

# GameCube DSP User's Manual

Reverse-engineered and documented by Duddie  
[duddie@walla.com](mailto:duddie@walla.com)

May 13, 2025  
v0.1.7



# Contents

|  |           |
|--|-----------|
| <b>Contents</b>                                      | <b>3</b>  |
| <b>Disclaimer</b>                                    | <b>7</b>  |
| <b>Version History</b>                               | <b>8</b>  |
| <b>GNU Documentation License 1.2</b>                 | <b>9</b>  |
| <b>1 Overview</b>                                    | <b>13</b> |
| 1.1 DSP Memory Map . . . . .                         | 13        |
| 1.1.1 Instruction Memory . . . . .                   | 13        |
| 1.1.2 Data Memory . . . . .                          | 14        |
| 1.2 Initialization . . . . .                         | 15        |
| <b>2 Registers</b>                                   | <b>16</b> |
| 2.1 Register names . . . . .                         | 16        |
| 2.2 Accumulators . . . . .                           | 17        |
| 2.2.1 Accumulators <b>\$acX</b> . . . . .            | 17        |
| 2.2.2 Short accumulators <b>\$acsX</b> . . . . .     | 17        |
| 2.2.3 Additional accumulators <b>\$axX</b> . . . . . | 17        |
| 2.2.4 16-bit and 40-bit modes . . . . .              | 17        |
| 2.3 Stacks . . . . .                                 | 18        |
| 2.4 Config register . . . . .                        | 19        |
| 2.5 Status register . . . . .                        | 20        |
| 2.6 Product register . . . . .                       | 21        |
| <b>3 Exceptions</b>                                  | <b>22</b> |
| 3.1 Exception processing . . . . .                   | 22        |
| 3.2 Exception vectors . . . . .                      | 23        |
| <b>4 Hardware interface</b>                          | <b>24</b> |
| 4.1 Hardware registers . . . . .                     | 24        |
| 4.2 DMA . . . . .                                    | 25        |
| 4.3 Accelerator . . . . .                            | 26        |
| 4.4 Interrupts . . . . .                             | 30        |
| 4.5 Mailboxes . . . . .                              | 31        |
| 4.5.1 DSP Mailbox . . . . .                          | 31        |
| 4.5.2 CPU Mailbox . . . . .                          | 32        |
| <b>5 Opcodes</b>                                     | <b>33</b> |
| 5.1 Opcode syntax . . . . .                          | 33        |
| 5.2 Operation — Used Functions . . . . .             | 34        |
| 5.3 Bit meanings . . . . .                           | 35        |
| 5.4 Conditional opcodes . . . . .                    | 36        |
| 5.5 Flags . . . . .                                  | 37        |
| 5.6 Alphabetical list of opcodes . . . . .           | 38        |
| 5.6.1 ABS . . . . .                                  | 39        |
| 5.6.2 ADD . . . . .                                  | 40        |

|        |         |    |
|--------|---------|----|
| 5.6.3  | ADDARN  | 41 |
| 5.6.4  | ADDAX   | 42 |
| 5.6.5  | ADDAXL  | 43 |
| 5.6.6  | ADDI    | 44 |
| 5.6.7  | ADDIS   | 45 |
| 5.6.8  | ADDP    | 46 |
| 5.6.9  | ADDPAXZ | 47 |
| 5.6.10 | ADDR    | 48 |
| 5.6.11 | ANDC    | 49 |
| 5.6.12 | ANDCF   | 50 |
| 5.6.13 | ANDF    | 51 |
| 5.6.14 | ANDI    | 52 |
| 5.6.15 | ANDR    | 53 |
| 5.6.16 | ASL     | 54 |
| 5.6.17 | ASR     | 55 |
| 5.6.18 | ASRN    | 56 |
| 5.6.19 | ASRNR   | 57 |
| 5.6.20 | ASRNRX  | 58 |
| 5.6.21 | ASR16   | 59 |
| 5.6.22 | BLOOP   | 60 |
| 5.6.23 | BLOOPI  | 61 |
| 5.6.24 | CALL    | 62 |
| 5.6.25 | CALLcc  | 63 |
| 5.6.26 | CALLR   | 64 |
| 5.6.27 | CALLRcc | 65 |
| 5.6.28 | CLR15   | 66 |
| 5.6.29 | CLR     | 67 |
| 5.6.30 | CLRL    | 68 |
| 5.6.31 | CLRP    | 69 |
| 5.6.32 | CMP     | 70 |
| 5.6.33 | CMPAXH  | 71 |
| 5.6.34 | CMPI    | 72 |
| 5.6.35 | CMPIS   | 73 |
| 5.6.36 | DAR     | 74 |
| 5.6.37 | DEC     | 75 |
| 5.6.38 | DECM    | 76 |
| 5.6.39 | HALT    | 77 |
| 5.6.40 | IAR     | 78 |
| 5.6.41 | IFcc    | 79 |
| 5.6.42 | ILRR    | 80 |
| 5.6.43 | ILRRD   | 81 |
| 5.6.44 | ILRRI   | 82 |
| 5.6.45 | ILRRN   | 83 |
| 5.6.46 | INC     | 84 |
| 5.6.47 | INCM    | 85 |
| 5.6.48 | JMP     | 86 |
| 5.6.49 | Jcc     | 87 |
| 5.6.50 | JMPR    | 88 |
| 5.6.51 | JRcc    | 89 |
| 5.6.52 | LOOP    | 90 |
| 5.6.53 | LOOPI   | 91 |
| 5.6.54 | LR      | 92 |
| 5.6.55 | LRI     | 93 |
| 5.6.56 | LRIS    | 94 |
| 5.6.57 | LRR     | 95 |
| 5.6.58 | LRRD    | 96 |
| 5.6.59 | LRRI    | 97 |
| 5.6.60 | LRRN    | 98 |

|         |         |     |
|---------|---------|-----|
| 5.6.61  | LRS     | 99  |
| 5.6.62  | LSL     | 100 |
| 5.6.63  | LSL16   | 101 |
| 5.6.64  | LSR     | 102 |
| 5.6.65  | LSRN    | 103 |
| 5.6.66  | LSRNR   | 104 |
| 5.6.67  | LSRNRX  | 105 |
| 5.6.68  | LSR16   | 106 |
| 5.6.69  | M0      | 107 |
| 5.6.70  | M2      | 108 |
| 5.6.71  | MADD    | 109 |
| 5.6.72  | MADDC   | 110 |
| 5.6.73  | MADDX   | 111 |
| 5.6.74  | MOV     | 112 |
| 5.6.75  | MOVAX   | 113 |
| 5.6.76  | MOVNP   | 114 |
| 5.6.77  | MOVP    | 115 |
| 5.6.78  | MOVPZ   | 116 |
| 5.6.79  | MOVR    | 117 |
| 5.6.80  | MRR     | 118 |
| 5.6.81  | MSUB    | 119 |
| 5.6.82  | MSUBC   | 120 |
| 5.6.83  | MSUBX   | 121 |
| 5.6.84  | MUL     | 122 |
| 5.6.85  | MULAC   | 123 |
| 5.6.86  | MULAXH  | 124 |
| 5.6.87  | MULC    | 125 |
| 5.6.88  | MULCAC  | 126 |
| 5.6.89  | MULCMV  | 127 |
| 5.6.90  | MULCMVZ | 128 |
| 5.6.91  | MULMV   | 129 |
| 5.6.92  | MULMVZ  | 130 |
| 5.6.93  | MULX    | 131 |
| 5.6.94  | MULXAC  | 132 |
| 5.6.95  | MULXMV  | 133 |
| 5.6.96  | MULXMVZ | 134 |
| 5.6.97  | NEG     | 135 |
| 5.6.98  | NOT     | 136 |
| 5.6.99  | NOP     | 137 |
| 5.6.100 | NX      | 138 |
| 5.6.101 | ORC     | 139 |
| 5.6.102 | ORI     | 140 |
| 5.6.103 | ORR     | 141 |
| 5.6.104 | RET     | 142 |
| 5.6.105 | RETcc   | 143 |
| 5.6.106 | RTI     | 144 |
| 5.6.107 | RTIcc   | 145 |
| 5.6.108 | SBCLR   | 146 |
| 5.6.109 | SBSET   | 147 |
| 5.6.110 | SET15   | 148 |
| 5.6.111 | SET16   | 149 |
| 5.6.112 | SET40   | 150 |
| 5.6.113 | SI      | 151 |
| 5.6.114 | SR      | 152 |
| 5.6.115 | SRR     | 153 |
| 5.6.116 | SRRD    | 154 |
| 5.6.117 | SRRI    | 155 |
| 5.6.118 | SRRN    | 156 |

|         |                                       |     |
|---------|---------------------------------------|-----|
| 5.6.119 | SRS                                   | 157 |
| 5.6.120 | SRSH                                  | 158 |
| 5.6.121 | SUB                                   | 159 |
| 5.6.122 | SUBARN                                | 160 |
| 5.6.123 | SUBAX                                 | 161 |
| 5.6.124 | SUBP                                  | 162 |
| 5.6.125 | SUBR                                  | 163 |
| 5.6.126 | TST                                   | 164 |
| 5.6.127 | TSTAXH                                | 165 |
| 5.6.128 | TSTPROD                               | 166 |
| 5.6.129 | XORC                                  | 167 |
| 5.6.130 | XORI                                  | 168 |
| 5.6.131 | XORR                                  | 169 |
| 5.7     | Extended opcodes                      | 170 |
| 5.8     | Alphabetical list of extended opcodes | 171 |
| 5.8.1   | 'DR                                   | 172 |
| 5.8.2   | 'IR                                   | 173 |
| 5.8.3   | 'L                                    | 174 |
| 5.8.4   | 'LN                                   | 175 |
| 5.8.5   | 'LD                                   | 176 |
| 5.8.6   | 'LDM                                  | 177 |
| 5.8.7   | 'LDNM                                 | 178 |
| 5.8.8   | 'LDN                                  | 179 |
| 5.8.9   | 'LDAX                                 | 180 |
| 5.8.10  | 'LDAXM                                | 181 |
| 5.8.11  | 'LDAXNM                               | 182 |
| 5.8.12  | 'LDAXN                                | 183 |
| 5.8.13  | 'LS                                   | 184 |
| 5.8.14  | 'LSM                                  | 185 |
| 5.8.15  | 'LSNM                                 | 186 |
| 5.8.16  | 'LSN                                  | 187 |
| 5.8.17  | 'MV                                   | 188 |
| 5.8.18  | 'NOP                                  | 189 |
| 5.8.19  | 'NR                                   | 190 |
| 5.8.20  | 'S                                    | 191 |
| 5.8.21  | 'SL                                   | 192 |
| 5.8.22  | 'SLM                                  | 193 |
| 5.8.23  | 'SLNM                                 | 194 |
| 5.8.24  | 'SLN                                  | 195 |
| 5.8.25  | 'SN                                   | 196 |
| 5.9     | Instructions sorted by opcode         | 197 |

## Disclaimer

This documentation is no way endorsed by or affiliated with Nintendo, Nintendo of America or its licenses. GameCube is a trademark of Nintendo of America. Other trademarked names used in this documentation are trademarks of their respective owners.

This documentation is provided “AS IS” and can be wrong, incomplete or in any other way useless.

This documentation cannot be used for any commercial purposes without prior agreement received from authors.

The purpose of this documentation is purely academic and it aims at understanding described hardware. It is based on academic reverse engineering of hardware.

## Version History

| Version | Date       | Author     | Change  |
|---------|------------|------------|---|
| 0.0.1   | 2005.05.08 | Duddie     | Initial release   |
| 0.0.2   | 2005.05.09 | Duddie     | Added <code>\$prod</code> and <code>\$config</code> registers, table of opcodes, disclaimer.  |
| 0.0.3   | 2005.05.09 | Duddie     | Fixed BLOOP and BLOOPI and added description of the loop stack.   |
| 0.0.4   | 2005.05.12 | Duddie     | Added preliminary DSP memory map and opcode syntax.   |
| 0.0.5   | 2018.04.09 | Lioncache  | Converted document over to LaTeX.   |
| 0.0.6   | 2018.04.13 | BhaaL      | Updated register tables, fixed opcode operations  |
| 0.0.7   | Mid 2020   | Tilka      | Fixed typos and register names, and improved readability.   |
| 0.1.0   | 2021.08.21 | Pokechu22  | Added missing instructions, improved documentation of hardware registers, documented additional behaviors, and improved formatting. |
| 0.1.1   | 2022.05.14 | xperia64   | Added tested DSP bootloading transfer size  |
| 0.1.2   | 2022.05.21 | Pokechu22  | Fixed “ILLR” typo in Instruction Memory section   |
| 0.1.3   | 2022.05.27 | Pokechu22  | Renamed <code>CMPAR</code> instruction to <code>CMPAXH</code>   |
| 0.1.4   | 2022.06.02 | Pokechu22  | Fixed typos; added sections on 16-bit and 40-bit modes and on main and extended opcode writing to the same register.                |
| 0.1.5   | 2022.09.29 | vpelletier | Fixed BLOOP and BLOOPI suboperation order   |
| 0.1.6   | 2022.06.20 | xperia64   | Accelerator documentation updates, fix register typo in ANDC and ORC descriptions   |
| 0.1.7   | 2025.04.21 | Tilka      | Fixed typos and complained about GFDL   |



## GNU Free Documentation License

Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc. 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image

formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

## 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

## 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the

Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

## 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

## 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <https://www.gnu.org/licenses/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

# Chapter 1

## Overview

### 1.1 DSP Memory Map

The DSP has two address spaces, one for data and one for instructions. The DSP accesses memory in words, so all addresses refer to words. A DSP word is 16 bits in size.

#### 1.1.1 Instruction Memory

Instruction Memory (IMEM) is divided into instruction RAM (IRAM) and instruction ROM (IROM).

Exception vectors are located at the top of the RAM and occupy the first 16 words, with 2 words available for each exception (enough for a [JMP](#) instruction for each exception).

There are no DSP instructions that write to IMEM; however, the [ILRR](#) family of instructions can read from it. This is sometimes used for jump tables or indexing into a list of pointers (which may point into either IMEM or DMEM).

|        |      |
|--------|------|
| 0x0000 | IRAM |
| 0x0FFF |      |
| 0x8000 | IROM |
| 0x8FFF |      |

### 1.1.2 Data Memory

Data Memory (DMEM) is divided into data RAM (DRAM) and resampling coefficient data (COEF). Hardware registers (IFX) are also mapped into this space.

It is possible to both read and write to DMEM, but coefficient data cannot be written to.

|        |      |
|--------|------|
| 0x0000 | DRAM |
| 0x0FFF |      |
| 0x1000 | COEF |
| 0x17FF |      |
| 0xFF00 | IFX  |
| 0xFFFF |      |

## 1.2 Initialization

The DSP is initialized before it is used. This is done by copying a small program to physical address `0x01000000` (virtual `0x81000000`) in GameCube/Wii main memory, and then writing to `DSP_CONTROL_STATUS` at `0xCC00500A` with the 11th and 0th bits set (SDK titles write `0x08ad`). The 11th bit being set appears to cause data from `0x01000000` to be DMA'd to the start of IMEM; a basic hardware test (sending increasingly larger payloads until they fail) indicates 1024 bytes of data (512 DSP words) are transferred. (None of this has been extensively hardware tested, and is instead based on libogc's `__dsp_bootstrap`.)

The program that SDK titles send does the following:

1. Reads all `0x1000` words of IROM from `0x8000` through `0x8FFF` (using [ILRRI](#))
2. Writes zero to all `0x1000` words of DRAM from `0x0000` through `0xFFFF` (using [SRRI](#))
3. Reads all `0x0800` words of COEF data from `0x1000` through `0x17FF` (using [LRRl](#))
4. Waits for the top bit of DMBH to be clear (indicating the CPU is ready to receive mail)
5. Writes `0x0054` to DMBH and `0x4348` to DMBL, sending `0x00543448` (“TCH”?) to the CPU. The CPU does not check for this value, but it does wait for mail to be sent.

It is not clear why this is done, as the values read from IROM and COEF are not used; perhaps it works around a hardware bug where incorrect values are read from ROM initially.

## Chapter 2

# Registers

### 2.1 Register names

The DSP has 32 16-bit registers, although their individual purpose and their function differ from register to register.

|      |       |           |                             |
|------|-------|-----------|-----------------------------|
|      |       |           |                             |
| \$0  | \$r00 | \$ar0     | Addressing register 0       |
| \$1  | \$r01 | \$ar1     | Addressing register 1       |
| \$2  | \$r02 | \$ar2     | Addressing register 2       |
| \$3  | \$r03 | \$ar3     | Addressing register 3       |
| \$4  | \$r04 | \$ix0     | Indexing register 0         |
| \$5  | \$r05 | \$ix1     | Indexing register 1         |
| \$6  | \$r06 | \$ix2     | Indexing register 2         |
| \$7  | \$r07 | \$ix3     | Indexing register 3         |
| \$8  | \$r08 | \$wr0     | Wrapping register 0         |
| \$9  | \$r09 | \$wr1     | Wrapping register 1         |
| \$10 | \$r0A | \$wr2     | Wrapping register 2         |
| \$11 | \$r0B | \$wr3     | Wrapping register 3         |
| \$12 | \$r0C | \$st0     | Call stack register         |
| \$13 | \$r0D | \$st1     | Data stack register         |
| \$14 | \$r0E | \$st2     | Loop address stack register |
| \$15 | \$r0F | \$st3     | Loop counter register       |
| \$16 | \$r10 | \$ac0.h   | 40-bit Accumulator 0 (high) |
| \$17 | \$r11 | \$ac1.h   | 40-bit Accumulator 1 (high) |
| \$18 | \$r12 | \$config  | Config register             |
| \$19 | \$r13 | \$sr      | Status register             |
| \$20 | \$r14 | \$prod.l  | Product register (low)      |
| \$21 | \$r15 | \$prod.m1 | Product register (mid 1)    |
| \$22 | \$r16 | \$prod.h  | Product register (high)     |
| \$23 | \$r17 | \$prod.m2 | Product register (mid 2)    |
| \$24 | \$r18 | \$ax0.l   | 32-bit Accumulator 0 (low)  |
| \$25 | \$r19 | \$ax1.l   | 32-bit Accumulator 1 (low)  |
| \$26 | \$r1A | \$ax0.h   | 32-bit Accumulator 0 (high) |
| \$27 | \$r1B | \$ax1.h   | 32-bit Accumulator 1 (high) |
| \$28 | \$r1C | \$ac0.l   | 40-bit Accumulator 0 (low)  |
| \$29 | \$r1D | \$ac1.l   | 40-bit Accumulator 1 (low)  |
| \$30 | \$r1E | \$ac0.m   | 40-bit Accumulator 0 (mid)  |
| \$31 | \$r1F | \$ac1.m   | 40-bit Accumulator 1 (mid)  |



## 2.2 Accumulators

The DSP has two long 40-bit accumulators (**\$acX**) and their short 24-bit forms (**\$acsX**) that reflect the upper part of 40-bit accumulator. There are additional two 32-bit accumulators (**\$axX**).

The high parts of the 40-bit accumulators (**acX.h**) are sign-extended 8-bit registers. Writes to the upper 8 bits are ignored, and the upper 8 bits read the same as the 7th bit. For instance, 0x007F reads back as 0x007F, but 0x0080 reads back as 0xFF80.

### 2.2.1 Accumulators \$acX

40-bit accumulator **\$acX** (**\$acX.hml**) consists of registers:

$$\text{\$acX} = \text{\$acX.h} \ll 32 \mid \text{\$acX.m} \ll 16 \mid \text{\$acX.l}$$

### 2.2.2 Short accumulators \$acsX

24-bit accumulator **\$acsX** (**\$acX.hm**) consists of the upper 24 bits of accumulator **\$acX**.

$$\text{\$acsX} = \text{\$acX.h} \ll 16 \mid \text{\$acX.m}$$

### 2.2.3 Additional accumulators \$axX

32-bit accumulators **\$axX** (**\$axX.hl**) consist of registers:

$$\text{\$axX} = \text{\$axX.h} \ll 16 \mid \text{\$axX.l}$$

### 2.2.4 16-bit and 40-bit modes

Depending on the value of **\$sr.SXM** (bit 14), loading to **\$acX.m** may also update **\$acX.h** and **\$acX.l**, and stores from **\$acX.m** may experience saturation based on **\$acX.h**. Regardless of the value of **\$sr.SXM**, arithmetic operations such as **ADDI**, **INCM**, **MOVR**, and **LSRN** will still affect the entire accumulator.

If **\$sr.SXM** is set to 0, then 16-bit mode (**SET16**) is in use. Loads to **\$acX.m** will only change **\$acX.m**, and storing **\$acX.m** will use the value directly contained in **\$acX.m**; the same applies to loads to and stores from **\$acX.h** or **\$acX.l** or any other register.

If **\$sr.SXM** is set to 1, then 40-bit mode (**SET40**) is in use. Loads to **\$acX.m** will set **\$acX.l** to 0 and will sign-extend into **\$acX.h** (setting it to 0xFF if the sign bit is set (**\$acX.m & 0x8000 != 0**), and to 0 otherwise). This means that in 40-bit mode, loads to **\$acX.m** are effectively loads to the whole accumulator **\$acX**. Loads to **\$acX.h** and **\$acX.l** do not have this special behavior; they only modify the specified register (as in 16-bit mode).

Additionally, if **\$sr.SXM** is set to 1, then moving or storing from **\$acX.m** may instead result in 0x7fff or 0x8000 being used. This happens if **\$acX.hml** is not the same as sign-extending **\$acX.ml**; 0x7fff is used if **\$acX** is positive and 0x8000 is used if **\$acX** is negative.

The conditions for this saturation are the same as the conditions for **\$sr.AS** (bit 4, above s32) to be set when flags are updated. (This does not mean that the saturation happens if and only if **\$sr.AS** is set, as the flags might have been set after an operation on a different register.)

The following instructions perform sign-extension when writing to **\$acX.m**: **ILRR**, **ILRRD**, **ILRRI**, and **ILRRN**; **LR**; **LRI**; **LRIS**; **LRR**, **LRRD**, **LRRI**, and **LRRN**; **LRS**; **MRR**; and **'L** and **'LN**.

The following instructions experience saturation when reading from **\$acX.m**: **BLOOP**; **LOOP**; **MRR**; **SR**; **SRR**, **SRRD**, **SRRI**, and **SRRN**; **SRS**; **'LS**, **'LSM**, **'LSM**, and **'LSNM**; **'MV**; **'SL**, **'SLM**, **'SLN**, and **'SLNM**; and **'S** and **'SN**.

## 2.3 Stacks

The GameCube DSP contains four stack registers:

- **\$st0** – Call stack register
- **\$st1** – Data stack register
- **\$st2** – Loop address stack register
- **\$st3** – Loop counter register

Stacks are implemented in hardware and have limited depth. The data stack is limited to four values and the call stack is limited to eight values. The loop stack is limited to four values. Upon underflow or overflow of any of the stack registers exception **STOVF** is raised.

The loop stack is used to control execution of repeated blocks of instructions. Whenever there is a value in **\$st2** and the current PC is equal to the value in **\$st2**, then the value in **\$st3** is decremented. If the value is not zero, then the PC is modified by the value from call stack **\$st0**. Otherwise values from the call stack **\$st0** and both loop stacks, **\$st2** and **\$st3**, are popped and execution continues at the next opcode.

## 2.4 Config register

Serves as a base offset for [SRS](#), [SRSH](#), and [LRS](#). Zelda uCode writes it with 0x0004, but otherwise it is usually 0x00FF.

This is an 8-bit register. Writes to the upper 8 bits are ignored and those bits always read back as 0.

## 2.5 Status register

Status register `$sr` reflects flags computed on accumulators after logical or arithmetic operations. Furthermore, it also contains control bits to configure the flow of certain operations.

| Bit | Name | Comment  |
|-----|------|--|
| 15  | SU   | Multiplication operands are signed (1 = unsigned)                              |
| 14  | SXM  | Sign extension mode (1 = 40-bit, see <a href="#">16-bit and 40-bit modes</a> ) |
| 13  | AM   | Product multiply result by 2 (when AM = 0)                                     |
| 12  |      |  |
| 11  | EIE  | External interrupt enable  |
| 10  |      |  |
| 9   | IE   | Interrupt enable   |
| 8   |      | Unknown, always reads back as 0  |
| 7   | OS   | Overflow (sticky)  |
| 6   | LZ   | Logic zero (used by <a href="#">ANDCF</a> and <a href="#">ANDF</a> )           |
| 5   | TB   | Top two bits are equal   |
| 4   | AS   | Above s32  |
| 3   | S    | Sign   |
| 2   | Z    | Arithmetic zero  |
| 1   | O    | Overflow   |
| 0   | C    | Carry  |

## 2.6 Product register

The product register is a register containing the intermediate product of a multiply or multiply and accumulation operation. Its result should never be used for calculation although the register can be read or written. It reflects the state of the internal multiply unit. The product is 40 bits with 1 bit of overflow.

$$\text{\$prod} = (\text{\$prod.h} \ll 32) + ((\text{\$prod.m1} + \text{\$prod.m2}) \ll 16) + \text{\$prod.l}$$

It needs to be noted that  $\text{\$prod.m1} + \text{\$prod.m2}$  overflow bit (bit 16) will be added to  $\text{\$prod.h}$ .

Bit  $\text{\$sr.AM}$  affects the result of the multiply unit. If  $\text{\$sr.AM}$  is equal 0 then the result of every multiply operation will be multiplied by two.

$\text{prod.h}$  is 8 bits. The upper 8 bits always read back as 0.

## Chapter 3

# Exceptions

### 3.1 Exception processing

Exception processing happens by setting the program counter to different exception vectors. At exception time, the exception program counter is stored at call stack `$st0` and status register `$sr` is stored at data stack `$st1`.

**Operation:**

```
PUSH_STACK($st0);  
$st0 = $pc;  
PUSH_STACK($st1);  
$st1 = $sr;  
$pc = exception_nr * 2;
```

## 3.2 Exception vectors

Exception vectors are located at address 0x0000 in Instruction RAM.

| Level | Address | Name   | Description                              |
|-------|---------|--------|--|
| 0     | 0x0000  | RESET  |  |
| 1     | 0x0002  | STOVF  | Stack under/overflow                     |
| 2     | 0x0004  |        |  |
| 3     | 0x0006  | ACRR0V | Accelerator raw read address overflow    |
| 4     | 0x0008  | ACRW0V | Accelerator raw write address overflow   |
| 5     | 0x000A  | ACSOV  | Accelerator sample read address overflow |
| 6     | 0x000C  |        |  |
| 7     | 0x000E  | INT    | External interrupt (from CPU)            |

## Chapter 4

# Hardware interface

### 4.1 Hardware registers

Hardware registers (IFX) occupy the address space at 0xFFxx in the Data Memory space. Each register is 16 bits in width.

| Address                   | Name       | Description                   |
|---------------------------|------------|-------------------------------|
| <i>ADPCM Coefficients</i> |            |                               |
| 0xFFA0                    | COEF_A1_0  | A1 Coefficient # 0            |
| 0xFFA1                    | COEF_A2_0  | A2 Coefficient # 0            |
|                           |            | ⋮                             |
| 0xFFAE                    | COEF_A1_7  | A1 Coefficient # 7            |
| 0xFFAF                    | COEF_A2_7  | A2 Coefficient # 7            |
| <i>DMA Interface</i>      |            |                               |
| 0xFFC9                    | DSCR       | DMA control                   |
| 0xFFCB                    | DSBL       | Block length                  |
| 0xFFCD                    | DSPA       | DSP memory address            |
| 0xFFCE                    | DSMAH      | Memory address H              |
| 0xFFCF                    | DSMAL      | Memory address L              |
| <i>Accelerator</i>        |            |                               |
| 0xFFD1                    | FORMAT     | Accelerator sample format     |
| 0xFFD2                    | ACUNK1     | Unknown, usually 3            |
| 0xFFD3                    | ACDRAW     | Accelerator raw data          |
| 0xFFD4                    | ACSAH      | Accelerator start address H   |
| 0xFFD5                    | ACSAL      | Accelerator start address L   |
| 0xFFD6                    | ACEAH      | Accelerator end address H     |
| 0xFFD7                    | ACEAL      | Accelerator end address L     |
| 0xFFD8                    | ACCAH      | Accelerator current address H |
| 0xFFD9                    | ACCAL      | Accelerator current address L |
| 0xFFDA                    | PRED_SCALE | ADPCM predictor and scale     |
| 0xFFDB                    | YN1        | ADPCM output history Y[N - 1] |
| 0xFFDC                    | YN2        | ADPCM output history Y[N - 2] |
| 0xFFDD                    | ACDSAMP    | Accelerator processed sample  |
| 0xFFDE                    | GAIN       | Gain                          |
| 0xFFDF                    | ACIN       | Accelerator input             |
| 0xFFED                    | AMDM       | ARAM DMA Request Mask         |
| <i>Interrupts</i>         |            |                               |
| 0xFFFB                    | DIRQ       | IRQ request                   |
| <i>Mailboxes</i>          |            |                               |
| 0xFFFC                    | DMBH       | DSP Mailbox H                 |
| 0xFFFD                    | DMBL       | DSP Mailbox L                 |
| 0xFFFE                    | CMBH       | CPU Mailbox H                 |
| 0xFFFF                    | CMBL       | CPU Mailbox L                 |



## 4.2 DMA

The GameCube DSP is connected to the memory bus through a DMA channel. DMA can be used to transfer data between DSP memory (both instruction and data) and main memory.

|        |      |             |
|--------|------|-------------|
| 0xFFC9 | DSCR | DMA Control |
| ----   | ---- | ---- -tid   |

| Bit | Name | R/W | Action   |
|-----|------|-----|--|
| 2   | t    | R   | Transfer currently in progress if set              |
| 1   | i    | R/W | 1 - DMA to/from IMEM<br>0 - DMA to/from DMEM       |
| 0   | d    | R/W | 1 - DMA to CPU from DSP<br>0 - DMA from CPU to DSP |

|        |      |              |
|--------|------|--------------|
| 0xFFCB | DSBL | Block length |
| dddd   | dddd | dddd dddd    |

| Bit  | Name | R/W | Action   |
|------|------|-----|--|
| 15-0 | d    | W   | Length in bytes to transfer. Writing to this register starts a DMA transfer. |

|        |      |             |
|--------|------|-------------|
| 0xFFCD | DSPA | DSP Address |
| dddd   | dddd | dddd dddd   |

| Bit  | Name | R/W | Action                              |
|------|------|-----|-------------------------------------|
| 15-0 | d    | R/W | Bits 15-0 of the DSP memory address |

|        |       |                  |
|--------|-------|------------------|
| 0xFFCE | DSMAH | Memory Address H |
| dddd   | dddd  | dddd dddd        |

| Bit  | Name | R/W | Action                                |
|------|------|-----|---------------------------------------|
| 15-0 | d    | R/W | Bits 31-16 of the main memory address |

|        |       |                  |
|--------|-------|------------------|
| 0xFFCF | DSMAL | Memory Address L |
| dddd   | dddd  | dddd dddd        |

| Bit  | Name | R/W | Action                               |
|------|------|-----|--------------------------------------|
| 15-0 | d    | R/W | Bits 15-0 of the main memory address |

### 4.3 Accelerator

The accelerator is used to transfer data from accelerator memory (ARAM) to DSP memory. The accelerator area can be marked with **ACSA** (start) and **ACEA** (end) addresses. Current address for the accelerator can be set or read from the **ACCA** register. Accessing accelerator memory is done by reading or writing the **ACDRAW** register for raw data, or reading the **ACDSAMP** register for processed sample data. These registers contain raw or processed sample data from ARAM pointed to by the **ACCA** register. After reading the data, **ACCA** is incremented by one. After **ACCA** grows bigger than the area pointed to by **ACEA**, it gets reset to a value from **ACSA** and an exception is generated. Raw reads generate exception **ACRROV**, raw writes generate exception **ACRWOV**, and sample reads generate exception **ACSOV**.

|        |        |                           |
|--------|--------|---------------------------|
| 0xFFD1 | FORMAT | Accelerator sample format |
| ----   | ----   | --gg ddss                 |

| Bit | Name | R/W | Action  |
|-----|------|-----|---|
| 5–4 | g    | R/W | 0 - PCM gain/coef scaling = 1/2048<br>1 - PCM gain/coef scaling = 1/1<br>2 - PCM gain/coef scaling = 1/65536  |
| 3–2 | d    | R/W | 0 - ADPCM decoding from ARAM<br>1 - PCM decoding from <b>ACIN</b> , <b>ACCA</b> doesn't increment<br>2 - PCM decoding from ARAM<br>3 - PCM decoding from <b>ACIN</b> , <b>ACCA</b> increments |
| 1–0 | s    | R/W | 0 - 4-bit<br>1 - 8-bit<br>2 - 16-bit  |

|        |        |           |
|--------|--------|-----------|
| 0xFFD2 | ACUNK1 | Unknown 1 |
| dddd   | dddd   | dddd dddd |

| Bit  | Name | R/W | Action    |
|------|------|-----|-----------|
| 15–0 | d    | R/W | Usually 3 |

|        |        |                 |
|--------|--------|-----------------|
| 0xFFD3 | ACDRAW | Raw ARAM Access |
| dddd   | dddd   | dddd dddd       |

| Bit  | Name | R/W | Action   |
|------|------|-----|--|
| 15–0 | d    | R/W | Reads from or writes to raw data pointed to by current accelerator address, and then increments the current address. Reads respect the <b>FORMAT</b> size. Writes are always 16-bit and treat the addresses as such. Writes require that the uppermost bit of the current address is set. Reads that overflow the end address throw exception <b>ACRROV</b> . Writes that overflow throw exception <b>ACRWOV</b> . |

|        |       |                             |
|--------|-------|-----------------------------|
| 0xFFD4 | ACSAH | Accelerator Start Address H |
| dddd   | dddd  | dddd dddd                   |

| Bit  | Name | R/W | Action                                      |
|------|------|-----|---|
| 15–0 | d    | R/W | Bits 31–16 of the accelerator start address |

|        |       |                             |
|--------|-------|-----------------------------|
| 0xFFD5 | ACSAL | Accelerator Start Address L |
| dddd   | dddd  | dddd dddd                   |

| Bit  | Name | R/W | Action                                     |
|------|------|-----|--|
| 15–0 | d    | R/W | Bits 15–0 of the accelerator start address |

|        |       |                           |
|--------|-------|---------------------------|
| 0xFFD6 | ACEAH | Accelerator End Address H |
| dddd   | dddd  | dddd dddd                 |

| Bit  | Name | R/W | Action                                    |
|------|------|-----|---|
| 15–0 | d    | R/W | Bits 31–16 of the accelerator end address |

|        |       |                           |
|--------|-------|---------------------------|
| 0xFFD7 | ACEAL | Accelerator End Address L |
| dddd   | dddd  | dddd dddd                 |

| Bit  | Name | R/W | Action                                   |
|------|------|-----|--|
| 15–0 | d    | R/W | Bits 15–0 of the accelerator end address |

|        |       |                               |
|--------|-------|-------------------------------|
| 0xFFD8 | ACCAH | Accelerator Current Address H |
| dddd   | dddd  | dddd dddd                     |

| Bit  | Name | R/W | Action  |
|------|------|-----|---|
| 15–0 | d    | R/W | Bits 31–16 of the accelerator current address |

|        |       |                               |
|--------|-------|-------------------------------|
| 0xFFD9 | ACSAH | Accelerator Current Address L |
| dddd   | dddd  | dddd dddd                     |

| Bit  | Name | R/W | Action                                       |
|------|------|-----|--|
| 15–0 | d    | R/W | Bits 15–0 of the accelerator current address |

|        |            |                           |
|--------|------------|---------------------------|
| 0xFFDA | PRED_SCALE | ADPCM predictor and scale |
| ----   | ----       | -ppp ssss                 |

| Bit | Name | R/W | Action   |
|-----|------|-----|--|
| 6–4 | d    | R/W | Used to decide which pair of coefficients to use (COEF_A1_p and COEF_A2_p, at 0xFFA0 + 2p and 0xFFA0 + 2p + 1) |
| 3–0 | s    | R/W | The scale to use, as 2 <sup>s</sup>  |

|        |      |           |
|--------|------|-----------|
| 0xFFDB | YN1  | ADPCM YN1 |
| dddd   | dddd | dddd dddd |

| Bit  | Name | R/W | Action  |
|------|------|-----|---|
| 15-0 | d    | R/W | Last value read by the accelerator, updated to the new value of ACDSAMP when ACDSAMP is read. Used and updated for all sample formats. Multiplied by the A1 coefficient selected by PRED_SCALE and scaled per FORMAT. |

|        |      |           |
|--------|------|-----------|
| 0xFFDC | YN2  | ADPCM YN2 |
| dddd   | dddd | dddd dddd |

| Bit  | Name | R/W | Action   |
|------|------|-----|--|
| 15-0 | d    | R/W | Second-last value read by the accelerator, updated to the previous value of YN1 when ACDSAMP is read. Used and updated for all sample formats. Multiplied by the A2 coefficient selected by PRED_SCALE and scaled per FORMAT. Writing this value starts the accelerator. |

|        |         |                  |
|--------|---------|------------------|
| 0xFFDD | ACDSAMP | Accelerator data |
| dddd   | dddd    | dddd dddd        |

| Bit  | Name | R/W | Action   |
|------|------|-----|--|
| 15-0 | d    | R   | Reads new processed sample data from the accelerator. Data is processed per FORMAT. When there is no data left, returns 0. |

|        |      |           |
|--------|------|-----------|
| 0xFFDE | GAIN | Gain      |
| dddd   | dddd | dddd dddd |

| Bit  | Name | R/W | Action   |
|------|------|-----|--|
| 15-0 | d    | R/W | Applied in PCM FORMATS. Raw sample is multiplied by GAIN, then scaled per the gain scale bits of FORMAT. |

|        |      |                   |
|--------|------|-------------------|
| 0xFFDF | ACIN | Accelerator Input |
| dddd   | dddd | dddd dddd         |

| Bit  | Name | R/W | Action  |
|------|------|-----|---|
| 15-0 | d    | R/W | Used as the sample input in place of ARAM reads when FORMAT specifies it. |

|        |      |                       |
|--------|------|-----------------------|
| 0xFFEF | AMDM | ARAM DMA Request Mask |
| ----   | ---- | ----m                 |

| Bit | Name | R/W | Action                                   |
|-----|------|-----|--|
| 0   | m    | R/W | 0 - DMA with ARAM unmasked<br>1 - masked |

## 4.4 Interrupts

The DSP can raise interrupts at the CPU. Interrupts are usually used to signal that a DSP mailbox has been filled with new data.

|         |      |             |
|---------|------|-------------|
| 0xFFFFB | DIRQ | IRQ Request |
| ----    | ---- | ----I       |

| Bit | Name | R/W | Action                     |
|-----|------|-----|----------------------------|
| 0   | I    | W   | 1 - Raise interrupt at CPU |

## 4.5 Mailboxes

### 4.5.1 DSP Mailbox

The DSP mailbox (DMB) is an interface to send 31 bits of information from the DSP to the CPU.

|        |      |               |
|--------|------|---------------|
| 0xFFFC | DMBH | DSP Mailbox H |
| Mddd   | dddd | dddd dddd     |

| Bit  | Name | R/W | Action  |
|------|------|-----|---|
| 15   | M    | R   | 1 - Mailbox has not been received by CPU<br>0 - Mailbox empty |
|      |      | W   | Does not matter. It will be set when DMBL is written to       |
| 14–0 | d    | W   | Bits 30–16 of mail sent from the DSP to the CPU               |

|        |      |               |
|--------|------|---------------|
| 0xFFFD | DMBL | DSP Mailbox L |
| dddd   | dddd | dddd dddd     |

| Bit  | Name | R/W | Action  |
|------|------|-----|---|
| 15–0 | d    | W   | Bits 15–0 of mail sent from the DSP to the CPU. Writing to this register by the DSP causes the DMBH.M bit to be set, indicating that the mail is ready. |

#### Operation:

Sending mail from the DSP to the CPU can be achieved by writing mail to register DMBH and then to register DMBL in that order. After writing to DMBL, bit DMBH.M will be set, signaling that the mail is ready to be received by the CPU. If the DSP needs to receive a response from the CPU, then it usually waits for the M bit to be cleared after sending a mail. If the DSP does processing when the CPU receives a mail, then it waits for the M bit to be cleared before issuing another mail to the CPU.

### 4.5.2 CPU Mailbox

The CPU Mailbox (CMB) is a register that allows sending 31 bits of information from the CPU to the DSP.

|         |      |               |
|---------|------|---------------|
| 0xFFFFE | CMBH | CPU Mailbox H |
| Mddd    | dddd | dddd dddd     |

| Bit  | Name | R/W | Action  |
|------|------|-----|---|
| 15   | M    | R   | 1 - Mailbox contains mail from the CPU<br>0 - Mailbox empty |
| 14-0 | d    | R   | Bits 30-16 of the mail sent from the CPU                    |

|        |      |               |
|--------|------|---------------|
| 0xFFFF | CMBL | CPU Mailbox L |
| dddd   | dddd | dddd dddd     |

| Bit  | Name | R/W | Action  |
|------|------|-----|---|
| 15-0 | d    | R   | Bits 15-0 of mail sent from the CPU. Reading of this register by the DSP causes the CMBH.M bit to be cleared. |

#### Operation:

From the CPU side, software usually checks the M bit of CMBH. It takes action only in the case that this bit is 0. Said action is to write CMBH first and then CMBL. After writing to CMBL, the mail is ready to be received by the DSP.

From the DSP side, the DSP loops by probing the M bit. When this bit is 1, the DSP reads CMBH first and then CMBL. After reading CMBL, CMBH.M will be cleared.



## Chapter 5

# Opcodes

### 5.1 Opcode syntax

#### Basic opcode syntax:

OPC <opcode parameters>

*Above syntax is correct for all opcodes.*

#### **EXAMPLES:**

```
JMP 0x0300
CALL loop
HALT
```

#### Extended syntax:

OPC'EXOPC <opcode parameters> : <extended opcode parameters>

*Above syntax is correct only for arithmetic opcodes, because those can be extended with additional load/store unit behavior.*

#### **EXAMPLES:**

```
DECM'L $acs0 : $acl.m, @ar0
NX'MV : $acx1.h, $ac0.l
```

## 5.2 Operation — Used Functions

Functions used for describing opcode operation.

`PUSH_STACK($stR)`

**Description:**

Pushes value onto given stack referenced by stack register `$stR`. Operation moves values down in internal stack.

**Operation:**

```
stack_stR[stack_ptr_stR++] = $stR;
```

`POP_STACK($stR)`

**Description:**

Pops value from stack referenced by stack register `$stR`. Operation moves values up in internal stack.

**Operation:**

```
$stR = stack_stR[--stack_ptr_stR];
```

`FLAGS(val)`

**Description:**

Calculates flags depending on given value or result of operation and sets corresponding bits in status register `$sr`.

`EXECUTE_OPCODE(new_pc)`

**Description:**

Executes opcode at the given `new_pc` address.

### 5.3 Bit meanings

Opcode decoding uses special naming for bits and their decimal representations to provide easier understanding of bit fields in the opcode.

| Binary form      | Decimal form | Meaning                                      |
|------------------|--------------|--|
| d, dd, ddd, dddd | D            | Destination register                         |
| s, ss, sss, ssss | S            | Source register                              |
| t, tt, ttt, tttt | T            | Source register                              |
| r, rr, rrr, rrrr | R            | Register (either source or destination)      |
| Aaaaa(a)         | A, addrA     | Address in either instruction or data memory |
| xxxx xxxx        | X            | Extended opcode                              |
| mmm(m)           | M, addrM     | Address in memory                            |
| iii(i)           | I, Imm       | Immediate value                              |
| cccc             | cc           | Condition (see conditional opcodes)          |

## 5.4 Conditional opcodes

Conditional opcodes are executed only when the condition described by their encoded conditional field has been met. The groups of conditional instructions are: **CALLcc**, **Jcc**, **IFcc**, **RETcc**, **RTIcc**, **JRcc**, and **CALLRcc**.

| Bits   | cc  | Name                  | Evaluated expression   |
|--------|-----|-----------------------|--|
| 0b0000 | GE  | Greater than or equal | <code>\$sr.O == \$sr.S</code>  |
| 0b0001 | L   | Less than             | <code>\$sr.O != \$sr.S</code>  |
| 0b0010 | G   | Greater than          | <code>(\$sr.O == \$sr.S) &amp;&amp; (\$sr.Z == 0)</code>                   |
| 0b0011 | LE  | Less than or equal    | <code>(\$sr.O != \$sr.S)    (\$sr.Z != 0)</code>                           |
| 0b0100 | NZ  | Not zero              | <code>\$sr.Z == 0</code>   |
| 0b0101 | Z   | Zero                  | <code>\$sr.Z != 0</code>   |
| 0b0110 | NC  | Not carry             | <code>\$sr.C == 0</code>   |
| 0b0111 | C   | Carry                 | <code>\$sr.C != 0</code>   |
| 0b1000 | x8  | Below s32             | <code>\$sr.AS == 0</code>  |
| 0b1001 | x9  | Above s32             | <code>\$sr.AS != 0</code>  |
| 0b1010 | xA  |                       | <code>(((\$sr.AS != 0)    (\$sr.TB != 0)) &amp;&amp; (\$sr.Z == 0))</code> |
| 0b1011 | xB  |                       | <code>(((\$sr.AS == 0) &amp;&amp; (\$sr.TB == 0))    (\$sr.Z != 0))</code> |
| 0b1100 | LNZ | Not logic zero        | <code>\$sr.LZ == 0</code>  |
| 0b1101 | LZ  | Logic zero            | <code>\$sr.LZ != 0</code>  |
| 0b1110 | O   | Overflow              | <code>\$sr.O != 0</code>   |
| 0b1111 |     | <always>              |  |

### Note:

There are two pairs of conditions that work similarly: Z/NZ and LZ/LNZ. Z/NZ pair operates on arithmetic zero flag (arithmetic 0) while LZ/LNZ pair operates on logic zero flag (logic 0). The logic zero flag is only set by **ANDCF** and **ANDF**.

## 5.5 Flags

Most opcodes update flags in the status register (\$sr) based on their result. (Extended opcodes do not update flags.)

Overflow (O) occurs when the result has wrapped around. The expression  $C = A + B$  has overflowed if  $A > 0$  and  $B > 0$  but  $C \leq 0$  or if  $A < 0$  and  $B < 0$  but  $C \geq 0$ . Any instruction that sets the O flag will also set the OS flag; when the O flag is set, OS is also set, but OS is not cleared when O is cleared.

Carry (C) occurs when an arithmetic carry occurs and should be added to the next most significant word. The expression  $C = A + B$  generates a carry if  $A > C$ . The DSP uses different logic for subtraction: the expression  $C = A - B$  generates a carry if  $A \geq C$  (so if  $B = 0$ , a carry is generated for all  $A$ ). This is because the DSP uses a carry flag, not a borrow flag.

Each instruction has a table showing what flags it updates, such as this:

**Flags:**

| OS | LZ | TB | S32 | S | AZ | O | C |
|----|----|----|-----|---|----|---|---|
| -  | -  | X  | X   | X | X  | 0 | 0 |

A “-” indicates that the flag retains its previous value, a “0” indicates that the flag is set to 0, and a “X” indicates that the value of the flag changes depending on what the instruction did.

## 5.6 Alphabetical list of opcodes

5.6.1 ABS

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1010 |    |    |    | d001 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

```
ABS $acD
```

Description:

Sets \$acD to the absolute value of \$acD.

Operation:

```
IF $acD < 0
    $acD = -$acD
ENDIF
FLAGS($acD)
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

5.6.2 ADD

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0100 |    |    |    | 110d |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

ADD \$acD , \$ac(1-D)

Description:

Adds accumulator \$ac(1-D) to accumulator register \$acD.

Operation:

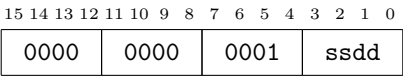
\$acD += \$ac(1-D)  
FLAGS(\$acD)  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |



5.6.3 ADDARN



Format:

```
ADDARN $arD, $ixS
```

Description:

Adds indexing register \$ixS to an addressing register \$arD.

Operation:

```
$arD += $ixS
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.4 ADDAX

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0100 |    |    |    | 10sd |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

ADDAX \$acD, \$axS

Description:

Adds secondary accumulator \$axS to accumulator register \$acD.

Operation:

\$acD += \$axS  
FLAGS(\$acD)  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

5.6.5 ADDAXL

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0111 |    |    |    | 00sd |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

```
ADDAXL $acD, $axS.1
```

Description:

Adds secondary accumulator `$axS.1` to accumulator register `$acD`. `$axS.1` is treated as an unsigned value.

Operation:

```
$acD += $axS.1
FLAGS($acD)
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

5.6.6 ADDI

|                                       |      |      |      |
|---------------------------------------|------|------|------|
| 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |      |      |      |
| 0000                                  | 001d | 0000 | 0000 |
| iiii                                  | iiii | iiii | iiii |

Format:

```
ADDI $acD, #I
```

Description:

Adds a 16-bit sign-extended immediate to mid accumulator `$acD.hm`.

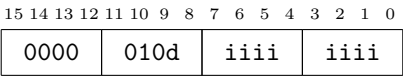
Operation:

```
$acD.hm += #I
FLAGS($acD)
$pc += 2
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

5.6.7 ADDIS



Format:

```
ADDIS $acD, #I
```

Description:

Adds an 8-bit sign-extended immediate to mid accumulator `$acD.hm`.

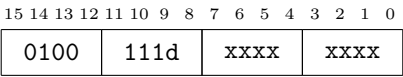
Operation:

```
$acD.hm += #I
FLAGS($acD)
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

5.6.8 ADDP



Format:

ADDP \$acD

Description:

Adds the product register to the accumulator register.

Operation:

\$acD += \$prod  
FLAGS(\$acD)  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

**5.6.9 ADDPAXZ**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1111 |    |    |    | 10sd |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

**Format:**

ADDPAXZ \$acD, \$axS

**Description:**

Adds secondary accumulator \$axS to product register and stores result in accumulator register. Low 16-bits of \$acD (\$acD.l) are set to 0.

**Operation:**

\$acD.hm = \$prod.hm + \$ax.h  
 \$acD.l = 0  
 FLAGS(\$acD)  
 \$pc++

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | X |

5.6.10 ADDR

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0100 |    |    |    | 0ssd |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

ADDR \$acD , \$(0x18+S)

Description:

Adds register \$(0x18+S) to the accumulator \$acD register.

Operation:

\$acD += (\$(0x18+S) << 16)  
FLAGS(\$acD)  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |



**5.6.11 ANDC**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0011 |    |    |    | 110d |    |   |   | 0xxx |   |   |   | xxxx |   |   |   |

**Format:**

ANDC \$acD.m, \$ac(1-D).m

**Description:**

Logic AND middle part of accumulator \$acD.m with middle part of accumulator \$ac(1-D).m.

**Operation:**

\$acD.m &= \$ac(1-D).m  
 FLAGS(\$acD)  
 \$pc++

**Note:**

The main opcode is 9 bits and the extension opcode is 7 bits. The extension opcode is treated as if the 8th bit was 0 (i.e. it is 0xxxxxxx).

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

**5.6.12 ANDCF**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 001d |    |   |   | 1100 |   |   |   | 0000 |   |   |   |
| iiii |    |    |    | iiii |    |   |   | iiii |   |   |   | iiii |   |   |   |

**Format:**

ANDCF \$acD.m, #I

**Description:**

Sets the logic zero (LZ) flag in status register \$sr if the result of the logical AND operation involving the mid part of accumulator \$acD.m and the immediate value I is equal to immediate value I. If the logical AND operation does not result in a value equal to I, then the LZ flag is cleared.

**Operation:**

```
IF ($acD.m & I) == I
    $sr.LZ = 1
ELSE
    $sr.LZ = 0
ENDIF
$pc += 2
```

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | X  | -  | -   | - | -  | - | - |

**5.6.13 ANDF**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 001d |    |   |   | 1010 |   |   |   | 0000 |   |   |   |
| iiii |    |    |    | iiii |    |   |   | iiii |   |   |   | iiii |   |   |   |

**Format:**

ANDF \$acD.m, #I

**Description:**

Sets the logic zero (LZ) flag in status register \$sr if the result of the logic AND operation involving the mid part of accumulator \$acD.m and the immediate value I is equal to zero. If the result is not equal to zero, then the LZ flag is cleared.

**Operation:**

```
IF ($acD.m & I) == 0
    $sr.LZ = 1
ELSE
    $sr.LZ = 0
ENDIF
$pc += 2
```

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | X  | -  | -   | - | -  | - | - |

**5.6.14 ANDI**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 001d |    |   |   | 0100 |   |   |   | 0000 |   |   |   |
| iiii |    |    |    | iiii |    |   |   | iiii |   |   |   | iiii |   |   |   |

**Format:**

ANDI \$acD.m, #I

**Description:**

Performs a logical AND with the mid part of accumulator \$acD.m and the immediate value I.

**Operation:**

\$acD.m &= #I  
 FLAGS(\$acD)  
 \$pc += 2

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

**5.6.15 ANDR**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0011 |    |    |    | 01sd |    |   |   | 0xxx |   |   |   | xxxx |   |   |   |

**Format:**

ANDR \$acD.m, \$axS.h

**Description:**

Performs a logical AND with the middle part of accumulator \$acD.m and the high part of secondary accumulator, \$axS.h.

**Operation:**

\$acD.m &= \$axS.h  
 FLAGS(\$acD)  
 \$pc++

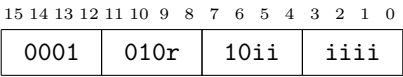
**Note:**

The main opcode is 9 bits and the extension opcode is 7 bits. The extension opcode is treated as if the 8th bit was 0 (i.e. it is 0xxxxxxx).

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

5.6.16 ASL



Format:

```
ASL $acR, #I
```

Description:

Arithmetically left shifts the accumulator `$acR` by the amount specified by immediate `I`.

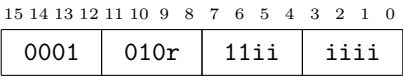
Operation:

```
$acR <<= I
FLAGS($acR)
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

5.6.17 ASR



Format:

```
ASR $acR, #I
```

Description:

Arithmetically right shifts accumulator `$acR` specified by the value calculated by negating sign-extended bits 0-6.

Operation:

```
IF I != 0
    $acR >>= (64 - I)
ENDIF
FLAGS($acR)
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

**5.6.18 ASRN**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 0010 |    |   |   | 1101 |   |   |   | 1011 |   |   |   |

**Format:**

ASRN

**Description:**

Arithmetically shifts accumulator `$ac0` either left or right based on `$ac1.m`: if bit 6 is set, a right by the amount calculated by negating sign-extended bits 0–5 occurs, while if bit 6 is clear, a left shift occurs by bits 0–5.

**Operation:**

```

IF ($ac1.m & 64)
    IF ($ac1.m & 63) != 0
        $ac0 >>= (64 - ($ac1.m & 63))
    ENDIF
ELSE
    $ac0 <<= $ac1.m
ENDIF
FLAGS($ac0)
$pc++

```

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |



**5.6.19 ASRNR**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0011 |    |    |    | 111d |    |   |   | 1xxx |   |   |   | xxxx |   |   |   |

**Format:**

ASRNR \$acD

**Description:**

Arithmetically shifts accumulator \$acD either left or right based on \$ac(1-D).m: if bit 6 is set, a right shift by the amount calculated by negating sign-extended bits 0–5 occurs, while if bit 6 is clear, a left shift occurs by bits 0–5.

**Operation:**

```

IF ($ac(1-D).m & 64)
    IF ($ac(1-D).m & 63) != 0
        $acD >>= (64 - ($ac(1-D).m & 63))
    ENDIF
ELSE
    $acD <<= $ac(1-D).m
ENDIF
FLAGS($acD)
$pc++

```

**Note:**

The main opcode is 9 bits and the extension opcode is 7 bits. The extension opcode is treated as if the 8th bit was 0 (i.e. it is 0xxxxxxx).

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

5.6.20 ASRNRX

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0011 |    |    |    | 10sd |    |   |   | 1xxx |   |   |   | xxxx |   |   |   |

Format:

ASRNRX \$acD, \$axS.h

Description:

Arithmetically shifts accumulator \$acD either left or right based on \$axS.h: if bit 6 is set, a right by the amount calculated by negating sign-extended bits 0–5 occurs, while if bit 6 is clear, a left shift occurs by bits 0–5.

Operation:

```
IF ($axS.h & 64)
    IF ($axS.h & 63) != 0
        $acD >>= (64 - ($axS.h & 63))
    ENDIF
ELSE
    $acD <<= $axS.h
ENDIF
FLAGS($acD)
$pc++
```

Note:

The main opcode is 9 bits and the extension opcode is 7 bits. The extension opcode is treated as if the 8th bit was 0 (i.e. it is 0xxxxxxx).

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

5.6.21 ASR16

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1001 |    |    |    | r001 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

ASR16 \$acR

Description:

Arithmetically right shifts accumulator \$acR by 16.

Operation:

\$acR >>= 16  
FLAGS(\$acR)  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

**5.6.22 BLOOP**

|      |      |      |      |    |    |   |   |   |   |   |   |   |   |   |   |
|------|------|------|------|----|----|---|---|---|---|---|---|---|---|---|---|
| 15   | 14   | 13   | 12   | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0000 | 0000 | 011r | rrrr |    |    |   |   |   |   |   |   |   |   |   |   |
| aaaa | aaaa | aaaa | aaaa |    |    |   |   |   |   |   |   |   |   |   |   |

**Format:**

BLOOP \$R, addrA

**Description:**

Repeatedly execute a block of code starting at the following opcode until the counter specified by the value from register **\$R** reaches zero. Block ends at specified address **addrA** inclusive. i.e. opcode at **addrA** is the last opcode included in loop. Counter is pushed on loop stack **\$st3**, end of block address is pushed on loop stack **\$st2** and the repeat address is pushed on call stack **\$st0**. Up to 4 nested loops are allowed.

When using **\$ac0.m** or **\$ac1.m** as the initial counter value, optionally apply saturation depending on the value of **\$sr.SXM** (see [16-bit and 40-bit modes](#)).

**Operation:**

```

$st0 = $pc + 2
$st2 = addrA
$st3 = $R
$pc += 2

// On real hardware, the below does not happen,
// this opcode only sets stack registers
WHILE ($st3)
    DO
        EXECUTE_OPCODE($pc)
        WHILE($pc != $st2)
            $st3--
            $pc = $st0
    END
    $pc = addrA + 1
    // Remove values from stack

```

**Flags:**

| OS | LZ | TB | S32 | S | AZ | O | C |
|----|----|----|-----|---|----|---|---|
| -  | -  | -  | -   | - | -  | - | - |

**5.6.23 BLOOPI**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0001 |    |    |    | 0001 |    |   |   | iiii |   |   |   | iiii |   |   |   |
| aaaa |    |    |    | aaaa |    |   |   | aaaa |   |   |   | aaaa |   |   |   |

**Format:**

BLOOPI #I, addrA

**Description:**

Repeatedly execute a block of code starting at the following opcode until the counter specified by the immediate value I reaches zero. Block ends at specified address **addrA** inclusive. i.e. opcode at **addrA** is the last opcode included in loop. Counter is pushed on loop stack **\$st3**, end of block address is pushed on loop stack **\$st2** and the repeat address is pushed on call stack **\$st0**. Up to 4 nested loops are allowed.

**Operation:**

```

$st0 = $pc + 2
$st2 = addrA
$st3 = I
$pc += 2

// On real hardware, the below does not happen,
// this opcode only sets stack registers
WHILE ($st3)
    DO
        EXECUTE_OPCODE($pc)
    WHILE($pc != $st2)
        $st3--
        $pc = $st0
END
$pc = addrA + 1
// Remove values from stack

```

**Flags:**

| OS | LZ | TB | S32 | S | AZ | O | C |
|----|----|----|-----|---|----|---|---|
| -  | -  | -  | -   | - | -  | - | - |

**5.6.24 CALL**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 0010 |    |   |   | 1011 |   |   |   | 1111 |   |   |   |
| aaaa |    |    |    | aaaa |    |   |   | aaaa |   |   |   | aaaa |   |   |   |

**Format:**

CALL addressA

**Description:**

Call function. Push program counter of the instruction following “call” to call stack `$st0`. Set program counter to address represented by the value that follows this **CALL** instruction.

**Operation:**

```
// Must skip value that follows "call"
PUSH_STACK($st0)
$st0 = $pc + 2
$pc = addressA
```

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

**5.6.25 CALLcc**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 0010 |    |   |   | 1011 |   |   |   | cccc |   |   |   |
| aaaa |    |    |    | aaaa |    |   |   | aaaa |   |   |   | aaaa |   |   |   |

**Format:**

CALLcc addressA

**Description:**

Call function if condition cc has been met. Push program counter of the instruction following “call” to call stack \$st0. Set program counter to address represented by the value that follows this [CALL](#) instruction.

**Operation:**

```
// Must skip value that follows "call"
IF (cc)
    PUSH_STACK($st0)
    $st0 = $pc + 2
    $pc = addressA
ELSE
    $pc += 2
ENDIF
```

**Flags:**

| OS | LZ | TB | S32 | S | AZ | O | C |
|----|----|----|-----|---|----|---|---|
| -  | -  | -  | -   | - | -  | - | - |

**5.6.26 CALLR**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0001 |    |    |    | 0111 |    |   |   | rrr1 |   |   |   | 1111 |   |   |   |

**Format:**

CALLR \$R

**Description:**

Call function. Push program counter of the instruction following “call” to call stack \$st0. Set program counter to register \$R.

**Operation:**

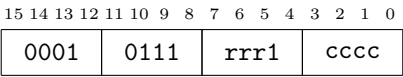
```
PUSH_STACK($st0)
$st0 = $pc + 1
$pc = $R
```

**Flags:**

| OS | LZ | TB | S32 | S | AZ | O | C |
|----|----|----|-----|---|----|---|---|
| -  | -  | -  | -   | - | -  | - | - |



5.6.27 CALLRcc



Format:

```
CALLRcc $R
```

Description:

Call function if condition cc has been met. Push program counter of the instruction following “call” to call stack \$st0. Set program counter to register \$R.

Operation:

```
IF (cc)
    PUSH_STACK($st0)
    $st0 = $pc + 1
    $pc = $R
ELSE
    $pc++
ENDIF
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.28 CLR15

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1000 |    |    |    | 1100 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

CLR15

Description:

Sets `$sr.SU` (bit 15) to 0, causing multiplication to treat its operands as signed.

Operation:

`$sr &= ~0x8000`  
`$pc++`

See also:

[SET15](#)

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.29 CLR

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1000 |    |    |    | r001 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

CLR \$acR

Description:

Clears accumulator \$acR.

Operation:

\$acR = 0  
FLAGS(\$acR)  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | 1  | 0   | 0 | 1  | 0 | 0 |

**5.6.30 CLRL**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1111 |    |    |    | 110r |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

**Format:**

CLRL \$acR.l

**Description:**

Rounds \$acR such that \$acR.l is 0. This is a round-to-even operation.

**Operation:**

```
IF ($acR & 0x10000) != 0
    $acR = ($acR + 0x8000) & ~0xffff
ELSE
    $acR = ($acR + 0x7fff) & ~0xffff
ENDIF
FLAGS($acR)
$pc++
```

**Note:**

An alternative interpretation is that if \$acR.m is odd, then increment \$acsR if \$acR.l is greater than or equal to 0x8000; if \$acR.m is even, then increment \$acsR if \$acR.l is greater than or equal to 0x7fff. Afterwards set \$acR.l to 0.

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

**5.6.31 CLRP**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1000 |    |    |    | 0100 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

**Format:**

CLRP

**Description:**Clears product register `$prod`.**Operation:**

```
$prod = 0 // See note below
$pc++
```

**Note:**

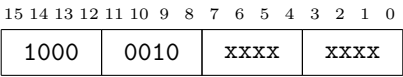
Actually product register gets cleared by setting registers with following values:

```
$prod.l  = 0x0000
$prod.m1 = 0xffff0
$prod.h  = 0x00ff
$prod.m2 = 0x0010
```

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.32    **CMP**



**Format:**

CMP

**Description:**

Compares accumulator `$ac0` with accumulator `$ac1`.

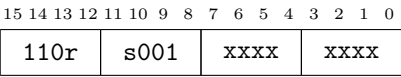
**Operation:**

```
$sr = FLAGS($ac0 - $ac1)
$pc++
```

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

5.6.33 CMPAXH



Format:

```
CMPAXH $acS, $axR.h
```

Description:

Compares accumulator `$acS` with high part of secondary accumulator `$axR.h`.

Operation:

```
$sr = FLAGS($acS - ($axR.h << 16))
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

**5.6.34 CMPI**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 001d |    |   |   | 1000 |   |   |   | 0000 |   |   |   |
| iiii |    |    |    | iiii |    |   |   | iiii |   |   |   | iiii |   |   |   |

**Format:**

CMPI \$acD, #I

**Description:**

Compares accumulator with immediate. Comparison is performed by subtracting the immediate (16-bit sign-extended) from mid accumulator \$acD.hm and computing flags based on whole accumulator \$acD.

**Operation:**

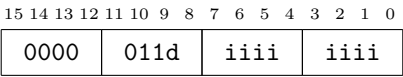
FLAGS(\$acD - (I << 16))  
\$pc += 2

**Flags:**

| OS | LZ | TB | S32 | S | AZ | O | C |
|----|----|----|-----|---|----|---|---|
| X  | -  | X  | X   | X | X  | X | X |



5.6.35    **CMPIS**



**Format:**

```
CMPIS $acD, #I
```

**Description:**

Compares accumulator with short immediate. Comparison is performed by subtracting the short immediate (8-bit sign-extended) from mid accumulator `$acD.hm` and computing flags based on whole accumulator `$acD`.

**Operation:**

```
FLAGS($acD - (I << 16))
$pc++
```

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

5.6.36 DAR

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 0000 |    |   |   | 0000 |   |   |   | 01dd |   |   |   |

Format:

DAR \$arD

Description:

Decrement address register \$arD.

Operation:

\$arD--  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.37 DEC

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0111 |    |    |    | 101d |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

**Format:**

DEC \$acD

**Description:**

Decrements accumulator \$acD.

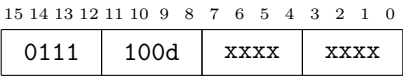
**Operation:**

\$acD--  
FLAGS(\$acD)  
\$pc++

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

5.6.38    **DECM**



**Format:**

DECM \$acsD

**Description:**

Decrements 24-bit mid-accumulator \$acsD.

**Operation:**

\$acsD--  
FLAGS(\$acD)  
\$pc++

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

5.6.39    **HALT**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 0000 |    |   |   | 0010 |   |   |   | 0001 |   |   |   |

**Format:**

HALT

**Description:**

Stops execution of DSP code. Sets bit DSP\_CR\_HALT in register DREG\_CR.

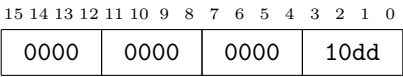
**Operation:**

DREG\_CR |= DSP\_CR\_HALT;

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.40 IAR



**Format:**

IAR \$arD

**Description:**

Increment address register \$arD.

**Operation:**

\$arD++  
\$pc++

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.41 IFcc

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 0010 |    |   |   | 0111 |   |   |   | cccc |   |   |   |

Format:

IFcc

Description:

Executes the following opcode if the condition described by cccc has been met.

Operation:

```
IF (cc)
    EXECUTE_OPCODE($pc + 1)
ELSE
    $pc += 2
ENDIF
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.42 ILRR

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 001d |    |   |   | 0001 |   |   |   | 00ss |   |   |   |

**Format:**

ILRR \$acD.m, @\$arS

**Description:**

Move value from instruction memory pointed by addressing register \$arS to mid accumulator register \$acD.m.

Optionally perform sign extension depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

**Operation:**

\$acD.m = MEM[\$arS]  
\$pc++

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |



5.6.43 ILRRD

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 001d |    |   |   | 0001 |   |   |   | 01ss |   |   |   |

Format:

```
ILRRD $acD.m, @$arS
```

Description:

Move value from instruction memory pointed by addressing register `$arS` to mid accumulator register `$acD.m`. Decrement addressing register `$arS`.  
Optionally perform sign extension depending on the value of `$sr.SXM` (see [16-bit and 40-bit modes](#)).

Operation:

```
$acD.m = MEM[$arS]  
$arS--  
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

**5.6.44 ILRRI**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 001d |    |   |   | 0001 |   |   |   | 10ss |   |   |   |

**Format:**

ILRRI \$acD.m, @\$S

**Description:**

Move value from instruction memory pointed by addressing register \$arS to mid accumulator register \$acD.m. Increment addressing register \$arS.

Optionally perform sign extension depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

**Operation:**

\$acD.m = MEM[\$arS]  
 \$arS++  
 \$pc++

**Flags:**

| OS | LZ | TB | S32 | S | AZ | O | C |
|----|----|----|-----|---|----|---|---|
| -  | -  | -  | -   | - | -  | - | - |

**5.6.45 ILRRN**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 001d |    |   |   | 0001 |   |   |   | 11ss |   |   |   |

**Format:**

```
ILRRN $acD.m, @$arS
```

**Description:**

Move value from instruction memory pointed by addressing register **\$arS** to mid accumulator register **\$acD.m**. Add corresponding indexing register **\$ixS** to addressing register **\$arS**.

Optionally perform sign extension depending on the value of **\$sr.SXM** (see [16-bit and 40-bit modes](#)).

**Operation:**

```
$acD.m = MEM[$arS]
$arS += $ixS
$pc++
```

**Flags:**

| OS | LZ | TB | S32 | S | AZ | O | C |
|----|----|----|-----|---|----|---|---|
| -  | -  | -  | -   | - | -  | - | - |

5.6.46 INC

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0111 |    |    |    | 011d |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

INC \$acD

Description:

Increments accumulator \$acD.

Operation:

\$acD++  
FLAGS(\$acD)  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

5.6.47 INCM

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0111 |    |    |    | 010d |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

INCM \$acsD

Description:

Increments 24-bit mid-accumulator \$acsD.

Operation:

\$acsD++  
FLAGS(\$acD)  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

5.6.48    **JMP**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 0010 |    |   |   | 1001 |   |   |   | 1111 |   |   |   |
| aaaa |    |    |    | aaaa |    |   |   | aaaa |   |   |   | aaaa |   |   |   |

**Format:**

**JMP**   addressA

**Description:**

        Jumps to addressA. Set program counter to the address represented by the value that follows this **JMP** instruction.

**Operation:**

        \$pc = addressA

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.49 Jcc

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 0010 |    |   |   | 1001 |   |   |   | cccc |   |   |   |
| aaaa |    |    |    | aaaa |    |   |   | aaaa |   |   |   | aaaa |   |   |   |

Format:

```
Jcc addressA
```

Description:

Jumps to `addressA` if condition `cc` has been met. Set program counter to the address represented by the value that follows this `Jcc` instruction.

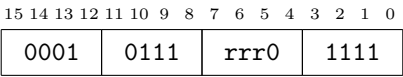
Operation:

```
IF (cc)
    $pc = addressA
ELSE
    $pc += 2
ENDIF
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.50 JMPR



Format:

JMPR \$R

Description:

Jump to address; set program counter to a value from register \$R.

Operation:

\$pc = \$R

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |



**5.6.51 JRcc**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0001 |    |    |    | 0111 |    |   |   | rrr0 |   |   |   | cccc |   |   |   |

**Format:**

JRcc \$R

**Description:**

Jump to address if condition `cc` has been met; set program counter to a value from register `$R`.

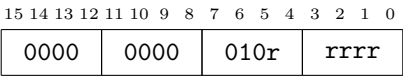
**Operation:**

```
IF (cc)
    $pc = $R
ELSE
    $pc++
ENDIF
```

**Flags:**

| OS | LZ | TB | S32 | S | AZ | O | C |
|----|----|----|-----|---|----|---|---|
| -  | -  | -  | -   | - | -  | - | - |

5.6.52 LOOP



**Format:**

LOOP \$R

**Description:**

Repeatedly execute the following opcode until the counter specified by the value from register \$R reaches zero. Each execution decrements the counter. Register \$R remains unchanged. If register \$R is set to zero at the beginning of loop then the looped instruction will not get executed.

When using \$ac0.m or \$ac1.m as the initial counter value, optionally apply saturation depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

**Operation:**

```
counter = $R
WHILE (counter--)
    EXECUTE_OPCODE($pc + 1)
END
$pc += 2
```

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

**5.6.53 LOOPI**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0001 |    |    |    | 0000 |    |   |   | iiii |   |   |   | iiii |   |   |   |

**Format:**

LOOPI #I

**Description:**

Repeatedly execute the following opcode until the counter specified by immediate value I reaches zero. Each execution decrements the counter. If immediate I is set to zero at the beginning of loop then the looped instruction will not get executed.

**Operation:**

```

counter = I
WHILE (counter--)
    EXECUTE_OPCODE($pc + 1)
END
$pc += 2

```

**Flags:**

| OS | LZ | TB | S32 | S | AZ | O | C |
|----|----|----|-----|---|----|---|---|
| -  | -  | -  | -   | - | -  | - | - |

5.6.54 LR

|                                       |  |  |  |      |  |  |  |      |  |  |  |      |  |  |  |
|---------------------------------------|--|--|--|------|--|--|--|------|--|--|--|------|--|--|--|
| 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |  |  |  |      |  |  |  |      |  |  |  |      |  |  |  |
| 0000                                  |  |  |  | 0000 |  |  |  | 110d |  |  |  | dddd |  |  |  |
| mmm                                   |  |  |  | mmm  |  |  |  | mmm  |  |  |  | mmm  |  |  |  |

Format:

LR \$D, @M

Description:

Move value from data memory pointed by address M to register \$D.

When loading to \$ac0.m or \$ac1.m, optionally perform sign extension depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

Operation:

\$D = MEM[M]  
\$pc += 2

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.55 LRI

|                                       |  |  |  |      |  |  |  |      |  |  |  |      |  |  |  |
|---------------------------------------|--|--|--|------|--|--|--|------|--|--|--|------|--|--|--|
| 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |  |  |  |      |  |  |  |      |  |  |  |      |  |  |  |
| 0000                                  |  |  |  | 0000 |  |  |  | 100d |  |  |  | dddd |  |  |  |
| iiii                                  |  |  |  | iiii |  |  |  | iiii |  |  |  | iiii |  |  |  |

Format:

```
LRI $D, #I
```

Description:

Load immediate value I to register \$D.

When loading to \$ac0.m or \$ac1.m, optionally perform sign extension depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

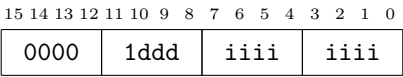
Operation:

```
$D = I
$pc += 2
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.56 LRIS



Format:

```
LRIS $(0x18+D), #I
```

Description:

Load immediate value I (8-bit sign-extended) to accumulator register \$(0x18+D).  
When loading to \$ac0.m or \$ac1.m, optionally perform sign extension depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

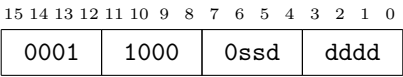
Operation:

```
$(0x18+D) = I  
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.57 LRR



Format:

```
LRR $D, @$arS
```

Description:

Move value from data memory pointed by addressing register `$arS` to register `$D`.  
When loading to `$ac0.m` or `$ac1.m`, optionally perform sign extension depending on the value of `$sr.SXM` (see [16-bit and 40-bit modes](#)).

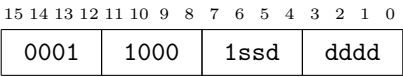
Operation:

```
$D = MEM[$arS]  
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.58    **LRRD**



**Format:**

```
LRRD $D, @$arS
```

**Description:**

Move value from data memory pointed by addressing register `$arS` to register `$D`. Decrements register `$arS`.

When loading to `$ac0.m` or `$ac1.m`, optionally perform sign extension depending on the value of `$sr.SXM` (see [16-bit and 40-bit modes](#)).

**Operation:**

```
$D = MEM[$arS]
$arS--
$pc++
```

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |



**5.6.59 LRRl**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0001 |    |    |    | 1001 |    |   |   | 0ssd |   |   |   | dddd |   |   |   |

**Format:**

LRRl \$D, @\$arS

**Description:**

Move value from data memory pointed by addressing register **\$arS** to register **\$D**. Increments register **\$arS**.

When loading to **\$ac0.m** or **\$ac1.m**, optionally perform sign extension depending on the value of **\$sr.SXM** (see [16-bit and 40-bit modes](#)).

**Operation:**

\$D = MEM[\$arS]  
 \$arS++  
 \$pc++

**Flags:**

| OS | LZ | TB | S32 | S | AZ | O | C |
|----|----|----|-----|---|----|---|---|
| -  | -  | -  | -   | - | -  | - | - |

**5.6.60 LRRN**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0001 |    |    |    | 1001 |    |   |   | 1ssd |   |   |   | dddd |   |   |   |

**Format:**

LRRN \$D, @\$arS

**Description:**

Move value from data memory pointed by addressing register **\$arS** to register **\$D**. Add indexing register **\$ixS** to register **\$arS**.

When loading to **\$ac0.m** or **\$ac1.m**, optionally perform sign extension depending on the value of **\$sr.SXM** (see [16-bit and 40-bit modes](#)).

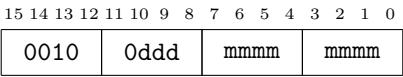
**Operation:**

```
$D = MEM[$arS]
$arS += $ixS
$pc++
```

**Flags:**

| OS | LZ | TB | S32 | S | AZ | O | C |
|----|----|----|-----|---|----|---|---|
| -  | -  | -  | -   | - | -  | - | - |

5.6.61 LRS



Format:

```
LRS $(0x18+D), @M
```

Description:

Move value from data memory pointed by address  $(\$cr \ll 8) \mid M$  to register  $$(0x18+D)$ .  
When loading to  $$ac0.m$  or  $$ac1.m$ , optionally perform sign extension depending on the value of  $$sr.SXM$  (see [16-bit and 40-bit modes](#)).

Operation:

```
$(0x18+D) = MEM[( $cr << 8) | M]  
$pc++
```

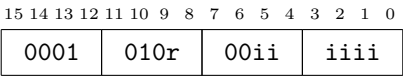
Note:

[LRS](#) can use  $$axD$ , but cannot use  $$acD.h$ , while [SRS](#) and [SRSH](#) only work on  $$acS$ .

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.62 LSL



Format:

```
LSL $acR, #I
```

Description:

Logically left shifts accumulator `$acR` by the amount specified by value `I`.

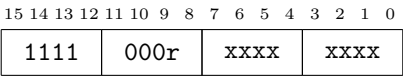
Operation:

```
$acR <<= I
FLAGS($acR)
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

5.6.63 LSL16



Format:

```
LSL16 $acR
```

Description:

Logically left shifts accumulator `$acR` by 16.

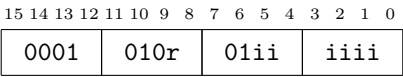
Operation:

```
$acR <<= 16
FLAGS($acR)
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

5.6.64 LSR



Format:

```
LSR $acR, #I
```

Description:

Logically right shifts accumulator `$acR` by the amount calculated by negating sign-extended bits 0–6.

Operation:

```
IF I != 0
    $acR >>= (64 - I)
ENDIF
FLAGS($acR)
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

5.6.65 LSRN

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 0010 |    |   |   | 1100 |   |   |   | 1010 |   |   |   |

Format:

LSRN

Description:

Logically shifts accumulator `$ac0` either left or right based on `$ac1.m`: if bit 6 is set, a right by the amount calculated by negating sign-extended bits 0–5 occurs, while if bit 6 is clear, a left shift occurs by bits 0–5.

Operation:

```
IF ($ac1.m & 64)
    IF ($ac1.m & 63) != 0
        $ac0 >>= (64 - ($ac1.m & 63))
    ENDIF
ELSE
    $ac0 <<= $ac1.m
ENDIF
FLAGS($ac0)
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

**5.6.66 LSRNR**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0011 |    |    |    | 110d |    |   |   | 1xxx |   |   |   | xxxx |   |   |   |

**Format:**

LSRNR \$acD

**Description:**

Logically shifts accumulator \$acD either left or right based on \$ac(1-D).m: if bit 6 is set, a right by the amount calculated by negating sign-extended bits 0–5 occurs, while if bit 6 is clear, a left shift occurs by bits 0–5.

**Operation:**

```

IF ($ac(1-D).m & 64)
    IF ($ac(1-D).m & 63) != 0
        $acD >>= (64 - ($ac(1-D).m & 63))
    ENDIF
ELSE
    $acD <<= $ac(1-D).m
ENDIF
FLAGS($acD)
$pc++

```

**Note:**

The main opcode is 9 bits and the extension opcode is 7 bits. The extension opcode is treated as if the 8th bit was 0 (i.e. it is 0xxxxxxx).

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |



**5.6.67 LSRNRX**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0011 |    |    |    | 01sd |    |   |   | 1xxx |   |   |   | xxxx |   |   |   |

**Format:**

LSRNRX \$acD, \$axS.h

**Description:**

Logically shifts accumulator \$acD either left or right based on \$axS.h: if bit 6 is set, a right by the amount calculated by negating sign-extended bits 0–5 occurs, while if bit 6 is clear, a left shift occurs by bits 0–5.

**Operation:**

```

IF ($axS.h & 64)
    IF ($axS.h & 63) != 0
        $acD >>= (64 - ($axS.h & 63))
    ENDIF
ELSE
    $acD <<= $axS.h
ENDIF
FLAGS($acD)
$pc++

```

**Note:**

The main opcode is 9 bits and the extension opcode is 7 bits. The extension opcode is treated as if the 8th bit was 0 (i.e. it is 0xxxxxxx).

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

5.6.68 LSR16

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1111 |    |    |    | 010r |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

LSR16 \$acR

Description:

Logically right shifts accumulator \$acR by 16.

Operation:

\$acR >>= 16  
FLAGS(\$acR)  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

5.6.69 M0

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1000 |    |    |    | 1011 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

M0

Description:

Sets `$sr.AM` (bit 13) to 1, **disabling** the functionality that doubles the result of every multiply operation.

Operation:

`$sr |= 0x2000`  
`$pc++`

See also:

[M2](#)

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.70 M2

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1000 |    |    |    | 1010 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

M2

Description:

Sets `$sr.AM` (bit 13) to 0, **enabling** the functionality that doubles the result of every multiply operation.

Operation:

```
$sr &= ~0x2000
$pc++
```

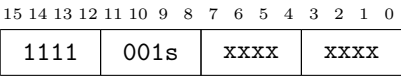
See also:

[M0](#)

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.71 MADD



Format:

```
MADD $axS.l, $axS.h
```

Description:

Multiply low part `$axS.l` of secondary accumulator `$axS` by high part `$axS.h` of secondary accumulator `$axS` (treat them both as signed) and add result to product register.

Operation:

```
$prod += $axS.l * $axS.h
$pc++
```

See also:

`$sr.AM` bit affects multiply result.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

**5.6.72 MADDC**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1110 |    |    |    | 10st |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

**Format:**

MADDC \$acS.m, \$axT.h

**Description:**

Multiply middle part of accumulator \$acS.m by high part of secondary accumulator \$axT.h (treat them both as signed) and add result to product register.

**Operation:**

```
$prod += $acS.m * $axT.h
$pc++
```

**See also:**

\$sr.AM bit affects multiply result.

**Flags:**

| OS | LZ | TB | S32 | S | AZ | O | C |
|----|----|----|-----|---|----|---|---|
| -  | -  | -  | -   | - | -  | - | - |

**5.6.73 MADDX**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1110 |    |    |    | 00st |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

**Format:**

MADDX \$(0x18+S\*2), \$(0x19+T\*2)

**Description:**

Multiply one part of secondary accumulator **\$ax0** (selected by **S**) by one part of secondary accumulator **\$ax1** (selected by **T**) (treat them both as signed) and add result to product register.

**Operation:**

\$prod += \$(0x18+S\*2) \* \$(0x19+T\*2)  
\$pc++

**See also:**

\$sr.AM bit affects multiply result.

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.74 MOV

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0110 |    |    |    | 110d |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

MOV \$acD, \$ac(1-D)

Description:

Moves accumulator \$ac(1-D) to accumulator \$acD.

Operation:

\$acD = \$ac(1-D)  
FLAGS(\$acD)  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | 0   | X | X  | 0 | 0 |



5.6.75 MOVAX

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0110 |    |    |    | 10sd |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

```
MOVAX $acD, $axS
```

Description:

Moves secondary accumulator `$axS` to accumulator `$acD`.

Operation:

```
$acD = $axS
FLAGS($acD)
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

5.6.76 MOVNP

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0111 |    |    |    | 111d |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

MOVNP \$acD

Description:

Moves negated multiply product from the \$prod register to the accumulator register \$acD.

Operation:

\$acD = -\$prod  
FLAGS(\$acD)  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | X |

5.6.77 MOVP

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0110 |    |    |    | 111d |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

```
MOVP $acD
```

Description:

Moves multiply product from the \$prod register to the accumulator register \$acD.

Operation:

```
$acD = $prod
FLAGS($acD)
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | X |

5.6.78    **MOVPZ**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1111 |    |    |    | 111d |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

**Format:**

```
MOVPZ $acD
```

**Description:**

Moves multiply product from the \$prod register to the accumulator \$acD and sets \$acD.l to 0.

**Operation:**

```
$acD.hm = $prod.hm
$acD.l = 0
FLAGS($acD)
$pc++
```

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | X |

5.6.79 MOVR

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0110 |    |    |    | 0ssd |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

MOVR \$acD, \$(0x18+S)

Description:

Moves register \$(0x18+S) (sign-extended) to middle accumulator \$acD.hm. Sets \$acD.l to 0.

Operation:

\$acD.hm = \$(0x18+S)  
\$acD.l = 0  
FLAGS(\$acD)  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

5.6.80 MRR

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0001 |    |    |    | 11dd |    |   |   | ddds |   |   |   | ssss |   |   |   |

Format:

MRR \$D, \$S

Description:

Move value from register \$S to register \$D.

When moving to \$ac0.m or \$ac1.m, optionally perform sign extension depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

When moving from \$ac0.m or \$ac1.m, optionally apply saturation depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

Operation:

\$D = \$S  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.81 MSUB

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1111 |    |    |    | 011s |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

```
MSUB $axS.l, $axS.h
```

Description:

Multiply low part `$axS.l` of secondary accumulator `$axS` by high part `$axS.h` of secondary accumulator `$axS` (treat them both as signed) and subtract result from product register.

Operation:

```
$prod -= $axS.l * $axS.h
$pc++
```

See also:

`$sr.AM` bit affects multiply result.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.82 MSUBC

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1110 |    |    |    | 11st |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

```
MSUBC $acS.m, $axT.h
```

Description:

Multiply middle part of accumulator `$acS.m` by high part of secondary accumulator `$axT.h` (treat them both as signed) and subtract result from product register.

Operation:

```
$prod -= $acS.m * $axT.h
$pc++
```

See also:

`$sr.AM` bit affects multiply result.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |



5.6.83 MSUBX

|                                       |  |  |  |      |  |  |  |      |  |  |  |      |  |  |  |
|---------------------------------------|--|--|--|------|--|--|--|------|--|--|--|------|--|--|--|
| 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |  |  |  |      |  |  |  |      |  |  |  |      |  |  |  |
| 1110                                  |  |  |  | 01st |  |  |  | xxxx |  |  |  | xxxx |  |  |  |

Format:

```
MSUBX $(0x18+S*2), $(0x19+T*2)
```

Description:

Multiply one part of secondary accumulator `$ax0` (selected by `S`) by one part of secondary accumulator `$ax1` (selected by `T`) (treat them both as signed) and subtract result from product register.

Operation:

```
$prod -= $(0x18+S*2) * $(0x19+T*2)
$pc++
```

See also:

`$sr.AM` bit affects multiply result.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.84 MUL

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1001 |    |    |    | s000 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

```
MUL $axS.l, $axS.h
```

Description:

Multiply low part `$axS.l` of secondary accumulator `$axS` by high part `$axS.h` of secondary accumulator `$axS` (treat them both as signed).

Operation:

```
$prod = $axS.l * $axS.h
$pc++
```

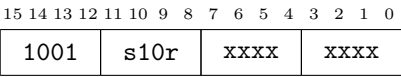
See also:

`$sr.AM` bit affects multiply result.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.85 MULAC



Format:

```
MULAC $axS.l, $axS.h, $acR
```

Description:

Add product register to accumulator register `$acR`. Multiply low part `$axS.l` of secondary accumulator `$axS` by high part `$axS.h` of secondary accumulator `$axS` (treat them both as signed).

Operation:

```
$acR += $prod
$prod = $axS.l * $axS.h
$pc++
```

See also:

`$sr.AM` bit affects multiply result.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | X |

5.6.86 MULAXH

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1000 |    |    |    | 0011 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

MULAXH

Description:

Multiplies `$ax0.h` by itself.

Operation:

```
$prod = $ax0.h * $ax0.h
$pc++
```

See also:

`$sr.AM` bit affects multiply result.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.87 MULC

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 110s |    |    |    | t000 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

**Format:**

MULC \$acS.m, \$axT.h

**Description:**

Multiply mid part of accumulator register \$acS.m by high part \$axS.h of secondary accumulator \$axS (treat them both as signed).

**Operation:**

\$prod = \$acS.m \* \$axS.h  
\$pc++

**See also:**

\$sr.AM bit affects multiply result.

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.88 MULCAC

|      |    |    |    |    |      |   |   |   |   |      |   |   |   |   |      |  |  |  |  |
|------|----|----|----|----|------|---|---|---|---|------|---|---|---|---|------|--|--|--|--|
| 15   | 14 | 13 | 12 | 11 | 10   | 9 | 8 | 7 | 6 | 5    | 4 | 3 | 2 | 1 | 0    |  |  |  |  |
| 110s |    |    |    |    | t10r |   |   |   |   | xxxx |   |   |   |   | xxxx |  |  |  |  |

Format:

MULCAC \$acS.m, \$axT.h, \$acR

Description:

Multiply mid part of accumulator register \$acS.m by high part \$axS.h of secondary accumulator \$axS (treat them both as signed). Add product register before multiplication to accumulator \$acR.

Operation:

```
temp = $prod
$prod = $acS.m * $axS.h
$acR += temp
$pc++
```

See also:

\$sr.AM bit affects multiply result.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | X |

5.6.89 MULCMV

|      |    |    |    |    |      |   |   |   |   |      |   |   |   |   |      |  |  |  |  |
|------|----|----|----|----|------|---|---|---|---|------|---|---|---|---|------|--|--|--|--|
| 15   | 14 | 13 | 12 | 11 | 10   | 9 | 8 | 7 | 6 | 5    | 4 | 3 | 2 | 1 | 0    |  |  |  |  |
| 110s |    |    |    |    | t11r |   |   |   |   | xxxx |   |   |   |   | xxxx |  |  |  |  |

**Format:**

MULCMV \$acS.m, \$axT.h, \$acR

**Description:**

Multiply mid part of accumulator register \$acS.m by high part \$axT.h of secondary accumulator \$axT (treat them both as signed). Move product register before multiplication to accumulator \$acR.

**Operation:**

```
temp = $prod
$prod = $acS.m * $axT.h
$acR = temp
$pc++
```

**See also:**

\$sr.AM bit affects multiply result.

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | X |

5.6.90 MULCMVZ

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 110s |    |    |    | t01r |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

MULCMVZ \$acS.m, \$axT.h, \$acR

Description:

Multiply mid part of accumulator register \$acS.m by high part \$aTS.h of secondary accumulator \$axT (treat them both as signed). Move product register before multiplication to accumulator \$acR. Set low part of accumulator \$acR.l to zero.

Operation:

```
temp = $prod
$prod = $acS.m * $axT.h
$acR.hm = temp.hm
$acR.l = 0
$pc++
```

See also:

\$sr.AM bit affects multiply result.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | X |



5.6.91 MULMV

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1001 |    |    |    | s11r |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

```
MULMV $axS.l, $axS.h, $acR
```

Description:

Move product register to accumulator register **\$acR**. Multiply low part **\$axS.l** of secondary accumulator Register**\$axS** by high part **\$axS.h** of secondary accumulator **\$axS** (treat them both as signed).

Operation:

```
$acR = $prod
$prod = $axS.l * $axS.h
$pc++
```

See also:

**\$sr.AM** bit affects multiply result.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | X |

5.6.92 MULMVZ

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1001 |    |    |    | s01r |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

```
MULMVZ $axS.l, $axS.h, $acR
```

Description:

Move product register to accumulator register `$acR` and clear low part of accumulator register `$acR.l`.  
Multiply low part `$axS.l` of secondary accumulator `$axS` by high part `$axS.h` of secondary accumulator `$axS` (treat them both as signed).

Operation:

```
$acR.hm = $prod.hm  
$acR.l = 0  
$prod = $axS.l * $axS.h  
$pc++
```

See also:

`$sr.AM` bit affects multiply result.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | X |

**5.6.93 MULX**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 101s |    |    |    | t000 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

**Format:**

MULX \$ax0.S, \$ax1.T

**Description:**

Multiply one part \$ax0 by one part \$ax1 (treat them both as signed). Part is selected by S and T bits. Zero selects low part, one selects high part.

**Operation:**

```
$prod = (S == 0) ? $ax0.l : ax0.h * (T == 0) ? $ax1.l : $ax1.h
$pc++
```

**See also:**

\$sr.AM bit affects multiply result.

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.94 MULXAC

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 101s |    |    |    | t10r |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

MULXAC \$ax0.S, \$ax1.T, \$acR

Description:

Add product register to accumulator register \$acR. Multiply one part \$ax0 by one part \$ax1 (treat them both as signed). Part is selected by S and T bits. Zero selects low part, one selects high part.

Operation:

```
$acR += $prod
$prod = (S == 0) ? $ax0.l : ax0.h * (T == 0) ? $ax1.l : $ax1.h
$pc++
```

See also:

\$sr.AM bit affects multiply result.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | X |

**5.6.95 MULXMV**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 101s |    |    |    | t11r |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

**Format:**

MULXMV \$ax0.S, \$ax1.T, \$acR

**Description:**

Move product register to accumulator register \$acR. Multiply one part \$ax0 by one part \$ax1 (treat them both as signed). Part is selected by S and T bits. Zero selects low part, one selects high part.

**Operation:**

```
$acR = $prod
$prod = (S == 0) ? $ax0.l : ax0.h * (T == 0) ? $ax1.l : $ax1.h
$pc++
```

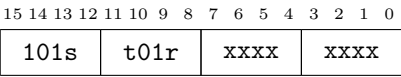
**See also:**

\$sr.AM bit affects multiply result.

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | X |

5.6.96 MULXMVZ



Format:

```
MULXMVZ $ax0.S, $ax1.T, $acR
```

Description:

Move product register to accumulator register `$acR` and clear low part of accumulator register `$acR.l`.  
Multiply one part `$ax0` by one part `$ax1` (treat them both as signed). Part is selected by `S` and `T` bits. Zero selects low part, one selects high part.

Operation:

```
$acR.hm = $prod.hm
$acR.l = 0
$prod = (S == 0) ? $ax0.l : ax0.h * (T == 0) ? $ax1.l : $ax1.h
$pc++
```

See also:

`$sr.AM` bit affects multiply result.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | X |

5.6.97 NEG

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0111 |    |    |    | 110d |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

NEG \$acD

Description:

Negates accumulator \$acD.

Operation:

\$acD = 0 - \$acD  
FLAGS(\$acD)  
\$pc++

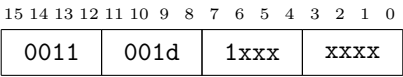
Note:

The carry flag is set only if \$acD was zero. The overflow flag is set only if \$acD was 0x8000000000 (the minimum value), as -INT\_MIN is INT\_MIN in two’s complement. In both of these cases, the value of \$acD after the operation is the same as it was before.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

5.6.98 NOT



Format:

```
NOT $acD.m
```

Description:

Invert all bits in the middle part of accumulator \$acD.m (i.e. XOR with 0xffff).

Operation:

```
$acD.m = ~acD.m
FLAGS($acD)
$pc++
```

Note:

The main opcode is 9 bits and the extension opcode is 7 bits. The extension opcode is treated as if the 8th bit was 0 (i.e. it is 0xxxxxxx).

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |



5.6.99 NOP

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 0000 |    |   |   | 0000 |   |   |   | 0000 |   |   |   |

Format:

NOP

Description:

No operation.

Operation:

`$pc++`

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.100 NX

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1000 |    |    |    | -000 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

NX

Description:

No operation, but can be extended with extended opcode.

Operation:

\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

**5.6.101 ORC**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0011 |    |    |    | 111d |    |   |   | 0xxx |   |   |   | xxxx |   |   |   |

**Format:**

ORC  $\$acD.m$ ,  $\$ac(1-D).m$

**Description:**

Logic OR middle part of accumulator  $\$acD.m$  with middle part of accumulator  $\$ac(1-D).m$ .

**Operation:**

$\$acD.m \mid= \$ac(1-D).m$   
 FLAGS( $\$acD$ )  
 $\$pc++$

**Note:**

The main opcode is 9 bits and the extension opcode is 7 bits. The extension opcode is treated as if the 8th bit was 0 (i.e. it is 0xxxxxxx).

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.102 ORI

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 001d |    |   |   | 0110 |   |   |   | 0000 |   |   |   |
| iiii |    |    |    | iiii |    |   |   | iiii |   |   |   | iiii |   |   |   |

Format:

```
ORI $acD.m, #I
```

Description:

Logical OR of accumulator mid part \$acD.m with immediate value I.

Operation:

```
$acD.m |= #I
FLAGS($acD)
$pc += 2
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | X |

**5.6.103 ORR**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0011 |    |    |    | 10sd |    |   |   | 0xxx |   |   |   | xxxx |   |   |   |

**Format:**

ORR \$acD.m, \$axS.h

**Description:**

Logical OR middle part of accumulator \$acD.m with high part of secondary accumulator \$axS.h.

**Operation:**

\$acD.m |= \$axS.h  
 FLAGS(\$acD)  
 \$pc++

**Note:**

The main opcode is 9 bits and the extension opcode is 7 bits. The extension opcode is treated as if the 8th bit was 0 (i.e. it is 0xxxxxxx).

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | X |

5.6.104 RET

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 0010 |    |   |   | 1101 |   |   |   | 1111 |   |   |   |

Format:

RET

Description:

Return from subroutine. Pops stored PC from call stack `$st0` and sets `$pc` to this location.

Operation:

`$pc = $st0`  
`POP_STACK($st0)`

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.105 RETcc

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 0010 |    |   |   | 1101 |   |   |   | cccc |   |   |   |

Format:

RETcc

Description:

Return from subroutine if condition cc has been met. Pops stored PC from call stack \$st0 and sets \$pc to this location.

Operation:

```
IF (cc)
    POP_STACK($st0)
ELSE
    $pc++
ENDIF
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

**5.6.106 RTI**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 0010 |    |   |   | 1111 |   |   |   | 1111 |   |   |   |

**Format:**

RTI

**Description:**

Return from exception. Pops stored status register **\$sr** from data stack **\$st1** and program counter PC from call stack **\$st0** and sets **\$pc** to this location.

**Operation:**

```

$sr = $st1
POP_STACK($st1)
$pc = $st0
POP_STACK($st0)

```

**Flags:**

| OS | LZ | TB | S32 | S | AZ | O | C |
|----|----|----|-----|---|----|---|---|
| -  | -  | -  | -   | - | -  | - | - |



5.6.107 RTIcc

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 0010 |    |   |   | 1111 |   |   |   | cccc |   |   |   |

Format:

RTIcc

Description:

Return from exception if condition cc has been met. Pops stored status register \$sr from data stack \$st1 and program counter PC from call stack \$st0 and sets \$pc to this location.

Operation:

```
IF (cc)
    $sr = $st1
    POP_STACK($st1)
    $pc = $st0
    POP_STACK($st0)
ELSE
    $pc++
ENDIF
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.108 SBCLR

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0001 |    |    |    | 0010 |    |   |   | 0000 |   |   |   | 0iii |   |   |   |

Format:

SBCLR #I

Description:

Clear bit of status register **\$sr**. Bit number is calculated by adding 6 to immediate value I; thus, bits 6 through 13 (LZ through AM) can be cleared with this instruction.

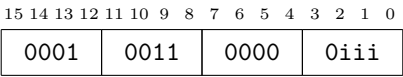
Operation:

$\$sr \&= \sim(1 \ll (I + 6))$   
 $\$pc++$

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.109 SBSET



Format:

SBSET #I

Description:

Set bit of status register `$sr`. Bit number is calculated by adding 6 to immediate value I; thus, bits 6 through 13 (LZ through AM) can be set with this instruction.

Operation:

`$sr |= 1 << (I + 6)`  
`$pc++`

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.110 SET15

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1000 |    |    |    | 1101 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

**Format:**

SET15

**Description:**

Sets `$sr.SU` (bit 15) to 1, causing multiplication to treat its operands as unsigned.

**Operation:**

`$sr |= 0x8000`  
`$pc++`

**See also:**

[CLR15](#)

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.111 SET16

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1000 |    |    |    | 1110 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

**Format:**

SET16

**Description:**

Sets `$sr.SXM` (bit 14) to 0, resulting in 16-bit sign extension.

**Operation:**

`$sr &= ~0x4000`  
`$pc++`

See also:

- [SET40](#)
- [16-bit and 40-bit modes](#)

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.112 SET40

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1000 |    |    |    | 1111 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

**Format:**

SET40

**Description:**

Sets `$sr.SXM` (bit 14) to 1, resulting in 40-bit sign extension.

**Operation:**

`$sr |= 0x4000`  
`$pc++`

**See also:**

[SET16](#)

[16-bit and 40-bit modes](#)

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.113 SI

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0001 |    |    |    | 0110 |    |   |   | mmmm |   |   |   | mmmm |   |   |   |
| iiii |    |    |    | iiii |    |   |   | iiii |   |   |   | iiii |   |   |   |

Format:

SI @M, #I

Description:

Store 16-bit immediate value I to a memory location pointed by address 0xFF00 | M.

Operation:

MEM[0xFF00 | M] = I  
\$pc += 2

Note:

Unlike [SRS](#), [SRSH](#), and [LRS](#), [SI](#) does not use \$cr to decide the base address and instead always uses 0xFF00.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.114 SR

|                                       |  |  |  |      |  |  |  |      |  |  |  |      |  |  |  |
|---------------------------------------|--|--|--|------|--|--|--|------|--|--|--|------|--|--|--|
| 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |  |  |  |      |  |  |  |      |  |  |  |      |  |  |  |
| 0000                                  |  |  |  | 0000 |  |  |  | 111s |  |  |  | ssss |  |  |  |
| mmm                                   |  |  |  | mmm  |  |  |  | mmm  |  |  |  | mmm  |  |  |  |

Format:

SR @M, \$S

Description:

Store value from register \$S to a memory pointed by address M.

When storing from \$ac0.m or \$ac1.m, optionally apply saturation depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

Operation:

MEM[M] = \$S  
\$pc += 2

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |



5.6.115 SRR

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0001 |    |    |    | 1010 |    |   |   | Odds |   |   |   | ssss |   |   |   |

Format:

SRR @*\$arD*, *\$S*

Description:

Store value from source register *\$S* to a memory location pointed by addressing register *\$arD*.

When storing from *\$ac0.m* or *\$ac1.m*, optionally apply saturation depending on the value of *\$sr.SXM* (see [16-bit and 40-bit modes](#)).

Operation:

MEM[*\$arD*] = *\$S*  
*\$pc*++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.116 SRRD

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0001 |    |    |    | 1010 |    |   |   | 1dds |   |   |   | ssss |   |   |   |

Format:

SRRD @*\$arD*, *\$S*

Description:

Store value from source register *\$S* to a memory location pointed by addressing register *\$arD*. Decrement register *\$arD*.

When storing from *\$ac0.m* or *\$ac1.m*, optionally apply saturation depending on the value of *\$sr.SXM* (see [16-bit and 40-bit modes](#)).

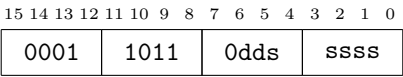
Operation:

MEM[*\$arD*] = *\$S*  
*\$arD*--  
*\$pc*++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.117 SRRI



Format:

```
SRRI @arD, $S
```

Description:

Store value from source register `$S` to a memory location pointed by addressing register `$arD`. Increment register `$arD`.

When storing from `$ac0.m` or `$ac1.m`, optionally apply saturation depending on the value of `$sr.SXM` (see [16-bit and 40-bit modes](#)).

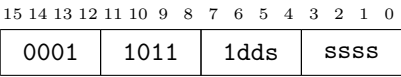
Operation:

```
MEM[$arD] = $S
$arD++
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.118 SRRN



Format:

```
SRRN @arD, $S
```

Description:

Store value from source register `$S` to a memory location pointed by addressing register `$arD`. Add indexing register `$ixD` to register `$arD`.

When storing from `$ac0.m` or `$ac1.m`, optionally apply saturation depending on the value of `$sr.SXM` (see [16-bit and 40-bit modes](#)).

Operation:

```
MEM[$arD] = $S
$arD += $ixD
$pc++
```

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.119 SRS

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0010 |    |    |    | 11ss |    |   |   | mmmm |   |   |   | mmmm |   |   |   |

Format:

SRS @M, \$(0x1C+S)

Description:

Store value from register \$(0x1C+S) to a memory pointed by address (\$cr << 8) | M.

When storing from \$ac0.m or \$ac1.m, optionally apply saturation depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

Operation:

MEM[( \$cr << 8) | M] = \$(0x1C+S)  
\$pc++

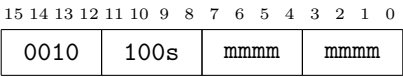
Note:

Unlike [LRS](#), [SRS](#) and [SRSH](#) only work on \$acS. The pattern 101s is unused and does not perform any write.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.120 SRS<sub>H</sub>



Format:

SRS<sub>H</sub> @M, \$acS.h

Description:

Store value from register \$acS.h to a memory pointed by address (\$cr << 8) | M.

Operation:

MEM[( $\$cr \ll 8$ ) | M] = \$acS.h  
\$pc++

Note:

Unlike [LRS](#), [SRS](#) and [SRS<sub>H</sub>](#) only work on \$acS. The pattern 101s is unused and does not perform any write.

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |

5.6.121 SUB

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0101 |    |    |    | 110d |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

SUB \$acD , \$ac(1-D)

Description:

Subtracts accumulator \$ac(1-D) from accumulator register \$acD.

Operation:

\$acD -= \$ac(1-D)  
FLAGS(\$acD)  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

5.6.122 SUBARN

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0000 |    |    |    | 0000 |    |   |   | 0000 |   |   |   | 11dd |   |   |   |

Format:

SUBARN \$arD

Description:

Subtracts indexing register \$ixD from addressing register \$arD.

Operation:

\$arD -= \$ixD  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | -  | -   | - | -  | - | - |



5.6.123 SUBAX

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0101 |    |    |    | 10sd |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

SUBAX \$acD, \$axS

Description:

Subtracts secondary accumulator \$axS from accumulator register \$acD.

Operation:

\$acD -= \$axS  
FLAGS(\$acD)  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

5.6.124 SUBP

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0101 |    |    |    | 111d |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

**Format:**

SUBP \$acD

**Description:**

Subtracts product register from accumulator register.

**Operation:**

\$acD -= \$prod  
FLAGS(\$acD)  
\$pc++

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

5.6.125 SUBR

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0101 |    |    |    | 0ssd |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

SUBR \$acD, \$(0x18+S)

Description:

Subtracts register \$(0x18+S) from accumulator \$acD register.

Operation:

\$acD -= (\$ (0x18+S) << 16)  
FLAGS(\$acD)  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| X  | -  | X  | X   | X | X  | X | X |

5.6.126 TST

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1011 |    |    |    | r001 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

TST \$acR

Description:

Test accumulator \$acR.

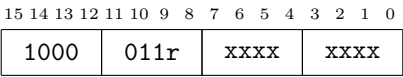
Operation:

FLAGS(\$acR)  
\$pc++

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

5.6.127    **TSTAXH**



**Format:**

```
TSTAXH  $axR.h
```

**Description:**

Test high part of secondary accumulator `$axR.h`.

**Operation:**

```
FLAGS($axR.h)
$pc++
```

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | 0   | X | X  | 0 | 0 |

5.6.128 TSTPROD

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 1000 |    |    |    | 0101 |    |   |   | xxxx |   |   |   | xxxx |   |   |   |

Format:

TSTPROD

Description:

Test the product register `$prod`.

Operation:

FLAGS(`$prod`)  
`$pc++`

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | 0   | X | X  | 0 | X |

5.6.129 XORC

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0011 |    |    |    | 000d |    |   |   | 1xxx |   |   |   | xxxx |   |   |   |

Format:

XORC \$acD.m, \$ac(1-D).m

Description:

Logical XOR (exclusive OR) middle part of accumulator \$acD.m with middle part of accumulator \$ac(1-D).m.

Operation:

\$acD.m ^= \$ac(1-D).m  
FLAGS(\$acD)  
\$pc++

Note:

The main opcode is 9 bits and the extension opcode is 7 bits. The extension opcode is treated as if the 8th bit was 0 (i.e. it is 0xxxxxxx).

Flags:

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

5.6.130    **XORI**

|                                       |  |  |  |      |  |  |  |      |  |  |  |      |  |  |  |
|---------------------------------------|--|--|--|------|--|--|--|------|--|--|--|------|--|--|--|
| 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |  |  |  |      |  |  |  |      |  |  |  |      |  |  |  |
| 0000                                  |  |  |  | 001d |  |  |  | 0010 |  |  |  | 0000 |  |  |  |
| iiii                                  |  |  |  | iiii |  |  |  | iiii |  |  |  | iiii |  |  |  |

**Format:**

```
XORI $acD.m, #I
```

**Description:**

Logical XOR (exclusive OR) of accumulator mid part `$acD.m` with immediate value `I`.

**Operation:**

```
$acD.m ^= #I
FLAGS($acD)
$pc += 2
```

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |



5.6.131    **XORR**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| 0011 |    |    |    | 00sd |    |   |   | 0xxx |   |   |   | xxxx |   |   |   |

**Format:**

```
XORR $acD.m, $axS.h
```

**Description:**

Logical XOR (exclusive OR) middle part of accumulator `$acD.m` with high part of secondary accumulator `$axS.h`.

**Operation:**

```
$acD.m ^= $axS.h
FLAGS($acD)
$pc++
```

**Note:**

The main opcode is 9 bits and the extension opcode is 7 bits. The extension opcode is treated as if the 8th bit was 0 (i.e. it is 0xxxxxxx).

**Flags:**

|    |    |    |     |   |    |   |   |
|----|----|----|-----|---|----|---|---|
| OS | LZ | TB | S32 | S | AZ | O | C |
| -  | -  | X  | X   | X | X  | 0 | 0 |

## 5.7 Extended opcodes

Extended opcodes do not exist on their own. These opcodes can only be attached to opcodes that allow extending. Specifically, opcodes where the first nybble is 0, 1, or 2 cannot be extended. Opcodes where the first nybble is 4 or higher can be extended, using the 8 lower bits. Opcodes where the first nybble is 3 can also be extended, but the main opcode is 9 bits and the extension opcode is 7 bits. For these instructions, the extension opcode is treated as if the first bit were 0 (i.e. 0xxxxxxx). (NX has no behavior of its own, so it can be used to get an extended opcode's behavior on its own.)

Extended opcodes do not modify the program counter (\$pc register).

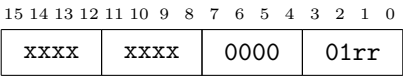
Extended opcodes are run *in parallel* with the main opcode; they see the same register state as the input. (For instance, `MOVR' MV $ac1, $ax0.l : $ax0.l, $ac1.m` (encoded as 0x6113) *swaps* the values of \$ac1.m and \$ax0.l (and also extends the new value of \$ac1.m into \$ac1.l and \$ac1.h).)

Since they are executed in parallel, the main and extension opcodes could theoretically write to the same registers. All opcodes that support extension only modify a main accumulator \$acD, as well as \$prod, \$sr, and/or \$pc, while the extension opcodes themselves generally only modify an additional accumulator \$axD and addressing registers \$arS. The exception is 'L and 'LN, which has the option of writing to \$acD. Thus, `INC'L $ac0 : $ac0.l, @ $ar0` (encoded as 0x7660) increments \$ac0 (and thus \$ac0.l), but also sets \$ac0.l to the value in data memory at address \$ar0 and increments \$ar0.

When the main and extension opcodes write to the same register, the register is set to the two values bitwise-or'd together. For the above example, \$ar0.l would be set to ( $\$ar0.l + 1$ ) | MEM[\$ar0]. **Note that no official uCode writes to the same register twice like this.**

## 5.8 Alphabetical list of extended opcodes

5.8.1 'DR



Format:

'DR \$arR

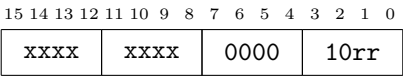
Description:

Decrement addressing register \$arR.

Operation:

\$arR--

5.8.2 'IR



Format:

'IR \$arR

Description:

Increment addressing register \$arR.

Operation:

\$arR++

### 5.8.3 'L

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 01dd |   |   |   | d0ss |   |   |   |

#### Format:

'L \$(0x18+D), @\$arS

#### Description:

Load register \$(0x18+D) with value from memory pointed by register \$arS. Post increment register \$arS.

When loading to \$ac0.m or \$ac1.m, optionally perform sign extension depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

#### Operation:

\$(0x18+D) = MEM[\$arS]  
\$arS++

**5.8.4 'LN**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 01dd |   |   |   | d1ss |   |   |   |

**Format:**

'LN \$(0x18+D), @\$arS

**Description:**

Load register \$(0x18+D) with value from memory pointed by register \$arS. Add indexing register \$ixS to register \$arS.

When loading to \$ac0.m or \$ac1.m, optionally perform sign extension depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

**Operation:**

```
$(0x18+D) = MEM[$arS]
$arS += $ixS
```

## 5.8.5 'LD

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 11dr |   |   |   | 00ss |   |   |   |

**Format:**

'LD \$ax0.D, \$ax1.R, @\$arS

**Description:**

Load register \$ax0.D (either \$ax0.l or \$ax0.h, as \$(0x18+D\*2)) with value from memory pointed by register \$arS. Load register \$ax1.R (either \$ax1.l or \$ax1.h, as \$(0x19+R\*2)) with value from memory pointed by register \$ar3. Increment both \$arS and \$ar3.

**Operation:**

```
$ax0.D = MEM[$arS]
$ax1.R = MEM[$ar3]
$arS++
$ar3++
```

**Note:**

S cannot be 3, as that instead encodes 'LDAX. Thus, \$arS is guaranteed to be distinct from \$ar3.



## 5.8.6 'LDM

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 11dr |   |   |   | 10ss |   |   |   |

**Format:**

```
'LDM $ax0.D, $ax1.R, @$arS
```

**Description:**

Load register `$ax0.D` (either `$ax0.l` or `$ax0.h`, as `$(0x18+D*2)`) with value from memory pointed by register `$arS`. Load register `$ax1.R` (either `$ax1.l` or `$ax1.h`, as `$(0x19+R*2)`) with value from memory pointed by register `$ar3`. Add corresponding indexing register `$ix3` to addressing register `$ar3` and increment `$arS`.

**Operation:**

```
$ax0.D = MEM[$arS]
$ax1.R = MEM[$ar3]
$arS++
$ar3 += $ix3
```

**Note:**

S cannot be 3, as that instead encodes 'LDAXM. Thus, `$arS` is guaranteed to be distinct from `$ar3`.

## 5.8.7 'LDNM

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 11dr |   |   |   | 11ss |   |   |   |

**Format:**

'LDNM \$ax0.D, \$ax1.R, @\$arS

**Description:**

Load register \$ax0.D (either \$ax0.l or \$ax0.h, as \$(0x18+D\*2)) with value from memory pointed by register \$arS. Load register \$ax1.R (either \$ax1.l or \$ax1.h, as \$(0x19+R\*2)) with value from memory pointed by register \$ar3. Add corresponding indexing register \$ixS to addressing register \$arS and add corresponding indexing register \$ix3 to addressing register \$ar3.

**Operation:**

```
$ax0.D = MEM[$arS]
$ax1.R = MEM[$ar3]
$arS += $ixS
$ar3 += $ix3
```

**Note:**

S cannot be 3, as that instead encodes 'LDAXNM. Thus, \$arS is guaranteed to be distinct from \$ar3.

## 5.8.8 'LDN

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 11dr |   |   |   | 01ss |   |   |   |

**Format:**

'LDN \$ax0.D, \$ax1.R, @ \$arS

**Description:**

Load register \$ax0.D (either \$ax0.l or \$ax0.h, as \$(0x18+D\*2)) with value from memory pointed by register \$arS. Load register \$ax1.R (either \$ax1.l or \$ax1.h, as \$(0x19+R\*2)) with value from memory pointed by register \$ar3. Add corresponding indexing register \$ixS to addressing register \$arS and increment \$ar3.

**Operation:**

```
$ax0.D = MEM[$arS]
$ax1.R = MEM[$ar3]
$arS += $ixS
$ar3++
```

**Note:**

S cannot be 3, as that instead encodes 'LDAXN. Thus, \$arS is guaranteed to be distinct from \$ar3.

**5.8.9 'LDAX**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 11sr |   |   |   | 0011 |   |   |   |

**Format:**

'LDAX \$axR, @\$arS

**Description:**

Load register \$axR.h with value from memory pointed by register \$arS. Load register \$axR.l with value from memory pointed by register \$ar3. Increment both \$arS and \$ar3.

**Operation:**

\$axR.h = MEM[\$arS]  
 \$axR.l = MEM[\$ar3]  
 \$arS++  
 \$ar3++

**Note:**

S can be either 0 or 1, corresponding to \$ar0 or \$ar1. Thus, \$arS is guaranteed to be distinct from \$ar3. \$ar2 cannot be used with this instruction.

**5.8.10 'LDAXM**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 11sr |   |   |   | 0011 |   |   |   |

**Format:**

```
'LDAXM $axR, @$arS
```

**Description:**

Load register `$axR.h` with value from memory pointed by register `$arS`. Load register `$axR.l` with value from memory pointed by register `$ar3`. Add corresponding indexing register `$ix3` to addressing register `$ar3` and increment `$arS`.

**Operation:**

```
$axR.h = MEM[$arS]
$axR.l = MEM[$ar3]
$arS++
$ar3 += $ix3
```

**Note:**

S can be either 0 or 1, corresponding to `$ar0` or `$ar1`. Thus, `$arS` is guaranteed to be distinct from `$ar3`. `$ar2` cannot be used with this instruction.

**5.8.11 'LDAXNM**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 11sr |   |   |   | 0011 |   |   |   |

**Format:**

```
'LDAXNM $axR, @$arS
```

**Description:**

Load register `$axR.h` with value from memory pointed by register `$arS`. Load register `$axR.l` with value from memory pointed by register `$ar3`. Add corresponding indexing register `$ixS` to addressing register `$arS` and add corresponding indexing register `$ix3` to addressing register `$ar3`.

**Operation:**

```
$axR.h = MEM[$arS]
$axR.l = MEM[$ar3]
$arS += $ixS
$ar3 += $ix3
```

**Note:**

S can be either 0 or 1, corresponding to `$ar0` or `$ar1`. Thus, `$arS` is guaranteed to be distinct from `$ar3`. `$ar2` cannot be used with this instruction.

**5.8.12 'LDAXN**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 11sr |   |   |   | 0011 |   |   |   |

**Format:**

```
'LDAXN $axR, @$arS
```

**Description:**

Load register `$axR.h` with value from memory pointed by register `$arS`. Load register `$axR.l` with value from memory pointed by register `$ar3`. Add corresponding indexing register `$ixS` to addressing register `$arS` and increment `$ar3`.

**Operation:**

```
$axR.h = MEM[$arS]
$axR.l = MEM[$ar3]
$arS += $ixS
$ar3++
```

**Note:**

S can be either 0 or 1, corresponding to `$ar0` or `$ar1`. Thus, `$arS` is guaranteed to be distinct from `$ar3`. `$ar2` cannot be used with this instruction.

**5.8.13 'LS**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 10dd |   |   |   | 000s |   |   |   |

**Format:**

'LS \$(0x18+D), \$acS.m

**Description:**

Load register \$(0x18+D) with value from memory pointed by register \$ar0. Store value from register \$acS.m to memory location pointed by register \$ar3. Increment both \$ar0 and \$ar3.

When storing from \$ac0.m or \$ac1.m, optionally apply saturation depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

**Operation:**

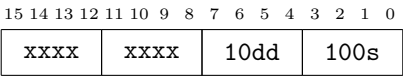
```
$(0x18+D) = MEM[$ar0]
MEM[$ar3] = $acS.m
$ar0++
$ar3++
```

**Note:**

Differs from 'SL in that \$(0x18+D) is associated with \$ar0 instead of \$ar3 and \$acS.m is associated with \$ar3 instead of \$ar0. In both cases, \$(0x18+D) is loaded and \$acS.m is stored.



5.8.14 'LSM



Format:

```
'LSM $(0x18+D), $acS.m
```

Description:

Load register  $\$(0x18+D)$  with value from memory pointed by register  $\$ar0$ . Store value from register  $\$acS.m$  to memory location pointed by register  $\$ar3$ . Add corresponding indexing register  $\$ix3$  to addressing register  $\$ar3$  and increment  $\$ar0$ .

When storing from  $\$ac0.m$  or  $\$ac1.m$ , optionally apply saturation depending on the value of  $\$sr.SXM$  (see [16-bit and 40-bit modes](#)).

Operation:

```
$(0x18+D) = MEM[$ar0]
MEM[$ar3] = $acS.m
$ar0++
$ar3 += $ix3
```

Note:

Differs from 'SLM in that  $\$(0x18+D)$  is associated with  $\$ar0$  instead of  $\$ar3$  and  $\$acS.m$  is associated with  $\$ar3$  instead of  $\$ar0$ . In both cases,  $\$(0x18+D)$  is loaded and  $\$acS.m$  is stored.

**5.8.15 'LSNM**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 10dd |   |   |   | 110s |   |   |   |

**Format:**

'LSNM \$(0x18+D), \$acS.m

**Description:**

Load register \$(0x18+D) with value from memory pointed by register \$ar0. Store value from register \$acS.m to memory location pointed by register \$ar3. Add corresponding indexing register \$ix0 to addressing register \$ar0 and add corresponding indexing register \$ix3 to addressing register \$ar3.

When storing from \$ac0.m or \$ac1.m, optionally apply saturation depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

**Operation:**

```
$(0x18+D) = MEM[$ar0]
MEM[$ar3] = $acS.m
$ar0 += $ix0
$ar3 += $ix3
```

**Note:**

Differs from 'SLNM in that \$(0x18+D) is associated with \$ar0 instead of \$ar3 and \$acS.m is associated with \$ar3 instead of \$ar0. In both cases, \$(0x18+D) is loaded and \$acS.m is stored.

**5.8.16 'LSN**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 10dd |   |   |   | 010s |   |   |   |

**Format:**

'LSN \$(0x18+D), \$acS.m

**Description:**

Load register \$(0x18+D) with value from memory pointed by register \$ar0. Store value from register \$acS.m to memory location pointed by register \$ar3. Add corresponding indexing register \$ix0 to addressing register \$ar0 and increment \$ar3.

When storing from \$ac0.m or \$ac1.m, optionally apply saturation depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

**Operation:**

```
$(0x18+D) = MEM[$ar0]
MEM[$ar3] = $acS.m
$ar0 += $ix0
$ar3++
```

**Note:**

Differs from 'SLN in that \$(0x18+D) is associated with \$ar0 instead of \$ar3 and \$acS.m is associated with \$ar3 instead of \$ar0. In both cases, \$(0x18+D) is loaded and \$acS.m is stored.

**5.8.17 'MV**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 0001 |   |   |   | ddss |   |   |   |

**Format:**

'MV \$(0x18+D), \$(0x1c+S)

**Description:**

Move value of register \$(0x1c+S) to the register \$(0x18+D).

When moving from \$ac0.m or \$ac1.m, optionally apply saturation depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

**Operation:**

\$(0x18+D) = \$(0x1c+S)

**5.8.18 'NOP**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 0000 |   |   |   | 00xx |   |   |   |

**Format:**

'NOP

**Description:**

No operation.

**Note:**

Generally written by not including any extension operation, such as writing `INC $ac0` instead of writing `INC'NOP $ac0`.

**5.8.19 'NR**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 0000 |   |   |   | 11rr |   |   |   |

**Format:**

'NR \$arR

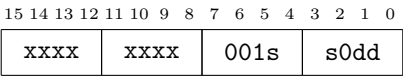
**Description:**

Add corresponding indexing register \$ixR to addressing register \$arR.

**Operation:**

\$arR += \$ixR

5.8.20 'S



Format:

```
'S @${arD}, ${0x1c+S}
```

Description:

Store value of register  $\$(0x1c+S)$  in the memory pointed by register  $\$arD$ . Post increment register  $\$arD$ .  
When storing from  $\$ac0.m$  or  $\$ac1.m$ , optionally apply saturation depending on the value of  $\$sr.SXM$  (see [16-bit and 40-bit modes](#)).

Operation:

```
MEM[${arD}] = ${0x1c+S}  
$arD++
```

**5.8.21 'SL**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 10dd |   |   |   | 001s |   |   |   |

**Format:**

'SL \$acS.m, \$(0x18+D)

**Description:**

Store value from register \$acS.m to memory location pointed by register \$ar0. Load register \$(0x18+D) with value from memory pointed by register \$ar3. Increment both \$ar0 and \$ar3.

When storing from \$ac0.m or \$ac1.m, optionally apply saturation depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

**Operation:**

```
$(0x18+D) = MEM[$ar3]
MEM[$ar0] = $acS.m
$ar0++
$ar3++
```

**Note:**

Differs from 'LS in that \$(0x18+D) is associated with \$ar3 instead of \$ar0 and \$acS.m is associated with \$ar0 instead of \$ar3. In both cases, \$(0x18+D) is loaded and \$acS.m is stored.



**5.8.22 'SLM**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 10dd |   |   |   | 101s |   |   |   |

**Format:**

'SLM \$acS.m, \$(0x18+D)

**Description:**

Store value from register \$acS.m to memory location pointed by register \$ar0. Load register \$(0x18+D) with value from memory pointed by register \$ar3. Add corresponding indexing register \$ix3 to addressing register \$ar3 and increment \$ar0.

When storing from \$ac0.m or \$ac1.m, optionally apply saturation depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

**Operation:**

```
$(0x18+D) = MEM[$ar3]
MEM[$ar0] = $acS.m
$ar0++
$ar3 += $ix3
```

**Note:**

Differs from 'LSM in that \$(0x18+D) is associated with \$ar3 instead of \$ar0 and \$acS.m is associated with \$ar0 instead of \$ar3. In both cases, \$(0x18+D) is loaded and \$acS.m is stored.

## 5.8.23 'SLNM

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 10dd |   |   |   | 111s |   |   |   |

**Format:**

'SLNM \$acS.m, \$(0x18+D)

**Description:**

Store value from register \$acS.m to memory location pointed by register \$ar0. Load register \$(0x18+D) with value from memory pointed by register \$ar3. Add corresponding indexing register \$ix0 to addressing register \$ar0 and add corresponding indexing register \$ix3 to addressing register \$ar3.

When storing from \$ac0.m or \$ac1.m, optionally apply saturation depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

**Operation:**

```
$(0x18+D) = MEM[$ar3]
MEM[$ar0] = $acS.m
$ar0 += $ix0
$ar3 += $ix3
```

**Note:**

Differs from 'LSNM in that \$(0x18+D) is associated with \$ar3 instead of \$ar0 and \$acS.m is associated with \$ar0 instead of \$ar3. In both cases, \$(0x18+D) is loaded and \$acS.m is stored.

**5.8.24 'SLN**

|      |    |    |    |      |    |   |   |      |   |   |   |      |   |   |   |
|------|----|----|----|------|----|---|---|------|---|---|---|------|---|---|---|
| 15   | 14 | 13 | 12 | 11   | 10 | 9 | 8 | 7    | 6 | 5 | 4 | 3    | 2 | 1 | 0 |
| xxxx |    |    |    | xxxx |    |   |   | 10dd |   |   |   | 011s |   |   |   |

**Format:**

'SLN \$acS.m, \$(0x18+D)

**Description:**

Store value from register \$acS.m to memory location pointed by register \$ar0. Load register \$(0x18+D) with value from memory pointed by register \$ar3. Add corresponding indexing register \$ix0 to addressing register \$ar0 and increment \$ar3.

When storing from \$ac0.m or \$ac1.m, optionally apply saturation depending on the value of \$sr.SXM (see [16-bit and 40-bit modes](#)).

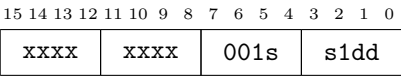
**Operation:**

```
$(0x18+D) = MEM[$ar3]
MEM[$ar0] = $acS.m
$ar0 += $ix0
$ar3++
```

**Note:**

Differs from 'LSN in that \$(0x18+D) is associated with \$ar3 instead of \$ar0 and \$acS.m is associated with \$ar0 instead of \$ar3. In both cases, \$(0x18+D) is loaded and \$acS.m is stored.

5.8.25 'SN



Format:

```
'SN @$arD, $(0x1c+S)
```

Description:

Store value of register `$(0x1c+S)` in the memory pointed by register `$arD`. Add indexing register `$ixD` to register `$arD`.

When storing from `$ac0.m` or `$ac1.m`, optionally apply saturation depending on the value of `$sr.SXM` (see [16-bit and 40-bit modes](#)).

Operation:

```
MEM[$arD] = $(0x1c+S)
$arD += $ixD
```

## 5.9 Instructions sorted by opcode

| Instruction | Opcode                                  | Page |
|-------------|---|------|
| NOP         | 0000 0000 0000 0000                     | 137  |
| DAR         | 0000 0000 0000 01dd                     | 74   |
| IAR         | 0000 0000 0000 10dd                     | 78   |
| SUBARN      | 0000 0000 0000 11dd                     | 160  |
| ADDARN      | 0000 0000 0001 ssdd                     | 41   |
| HALT        | 0000 0000 0010 0001                     | 77   |
| LOOP        | 0000 0000 010r rrrr                     | 90   |
| BLOOP       | 0000 0000 011r rrrr aaaa aaaa aaaa aaaa | 60   |
| LRI         | 0000 0000 100d dddd iiii iiii iiii iiii | 93   |
| Unknown     | 0000 0000 101x xxxx                     |      |
| LR          | 0000 0000 110d dddd mmmm mmmm mmmm mmmm | 92   |
| SR          | 0000 0000 111s ssss mmmm mmmm mmmm mmmm | 152  |
| IFcc        | 0000 0010 0111 cccc                     | 79   |
| Jcc         | 0000 0010 1001 cccc aaaa aaaa aaaa aaaa | 87   |
| CALLcc      | 0000 0010 1011 cccc aaaa aaaa aaaa aaaa | 63   |
| RETcc       | 0000 0010 1101 cccc                     | 143  |
| RTIcc       | 0000 0010 1111 cccc                     | 145  |
| ADDI        | 0000 001d 0000 0000 iiii iiii iiii iiii | 44   |
| XORI        | 0000 001d 0010 0000 iiii iiii iiii iiii | 168  |
| ANDI        | 0000 001d 0100 0000 iiii iiii iiii iiii | 52   |
| ORI         | 0000 001d 0110 0000 iiii iiii iiii iiii | 140  |
| CMPI        | 0000 001d 1000 0000 iiii iiii iiii iiii | 72   |
| ANDF        | 0000 001d 1010 0000 iiii iiii iiii iiii | 51   |
| ANDCF       | 0000 001d 1100 0000 iiii iiii iiii iiii | 50   |
| LSRN        | 0000 0010 1100 1010                     | 103  |
| ASRN        | 0000 0010 1100 1011                     | 56   |
| ILRR        | 0000 001d 0001 00ss                     | 80   |
| ILRRD       | 0000 001d 0001 01ss                     | 81   |
| ILRRI       | 0000 001d 0001 10ss                     | 82   |
| ILRRN       | 0000 001d 0001 11ss                     | 83   |
| ADDIS       | 0000 010d iiii iiii                     | 45   |
| CMPIS       | 0000 011d iiii iiii                     | 73   |
| LRIS        | 0000 1dd iiii iiii                      | 94   |
| LOOPI       | 0001 0000 iiii iiii                     | 91   |
| BLOOPI      | 0001 0001 iiii iiii aaaa aaaa aaaa aaaa | 61   |
| SBCLR       | 0001 0010 xxxx xiii                     | 146  |
| SBSET       | 0001 0011 xxxx xiii                     | 147  |
| LSL         | 0001 010r 00ii iiii                     | 100  |
| LSR         | 0001 010r 01ii iiii                     | 102  |
| ASL         | 0001 010r 10ii iiii                     | 54   |
| ASR         | 0001 010r 11ii iiii                     | 55   |
| SI          | 0001 0110 mmmm mmmm iiii iiii iiii iiii | 151  |
| JRcc        | 0001 0111 rrr0 cccc                     | 89   |
| CALLRcc     | 0001 0111 rrr1 cccc                     | 65   |

| Instruction | Opcode              | Page |
|-------------|---------------------|------|
| LRR         | 0001 1000 0ssd dddd | 95   |
| LRRD        | 0001 1000 1ssd dddd | 96   |
| LRRI        | 0001 1001 0ssd dddd | 97   |
| LRRN        | 0001 1001 1ssd dddd | 98   |
| SRR         | 0001 1010 0dds ssss | 153  |
| SRRD        | 0001 1010 1dds ssss | 154  |
| SRRI        | 0001 1011 0dds ssss | 155  |
| SRRN        | 0001 1011 1dds ssss | 156  |
| MRR         | 0001 11dd ddds ssss | 118  |
| LRS         | 0010 0ddd mmmm mmmm | 99   |
| SRSH        | 0010 100s mmmm mmmm | 158  |
| Unknown     | 0010 101x mmmm mmmm |      |
| SRS         | 0010 11ss mmmm mmmm | 157  |
| XORR        | 0011 00sd 0xxx xxxx | 169  |
| ANDR        | 0011 01sd 0xxx xxxx | 53   |
| ORR         | 0011 10sd 0xxx xxxx | 141  |
| ANDC        | 0011 110d 0xxx xxxx | 49   |
| ORC         | 0011 111d 0xxx xxxx | 139  |
| XORC        | 0011 000d 1xxx xxxx | 167  |
| NOT         | 0011 001d 1xxx xxxx | 136  |
| LSRNRX      | 0011 01sd 1xxx xxxx | 105  |
| ASRNRX      | 0011 10sd 1xxx xxxx | 58   |
| LSRNR       | 0011 110d 1xxx xxxx | 104  |
| ASRNR       | 0011 111d 1xxx xxxx | 57   |
| ADDR        | 0100 0ssd xxxx xxxx | 48   |
| ADDAX       | 0100 10sd xxxx xxxx | 42   |
| ADD         | 0100 110d xxxx xxxx | 40   |
| ADDP        | 0100 111d xxxx xxxx | 46   |
| SUBR        | 0101 0ssd xxxx xxxx | 163  |
| SUBAX       | 0101 10sd xxxx xxxx | 161  |
| SUB         | 0101 110d xxxx xxxx | 159  |
| SUBP        | 0101 111d xxxx xxxx | 162  |
| MOVR        | 0110 0ssd xxxx xxxx | 117  |
| MOVAX       | 0110 10sd xxxx xxxx | 113  |
| MOV         | 0110 110d xxxx xxxx | 112  |
| MOVP        | 0110 111d xxxx xxxx | 115  |
| ADDAXL      | 0111 00sd xxxx xxxx | 43   |
| INCM        | 0111 010d xxxx xxxx | 85   |
| INC         | 0111 011d xxxx xxxx | 84   |
| DECM        | 0111 100d xxxx xxxx | 76   |
| DEC         | 0111 101d xxxx xxxx | 75   |
| NEG         | 0111 110d xxxx xxxx | 135  |
| MOVNP       | 0111 111d xxxx xxxx | 114  |

| Instruction             | Opcode              | Page                |
|-------------------------|---------------------|---------------------|
| <a href="#">NX</a>      | 1000 x000 xxxx xxxx | <a href="#">138</a> |
| <a href="#">CLR</a>     | 1000 r001 xxxx xxxx | <a href="#">67</a>  |
| <a href="#">CMP</a>     | 1000 0010 xxxx xxxx | <a href="#">70</a>  |
| <a href="#">MULAXH</a>  | 1000 0011 xxxx xxxx | <a href="#">124</a> |
| <a href="#">CLRP</a>    | 1000 0100 xxxx xxxx | <a href="#">69</a>  |
| <a href="#">TSTPROD</a> | 1000 0101 xxxx xxxx | <a href="#">166</a> |
| <a href="#">TSTAXH</a>  | 1000 011r xxxx xxxx | <a href="#">165</a> |
| <a href="#">M2</a>      | 1000 1010 xxxx xxxx | <a href="#">108</a> |
| <a href="#">M0</a>      | 1000 1011 xxxx xxxx | <a href="#">107</a> |
| <a href="#">CLR15</a>   | 1000 1100 xxxx xxxx | <a href="#">66</a>  |
| <a href="#">SET15</a>   | 1000 1101 xxxx xxxx | <a href="#">148</a> |
| <a href="#">SET16</a>   | 1000 1110 xxxx xxxx | <a href="#">149</a> |
| <a href="#">SET40</a>   | 1000 1111 xxxx xxxx | <a href="#">150</a> |
| <a href="#">MUL</a>     | 1001 s000 xxxx xxxx | <a href="#">122</a> |
| <a href="#">ASR16</a>   | 1001 r001 xxxx xxxx | <a href="#">59</a>  |
| <a href="#">MULMVZ</a>  | 1001 s01r xxxx xxxx | <a href="#">130</a> |
| <a href="#">MULAC</a>   | 1001 s10r xxxx xxxx | <a href="#">123</a> |
| <a href="#">MULMV</a>   | 1001 s11r xxxx xxxx | <a href="#">129</a> |
| <a href="#">MULX</a>    | 101s t000 xxxx xxxx | <a href="#">131</a> |
| <a href="#">ABS</a>     | 1010 d001 xxxx xxxx | <a href="#">39</a>  |
| <a href="#">TST</a>     | 1011 r001 xxxx xxxx | <a href="#">164</a> |
| <a href="#">MULXMVZ</a> | 101s t01r xxxx xxxx | <a href="#">134</a> |
| <a href="#">MULXAC</a>  | 101s t10r xxxx xxxx | <a href="#">132</a> |
| <a href="#">MULXMV</a>  | 101s t11r xxxx xxxx | <a href="#">133</a> |
| <a href="#">MULC</a>    | 110s t000 xxxx xxxx | <a href="#">125</a> |
| <a href="#">CMPAXH</a>  | 110r s001 xxxx xxxx | <a href="#">71</a>  |
| <a href="#">MULCMVZ</a> | 110s t01r xxxx xxxx | <a href="#">128</a> |
| <a href="#">MULCAC</a>  | 110s t10r xxxx xxxx | <a href="#">126</a> |
| <a href="#">MULCMV</a>  | 110s t11r xxxx xxxx | <a href="#">127</a> |
| <a href="#">MADDX</a>   | 1110 00st xxxx xxxx | <a href="#">111</a> |
| <a href="#">MSUBX</a>   | 1110 01st xxxx xxxx | <a href="#">121</a> |
| <a href="#">MADDC</a>   | 1110 10st xxxx xxxx | <a href="#">110</a> |
| <a href="#">MSUBC</a>   | 1110 11st xxxx xxxx | <a href="#">120</a> |
| <a href="#">LSL16</a>   | 1111 000r xxxx xxxx | <a href="#">101</a> |
| <a href="#">MADD</a>    | 1111 001s xxxx xxxx | <a href="#">109</a> |
| <a href="#">LSR16</a>   | 1111 010r xxxx xxxx | <a href="#">106</a> |
| <a href="#">MSUB</a>    | 1111 011s xxxx xxxx | <a href="#">119</a> |
| <a href="#">ADDPAXZ</a> | 1111 10sd xxxx xxxx | <a href="#">47</a>  |
| <a href="#">CLRL</a>    | 1111 110r xxxx xxxx | <a href="#">68</a>  |
| <a href="#">MOVPZ</a>   | 1111 111d xxxx xxxx | <a href="#">116</a> |

## Extension Opcodes

| Instruction | Opcode              | Page                |
|-------------|---------------------|---------------------|
| 'NOP        | xxxx xxxx 0000 00xx | <a href="#">189</a> |
| 'DR         | xxxx xxxx 0000 01rr | <a href="#">172</a> |
| 'IR         | xxxx xxxx 0000 10rr | <a href="#">173</a> |
| 'NR         | xxxx xxxx 0000 11rr | <a href="#">190</a> |
| 'MV         | xxxx xxxx 0001 ddss | <a href="#">188</a> |
| 'S          | xxxx xxxx 001s s0dd | <a href="#">191</a> |
| 'SN         | xxxx xxxx 001s s1dd | <a href="#">196</a> |
| 'L          | xxxx xxxx 01dd d0ss | <a href="#">174</a> |
| 'LN         | xxxx xxxx 01dd d1ss | <a href="#">175</a> |
| 'LS         | xxxx xxxx 10dd 000s | <a href="#">184</a> |
| 'SL         | xxxx xxxx 10dd 001s | <a href="#">192</a> |
| 'LSN        | xxxx xxxx 10dd 010s | <a href="#">187</a> |
| 'SLN        | xxxx xxxx 10dd 011s | <a href="#">195</a> |
| 'LSM        | xxxx xxxx 10dd 100s | <a href="#">185</a> |
| 'SLM        | xxxx xxxx 10dd 101s | <a href="#">193</a> |
| 'LSNM       | xxxx xxxx 10dd 110s | <a href="#">186</a> |
| 'SLNM       | xxxx xxxx 10dd 111s | <a href="#">194</a> |
| 'LD         | xxxx xxxx 11dr 00ss | <a href="#">176</a> |
| 'LDAX       | xxxx xxxx 11sr 0011 | <a href="#">180</a> |
| 'LDN        | xxxx xxxx 11dr 01ss | <a href="#">179</a> |
| 'LDAXN      | xxxx xxxx 11sr 0111 | <a href="#">183</a> |
| 'LDM        | xxxx xxxx 11dr 10ss | <a href="#">177</a> |
| 'LDAXM      | xxxx xxxx 11sr 1011 | <a href="#">181</a> |
| 'LDNM       | xxxx xxxx 11dr 11ss | <a href="#">178</a> |
| 'LDAXNM     | xxxx xxxx 11dr 1111 | <a href="#">182</a> |