The cLEMENCy Architecture

Lightning
Legitimate Business Syndicate
July 2017
The cLEMENCy Architecture

Legitimate Business Syndicate

Deadwood Fuzyll Gyno HJ Hoju Jetboy Jymbolia Lightning Selir Sirgoon Thing2 Vito

To the extent possible under law, Legitimate Business Syndicate have waived all copyright and related or neighboring rights to “The cLEMENCy Architecture”. This work is published in the United States. For more information about this license, please visit https://creativecommons.org/publicdomain/zero/1.0/

Compiled from revision 800e888b75c209289bd4676299fb4da250ae7008 on July 26, 2017.
Contents

1 Basic Architecture .................................................................................................................. 1
   1.1 Registers .......................................................................................................................... 1
   1.2 Stack ................................................................................................................................. 2
   1.3 Relative Memory Reference .............................................................................................. 3

2 Memory layout and IO ............................................................................................................... 5
   2.1 Memory Protection ............................................................................................................. 5
   2.2 Clock IO ............................................................................................................................. 6
   2.3 Flag IO ............................................................................................................................... 6
   2.4 Data Received ................................................................................................................... 6
   2.5 Data Received Size .......................................................................................................... 7
   2.6 Data Sent .......................................................................................................................... 7
   2.7 Data Sent Size .................................................................................................................. 7
   2.8 Shared Memory ............................................................................................................... 7
   2.9 NVRAM Memory ............................................................................................................ 7
   2.10 Interrupt Pointers .......................................................................................................... 7
   2.11 Processor Identification ................................................................................................. 8

3 Interrupts and Exceptions ....................................................................................................... 11
   3.1 Timer 1 to 4 Interrupts .................................................................................................... 11
   3.2 Invalid Instruction Exception ......................................................................................... 11
   3.3 Divide by 0 Exception ..................................................................................................... 11
   3.4 Memory Exception .......................................................................................................... 11
   3.5 Data Received Interrupt ................................................................................................. 12
   3.6 Data Sent Interrupt ......................................................................................................... 12
   3.7 Exceptions ...................................................................................................................... 12

4 Instruction Set ....................................................................................................................... 13
   4.1 AD: Add ........................................................................................................................... 14
   4.2 ADC: Add With Carry ..................................................................................................... 14
   4.3 ADCI: Add Immediate With Carry .................................................................................. 15
   4.4 ADCIM: Add Immediate Multi Reg With Carry ............................................................... 15
   4.5 ADCM: Add Multi Reg With Carry ................................................................................ 16
   4.6 ADF: Add Floating Point ............................................................................................... 16
   4.7 ADFM: Add Floating Point Multi Reg ........................................................................... 16
<table>
<thead>
<tr>
<th>Section</th>
<th>Instruction</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.8</td>
<td>ADI: Add Immediate</td>
<td>17</td>
</tr>
<tr>
<td>4.9</td>
<td>ADIM: Add Immediate Multi Reg</td>
<td>17</td>
</tr>
<tr>
<td>4.10</td>
<td>ADM: Add Multi Reg</td>
<td>17</td>
</tr>
<tr>
<td>4.11</td>
<td>AN: And</td>
<td>18</td>
</tr>
<tr>
<td>4.12</td>
<td>ANI: And Immediate</td>
<td>18</td>
</tr>
<tr>
<td>4.13</td>
<td>ANM: And Multi Reg</td>
<td>18</td>
</tr>
<tr>
<td>4.14</td>
<td>B: Branch Conditional</td>
<td>19</td>
</tr>
<tr>
<td>4.15</td>
<td>BF: Bit Flip</td>
<td>20</td>
</tr>
<tr>
<td>4.16</td>
<td>BFM: Bit Flip Multi Reg</td>
<td>20</td>
</tr>
<tr>
<td>4.17</td>
<td>BR: Branch Register Conditional</td>
<td>21</td>
</tr>
<tr>
<td>4.18</td>
<td>BRA: Branch Absolute</td>
<td>22</td>
</tr>
<tr>
<td>4.19</td>
<td>BRR: Branch Relative</td>
<td>22</td>
</tr>
<tr>
<td>4.20</td>
<td>C: Call Conditional</td>
<td>23</td>
</tr>
<tr>
<td>4.21</td>
<td>CAA: Call Absolute</td>
<td>24</td>
</tr>
<tr>
<td>4.22</td>
<td>CAR: Call Relative</td>
<td>24</td>
</tr>
<tr>
<td>4.23</td>
<td>CM: Compare</td>
<td>25</td>
</tr>
<tr>
<td>4.24</td>
<td>CMF: Compare Floating Point</td>
<td>26</td>
</tr>
<tr>
<td>4.25</td>
<td>CMFM: Compare Floating Point Multi Reg</td>
<td>26</td>
</tr>
<tr>
<td>4.26</td>
<td>CMI: Compare Immediate</td>
<td>27</td>
</tr>
<tr>
<td>4.27</td>
<td>CMIM: Compare Immediate Multi Reg</td>
<td>28</td>
</tr>
<tr>
<td>4.28</td>
<td>CMM: Compare Multi Reg</td>
<td>29</td>
</tr>
<tr>
<td>4.29</td>
<td>CR: Call Register Conditional</td>
<td>30</td>
</tr>
<tr>
<td>4.30</td>
<td>DBRK: Debug Break</td>
<td>31</td>
</tr>
<tr>
<td>4.31</td>
<td>DI: Disable Interrupts</td>
<td>31</td>
</tr>
<tr>
<td>4.32</td>
<td>DMT: Direct Memory Transfer</td>
<td>32</td>
</tr>
<tr>
<td>4.33</td>
<td>DV: Divide</td>
<td>32</td>
</tr>
<tr>
<td>4.34</td>
<td>DVF: Divide Floating Point</td>
<td>33</td>
</tr>
<tr>
<td>4.35</td>
<td>DVFM: Divide Floating Point Multi Reg</td>
<td>33</td>
</tr>
<tr>
<td>4.36</td>
<td>DVI: Divide Immediate</td>
<td>33</td>
</tr>
<tr>
<td>4.37</td>
<td>DVIM: Divide Immediate Multi Reg</td>
<td>34</td>
</tr>
<tr>
<td>4.38</td>
<td>DVIS: Divide Immediate Signed</td>
<td>34</td>
</tr>
<tr>
<td>4.39</td>
<td>DVISM: Divide Immediate Signed Multi Reg</td>
<td>34</td>
</tr>
<tr>
<td>4.40</td>
<td>DVM: Divide Multi Reg</td>
<td>35</td>
</tr>
<tr>
<td>4.41</td>
<td>DVS: Divide Signed</td>
<td>35</td>
</tr>
<tr>
<td>4.42</td>
<td>DVSM: Divide Signed Multi Reg</td>
<td>35</td>
</tr>
<tr>
<td>4.43</td>
<td>EI: Enable Interrupts</td>
<td>36</td>
</tr>
<tr>
<td>4.44</td>
<td>FTI: Float to Integer</td>
<td>36</td>
</tr>
<tr>
<td>4.45</td>
<td>FTIM: Float to Integer Multi Reg</td>
<td>36</td>
</tr>
<tr>
<td>4.46</td>
<td>HT: Halt</td>
<td>37</td>
</tr>
<tr>
<td>4.47</td>
<td>IR: Interrupt Return</td>
<td>37</td>
</tr>
<tr>
<td>4.48</td>
<td>ITF: Integer to Float</td>
<td>38</td>
</tr>
<tr>
<td>4.49</td>
<td>ITFM: Integer to Float Multi Reg</td>
<td>38</td>
</tr>
<tr>
<td>4.50</td>
<td>LDS: Load Single</td>
<td>39</td>
</tr>
<tr>
<td>4.51</td>
<td>LDT: Load Tri</td>
<td>40</td>
</tr>
<tr>
<td>4.52</td>
<td>LDW: Load Word</td>
<td>41</td>
</tr>
<tr>
<td>4.53</td>
<td>MD: Modulus</td>
<td>42</td>
</tr>
</tbody>
</table>
CONTENTS

4.54 MDF: Modulus Floating Point .................................. 42
4.55 MDFM: Modulus Floating Point Multi Reg .......................... 42
4.56 MDI: Modulus Immediate ........................................... 43
4.57 MDIM: Modulus Immediate Multi Reg ............................... 43
4.58 MDIS: Modulus Immediate Signed ................................. 44
4.59 MDISM: Modulus Immediate Signed Multi Reg ....................... 44
4.60 MDM: Modulus Multi Reg ............................................ 45
4.61 MDS: Modulus Signed ................................................ 45
4.62 MDSM: Modulus Signed Multi Reg ................................. 45
4.63 MH: Move High .................................................... 46
4.64 ML: Move Low ...................................................... 46
4.65 MS: Move Low Signed ................................................ 46
4.66 MU: Multiply ....................................................... 47
4.67 MUF: Multiply Floating Point ..................................... 47
4.68 MUFM: Multiply Floating Point Multi Reg ......................... 47
4.69 MUI: Multiply Immediate .......................................... 48
4.70 MUIM: Multiply Immediate Multi Reg ........................... 48
4.71 MUIS: Multiply Immediate Signed ................................... 48
4.72 MUISM: Multiply Immediate Signed Multi Reg ..................... 49
4.73 MUIM: Multiply Immediate ........................................ 49
4.74 MUS: Multiply Signed ............................................... 49
4.75 MUSM: Multiply Signed Multi Reg ............................... 50
4.76 NG: Negate .......................................................... 50
4.77 NGF: Negate Floating Point ....................................... 50
4.78 NGFM: Negate Floating Point Multi Reg ............................ 51
4.79 NGM: Negate Multi Reg ............................................ 51
4.80 NH: Not ............................................................. 51
4.81 NTM: Not Multi Reg .................................................. 52
4.82 OR: Or .............................................................. 52
4.83 ORI: Or Immediate .................................................. 52
4.84 ORM: Or Multi Reg .................................................. 53
4.85 RE: Return .......................................................... 53
4.86 RF: Read Flags ....................................................... 53
4.87 RL: Rotate Left ...................................................... 54
4.88 RL: Rotate Left Immediate ......................................... 54
4.89 RLM: Rotate Left Immediate Multi Reg ........................... 54
4.90 RLM: Rotate Left Multi Reg ....................................... 55
4.91 RMP: Read Memory Protection .................................... 56
4.92 RND: Random ........................................................ 57
4.93 RNDM: Random Multi Reg ......................................... 57
4.94 RR: Rotate Right ..................................................... 57
4.95 RRI: Rotate Right Immediate ...................................... 58
4.96 RRIM: Rotate Right Immediate Multi Reg .......................... 58
4.97 RRIM: Rotate Right Multi Reg .................................... 58
4.98 SA: Shift Arithmetic Right ......................................... 59
4.99 SAI: Shift Arithmetic Right Immediate ............................ 59
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.100</td>
<td>SAIM: Shift Arithmetic Right Immediate Multi Reg</td>
<td>59</td>
</tr>
<tr>
<td>4.101</td>
<td>SAM: Shift Arithmetic Right Multi Reg</td>
<td>60</td>
</tr>
<tr>
<td>4.102</td>
<td>SB: Subtract</td>
<td>60</td>
</tr>
<tr>
<td>4.103</td>
<td>SBC: Subtract With Carry</td>
<td>61</td>
</tr>
<tr>
<td>4.104</td>
<td>SBCI: Subtract Immediate With Carry</td>
<td>61</td>
</tr>
<tr>
<td>4.105</td>
<td>SBCM: Subtract Immediate Multi Reg With Carry</td>
<td>62</td>
</tr>
<tr>
<td>4.106</td>
<td>SBCM: Subtract Multi Reg With Carry</td>
<td>62</td>
</tr>
<tr>
<td>4.107</td>
<td>SBF: Subtract Floating Point</td>
<td>63</td>
</tr>
<tr>
<td>4.108</td>
<td>SBFM: Subtract Floating Point Multi Reg</td>
<td>63</td>
</tr>
<tr>
<td>4.109</td>
<td>SBE: Subtract Immediate</td>
<td>63</td>
</tr>
<tr>
<td>4.110</td>
<td>SBIM: Subtract Immediate Multi Reg</td>
<td>64</td>
</tr>
<tr>
<td>4.111</td>
<td>SBM: Subtract Multi Reg</td>
<td>64</td>
</tr>
<tr>
<td>4.112</td>
<td>SES: Sign Extend Single</td>
<td>64</td>
</tr>
<tr>
<td>4.113</td>
<td>SEW: Sign Extend Word</td>
<td>65</td>
</tr>
<tr>
<td>4.114</td>
<td>SF: Set Flags</td>
<td>65</td>
</tr>
<tr>
<td>4.115</td>
<td>SL: Shift Left</td>
<td>65</td>
</tr>
<tr>
<td>4.116</td>
<td>SLI: Shift Left Immediate</td>
<td>66</td>
</tr>
<tr>
<td>4.117</td>
<td>SLIM: Shift Left Immediate Multi Reg</td>
<td>66</td>
</tr>
<tr>
<td>4.118</td>
<td>SLM: Shift Left Multi Reg</td>
<td>66</td>
</tr>
<tr>
<td>4.119</td>
<td>SMP: Set Memory Protection</td>
<td>67</td>
</tr>
<tr>
<td>4.120</td>
<td>SR: Shift Right</td>
<td>68</td>
</tr>
<tr>
<td>4.121</td>
<td>SRIM: Shift Right Immediate</td>
<td>68</td>
</tr>
<tr>
<td>4.122</td>
<td>SRIM: Shift Right Immediate Multi Reg</td>
<td>69</td>
</tr>
<tr>
<td>4.123</td>
<td>SRM: Shift Right Multi Reg</td>
<td>69</td>
</tr>
<tr>
<td>4.124</td>
<td>STS: Store Single</td>
<td>70</td>
</tr>
<tr>
<td>4.125</td>
<td>STT: Store Tri</td>
<td>71</td>
</tr>
<tr>
<td>4.126</td>
<td>STW: Store Word</td>
<td>72</td>
</tr>
<tr>
<td>4.127</td>
<td>WT: Wait</td>
<td>73</td>
</tr>
<tr>
<td>4.128</td>
<td>XR: Xor</td>
<td>73</td>
</tr>
<tr>
<td>4.129</td>
<td>XRI: Xor Immediate</td>
<td>73</td>
</tr>
<tr>
<td>4.130</td>
<td>XRM: Xor Multi Reg</td>
<td>74</td>
</tr>
<tr>
<td>4.131</td>
<td>ZES: Zero Extend Single</td>
<td>74</td>
</tr>
<tr>
<td>4.132</td>
<td>ZEW: Zero Extend Word</td>
<td>74</td>
</tr>
</tbody>
</table>
List of Tables

1.1 Registers .................................................. 2
1.2 Flags Register Layout .................................... 2

2.1 Memory Mapping ............................................ 5
2.2 Memory Protection States .................................. 6
2.3 Clock and Timer Parameters ................................. 6
2.4 Interrupt Pointer Addresses ................................. 8
2.5 Processor Identification Attributes ......................... 8
2.6 Processor Functionality Flags ............................... 8

3.1 Exception Identifiers ...................................... 12
Chapter 1

Basic Architecture

cLEMENCy is the LEgitbs Middle ENdian Computer architecture developed by Lightning for DEF CON CTF.

Each byte is 9 bits of data, bit 0 is the left most significant bit. Middle-Endian data stores bits 9 to 17, followed by bits 0 to 8, then bits 18 to 27 in memory when handling three bytes. Two bytes of data will have bits 9-17 then bits 0 to 8 written to memory.

Register XXYYZZ → Memory YYXXZZ
Register XXYY → Memory YYXX

1.1 Registers

cLEMENCy has 32 general purpose 27-bit wide registers that serve both floating point and integer math operations along with a separate flag register. The multi-register format allows for a 54-bit value to be used during math operations by using two registers side by side while the starting register can be any register. If an instruction goes past the PC register while accessing multiple registers then access continues to R0. Attempts to write to PC by a load instruction are ignored.

The multi-register format and floating point operations are optional components of the processor, checking the processor features is recommended before attempts are made in using such instructions.

The ST, RA, and PC registers have a special purpose while all other registers are general use. The following table is a recommended register setup for compilers:
CHAPTER 1. BASIC ARCHITECTURE

Table 1.1: Registers

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Register Number</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>0</td>
<td>General purpose, parameter 1 and function return value</td>
</tr>
<tr>
<td>R1 to R8</td>
<td>1 to 8</td>
<td>General purpose, parameters 2 to 8</td>
</tr>
<tr>
<td>R9 to R28</td>
<td>9 to 28</td>
<td>General purpose, saved between function calls</td>
</tr>
<tr>
<td>ST</td>
<td>29</td>
<td>Pointer into the current memory location holding the current end of the stack</td>
</tr>
<tr>
<td>RA</td>
<td>30</td>
<td>Return address register, filled in by the call instruction, used when a return is executed</td>
</tr>
<tr>
<td>PC</td>
<td>31</td>
<td>Program counter, register is read-only</td>
</tr>
<tr>
<td>FL</td>
<td></td>
<td>Current flag and interrupt state</td>
</tr>
</tbody>
</table>

The flags register has the following layout:

Table 1.2: Flags Register Layout

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X | X | X | X | X | X | X | X | S | O | C | Z |
| Data Sent Interrupt Enabled |
| Data Received Interrupt Enabled |
| Memory Exception Enabled |
| Divide by 0 Exception Enabled |
| Invalid Instruction Exception Enabled |
| Timer4 Interrupt Enabled |
| Timer3 Interrupt Enabled |
| Timer2 Interrupt Enabled |
| Timer1 Interrupt Enabled |
| Signed bit |
| Overflow bit |
| Carry bit |
| Zero bit |

1.2 Stack

Due to no specific stack based instructions, there is no expected direction the stack should grow, however during an interrupt the processor will subtract 99 bytes of data from the stack pointer when storing all registers to the stack and the interrupt return will read 99 bytes from the current stack pointer. If an implementation needs the interrupts to add to the stack instead of subtracting
then changing the 'Interrupt stack direction flag' bit in the features area of the processor will accomplish this.

1.3 Relative Memory Reference

All relative references are from the beginning of the instruction handling the relative reference. A relative branch will adjust PC by the amount in the relative offset without including the branch instruction size. Only if the branch is not taken is PC adjusted by the instruction size.
Chapter 2

Memory layout and IO

There are 2 main areas of memory, the RAM area and the DMA mapped areas. Processor execution starts at memory offset 0 and all DMA memory has the high bit of the memory address set. The following table provides the memory mapping for cLEMENCy processors:

<table>
<thead>
<tr>
<th>Memory Start</th>
<th>Memory End</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
<td>3FFFFFF</td>
<td>Main Program Memory</td>
</tr>
<tr>
<td>4000000</td>
<td>400001D</td>
<td>Clock IO</td>
</tr>
<tr>
<td>4010000</td>
<td>4010FFF</td>
<td>Flag IO</td>
</tr>
<tr>
<td>5000000</td>
<td>5001FFF</td>
<td>Data Received</td>
</tr>
<tr>
<td>5002000</td>
<td>5002002</td>
<td>Data Received Size</td>
</tr>
<tr>
<td>5010000</td>
<td>5011FFF</td>
<td>Data Sent</td>
</tr>
<tr>
<td>5012000</td>
<td>5012002</td>
<td>Data Sent Size</td>
</tr>
<tr>
<td>6000000</td>
<td>67FFFFF</td>
<td>Shared Memory</td>
</tr>
<tr>
<td>6800000</td>
<td>68FFFFF</td>
<td>NVRAM Memory</td>
</tr>
<tr>
<td>7FFFFFF0</td>
<td>7FFFF1B</td>
<td>Interrupt Pointers</td>
</tr>
<tr>
<td>7FFFFFF80</td>
<td>7FFFFFFF</td>
<td>Processor Identification and Features</td>
</tr>
</tbody>
</table>

2.1 Memory Protection

Memory is broken up into 1024 byte pages. Each page can have 1 of 4 states applied to it.
Table 2.2: Memory Protection States

<table>
<thead>
<tr>
<th>State</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Access</td>
</tr>
<tr>
<td>1</td>
<td>Read Only</td>
</tr>
<tr>
<td>2</td>
<td>Read/Write</td>
</tr>
<tr>
<td>3</td>
<td>Read/Execute</td>
</tr>
</tbody>
</table>

Attempting to interact with memory that is inconsistent with its current state will result in a memory exception occurring. If the exception is turned off and the attempt is execution then the processor will halt. The memory protection flags are from the processor only allowing for external IO controllers to modify any memory they are associated with even if the memory protection flags are set on their regions to Read Only or No Access.

### 2.2 Clock IO

There are 6 bytes per timer with 4 timers maximum. A 0 for the timer delay disables that specific timer. Each timer has a 1 millisecond accuracy.

<table>
<thead>
<tr>
<th>Memory Start</th>
<th>Bytes</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000000</td>
<td>3</td>
<td>Timer 1 Delay</td>
</tr>
<tr>
<td>4000003</td>
<td>3</td>
<td>Number of milliseconds left for Timer 1</td>
</tr>
<tr>
<td>4000006</td>
<td>3</td>
<td>Timer 2 Delay</td>
</tr>
<tr>
<td>4000009</td>
<td>3</td>
<td>Number of milliseconds left for Timer 2</td>
</tr>
<tr>
<td>400000C</td>
<td>3</td>
<td>Timer 3 Delay</td>
</tr>
<tr>
<td>400000F</td>
<td>3</td>
<td>Number of milliseconds left for Timer 3</td>
</tr>
<tr>
<td>4000012</td>
<td>3</td>
<td>Timer 4 Delay</td>
</tr>
<tr>
<td>4000015</td>
<td>3</td>
<td>Number of milliseconds left for Timer 4</td>
</tr>
<tr>
<td>4000018</td>
<td>6</td>
<td>Number of seconds since Aug. 02, 2013 09:00 PST</td>
</tr>
<tr>
<td>400001B</td>
<td>3</td>
<td>Number of processing ticks since processor start</td>
</tr>
</tbody>
</table>

### 2.3 Flag IO

This memory area contains the flag of the current instance of the running firmware. Its default is readable and writable however writes are ignored. The processor enforces a minimum readable setting.

### 2.4 Data Received

When the Data Received interrupt fires, this area has been filled in with network related traffic. No more data can be received until the value stored in the Data Received Size area is set to 0.
2.5 Data Received Size

This area is a 3 byte value storing the size of data received. If this value is non-zero then no more data can be received from the network.

2.6 Data Sent

This area is a buffer to write data to that is to be sent over the network. The data is not sent until the Data Sent Size value is specified.

2.7 Data Sent Size

This area is a 3 byte value storing the size of data being sent. When a value is written to this memory location the amount of data specified will be sent over the network. Upon completion the value is set to 0 and the Data Sent interrupt is fired to indicate success.

2.8 Shared Memory

This area of memory is an optional component that allows a processor to have a shared memory region with other processors on the same bus. If this area is detected on initialization then it will be marked Read/Write. Care must be taken in communicating with other processors as data may be overwritten.

2.9 NVRAM Memory

This area of memory is an optional component that allows a processor to have storage between a full power cycle. If this area is detected on initialization then it will be marked Read/Write.

2.10 Interrupt Pointers

Each interrupt has 3 bytes to indicate the area of memory to jump to upon the interrupt firing. If the value is 0 then the interrupt is not fired. It is possible for an interrupt to fire while another interrupt is processing so disabling and enabling interrupts is highly recommended to avoid conflicts.
2.11 Processor Identification

The last 128 bytes of memory are used for processor identification and information of supported functionality. Writes to this area are ignored with the exception of the Interrupt Stack Direction Flag.

<table>
<thead>
<tr>
<th>Memory Start</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>7FFFF80</td>
<td>Processor name</td>
</tr>
<tr>
<td>7FFFFA0</td>
<td>Processor version</td>
</tr>
<tr>
<td>7FFFFA3</td>
<td>Processor functionality flags</td>
</tr>
<tr>
<td>7FFFFA6</td>
<td>For future use</td>
</tr>
<tr>
<td>7FFFFF0</td>
<td>Interrupt stack direction flag</td>
</tr>
<tr>
<td>7FFFFF1</td>
<td>For future use</td>
</tr>
</tbody>
</table>

It is recommended to implementors that the processor version contains a Major, Minor, and Revision value, one value per byte entry.

The functionality flags has the following information:

<table>
<thead>
<tr>
<th>Functionality Flags</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000000000000</td>
<td>Interrupts are able to flip stack storage direction</td>
</tr>
<tr>
<td>X</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

The low bit of the interrupt stack direction flag dictates the direction the interrupt writes to the stack. A value of 0, the default, results in the interrupt subtracting 99 bytes from the stack pointer then storing all registers in the 99 byte buffer starting with register 0. A value of 1 results in the interrupt storing...
and incrementing the stack pointer starting with register 0. The interrupt return behaves in the opposite manner to restore the registers.
Chapter 3

Interrupts and Exceptions

When any interrupt is fired, all 32 general purpose registers and low 4 bits of the flags register are stored to the current stack before the processor begins executing the specified interrupt routine. Upon returning from an interrupt, all registers and low bits of the flag register are restored from the stack. The Disable Interrupts, DI, and Enable Interrupts, EI, instructions are used to temporarily disable an interrupt. Any interrupt with a value of 0 will not be called and ignored.

3.1 Timer 1 to 4 Interrupts

Each timer has an accuracy of 1 millisecond and can be configured through the Clock IO.

3.2 Invalid Instruction Exception

When an invalid instruction is detected this interrupt is fired. If this interrupt is disabled with DI, (the "Disable Interrupts" instruction,) or by having this value be 0 then the processor will halt.

3.3 Divide by 0 Exception

Division with a divisor of 0 will trigger this interrupt. Additionally, all other floating point exceptions will also trigger this interrupt.

3.4 Memory Exception

An attempt to read, write, or execute memory with invalid permission bits will cause this interrupt to trigger.
3.5 Data Received Interrupt

When data is received over the network this interrupt is fired.

3.6 Data Sent Interrupt

When data is fully sent over the network this interrupt is fired.

3.7 Exceptions

Upon an exception occurring, all registers are moved to the stack, R0 is set to the PC location that failed, R1 is set to one of the following IDs indicating the type of exception, and R2 is a value specific to the exception type. If an exception occurs while the interrupt handling the exception is still active then the processor will halt.

<table>
<thead>
<tr>
<th>Exception</th>
<th>ID</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Read</td>
<td>0</td>
<td></td>
<td>Address that failed to be read</td>
</tr>
<tr>
<td>Memory Write</td>
<td>1</td>
<td></td>
<td>Address that failed to be written</td>
</tr>
<tr>
<td>Memory Execute</td>
<td>2</td>
<td></td>
<td>Address that failed to execute</td>
</tr>
<tr>
<td>Invalid Instruction</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Floating Point Error</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Divide By 0</td>
<td>5</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

If the exception is disabled or the interrupt has no registered handler then the exception is ignored and the result of the operation that failed is undefined. The only case this is problematic is upon an instruction fault or execution in non-executable memory. If no exception is registered it will cause the CPU to halt due to not advancing the PC which would otherwise cause an infinite fault loop. If multiple faults can occur on the same instruction then only one fault will occur although no guarantee of which fault takes priority.
Chapter 4

Instruction Set

Unless specified otherwise, all math and immediate values are unsigned for integer arithmetic while all floating point math is signed. All rX values can reference a general purpose register from 0 to 31. Any time the format rX:rX+Y is seen, the instruction will work on registers rX through and including rX+Y based on the value of Y. If present, the UF field controls if the flags get updated for the instruction.
4.1 AD: Add

Format: AD rA, rB, rC

Purpose: Add two 27-bit integer registers together

Description: The 27-bit value in rC is added to the 27-bit value in rB, the result is placed in rA.

Operation: rA ← rB + rC

Flags affected: Z C 0 S

4.2 ADC: Add With Carry

Format: ADC rA, rB, rC

Purpose: Add two 27-bit integer registers together including the carry bit

Description: The 27-bit value in rC is added to the 27-bit value in rB, the carry bit from any previous operation is added to the result. The result is placed in rA.

Operation: rA ← rB + rC + Carry_Bit

Flags affected: Z C 0 S
4.3 ADCI: Add Immediate With Carry

Format: ADCI rA, rB, IMM

Purpose: Add a 7-bit immediate value to a 27-bit integer register including the carry bit.

Description: The 7-bit immediate value is added to the 27-bit value in rB, the carry bit from any previous operation is added to the result. The result is placed in rA.

Operation: \( rA \leftarrow rB + IMM + \text{Carry\_Bit} \)

Flags affected: Z C O S

4.4 ADCIM: Add Immediate Multi Reg With Carry

Format: ADCIM rA, rB, IMM

Purpose: Add a 7-bit immediate value to a 54-bit integer register including the carry bit.

Description: The 7-bit immediate value is added to the 54-bit value in rB:rB+1, the carry bit from any previous operation is added to the result. The result is placed in rA:rA+1.

Operation: \( rA:rA+1 \leftarrow rB:rB+1 + IMM + \text{Carry\_Bit} \)

Flags affected: Z C O S
4.5 ADCM: Add Multi Reg With Carry

| 0 100010 | rA | rB | rC | 0000 | UF |

**Format:** ADCM rA, rB, rC

**Purpose:** Add two 54-bit integer registers together including the carry bit

**Description:** The 54-bit value in rC:rC+1 is added to the 54-bit value in rB:rB+1, the carry bit from any previous operation is added to the result. The result is placed in rA:rA+1.

**Operation:** rA:rA+1 ← rB:rB+1 + rC:rC+1 + Carry_Bit

**Flags affected:** Z C 0 S

4.6 ADF: Add Floating Point

| 0 000001 | rA | rB | rC | 0000 | UF |

**Format:** ADF rA, rB, rC

**Purpose:** Add two 27-bit floating point registers together

**Description:** The 27-bit floating point value in rC is added to the 27-bit floating point value in rB, the result is placed in rA.

**Operation:** rA ← rB + rC

**Flags affected:** Z C 0 S

4.7 ADFM: Add Floating Point Multi Reg

| 0 000011 | rA | rB | rC | 0000 | UF |

**Format:** ADFM rA, rB, rC

**Purpose:** Add two 54-bit floating point registers together

**Description:** The 54-bit floating point value in rC:rC+1 is added to the 54-bit floating point value in rB:rB+1, the result is placed in rA:rA+1.

**Operation:** rA:rA+1 ← rB:rB+1 + rC:rC+1

**Flags affected:** Z C 0 S
4.8 ADI: Add Immediate

Format: ADI rA, rB, IMM

Purpose: Add a 7-bit immediate value to a 27-bit integer register

Description: The 7-bit immediate value is added to the 27-bit value in rB, the result is placed in rA.

Operation: rA ← rB + IMM

Flags affected: Z C 0 S

4.9 ADIM: Add Immediate Multi Reg

Format: ADIM rA, rB, IMM

Purpose: Add a 7-bit immediate value to a 54-bit integer register

Description: The 7-bit immediate value is added to the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: rA:rA+1 ← rB:rB+1 + IMM

Flags affected: Z C 0 S

4.10 ADM: Add Multi Reg

Format: ADM rA, rB, rC

Purpose: Add two 54-bit integer registers together

Description: The 54-bit value in rC:rC+1 is added to the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: rA:rA+1 ← rB:rB+1 + rC:rC+1

Flags affected: Z C 0 S
4.11 AN: And

Format: AN rA, rB, rC

Purpose: Bit-wise AND two 27-bit integer registers together

Description: The 27-bit value in rC is bit-wise AND to the 27-bit value in rB, the result is placed in rA.

Operation: \( rA \leftarrow rB \& rC \)

Flags affected: Z C O S

4.12 ANI: And Immediate

Format: ANI rA, rB, IMM

Purpose: Bit-wise AND a 27-bit integer register and 7-bit immediate together

Description: The 7-bit immediate value is bit-wise AND to the 27-bit value in rB, the result is placed in rA.

Operation: \( rA \leftarrow rB \& IMM \)

Flags affected: Z C O S

4.13 ANM: And Multi Reg

Format: ANM rA, rB, rC

Purpose: Bit-wise AND two 54-bit integer registers together

Description: The 54-bit value in rC:rC+1 is bit-wise AND to the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: \( rA:rA+1 \leftarrow rB:rB+1 \& rC:rC+1 \)

Flags affected: Z C O S
4.14  B: Branch Conditional

0  5  6  9  10  26
110000 Condition Offset

Format: Bcc Offset

Purpose: Conditional branch to an offset

Description: If the specified condition is true then the sign extended offset is added to the current program counter.

The condition table is as follows:

<table>
<thead>
<tr>
<th>cc</th>
<th>Value</th>
<th>Description</th>
<th>Flag State Checked</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0000</td>
<td>Not Equal / Not Zero</td>
<td>Z == 0</td>
</tr>
<tr>
<td>e</td>
<td>0001</td>
<td>Equal / Zero</td>
<td>Z == 1</td>
</tr>
<tr>
<td>l</td>
<td>0010</td>
<td>Less Than</td>
<td>C == 1 AND Z == 0</td>
</tr>
<tr>
<td>le</td>
<td>0011</td>
<td>Less Than or Equal</td>
<td>C == 1 OR Z == 1</td>
</tr>
<tr>
<td>g</td>
<td>0100</td>
<td>Greater Than</td>
<td>C == 0 AND Z == 0</td>
</tr>
<tr>
<td>ge</td>
<td>0101</td>
<td>Greater Than or Equal</td>
<td>C == 0 OR Z == 1</td>
</tr>
<tr>
<td>no</td>
<td>0110</td>
<td>Not Overflow</td>
<td>O == 0</td>
</tr>
<tr>
<td>o</td>
<td>0111</td>
<td>Overflow</td>
<td>O == 1</td>
</tr>
<tr>
<td>ns</td>
<td>1000</td>
<td>Not Signed</td>
<td>S == 0</td>
</tr>
<tr>
<td>s</td>
<td>1001</td>
<td>Signed</td>
<td>S == 1</td>
</tr>
<tr>
<td>sl</td>
<td>1010</td>
<td>Signed Less Than</td>
<td>S != 0</td>
</tr>
<tr>
<td>sle</td>
<td>1011</td>
<td>Signed Less Than or Equal</td>
<td>S != 0 OR Z == 1</td>
</tr>
<tr>
<td>sg</td>
<td>1100</td>
<td>Signed Greater Than</td>
<td>S == 0 AND Z == 0</td>
</tr>
<tr>
<td>sge</td>
<td>1101</td>
<td>Signed Greater Than or Equal</td>
<td>S == 0</td>
</tr>
<tr>
<td>1111</td>
<td></td>
<td>Always</td>
<td></td>
</tr>
</tbody>
</table>

Operation:

if Flags == Condition then
   PC += (signed)Offset

Flags affected: None
4.15  BF: Bit Flip

Format: BF rA, rB

Purpose: Bit flip a 27-bit register

Description: Invert all bits of the 27-bit rB register value and store the result in rA.

Operation: rA ← ~rB

Flags affected: Z C O S

4.16  BFM: Bit Flip Multi Reg

Format: BFM rA, rB

Purpose: Bit flip a 54-bit register

Description: Invert all bits of the 54-bit rB:rB+1 register value and store the result in rA:rA+1.

Operation: rA:rA+1 ← ~rB:rB+1

Flags affected: Z C O S
4.17 BR: Branch Register Conditional

Format: BRcc rA

Purpose: Conditional branch to a register

Description: If the specified condition is true then the value in rA is placed into the program counter.

The condition table is as follows:

<table>
<thead>
<tr>
<th>cc</th>
<th>Value</th>
<th>Description</th>
<th>Flag State Checked</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0000</td>
<td>Not Equal / Not Zero</td>
<td>Z == 0</td>
</tr>
<tr>
<td>e</td>
<td>0001</td>
<td>Equal / Zero</td>
<td>Z == 1</td>
</tr>
<tr>
<td>l</td>
<td>0010</td>
<td>Less Than</td>
<td>C == 1 AND Z == 0</td>
</tr>
<tr>
<td>le</td>
<td>0011</td>
<td>Less Than or Equal</td>
<td>C == 1 OR Z == 1</td>
</tr>
<tr>
<td>g</td>
<td>0100</td>
<td>Greater Than</td>
<td>C == 0 AND Z == 0</td>
</tr>
<tr>
<td>ge</td>
<td>0101</td>
<td>Greater Than or Equal</td>
<td>C == 0 OR Z == 1</td>
</tr>
<tr>
<td>no</td>
<td>0110</td>
<td>Not Overflow</td>
<td>O == 0</td>
</tr>
<tr>
<td>o</td>
<td>0111</td>
<td>Overflow</td>
<td>O == 1</td>
</tr>
<tr>
<td>ns</td>
<td>1000</td>
<td>Not Signed</td>
<td>S == 0</td>
</tr>
<tr>
<td>s</td>
<td>1001</td>
<td>Signed</td>
<td>S == 1</td>
</tr>
<tr>
<td>sl</td>
<td>1010</td>
<td>Signed Less Than</td>
<td>S != O</td>
</tr>
<tr>
<td>sle</td>
<td>1011</td>
<td>Signed Less Than or Equal</td>
<td>S != O OR Z == 1</td>
</tr>
<tr>
<td>sg</td>
<td>1100</td>
<td>Signed Greater Than</td>
<td>S == 0 OR Z == 1</td>
</tr>
<tr>
<td>sge</td>
<td>1101</td>
<td>Signed Greater Than or Equal</td>
<td>S == 0</td>
</tr>
<tr>
<td></td>
<td>1111</td>
<td>Always</td>
<td></td>
</tr>
</tbody>
</table>

Operation:

if Flags == Condition then
PC = rA

Flags affected: None
4.18  BRA: Branch Absolute

0 8 | 9 35
111000100 | Location

Format: BRA Location

Purpose: Branch to a set absolute location

Description: The program counter is set to the specified 27-bit location.

Operation: \( PC = Location \)

Flags affected: None

4.19  BRR: Branch Relative

0 8 | 9 35
111000000 | Offset

Format: BRR Offset

Purpose: Branch to a relative offset

Description: The 27-bit offset is added to the current program counter.

Operation: \( PC = PC + Offset \)

Flags affected: None
4.20  C: CALL CONDITIONAL

### 4.20  C: Call Conditional

- **Format:** Ccc Offset
- **Purpose:** Conditional call to an offset

**Description:** If the specified condition is true then the current program counter + 3 is saved into the RA register and the sign extended offset is added to the current program counter.

The condition table is as follows:

<table>
<thead>
<tr>
<th>cc</th>
<th>Value</th>
<th>Description</th>
<th>Flag State Checked</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0000</td>
<td>Not Equal / Not Zero</td>
<td>Z == 0</td>
</tr>
<tr>
<td>e</td>
<td>0001</td>
<td>Equal / Zero</td>
<td>Z == 1</td>
</tr>
<tr>
<td>l</td>
<td>0010</td>
<td>Less Than</td>
<td>C == 1 AND Z == 0</td>
</tr>
<tr>
<td>le</td>
<td>0011</td>
<td>Less Than or Equal</td>
<td>C == 1 OR Z == 1</td>
</tr>
<tr>
<td>g</td>
<td>0100</td>
<td>Greater Than</td>
<td>C == 0 AND Z == 0</td>
</tr>
<tr>
<td>ge</td>
<td>0101</td>
<td>Greater Than or Equal</td>
<td>C == 0 OR Z == 1</td>
</tr>
<tr>
<td>no</td>
<td>0110</td>
<td>Not Overflow</td>
<td>O == 0</td>
</tr>
<tr>
<td>o</td>
<td>0111</td>
<td>Overflow</td>
<td>O == 1</td>
</tr>
<tr>
<td>ns</td>
<td>1000</td>
<td>Not Signed</td>
<td>S == 0</td>
</tr>
<tr>
<td>s</td>
<td>1001</td>
<td>Signed</td>
<td>S == 1</td>
</tr>
<tr>
<td>sl</td>
<td>1010</td>
<td>Signed Less Than</td>
<td>S != 0</td>
</tr>
<tr>
<td>sle</td>
<td>1011</td>
<td>Signed Less Than or Equal</td>
<td>S != 0 OR Z == 1</td>
</tr>
<tr>
<td>sg</td>
<td>1100</td>
<td>Signed Greater Than</td>
<td>S == 0 AND Z == 0</td>
</tr>
<tr>
<td>sge</td>
<td>1101</td>
<td>Signed Greater Than or Equal</td>
<td>S == 0</td>
</tr>
<tr>
<td>1111</td>
<td>Always</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Operation:**

```plaintext
if Flags == Condition then
    RA = PC + 3
    PC += (signed)Offset
```

**Flags affected:** None
4.21 CAA: Call Absolute

Format: CAA Location

Purpose: Call to a set absolute location

Description: The current program counter + 4 is stored into RA and the program counter is set to the specified 27-bit location.

Operation:
- RA = PC + 4
- PC = Location

Flags affected: None

4.22 CAR: Call Relative

Format: CAR Offset

Purpose: Call to a relative offset

Description: The current program counter + 4 is stored into RA and the 27-bit offset is added to the current program counter.

Operation:
- RA = PC + 4
- PC = PC + Offset

Flags affected: None
### 4.23 CM: Compare

<table>
<thead>
<tr>
<th>0</th>
<th>7</th>
<th>8</th>
<th>12</th>
<th>13</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>10111000</td>
<td>rA</td>
<td>rB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** CM rA, rB

**Purpose:** Compare two registers

**Description:** The 27-bit values in rB is subtracted from rA. The flags are set appropriately.

**Operation:**

1. \( \text{FLAGS} = \text{FLAGS} \& \sim 0xF \)
2. 28-bit temp = rA - rB
3. if temp == 0 then
   - ZERO FLAG = 1
4. if temp & 0x4000000 then
   - SIGNED FLAG = 1
5. if (temp & 0x4000000) != (rA & 0x4000000) then
   - OVERFLOW FLAG = 1
6. if (temp & 0x8000000) then
   - CARRY FLAG = 1

**Flags affected:** Z S O C
4.24  CMF: Compare Floating Point

<table>
<thead>
<tr>
<th>0</th>
<th>7</th>
<th>8</th>
<th>12</th>
<th>13</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>10111010</td>
<td>rA</td>
<td>rB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Format: CMF rA, rB

Purpose: Compare two floating point registers

Description: The 27-bit floating point value in rB is subtracted from the floating point rA. The flags are set appropriately.

Operation:
```c
FLAGS = FLAGS & ~0xF
temp = rA - rB
if temp == 0.0 then
    ZERO FLAG = 1
if temp < 0.0 then
    SIGNED FLAG = 1
if (temp < 0.0) != (rA < 0.0) then
    OVERFLOW FLAG = 1
```

Flags affected: Z S O

4.25  CMFM: Compare Floating Point Multi Reg

<table>
<thead>
<tr>
<th>0</th>
<th>7</th>
<th>8</th>
<th>12</th>
<th>13</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>10111110</td>
<td>rA</td>
<td>rB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Format: CMFM rA, rB

Purpose: Compare two floating point registers

Description: The 54-bit floating point value in rB is subtracted from the floating point rA. The flags are set appropriately.

Operation:
```c
FLAGS = FLAGS & ~0xF
temp = rA:rA+1 - rB:rB+1
if temp == 0 then
    ZERO FLAG = 1
if temp < 0.0 then
    SIGNED FLAG = 1
if (temp < 0.0) != (rA:rA+1 < 0.0) then
    OVERFLOW FLAG = 1
```

Flags affected: Z S O
4.26 CMI: Compare Immediate

Format: CMI rA, IMM

Purpose: Compare a register and immediate value

Description: The sign extended immediate value is subtracted from the 27-bit rA. The flags are set appropriately.

Operation:

\[
\text{FLAGS} = \text{FLAGS} \& \sim 0xF \\
\text{28-bit temp} = rA - (\text{signed})\text{IMM} \\
\text{if temp} == 0 \text{ then} \\
\quad \text{ZERO FLAG} = 1 \\
\text{if temp} \& 0x4000000 \text{ then} \\
\quad \text{SIGNED FLAG} = 1 \\
\text{if (temp} \& 0x4000000) != (rA \& 0x4000000) \text{ then} \\
\quad \text{OVERFLOW FLAG} = 1 \\
\text{if (temp} \& 0x8000000) \text{ then} \\
\quad \text{CARRY FLAG} = 1
\]

Flags affected: Z S O C
4.27 CMIM: Compare Immediate Multi Reg

Format: CMIM rA, IMM

Purpose: Compare a register and immediate value

Description: The sign extended immediate value is subtracted from the 54-bit rA:rA+1. The flags are set appropriately.

Operation:

FLAGS = FLAGS & -0xF
55-bit temp = rA:rA+1 - (signed)IMM
if temp == 0 then
    ZERO FLAG = 1
if temp & 0x20000000000000 then
    SIGNED FLAG = 1
if ((temp & 0x20000000000000) != (rA:rA+1 & 0x20000000000000)) then
    OVERFLOW FLAG = 1
if (temp & 0x40000000000000) then
    CARRY FLAG = 1

Flags affected: Z S O C
4.28  CMM: Compare Multi Reg

Format: CMM rA, rB

Purpose: Compare two registers

Description: The 54-bit values in rB:rB+1 is subtracted from rA:rA+1. The flags are set appropriately.

Operation:

```plaintext
FLAGS = FLAGS & ~0xF
55-bit temp = rA:rA+1 - rB:rB+1
if temp == 0 then
    ZERO FLAG = 1
if temp & 0x20000000000000 then
    SIGNED FLAG = 1
if ((temp & 0x20000000000000) != (rA:rA+1 & 0x20000000000000)) then
    OVERFLOW FLAG = 1
if (temp & 0x40000000000000) then
    CARRY FLAG = 1
```

Flags affected: Z S O C
4.29 CR: Call Register Conditional

Format: CRcc rA

Purpose: Conditional call to a register

Description: If the specified condition is true then the current program counter + 2 is saved into the RA register and the program counter is set to the value in rA.

The condition table is as follows:

<table>
<thead>
<tr>
<th>cc</th>
<th>Value</th>
<th>Description</th>
<th>Flag State Checked</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0000</td>
<td>Not Equal / Not Zero</td>
<td>Z == 0</td>
</tr>
<tr>
<td>e</td>
<td>0001</td>
<td>Equal / Zero</td>
<td>Z == 1</td>
</tr>
<tr>
<td>l</td>
<td>0010</td>
<td>Less Than</td>
<td>C == 1 AND Z == 0</td>
</tr>
<tr>
<td>le</td>
<td>0011</td>
<td>Less Than or Equal</td>
<td>C == 1 OR Z == 1</td>
</tr>
<tr>
<td>g</td>
<td>0100</td>
<td>Greater Than</td>
<td>C == 0 AND Z == 0</td>
</tr>
<tr>
<td>ge</td>
<td>0101</td>
<td>Greater Than or Equal</td>
<td>C == 0 OR Z == 1</td>
</tr>
<tr>
<td>no</td>
<td>0110</td>
<td>Not Overflow</td>
<td>O == 0</td>
</tr>
<tr>
<td>o</td>
<td>0111</td>
<td>Overflow</td>
<td>O == 1</td>
</tr>
<tr>
<td>ns</td>
<td>1000</td>
<td>Not Signed</td>
<td>S == 0</td>
</tr>
<tr>
<td>s</td>
<td>1001</td>
<td>Signed</td>
<td>S == 1</td>
</tr>
<tr>
<td>sl</td>
<td>1010</td>
<td>Signed Less Than</td>
<td>S != O</td>
</tr>
<tr>
<td>sle</td>
<td>1011</td>
<td>Signed Less Than or Equal</td>
<td>S != O OR Z == 1</td>
</tr>
<tr>
<td>sg</td>
<td>1100</td>
<td>Signed Greater Than</td>
<td>S == 0 AND Z == 0</td>
</tr>
<tr>
<td>sge</td>
<td>1101</td>
<td>Signed Greater Than or Equal</td>
<td>S == 0</td>
</tr>
<tr>
<td>1111</td>
<td></td>
<td>Always</td>
<td></td>
</tr>
</tbody>
</table>

Operation:

if Flags == Condition then
RA = PC + 2
PC = rA

Flags affected: None
4.30  DBRK: Debug Break

Format: DBRK

Purpose: Debug break

Description: A special instruction for emulators to allow forcing a break in
the emulation. Normal execution will cause an illegal instruction fault.

Operation:

Flags affected: None

4.31  DI: Disable Interrupts

Format: DI rA

Purpose: Disable interrupts based on rA

Description: Disable interrupts based on the mask in rA.

Operation:  FLAGS = (FLAGS & 0x7FFE00F) | (((-rA << 4) & 0x1FF0)

Flags affected: None
4.32  DMT: Direct Memory Transfer

\[ \begin{array}{c|c|c|c|c|c|c|c} 0 & 6 & 7 & 11 & 12 & 16 & 17 & 21 & 22 & 26 \\ 0110100 & rA & rB & rC & 00000 \\ \end{array} \]

**Format:** DMT rA, rB, rC

**Purpose:** Directly transfer memory between locations

**Description:** Directly transfer \( rC \) bytes of memory from location pointed to by \( rB \) to memory location \( rA \).

**Operation:**
\[
P = 0 \\
\text{while}(P < rC) \\
\text{(byte)}[rA + P] = \text{(byte)}[rB + P] \\
P = P + 1
\]

**Flags affected:** None

4.33  DV: Divide

\[ \begin{array}{c|c|c|c|c|c|c|c} 0 & 6 & 7 & 11 & 12 & 16 & 17 & 21 & 22 & 25 & 26 \\ 0001100 & rA & rB & rC & 0000 & UF \\ \end{array} \]

**Format:** DV rA, rB, rC

**Purpose:** Divide two 27-bit integer registers

**Description:** The 27-bit value in \( rB \) is divided with the 27-bit value in \( rC \), the result is placed in \( rA \).

**Operation:** \( rA \leftarrow rB / rC \)

**Flags affected:** Z C O S
4.34  **DVF: Divide Floating Point**

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>21</th>
<th>22</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001101</td>
<td>rA</td>
<td>rB</td>
<td>rC</td>
<td>0000</td>
<td>UF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** DVF rA, rB, rC

**Purpose:** Divide two 27-bit floating point registers

**Description:** The 27-bit floating point value in rB is divided with the 27-bit floating point value in rC, the result is placed in rA.

**Operation:** \( rA \leftarrow \frac{rB}{rC} \)

**Flags affected:** Z C O S

---

4.35  **DVFM: Divide Floating Point Multi Reg**

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>21</th>
<th>22</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001111</td>
<td>rA</td>
<td>rB</td>
<td>rC</td>
<td>0000</td>
<td>UF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** DVFM rA, rB, rC

**Purpose:** Divide two 54-bit floating point registers

**Description:** The 54-bit floating point value in rB:rB+1 is divided with the 54-bit floating point value in rC:rC+1, the result is placed in rA:rA+1.

**Operation:** \( rA:rA+1 \leftarrow \frac{rB:rB+1}{rC:rC+1} \)

**Flags affected:** Z C O S

---

4.36  **DVI: Divide Immediate**

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001100</td>
<td>rA</td>
<td>rB</td>
<td>Immediate</td>
<td>01</td>
<td>UF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** DVI rA, rB, IMM

**Purpose:** Divide a 27-bit integer register by a 7-bit immediate

**Description:** The 27-bit value in rB is divided with the 7-bit immediate value, the result is placed in rA.

**Operation:** \( rA \leftarrow \frac{rB}{IMM} \)

**Flags affected:** Z C O S
4.37  DVIM: Divide Immediate Multi Reg

Format: DVIM rA, rB, IMM

Purpose: Divide a 54-bit integer register by a 7-bit immediate

Description: The 54-bit value in rB:rB+1 is divided with the 7-bit immediate value, the result is placed in rA:rA+1.

Operation: rA:rA+1 ← rB:rB+1 / IMM

Flags affected: Z C O S

4.38  DVIS: Divide Immediate Signed

Format: DVIS rA, rB, IMM

Purpose: Divide a signed 27-bit integer register by a signed 7-bit immediate

Description: The signed 27-bit value in rB is divided with the signed 7-bit immediate value, the result is placed in rA.

Operation: rA ← (signed)rB / (signed)IMM

Flags affected: Z C O S

4.39  DVISM: Divide Immediate Signed Multi Reg

Format: DVISM rA, rB, IMM

Purpose: Divide a signed 54-bit integer register by a signed 7-bit immediate

Description: The signed 54-bit value in rB:rB+1 is divided with the signed 7-bit immediate value, the result is placed in rA:rA+1.

Operation: rA:rA+1 ← (signed)rB:rB+1 / (signed)IMM

Flags affected: Z C O S
4.40  DVM: Divide Multi Reg

Format: DVM rA, rB, rC

Purpose: Divide two 54-bit integer registers

Description: The 54-bit value in rB:rB+1 is divided with the 54-bit value in rC:rC+1, the result is placed in rA:rA+1.

Operation: rA:rA+1 ← rB:rB+1 / rC:rC+1

Flags affected: Z C O S

4.41  DVS: Divide Signed

Format: DVS rA, rB, rC

Purpose: Divide two signed 27-bit integer registers

Description: The signed 27-bit value in rB is divided with the signed 27-bit value in rC, the result is placed in rA.

Operation: rA ← (signed)rB / (signed)rC

Flags affected: Z C O S

4.42  DVSM: Divide Signed Multi Reg

Format: DVSM rA, rB, rC

Purpose: Divide two 54-bit integer registers

Description: The 54-bit value in rB:rB+1 is divided with the 54-bit value in rC:rC+1, the result is placed in rA:rA+1.

Operation: rA:rA+1 ← (signed)rB:rB+1 / (signed)rC:rC+1

Flags affected: Z C O S
4.43 EI: Enable Interrupts

Format: EI rA

Purpose: Enable interrupts based on rA

Description: Enable interrupts based on the mask in rA.

Operation: \( \text{FLAGS} = (\text{FLAGS} \& 0x7FFE00F) \mid ((rA << 4) \& 0x1FF0) \)

Flags affected: None

4.44 FTI: Float to Integer

Format: FTI rA, rB

Purpose: Convert a 27-bit float to integer

Description: The 27-bit float in rB is converted to a 27-bit integer value and stored in rA.

Operation: \( rA \leftarrow \text{int}(rB) \)

Flags affected: Z C O S

4.45 FTIM: Float to Integer Multi Reg

Format: FTIM rA, rB

Purpose: Convert a 54-bit float to integer

Description: The 54-bit float in rB:rB+1 is converted to a 54-bit integer value and stored in rA:rA+1.

Operation: \( rA:rA+1 \leftarrow \text{int}(rB:rB+1) \)

Flags affected: Z C O S
4.46  HT: Halt

0  17
101000000011000000

Format: HT

Purpose: Halt the processor

Description: Disables all interrupts and stops the processor from responding to any more instructions.

Operation:

Flags affected: None

4.47  IR: Interrupt Return

0  17
101000000001000000

Format: IR

Purpose: Return from an interrupt

Description: Return from an interrupt routine.

Operation:

if Stack_Direction_Flag then
    SP -= 0x63
    R0:R31 ← [SP:SP+0x60]
    FLAGS ← [SP+0x60:SP+0x63]
if not Stack_Direction_Flag then
    SP += 0x63

Flags affected: Z C O S
4.48 ITF: Integer to Float

Format: ITF rA, rB

Purpose: Convert a 27-bit integer to float

Description: The 27-bit integer in rB is converted to a 27-bit float value and stored in rA.

Operation: \( rA \leftarrow \text{float}(rB) \)

Flags affected: Z C 0 S

4.49 ITFM: Integer to Float Multi Reg

Format: ITFM rA, rB

Purpose: Convert a 54-bit integer to float

Description: The 54-bit integer in rB:rB+1 is converted to a 54-bit float value and stored in rA:rA+1.

Operation: \( rA:rA+1 \leftarrow \text{float}(rB:rB+1) \)

Flags affected: Z C 0 S
4.50  **LDS: Load Single**

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>50</th>
<th>51</th>
<th>53</th>
</tr>
</thead>
<tbody>
<tr>
<td>rA</td>
<td>rB</td>
<td>Register Count</td>
<td>Adjust rB</td>
<td>Memory Offset</td>
<td>000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** LDS\_m rA, [rB + Offset, RegCount]

**Purpose:** Load a single byte from memory

**Description:** The value in rB is added to Offset and a single byte per register is loaded into the low 9 bits of rA to rA+RegCount. The top 18 bits are set to zero for each affected register.

\( m \) refers to the Adjust RB value which can have the following designations to indicate the mode:

<table>
<thead>
<tr>
<th>( m )</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>rB is not adjusted after load</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>rB has the number of bytes read added to the start value after all data is read</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>rB has the number of bytes to read subtracted from the start value after all data is read</td>
</tr>
</tbody>
</table>

**Operation:**

1. `StartReg = rA`
2. `RegCount = RegCount + 1`
3. `CurCount = RegCount`
4. `Temp = rB`
5. `TempPC = PC`
6. if `Mode` is 2 then
   - `Temp = Temp - CurCount`
7. `MemLocation = (Temp + Offset)`
8. While `CurCount` is not 0
   - `Registers[StartReg] = Memory[MemLocation]`
   - `MemLocation += 1`
   - `StartReg = (StartReg + 1) \% 32`
   - `CurCount = CurCount - 1`
9. if `Mode` is 1 then
   - `rB = rB + RegCount`
10. if `Mode` is 2 then
    - `rB = Temp`
11. `PC = TempPC`

**Flags affected:** *None*
4.51 LDT: Load Tri

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>50</th>
<th>51</th>
<th>53</th>
</tr>
</thead>
<tbody>
<tr>
<td>1010110</td>
<td>rA</td>
<td>rB</td>
<td>Register Count</td>
<td>Adjust rB</td>
<td>Memory Offset</td>
<td>000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** LDTm rA, [rB + Offset, RegCount]

**Purpose:** Load three bytes from memory

**Description:** The value in rB is added to Offset and three bytes per register are loaded into rA to rA+RegCount.

m refers to the Adjust RB value which can have the following designations to indicate the mode:

<table>
<thead>
<tr>
<th>m</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rB is not adjusted after load</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>rB has the number of bytes read added to the start value after all data is read</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>rB has the number of bytes to read subtracted from the start value after all data is read</td>
</tr>
</tbody>
</table>

**Operation:**

StartReg = rA
RegCount = RegCount + 1
CurCount = RegCount
Temp = rB
TempPC = PC

if Mode is 2 then
    Temp = Temp - (CurCount * 3)
MemLocation = (Temp + Offset)
While CurCount is not 0
    Registers[StartReg] =
        (Memory[MemLocation] << 9) |
        (Memory[(MemLocation + 1)] << 18) |
        Memory[(MemLocation + 2)]
    MemLocation += 3
    StartReg = (StartReg + 1) % 32
    CurCount = CurCount - 1
if Mode is 1 then
    rB = rB + (RegCount * 3)
if Mode is 2 then
    rB = Temp
PC = TempPC

**Flags affected:** None
4.52. **LDW: LOAD WORD**

**Format:** \( \text{LDW}_m \ rA, \ [rB + \text{Offset}, \text{RegCount}] \)

**Purpose:** Load two bytes from memory

**Description:** The value in \( rB \) is added to Offset and two bytes per register are loaded into the low 18 bits of \( rA \) to \( rA + \text{RegCount} \). The top 9 bits are set to zero for each affected register.

\( m \) refers to the Adjust RB value which can have the following designations to indicate the mode:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( rB ) is not adjusted after load</td>
</tr>
<tr>
<td>I</td>
<td>( rB ) has the number of bytes read added to the start value after all data is read</td>
</tr>
<tr>
<td>D</td>
<td>( rB ) has the number of bytes to read subtracted from the start value after all data is read</td>
</tr>
</tbody>
</table>

**Operation:**

- \( \text{StartReg} = rA \)
- \( \text{RegCount} = \text{RegCount} + 1 \)
- \( \text{CurCount} = \text{RegCount} \)
- \( \text{Temp} = rB \)
- \( \text{TempPC} = \text{PC} \)
- if Mode is 2 then 
  - \( \text{Temp} = \text{Temp} - (\text{CurCount} \times 2) \)
  - \( \text{MemLocation} = (\text{Temp} + \text{Offset}) \)
  - While CurCount is not 0 
    - \( \text{Registers}[\text{StartReg}] = \)
      - \( (\text{Memory}[\text{MemLocation}] \ll 9) | \text{Memory}(\text{MemLocation} + 1) \)
    - \( \text{MemLocation} += 2 \)
    - \( \text{StartReg} = (\text{StartReg} + 1) \mod 32 \)
    - \( \text{CurCount} = \text{CurCount} - 1 \)
- if Mode is 1 then
  - \( rB = rB + (\text{RegCount} \times 2) \)
- if Mode is 2 then
  - \( rB = \text{Temp} \)
  - \( \text{PC} = \text{TempPC} \)

**Flags affected:** None
4.53  MD: Modulus

Format: MD rA, rB, rC

Purpose: Access the remainder of dividing two 27-bit integer registers

Description: The 27-bit value in rB is divided with the 27-bit value in rC, the remainder of the division is placed in rA.

Operation: rA ← rB % rC

Flags affected: Z C O S

4.54  MDF: Modulus Floating Point

Format: MDF rA, rB, rC

Purpose: Access the remainder of dividing two 27-bit floating point registers

Description: The 27-bit floating point value in rB is divided with the 27-bit floating point value in rC, the remainder of the division is placed in rA.

Operation: rA ← rB % rC

Flags affected: Z C O S

4.55  MDFM: Modulus Floating Point Multi Reg

Format: MDFM rA, rB, rC

Purpose: Access the remainder of dividing two 54-bit floating point registers

Description: The 54-bit floating point value in rB:rB+1 is divided with the 54-bit floating point value in rC:rC+1, the remainder of the division is placed in rA:rA+1.

Operation: rA:rA+1 ← rB:rB+1 % rC:rC+1

Flags affected: Z C O S
4.56  MDI: Modulus Immediate

```
0  6 |  7 11 | 12 16 | 17 | 23 | 24 | 25 | 26
0010000 | rA | rB | Immediate | 01 | UF
```

**Format:** MDI rA, rB, IMM

**Purpose:** Access the remainder of dividing a 27-bit integer register by a 7-bit immediate

**Description:** The 27-bit value in rB is divided with a 7-bit immediate value, the remainder of the division is placed in rA.

**Operation:** rA ← rB % IMM

**Flags affected:** Z C O S

---

4.57  MDIM: Modulus Immediate Multi Reg

```
0  6 |  7 11 | 12 16 | 17 | 23 | 24 | 25 | 26
0010010 | rA | rB | Immediate | 01 | UF
```

**Format:** MDIM rA, rB, IMM

**Purpose:** Access the remainder of dividing a 54-bit integer register by a 7-bit immediate

**Description:** The 54-bit value in rB:rB+1 is divided with the 7-bit immediate value, the remainder of the division is placed in rA:rA+1.

**Operation:** rA:rA+1 ← rB:rB+1 % IMM

**Flags affected:** Z C O S
4.58 **MDIS: Modulus Immediate Signed**

```
0  6  7  11  12  16  17  23  24  25  26
0010000  rA  rB  Immediate  11  UF
```

**Format:** MDIS rA, rB, IMM

**Purpose:** Access the remainder of dividing a signed 27-bit integer register by a signed 7-bit immediate

**Description:** The signed 27-bit value in rB is divided with a signed 7-bit immediate value, the remainder of the division is placed in rA.

**Operation:** \( rA \leftarrow (\text{signed})rB \mod (\text{signed})\text{IMM} \)

**Flags affected:** Z C O S

---

4.59 **MDISM: Modulus Immediate Signed Multi Reg**

```
0  6  7  11  12  16  17  23  24  25  26
0010010  rA  rB  Immediate  11  UF
```

**Format:** MDISM rA, rB, IMM

**Purpose:** Access the remainder of dividing a signed 54-bit integer register by a signed 7-bit immediate

**Description:** The signed 54-bit value in rB:rB+1 is divided with the signed 7-bit immediate value, the remainder of the division is placed in rA:rA+1.

**Operation:** \( rA:rA+1 \leftarrow (\text{signed})rB:rB+1 \mod (\text{signed})\text{IMM} \)

**Flags affected:** Z C O S
4.60  **MDM: Modulus Multi Reg**

Format:  MDM rA, rB, rC

**Purpose:** Access the remainder of dividing two 54-bit integer registers

**Description:** The 54-bit value in rB:rB+1 is divided with the 54-bit value in rC:rC+1, the remainder of the division is placed in rA:rA+1.

**Operation:**  \( rA:rA+1 \leftarrow rB:rB+1 \% rC:rC+1 \)

**Flags affected:** Z C O S

4.61  **MDS: Modulus Signed**

Format:  MDS rA, rB, rC

**Purpose:** Access the remainder of dividing two signed 27-bit integer registers

**Description:** The signed 27-bit value in rB is divided with the signed 27-bit value in rC, the remainder of the division is placed in rA.

**Operation:**  \( rA \leftarrow (\text{signed})rB \% (\text{signed})rC \)

**Flags affected:** Z C O S

4.62  **MDSM: Modulus Signed Multi Reg**

Format:  MDSM rA, rB, rC

**Purpose:** Access the remainder of dividing two signed 54-bit integer registers

**Description:** The signed 54-bit value in rB:rB+1 is divided with the signed 54-bit value in rC:rC+1, the remainder of the division is placed in rA:rA+1.

**Operation:**  \( rA:rA+1 \leftarrow (\text{signed})rB:rB+1 \% (\text{signed})rC:rC+1 \)

**Flags affected:** Z C O S
4.63  **MH: Move High**

0 4 5 9 10 26
10001 rA Immediate

**Format:** MH rA, IMM

**Purpose:** Move an immediate value to the high bits of a register

**Description:** The high 17 bits of a 27-bit value in rA is set to the specified immediate value.

**Operation:** rA ← (IMM << 10) | (rA & 0x3FF)

**Flags affected:** Z C 0 S

4.64  **ML: Move Low**

0 4 5 9 10 26
10010 rA Immediate

**Format:** ML rA, IMM

**Purpose:** Move a 17-bit immediate value to the register

**Description:** A 27-bit rA is set to the specified 17-bit immediate value. The high bits are zero’d out.

**Operation:** rA ← IMM

**Flags affected:** Z C 0 S

4.65  **MS: Move Low Signed**

0 4 5 9 10 26
10011 rA Immediate

**Format:** MS rA, IMM

**Purpose:** Move a signed 17-bit immediate value to the register

**Description:** A 27-bit rA is set to the specified signed 17-bit immediate value. The high bits are set based on the signed bit of the immediate value specified.

**Operation:** rA ← (signed)IMM

**Flags affected:** Z C 0 S
4.66 MU: Multiply

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>21</th>
<th>22</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>rA</td>
<td>rB</td>
<td>rC</td>
<td>0000</td>
<td>UF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** MU rA, rB, rC

**Purpose:** Multiply two 27-bit integer registers together

**Description:** The 27-bit value in rC is multiplied with the 27-bit value in rB, the result is placed in rA.

**Operation:** \( rA \leftarrow rB \times rC \)

**Flags affected:** Z, C, O, S

4.67 MUF: Multiply Floating Point

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>21</th>
<th>22</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>rA</td>
<td>rB</td>
<td>rC</td>
<td>0000</td>
<td>UF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** MUF rA, rB, rC

**Purpose:** Multiply two 27-bit floating point registers together

**Description:** The 27-bit floating point value in rC is multiplied with the 27-bit floating point value in rB, the result is placed in rA.

**Operation:** \( rA \leftarrow rB \times rC \)

**Flags affected:** Z, C, O, S

4.68 MUFM: Multiply Floating Point Multi Reg

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>21</th>
<th>22</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>rA</td>
<td>rB</td>
<td>rC</td>
<td>0000</td>
<td>UF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** MUFM rA, rB, rC

**Purpose:** Multiply two 54-bit floating point registers together

**Description:** The 54-bit floating point value in rC:rC+1 is multiplied with the 54-bit floating point value in rB:rB+1, the result is placed in rA:rA+1.

**Operation:** \( rA:rA+1 \leftarrow rB:rB+1 \times rC:rC+1 \)

**Flags affected:** Z, C, O, S
4.69  MUI: Multiply Immediate

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001000</td>
<td>rA</td>
<td>rB</td>
<td>Immediate</td>
<td>01</td>
<td>UF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** MUI rA, rB, IMM

**Purpose:** Multiply a 27-bit integer register by a 7-bit immediate

**Description:** The 7-bit immediate value is multiplied with the 27-bit value in rB, the result is placed in rA.

**Operation:** rA ← rB * IMM

**Flags affected:** Z C O S

4.70  MUIM: Multiply Immediate Multi Reg

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001010</td>
<td>rA</td>
<td>rB</td>
<td>Immediate</td>
<td>01</td>
<td>UF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** MUIM rA, rB, IMM

**Purpose:** Multiply a 54-bit integer register by a 7-bit immediate

**Description:** The 7-bit immediate value is multiplied with the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

**Operation:** rA:rA+1 ← rB:rB+1 * IMM

**Flags affected:** Z C O S

4.71  MUIS: Multiply Immediate Signed

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001000</td>
<td>rA</td>
<td>rB</td>
<td>Immediate</td>
<td>11</td>
<td>UF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** MUIS rA, rB, IMM

**Purpose:** Multiply a signed 27-bit integer register by a signed 7-bit immediate

**Description:** The signed 7-bit immediate value is multiplied with the signed 27-bit value in rB, the result is placed in rA.

**Operation:** rA ← (signed)rB * (signed)IMM

**Flags affected:** Z C O S
4.72 MUISM: Multiply Immediate Signed Multi Reg

Format: MUISM rA, rB, IMM

Purpose: Multiply a signed 54-bit integer register by a signed 7-bit immediate

Description: The signed 7-bit immediate value is multiplied with the signed 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: rA:rA+1 ← (signed)rB:rB+1 * (signed)IMM

Flags affected: Z C O S

4.73 MUM: Multiply Multi Reg

Format: MUM rA, rB, rC

Purpose: Multiply two 54-bit integer registers together

Description: The 54-bit value in rC:rC+1 is multiplied with the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: rA:rA+1 ← rB:rB+1 * rC:rC+1

Flags affected: Z C O S

4.74 MUS: Multiply Signed

Format: MUS rA, rB, rC

Purpose: Multiply two signed 27-bit integer registers together

Description: The signed 27-bit value in rC is multiplied with the signed 27-bit value in rB, the result is placed in rA.

Operation: rA ← (signed)rB * (signed)rC

Flags affected: Z C O S
4.75  MUSM: Multiply Signed Multi Reg

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>21</th>
<th>22</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001010</td>
<td>rA</td>
<td>rB</td>
<td>rC</td>
<td>0010</td>
<td>UF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** MUSM rA, rB, rC

**Purpose:** Multiply two 54-bit integer registers together

**Description:** The signed 54-bit value in rC:rC+1 is multiplied with the signed 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

**Operation:** rA:rA+1 ← (signed)rB:rB+1 * (signed)rC:rC+1

**Flags affected:** Z C 0 S

4.76  NG: Negate

<table>
<thead>
<tr>
<th>0</th>
<th>8</th>
<th>9</th>
<th>13</th>
<th>14</th>
<th>18</th>
<th>19</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>101001100</td>
<td>rA</td>
<td>rB</td>
<td>0000000</td>
<td>UF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** NG rA, rB

**Purpose:** Negate a 27-bit register

**Description:** Negate the 27-bit rB register value and store the result in rA.

**Operation:** rA ← -rB

**Flags affected:** Z C 0 S

4.77  NGF: Negate Floating Point

<table>
<thead>
<tr>
<th>0</th>
<th>8</th>
<th>9</th>
<th>13</th>
<th>14</th>
<th>18</th>
<th>19</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>101001101</td>
<td>rA</td>
<td>rB</td>
<td>0000000</td>
<td>UF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** NGF rA, rB

**Purpose:** Negate a 27-bit floating-point register

**Description:** Negate the 27-bit rB floating-point register value and store the result in rA.

**Operation:** rA ← -rB

**Flags affected:** Z S
4.78  NGFM: Negate Floating Point Multi Reg

Format: NGFM rA, rB

Purpose: Negate a 54-bit floating-point register

Description: Negate the 54-bit rB:rB+1 floating-point register value and store the result in rA:rA+1.

Operation: rA:rA+1 ← -rB:rB+1

Flags affected: Z S

4.79  NGM: Negate Multi Reg

Format: NGM rA, rB

Purpose: Negate a 54-bit register

Description: Negate the 54-bit rB:rB+1 register value and store the result in rA:rA+1.

Operation: rA:rA+1 ← -rB:rB+1

Flags affected: Z C O S

4.80  NT: Not

Format: NT rA, rB

Purpose: Bit test a 27-bit register

Description: Test all bits of the 27-bit rB register value. If any bits are set then rA is set to 0. If all bits are off then rA is set to 1.

Operation: rA ← !rB

Flags affected: Z C O S
4.81  NTM: Not Multi Reg

0 8 9 13 14 18 19 25 26
101001110 rA rB 0100000 UF

Format: NTM rA, rB

Purpose: Bit test a 54-bit register

Description: Test all bits of the 54-bit rB:rB+1 register value. If any bits are set then rA:rA+1 is set to 0. If all bits are off then rA:rA+1 is set to 1.

Operation: rA:rA+1 ← !rB:rB+1

Flags affected: Z C O S

4.82  OR: Or

0 6 7 11 12 16 17 21 22 25 26
0011000 rA rB rC 0000 UF

Format: OR rA, rB, rC

Purpose: Bit-wise OR two 27-bit integer registers together

Description: The 27-bit value in rC is bit-wise OR to the 27-bit value in rB, the result is placed in rA.

Operation: rA ← rB | rC

Flags affected: Z C O S

4.83  ORI: Or Immediate

0 6 7 11 12 16 17 23 24 25 26
0011000 rA rB Immediate 01 UF

Format: ORI rA, rB, IMM

Purpose: Bit-wise OR a 27-bit integer register and 7-bit immediate together

Description: The 7-bit immediate value is bit-wise OR to the 27-bit value in rB, the result is placed in rA.

Operation: rA ← rB | IMM

Flags affected: Z C O S
4.84 ORM: Or Multi Reg

Format: ORM rA, rB, rC

Purpose: Bit-wise OR two 54-bit integer registers together

Description: The 54-bit value in rC:rC+1 is bit-wise OR to the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: rA:rA+1 ← rB:rB+1 | rC:rC+1

Flags affected: Z C 0 S

4.85 RE: Return

Format: RE

Purpose: To return from a call

Description: To return from a call.

Operation: PC ← RETADDR

Flags affected: None

4.86 RF: Read Flags

Format: RF rA

Purpose: Read flags into the rA register

Description: Set rA to the value in the flag register.

Operation: rA = FLAGS

Flags affected: None
4.87  **RL: Rotate Left**

Format: RL rA, rB, rC

**Purpose:** Rotate left a 27-bit integer

**Description:** The 27-bit value in rB is left rotated the number of bits specified in rC. The result is placed in rA.

**Operation:** \( rA \leftarrow (rB \ll rC) \mid (rB \gg (27-rC)) \)

**Flags affected:** Z C 0 S

4.88  **RLI: Rotate Left Immediate**

Format: RLI rA, rB, IMM

**Purpose:** Rotate left a 27-bit integer by a 7-bit immediate value

**Description:** The 27-bit value in rB is left rotated by the 7-bit immediate value. The result is placed in rA.

**Operation:** \( rA \leftarrow (rB \ll IMM) \mid (rB \gg (27-IMM)) \)

**Flags affected:** Z C 0 S

4.89  **RLIM: Rotate Left Immediate Multi Reg**

Format: RLIM rA, rB, IMM

**Purpose:** Rotate left a 54-bit integer by a 7-bit immediate value

**Description:** The 54-bit value in rB:rB+1 is left rotated by the 7-bit immediate value. The result is placed in rA:rA+1.

**Operation:** \( rA:rA+1 \leftarrow (rB:rB+1 \ll IMM) \mid (rB:rB+1 \gg (54-IMM)) \)

**Flags affected:** Z C 0 S
4.90  **RLM: Rotate Left Multi Reg**

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>21</th>
<th>22</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>rA</td>
<td>rB</td>
<td>rC</td>
<td>0000</td>
<td>UF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** RLM rA, rB, rC

**Purpose:** Rotate left a 54-bit integer

**Description:** The 54-bit value in rB:rB+1 is left rotated the number of bits specified in rC. The result is placed in rA:rA+1.

**Operation:**

\[
\text{rA}:\text{rA+1} \leftarrow (\text{rB}:\text{rB+1} \ll \text{rC}) \mid (\text{rB}:\text{rB+1} \gg (54-\text{rC}))
\]

**Flags affected:** Z C O S
4.91 RMP: Read Memory Protection

Format: RMP rA, rB

Purpose: Read the memory protections for a page of memory

Description: rA is set to the page protection flags of the pages of the specified memory range starting at the memory location specified by rA and moving forward rB pages. A set zero flag indicates success, upon failure rA holds the error reason. Every 2 bits indicates the memory protection state of a page. Only 13 page statuses can be returned at a time.

Errors:

<table>
<thead>
<tr>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Memory start is not page aligned</td>
</tr>
<tr>
<td>1</td>
<td>Too many pages specified</td>
</tr>
</tbody>
</table>

Flags:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No access</td>
</tr>
<tr>
<td>1</td>
<td>Read</td>
</tr>
<tr>
<td>2</td>
<td>Read/Write</td>
</tr>
<tr>
<td>3</td>
<td>Read/Execute</td>
</tr>
</tbody>
</table>

Operation:

\[ rA \leftarrow \text{Memory\_Page\_Flags[rA]:Memory\_Page\_Flags[rA+(rB*1024)]} \]

Flags affected: Z
4.92  RND: Random

Format: RND rA

Purpose: Generate a random value into rA

Description: A random value is placed into the 27-bit rA register.

Operation: \( rA \leftarrow <\text{random value}> \)

Flags affected: Z C O S

4.93  RNDM: Random Multi Reg

Format: RNDM rA

Purpose: Generate a random value into rA:rA+1

Description: A random value is generated and placed into the 54-bit rA:rA+1 register.

Operation: \( rA:rA+1 \leftarrow <\text{random value}> \)

Flags affected: Z C O S

4.94  RR: Rotate Right

Format: RR rA, rB, rC

Purpose: Rotate right a 27-bit integer

Description: The 27-bit value in rB is right rotated the number of bits specified in rC. The result is placed in rA.

Operation: \( rA \leftarrow (rB \gg rC) \mid (rB \ll (27-rC)) \)

Flags affected: Z C O S
4.95  RRI: Rotate Right Immediate

Format: RRI rA, rB, IMM

Purpose: Rotate right a 27-bit integer by a 7-bit immediate value

Description: The 27-bit value in rB is right rotated by the 7-bit immediate value. The result is placed in rA.

Operation: rA ← (rB >> IMM) | (rB << (27-IMM))

Flags affected: Z C O S

4.96  RRIM: Rotate Right Immediate Multi Reg

Format: RRIM rA, rB, rC

Purpose: Rotate right a 54-bit integer by an immediate value

Description: The 54-bit value in rB:rB+1 is right rotated by the 7-bit immediate value. The result is placed in rA:rA+1.

Operation: rA:rA+1 ← (rB:rB+1 >> IMM) | (rB:rB+1 << (54-IMM))

Flags affected: Z C O S

4.97  RRM: Rotate Right Multi Reg

Format: RRM rA, rB, rC

Purpose: Rotate right a 54-bit integer

Description: The 54-bit value in rB:rB+1 is right rotated the number of bits specified in rC. The result is placed in rA:rA+1.

Operation: rA:rA+1 ← (rB:rB+1 >> rC) | (rB:rB+1 << (54-rC))

Flags affected: Z C O S
4.98 SA: Shift Arithmetic Right

\[ \text{Format: } \text{SA } rA, \ rB, \ rC \]

**Purpose:** Shift right arithmetic a 27-bit integer

**Description:** The 27-bit value in \( rB \) is right arithmetic shifted the number of bits specified in \( rC \). The result is placed in \( rA \).

**Operation:** \( rA \leftarrow (\text{signed})rB \gg rC \)

**Flags affected:** Z C O S

4.99 SAI: Shift Arithmetic Right Immediate

\[ \text{Format: } \text{SAI } rA, \ rB, \ \text{Immediate} \]

**Purpose:** Shift right arithmetic a 27-bit integer by a 7-bit immediate value

**Description:** The 27-bit value in \( rB \) is right arithmetic shifted by the 7-bit immediate value. The result is placed in \( rA \).

**Operation:** \( rA \leftarrow (\text{signed})rB \gg \text{IMM} \)

**Flags affected:** Z C O S

4.100 SAIM: Shift Arithmetic Right Immediate Multi Reg

\[ \text{Format: } \text{SAIM } rA, \ rB, \ \text{Immediate} \]

**Purpose:** Shift right arithmetic a 54-bit integer by a 7-bit immediate value

**Description:** The 54-bit value in \( rB:rB+1 \) is right arithmetic shifted by the 7-bit immediate value. The result is placed in \( rA:rA+1 \).

**Operation:** \( rA:rA+1 \leftarrow (\text{signed})rB:rB+1 \gg \text{IMM} \)

**Flags affected:** Z C O S
4.101 SAM: Shift Arithmetic Right Multi Reg

Format: \texttt{SAM rA, rB, rC}

Purpose: Shift right arithmetic a 54-bit integer

Description: The 54-bit value in \texttt{rB:rB+1} is right arithmetic shifted the number of bits specified in \texttt{rC}. The result is placed in \texttt{rA:rA+1}.

Operation: \[ \texttt{rA:rA+1} \leftarrow (\text{signed})\texttt{rB:rB+1} >> \texttt{rC} \]

Flags affected: Z C O S

4.102 SB: Subtract

Format: \texttt{SB rA, rB, rC}

Purpose: Subtract two 27-bit integer registers from each other

Description: The 27-bit value in \texttt{rC} is subtracted from the 27-bit value in \texttt{rB}, the result is placed in \texttt{rA}.

Operation: \[ \texttt{rA} \leftarrow \texttt{rB} - \texttt{rC} \]

Flags affected: Z C O S
4.103 SBC: Subtract With Carry

0 6 7 11 12 16 17 21 22 25 26
0100100 rA rB rC 0000 UF

**Format:** SBC rA, rB, rC

**Purpose:** Subtract two 27-bit integer registers from each other including the carry bit

**Description:** The 27-bit value in rC is subtracted from the 27-bit value in rB, the carry bit from any previous operation is subtracted from the result. The result is placed in rA.

**Operation:**

\[
\text{rA} \leftarrow \text{rB} - \text{rC} - \text{Carry.Bit}
\]

**Flags affected:** Z C O S

4.104 SBCI: Subtract Immediate With Carry

0 6 7 11 12 16 17 23 24 25 26
0100100 rA rB Immediate 01 UF

**Format:** SBCI rA, rB, IMM

**Purpose:** Subtract a 7-bit immediate from a 27-bit integer register including the carry bit

**Description:** The 7-bit immediate value is subtracted from the 27-bit value in rB, the carry bit from any previous operation is subtracted from the result. The result is placed in rA.

**Operation:**

\[
\text{rA} \leftarrow \text{rB} - \text{IMM} - \text{Carry.Bit}
\]

**Flags affected:** Z C O S
4.105  **SBCIM: Subtract Immediate Multi Reg With Carry**

**Format:** SBCIM rA, rB, IMM

**Purpose:** Subtract a 7-bit immediate from a 54-bit integer register including the carry bit

**Description:** The 7-bit immediate value is subtracted from the 54-bit value in rB:rB+1, the carry bit from any previous operation is subtracted from the result. The result is placed in rA:rA+1.

**Operation:** rA:rA+1 ← rB:rB+1 - IMM - Carry_Bit

**Flags affected:** Z C O S

4.106  **SBCM: Subtract Multi Reg With Carry**

**Format:** SBCM rA, rB, rC

**Purpose:** Subtract two 54-bit integer registers from each other including the carry bit

**Description:** The 54-bit value in rC:rC+1 is subtracted from the 54-bit value in rB:rB+1, the carry bit from any previous operation is subtracted from the result. The result is placed in rA:rA+1.

**Operation:** rA:rA+1 ← rB:rB+1 - rC:rC+1 - Carry_Bit

**Flags affected:** Z C O S
4.107 SBF: Subtract Floating Point

Format:  $\text{SBF } rA, rB, rC$

Purpose: Subtract two 27-bit floating point registers from each other

Description: The 27-bit floating point value in $rC$ is subtracted from the 27-bit floating point value in $rB$, the result is placed in $rA$.

Operation: $rA \leftarrow rB - rC$

Flags affected: $Z C O S$

4.108 SBFM: Subtract Floating Point Multi Reg

Format:  $\text{SBFM } rA, rB, rC$

Purpose: Subtract two 54-bit floating point registers from each other

Description: The 54-bit floating point value in $rC:rC+1$ is subtracted from the 54-bit floating point value in $rB:rB+1$, the result is placed in $rA:rA+1$.

Operation: $rA:rA+1 \leftarrow rB:rB+1 - rC:rC+1$

Flags affected: $Z C O S$

4.109 SBI: Subtract Immediate

Format:  $\text{SBI } rA, rB, \text{ IMM}$

Purpose: Subtract a 7-bit immediate value from a 27-bit integer register

Description: The 7-bit immediate value is subtracted from the 27-bit value in $rB$, the result is placed in $rA$.

Operation: $rA \leftarrow rB - \text{ IMM}$

Flags affected: $Z C O S$
4.110 SBIM: Subtract Immediate Multi Reg

\[
0 | 6 | 7 | 11 | 12 | 16 | 17 | 23 | 24 | 25 | 26 \\
0000110 | rA | rB | Immediate | 01 | UF
\]

Format: SBIM rA, rB, IMM

Purpose: Subtract a 7-bit immediate value from a 54-bit integer register

Description: The 7-bit immediate value is subtracted from the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: \( rA:rA+1 \leftarrow rB:rB+1 - \text{IMM} \)

Flags affected: Z C O S

4.111 SBM: Subtract Multi Reg

\[
0 | 6 | 7 | 11 | 12 | 16 | 17 | 21 | 22 | 25 | 26 \\
0000110 | rA | rB | rC | 0000 | UF
\]

Format: SBM rA, rB, rC

Purpose: Subtract two 54-bit integer registers from each other

Description: The 54-bit value in rC:rC+1 is subtracted from the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

Operation: \( rA:rA+1 \leftarrow rB:rB+1 - rC:rC+1 \)

Flags affected: Z C O S

4.112 SES: Sign Extend Single

\[
0 | 11 | 12 | 16 | 17 | 21 | 22 | 26 \\
10100000011 | rA | rB | 00000
\]

Format: SES rA, rB

Purpose: Sign extend from a byte

Description: Sign extend a single byte from register rB into the 27-bit rA.

Operation: \( rA \leftarrow (\text{signed}) (rB \ll 18) \gg 18 \)

Flags affected: Z C O S
4.113  SEW: Sign Extend Word

Format: SEW rA, rB

Purpose: Sign extend from two bytes

Description: Sign extend a 18-bit value from register rB into the 27-bit rA.

Operation: \( rA \leftarrow (\text{signed}) (rB \ll 9) \gg 9 \)

Flags affected: Z C O S

4.114  SF: Set Flags

Format: SF rA

Purpose: Set flag register to rA

Description: Set the flag register to the value in rA.

Operation: \( \text{FLAGS} = rA \& 0x7FFFFFF \)

Flags affected: Z C O S

4.115  SL: Shift Left

Format: SL rA, rB, rC

Purpose: Shift left a 27-bit integer

Description: The 27-bit value in rB is left shifted the number of bits specified in rC. The result is placed in rA.

Operation: \( rA \leftarrow rB \ll rC \)

Flags affected: Z C O S
4.116 SLI: Shift Left Immediate

Format: SLI rA, rB, IMM
Purpose: Shift left a 27-bit integer by a 7-bit immediate value
Description: The 27-bit value in rB is left shifted by the 7-bit immediate value. The result is placed in rA.
Operation: rA ← rB << IMM
Flags affected: Z C 0 S

4.117 SLIM: Shift Left Immediate Multi Reg

Format: SLIM rA, rB, IMM
Purpose: Shift left a 54-bit integer by a 7-bit immediate value
Description: The 54-bit value in rB is left shifted by the 7-bit immediate value. The result is placed in rA:rA+1.
Operation: rA:rA+1 ← rB:rB+1 << IMM
Flags affected: Z C 0 S

4.118 SLM: Shift Left Multi Reg

Format: SLM rA, rB, rC
Purpose: Shift left a 54-bit integer
Description: The 54-bit value in rB:rB+1 is left shifted the number of bits specified in rC. The result is placed in rA:rA+1.
Operation: rA:rA+1 ← rB:rB+1 << rC
Flags affected: Z C 0 S
4.119 SMP: Set Memory Protection

**Format:** SMP rA, rB, FLAGS

**Purpose:** Set the memory protections for a range of pages of memory

**Description:** Memory starting at a page boundary specified by rA has its page protection flags set to FLAGS for rB pages. A set zero flag indicates success. Upon failure rA holds the error reason.

**Errors:**

<table>
<thead>
<tr>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Memory start is not page aligned</td>
</tr>
<tr>
<td>1</td>
<td>Too many pages specified</td>
</tr>
</tbody>
</table>

**Flags:**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No access</td>
</tr>
<tr>
<td>1</td>
<td>Read</td>
</tr>
<tr>
<td>2</td>
<td>Read/Write</td>
</tr>
<tr>
<td>3</td>
<td>Read/Execute</td>
</tr>
</tbody>
</table>

**Operation:**

Flags affected: Z
4.120  **SR: Shift Right**

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>21</th>
<th>22</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>0101001</td>
<td>rA</td>
<td>rB</td>
<td>rC</td>
<td>0000</td>
<td>UF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** SR rA, rB, rC

**Purpose:** Shift right a 27-bit integer

**Description:** The 27-bit value in rB is right shifted the number of bits specified in rC. The result is placed in rA.

**Operation:** rA ← rB >> rC

**Flags affected:** Z C 0 S

---

4.121  **SRI: Shift Right Immediate**

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>0111001</td>
<td>rA</td>
<td>rB</td>
<td>Immediate</td>
<td>00</td>
<td>UF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** SRI rA, rB, IMM

**Purpose:** Shift right a 27-bit integer by a 7-bit immediate value

**Description:** The 27-bit value in rB is right shifted by the 7-bit immediate value. The result is placed in rA.

**Operation:** rA ← rB >> IMM

**Flags affected:** Z C 0 S
### 4.122 SRIM: Shift Right Immediate Multi Reg

**Format:** SRIM rA, rB, IMM

**Purpose:** Shift right a 54-bit integer by a 7-bit immediate value

**Description:** The 54-bit value in rB:rB+1 is right shifted by the 7-bit immediate value. The result is placed in rA:rA+1.

**Operation:**

\[
\begin{align*}
\text{rA} : \text{rA} + 1 & \leftarrow \text{rB} : \text{rB} + 1 \gg \text{IMM} \\
\text{Flags affected: } & \text{Z C O } S
\end{align*}
\]

### 4.123 SRM: Shift Right Multi Reg

**Format:** SRM rA, rB, rC

**Purpose:** Shift right a 54-bit integer

**Description:** The 54-bit value in rB:rB+1 is right shifted the number of bits specified in rC. The result is placed in rA:rA+1.

**Operation:**

\[
\begin{align*}
\text{rA} : \text{rA} + 1 & \leftarrow \text{rB} : \text{rB} + 1 \gg \text{rC} \\
\text{Flags affected: } & \text{Z C O } S
\end{align*}
\]
4.124 STS: Store Single

0 6 7 11 12 16 17 21 22 23 24 50 51 53
1011000 rA rB Register Count Adjust rB Memory Offset 000

Format: STSm rA, [rB + Offset, RegCount]

Purpose: Store a single byte into memory

Description: The value in rB is added to Offset and a single byte per register is stored from the low 9 bits of each register from rA to rA+RegCount. m refers to the Adjust RB value which can have the following designations to indicate the mode:

<table>
<thead>
<tr>
<th>m</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>rB is not adjusted after store</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>rB has the number of bytes written added to the start value after all data is written</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>rB has the number of bytes to write subtracted from the start value after all data is written</td>
</tr>
</tbody>
</table>

Operation:

StartReg = rA
RegCount = RegCount + 1
CurCount = RegCount
Temp = rB
if Mode is 2 then
    Temp = Temp - CurCount
MemLocation = (Temp + Offset)
While CurCount is not 0
    Memory[MemLocation] = Registers[StartReg] & 0x1ff
    MemLocation += 1
    StartReg = (StartReg + 1) % 32
    CurCount = CurCount - 1
if Mode is 1 then
    rB = rB + RegCount
if Mode is 2 then
    rB = Temp

Flags affected: None
4.125  **STT: Store Tri**

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>50</th>
<th>51</th>
<th>53</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rA</td>
<td>rB</td>
<td>Register Count</td>
<td>Adjust rB</td>
<td>Memory Offset</td>
<td>000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** \( \text{STT} m \ rA, [rB + \text{Offset}, \text{RegCount}] \)

**Purpose:** Store three bytes into memory

**Description:** The value in rB is added to Offset and a three bytes per register are stored for each register from rA to rA+RegCount. \( m \) refers to the Adjust RB value which can have the following designations to indicate the mode:

<table>
<thead>
<tr>
<th>( m )</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rB is not adjusted after store</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>rB has the number of bytes written added to the start value after all data is written</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>rB has the number of bytes to write subtracted from the start value after all data is written</td>
</tr>
</tbody>
</table>

**Operation:**

\[
\text{StartReg} = rA \\
\text{RegCount} = \text{RegCount} + 1 \\
\text{CurCount} = \text{RegCount} \\
\text{Temp} = rB \\
\text{if Mode is 2 then} \\
\quad \text{Temp} = \text{Temp} - (\text{CurCount} \times 3) \\
\quad \text{MemLocation} = (\text{Temp} + \text{Offset}) \\
\text{While CurCount is not 0} \\
\quad \text{Memory[MemLocation]} = (\text{Registers[StartReg]} \gg 9) \& \text{0x1ff} \\
\quad \text{MemLocation} += 1 \\
\quad \text{Memory[MemLocation]} = (\text{Registers[StartReg]} \gg 18) \& \text{0x1ff} \\
\quad \text{MemLocation} += 1 \\
\quad \text{Memory[MemLocation]} = \text{Registers[StartReg]} \& \text{0x1ff} \\
\quad \text{MemLocation} += 1 \\
\quad \text{StartReg} = (\text{StartReg} + 1) \% \text{32} \\
\quad \text{CurCount} = \text{CurCount} - 1 \\
\text{if Mode is 1 then} \\
\quad rB = rB + (\text{RegCount} \times 3) \\
\text{if Mode is 2 then} \\
\quad rB = \text{Temp} \\
\]

**Flags affected:** None
4.126 STW: Store Word

Format: STWm rA, [rB + Offset, RegCount]

Purpose: Store two bytes into memory

Description: The value in rB is added to Offset and a two bytes per register are stored from the low 18 bits of each register from rA to rA+RegCount. m refers to the Adjust RB value which can have the following designations to indicate the mode:

<table>
<thead>
<tr>
<th>m</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>rB is not adjusted after store</td>
</tr>
<tr>
<td>1</td>
<td>I</td>
<td>rB has the number of bytes written added to the start value after all data is written</td>
</tr>
<tr>
<td>2</td>
<td>D</td>
<td>rB has the number of bytes to write subtracted from the start value after all data is written</td>
</tr>
</tbody>
</table>

Operation:

StartReg = rA
RegCount = RegCount + 1
CurCount = RegCount
Temp = rB
if Mode is 2 then
   Temp = Temp - (CurCount * 2)
MemLocation = (Temp + Offset)
While CurCount is not 0
   Memory[MemLocation] = (Registers[StartReg] >> 9) & 0x1ff
   MemLocation = (MemLocation + 1)
   Memory[MemLocation] = Registers[StartReg] & 0x1ff
   MemLocation = (MemLocation + 1)
   StartReg = (StartReg + 1) % 32
   CurCount = CurCount - 1
if Mode is 1 then
   rB = rB + (RegCount * 2)
if Mode is 2 then
   rB = Temp

Flags affected: None
4.127 WT: Wait

0 17
10100000001000000

Format: \texttt{WT}

Purpose: Wait for interrupt

Description: Pauses the processor until an interrupt fires.

Operation:

Flags affected: \textit{None}

4.128 XR: Xor

0 6 7 11 12 16 17 21 22 25 26
0011100 rA rB rC 0000 UF

Format: \texttt{XR rA, rB, rC}

Purpose: Bit-wise eXclusive OR two 27-bit integer registers together

Description: The 27-bit value in rC is bit-wise eXclusive OR to the 27-bit value in rB, the result is placed in rA.

Operation: \( rA \leftarrow rB \oplus rC \)

Flags affected: \( Z\ C\ O\ S \)

4.129 XRI: Xor Immediate

0 6 7 11 12 16 17 23 24 25 26
0011100 rA rB Immediate 01 UF

Format: \texttt{XRI rA, rB, IMM}

Purpose: Bit-wise Exclusive OR a 27-bit integer register and 7-bit immediate together

Description: The 7-bit immediate value is bit-wise Exclusive OR to the 27-bit value in rB, the result is placed in rA.

Operation: \( rA \leftarrow rB \oplus IMM \)

Flags affected: \( Z\ C\ O\ S \)
### 4.130 XRM: Xor Multi Reg

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>7</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>21</th>
<th>22</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011110</td>
<td>rA</td>
<td>rB</td>
<td>rC</td>
<td>0000</td>
<td>UF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** XRM rA, rB, rC

**Purpose:** Bit-wise eXclusive OR two 54-bit integer registers together

**Description:** The 54-bit value in rC:rC+1 is bit-wise eXclusive OR to the 54-bit value in rB:rB+1, the result is placed in rA:rA+1.

**Operation:** rA:rA+1 ← rB:rB+1 ^ rC:rC+1

**Flags affected:** Z C O S

### 4.131 ZES: Zero Extend Single

<table>
<thead>
<tr>
<th>0</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>21</th>
<th>22</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>101000001001</td>
<td>rA</td>
<td>rB</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** ZES rA, rB

**Purpose:** Zero extend from a byte

**Description:** Zero extend a single byte from register rB into the 27-bit rA.

**Operation:** rA ← rB & 0x00001FF

**Flags affected:** Z C O S

### 4.132 ZEW: Zero Extend Word

<table>
<thead>
<tr>
<th>0</th>
<th>11</th>
<th>12</th>
<th>16</th>
<th>17</th>
<th>21</th>
<th>22</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>101000001010</td>
<td>rA</td>
<td>rB</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** ZEW rA, rB

**Purpose:** Zero extend from two bytes

**Description:** Zero extend a 18-bit value from register rB into the 27-bit rA.

**Operation:** rA ← rB & 0x003FFFF

**Flags affected:** Z C O S