW</td>
<td>Descriptor segment PTW fetch</td>
</tr>
<tr>
<td>c</td>
<td>MDSPTW</td>
<td>Descriptor segment PTW modification</td>
</tr>
<tr>
<td>d</td>
<td>FSDWP</td>
<td>SDW fetch from paged descriptor segment</td>
</tr>
<tr>
<td>e</td>
<td>FPTW</td>
<td>PTW fetch</td>
</tr>
<tr>
<td>f</td>
<td>FPTW2</td>
<td>PTW+1 fetch</td>
</tr>
<tr>
<td>g</td>
<td>MPTW</td>
<td>PTW modification</td>
</tr>
<tr>
<td>h</td>
<td>FANP</td>
<td>Final address fetch from non-paged segment</td>
</tr>
<tr>
<td>i</td>
<td>FAP</td>
<td>Final address fetch from paged segment</td>
</tr>
<tr>
<td>j</td>
<td>SDWAMM</td>
<td>SDWAM match occurred</td>
</tr>
<tr>
<td></td>
<td>SDWAMR</td>
<td>SDWAM register number for SDWAMM=1</td>
</tr>
<tr>
<td>k</td>
<td>PTWAMM</td>
<td>PTWAM match occurred</td>
</tr>
<tr>
<td></td>
<td>PTWAMR</td>
<td>PTWAM register number for PTWAMM=1</td>
</tr>
<tr>
<td>l</td>
<td>FLT</td>
<td>ACV or DFTn fault on this cycle</td>
</tr>
<tr>
<td></td>
<td>ADD</td>
<td>24 bit final address from this cycle</td>
</tr>
<tr>
<td></td>
<td>TRR</td>
<td>Ring number from this cycle (TPR, TRR)</td>
</tr>
<tr>
<td>m</td>
<td>CA</td>
<td>Segment is encacheable</td>
</tr>
<tr>
<td>n</td>
<td>FHLD</td>
<td>An ACV or DFTn is waiting</td>
</tr>
</tbody>
</table>
CONFIGURATION SWITCH DATA

**Format 1 - 35 bits each**

Data read by Read Switches (RSW), Y = xxxxx0, instruction

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

Maintenance Panel Data Switches

Data read by Read Switches (RSW), Y = xxxxx2, instruction

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

Data read by Read Switches (RSW), Y = xxxxx1/3, instruction

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

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

Data read by Read Switches (RSW), Y = xxxxx4, instruction

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<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>

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

Figure 4-23 Configuration Switch Data Formats

**Description**

The Read Switches (RSW) instruction provides the ability to interrogate various switches and options on the Processor Maintenance and Configuration panels. The least significant digit (bits 15-17) of the address field is used to select the switches to be read. High order address bits are ignored. Data is placed in the A-Register. Bits pictured as "0" are unimplemented or represent options that are standard on the Multics Processor. Bits pictured as "1" represent options that are standard on the Multics Processor.

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975

4-39

AL39
Read Switches (RSW), \( Y = \text{xxxxx1} \) reads data for Ports A, B, C, and D.

Read Switches (RSW), \( Y = \text{xxxxx3} \) reads data for Ports E, F, G, and H.

**Functions**

The meanings of the constituent fields are:

**Key Field Name** | **Meaning**
--- | ---
FLT BASE | Seven most significant bits of the 12 bit Fault Base Address
a | Cache option
\( 0 = \text{enabled} \)
\( 1 = \text{disabled} \)
b | Main store speed option
\( 0 = \text{slow} \)
\( 1 = \text{fast} \)
CPU | Processor number
PORT A/E, etc. | Port data fields further substructured as...
ADR | Address Assignment Switch setting for port
c | Port enabled flag
d | System Initialize enabled flag
e | Interlace enabled flag
MEN | Coded memory size...

<table>
<thead>
<tr>
<th>Value</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>32K</td>
</tr>
<tr>
<td>001</td>
<td>64K</td>
</tr>
<tr>
<td>010</td>
<td>96K or 160K</td>
</tr>
<tr>
<td>011</td>
<td>128K</td>
</tr>
<tr>
<td>100</td>
<td>512K</td>
</tr>
<tr>
<td>101</td>
<td>1024K</td>
</tr>
<tr>
<td>110</td>
<td>2048K</td>
</tr>
<tr>
<td>111</td>
<td>256K</td>
</tr>
</tbody>
</table>

A, B, etc. | Port data fields further substructured as...
f | Interlace mode
\( 0 = 4 \text{ word if interlace enabled for port} \)
\( 1 = 2 \text{ word if interlace enabled for port} \)
g | Main store size
\( 0 = \text{full, all of MEM is configured} \)
\( 1 = \text{half, half of MEM is configured} \)

**CONTROL_UNIT DATA**

*Format* - 288 bits, 8 machine words
Data as stored by Store Control Unit (SCU) instruction

Figure 4-24 Control Unit Data Format

Description:
A logical collection of flags and registers from the Appending Unit and the Control Unit. In general, the data has valid meaning only when stored with the Store Control Unit (SCU) instruction as the first instruction of a Fault Trap pair. Bits pictured as "0" are reserved and must have the value 0.

Functional

The Control Unit Data allows the Processor to restart an instruction at the point of interruption when it is interrupted by an Access Violation Fault, a Directed Fault, or (for certain EIS instructions) a Program Interrupt. Directed Faults are intentional, and most Access Violation Faults and Program Interrupts are recoverable. If the interruption is not recoverable, the Control Unit Data provides enough information to determine the exact nature of the error.

Instruction execution restarts immediately upon execution of a Restore Control Unit (RCU) instruction referencing the Y-block area into which the Control Unit Data was stored.

Fields having an "x" in the column headed L are not restored by the Restore Control Unit (RCU) instruction.

<table>
<thead>
<tr>
<th>Word</th>
<th>Key</th>
<th>Field Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PPR.PRR</td>
<td>Procedure ring register</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>PPR.PSR</td>
<td>Procedure segment register</td>
<td></td>
</tr>
<tr>
<td>0 a</td>
<td>PPR.P</td>
<td>Privileged bit</td>
<td></td>
</tr>
<tr>
<td>0 b</td>
<td>XSF</td>
<td>External segment flag</td>
<td></td>
</tr>
<tr>
<td>0 c</td>
<td>SDWAM.SDWAHMM</td>
<td>Match on SDWAM</td>
<td></td>
</tr>
<tr>
<td>0 d</td>
<td>SD-ON</td>
<td>SDWAM enabled</td>
<td></td>
</tr>
<tr>
<td>0 e</td>
<td>PTWAM.PTWAMMM</td>
<td>Match on PTWAM</td>
<td></td>
</tr>
<tr>
<td>0 f</td>
<td>PT-ON</td>
<td>PTWAM enabled</td>
<td></td>
</tr>
<tr>
<td>0 g</td>
<td>PI-AP</td>
<td>Instruction fetch append cycle</td>
<td></td>
</tr>
<tr>
<td>0 h</td>
<td>DSPTW</td>
<td>Fetch Descriptor Segment PTW</td>
<td></td>
</tr>
<tr>
<td>0 i</td>
<td>SDNP</td>
<td>Fetch SDW - nonpaged</td>
<td></td>
</tr>
<tr>
<td>0 j</td>
<td>SDWP</td>
<td>Fetch SDW - paged</td>
<td></td>
</tr>
<tr>
<td>0 k</td>
<td>PTH</td>
<td>Fetch PTW</td>
<td></td>
</tr>
<tr>
<td>0 l</td>
<td>PTH2</td>
<td>Fetch prepage PTW</td>
<td></td>
</tr>
<tr>
<td>0 m</td>
<td>FAP</td>
<td>Fetch final address - paged</td>
<td></td>
</tr>
<tr>
<td>0 n</td>
<td>FANP</td>
<td>Fetch final address - nonpaged</td>
<td></td>
</tr>
<tr>
<td>0 o</td>
<td>FABS</td>
<td>Fetch final address - absolute</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>FCT</td>
<td>Fault counter - counts instruction retries</td>
<td></td>
</tr>
<tr>
<td>Word Key</td>
<td>Field Name</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a x IRO</td>
<td>For ACV - illegal ring order</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>For STR - illegal segment number</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>b x OEB</td>
<td>For ACV - out of execute bracket</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>For IPR - illegal op code</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>c x E-OFF</td>
<td>For ACV - execute bit is off.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>For IPR - illegal address or modifier</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>d x ORB</td>
<td>For ACV - out of read bracket</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>For IPR - illegal slave procedure</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>e x R-OFF</td>
<td>For ACV - read bit is off</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>For IPR - illegal EIS digit</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>f x Q0B</td>
<td>For ACV - out of write bracket</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>For STR - nonexistent address</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>g x W-OFF</td>
<td>For ACV - write bit is off</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>For STR - out of bounds</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>h x NO GA</td>
<td>For ACV - not a gate</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>i x OCB</td>
<td>For ACV - out of call bracket</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>j x OCALL</td>
<td>For ACV - outward call</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>k x BOC</td>
<td>For ACV - bad outward call</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>l x INRET</td>
<td>For ACV - inward return</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>m x CRT</td>
<td>For ACV - cross ring transfer</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>n x RALR</td>
<td>For ACV - ring alarm</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>o x AM-ER</td>
<td>For ACV - associative memory error</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>p x OOSB</td>
<td>For ACV - out of segment bounds</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>q x PARU</td>
<td>For PAR - processor parity upper</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>r x PARL</td>
<td>For PAR - processor parity lower</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>s x ONC1</td>
<td>For ONC - CPU/SCU sequence error #1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>t x ONC2</td>
<td>For ONC - CPU/SCU sequence error #2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x IA</td>
<td>SCU illegal action lines (See Table 4-2)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x IACHN</td>
<td>Illegal action CPU port.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x CNCHN</td>
<td>For CON - connect (CIOC) CPU port</td>
<td></td>
</tr>
</tbody>
</table>

1 x F/I ADDR  Modulo 2 fault/interrupt vector address
1 u x F/I  Fault/interrupt bit flag bit
0 = interrupt  
1 = fault
2 TPR:TRR  Temporary ring register
<table>
<thead>
<tr>
<th>Word hex</th>
<th>Field Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>TPR.TSR</td>
<td>Temporary segment register</td>
</tr>
<tr>
<td>2</td>
<td>CPU</td>
<td>CPU number</td>
</tr>
<tr>
<td>2</td>
<td>DELTA</td>
<td>Address increment for repeats</td>
</tr>
<tr>
<td>3</td>
<td>TSNA</td>
<td>Pointer Register number for non-EIS operands or for EIS operand #1 further substructured as...</td>
</tr>
<tr>
<td>3 a</td>
<td>PRNO</td>
<td>Pointer register number</td>
</tr>
<tr>
<td>3 b</td>
<td>----</td>
<td>1 = PRNO is valid</td>
</tr>
<tr>
<td>3</td>
<td>TSNB</td>
<td>Pointer Register number for EIS operand #2 further substructured as for TSNA above</td>
</tr>
<tr>
<td>3</td>
<td>TSNC</td>
<td>Pointer Register number for EIS operand #3 further substructured as for TSNA above</td>
</tr>
<tr>
<td>3</td>
<td>TEMP.BIT</td>
<td>BITNO field of Temporary Pointer Register (TPR.TBR)</td>
</tr>
<tr>
<td>4</td>
<td>PPR.IC</td>
<td>Instruction counter</td>
</tr>
<tr>
<td>4 a</td>
<td>ZERO</td>
<td>Zero indicator</td>
</tr>
<tr>
<td>4 b</td>
<td>NEG</td>
<td>Negative indicator</td>
</tr>
<tr>
<td>4 c</td>
<td>CARY</td>
<td>Carry indicator</td>
</tr>
<tr>
<td>4 d</td>
<td>OVFL</td>
<td>Overflow indicator</td>
</tr>
<tr>
<td>4 e</td>
<td>EOVF</td>
<td>Exponent overflow indicator</td>
</tr>
<tr>
<td>4 f</td>
<td>EUFL</td>
<td>Exponent underflow indicator</td>
</tr>
<tr>
<td>4 g</td>
<td>OFLM</td>
<td>Overflow mask indicator</td>
</tr>
<tr>
<td>4 h</td>
<td>TRO</td>
<td>Tally runout indicator</td>
</tr>
<tr>
<td>4 i</td>
<td>PAR</td>
<td>Parity error indicator</td>
</tr>
<tr>
<td>4 j</td>
<td>PARM</td>
<td>Parity mask indicator</td>
</tr>
<tr>
<td>4 k</td>
<td>BM</td>
<td>Not BAR Mode indicator</td>
</tr>
<tr>
<td>4 l</td>
<td>TRU</td>
<td>EIS truncation indicator</td>
</tr>
<tr>
<td>4 m</td>
<td>MIF</td>
<td>Mid-instruction interrupt</td>
</tr>
<tr>
<td>4 n</td>
<td>ABS</td>
<td>Absolute mode</td>
</tr>
<tr>
<td>5 x</td>
<td>TPR.CA</td>
<td>Current Effective Address</td>
</tr>
<tr>
<td>5 a</td>
<td>RF</td>
<td>First cycle of a repeat operation</td>
</tr>
<tr>
<td>5 b</td>
<td>RPT</td>
<td>Executing a repeat</td>
</tr>
<tr>
<td>5 c</td>
<td>RD</td>
<td>Executing a repeat double</td>
</tr>
</tbody>
</table>
Word key & Field Name | Meaning
--- | ---
5 d RL | Executing a repeat link
5 e POT | Prepare operand tally
This flag is up until the indirect word of an IT indirect cycle is successfully fetched.
5 f PON | Prepare operand notally
This flag is up until the indirect word of a "return" type instruction is successfully fetched. It indicates that there is no indirect chain even though an indirect fetch is being done.
5 g XDE | Execute double from even IC
5 h XDO | Execute double from odd IC
5 i ITP | ITP cycle
5 j RST | Restart this instruction
5 k ITS | Executing ITS indirect cycle
5 l FIF | Fault occurred during instruction fetch
5 m CT HOLD | Contents of the "remember modifier" register

Word 6 is the contents of the "working instruction register" and reflects conditions at the exact point of address preparation when the fault/interrupt occurred. The ADDRESS and TAG fields are replaced with data from pointer registers, indirect pointers, and/or indirect words during each indirect cycle. Each instruction of the current pair is moved to this register before actual address preparation begins.

Word 7 is the contents of the "instruction holding register". It contains the odd word of the last instruction pair fetched from main store. Note that, primarily because of store overlap, this instruction is not necessarily paired with the instruction in Word 6.

DECIMAL UNIT DATA

Format - 288 bits, 8 machine words
<table>
<thead>
<tr>
<th>#</th>
<th>Data as stored by Store Pointers and Lengths (SPL) instruction Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 1 1 1</td>
</tr>
<tr>
<td>0</td>
<td>8 9 0 1 2</td>
</tr>
<tr>
<td>0</td>
<td>3 4 5 6 7</td>
</tr>
<tr>
<td>0</td>
<td>9 0 1 2 3 4 5</td>
</tr>
<tr>
<td>0</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>2 2 2 2 2 2 3 3 3 3 3</td>
</tr>
<tr>
<td>1</td>
<td>3 4 5 6 7 9 0 1 2 3 4 5</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>2 2 2 2 2 2 3 3 3 3 3</td>
</tr>
<tr>
<td>1</td>
<td>3 4 5 6 7 9 0 1 2 3 4 5</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>2 2 2 2 2 2 3 3 3 3 3</td>
</tr>
<tr>
<td>1</td>
<td>3 4 5 6 7 9 0 1 2 3 4 5</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>2 2 2 2 2 2 3 3 3 3 3</td>
</tr>
<tr>
<td>1</td>
<td>3 4 5 6 7 9 0 1 2 3 4 5</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>2 2 2 2 2 2 3 3 3 3 3</td>
</tr>
<tr>
<td>1</td>
<td>3 4 5 6 7 9 0 1 2 3 4 5</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>2 2 2 2 2 2 3 3 3 3 3</td>
</tr>
<tr>
<td>1</td>
<td>3 4 5 6 7 9 0 1 2 3 4 5</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>2 2 2 2 2 2 3 3 3 3 3</td>
</tr>
<tr>
<td>1</td>
<td>3 4 5 6 7 9 0 1 2 3 4 5</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>2 2 2 2 2 2 3 3 3 3 3</td>
</tr>
<tr>
<td>1</td>
<td>3 4 5 6 7 9 0 1 2 3 4 5</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>2 2 2 2 2 2 3 3 3 3 3</td>
</tr>
<tr>
<td>1</td>
<td>3 4 5 6 7 9 0 1 2 3 4 5</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>2 2 2 2 2 2 3 3 3 3 3</td>
</tr>
<tr>
<td>1</td>
<td>3 4 5 6 7 9 0 1 2 3 4 5</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

Figure 4-25 Decimal Unit Data Format
**Description**

A logical collection of flags and registers from the Decimal Unit. Bits pictured as "0" are reserved and must have the value 0.

**Function**

The Decimal Unit Data allows the Processor to restart an EIS instruction at the point of interruption when it is interrupted by an Access Violation Fault, a Directed Fault, or (for certain EIS instructions) a Program Interrupt. Directed Faults are intentional, and most Access Violation Faults and Program Interrupts are recoverable.

The data are restored with the Load Pointers and Lengths (LPL) instruction. Fields having an "x" in the column headed L are not restored. When starting execution of an EIS instruction, the decimal unit registers and flags are not initialized from the Operand Descriptors if the Mid-instruction Interrupt Fault (MIF) indicator is set ON.

The meanings of the constituent flags and registers are:

<table>
<thead>
<tr>
<th>Word L</th>
<th>Field Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Z</td>
<td>All bit string instruction results are zero</td>
<td></td>
</tr>
<tr>
<td>0 B</td>
<td>Negative overpunch found in 6-4 expanded move</td>
<td></td>
</tr>
<tr>
<td>0 CHTALLY</td>
<td>The number of characters examined by the SCAN, TCT, or TCTR instruction (up to the interrupt or match)</td>
<td></td>
</tr>
<tr>
<td>2 D1 PTR</td>
<td>Address of last double word accessed by Operand Descriptor 1; bits 17-23 (bit address) valid only for initial access</td>
<td></td>
</tr>
<tr>
<td>2,4,6 TA</td>
<td>Alphanumeric type of Operand Descriptor 1,2,3</td>
<td></td>
</tr>
<tr>
<td>2 x I</td>
<td>Decimal Unit interrupted flag; a copy of the Mid-Instruction Interrupt Fault indicator</td>
<td></td>
</tr>
<tr>
<td>2,4,6 F</td>
<td>First time; data in Operand Descriptor 1,2,3 is valid</td>
<td></td>
</tr>
<tr>
<td>2,4,6 A</td>
<td>Operand Descriptor 1,2,3 is active</td>
<td></td>
</tr>
<tr>
<td>3 LEVEL</td>
<td>Difference in the count of characters loaded into the CPU and characters stored back to main store</td>
<td></td>
</tr>
<tr>
<td>3 D1 RES</td>
<td>Count of characters remaining in Operand Descriptor 1</td>
<td></td>
</tr>
<tr>
<td>4 D2 PTR</td>
<td>Address of last double word accessed by Operand Descriptor 2; bits 17-23 (bit address) valid only for initial access</td>
<td></td>
</tr>
<tr>
<td>4,6 x R</td>
<td>Last cycle performed must be repeated</td>
<td></td>
</tr>
<tr>
<td>5 D2 RES</td>
<td>Count of characters remaining in Operand Descriptor 2</td>
<td></td>
</tr>
<tr>
<td>6 D3 PTR</td>
<td>Address of the last double word accessed by Operand Descriptor 3; bits 17-23 (bit address) valid only for initial access</td>
<td></td>
</tr>
<tr>
<td>Word</td>
<td>Field Name</td>
<td>Meaning</td>
</tr>
<tr>
<td>------</td>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>6</td>
<td>JMP</td>
<td>Descriptor count; number of words to skip to find the next instruction following this multiword instruction</td>
</tr>
<tr>
<td>7</td>
<td>DJ RES</td>
<td>Count of characters remaining in Operand Descriptor 3</td>
</tr>
</tbody>
</table>
SECTION V

ADDRESSING -- SEGMENTATION AND PAGING

ADDRESSING MODES

The Multics Processor is able to access the main store in either of two modes; Absolute Mode or Append Mode.

The Processor prepares an Effective Address for each main store reference for instructions or operands. An Effective Address consists of a 12-bit segment number and an 18-bit offset within that segment. An offset is defined as the number of machine words from the segment base or origin to the referent. The Processor uses the Effective Address to generate a 24-bit final address. The final address is used either as a direct operand or as an address for a main store access. The various means of Effective Address formation are explained in Section VI, Effective Address Formation. The generation of the final address is different in the two Addressing Modes.

Absolute Mode

In Absolute Mode, the segment number is null, that is, undefined, and the segment base is the origin of main store. The final address is generated by high-order zero filling the offset with six binary 0's. Absolute Mode addressing is limited to the first 262,144 words of main store.

In Absolute Mode, all instruction fetches are made from Absolute addresses. Instruction operands may be located anywhere in main store and may be accessed by specifying ITS Address Modification for the instruction or by loading a Pointer Register with an appropriate value and specifying ITP Address Modification or using bit 29 of the instruction word. The use of ITS or ITP Address Modification in an Indirect Word will have the same effect.

WARNING: The use any of the above constructs in Absolute Mode places the Processor in Append Mode for one or more Address Preparation cycles. All necessary registers must be properly loaded and all Fault conditions must be considered (See Append Mode below).

If a transfer of control is made with any of the above constructs, the processor remains in Append Mode after the transfer and subsequent instruction fetches are made in Append Mode.
Although no segment is defined for Absolute Mode, it may be helpful to understand to visualize a virtual, unpaged segment overlaying the first 262,144 words of main store.

Append Mode

In Append mode, the appending mechanism is employed for all main store references. The appending mechanism is described in "Segmentation" and "Paging" following in this section.

SEGMENTATION

A Multics segment is defined as an array of machine words of arbitrary (but limited) size containing arbitrary data. A segment is identified within the Processor by a segment number (segno), unique to the segment for the process, that is assigned by the operating system when the segment is first referenced by the process.

To simplify this discussion, the operation of the hardware ring mechanism is not described although it is an integral part of Address Preparation. See Section VIII, Hardware Ring Implementation, for a discussion of the ring mechanism hardware.

An Effective Address in the Processor consists of a pair of integers (segno, offset). The range of segno is \((0, 2^{*12} - 1)\), the range of offset is \((0, 2^{*18} - 1)\). The description of the segment identified by segno value \(n\) is kept in the \(n\)th word-pair (offset\(=2 \cdot n\)) in a table known as the descriptor segment (dseg). The descriptor segment always has segno value 0 and contains descriptions of all segments accessible by the process including its own description in Y-pair 0. The location of the descriptor segment for a running process is held by the Processor in the Descriptor Segment Base Register (DSBR). (See Section IV, Program Accessible Registers) Each word-pair of a descriptor segment is known as a Segment Descriptor Word (SDW) and is 72 bits long. (See Figure 5-5, Segment Descriptor Word (SDW) Format, later in this section.)

A bit in the SDW for a segment (SDW\(, \text{u}\)) specifies whether the segment is paged or unpaged. The following is a simplified description of the appending process for unpaged segments. (Refer to Figures 4-14 and 5-5)

1. If \(2 \cdot \text{segno} \geq 16 \cdot (\text{DSBR.BND} + 1)\), then generate an Access Violation, Out of Segment Bounds Fault.

2. Fetch the SDW from DSBR.ADDR + 2 \cdot \text{segno}.

3. If SDW.F = "0", then generate Directed Fault \(n\) where \(n\) is given in SDW.FC. The value of \(n\) used here is the value assigned to define a missing segment fault or segment fault.

4. If offset \(\geq 16 \cdot (\text{SDW.BOUND} + 1)\), then generate an Access Violation, Out of Segment Bounds Fault.

5. If the access bits (SDW.R, SDW.E, etc.) of the segment are incompatible with the reference, generate the appropriate Access Violation Fault.
6. Generate final address SDW.ADR + offset.

Figure 5-1 depicts the relationships described above.

\[
\begin{array}{c}
\text{dseg} \\
T \quad 1 \quad \leftarrow \quad \text{DSBR.ADR} \\
\vdots \\
2 \times \text{sega} \\
\vdots \\
\text{segment} \\
\vdots \\
T \quad 1 \quad \leftarrow \quad \text{SDW} \quad \text{SDW(sega)} \\
\vdots \\
\text{offset} \\
\vdots \\
\text{data} \\
\vdots \\
\vdots \\
\vdots \\
1 \quad \leftarrow \quad \text{DSBR.BND} \\
\vdots \\
\vdots \\
\vdots \\
1 \quad \leftarrow \quad \text{SDW.BOUND}
\end{array}
\]

Figure 5-1 Final Address Generation for an Unpaged Segment

**PAGING**

A page is defined as a block of $2^p$ machine words. The Multics Processor is designed in such a way that $p$ is adjustable in the range (6,12). Experience has shown that the optimum value for $p$ is 10 yielding a page size of 1024 words.

With the value of $p$ established, the Processor divides a $k$-bit offset or sego value into two parts: the high order $(k-p)$ bits forming a page number, $x$, and the low order $p$ bits forming a word number, $y$. Algorithmically, this may stated as:

\[
x = \text{value} \mod (\text{page size})
\]

\[
y = (\text{value} - x) / (\text{page size})
\]

The symbols $x$ and $y$ will be used in this context throughout this section. Examples of page number formation are shown in Figure 5-2 below.
A bit in the SDW for a segment (SDW.U) specifies whether the segment is paged or unpaged. A paged segment may be defined as an array of pages of arbitrary (but limited) size with each page an array of 1024 machine words. Thus, a reference to a word or words of a paged segment may be treated as a reference to word \( x \) of page \( x \) of the segment.

Multics subdivides the Virtual Memory into page size blocks of 1024 words each. In the main store, the blocks are known as main store pages; on the paging device and the secondary storage, the blocks are known as records. Such a subdivision of space allows a segment page to be handled as a physical block independently from the other pages of the segment and from other segments. When a reference to a word in a paged segment is required (and the word is not already in main store), a main store page is allocated and the record containing the segment page is read in. Unneeded segment pages need not occupy space in main store.

The location and status of page \( x \) of a paged segment is kept in the \( x \)th word of a table known as the page table for the segment. The words in this table are known as Page Table Words (PTWs). (See Figure 5-6, Page Table Word (PTW) Format, later in this section.)

Any segment may be paged as appropriate and convenient. SDW.ADR for a paged segment points to the page table for the segment instead of the base of the segment. If \( \text{dsseg} \) for a process is paged, DSBR.ADD points to the page table for \( \text{dsseg} \).

The full algorithm used by the processor to access word \( \text{offset} \) of paged segment \( \text{segno} \) (including \( \text{dsseg} \) paging) is as follows. (Refer to Figures 4-14, 5-5, and 5-6)

1. If \( 2 \times \text{segno} \geq 16 \times \text{DSBR.BND} \), the generate an Access Violation, Out of Segment Bounds Fault.

2. Form the quantities:
   \[
   x_1 = (2 \times \text{segno}) \mod 1024 \\
   x_1 = \frac{(2 \times \text{segno} - x_1)}{1024}
   \]
Avoid using common keywords as names. When names such as

```
  goto declare del if then
```

and so on are used as names, a superficial but irritating confusion is introduced. On the other hand, use uncommon keywords as names where that is convenient. There is certainly no harm in using 'dft' to name a variable for the "debit final total" (or something of the sort) even though 'dft' is a keyword.

Where possible, avoid using troublesome letters in identifiers. For example, the digits zero and one are troublesome because some output devices do not clearly distinguish between zero and the letter 'O', or between one and the letter '1'.

**Literal Constants**

There is a literal constant lexeme for each type of arithmetic and string value. The full syntax and interpretation of these lexemes are given later in the section on "Expressions". The following is a representative set of examples of arithmetic literal constants:

<table>
<thead>
<tr>
<th>Arithmetic Constant</th>
<th>Data_Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.04</td>
<td>fixed dec(3)</td>
</tr>
<tr>
<td>3.04</td>
<td>fixed dec(3,2)</td>
</tr>
<tr>
<td>3.04e-5</td>
<td>float dec(3)</td>
</tr>
<tr>
<td>3.04e-51</td>
<td>complex float dec(3)</td>
</tr>
<tr>
<td>0110001b</td>
<td>fixed(7)</td>
</tr>
<tr>
<td>0110001b</td>
<td>fixed(7,4)</td>
</tr>
<tr>
<td>0110001e-2b</td>
<td>float(7)</td>
</tr>
<tr>
<td>0110001e-2bi</td>
<td>complex float(7)</td>
</tr>
</tbody>
</table>

Observe that an arithmetic constant does not begin with a sign. When a negative constant is required, it is written as two lexemes, a sign followed by an arithmetic constant.

The following is a representative set of examples of string literal constants:

<table>
<thead>
<tr>
<th>String Constant</th>
<th>Data_Type</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;abcd&quot;</td>
<td>char(4)</td>
<td>means &quot;abcdabcdabcd&quot;</td>
</tr>
<tr>
<td>(3)&quot;abcd&quot;</td>
<td>char(12)</td>
<td>means the null string</td>
</tr>
<tr>
<td>&quot;Hello,&quot;&quot; he said.&quot;</td>
<td>char(17)</td>
<td>&quot;&quot; counts as &quot; in value</td>
</tr>
<tr>
<td>&quot;11101&quot;b</td>
<td>bit(5)</td>
<td>means &quot;11101&quot;b</td>
</tr>
<tr>
<td>(4)&quot;01&quot;b</td>
<td>bit(4)</td>
<td>means &quot;01010101&quot;b</td>
</tr>
<tr>
<td>&quot;b&quot;</td>
<td>bit(0)</td>
<td>means the null string</td>
</tr>
</tbody>
</table>

Any ASCII character can be used in a 'character' string constant, including such non-printing characters as tab, newline, and so on. A string constant is a single lexeme and is not considered to contain smaller lexemes.
Punctuators

There are six punctuator lexemes; each is given, together with its purpose, in the following table:

<table>
<thead>
<tr>
<th>Punctuator</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>. (period)</td>
<td>indicates the decimal or binary point; also, separates names in a qualified reference</td>
</tr>
<tr>
<td>, (comma)</td>
<td>separates items in a list of arguments, parameters, subscripts, declarations, options, and so on</td>
</tr>
<tr>
<td>: (colon)</td>
<td>terminates a condition prefix or a label prefix; also, separates the bounds of an array</td>
</tr>
<tr>
<td>; (semicolon)</td>
<td>terminates a statement</td>
</tr>
<tr>
<td>( (left parenthesis)</td>
<td>indicates the beginning of a list, an expression, an iteration factor, and so on</td>
</tr>
<tr>
<td>) (right parenthesis)</td>
<td>indicates the end of a list, an expression, an iteration factor, and so on</td>
</tr>
</tbody>
</table>

These lexemes are used in most of the features of PL/I.

Operators

There are five kinds of operator lexemes; they are defined as follows:

<table>
<thead>
<tr>
<th>Classification</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>arithmetic</td>
<td>+ - * / **</td>
</tr>
<tr>
<td>relational</td>
<td>= -= &lt; &lt; &gt; &gt; &lt;= &gt;=</td>
</tr>
<tr>
<td>logical</td>
<td>= ! !=</td>
</tr>
<tr>
<td>string</td>
<td>#</td>
</tr>
<tr>
<td>qualifier</td>
<td>-&gt;</td>
</tr>
</tbody>
</table>

Most of the operators are defined in the section on "Operators". The only exception is the qualifier operator, which is defined in the section on "Expressions".
ADDRESS APPENDING

At the completion of the formation of the Effective Address (See Section VI, Effective Address Formation) an Effective Segment Number (segno) is in the Segment Number Register of the Temporary Pointer Register (TPR.SNR) and a Computed Address (offset) is in the Computed Address register of the Temporary Pointer Register (TPR.CA) (See Section IV, Program Accessible Registers, for a discussion of the Temporary Pointer Register).

Address Appending Sequences

Once segno and offset are formed in TPR.SNR and TPR.CA, respectively, the process of generating the final address can involve a number of different and distinct Appending Unit cycles.

The operation of the Appending Unit is shown in the flowchart in Figure 5-4. This flowchart assumes that Directed Faults Store Faults, or Parity Faults do not occur.

A segment boundary check is made in every cycle except PSDW. If a boundary violation is detected, an Access Violation, Out of Segment Bounds Fault will be generated and the execution of the instruction aborted. The occurrence of any Fault will abort the sequence at the point of occurrence. The operating system will save store the Control Unit Data for possible retry and will attempt to resolve the Fault condition.

The value of the Associative Memories may be seen in the flowchart by observing the number of cycles bypassed if an SDW or PTW is found in the Associative Memory.

There are nine different Appending Unit cycles that involve accesses to main store. Two of these (FANP, FAP) generate the final address and initiate a main store access for the operand or instruction pair; five (NSDW, PSDW, PTW, PTW2 and DSPTW) generate a main store access to fetch an SDW or PTW; and two (NSDPTW and HPTW) generate a main store access to update page status bits (PTW.U and PTW.M) in a PTW. The cycles are defined in Table 5-1 below.
<table>
<thead>
<tr>
<th>Cycle Name</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FANP</td>
<td>Final Address NonPaged</td>
<td>Generates the final address and initiates a main store access to an unpaged segment for operands or instructions.</td>
</tr>
<tr>
<td>FAP</td>
<td>Final Address Paged</td>
<td>Generates the final address and initiates a main store access to a paged segment for operands or instructions.</td>
</tr>
<tr>
<td>NSDW</td>
<td>Nonpaged SDW Fetch</td>
<td>Fetches an SDW from an unpaged dseg.</td>
</tr>
<tr>
<td>PSDW</td>
<td>Paged SDW Fetch</td>
<td>Fetches an SDW from a paged descriptor segment.</td>
</tr>
<tr>
<td>PTW</td>
<td>PTW Fetch</td>
<td>Fetches a PTW from a page table other than a dseg page table.</td>
</tr>
<tr>
<td>PTWZ</td>
<td>Second PTW Fetch (Same as PTW above)</td>
<td>Fetches the next PTW from a page table other than a dseg page table during hardware pre-paging for certain uninterruptable EIS instructions. This cycle does not load the next PTW into the Appending Unit. It merely assures that the PTW is not faulted (PTW.F = &quot;1&quot;) and that the target page will be in main store when and if needed by the instruction.</td>
</tr>
<tr>
<td>DSPTW</td>
<td>Descriptor Segment PTW Fetch</td>
<td>Fetches a PTW from a dseg page table.</td>
</tr>
<tr>
<td>MDSPTW</td>
<td>Modify DSPTW</td>
<td>Sets the page accessed bit (PTW.U) in the PTW for a page in a dseg page table. This cycle always immediately follows a DSPTW cycle.</td>
</tr>
<tr>
<td>MPTW</td>
<td>Modify PTW</td>
<td>Sets the page modified bit (PTW.M) in the PTW for a page in other than a dseg page table.</td>
</tr>
</tbody>
</table>

Table 5-1 Appending Unit Cycle Definitions
Note: A STR-OP is any Processor function that writes data to main store.

Figure 5-4  Appending Unit Operation Flowchart
The Segment Descriptor Word (SDW) pair contains information necessary to control the access to a segment by a process. The SDW for a segment is constructed from data in the directory entry for the segment and in the System Segment Table (SST) when the segment is initiated by the process. The SDW for segment D (unique within the process) is placed at offset 20 in the Descriptor Segment (desg) of the process.

**Even Word**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDR</td>
<td>24 bit base address of segment (U=1) or segment page table (U=0).</td>
</tr>
<tr>
<td>R1</td>
<td>highest effective read/write ring.</td>
</tr>
<tr>
<td>R2</td>
<td>highest effective read/execute ring.</td>
</tr>
<tr>
<td>R3</td>
<td>highest effective call ring.</td>
</tr>
<tr>
<td>F</td>
<td>directed fault indicator.</td>
</tr>
<tr>
<td>$I$</td>
<td>$I$ = the necessary unpaged segment or segment page table is in memory.</td>
</tr>
</tbody>
</table>

**Odd Word**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOUND</td>
<td>largest 16-word block number that may be accessed without causing an Access Violation, Out of Segment Bounds Fault.</td>
</tr>
<tr>
<td>R</td>
<td>read permission bit.</td>
</tr>
</tbody>
</table>

Figure 5-5  Segment Descriptor Word (SDW) Format

---

**APPENDIX UNIT DATA WORD FORMATS**

**Segment Descriptor Word (SDW) Format**

The Segment Descriptor Word (SDW) pair contains information necessary to control the access to a segment by a process. The SDW for a segment is constructed from data in the directory entry for the segment and in the System Segment Table (SST) when the segment is initiated by the process. The SDW for segment D (unique within the process) is placed at offset 20 in the Descriptor Segment (desg) of the process.

**Even Word**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDR</td>
<td>24 bit base address of segment (U=1) or segment page table (U=0).</td>
</tr>
<tr>
<td>R1</td>
<td>highest effective read/write ring.</td>
</tr>
<tr>
<td>R2</td>
<td>highest effective read/execute ring.</td>
</tr>
<tr>
<td>R3</td>
<td>highest effective call ring.</td>
</tr>
<tr>
<td>F</td>
<td>directed fault indicator.</td>
</tr>
<tr>
<td>$I$</td>
<td>$I$ = the necessary unpaged segment or segment page table is in memory.</td>
</tr>
</tbody>
</table>

**Odd Word**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOUND</td>
<td>largest 16-word block number that may be accessed without causing an Access Violation, Out of Segment Bounds Fault.</td>
</tr>
<tr>
<td>R</td>
<td>read permission bit.</td>
</tr>
</tbody>
</table>
**Field Name** | **Description**
---|---
E | execute permission bit. (XEC & XED excluded)
W | write permission bit.
P | privileged mode bit.
0 = privileged instructions cannot be executed.
1 = privileged instructions may be executed if in ring 0.
U | paged/unpaged bit.
0 = segment is paged and ADDR is the address of the page table.
1 = segment is unpaged and ADDR is the base address of the segment.
G | gate indicator bit.
0 = any call from an external segment must be to an offset less than the value of CL.
1 = any legal segment offset may be called.
C | cache control bit.
0 = words (operands or instructions) from this segment may not be placed in the cache.
1 = words from this segment may be placed in the cache.
CL | call limiter.
Any external call to this segment must be to an offset less than CL if G=0.

---

**Page Table Word (PTW) Format**

The Page Table Word (PTW) contains location and status information for a page of a paged segment. The PTWs for a paged segment are copied from the directory entry file map for the segment into the Page Table Word Array (PTWA) of a free area in the Active Segment Table (AST) area of the SST when the segment is first initiated by a process. Subsequent initiations by other processes reference the existing PTWA.

| Field Name | Description |
---|---|
ADDR | 18 bit modulo 64 page address if page is in store,
| 18 bit record number of page if page is not in store.
The hardware ignores low order bits of the in-store page address according to page size based on the following ...
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DID</td>
<td>device id for device containing the page.</td>
</tr>
<tr>
<td>W</td>
<td>1 = page has not yet been written out.</td>
</tr>
<tr>
<td>P</td>
<td>temporary bit used in post-processing.</td>
</tr>
<tr>
<td>U</td>
<td>1 = page has been used (touched).</td>
</tr>
<tr>
<td>M</td>
<td>1 = page has been modified.</td>
</tr>
<tr>
<td>Q</td>
<td>1 = page has been used during the quantum.</td>
</tr>
<tr>
<td>W</td>
<td>1 = page is wired.</td>
</tr>
<tr>
<td>S</td>
<td>1 = page is out of service (I/O in progress).</td>
</tr>
<tr>
<td>F</td>
<td>1 = page is in store. 0 = page not in store. Execute directed fault FC.</td>
</tr>
<tr>
<td>FC</td>
<td>directed fault number for page fault.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page Size in_words</th>
<th>ADDR Bits Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>none</td>
</tr>
<tr>
<td>128</td>
<td>17</td>
</tr>
<tr>
<td>256</td>
<td>16-17</td>
</tr>
<tr>
<td>512</td>
<td>15-17</td>
</tr>
<tr>
<td>1024</td>
<td>14-17</td>
</tr>
<tr>
<td>2048</td>
<td>13-17</td>
</tr>
<tr>
<td>4096</td>
<td>12-17</td>
</tr>
</tbody>
</table>

Page Size
126 512 1024 2048 4096

Address Bits

for device containing the page.
SECTION VI

EFFECTIVE ADDRESS FORMATION

DEFINITION OF EFFECTIVE ADDRESS

The Effective Address in the Multics Processor is the user's specification of the location of a data item in the Multics Virtual Memory. Each reference to the Virtual Memory for operands, indirect words, indirect pointers, Operand Descriptors, or instructions must provide an Effective Address. The hardware and the operating system translate the Effective Address into the true location of the data item and assure that the data item is in main store for the reference.

The Effective Address consists of two parts, a segment number and an offset. The value of each part is the result of the evaluation of a hardware algorithm (expression) of one or more terms. The selection of the algorithm is made by the use of control bits in the Instruction Word; namely, bit 29 for segment number modification and the Address Modification (or TAG) field for offset modification. If the TAG field of the Instruction Word specifies certain "indirect" modifications, the TAG field of the Indirect Word is also treated as an Address Modifier, thus establishing a continuing "indirect chain". Bit 29 of an Indirect Word has no meaning in the context of Address Modification.

The results of evaluation of the Address Modification algorithm are stored in temporary registers used as working registers by the Processor. The segment number is stored in the Temporary Segment Register (TPR.TSR). The offset is stored in the Computed Address Register (TPR.CA). When each Effective Address computation has been completed, the C(TPR.TSR) and the C(TPR.CA) are presented to the Appending Unit for translation to a 24-bit final Address (See Section V, Addressing -- Segmentation and Paging).

TYPES OF EFFECTIVE ADDRESS FORMATION

There are two types of Effective Address formation. The first type does not make explicit use of segment numbers. The algorithm selected produces a value for C(TPR.CA) only. The segment number in C(TPR.TSR) does not change and is the segment number used to fetch the instruction. In this case, all references are said to be "local" to the procedure segment as held in C(PPR.PSR).

The second type makes use of a segment number stored either in an Indirect Word-pair in main store or in a Pointer Register (PRn). The algorithm selected produces values for both C(TPR.TSRk) and C(TPR.CA). The segment number in C(TPR.TSR) may change and, if it changes, references are said to be "external" to the procedure segment as held in C(PPR.PSR).
The two types of Effective Address formation can be intermixed. In cases where Effective Address calculations are chained together through Pointer Registers or Indirect Words, each Effective Address is translated to a 24-bit final address to fetch the next item in the chain.

**EFFECTIVE ADDRESS FORMATION DESCRIPTION**

This description of Effective Address formation is divided into two parts corresponding to the two types. The first part describes the type that involves only the offset value \( CT_{PR}.CA \). The segment number \( CT_{PR}.TSR \) is assumed constant and equal to \( CP_{PR}.PSR \).

The second part describes the type that involves both the segment number \( CT_{PR}.TSR \) and the offset \( CT_{PR}.CA \).

**EFFECTIVE ADDRESS FORMATION INVOLVING OFFSET ONLY**

The Address Modifications described here produce values for \( CT_{PR}.CA \) only. The segment number \( CT_{PR}.TSR \) is assumed constant and equal to \( CP_{PR}.PSR \).

The Address Modifier (TAG) Field

Bits 30-35 of an Instruction Word or Indirect Word constitute the Address Modifier or TAG field. The format of the TAG field is:

\[
\begin{array}{ccccccc}
3 & 3 & 3 & 3 & 0 & 1 & 2 & 5 \\
1 & 1 & 1 & 0 & 1 & 2 & 4 \\
\end{array}
\]

Figure 6-1 Address Modifier (TAG) Field Format

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Tm )</td>
<td>The &quot;modifier&quot; field specifies one of four general types of offset modification.</td>
</tr>
<tr>
<td>( Td )</td>
<td>The &quot;designator&quot; field specifies a register number or an Indirect Then Tally variation.</td>
</tr>
</tbody>
</table>

**General Types of Offset Modification**

There are four general types of offset modification: Register, Register Then Indirect, Indirect Then Register, and Indirect Then Tally. The general types are described in Table 6-1 below.
Each Effective Address formation for an operand begins with a preliminary step of loading TPR.CA with the ADDRESS field of the Instruction Word. This preliminary step takes place during instruction decode. The value loaded into TPR.CA is symbolized by "y" in the descriptions following.

Table 6-1 General Offset Modification Types

<table>
<thead>
<tr>
<th>Tm Value</th>
<th>Modifier Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Register (R)</td>
<td>The contents of the designated register, Td, are added to the current Computed Address to form the modified Computed Address. Addition is two's complement, modulo $2^{18}$ and overflow is not possible.</td>
</tr>
<tr>
<td>1</td>
<td>Register Then Indirect (RI)</td>
<td>The contents of the designated register, Td, are added to the current Computed Address to form the modified Computed Address as for Register modification. The word at CITPR.CA is then fetched and interpreted as an Indirect Word. The TAG field of the Indirect Word specifies the next step in Effective Address formation. The use of du or dl as the designator in this modification type will cause an Illegal Procedure, Illegal Modifier Fault.</td>
</tr>
<tr>
<td>2</td>
<td>Indirect Then Tally (IT)</td>
<td>The Indirect Word at CITPR.CA is fetched and the modification performed according to the variation specified in Td and the contents of the Indirect Word. This modification type allows automatic incrementing and decrementing of addresses and tally counting.</td>
</tr>
<tr>
<td>3</td>
<td>Indirect Then Register (IR)</td>
<td>The register designator, Td, is safe-stored in a special holding register (CT-HOLD). The word at the current CITPR.CA is fetched and interpreted as an Indirect Word. The TAG field of the Indirect Word specifies the next step in Effective Address formation as follows.</td>
</tr>
</tbody>
</table>

If Indirect TAG is 1 then

- R or IT Perform Register modification using Td from CT-HOLD.
- RI Perform the Register Then Indirect modification immediately and fetch the next Indirect Word from the result of that modification.

IR Replace the safe-stored Td value in CT-HOLD with the Td value of the Indirect Word TAG field and fetch the next Indirect Word from the ADDRESS given in the Indirect Word.
Effective Address Formation Flowcharts

The algorithmic flowcharts depicting the Effective Address formation process are scattered throughout this section and are linked together with "Go to" labels. The flowchart starts with Figure 6-2 below.

START EA

\[ \text{Interpret Tm} \]

\[ \text{Go to START} \]

\[ \text{Go to START} \]

\[ \text{Go to START} \]

\[ \text{Go to START} \]

(Figure 6-3) (Figure 6-4) (Figure 6-5) (Figure 6-6)

Figure 6-2 Common Effective Address Formation Flowchart

Register (R) Modification

In Register modification \((Tm = 0)\) the value of \(T_d\) designates a register whose contents are to be added to \(C(TPR, CA)\) to form a modified \(C(TPR, CA)\). This modified \(C(TPR, CA)\) becomes the Effective Address of the operand. See Table 6-2 and Figure 6-3 below for details.

EXAMPLES:

<table>
<thead>
<tr>
<th>Label</th>
<th>Instruction</th>
<th>Effective Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.  a</td>
<td>lda y</td>
<td>y</td>
</tr>
<tr>
<td>2.  a</td>
<td>sta y,n</td>
<td>y</td>
</tr>
<tr>
<td>3.  a</td>
<td>ldaq y,a</td>
<td>y + C(A)0,17</td>
</tr>
<tr>
<td>4.  a</td>
<td>tra 3,ic</td>
<td>a + 3</td>
</tr>
<tr>
<td>5.  a</td>
<td>ldq y,du</td>
<td>y; operand has the form zero y,0</td>
</tr>
<tr>
<td>6.  a</td>
<td>lxl4 y,dl</td>
<td>y; operand has the form zero 0,y</td>
</tr>
<tr>
<td>7.  a</td>
<td>mpv y,1</td>
<td>y + C(X1)</td>
</tr>
<tr>
<td>8.  a</td>
<td>stx4 y,7</td>
<td>y + C(X7)</td>
</tr>
<tr>
<td>Td Value</td>
<td>Register Selected</td>
<td>Coding</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>--------</td>
</tr>
<tr>
<td>0</td>
<td>none</td>
<td>n or null</td>
</tr>
<tr>
<td>1</td>
<td>A0,17</td>
<td>au</td>
</tr>
<tr>
<td>2</td>
<td>Q0,17</td>
<td>qu</td>
</tr>
<tr>
<td>3</td>
<td>none</td>
<td>du</td>
</tr>
<tr>
<td>4</td>
<td>PPR,IC</td>
<td>lc</td>
</tr>
<tr>
<td>5</td>
<td>A18,35</td>
<td>al</td>
</tr>
<tr>
<td>6</td>
<td>Q18,35</td>
<td>ql</td>
</tr>
<tr>
<td>7</td>
<td>none</td>
<td>dl</td>
</tr>
<tr>
<td>10</td>
<td>X0</td>
<td>0 or x0</td>
</tr>
<tr>
<td>11</td>
<td>X1</td>
<td>1 or x1</td>
</tr>
<tr>
<td>12</td>
<td>X2</td>
<td>2 or x2</td>
</tr>
<tr>
<td>13</td>
<td>X3</td>
<td>3 or x3</td>
</tr>
<tr>
<td>14</td>
<td>X4</td>
<td>4 or x4</td>
</tr>
<tr>
<td>15</td>
<td>X5</td>
<td>5 or x5</td>
</tr>
<tr>
<td>16</td>
<td>X6</td>
<td>6 or x6</td>
</tr>
<tr>
<td>17</td>
<td>X7</td>
<td>7 or x7</td>
</tr>
</tbody>
</table>
Register Then Indirect (RI) Modification

In Register Then Indirect modification ($Tm = 1$) the value of $Td$ designates a register whose contents are to be added to $C(TPR, CA)$ to form a modified $C(TPR, CA)$. This modified $C(TPR, CA)$ is used as an Effective Address to fetch an Indirect Word. The ADDRESS field of the Indirect Word is loaded into TPR, CA and the TAG field field of the Indirect Word is interpreted in the next step of an indirect chain. The TALLY field of the Indirect Word is ignored.

The indirect chain continues until an Indirect Word TAG field specifies a modification without indirection, namely, a Register modification.

The coding mnemonic for Register Then Indirect modification is $c^* \text{ where } c$ is any of the coding mnemonics for Register modification as given in Table 6-2 above except $du$ and $dl$. The $du$ and $dl$ register codes are illegal and will cause an Illegal Procedure, Illegal Modifier fault. See flowchart in Figure 6-4 below.
EXAMPLES

<table>
<thead>
<tr>
<th>Label</th>
<th>Instruction</th>
<th>Effective Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. a</td>
<td>lds b,*</td>
<td>y</td>
</tr>
<tr>
<td>b</td>
<td>arg y</td>
<td></td>
</tr>
<tr>
<td>2. a</td>
<td>ldq b,1,*</td>
<td>y + C(A)0,17</td>
</tr>
<tr>
<td>b+C(X1)</td>
<td>arg y,au</td>
<td></td>
</tr>
<tr>
<td>3. a</td>
<td>tra 4,ic,*</td>
<td></td>
</tr>
<tr>
<td>a+4</td>
<td>arg c,*</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>arg y</td>
<td>y</td>
</tr>
<tr>
<td>4. a</td>
<td>lxi b,0,*</td>
<td></td>
</tr>
<tr>
<td>b+C(X0)</td>
<td>arg c,1,*</td>
<td>y; operand has the form</td>
</tr>
<tr>
<td>c+C(X1)</td>
<td>arg y,d1</td>
<td>zero 0,y</td>
</tr>
</tbody>
</table>

START
RI MODE

1. Td=3
2. Td=0?

START EA
(Figure 6-2)

Figure 6-4 Register Then Indirect Modification Flowchart
**Indirect Then Register (IR) Modification**

In Indirect Then Register modification (Tm = 3) the value of Td designates a register whose contents are to be added to CTPR.CA to form the final modified C(TPR.CA) during the last step in the indirect chain. The value of Td is safe-stored in a special holding register, CT-HOLD. The initial C(TPR.CA) is used as Effective Address to fetch an Indirect Word. The ADDRESS field of the Indirect Word is loaded into TPR.CA and the TAG field field of the Indirect Word is interpreted in the next step of an indirect chain. The TALLY field of the Indirect Word is ignored.

If the Indirect Word TAG field specifies a Register Then Indirect modification, that modification is performed and the indirect chain continues.

If the Indirect Word TAG field specifies Indirect Then Register modification, the Td value from that TAG field replaces the safe-stored Td value in CT-HOLD and the indirect chain continues.

If the Indirect Word TAG specifies Register or Indirect Then Tally modification, that modification is replaced with a Register modification using the Td value safe-stored in CT-HOLD and the indirect chain ends.

The coding mnemonic for Indirect Then Register modification is *C where C is any of the coding mnemonics for Register modification as given in Table 6-2 above except null.

**EXAMPLES:**

<table>
<thead>
<tr>
<th>Label</th>
<th>Instruction</th>
<th>Effective Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. a</td>
<td>lda b,*n</td>
<td>(CT-HOLD = n)</td>
</tr>
<tr>
<td>b</td>
<td>arg y,2</td>
<td>y</td>
</tr>
<tr>
<td>2. a</td>
<td>lxl2 b,*d1</td>
<td>(CT-HOLD = d1)</td>
</tr>
<tr>
<td>b</td>
<td>sta y,au</td>
<td>y; operand has the form zero 0,y</td>
</tr>
<tr>
<td>3. a</td>
<td>lda b,*l</td>
<td>(CT-HOLD = x1)</td>
</tr>
<tr>
<td>b</td>
<td>arg c,n*</td>
<td>(CT-HOLD = x4)</td>
</tr>
<tr>
<td>c</td>
<td>arg d,4</td>
<td>y + C(x4)</td>
</tr>
<tr>
<td>d</td>
<td>arg y,dl</td>
<td>a + 5</td>
</tr>
<tr>
<td>4. a</td>
<td>ldx0 b,1*</td>
<td>(CT-HOLD = ic)</td>
</tr>
<tr>
<td>b+C(x1)</td>
<td>arg c,ic</td>
<td>a + 5</td>
</tr>
<tr>
<td>c</td>
<td>arg 5,dl</td>
<td></td>
</tr>
</tbody>
</table>
Indirect Then Tally (IT) Modification

In Indirect Then Tally modification (Tm = 2) the value of Td specifies a variation. The initial C(TPR,CA) is used as Effective Address to fetch an Indirect Word. The Indirect Word is interpreted and possibly altered as the modification is performed.

The TALLY field of the Indirect Word is used to count references made to the Indirect Word. It has a maximum range of 4095. If the TALLY field has the value 0 after a reference to the Indirect Word, the Tally Runout indicator will be set ON, otherwise the Tally Runout indicator will be set OFF. The value of the TALLY field and the state of the Tally Runout indicator have no effect on Effective Address formation.

WARNING: If there is more than one Indirect Word in an indirect chain that is referenced by a tally counting modification, only the state of the TALLY field of the last such word will be reflected in the Tally Runout indicator.
The variations of the Indirect Then Tally modification are given in Table 6-3 below and explained in detail in the paragraphs following. See flowchart in figure 6-6. Those entries given as "Undefined" cause an Illegal Procedure, Illegal Modifier Fault. (See "Effective Address Formation Involving Both Segment Number and Offset" later in this section for certain special cases.)

<table>
<thead>
<tr>
<th>Td Value</th>
<th>Coding Mnemonic</th>
<th>Variation Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>f1</td>
<td>Fault Tag 1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Undefined</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Undefined</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Undefined</td>
</tr>
<tr>
<td>4</td>
<td>sd</td>
<td>Subtract Delta</td>
</tr>
<tr>
<td>5</td>
<td>scr</td>
<td>Sequence Character Reverse</td>
</tr>
<tr>
<td>6</td>
<td>f2</td>
<td>Fault Tag 2</td>
</tr>
<tr>
<td>7</td>
<td>f3</td>
<td>Fault Tag 3</td>
</tr>
<tr>
<td>10</td>
<td>ci</td>
<td>Character Indirect</td>
</tr>
<tr>
<td>11</td>
<td>i</td>
<td>Indirect</td>
</tr>
<tr>
<td>12</td>
<td>sc</td>
<td>Sequence Character</td>
</tr>
<tr>
<td>13</td>
<td>ad</td>
<td>Add Delta</td>
</tr>
<tr>
<td>14</td>
<td>di</td>
<td>Decrement Address, Increment Tally</td>
</tr>
<tr>
<td>15</td>
<td>dic</td>
<td>Decrement Address, Increment Tally, and Continue</td>
</tr>
<tr>
<td>16</td>
<td>id</td>
<td>Increment Address, Decrement Tally</td>
</tr>
<tr>
<td>17</td>
<td>idc</td>
<td>Increment Address, Decrement Tally, and Continue</td>
</tr>
</tbody>
</table>

**Fault Tag 1 (Td = 0)**

Effective Address formation is terminated immediately and a Fault Tag 1 Fault is generated. A Fault Tag 1 Fault executes the Fault Trap pair at C + 6 where the value of C is obtained from the FAULT BASE switches on the Processor Configuration panel.

This variation may be used in Indirect Word or program control transfer vectors or tree structures to signal invalid entries or entries that require special handling. C(TPR,CA) at the time of the fault contains the Effective Address of the word containing the Fault Tag 1 modification. Thus, the ADDRESS and TALLY fields of that word may contain information relative to recovery from the fault.
**Subtract Delta (Td = 4)**

The TAG field of the Indirect Word is interpreted as a 6-bit, unsigned, positive address increment value, \( \delta \). For each reference to the Indirect Word, the ADDRESS field is reduced by \( \delta \) and the TALLY field is increased by 1 before the Effective Address is formed. ADDRESS arithmetic is modulo \( 2^{18} \). TALLY arithmetic is modulo \( 4096 \). If the TALLY field overflows to 0, the Tally Runout indicator is set ON, otherwise it is set OFF. The Effective Address is the value of the modified ADDRESS field.

**EXAMPLE:**

<table>
<thead>
<tr>
<th>Label</th>
<th>Instruction</th>
<th>Reference Effective Tally</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>lda b,ad</td>
<td>c+1</td>
</tr>
<tr>
<td>b</td>
<td>vfd 18/kg,12/t,6/d</td>
<td>c-2d, t+2</td>
</tr>
<tr>
<td>c</td>
<td></td>
<td>c-3d, t+3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c-nd, t+n</td>
</tr>
</tbody>
</table>

**Sequence Character Reverse (Td = 5)**

Bit 30 of the TAG field of the Indirect Word is interpreted as a character size flag, \( \text{ch} \), with the value 0 indicating 6-bit characters and the value 1 indicating 9-bit characters. Bits 33-35 of the TAG field are interpreted as a 3-bit character position counter, \( \text{cl} \). Bits 31-32 of the TAG field must be zero.

For each reference to the Indirect Word, the character counter, \( \text{cl} \), is reduced by 1 and the TALLY field is increased by 1 before the Effective Address is formed. Character count arithmetic is modulo 6 for 6-bit characters and modulo 4 for 9-bit characters. If the character count, \( \text{cl} \), underflows to -1, it is reset to 5 for 6-bit characters or to 3 for 9-bit characters and ADDRESS is reduced by 1. ADDRESS arithmetic is modulo \( 2^{18} \). TALLY arithmetic is modulo \( 4096 \). If the TALLY field overflows to 0, the Tally Runout indicator is set ON, otherwise it is set OFF. The Effective Address is the modified value of the ADDRESS field.

A 36-bit operand is formed by high-order zero filling the value of character \( \text{cl} \) of ADDRESS with an appropriate number of bits.

**EXAMPLES:**

<table>
<thead>
<tr>
<th>Label</th>
<th>Instruction</th>
<th>Reference Effective Tally</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>lda b,scr</td>
<td>1 2 c+1 t+1 00...0&quot;m&quot;</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>vfd 18/kg,12/t,1/0,5/2</td>
<td>2 1 c+1 t+2 00...0&quot;n&quot;</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>bci &quot;abcdefghijk...&quot;</td>
<td>3 0 c+1 t+3 00...0&quot;c&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 5 c t+4 00...0&quot;f&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 4 c t+5 00...0&quot;e&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>lda b,scr</td>
<td>1 2 c+1 t+1 00...0&quot;g&quot;</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>vfd 18/kg,12/t,1/1,5/2</td>
<td>2 1 c+1 t+2 00...0&quot;n&quot;</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>aci &quot;abcdefghijk...&quot;</td>
<td>3 0 c+1 t+3 00...0&quot;e&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 3 c t+4 00...0&quot;d&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 2 c t+5 00...0&quot;c&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Fault Tag 2 (Td = 6)

The action for this variation is identical to that for Fault Tag 1 except that the Trap Pair at C + 60 (octal) is executed.

WARNING: Fault Tag 2 is reserved to the Multics operating system for use in the Dynamic Linking feature. Its attempted use for other purposes could cause serious system inconsistencies and/or system crashes.

Fault Tag 3 (Td = 7)

The action for this variation is identical to that for Fault Tag 1 except that the Trap Pair at C + 62 (octal) is executed.

Character Indirect (Td = 10)

Bit 30 of the TAG field of the Indirect Word is interpreted as a character size flag, #b, with the value 0 indicating 6-bit characters and the value 1 indicating 9-bit characters. Bits 33-35 of the TAG field are interpreted as a 3-bit character position value, ci. Bits 31-32 of the TAG field must be zero.

If the character position value is greater than 5 for 6-bit characters or greater than 3 for 9-bit characters, an Illegal Procedure, Illegal Modifier Fault will occur. The TALLY field is ignored. The Effective Address is the value of the ADDRESS field.

A 36-bit operand is formed by high-order zero filling the value of character ci of ADDRESS with an appropriate number of bits.

EXAMPLES:

<table>
<thead>
<tr>
<th>Label</th>
<th>Instruction</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>lda b,ci</td>
<td>00...0&quot;I&quot;</td>
</tr>
<tr>
<td>b</td>
<td>vfd 18/c+1,12/0,1/0,5/2</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>bci &quot;ABCDEFHIJKLMNOP&quot;</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>lda d,ci</td>
<td>00...0&quot;B&quot;</td>
</tr>
<tr>
<td>e</td>
<td>vfd 18/f,12/0,1/0,5/3</td>
<td>00...0&quot;d&quot;</td>
</tr>
<tr>
<td>f</td>
<td>aci &quot;abcdefg&quot;</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>lda a,ci</td>
<td>00...0&quot;e&quot;</td>
</tr>
<tr>
<td>vfd 18/f+1,12/0,1/0,5/0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Indirect (Td = 11)

The Effective Address is the value of the ADDRESS field. The TALLY and TAG fields are ignored.

Sequence Character (Td = 12)

Bit 30 of the TAG field of the Indirect Word is interpreted as a character size flag, #b, with the value 0 indicating 6-bit characters and the value 1 indicating 9-bit characters. Bits 33-35 of the TAG field are interpreted as a 3-bit character position counter, ci. Bits 31-32 of the TAG field must be zero.
For each reference to the Indirect Word, the character counter, \( c_i \), is increased by 1 and the TALLY field is reduced by 1 after the Effective Address is formed. Character count arithmetic is modulo 6 for 6-bit characters and modulo 4 for 9-bit characters. If the character count, \( c_i \), overflows to 6 for 6-bit characters or to 4 for 9-bit characters, it is reset to 0 and ADDRESS is increased by 1. ADDRESS arithmetic is modulo \( 2^{**18} \). TALLY arithmetic is modulo 4096. If the TALLY field is reduced to 0, the Tally Runout indicator is set ON, otherwise it is set OFF. The Effective Address is the original unmodified value of the ADDRESS field.

A 36-bit operand is formed by high-order zero filling the value of character \( c_i \) of ADDRESS with an appropriate number of bits.

**EXAMPLES**

<table>
<thead>
<tr>
<th>Label</th>
<th>Instruction</th>
<th>Reference Count</th>
<th>c_i</th>
<th>Effective Address</th>
<th>Tally Value</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>lda b,sc</td>
<td>1</td>
<td>4</td>
<td>c</td>
<td>t-1</td>
<td>00...0&quot;E&quot;</td>
</tr>
<tr>
<td>b</td>
<td>vfd 15/c,12/t,1/0,5/4</td>
<td>2</td>
<td>5</td>
<td>c</td>
<td>t-2</td>
<td>00...0&quot;F&quot;</td>
</tr>
<tr>
<td>c</td>
<td>bci &quot;ABCDEF&quot;</td>
<td>3</td>
<td>0</td>
<td>c+i</td>
<td>t-3</td>
<td>00...0&quot;G&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>1</td>
<td>c+i</td>
<td>t-4</td>
<td>00...0&quot;H&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>2</td>
<td>c+i</td>
<td>t-5</td>
<td>00...0&quot;I&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>lda b,sc</td>
<td>1</td>
<td>2</td>
<td>c</td>
<td>t-1</td>
<td>00...0&quot;C&quot;</td>
</tr>
<tr>
<td>b</td>
<td>vfd 15/c,12/t,1/1,5/2</td>
<td>2</td>
<td>3</td>
<td>c</td>
<td>t-2</td>
<td>00...0&quot;D&quot;</td>
</tr>
<tr>
<td>c</td>
<td>aci &quot;abcdefgh&quot;</td>
<td>3</td>
<td>0</td>
<td>c+i</td>
<td>t-3</td>
<td>00...0&quot;E&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>1</td>
<td>c+i</td>
<td>t-4</td>
<td>00...0&quot;F&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>2</td>
<td>g+i</td>
<td>t-5</td>
<td>00...0&quot;G&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Add Delta (Td = 13)

The TALLY field of the Indirect Word is interpreted as a 6-bit, unsigned, positive address increment value, \( \delta \). For each reference to the Indirect Word, the ADDRESS field is increased by \( \delta \) and the TALLY field is reduced by 1 after the Effective Address is formed. ADDRESS arithmetic is modulo \( 2^{**18} \). TALLY arithmetic is modulo 4096. If the TALLY field is reduced to 0, the Tally Runout indicator is set ON, otherwise it is set OFF. The Effective Address is the value of the original unmodified ADDRESS field.

**EXAMPLES**

<table>
<thead>
<tr>
<th>Label</th>
<th>Instruction</th>
<th>Reference Count</th>
<th>Address</th>
<th>Tally Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>lda b,ad</td>
<td>1</td>
<td>c</td>
<td>t-1</td>
</tr>
<tr>
<td>b</td>
<td>vfd 18/c,1/t,5/d</td>
<td>2</td>
<td>c-d</td>
<td>t-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>c-2d</td>
<td>t-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Decrement Address, Increment Tally (Td = 14)

For each reference to the Indirect Word, the ADDRESS field is reduced by 1 and the TALLY field is increased by 1 after the Effective Address is formed. ADDRESS arithmetic is modulo \( 2^{**18} \). TALLY arithmetic is modulo 4096. If the TALLY field overflows to 0, the Tally Runout indicator is set ON, otherwise it is set OFF. The TALLY field
field of the Indirect Word is ignored. The Effective Address is the value of the modified ADDRESS field.

**Example:**

<table>
<thead>
<tr>
<th>Label</th>
<th>Instruction</th>
<th>Reference Count</th>
<th>Effective Address</th>
<th>Tally Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>lda b,di</td>
<td>1</td>
<td>c-1</td>
<td>t+1</td>
</tr>
<tr>
<td>b</td>
<td>vfd 18/c,12/t</td>
<td>2</td>
<td>c-2</td>
<td>t+2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>c-3</td>
<td>t+3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>***</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>c-0</td>
<td>t+n</td>
</tr>
</tbody>
</table>

Decrement Address, Increment Tally, and Continue (Td = 15)

The action for this variation is identical to that for the Decrement Address, Increment Tally variation except that the TAG field of the Indirect Word is interpreted and continuation of the indirect chain is possible. If the TAG of the Indirect Word invokes a register, that is, specifies R, RI, or IR modification, the effective Td value for the register is forced to "null" before the next Effective Address is formed.

Increment Address, Decrement Tally (Td = 15)

For each reference to the Indirect Word, the ADDRESS field is increased by 1 and the TALLY field is reduced by 1 after the Effective Address is formed. ADDRESS arithmetic is modulo 2**18, TALLY arithmetic is modulo 4096. If the TALLY field is reduced to 0, the Tally Runout indicator is set ON, otherwise it is set OFF. The TAG field of the Indirect Word is ignored. The Effective Address is the value of the original unmodified ADDRESS field.

**Example:**

<table>
<thead>
<tr>
<th>Label</th>
<th>Instruction</th>
<th>Reference Count</th>
<th>Effective Address</th>
<th>Tally Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>lda b,di</td>
<td>1</td>
<td>c</td>
<td>t-1</td>
</tr>
<tr>
<td>b</td>
<td>vfd 18/c,12/t</td>
<td>2</td>
<td>c-1</td>
<td>t-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>c-2</td>
<td>t-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>***</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>c-(n-1)</td>
<td>t-n</td>
</tr>
</tbody>
</table>

Increment Address, Decrement Tally, and Continue (Td = 17)

The action for this variation is identical to that for the Increment Address, Decrement Tally variation except that the TAG field of the Indirect Word is interpreted and continuation of the indirect chain is possible. If the TAG of the Indirect Word invokes a register, that is, specifies R, RI, or IR modification, the effective Td value for the register is forced to "null" before the next Effective Address is formed.
Figure 6-6 Indirect Then Tally Modification Flowchart

EFFECTIVE ADDRESS FORMATION INVOLVING BOTH SEGMENT NUMBER AND OFFSET

The second type of Address Formation allows formation of a modified Segment Number and a modified Offset simultaneously. See Figure 6-10, Effective Segment Number Generation Flowchart, for details.
The Use of Bit 29 of the Instruction Word

In the foregoing discussion of Effective Address Formation Involving Offset Only it was noted that a preliminary step of loading the ADDRESS field (y) of the Instruction Word into C(TPR.CA) was performed before the specified modification was carried out. C(TPR.CA) was then used as one data input to the modification process.

If bit 29 of the Instruction Word is set to "1", so-called Pointer Register modification is invoked and the preliminary step is executed as follows:

1. The ADDRESS field of the Instruction Word is interpreted as shown in Figure 6-7 below.
2. C(PRn.SNR) -> C(TPR.TSR)
3. maximum (C(PRn.RNR), C(TPR.TRR), C(PPR.PRR)) -> C(TPR.TRR)
4. C(PRn.WORDNO) + OFFSET -> C(TPR.CA)

Figure 6-7 Format of Instruction Word ADDRESS When Bit 29 = 1

After this preliminary step is performed, Effective Address Formation proceeds as discussed above or as discussed for the Special Modifiers below.

Special Modifiers

Whenever the Processor is forming an Append Mode Effective Address two special Address Modifiers may be specified and are effective under certain restrictive conditions. The special Address Modifiers are shown in Table 6-4 below and discussed in the paragraphs following.

The conditions for which the special Address Modifiers are effective are as follows:

1. The Processor must be forming an Append Mode Effective Address, that is, it must be in Append Mode or in Absolute Mode with bit 29 set in the Instruction Word.
2. The Instruction Word (or previous Indirect Word) must specify Indirect Then Register or Register Then Indirect modification.
3. The Effective Address for the Indirect Word must be even.
If any of these conditions is violated, the special Address Modifier will be interpreted as a normal Address Modifier and will cause an Illegal Procedure, Illegal Modifier Fault.

Table 6-4 Special Append Mode Address Modifiers

<table>
<thead>
<tr>
<th>TAG Value</th>
<th>Coding</th>
<th>Mnemonic</th>
<th>Modification Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>itp</td>
<td>Indirect to Pointer</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>its</td>
<td>Indirect to Segment</td>
<td></td>
</tr>
</tbody>
</table>

INDIRECT TO POINTER (ITP) MODIFICATION

If the conditions above are satisfied, the Processor examines the TAG field of the Indirect Word for the value 41 (octal). If that value is found, the Indirect Word-pair is interpreted as an ITP Pointer Pair (See Figure 6-8 below for format) and the following actions take place:

For \( n = C(ITP,PRNUM) \):

\[
C(PRn,SNR) \rightarrow C(TPR,TSR)
\]

- maximum \( C(PRn,RNR), C(SDH,R1), C(TPR,TRR) \) \( \rightarrow C(TPR,TRR) \)
- \( C(ITP,BITNO) \rightarrow C(TPR,TBR) \)
- \( C(PRn,WORDNO) + C(ITP,WORDNO) + C(c) \rightarrow C(TPR,CA) \)

where:

1. \( c = C(ITP,HOLD) \) if the TAG field of the Instruction Word or preceding Indirect Word specified Indirect Then Register modification, or

2. \( c = C(ITP,MOD,td) \) if the TAG field of the Instruction Word or preceding Indirect Word specified Register Then Indirect modification and ITP.MOD specifies either Register or Indirect Then Indirect modification.

3. SDH.R1 is the upper limit of the read/write Ring Bracket for the segment \( C(PRn,SNR) \). (See Section VIII, Hardware Ring Implementation.)
Even Word

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>2</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
<td>9</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPRNUM</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (41) 8</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Odd Word

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>W ORD NO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>BIT NO</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6-8 ITP Pointer Pair Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRNUM</td>
<td>The number of the Pointer Register through which to make the segment reference.</td>
<td></td>
</tr>
<tr>
<td>WORD NO</td>
<td>A word offset value to be added to C(PRNO.WORD NO).</td>
<td></td>
</tr>
<tr>
<td>BIT NO</td>
<td>A bit offset value for the data item.</td>
<td></td>
</tr>
<tr>
<td>MOD</td>
<td>Any normal Address Modifier (not ITP or ITS).</td>
<td></td>
</tr>
</tbody>
</table>

INDIRECT TO SEGMENT (ITS) MODIFICATION

If the conditions above are satisfied, the Processor examines the TAG field of the Indirect Word for the value 43 (octal). If that value is found, the Indirect Word-pair is interpreted as an ITS Pointer Pair (See Figure 6-9 below for format) and the following actions take place:

C(ITS.SEGNO) -> C(ITP.TSR)

maximum (C(ITS.RN), C(SDW.R1), C(TPR.TRR)) -> C(TPR.TRR)

C(ITS.BIT NO) -> C(ITP.TBR)

C(ITS.WORD NO) + C(c) -> C(ITP.CA)

where:

1. C = C(CT-HOD) if the TAG field of the Instruction Word or preceding Indirect Word specified Indirect Then Register modification, or

2. C = C(ITP.MOD.Id) if the TAG field of the Instruction Word or preceding Indirect Word specified Register Then Indirect modification and ITP.MOD specifies either Register or Register Then Indirect modification.
3. SDW.RI is the upper limit of the read/write Ring Bracket for the segment C(ITS.SEGNO). (See Section VIII, Hardware Ring Implementation.)

Even Word

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2 3</td>
<td>7 6 0</td>
<td>1</td>
<td>9 0 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 0 0 0 0 0 0 0 0 0 1</td>
<td>(43) 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>15 3</td>
<td>9 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Odd Word

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7 8 9</td>
<td>6 7 3 0 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 WORDNO</td>
<td>10 0 0 1 BITNO 10 0 0 1 MOD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18 3</td>
<td>6 3</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6-9 ITS Pointer Pair Format

Field Name | Meaning
---|---
SEGNO | The number of the segment to be referenced.
WORDNO | Word offset to be used in the effective address formation.
BITNO | The bit offset for the data item.
MOD | Any valid normal Address Modifier.

Effective Segment Number Generation

The details of Effective Segment Number generation are shown in the flowchart in Figure 6-10 below.
Figure 6-10 Effective Segment Number Generation Flowchart
Figure 6-10a Effective Segment Number Generation Flowchart (Con't.)

EFFECTIVE ADDRESS FORMATION FOR EXTENDED INSTRUCTION SET

A flowchart of the steps involved in Operand Descriptor Effective Address Formation is shown in Figure 6-11 below. The flowchart depicts the Effective Address Formation for operand \( a \) as described by its Modification Field, \( MF_a \). This Effective Address Formation is performed for each operand as its Operand Descriptor is decoded.
The symbol "y" stands for the contents of the ADDRESS field of the Operand Descriptor. The symbols "CN" and "C" stand for the contents of the Character Number field. The symbol "B" stands for the contents of the Bit Number field.

The algorithms used in the formation of the Effective Word/Char/Bit Address are described in "Character- and Bit-String Addressing" following.
Character- and Bit-String Addressing

The Processor represents the Effective Address of a character- or bit-string operand in three different forms as follows:

1. Pointer Register Form

This form consists of a word value (PRn.WORDNO) and a bit value (PRn.BITNO). The word value is the word offset of the word containing the first character or bit of the operand and the bit value is the bit position of that character or bit within the word. This form is seen when C(PRn) are stored as an ITS Pointer Pair or as a Packed Pointer (See "Indirect to Segment (its) Modification" earlier in this Section).

2. Address Register Form

This form consists of a word value (ARn.WORDNO), a character number (ARn.CHAR), and a bit value (ARn.BITNO). The word value is the word offset of the word containing the first character or bit of the operand. The character number is the number of the 9-bit character containing the first character or bit. The bit value is the bit bit position within ARn.CHAR of the first character or bit. This form is seen when C(ARn) are stored with the Store Address Register (SARn) instruction.

3. Operand Descriptor Form

This form is valid for character-string operands only. It consists of a word value (ADDRESS) and a character number (CN). The word value is the word offset of the word containing the first character of the operand and the character number is the number of that character within the word. This form is seen when C(ARn) is stored with the ARn to Alphanumeric Descriptor (ARN) or ARn to Numeric Descriptor (ARN) instructions. (The Operand Descriptor form for bit-string operands is identical to the Address Register form.)

NOTE: The terms "Pointer Register" and "Address Register" both apply to the same physical hardware register. The distinction arises from the manner in which the register is invoked and used and in the interpretation of the register contents. "Pointer Register" refers to the register as used by the Appending Unit and "Address Register" refers to the register as used by the Decimal unit.

The three forms are compatible and may be freely intermixed. For example, PRn may be loaded in Pointer Register form with the Effective Pointer to PRn (EPPn) instruction, then modified in Pointer Register form with the Effective Address to Word/Bit Number of PRn (EAWPn), then further modified in Address Register form (assuming character size k) with the Add k-Bit Displacement to Address Register (AARBD) instruction, and finally invoked in Operand Descriptor form by the use of AFTAR in an EIS Multiword instruction.

Character- and Bit-String Address Arithmetic Algorithms

The arithmetic algorithms for calculating character- and bit-string addresses are presented below. The symbols "ADDRESS" and "CN" represent the...
ADDRESS and CN fields of the Operand Descriptor being decoded. "C" and "D" are set according to the flowchart in Figure 6-11 above. If either has the value "null", the contents of all fields shown is identically zero.

9-BIT CHARACTER STRING ADDRESS ARITHMETIC

Effective BITNO = 0000
Effective CHAR = (CN + C(ARQ-CHAR) + C(C)) modulo 4
Effective WORDNO = ADDRESS + C(ARQ-WORDNO) + (CN + C(ARQ-CHAR) + C(C)) / 4

6-BIT CHARACTER STRING ADDRESS ARITHMETIC

Effective BITNO = (9*C(ARQ-CHAR) + 6*C(C) + C(ARQ-BITNO)) modulo 9
Effective CHAR = (((9*C(ARQ-CHAR) + 6*C(C) + C(ARQ-BITNO)) modulo 36) / 9
Effective WORDNO = ADDRESS + C(ARQ-WORDNO) + (9*C(ARQ-CHAR) + 6*C(C) + C(ARQ-BITNO)) / 36

4-BIT CHARACTER STRING ADDRESS ARITHMETIC

Effective BITNO = 4 * (C(ARQ-CHAR) + 2*C(C) + C(ARQ-BITNO))/4 modulo 2 + 1
Effective CHAR = (((9*C(ARQ-CHAR) + 4*C(C) + C(ARQ-BITNO)) modulo 36) / 9
Effective WORDNO = ADDRESS + C(ARQ-WORDNO) + (9*C(ARQ-CHAR) + 4*C(C) + C(ARQ-BITNO)) / 36

BIT STRING ADDRESS ARITHMETIC

Effective BITNO = (9*C(ARQ-CHAR) + 36*C(C) + C(ARQ-BITNO)) modulo 9
Effective CHAR = (((9*C(ARQ-CHAR) + 36*C(C) + C(ARQ-BITNO)) modulo 36) / 9
Effective WORDNO = ADDRESS + C(ARQ-WORDNO) + (9*C(ARQ-CHAR) + 36*C(C) + C(ARQ-BITNO)) / 36
SECTION VII

FAULTS AND INTERRUPTS

Faults and Interrupts both result in an interruption of normal sequential processing, but there is a difference in how they originate. Generally, Faults are caused by events or conditions that are internal to the Processor and Interrupts are caused by events or conditions that are external to the Processor. Faults and Interrupts enable the Processor to respond promptly when conditions occur that require system attention. A unique word-pair is dedicated for the instructions to service each Fault and Interrupt condition. The instruction pair associated with a Fault is called the Fault Vector. The instruction pair associated with an Interrupt is called the Interrupt Vector.

FAULT CYCLE SEQUENCE

Following the detection of a Fault condition, the Control Unit determines the proper time to initiate the Fault Sequence according to the Fault Group. At that time, the Control Unit interrupts normal sequential processing with an Abort Cycle. The Abort Cycle brings all overlapped and asynchronous functions within the Processor to an orderly halt. At the end of the Abort Cycle, the Control Unit initiates a Fault Cycle.

In the Fault Cycle, the Processor safe-stores the Control Unit Data (See Section IV, Program Accessible Registers) into program-invisible holding registers in preparation for a Store Control Unit (SCU) instruction, then enters Temporary Absolute Mode and generates an Effective Address for the Fault Vector by concatenating the setting of the FAULT CONTROL switches on the Processor Maintenance panel with twice the Fault Number (See Table 7-1). This Effective Address and the Operation Code for the Execute Double (EDO) instruction are forced into the Instruction Register and executed as an instruction. Note that the execution of the instruction is not done in a normal Execute Cycle but in the Fault Cycle with the Processor in Temporary Absolute Mode.

If the attempt to fetch and execute the instruction pair at the Fault Vector results in another Fault, the current Fault Cycle is aborted and a new Fault Cycle for the Trouble Fault (Fault Number 31) is initiated. In the Fault Cycle for a Trouble Fault, the Processor does not safe-store the Control Unit Data. Therefore, it may be possible to recover the conditions for the original Fault by use of the Store Control Unit (SCU) instruction.

If either of the two instructions in the Fault Vector results in a transfer of control to an Effective Address generated in Absolute Mode, the Absolute Mode indicator is set ON for the transfer and remains ON thereafter until changed by program action.
If either of the two instructions in the Fault Vector results in a transfer of control to an Effective Address generated in Append Mode (through the use of bit 29 of the instruction word or by use of the itp or itp modifiers), the transfer is made in the Normal Mode and the Processor remains in Normal Mode thereafter.

If no transfer of control takes place, the Processor returns to the mode in effect at the time of the fault and resumes normal sequential execution with the instruction following the faulting instruction (CPR. IC + 1).

Many of the fault conditions are deliberately or inadvertently caused by the software and do not necessarily involve error conditions. The operating supervisor determines the proper action for each fault condition by analyzing the machine conditions at the time of the fault. Therefore, it is necessary that the first instruction in each of the Fault Vectors be the Store Control Unit (SCU) instruction and the second be a transfer to a routine to analyze the machine conditions. If a fault condition is to be intentionally ignored, the Fault Vector for that condition should contain an SCU/RCU pair referencing a unique V-block. By use of this pair, the machine conditions for the ignored fault condition may be recovered if the ignored fault causes a trouble fault.

**FAULT PRIORITY**

The Multics Processor has provision for 32 faults of which 27 are implemented. The faults are classified into seven fault priority groups that roughly correspond to the severity of the faults. Fault priority groups are defined so that fault recognition precedence may be established when two or more faults exist concurrently. Overlap and asynchronous functions in the Processor allow the simultaneous occurrence of faults. Group 1 has the highest priority and Group 7 has the lowest. In Groups 1 through 6, only one fault within each group is allowed to be active at any one time. The first fault within a group occurring through the normal program sequence is the one serviced.

In Group 7 faults are saved by the hardware for eventual recognition. In the case of simultaneous faults within group 7, shutdown has the highest priority, timer runout is next, and connect has the lowest priority.

There is a single exception to the handling of faults in priority group order. If an operand fetch generates a parity fault and the use of the operand in "closing out" instruction execution generates an overflow fault or a divide check fault, these faults are considered simultaneous but the parity fault takes precedence.
Table 7-1. List of Faults

<table>
<thead>
<tr>
<th>OCTAL NUMBER</th>
<th>DECIMAL NUMBER</th>
<th>Mnemonic</th>
<th>Name</th>
<th>Priority</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>shutdown</td>
<td>Shutdown</td>
<td>27</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>str</td>
<td>Store</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>mme</td>
<td>Master Mode Entry 1</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>f1</td>
<td>Fault Tag 1</td>
<td>17</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>tro</td>
<td>Timer Runout</td>
<td>26</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>cmd</td>
<td>Command</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>dfl</td>
<td>Derail</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>luf</td>
<td>Lockup</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>con</td>
<td>Connect</td>
<td>25</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>par</td>
<td>Parity</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>ipr</td>
<td>Illegal Procedure</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>onc</td>
<td>Op Not Complete</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>suf</td>
<td>Startup</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>off</td>
<td>Overflow</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>div</td>
<td>Divide Check</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>exf</td>
<td>Execute</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>df0</td>
<td>Directed Fault 0</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>df1</td>
<td>Directed Fault 1</td>
<td>21</td>
<td>5</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>df2</td>
<td>Directed Fault 2</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>df3</td>
<td>Directed Fault 3</td>
<td>23</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>acv</td>
<td>Access Violation</td>
<td>24</td>
<td>6</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>mme2</td>
<td>Master Mode Entry 2</td>
<td>12</td>
<td>5</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>mme3</td>
<td>Master Mode Entry 3</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>mme4</td>
<td>Master Mode Entry 4</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>f2</td>
<td>Fault Tag 2</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>f3</td>
<td>Fault Tag 3</td>
<td>19</td>
<td>5</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>unassigned</td>
<td>Unassigned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>27</td>
<td>unassigned</td>
<td>Unassigned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td>unassigned</td>
<td>Unassigned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>29</td>
<td>unassigned</td>
<td>Unassigned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>unassigned</td>
<td>Unassigned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>trb</td>
<td>Trouble</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

**Fault Recognition**

For the discussion following, the term "function" is defined as a major Processor functional cycle. Examples are: APPEND CYCLE, EA CYCLE, Instruction Fetch Cycle, Operand Store Cycle, Divide Execution Cycle.

Faults in Groups 1 and 2 cause the Processor to abort all functions immediately by initializing itself and enter a Fault Cycle.

Faults in Group 3 cause the Processor to "close out" current functions without taking any irrevocable action (such as setting PTW.J in an APPEND CYCLE or modifying an Indirect Word in an EA CYCLE), then to discard any pending functions (such as an APPEND CYCLE needed during an EA CYCLE), and to enter a Fault Cycle.

Faults in Group 4 cause the Processor to suspend overlapped operation, complete current and pending functions for the current instruction, and then enter a Fault Cycle.
Faults in groups 5 or 6 are normally detected during Address Preparation and Instruction Decode. These faults cause the processor to suspend overlapped operation, complete the current and pending instructions, and to enter a Fault Cycle. If a fault in a higher priority group is generated by the execution of the current or pending instructions, that higher priority fault will take precedence and the group 5 or 6 fault will be lost. If a group 5 or 6 fault is detected during execution of the current instruction, (for example, an Access Violation, Out of Segment Bounds fault during certain interruptable EIS instructions), the instruction is considered "complete" upon detection of the fault.

Faults in group 7 are held and processed (with program interrupts) at the completion of the current instruction pair. Group 7 faults are inhitenable by use of bit 28 of the instruction word.

Faults in groups 3 through 6 must wait for the system controller to acknowledge the last access request before entering the Fault Cycle.

FAULT DESCRIPTIONS

Group 1 Faults

Startup

DC power has been turned on. When the power on button is depressed, the processor is first initialized and then the startup fault is recognized.

Execute

1. The execute pushbutton on the processor maintenance panel has been pressed.

2. An external gate signal has been substituted the execute pushbutton for execute pushbutton.

The selection between the above conditions is made by settings of various switches on the processor maintenance panel.

Group 2 Faults

Op Not Complete

Any of the following will cause an op not complete fault:

1. The processor has addressed a system controller to which it is not attached.

2. The addressed system controller failed to acknowledge the processor.

3. The processor has not generated a main store access request or a direct operand within 1 to 2 milliseconds and is not in the DIS
4. A Processor port received a data strobe without a preceding acknowledgement from the System Controller that it has received the access request.

5. A Processor port received a data strobe before the data previously sent to it was unloaded.

Trouble

The Trouble Fault is defined as the occurrence of a Fault during the fetch or execution of a Fault Vector or Interrupt Vector. Such Faults may be hardware generated (for example, Op Not Complete or Parity), or operating system-generated (for example, the page containing the effective address of an instruction is missing).

Group 3 Faults

Overflow

An arithmetic overflow, exponent overflow, or exponent underflow has been generated. The generation of this Fault is inhibited with the Overflow Mask indicator set ON. Subsequent resetting of the Overflow Mask indicator to OFF does not generate this Fault from previously set Overflow indicators. The Overflow Fault Mask state does not affect the setting, testing or storing of indicators. The determination of the specific overflow condition is by indicator testing by the control program.

Divide Check

A Divide Check Fault occurs when the actual division cannot be carried out for one of the reasons specified with individual divide instructions.

Group 4 Faults

Store

The Processor attempted to select a disabled port, an out-of-bounds address was generated in the BAR Mode or Absolute Mode, or an attempt was made to access a store unit that was not ready.

Command

1. The Processor attempted to load or read the Interrupt Mask Register in a System Controller in which it did not have an Interrupt Mask assigned.

2. The Processor issued an XEC command to a System Controller in which it did not have an Interrupt Mask assigned.

3. The Processor issued a Connect to a System Controller port that is masked OFF.
4. The selected System Controller is in TEST mode and a condition determined by certain System Controller Maintenance panel switches has been trapped.

5. An attempt was made to load a Pointer Register with Packed Pointer data in which the BITNO field value was greater than 60 octal.

Lockup

The program is in a code sequence which has inhibited sampling for an external interrupt (whether present or not) or Group 7 Fault for longer than the prescribed time. In Absolute Mode or Privileged Mode the lockup time is 32 milliseconds. In Normal Mode or BAR Mode the lockup time is specified by the setting of the Lockup Timer in the Cache Mode Register. The Lock Timer is program settable to 2, 4, 8, or 16 milliseconds.

While in Absolute Mode or Privileged Mode the Lockup Fault is signalled at the end of the time limit set in the Lockup Timer but is not recognized until the 32 millisecond limit. If the Processor returns to Normal Mode or BAR Mode after the Fault has been signalled but before the 32 millisecond limit, the Fault is recognized before any instruction in the new mode is executed.

Parity

1. The selected System Controller has returned an Illegal Action signal with an Illegal Action Code for one of the various main store parity error conditions.

2. A Cache data parity error has occurred either for read, write, or block load. Cache status bits for the condition have been set in the Cache Mode Register.

3. The Processor has detected a parity error in the System Controller interface port while either generating outgoing parity or verifying incoming parity.

Group 5 Faults

Master Mode Entries 1-4

The corresponding Master Mode Entry instruction has been decoded.

Fault Tags 1-3

The corresponding Indirect Then Tally variation designator has been detected during Address Preparation.

Derail

The Derail instruction has been decoded.

Illegal Procedure
1. An illegal operation code has been decoded or an illegal instruction sequence has been encountered.

2. An illegal modifier or modifier sequence has been encountered during Address Preparation.

3. An illegal address has been given in an instruction that the ADDRESS field for register selection.

4. An attempt was made to execute a privileged instruction in Normal Mode or BAR Mode.

5. An illegal digit was encountered in a Decimal Numeric operand.

The conditions for the Fault will be set in the Fault Register, Word 1 of the Control Unit Data, or in both.

Group 6 Faults

Directed Faults 0-3

A faulted Segment Descriptor Word (SDW) or Page Table Word (PTW) with the corresponding Directed Fault number has been fetched by the Appending Unit.

Access Violation

The Appending Unit has detected one of the several access violations below. Word 1 of the Control Unit Data contains status bits for the condition.

1. Not in read bracket (ACV3=ORB)
2. Not in write bracket (ACV5=OWB)
3. Not in execute bracket (ACV1=OEB)
4. No read permission (ACV4=R-OFF)
5. No write permission (ACV6=W-OFF)
6. No execute permission (ACV2=E-OFF)
7. Invalid ring crossing (ACV12=CRT)
8. Call limiter fault (ACV7=NO GA)
9. Outward call (ACV9=OCALL)
10. Bad outward call (ACV10=BOC)
11. Inward return (ACV11=INRET)
12. Ring alarm (ACV13=RALR)
13. Associative Memory error
14. Out of segment bounds
15. Illegal ring order (ACV0=IRO)
16. Out of call brackets (ACV8=OCB)

Group 7 Faults

Shutdown

An external power shutdown condition has been detected. DC POWER shutdown will occur in approximately one millisecond.
Timer Runout

The Timer Register has decremented to or through the value zero. If the Processor is in Privileged Mode or Absolute Mode, recognition of this Fault is delayed until a return to Normal Mode or BAR Mode. Counting in the Timer Register continues.

Connect

A connect signal ($CON strobe) has been received from a System Controller. This event is to be distinguished from a CIOC (connect) instruction encountered in the program sequence.

PROGRAM INTERRUPTS AND EXTERNAL FAULTS

Each System Controller contains 32 Execute Interrupt Cells that are used for communication among the active system modules (Processors, I/O Multiplexers, etc.). The Execute Interrupt Cells are organized in a numbered priority chain. Any active system module connected to a System Controller port may request the setting of an Execute Interrupt Cell with the SXC command.

When one or more Execute Interrupt Cells in a System Controller is set, the System Controller activates the Execute Interrupt Present (XIP) line to all System Controller ports having an Execute Interrupt Mask assigned in which one or more of the Execute Interrupt Cells that are set is unmasked. Execute Interrupt Masks are assigned only to Processors. Each Execute Interrupt Cell has a unique Interrupt Vector located at an Absolute Address equal to twice the cell number.

Execute Interrupt Sampling

The Processor always fetches instructions in pairs. At an appropriate point (as early as possible) in the execution of a pair of instructions, the next sequential instruction pair is fetched and held in a special instruction buffer register. The exact point depends on instruction sequence and other conditions.

If the Interrupt Inhibit Bit (bit 28) is not set in the current instruction word at the point of next sequential instruction address preparation, the Processor samples the Group 7 Faults. If any of the Group 7 Faults is found, the next sequential instruction pair is not fetched and an internal flag is set reflecting the presence of the Fault. The Processor next samples the Execute Interrupt Present lines from all eight Processor ports and loads a register with bits correponding to the states of the lines. If any bit in the register is set ON, the next sequential instruction pair is not fetched and an internal flag is set reflecting the presence of the bit(s) in the register.

NOTE: If the instruction pair address is being prepared as the result of a transfer of control condition or if the current instruction is Execute (XEC), Execute Double (XED), Repeat (RPT), Repeat Double (RPD), or Repeat Link (RPL), the Group 7 Faults and Execute Interrupt Present lines are not sampled.
At the completion of the current instruction pair (if no transfer of control has occurred) and the Processor is ready for the next instruction pair and the Group 7 Fault flag is set, the Processor will enter a Fault Cycle for the highest priority Group 7 Fault present.

At the completion of the current instruction pair (if no transfer of control has occurred) and the Processor is ready for the next instruction pair and the Execute Interrupt Present flag is set, the Processor will enter an Execute Interrupt Cycle.

**Execute Interrupt Cycle Sequence**

In the Execute Interrupt Cycle, the Processor safe-stores the Control Unit Data (See Section IV, Program Accessible Registers) into program-invisible holding registers in preparation for a Store Control Unit (SCU) instruction, then enters Temporary Absolute Mode. It then issues an XEC command to the System Controller on the highest priority port for which there is a bit set in the Execute Interrupt Present register.

The selected System Controller responds by clearing its highest priority Execute Interrupt Cell and returning the Interrupt Vector address for that cell to the Processor.

**NOTE:** If there is no Execute Interrupt Cell set in the selected System Controller (implying that all have been cleared in response to XEC commands from other Processors), the System Controller will return the address value 1 which is not a valid Interrupt Vector address. The Processor senses this value, aborts the Execute Interrupt Cycle, and returns to normal sequential instruction processing.

The Interrupt Vector address returned and the Operation Code for the Execute Double (XED) instruction are forced into the Instruction Register and executed as an instruction. Note that the execution of the instruction is not done in a normal Execute Cycle but in the Execute Interrupt Cycle with the Processor in Temporary Absolute Mode.

If the attempt to fetch and execute the instruction pair at the Interrupt Vector results in a Fault, the Execute Interrupt Cycle is aborted and a Fault Cycle for the Trouble Fault (Fault Number 31) is initiated. In the Fault Cycle for a Trouble Fault, the Processor does not safe-store the Control Unit Data. Therefore, it may be possible to recover the conditions for the Execute Interrupt by use of the Store Control Unit (SCU) instruction.

If either of the two instructions in the Interrupt Vector results in a transfer of control to an Effective Address generated in Absolute Mode, the Absolute Mode indicator is set ON for the transfer and remains ON thereafter until changed by program action.

If either of the two instructions in the Interrupt Vector results in a transfer of control to an Effective Address generated in Append Mode (through the use of bit 29 of the instruction word or by use of the ltp or ltp modifiers), the transfer is made in the Normal Mode and the Processor...
remains in Normal Mode thereafter.

If no transfer of control takes place, the Processor returns to the mode in effect at the time of the Fault and resumes normal sequential execution with the instruction following the interrupted instruction (C(PPR,IC) + 1).

**NOTE:** Due to the time required for many of the EIS data movement instructions, additional Group 7 Fault and Execute Interrupt present sampling is done during these instructions. After the initial load of the Decimal Unit input data buffer, Group 7 Faults and Execute Interrupt Present are sampling for each input operand address preparation. The instruction in execution is interrupted before the operand is fetched and flags are set into Control Unit Data to signal the restart of the instruction.
SECTION VIII
HARDWARE RING IMPLEMENTATION

RING PROTECTION PHILOSOPHY

The basic concept in the ring protection philosophy is the existence of a set of hierarchical levels of protection. A graphic representation of the concept may be given by a set of N consecutive circles, numbered 0,1,2, ..., N-1 from the inside out. The space included in circle 0 is called ring 0, the space included between circle i-1 and i is called ring i. Any segment in the system is placed in one and only one ring. The closer a segment to the center, the greater its protection and access privileges.

When a process is executing a procedure segment placed in ring R, the process is said to be in ring R or also it is said that the current ring of the process is ring R. A process in ring R potentially has access to any segment located in ring R and in outer rings. The word "potentially" is used because the final decision is subject to what access rights (read,write,execute) the user has for the given segment. On the other hand, this same process in ring R has no access to any segment located in inner rings, except to special procedures called "gates." Gates are procedures residing in a given ring and intended to provide controlled access to this ring. A process that is in ring R can enter an inner ring r only by calling one of the gate procedures associated with this inner ring r. Gates must be carefully coded and must not trust any data that has been manufactured or modified by the caller in a less privileged ring. In particular, they must validate all arguments passed to them by the caller so as not to compromise the protection of any segment residing in the inner ring.

Calls from an outer ring to an inner ring are referred to as "inward calls." They are associated with an increase in the access capability of the process and are controlled by gates. On the other hand, calls from an inner ring to an outer ring, referred to as "outward calls" are associated with a decrease in the access capability of the process and do not need to be controlled.

RING PROTECTION IN MULTICS

The ring protection designed for the Multics System uses the philosophy described above, but a few points have been altered in order to obtain more flexibility and better efficiency.
First, the assignment of a segment to one and only one ring, although sufficient to implement the solution of the protection problem, may be very inconvenient for a class of procedure segments, such as the library routines. Such procedures operate correctly in whatever ring the process is at the time they are called; they need no more access than the caller, and they might not perform correctly with less access than the caller. One solution could have been to have one copy of the library in each ring. Instead, the solution adopted by Multics was to relax the condition that a segment can be assigned to only one ring and allow a procedure segment to be assigned to a set of consecutive rings defined by two integers (r1, r2), with \( r1 \leq r2 \). Such a procedure now resides in rings \( r1 \) to \( r2 \). If it is called from ring \( R \) such that \( r1 \leq R \leq r2 \), then it behaves as if it were in ring \( R \), and executes without changing the current ring of the process. If it is called from ring \( R \) such that \( R > r2 \), then it behaves like a gate associated with ring \( r2 \), accepting the call as an inward call and decreasing the current ring of the process from \( R \) to \( r2 \). Upon return to the caller, the current ring is restored to \( R \), of course. Note that by allowing the multiple ring residency for a procedure segment, the current ring of a process is no longer defined by the procedure in execution; a new variable must be introduced to keep track of the value of the current ring.

Second, it was found desirable to be able to specify the maximum ring number from which a given gate was allowed to be called. And a third integer \( r3 \) was added to the pair of integers already associated with a segment. Any procedure segment, now, is associated with three ring numbers \( (r1, r2, r3) \) called its "ring brackets", such that \( r1 \leq r2 \leq r3 \). By convention, if \( r3 > r2 \), the procedure is a gate for ring \( r2 \), accessible from rings no higher than \( r3 \); if \( r2 = r3 \), the procedure is not a gate.

Third, it was found useful to relax, also for data segments, the condition that they be assigned to only one ring. One would like to be able to specify that a segment resides in ring \( r1 \) for "write" purposes but resides in a less privileged ring \( r2 \) for "read" purposes.

Fourth, several difficulties were encountered in the implementation of outward calls and their associated returns. Because outward calls were not found essential for implementing the Multics system, they were simply declared illegal, and as a result, a procedure with ring brackets \( (r1, r2, r3) \) cannot be called from a ring \( R \) such that \( R < r1 \).

In summary, the operations that are potentially permitted to a process in ring \( R \) on a segment whose ring brackets are \( (r1, r2, r3) \) are as follows:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Access Rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>( 1 ) if ( 0 \leq R \leq r1 )</td>
</tr>
<tr>
<td>Read</td>
<td>( 1 ) if ( 0 \leq R \leq r2 )</td>
</tr>
<tr>
<td>Execute</td>
<td>( 1 ) if ( r1 \leq R \leq r2 ) (Execution in ring ( R ))</td>
</tr>
<tr>
<td>Inward call</td>
<td>( 1 ) if ( r2 &lt; R \leq r3 ) (Execution in ring ( r2 ))</td>
</tr>
</tbody>
</table>

The attempted operations are permitted if, in addition, the user has the appropriate access rights (read, write, execute) on that segment.

RING PROTECTION IN THE MULTICS PROCESSOR

The Multics Processor offers hardware support for the implementation of the Multics ring protection. A particular effort was made to minimize the overhead associated with all authorized ring crossings, which the processor performs without operating system intervention, and to minimize the overhead associated with the validation of arguments, for which the processor provides a valuable
assistance.

The number of rings available in the processor is eight, numbered from 0 to 7. The current ring R of a process is recorded in a hardware register (PPR.PRR).

The ring brackets (r1, r2, r3) of a segment are recorded in the Segment Descriptor Word (SDW) used by the hardware to access the segment. In addition, the SDW contains the number of gates (SDW.CL) existing in the segment. The hardware assumes that all gates are located from word 0 to word (CL-1) and does not accept an inward call to this segment if the word number specified in the call is greater than (CL-1). The reason for this control is to prevent a malicious user from generating a call that would transfer control to any machine instruction of the gate procedure. (Such a call would defeat the purpose of the gate.) The SDW also contains the access rights (read, write, execute) that the user has on that segment. If the same segment is used by several processes, there is an SDW describing the segment in the Descriptor Segment of each process. In all SDWs pointing to the same segment, the values of r1, r2, r3 and CL are identical since they are user independent. The value of the access rights (read, write, execute) are not necessarily the same because they are user dependent.

In order to provide assistance in argument validation, any pointer, being stored into an ITS Pointer Pair or loaded into a Pointer Register, also contains a ring number. Although the hardware does not prevent a process from writing any ring number in an ITS Pointer Pair, it ensures that, if (r1, r2, r3) are the ring brackets of the segment in which the ITS Pointer Pair is located, the ring number field of this ITS Pointer Pair can be set or modified only from ring R such that \( R \leq r1 \). As for the ring number recorded in a Pointer Register, the hardware ensures that a process in ring \( R \) can set it to a value equal to or greater than \( R \), but never smaller.

During the execution of a machine instruction, the hardware may examine several SDWs, ITS Pointer Pairs and Pointer Registers. For any given such examination, the hardware records the maximum of the current ring, the r1 value found in an SDW, the ring number found in an ITS Pointer Pair, or the ring number found in an Pointer Register. This maximum, called the Temporary Ring Number, is kept in a hardware register (TPR.TRR) that is updated each such examination.

The reason for having this Temporary Ring Number available at any point of a machine instruction is because it represents the highest ring (least privileged) that might have created or modified any information that led the hardware to the target segment if is about to reference. Although the current ring is \( R \), the hardware uses the most pessimistic approach and pretends the current ring is (TPR.TRR), which is always equal to or greater than \( R \). Thus the hardware uses (TPR.TRR) instead of \( R \) in all comparisons with the ring brackets involved in the enforcement of the ring protection rules given in the previous paragraph.

The use of (TPR.TRR) by the hardware allows the gate procedures to rely on the hardware to perform the validation of all addresses passed to the gate by the less privileged ring. The general rule enforced here by the hardware regarding argument validation can be stated as follows: whenever an inner ring performs an operation on a given segment and references that segment through pointers manufactured by an outer ring, the operation is considered valid only if it could have been performed while in the outer ring.
APPENDING UNIT OPERATION WITH RING MECHANISM

The complete flowchart for Effective Segment Number generation, including the hardware ring mechanism, is shown in Figure 8-1 below. See the description of the Access Violation Fault in Section VII of this document for the meanings of the coded faults. The current instruction is in the Instruction Working Buffer (IWB).

START

APPEND

Was last cycle an indirect word fetch?

Yes

Was it an RTCD operand fetch?

Yes

Was it a sequential instruction fetch?

Yes

Is bit 29 ON?

Yes

n = C(IWB)0,2

No

Go To "A" (Figure 8-1a)

Figure 8-1 Complete Appending Unit Operation Flowchart
(Figure 8-1b)

Figure 8-1a Complete Appending Unit Operation Flowchart (cont.)
Figure 8-1c Complete Appending Unit Operation Flowchart (cont.)
Figure 8-1d Complete Appending Unit Operation Flowchart (con't.)

Set Fault
ACV9=OCALL

Go to "0"
(Figure 8-1c)

REVIEW DRAFT
SUBJECT TO CHANGE
October, 1975

AL79
Figure 8-1e Complete Appending Unit Operation Flowchart (con't.)
Figure 8-1f Complete Appending Unit Operation Flowchart (cont.)
Figure 8-19 CompleteAppending Unit Operation Flowchart (con't.)
Figure 8-1h Complete Appending Unit Operation Flowchart (cont.)
Figure 8-11 Complete Appending Unit Operation Flowchart (cont.)
Figure 8-1] Complete Appending Unit Operation Flowchart (cont.)
Figure 8-1k Complete Appending Unit Operation Flowchart (con't.)

Figure 8-1k Complete Appending Unit Operation Flowchart (con't.)
SECTION IX
CACHE STORE OPERATION

The Multics processor may be fitted with an optional Cache Store. The operation of this Cache Store is described in this section.

PHILOSOPHY OF CACHE STORE

The Cache Store is a high speed buffer store located within the Processor that is intended to hold operands and/or instructions in expectation of their immediate use.

This concept is different from that of holding a single operand (such as the Divisor for a Divide Instruction) in the Processor during execution of a single instruction. A Cache Store depends on the Locality of Reference Principle.

Locality of Reference involves the calculation of the probability, for any value of d, that the next instruction or operand reference after a reference to the instruction or operand at location A is to location A+d.

The calculation of probabilities for a set of values of d requires the statistical analysis of large masses of real and simulated instruction sequences and data organizations.

If it can be shown that the average expected data/instruction access time reduction (over the range 1 to d) is statistically significant in comparison to the fixed Main Store access time, then the implementation of a Cache Store with block size d will contribute a significant improvement in performance.

The results of such studies for the Multics Processor with a Cache Store as described below show a hit probability ranging between 80% and 95% (depending on instruction mix and data organization) and a performance improvement ranging up to 30%.

CACHE STORE ORGANIZATION

The Cache Store is implemented as 2048 36-bit words of high speed register storage with associated control and content directory circuitry within the Processor. It is fully integrated with the normal data path circuitry and is virtually invisible to all programming sequences.

Parity is generated, stored, and checked just as in Main Store.

The total storage is divided into 512 blocks of 4 words each and the blocks are organized into 128 “Columns” of four “Levels” each.
Cache Store/Main Store Mapping

Main Store is mapped into the Cache Store as described below and shown in Figure 9-1.

- Main Store is divided into blocks of 4 words each arranged in ascending order and numbered with the value of Final Address bits 15 through 21 of the first word of the block.
- All Main Store blocks with numbers n modulo 128 are grouped associatively with Cache Store Column n.
- Each Cache Store Column may hold any four blocks of the associated set of Main Store blocks.
- Each Cache Store column has associated with it a four entry directory (one entry for each Level and a two bit "round robin" counter.
- Parity is generated, stored, and checked on each directory entry.
- A Cache Directory entry consists of a fifteen bit ADDRESS register, a pre-set, two bit T/G or Level Number register and a level Full flag bit.
- When a Main Store block is loaded into a Cache Store block at some Level in the associated Column, the Directory ADDRESS register for that Column and Level is loaded with the Final ADDRESS bits 0 through 14.
- (Level selection is discussed in Cache Store Control following.)
<table>
<thead>
<tr>
<th>Block</th>
<th>Block</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Words</td>
<td>Words</td>
<td>Words</td>
</tr>
<tr>
<td>0,3</td>
<td>4,7</td>
<td>8,11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block</th>
<th>Block</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>129</td>
<td>130</td>
</tr>
<tr>
<td>Words</td>
<td>Words</td>
<td>Words</td>
</tr>
<tr>
<td>512,515</td>
<td>516,519</td>
<td>520,523</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>126</td>
<td>127</td>
</tr>
<tr>
<td>Words</td>
<td>Words</td>
</tr>
<tr>
<td>504,507</td>
<td>508,511</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>254</td>
<td>255</td>
</tr>
<tr>
<td>Words</td>
<td>Words</td>
</tr>
<tr>
<td>11016,11019</td>
<td>11020,11023</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>8,5</td>
<td>-4,1</td>
</tr>
<tr>
<td>Words</td>
<td>Words</td>
</tr>
<tr>
<td>-512,-509</td>
<td>-508,-5051,-504,-5011</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Column</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Level</td>
<td>Level</td>
<td>Level</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Column</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Level</td>
<td>Level</td>
<td>Level</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Level</td>
<td>Level</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Level</td>
<td>Level</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Level</td>
<td>Level</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

**Figure 9-1**  Main Store/Cache Store Mapping
Cache Store Addressing

For a read operation, the 24 bit Final Address prepared by the Appending Unit is presented simultaneously to the Cache Control and to the Main Store port selection circuitry. While port selection is being accomplished, the Cache Store is accessed as follows:

- Final Address bits 15 through 21 are used to select a Cache Store Column.
- Final Address bits 0 through 14 are matched associatively against the four Directory ADDRESS registers for the selected Column.
- If a match occurs for a Level whose Full flag is 0, a hit is signalled, the Main Store reference cycle is cancelled, and the TAG register is read out.
- The TAG value and Final Address bits 22 and 23 are used to select the Level and Word in the selected Column and the Cache Store data is read out into the data circuitry.
- If no hit is signalled, the Main Store reference cycle proceeds and a Cache Store block load cycle is initiated (See Cache Store Control below).

For a write operation, the 24 bit Final Address prepared by the Appending Unit is presented simultaneously to the Cache Control and to the Main Store port selection circuitry. While port selection is being accomplished, the Cache Store is accessed as follows:

- Final Address bits 15 through 21 are used to select a Cache Store Column.
- Final Address bits 0 through 14 are matched associatively against the four Directory ADDRESS registers for the selected Column.
- If a match occurs for a Level whose Full flag is 0, a hit is signalled and the TAG register is read out.
- The TAG value and Final Address bits 22 and 23 are used to select the Level and Word in the selected Column, a Cache Store write cycle is enabled, and the data is written to the Main Store and the Cache Store simultaneously.
- If no hit is signalled, the Main Store reference cycle proceeds with no further Cache Store action.

CACHE Store Control

Enabling and Disabling Cache Store

The Cache Store is controlled by the state of several bits in the Cache Mode Register (See Section IV, Program Accessible Registers, for a discussion of the Cache Mode Register). The Cache Mode Register may be loaded with the Load Central Processor Register (LCPR) instruction.

The Cache Store control bits are as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0</td>
<td>The lower half of the Cache Store (Levels 0 and 1) is disabled and is totally inactive.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>The lower half of the Cache Store is active and enabled as per the state of bits 20 and 21.</td>
</tr>
</tbody>
</table>
The upper half of the Cache Store (Levels 2 and 3) is disabled and is totally inactive.

The upper half of the Cache Store is active and enabled as per the state of bits 20 and 21.

The Cache Store (if active) is not used for Operands and Indirect Words.

The Cache Store (if active) is used for Operands and Indirect Words.

The Cache Store (if active) is not used for Instructions.

The Cache Store (if active) is used for Instructions.

The Cache-to-Register mode is not in effect (See "Dumping the Cache Store" following in this Section).

The Cache-to-Register mode is in effect.

NOTE: The Cache Store option furnishes a switch panel maintenance aid that attaches to the free edge of the Cache Store Control Logic Board.

The switch panel provides six switches for manual control of the Cache Store. Four of the switches inhibit the control functions of bits 18 through 21 of the Cache Mode Register and have the effect of forcing the corresponding function to be disabled.

The fifth switch inhibits the "store-aside" feature wherein the Processor is permitted to proceed immediately after the Cache Store write cycle on write operations without waiting for a data acknowledge from Main Store. (There is no software control corresponding this switch).

The sixth switch forces the "enabled" condition on all Cache Store controls without regard to the corresponding Cache Mode Register control bit. There is no switch corresponding to the Cache-to-Register control bit.

While these switches are intended primarily for Maintenance sessions, they have been found useful in testing the Cache Store during normal operation and in permitting operation of the Processor with the Cache Store in degraded or partially disabled mode.

Cache Store Control in Segment Descriptor Words

Certain data have characteristics such that they should never be loaded into the Cache Store. Primary examples of such data are hardware mailboxes for the I/O Multiplexer, Bulk Store Controller, etc., status return words, and various dynamic system data base segments such as the System Segment Table and shared Directory Segments.

In general, any data that is purposely modified by an agency external to the Processor with the intent to convey information to the Processor should never be loaded into Cache Store.

Bit 57 of the Segment Descriptor Word is used to reflect this property of "encachability" for each segment (See Section V. Addressing -- Segmentation and Paging, for a discussion of the Segment Descriptor Word).

If the bit is set ON, data from the segment may be loaded into the Cache Store; if the bit is OFF, they may not.

The encachability property may be treated as permanent (e.g., for hardware mailboxes) or dynamic (e.g., certain shared data bases) by the operating system. The operating system sets bit 57 ON or OFF as appropriate for the function to be performed on the segment.

Loading the Cache Store

The Cache Store is loaded with data implicitly whenever a Cache Store Block Load is signalled (See the discussion of read operations in "Cache Store Addressing" above in this section). There is no explicit method or instruction to load data into the Cache Store.
When a Cache Store Block Load is signalled, the Level is selected from the value of the Round Robin Counter for the selected Column, and the Cache Store Write function is enabled. (The Round Robin Counter contains the number of the least recently loaded Level.) When the data arrives from Main Store, it is written into the Cache Store and entered into the data circuitry. The Processor proceeds with the execution of the instruction requiring the operand if appropriate.

When the Cache Store Write is complete, further Address Preparation is inhibited, bit 22 of the Final Address is inverted, and a second Main Store access for the other half of the block is made. When the second half data arrives from Main Store, it is written into the Cache Store, the Level Full flag is set ON, the Round Robin Counter is advanced by 1, and Address Preparation is permitted to proceed.

If all four Level Full flags for a Column are set ON, a Column Full flag is also set ON and remains ON until one or more Levels in the Column are cleared.

**Clearing the Cache Store**

Cache Store can be cleared in two ways; General Clear and Selective Clear. The clearing action is the same in both cases, namely, the Full flags of the selected Column(s) and/or Level(s) are set OFF.

**GENERAL CLEAR**

The entire Cache Store is cleared by setting all Column and Level Full flags to OFF in the following situations:

- Upper or lower Cache Store or both becoming enabled by appropriate bits in the operand of a Load Central Processor Register (LCPRI) Instruction, or by action of the Logic Board free edge switches
- Execution of a Clear Associative Memory Segments (CAMS) Instruction with bit 15 of the address field

**SELECTIVE CLEAR**

The Cache Store is cleared selectively as follows:

- If a Read-and-Clear operation (LDAC, SZHC, etc.) results in a hit on the Cache Store, the Cache Store block hit is cleared.
- Execution of a Clear Associative Memory Pages (CAMP) with address bit 15 set ON causes Final Address bits 8 through 14 to be matched against all Cache Directory ADDRESS Registers. All Cache Store blocks hit are cleared.

**Dumping the Cache Store**

When the Cache-to-Register mode flag (bit 24 of the Cache Mode Register) is set ON, the Processor is forced to fetch the operands of all Double Precision Operations Unit Load operations from...
the Cache Store.

Final Address bits 0 through 14 are ignored. Final Address bits 15 through 21
select a Column, and final Address bits 22 and 23 select a Level.

All other operations (e.g., Instruction Fetches, Single Precision Operands, etc.) are
treated normally.

**WARNING:** Note that the phrase "treated normally" as used here includes the case where
the Cache Store is enabled. If the Cache Store is enabled, the "other" operations will
cause normal Block Loads and Cache Store Writes thus destroying the original contents
of the Cache Store.

The user is warned that the Cache Store should be disabled before dumping is attempted.

An indexed program loop involving the LOAQ and STAQ Instructions with the Cache-to-Register mode bit set ON
will serve to dump any or all of the Cache Store.

**Note:**

If a Fault or Program Interrupt should occur during the execution
of a Cache Store dumping loop, the Cache-to-Register mode bit would seriously interfere with normal address
in the servicing of such Fault or Interrupt. Hence, the Cache-to-Register mode bit is reset
automatically by any Fault or Program Interrupt.

APPENDIX A

**OPERATION CODE MAP (BIT 27 = 0)**

<table>
<thead>
<tr>
<th>000</th>
<th>001</th>
<th>002</th>
<th>003</th>
<th>004</th>
<th>005</th>
<th>006</th>
<th>007</th>
<th>010</th>
<th>011</th>
<th>012</th>
<th>013</th>
<th>014</th>
<th>015</th>
<th>016</th>
<th>017</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>IMHE</td>
<td>IDRL</td>
<td>IIMME1</td>
<td>IMME2</td>
<td>IMME3</td>
<td>IMME4</td>
<td>IINP</td>
<td>IPULS1</td>
<td>IPULS2</td>
<td>IICDC</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>020</td>
<td>IADLX0</td>
<td>IADLX1</td>
<td>IADLX2</td>
<td>IADLX3</td>
<td>IADLX4</td>
<td>IADLX5</td>
<td>IADLX6</td>
<td>IADLX7</td>
<td>IILDC</td>
<td>IADL</td>
<td>IADLX</td>
<td>IADLQ</td>
<td>IADLQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>040</td>
<td>IAUX</td>
<td>IAUX1</td>
<td>IAUX2</td>
<td>IAUX3</td>
<td>IAUX4</td>
<td>IAUX5</td>
<td>IAUX6</td>
<td>IAUX7</td>
<td>IAUXP</td>
<td>IAUXP1</td>
<td>IAUXP2</td>
<td>IAUXP3</td>
<td>IAUXP3</td>
<td>IAUXQ</td>
<td>IAUXQ</td>
</tr>
<tr>
<td>060</td>
<td>IADX0</td>
<td>IADX1</td>
<td>IADX2</td>
<td>IADX3</td>
<td>IADX4</td>
<td>IADX5</td>
<td>IADX6</td>
<td>IADX7</td>
<td>IAUX</td>
<td>IAUXC</td>
<td>IAUXC</td>
<td>IAUXC</td>
<td>IAUXC</td>
<td>IAUXQ</td>
<td>IAUXQ</td>
</tr>
</tbody>
</table>

**REVIEW DRAFT**

**SUBJECT TO CHANGE**

October, 1975

9-7

AL39
<table>
<thead>
<tr>
<th>Bit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>020</td>
<td>INVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>040</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>060</td>
<td>ICSR</td>
<td>ICSR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>IMRL</td>
<td>IMRL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>ISGD</td>
<td>ISGD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>INV</td>
<td>INV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>260</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>IMVD</td>
<td>IMVD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>320</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>340</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>360</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>420</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>440</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>460</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>520</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>540</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>560</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>620</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>640</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>660</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>700</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>720</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>740</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>760</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**OPERATION CODE MAP (BIT 27 = 0)**

**REVIEW DRAFT**

**SUBJECT TO CHANGE**

**October 1975**

**AL39**
APPENDIX B

ALPHABETIC OPERATION CODE LIST

This appendix presents a list of all Processor instruction operation codes sorted on mnemonic and giving the octal operation code value, the instruction name, and the functional category.

The function category codes are as follows:

<table>
<thead>
<tr>
<th>Function Category</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed Point</td>
<td>FD0</td>
</tr>
<tr>
<td>Boolean Operations</td>
<td>BOOL</td>
</tr>
<tr>
<td>Floating Point</td>
<td>FLTG</td>
</tr>
<tr>
<td>Pointer Register</td>
<td>PREG</td>
</tr>
<tr>
<td>Privileged</td>
<td>PRIV</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>MISC</td>
</tr>
<tr>
<td>Extended Instruction Set</td>
<td>EIS</td>
</tr>
<tr>
<td>Transfer of Control</td>
<td>TXFR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Code</th>
<th>Category</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL39</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4BD</td>
<td>502(1)</td>
<td>EIS</td>
<td>Add 4-bit Character Displacement to AR</td>
</tr>
<tr>
<td>A6BD</td>
<td>501(1)</td>
<td>EIS</td>
<td>Add 6-bit Character Displacement to AR</td>
</tr>
<tr>
<td>A9BD</td>
<td>500(1)</td>
<td>EIS</td>
<td>Add 9-bit Character Displacement to AR</td>
</tr>
<tr>
<td>AARq</td>
<td>56q(1)</td>
<td>EIS</td>
<td>Alphanumeric Descriptor to ARq</td>
</tr>
<tr>
<td>ABD</td>
<td>503(1)</td>
<td>EIS</td>
<td>Add Bit Displacement to AR</td>
</tr>
<tr>
<td>ABSA</td>
<td>212(0)</td>
<td>PRIV</td>
<td>Absolute Address to A-Register</td>
</tr>
<tr>
<td>AD2D</td>
<td>202(1)</td>
<td>EIS</td>
<td>Add Using 2 Decimal Operands</td>
</tr>
<tr>
<td>AD3D</td>
<td>222(1)</td>
<td>EIS</td>
<td>Add Using 3 Decimal Operands</td>
</tr>
<tr>
<td>ADA</td>
<td>075(0)</td>
<td>FXD</td>
<td>Add to A-Register</td>
</tr>
<tr>
<td>ADAQ</td>
<td>077(0)</td>
<td>FXD</td>
<td>Add to AQ-Register</td>
</tr>
<tr>
<td>ADE</td>
<td>415(0)</td>
<td>FLTG</td>
<td>Add to E-Register</td>
</tr>
<tr>
<td>ADL</td>
<td>033(0)</td>
<td>FXD</td>
<td>Add Low to AQ-Register</td>
</tr>
<tr>
<td>ADLA</td>
<td>035(0)</td>
<td>FXD</td>
<td>Add Logical to A-Register</td>
</tr>
<tr>
<td>ADLAQ</td>
<td>037(0)</td>
<td>FXD</td>
<td>Add Logical to AQ-Register</td>
</tr>
<tr>
<td>ADLQ</td>
<td>036(0)</td>
<td>FXD</td>
<td>Add Logical to Q-Register</td>
</tr>
<tr>
<td>ADLXq</td>
<td>02q(0)</td>
<td>FXD</td>
<td>Add Logical to Index q</td>
</tr>
<tr>
<td>ADQ</td>
<td>076(0)</td>
<td>FXD</td>
<td>Add to Q-Register</td>
</tr>
<tr>
<td>ADWP0</td>
<td>050(0)</td>
<td>PREG</td>
<td>Add to Word Number Field of PR0</td>
</tr>
<tr>
<td>ADWP1</td>
<td>051(0)</td>
<td>PREG</td>
<td>Add to Word Number Field of PR1</td>
</tr>
<tr>
<td>ADWP2</td>
<td>052(0)</td>
<td>PREG</td>
<td>Add to Word Number Field of PR2</td>
</tr>
<tr>
<td>ADWP3</td>
<td>053(0)</td>
<td>PREG</td>
<td>Add to Word Number Field of PR3</td>
</tr>
<tr>
<td>ADWP4</td>
<td>150(0)</td>
<td>PREG</td>
<td>Add to Word Number Field of PR4</td>
</tr>
<tr>
<td>ADWP5</td>
<td>151(0)</td>
<td>PREG</td>
<td>Add to Word Number Field of PR5</td>
</tr>
<tr>
<td>ADWP6</td>
<td>152(0)</td>
<td>PREG</td>
<td>Add to Word Number Field of PR6</td>
</tr>
<tr>
<td>ADWP7</td>
<td>163(0)</td>
<td>PREG</td>
<td>Add to Word Number Field of PR7</td>
</tr>
<tr>
<td>Instruction</td>
<td>Code</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>ADCX</td>
<td>06H</td>
<td>FXD</td>
<td>Add to Index N</td>
</tr>
<tr>
<td>ALE</td>
<td>775H</td>
<td>FXD</td>
<td>A-Register Left Rotate</td>
</tr>
<tr>
<td>ALS</td>
<td>735H</td>
<td>FXD</td>
<td>A-Register Left Shift</td>
</tr>
<tr>
<td>ANDA</td>
<td>376H</td>
<td>BOOL</td>
<td>AND to A-Register</td>
</tr>
<tr>
<td>ANQA</td>
<td>377H</td>
<td>BOOL</td>
<td>AND to Q-Register</td>
</tr>
<tr>
<td>ANSA</td>
<td>356H</td>
<td>BOOL</td>
<td>AND to Storage from A-Register</td>
</tr>
<tr>
<td>ANQ</td>
<td>357H</td>
<td>BOOL</td>
<td>AND to Storage from Q-Register</td>
</tr>
<tr>
<td>ANDX</td>
<td>366H</td>
<td>BOOL</td>
<td>AND to Storage from Index n</td>
</tr>
<tr>
<td>AOS</td>
<td>554H</td>
<td>FXD</td>
<td>Add One to Storage</td>
</tr>
<tr>
<td>ARAQ</td>
<td>544H</td>
<td>EIS</td>
<td>Add One to Storage to A-Register</td>
</tr>
<tr>
<td>ARSQ</td>
<td>545H</td>
<td>EIS</td>
<td>Add One to Storage to Q-Register</td>
</tr>
<tr>
<td>ASQ</td>
<td>054H</td>
<td>FXD</td>
<td>Add One to Storage to Index n</td>
</tr>
<tr>
<td>ASXQ</td>
<td>055H</td>
<td>FXD</td>
<td>Add One to Storage to A-Register</td>
</tr>
<tr>
<td>AWCQ</td>
<td>072H</td>
<td>FXD</td>
<td>Add One to Storage to Q-Register</td>
</tr>
<tr>
<td>AND</td>
<td>5071H</td>
<td>EIS</td>
<td>Add One to Storage to A-Register</td>
</tr>
<tr>
<td>BCD</td>
<td>505H</td>
<td>MISC</td>
<td>Binary-to-BCD</td>
</tr>
<tr>
<td>BTN</td>
<td>301H</td>
<td>EIS</td>
<td>Binary-to-Decimal</td>
</tr>
<tr>
<td>CALLG</td>
<td>713H</td>
<td>TXFR</td>
<td>Call</td>
</tr>
<tr>
<td>CAMP</td>
<td>532H</td>
<td>PRIV</td>
<td>Clear Associative Memory Pages</td>
</tr>
<tr>
<td>CAMS</td>
<td>532H</td>
<td>PRIV</td>
<td>Clear Associative Memory Segments</td>
</tr>
<tr>
<td>CANA</td>
<td>315H</td>
<td>BOOL</td>
<td>Comparative AND with A-Register</td>
</tr>
<tr>
<td>CANAQ</td>
<td>317H</td>
<td>BOOL</td>
<td>Comparative AND with AQ-Register</td>
</tr>
<tr>
<td>CANQ</td>
<td>316H</td>
<td>BOOL</td>
<td>Comparative AND with Q-Register</td>
</tr>
<tr>
<td>CANY</td>
<td>306H</td>
<td>BOOL</td>
<td>Comparative AND with Index n</td>
</tr>
<tr>
<td>CIOC</td>
<td>015H</td>
<td>PRIV</td>
<td>Connect</td>
</tr>
<tr>
<td>CMG</td>
<td>405H</td>
<td>FXD</td>
<td>Compare Magnitude</td>
</tr>
<tr>
<td>CMK</td>
<td>211H</td>
<td>FXD</td>
<td>Compare Masked</td>
</tr>
<tr>
<td>CMPA</td>
<td>115H</td>
<td>FXD</td>
<td>Compare with A-Register</td>
</tr>
<tr>
<td>CMPAQ</td>
<td>117H</td>
<td>FXD</td>
<td>Compare with AQ-Register</td>
</tr>
<tr>
<td>CMPB</td>
<td>066H</td>
<td>EIS</td>
<td>Compare Bit Strings</td>
</tr>
<tr>
<td>CMPIC</td>
<td>106H</td>
<td>EIS</td>
<td>Compare Alphanumeric Character Strings</td>
</tr>
<tr>
<td>CMPN</td>
<td>303H</td>
<td>EIS</td>
<td>Compare Numeric</td>
</tr>
<tr>
<td>CMPQ</td>
<td>116H</td>
<td>FXD</td>
<td>Compare with Q-Register</td>
</tr>
<tr>
<td>CMPQN</td>
<td>108H</td>
<td>FXD</td>
<td>Compare with Index n</td>
</tr>
<tr>
<td>CNAQ</td>
<td>215H</td>
<td>BOOL</td>
<td>Comparative NOT with A-Register</td>
</tr>
<tr>
<td>CNAQK</td>
<td>217H</td>
<td>BOOL</td>
<td>Comparative NOT with AQ-Register</td>
</tr>
<tr>
<td>CNXQ</td>
<td>216H</td>
<td>BOOL</td>
<td>Comparative NOT with Q-Register</td>
</tr>
<tr>
<td>CNAQN</td>
<td>206H</td>
<td>BOOL</td>
<td>Comparative NOT with Index n</td>
</tr>
<tr>
<td>CSL</td>
<td>060H</td>
<td>EIS</td>
<td>Combine Bit Strings Left</td>
</tr>
<tr>
<td>CSR</td>
<td>060H</td>
<td>EIS</td>
<td>Combine Bit Strings Right</td>
</tr>
<tr>
<td>CWL</td>
<td>111H</td>
<td>FXD</td>
<td>Compare with Limits</td>
</tr>
<tr>
<td>DFAD</td>
<td>477H</td>
<td>FLTG</td>
<td>Double Precision Floating Add</td>
</tr>
<tr>
<td>DFCM</td>
<td>427H</td>
<td>FLTG</td>
<td>Double Precision Floating Compare Magnitude</td>
</tr>
<tr>
<td>DFCMP</td>
<td>577H</td>
<td>FLTG</td>
<td>Double Precision Floating Compare</td>
</tr>
<tr>
<td>DFDI</td>
<td>527H</td>
<td>FLTG</td>
<td>Double Precision Floating Divide Inverted</td>
</tr>
<tr>
<td>DFDV</td>
<td>567H</td>
<td>FLTG</td>
<td>Double Precision Floating Divide</td>
</tr>
<tr>
<td>DFLO</td>
<td>433H</td>
<td>FLTG</td>
<td>Double Precision Floating Load</td>
</tr>
<tr>
<td>DFMP</td>
<td>463H</td>
<td>FLTG</td>
<td>Double Precision Floating Multiply</td>
</tr>
<tr>
<td>DFRO</td>
<td>473H</td>
<td>FLTG</td>
<td>Double Precision Floating Round</td>
</tr>
</tbody>
</table>
DFSB  577(0)  FLTG  Double Precision Floating Subtract
DFST  457(0)  FLTG  Double Precision Floating Store
DFSTR  472(0)  FLTG  Double Precision Floating Store Rounded
DIS   616(0)  PRIV  Delay Until Interrupt Signal
DIV   506(0)  FLTG  Divide Integer

DRL   002(0)  MISC  Detail
DTB   305(1)  EIS   Decimal-to-Binary Convert
DUFA  437(0)  FLTG  Double Precision Unnormalized Floating Add
DUFM  423(0)  FLTG  Double Precision Unnormalized Floating Multiply
DUFS  537(0)  FLTG  Double Precision Unnormalized Floating Subtract

DV2D  207(1)  EIS   Divide Using 2 Decimal Operands
DV3D  227(1)  EIS   Divide Using 3 Decimal Operands
DVF   507(0)  FXD   Divide Fraction
EAA   635(0)  FXD   Effective Address to A-Register
EAQ   636(0)  FXD   Effective Address to Q-Register

EASPO 311(0)  PREG  Effective Address to Segment Number Field of PR0
EASPI 310(1)  PREG  Effective Address to Segment Number Field of PR1
EASP2 313(0)  PREG  Effective Address to Segment Number Field of PR2
EASP3 312(1)  PREG  Effective Address to Segment Number Field of PR3
EASP4 331(0)  PREG  Effective Address to Segment Number Field of PR4

EASPS 330(1)  PREG  Effective Address to Segment Number Field of PR5
EASPE 333(0)  PREG  Effective Address to Segment Number Field
The Other Computer Company:

Honeywell