User's Manual

GameCube DSP (GDSP)

Reversed and documented by Duddie (duddie@walla.com)

Document date: 2005.05.12
Document version: 0.0.4

Table of Contents

I. Disclaimer

This documentation is no way enorsed by or affiliated with Nintedo, 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.

II. GNU Free Documentation License

GNU Free Documentation License
Version 1.2, November 2002

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. Secondarily, 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 http://www.gnu.org/copyleft/.

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.

III. Version history

Version	Date	Author	Change
0.0.1	2005.05.08	Duddie	Initial release
0.0.2	2005.05.09	Duddie	Added $prod and $config registers, table of opcodes, disclaimer
0.0.3	2005.05.09	Duddie	Fixed BLOOP and BLOOPI and added description of Loop Stack
0.0.4	2005.05.12	Duddie	Added preliminary DSP memory map and opcode syntax

IV. Overview

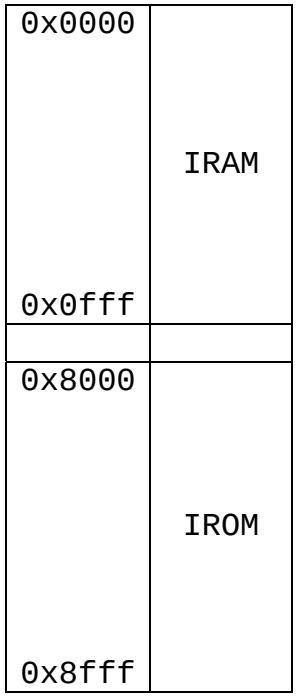
1. DSP Memory Map

DSP accesses memory in words, so all addresses refer to words. DSP word is 16 bit long.

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 first 8 words.

DSP IRAM is mapped through as first 8kB (4kW) of ARAM (Accelerator RAM) therefore CPU can directly DMA DSP code to DSP IRAM. This usually happens during boottime because DSP ROM is not enabled at cold reset and needs to be reenabled by small stub executed in IRAM.



V. Registers

1. Register names

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

$0	$r00	$ar0	Addressing register 0
$1	$r01	$ar1	
$2	$r02	$ar2	
$3	$r03	$ar3	
$4	$r04	$ix0	
$5	$r05	$ix1	
$6	$r06	$ix2	
$7	$r07	$ix3	
$8	$r08		
$9	$r09		
$10	$r0a		
$11	$r0b		
$12	$r0c	$st0	
$13	$r0d	$st1	
$14	$r0e	$st2	
$15	$r0f	$st3	
$16	$r10	$ac0.h	
$17	$r11	$ac1.h	
$18	$r12	$config	
$19	$r13	$sr	
$20	$r14	$prod.l	
$21	$r15	$prod.m1	
$22	$r16	$prod.h	
$23	$r17	$prod.m2	
$24	$r18	$ax0.l	
$25	$r19	$ax1.l	
$26	$r1a	$ax1.h	
$27	$r1b	$ax1.h	
$28	$r1c	$ac0.l	
$29	$r1d	$ac1.l	
$30	$r1e	$ac0.m	
$31	$r1f	$ac1.m	

2. Accumulators

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

Accumulators $acX:

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

```
$acX = $acX.h << 32 | $acX.m << 16 | $acX.l
```

Short accumulators $acsX:

24-bit accumulator $acsX ($acX.hm) consists of upper 24bit of accumulator $acX

```
$acsX = $acX.h << 16 | $acX.m
```

Additonal accumulators $axX:

```
$axX = $axX.h << 16 | $axX.l
```

3. Stacks

GDSP contains 4 stack registers:
- $st0 - call stack
- $st1 - data stack
- $st2 - loop address stack
- $st3 - loop counter

Stacks are implemented in hardware and have limited depth. Data stack is limited to 4 values and call stack is limited to 8 values. Loop stack is limited to 4 values. Upon underflow or overflow of any of the stack registers exception STOVF is raised.

Loop stack is used to control execution of repeated blocks of instructions. Whenever there is value on stack $st2 and current PC is equal value at $st2, then value at stack $st3 is decremented. If value is not zero then PC is modified with calue from call stack $st0. Otherwise values from callstack $st0 and both loop stacks $st2 and $st3 are poped and execution continues at next opcode.

4. Config register

It’s purpose is unknown at this time. It is written with 0x00ff and 0x0004 values.

5. Status register

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

Bit	Name	Comment
14	AM	Product multiply result by 2 (when AM = 0)
9	IE	Interrupt enable
8	0	Hardwired to 0 (?)
6	LZ	Logic zero
4	AS	
3	S	Sign
2	Z	Zero

6. Product register

Product register is an intermediate product of multiply or multiply and accumulation. It's result should never be used for calculation although the register can be read or writtent. It reflects state of internal multiply unit. Product is 40 bit with 1 bit of overflow.

```
$prod = ($prod.h << 32) + (($prod.m1 + $prod.m2) << 16) + $prod.l
```

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

Bit $sr.AM affects result of multiply unit. If bit $sr.AM is equal 0 then result of every multiply operation will be multiplied by 2 (two).

VI. Exceptions

1. Exception processing

Exception processing happens by setting program counter to different exception vectors. At the exception time, 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
```

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		
4	0x0008		
5	0x000a	ACCOV	Accelerator address overflow
6	0x000c		
7	0x000e		

VII. Hardware interface

1. Hardware registers

Hardware registers occupy address space at 0xffxx in DSP memory space. Each register is 16 bit.

Address	**Name**	**Description**
Mailboxes		
0xfffe	CMBH	CPU Mailbox H
0xffff	CMBL	CPU Mailbox L
0xfffc	DMBH	DSP Mailbox H
0xfffd	DMBL	DSP Mailbox L
DMA interface		
0xffce	DSMAH	Memory address H
0xffcf	DSMAL	Memory address L
0xffcd	DSPA	DSP memory address
0xffc9	DSCR	DMA Control
0xffcb	DSBL	Block size
Accelerator		
0xffd4	ACSAH	Accelerator start address H
0xffd5	ACSAL	Acceleratir 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
0xffdd	ACDAT	Accelerator data
Interrupts		
0xfffb	DIRQ	IRQ request

2. Interrupts

DSP can raise interrupts at CPU. Usually interrupts are used to signal that new DSP mbox has been filled with data.

0xFFFB	DIRQ	IRQ Request
---- ---- ---- ---I		

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

3. Mailboxes

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

0xFFFE	**CMBH**	**CPU Mailbox H**
Mddd dddd dddd dddd		

Bit	**Name**	**R/W**	**Action**
15	M	R	1 – Mailbox contains mail from CPU 0 – Mailbox empty
14-0	d	R	bits 30-16 of mail from CPU

0xFFFF	**CMBL**	**CPU Mailbox L**
dddd dddd dddd dddd		

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

Operation:

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

From DSP side, DSP loops by probing M bit. When this bit is 1 it reads CMBH first and then CMBL. After reading CMBL bit M of CMBH signalizing mail from CPU will be cleared.

DSP mailbox (DMB) is an interface to send 31 bits of information from DSP to 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 0 – Mailbox empty
		W	Does not matter. It will be set when DMBL is written.
14-0	d	W	bits 30-16 of mail from DSP to CPU

0xFFFD	DMBL	DSP Mailbox L
dddd dddd dddd dddd		

Bit	Name	R/W	Action
15-0	d	W	bits 15-0 of mail from DSP to CPU. Writing this register by DSP causes M bit of register DMBH to be set signalizing that mail is ready.

Operation:

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

4. DMA

GDSP is connected with memory bus through DMA channel. DMA can transfer data between DSP memory (both instruction and data) and main memory.

0xFFCE	DSMAH	Memory Address H
dddd dddd dddd dddd		

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

0xFFCF	DSMAL	Memory address L
dddd dddd dddd dddd		

Bit	Name	R/W	Action
15-0	d	R	bits 15-0 of main memory address

0xFFCD	DSPA	DSP Address
dddd dddd dddd dddd		

Bit	Name	R/W	Action
15-0	d	W	bits 15-0 of DSP memory address

0xFFCB	DSBL	DSP Address
dddd dddd dddd dddd		

Bit	Name	R/W	Action
15-0	d	W	length in bytes of transfer. writing to this register starts DMA transfer.

0xFFC9	DSCR	DSP Address
	---- ---- ---- ----	

Bit	Name	R/W	Action
15-0	d	W	

5. Accelerator

Accelerator is used to transfer data from accelerator memory (ARAM) to DSP. Accelerator area can be marked with ACSA (start) and ACEA (end) addresses. Current address for can be set or read from ACCA register. Reading from accelerator memory is done by reading from ACDAT register. This register contains data from ARAM pointed by ACCA register. After reading, ACCA is incremented by one. After ACCA grows bigger than area pointed by ACEA, it gets reset to a value from ACSA and ACCOV interrupt is generated.

VIII. Opcodes

1. Opcode syntax

Basic syntax of opcode:

```
OPC          opc_params
```

Above syntax is correct for all opcodes.

`OPC`	- opcode
`opc_params`	- opcode parameters if necessary

EXAMPLES:

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

Extended syntax:

```
OPC'EXOPC    opc_params : exopc_params
```

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

`OPC`	- opcode
`OPC`	- extended opcode
`opc_params`	- opcode parameters if necessary
`opc_params`	- opcode parameters for extended part if necessary

EXAMPLES:

```
DECM'L    $acs0 : $ac1.m, @ar0
NX'MV     : $acx1.h, $ac0.l
```

2. Operation - used functions

Functions used for describing operation of opcodes

PUSH_STACK($stR)

Description:

Pushes value onto given stack referenced by stack register *$stR*. Operation moves down values 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 setting corresponding bits in status register *$sr*.

Operation:

`EXECUTE_OPCODE(new_pc)`

Description:
Executes opcode at given new_pc address.

Operation:

3. Meaning of bits

Opcode decoding uses special naming for bits and their decimal representations to provide easier understanding of bit fields in 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 I or D 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)

4. Conditional opcodes

Conditional opcodes are being executed only when given condition described by conditional field has been met. To the group of conditional opcodes belong: `CALL, JMP, IF, RET.`

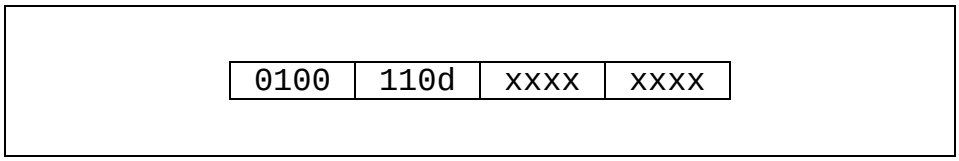
Bits	cc	Name	Evaluated expression
0000			
0001			
0010			
0011			
0100	EQ	Equal	
0101	NE	Not equal	
0110			
0111			
1000			
1001			
1010			
1011			
1100	ZR	Zero	$sr & 0x40
1101	NZ	Not zero	!($sr & 0x40)
1110			
1111		<always>	

Note:

There is two pairs of conditions that work similar: EQ/NE and ZR/NZ. EQ/NE pair operates on arithmetic zero flag (arithmetic 0) while ZR/NZ pair operates on logic zero flag (logic 0).

5. Opcodes decoding

ADD



Format:

ADD $acD, $ac(1-D)

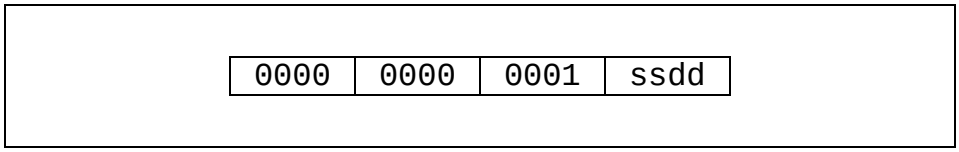
Description:

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

Operation:

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

ADDARN



Format:

ADDARN $arD, $ixS

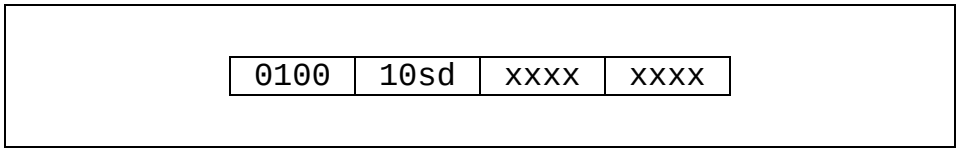
Description:

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

Operation:

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

ADDAX



Format:

ADDAX $acD, $axS

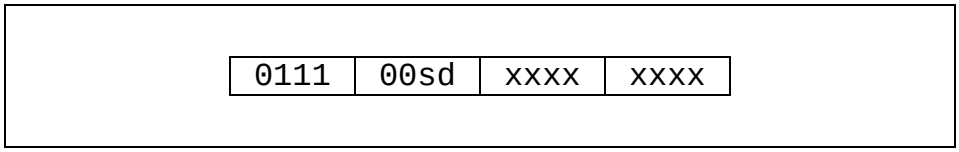
Description:

Adds secondary accumulator $axS to accumulator register $acD.

Operation:

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

ADDAXL



Format:

ADDAXL $acD, $axS.l

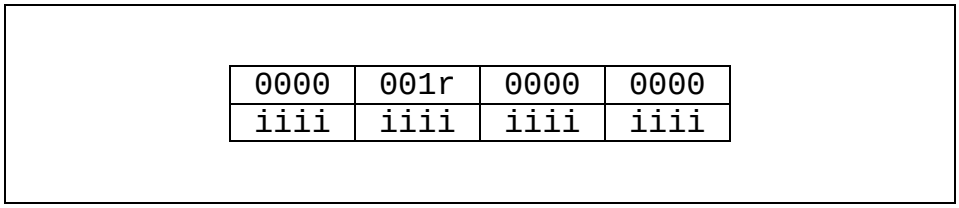
Description:

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

Operation:

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

ADDI



Format:

ADDI $amR, #I

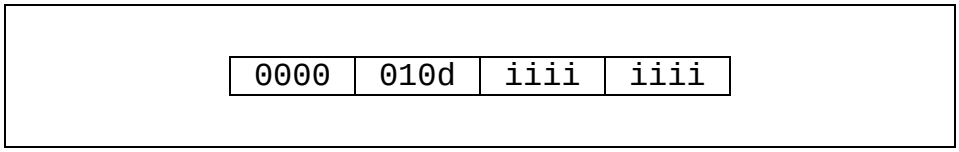
Description:

Adds immediate (16-bit sign extended) to mid accumulator $acD.hm.

Operation:

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

ADDIS



Format:

ADDIS $acD, #I

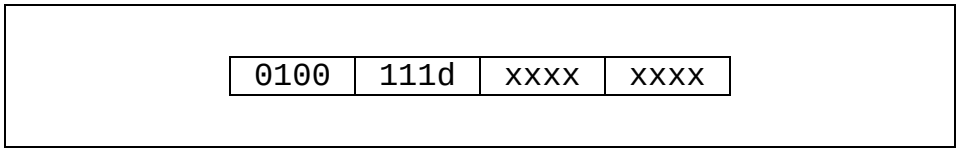
Description:

Adds short immediate (8-bit sign extended) to mid accumulator $acD.hm.

Operation:

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

ADDP



Format:

ADDP $acD

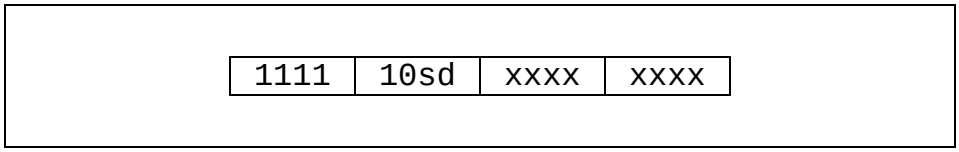
Description:

Adds product register to accumulator register.

Operation:

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

ADDPAXZ



Format:

ADDPAXZ $acD, $axS

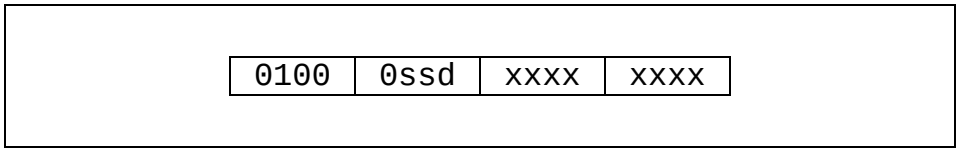
Description:

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++
```

ADDR



Format:

ADDR $acD, $(0x18+S)

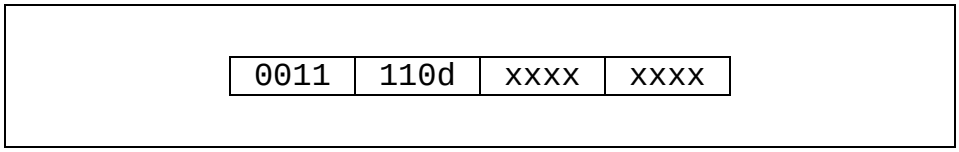
Description:

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

Operation:

```
$acD += $(0x18+S)
FLAGS($acD)
$pc++
```

ANDC



Format:

AMDC $acD.m, $ac(1-D).m

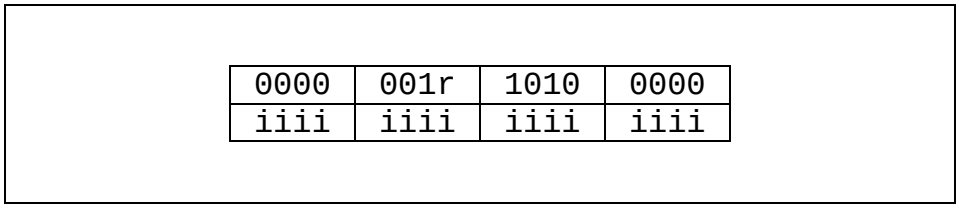
Description:

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

Operation:

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

ANDCF



0000	001r	1010	0000
iiii	iiii	iiii	iiii

Format:

ANDCF $acD.m, #I

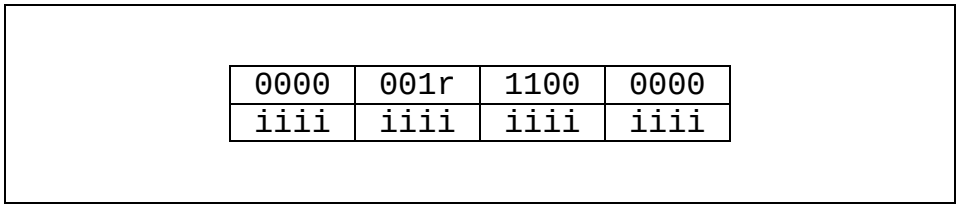
Description:

Set logic zero (LZ) flag in status register $sr if result of logical AND operation of accumulator mid part $acD.m with immediate value I is equal immediate value I.

Operation:

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

ANDF



0000	001r	1100	0000
iiii	iiii	iiii	iiii

Format:

ANDF $acD.m, #I

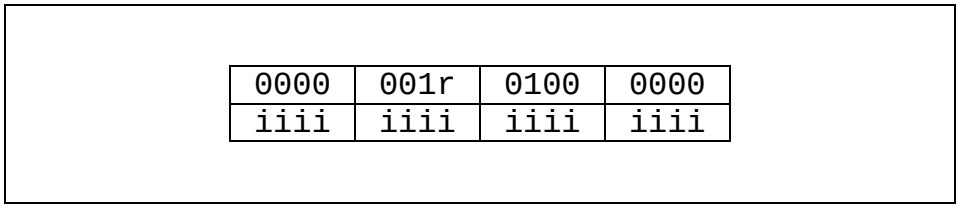
Description:

Set logic zero (LZ) flag in status register $sr if result of logic AND of accumulator mid part $acD.m with immediate value I is equal zero.

Operation:

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

ANDI



Format:

ANDI $acD.m, #I

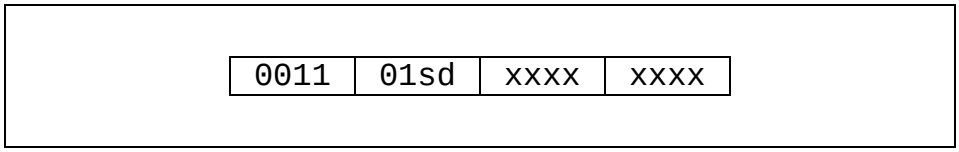
Description:

Logic AND of accumulator mid part $acD.m with immediate value I.

Operation:

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

ANDR



Format:

ANDR $acD.m, $axS.h

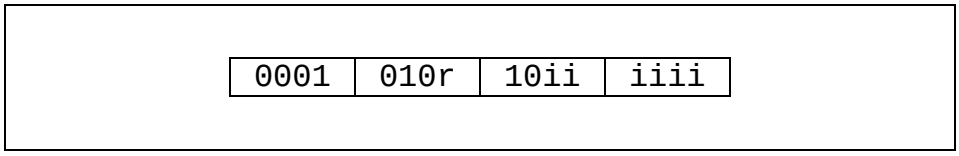
Description:

Logic AND middle part of accumulator $acD.m with hight part of secondary accumulator $axS.h.

Operation:

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

ASL



Format:

ASL $acR, #I

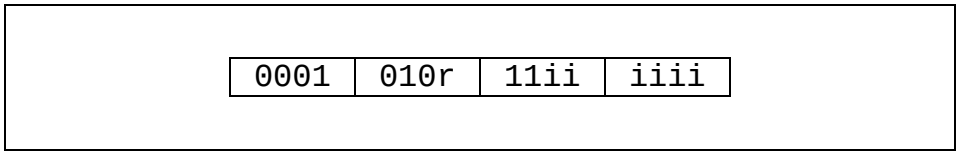
Description:

Logically shifts left accumulator $acR by number specified by value I.

Operation:

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

ASR



Format:

ASR $acR, #I

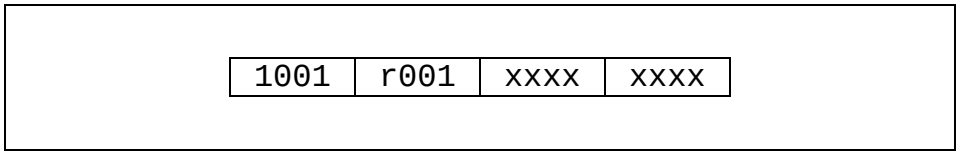
Description:

Arithmetically shifts left accumulator $acR by number specified by value calculated by negating sign extended bits 0-6.

Operation:

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

ASR16



Format:

ASR16 $acR

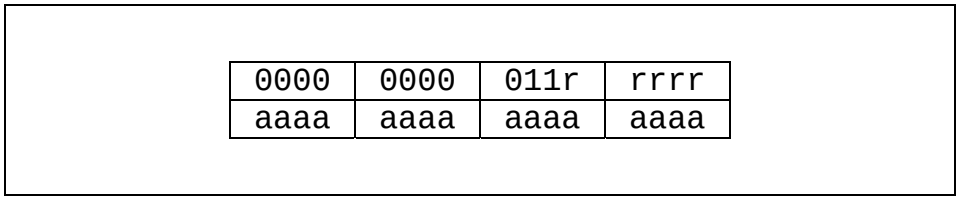
Description:

Arithmetically shifts right accumulator $acR by 16.

Operation:

```
$acR >>= 16
FLAGS($acD)
$pc++
```

BLOOP



0000	0000	011r	rrrr
aaaa	aaaa	aaaa	aaaa

Format:

BLOOP $R, addrA

Description:

Repeatedly execute block of code starting at following opcode until counter specified by value from register $R reaches zero. Block ends at specified address addrA inclusive, ie. 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 repeat address is pushed on call stack $st0. Up to 4 nested loops is allowed.

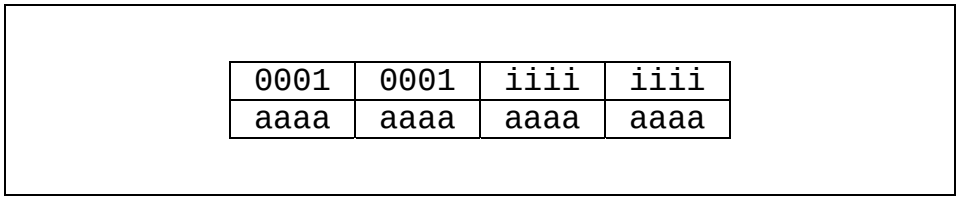
Operation:

```
     $st0 = $pc + 2
     $st2 = addrA
     $st3 = $R
     $pc + 2
     // in real hardware below does not happen, this
opcode only sets stack registers
     WHILE ($st3--)
          DO
               EXECUTE_OPCODE($pc)
          WHILE($pc != $st2)
          $pc = $st0
     $pc = addrA + 1
     // remove vaues from stack
```

See also:

Description of Stack registers explains how loop stacks are working

BLOOPI



0001	0001	iiii	iiii
aaaa	aaaa	aaaa	aaaa

Format:

BLOOPI #I, addrA

Description:

Repeatedly execute block of code starting at following opcode until counter specified by immediate value I reaches zero. Block ends at specified address addrA inclusive, ie. 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 repeat address is pushed on call stack $st0. Up to 4 nested loops is allowed.

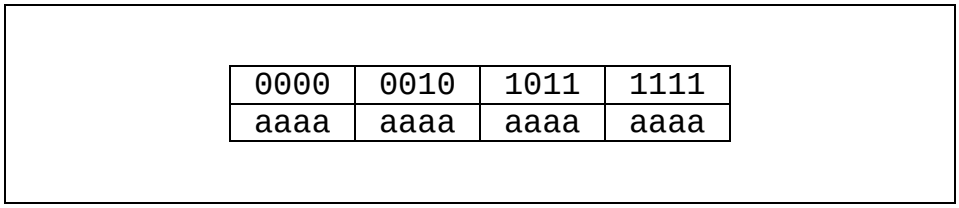
Operation:

```
     $st0 = $pc + 2
     $st2 = addrA
     $st3 = I
     $pc + 2
     // in real hardware below does not happen, this
opcode only sets stack registers
     WHILE ($st3--)
          DO
               EXECUTE_OPCODE($pc)
          WHILE($pc != $st2)
          $pc = $st0
     $pc = addrA + 1
     // remove vaues from stack
```

See also:

Description of Stack registers explains how loop stacks are working

CALL



0000	0010	1011	1111
aaaa	aaaa	aaaa	aaaa

Format:

CALL addressA

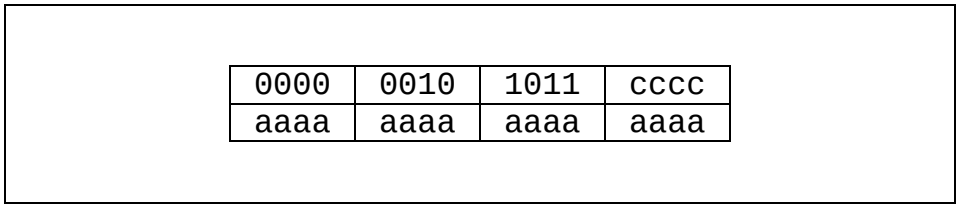
Description:

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

Operation:

```
// must skip value that follows “call”
PUSH_STACK($st0)
$st0 = $pc + 2
$pc = addressA
```

CALLcc



0000	0010	1011	cccc
aaaa	aaaa	aaaa	aaaa

Format:

CALLcc addressA

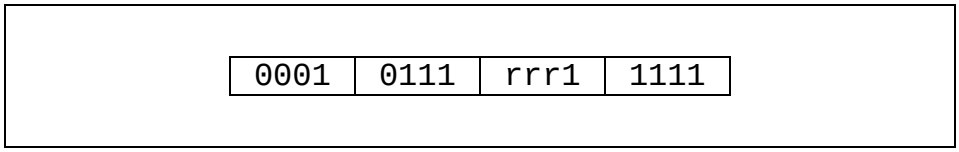
Description:

Call function if condition cc has been met. Push program counter of instruction following “call” to call stack $st0. Set program counter to address represented by 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
```

CALLR



Format:

CALLR $R

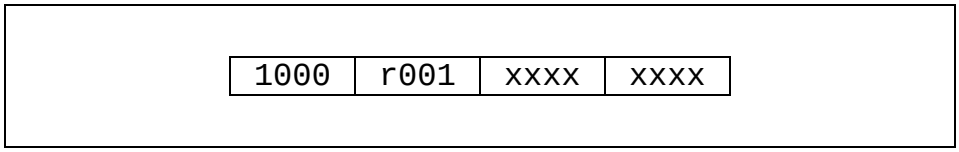
Description:

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

Operation:

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

CLR



Format:

CLR $acR

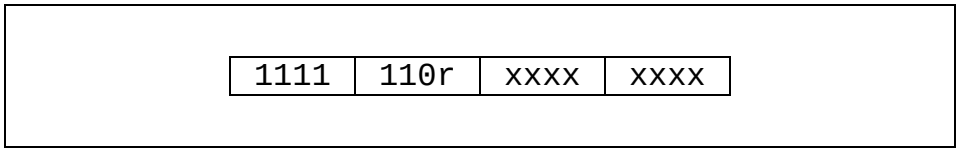
Description:

Clears accumulator $acR

Operation:

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

CLRL



Format:

CLRD $acR.l

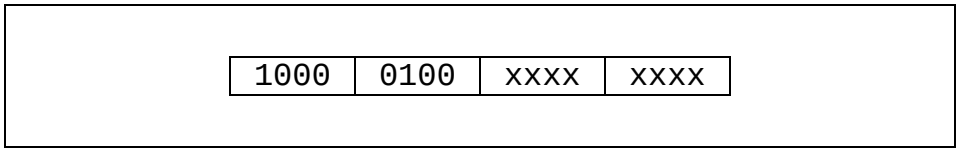
Description:

Clears $acR.l - low 16 bits of accumulator $acR.

Operation:

```
$acR.l = 0
FLAGS($acR)
$pc++
```

CLRP



Format:

CLRP

Description:

Clears product register $prod.

Operation:

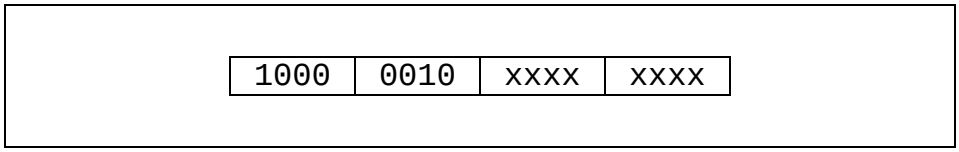
```
$prod = 0 // see note below
$pc++
```

Note:

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

```
$14 = 0x0000
$15 = 0xfff0
$16 = 0x00ff
$17 = 0x0010
```

CMP



Format:

CMP

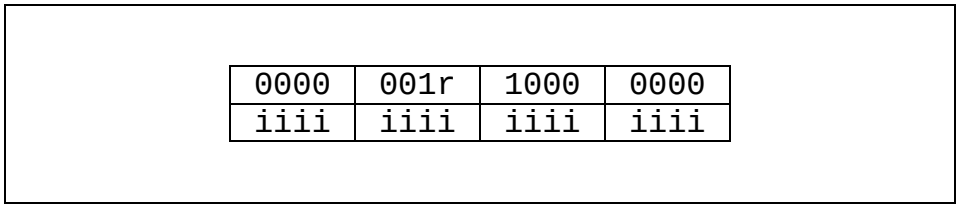
Description:

Compares accumulator $ac0 with accumulator $ac1.

Operation:

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

CMPI



Format:

CMPI $amD, #I

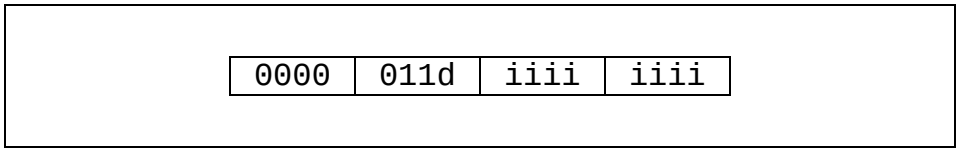
Description:

Compares mid accumulator $acD.hm ($amD) with sign extended immediate value I. Although flags are being set regarding whole accumulator register.

Operation:

```
res = ($acD.hm – I) | $acD.l
FLAGS(res)
$pc++
```

CMPIS



Format:

CMPIS $acD, #I

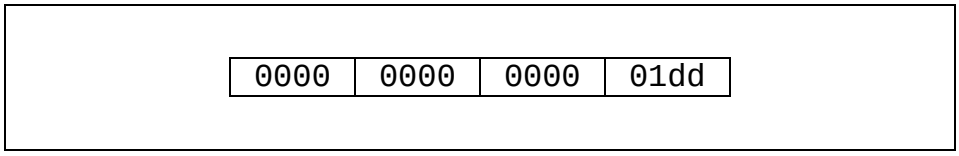
Description:

Compares accumulator with short immediate. Comaprison is executed by subtracting short immediate (8bit sign extended) from mid accumulator $acD.hm and computing flags based on whole accumulator $acD.

Operation:

```
FLAGS($acD - #I)
$pc++
```

DAR



Format:

DAR $arD

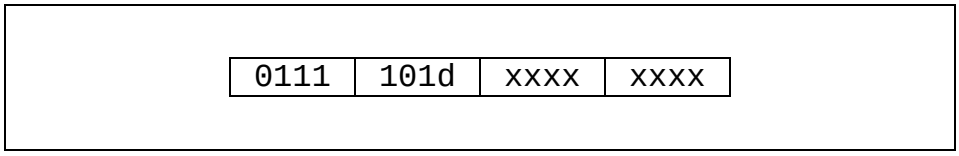
Description:

Decrement address register $arD.

Operation:

```
$arD--
$pc++
```

DEC



Format:

DEC $acD

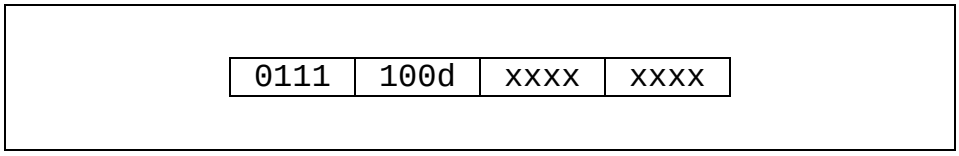
Description:

Decrement accumulator $acD.

Operation:

```
$acD--;
FLAGS($acD);
$pc++;
```

DECM



Format:

DECM $acsD

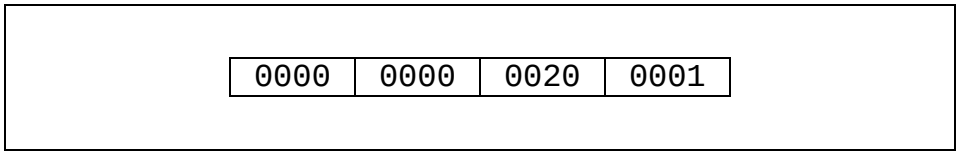
Description:

Decrement 24-bit mid-accumulator $acsD.

Operation:

```
$acsD--;
FLAGS($acD);
$pc++;
```

HALT



Format:

HALT

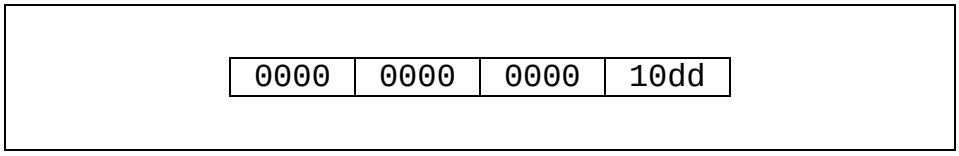
Description:

Stops execution of DSP code. Sets bit DSP_CR_HALT in register DREG_CR.

Operation:

```
DREG_CR |= DSP_CR_HALT;
```

IAR



Format:

IAR $arD

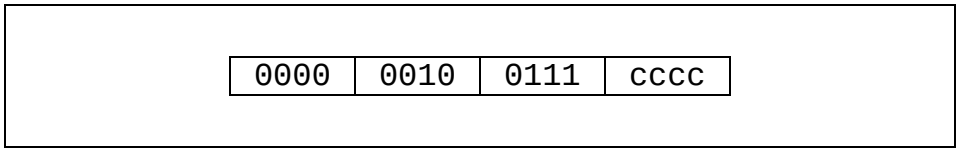
Description:

Increment address register $arD.

Operation:

```
$arD++
$pc++
```

IFcc



Format:

IFcc

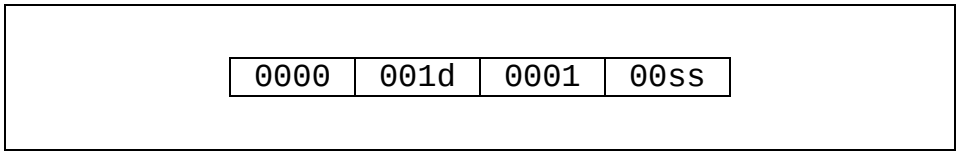
Description:

Execute following opcode if the condition has been met.

Operation:

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

ILRR



Format:

ILRR $acD.m, @$arS

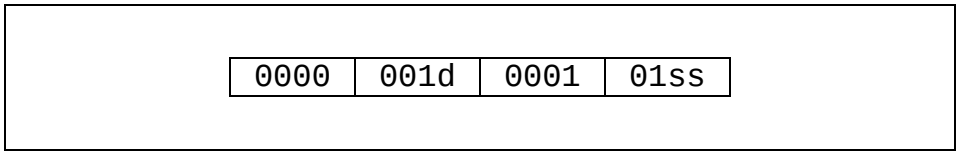
Description:

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

Operation:

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

ILRRD



Format:

ILRRD $acD.m, @$arS

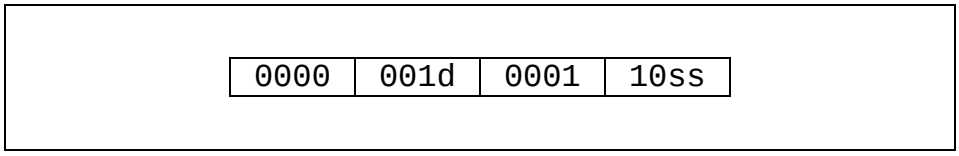
Description:

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

Operation:

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

ILRRI



Format:

ILRRI $acD.m, @$S

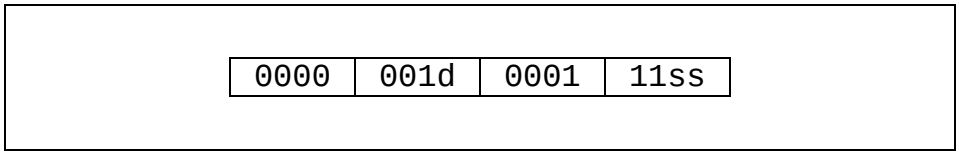
Description:

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

Operation:

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

ILRRN



Format:

ILRRN $acD.m, @$arS

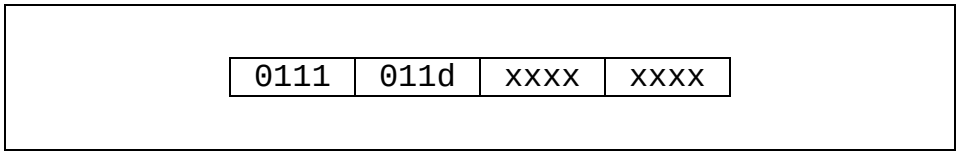
Description:

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.

Operation:

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

INC



Format:

INC $acD

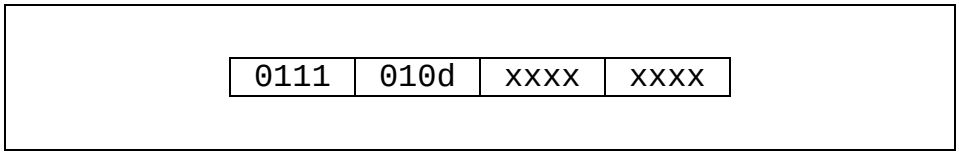
Description:

Increment accumulator $acD.

Operation:

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

INCM



Format:

INCM $acsD

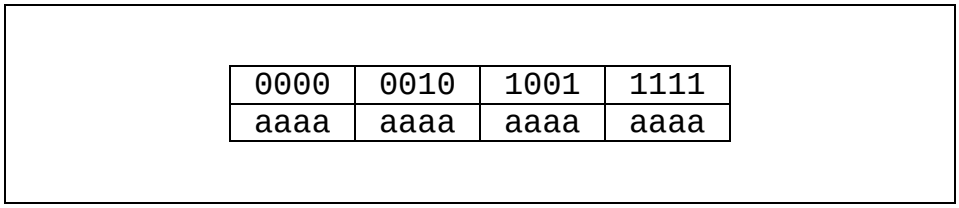
Description:

Increment 24-bit mid-accumulator $acsD.

Operation:

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

JMP



Format:

JMP addressA

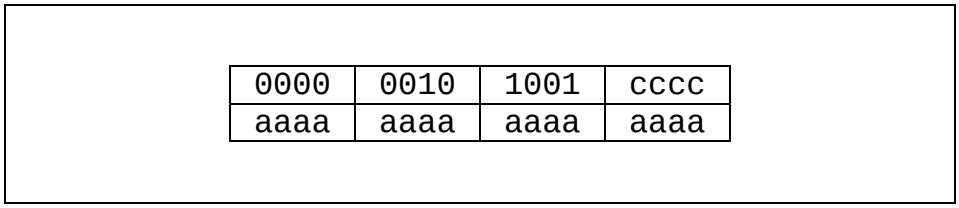
Description:

Jump to addressA. Set program counter to address represented by value that follows this “jmp” instruction.

Operation:

```
$pc = addressA
```

Jcc



Format:

Jcc addressA

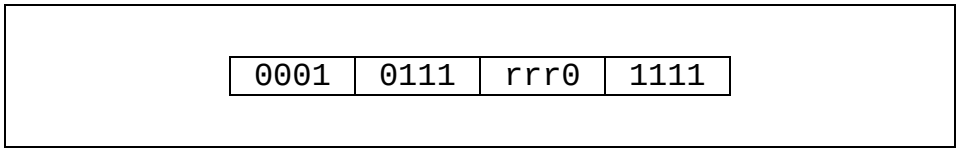
Description:

Jump to addressA if condition cc has been met. Set program counter to address represented by value that follows this “jmp” instruction.

Operation:

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

JMPR



Format:

JMP $R

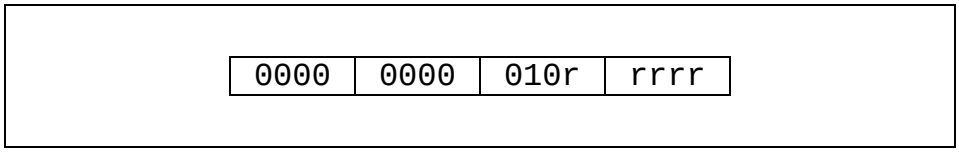
Description:

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

Operation:

```
$pc = $R
```

LOOP



Format:

LOOP $R

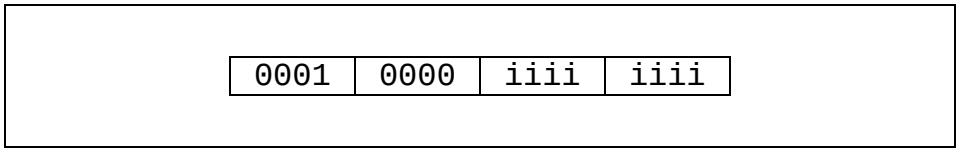
Description:

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

Operation:

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

LOOPI



Format:

LOOPI #I

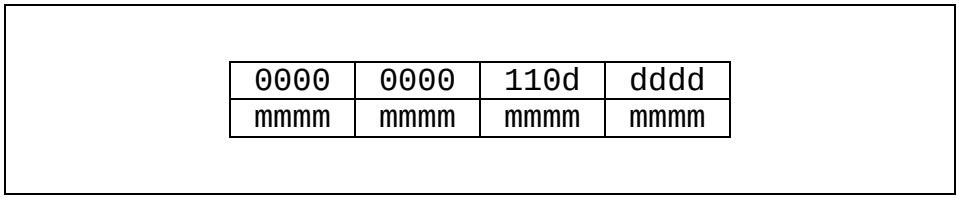
Description:

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

Operation:

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

LR



Format:

LR $D, @M

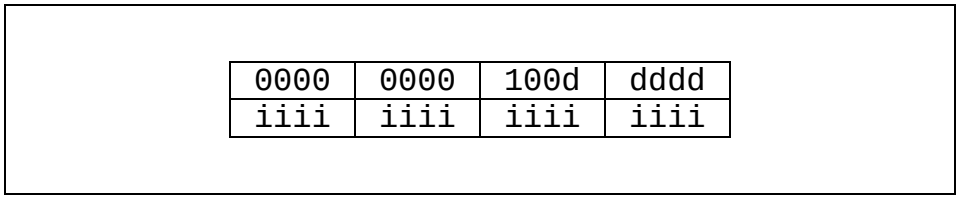
Description:

Move value from data memory pointed by address M to register $D. Perform additional operation depending on destination register.

Operation:

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

LRI



Format:

LRI $D, #I

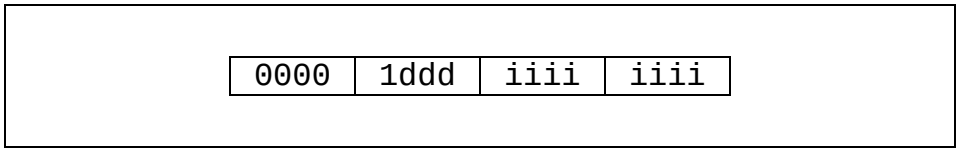
Description:

Load immediate value I to register $D. Perform additional operation depending on destination register.

Operation:

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

LRIS



Format:

LRIS $(0x18+D), #I

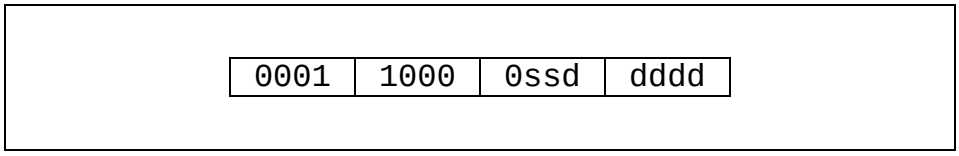
Description:

Load immediate value I (8-bit sign extended) to accumulator register $(0x18+D). Perform additional operation depending on destination register.

Operation:

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

LRR



Format:

LRR $D, @$S

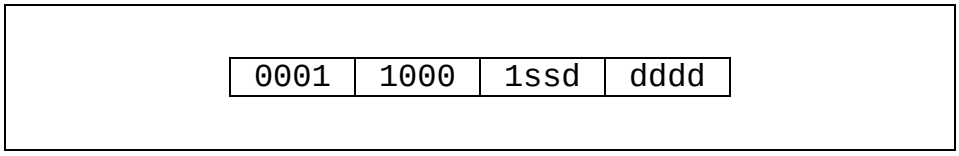
Description:

Move value from data memory pointed by addressing register $S to register $D. Perform additional operation depending on destination register.

Operation:

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

LRRD



Format:

LRRD $D, @$S

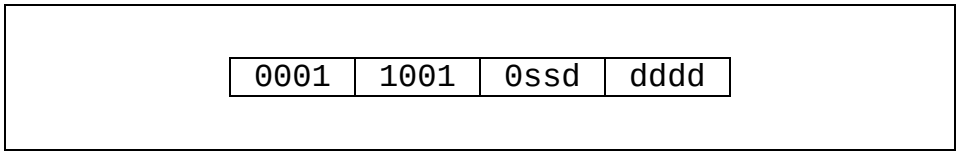
Description:

Move value from data memory pointed by addressing register $S to register $D. Decrement register $S. Perform additional operation depending on destination register.

Operation:

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

LRRI



Format:

LRRI $D, @$S

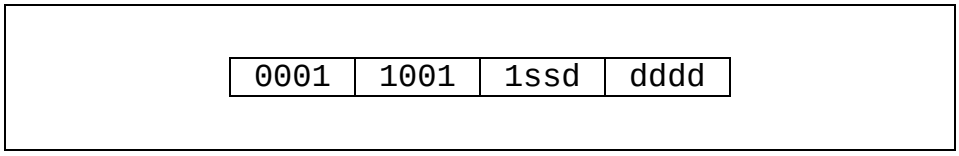
Description:

Move value from data memory pointed by addressing register $S to register $D. Increment register $S. Perform additional operation depending on destination register.

Operation:

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

LRRN



Format:

LRRN $D, @$S

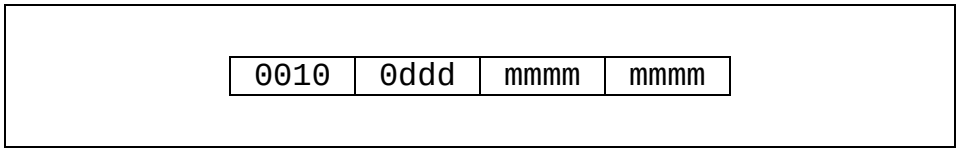
Description:

Move value from data memory pointed by addressing register $S to register $D. Add indexing register $(0x4+S) to register $S. Perform additional operation depending on destination register.

Operation:

```
$D = MEM[$S]
$S += $(4+S)
$pc++
```

LRS



Format:

LRS $(0x18+D), @M

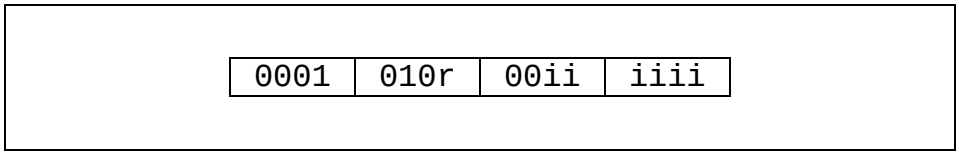
Description:

Move value from data memory pointed by address M (8-bit sign extended) to register $(0x18+D). Perform additional operation depending on destination register.

Operation:

```
$(0x18+D) = MEM[M]
$pc += 2
```

LSL



Format:

LSL $acR, #I

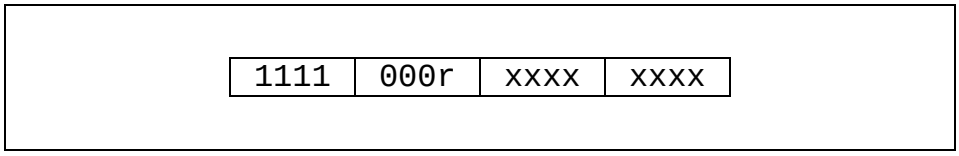
Description:

Logically shifts left accumulator $acR by number specified by value I.

Operation:

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

LSL16



Format:

LSL16$acR

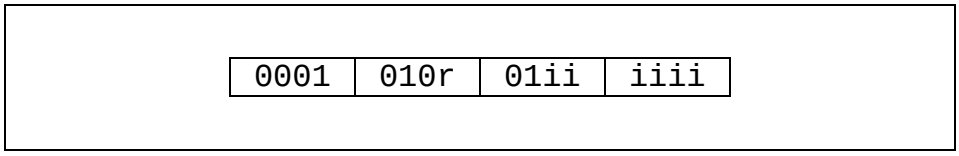
Description:

Logically shifts left accumulator $acR by 16.

Operation:

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

LSR



Format:

LSR $acR, #I

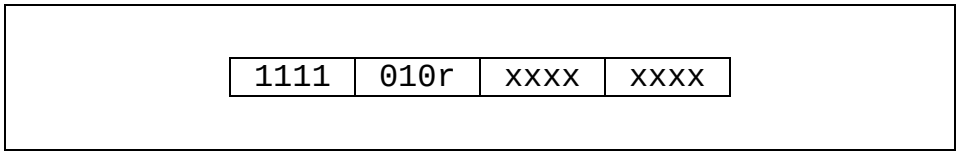
Description:

Logically shifts left accumulator $acR by number specified by value calculated by negating sign extended bits 0-6.

Operation:

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

LSR16



Format:

LSR16 $acR

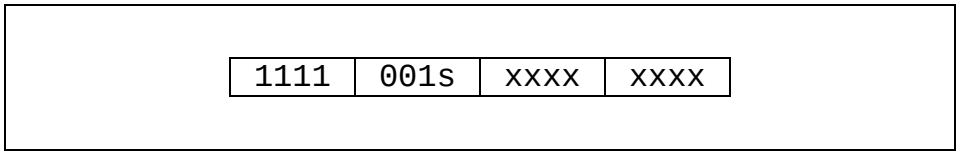
Description:

Logically shifts right accumulator $acR by 16.

Operation:

```
$acR >>= 16
FLAGS($acD)
$pc++
```

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.

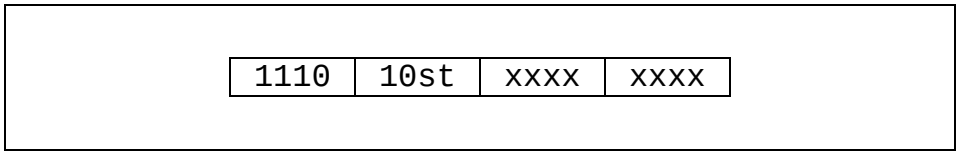
Operation:

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

See also:

$sr.AM bit affects multiply result

MADDC



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.

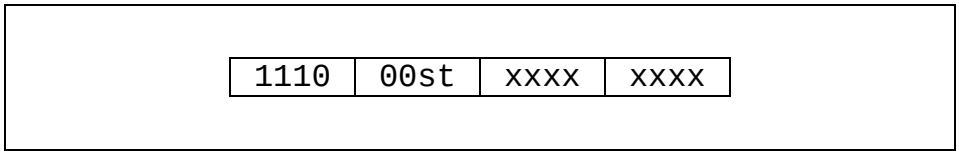
Operation:

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

See also:

$sr.AM bit affects multiply result

MADDX



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.

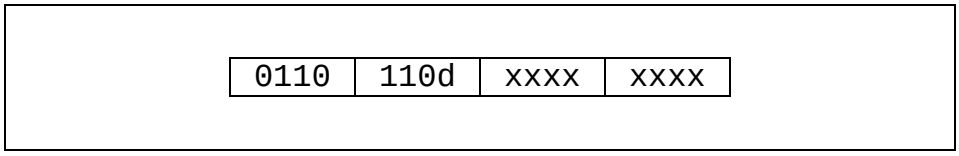
Operation:

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

See also:

$sr.AM bit affects multiply result

MOV



Format:

MOV $acD, $ac(1-D)

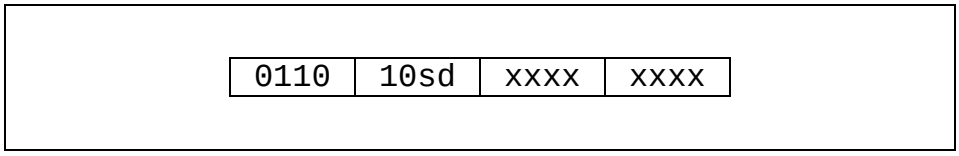
Description:

Moves accumulator $ax(1-D) to accumulator $axD.

Operation:

```
$acD = $ax(1-D)
FLAGS($acD)
$pc++
```

MOVAX



Format:

MOVAX $acD, $axS

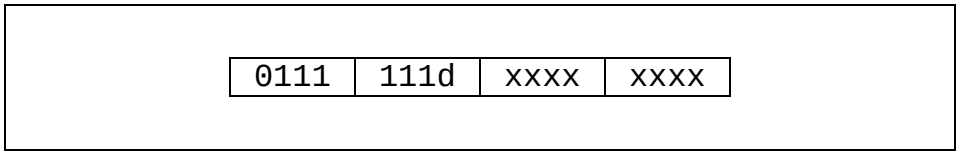
Description:

Moves secondary accumulator $axS to accumulator $axD.

Operation:

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

MOVNP



Format:

MOVNP $acD

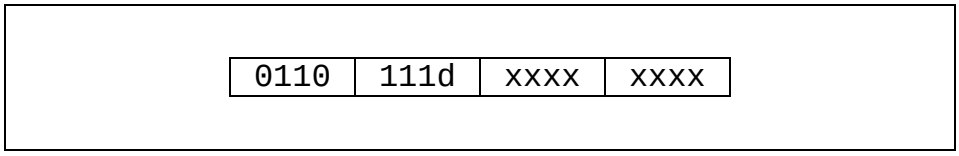
Description:

Moves negative of multiply product from $prod register to accumulator $acD register.

Operation:

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

MOVP



Format:

MOVP $acD

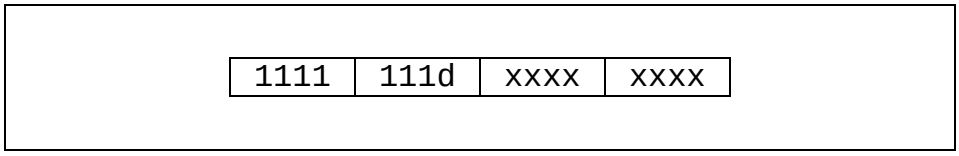
Description:

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

Operation:

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

MOVPZ



Format:

MOVPZ $acD

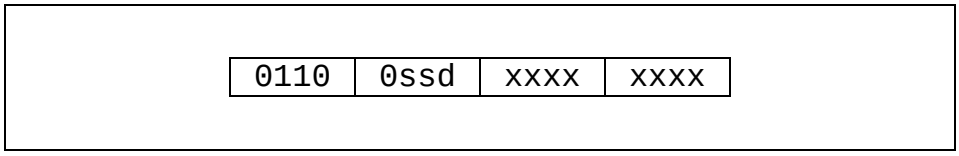
Description:

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

Operation:

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

MOVR



Format:

MOVR $acD, $(0x18+S)

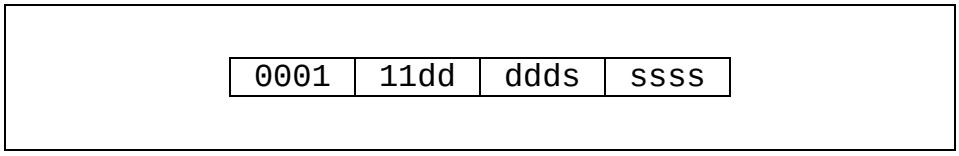
Description:

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++
```

MRR



Format:

MRR $D, $S

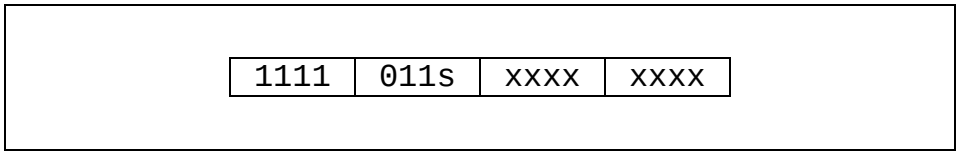
Description:

Move value from register $S to register $D. Perform additional operation depending on destination register.

Operation:

```
$D = $S
$pc++
```

MSUB



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.

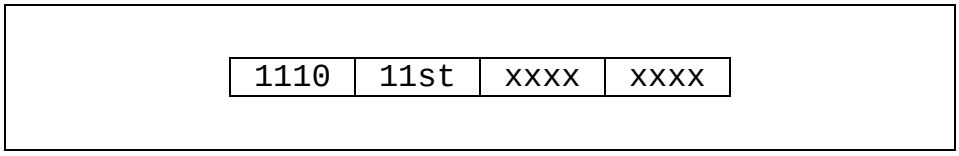
Operation:

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

See also:

$sr.AM bit affects multiply result

MSUBC



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.

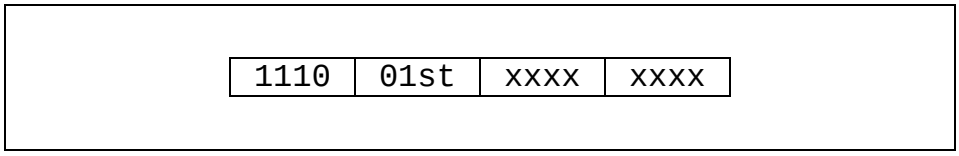
Operation:

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

See also:

$sr.AM bit affects multiply result

MSUBX



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.

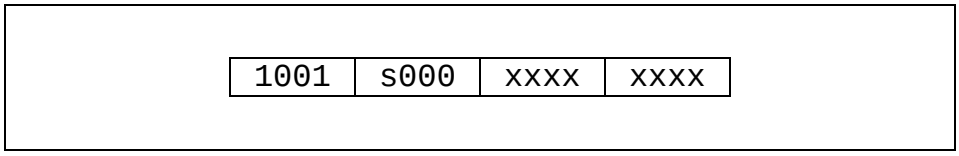
Operation:

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

See also:

$sr.AM bit affects multiply result

MUL



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).

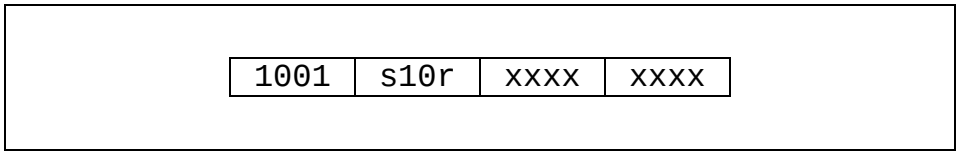
Operation:

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

See also:

$sr.AM bit affects multiply result

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).

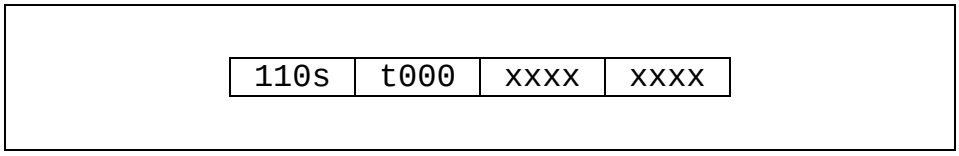
Operation:

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

See also:

$sr.AM bit affects multiply result

MULC



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).

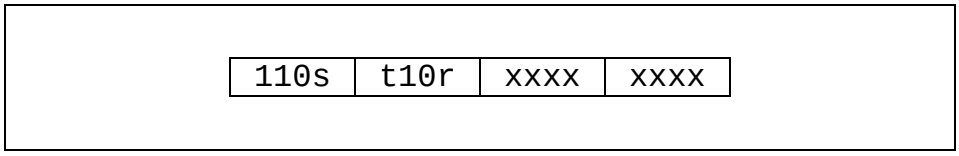
Operation:

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

See also:

$sr.AM bit affects multiply result

MULCAC



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.

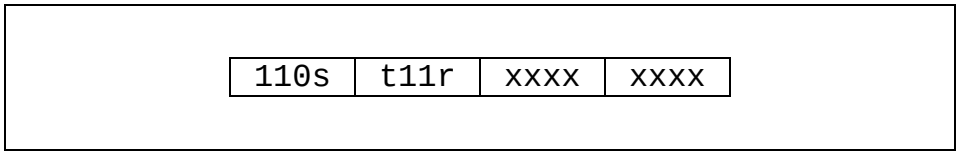
Operation:

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

See also:

$sr.AM bit affects multiply result

MULCMV



Format:

MULCMV $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). Move product register before multiplication to accumulator $acR.

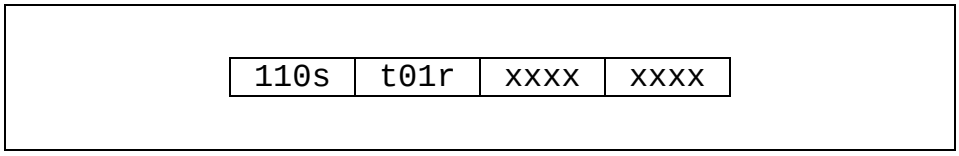
Operation:

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

See also:

$sr.AM bit affects multiply result

MULCMVZ



Format:

MULCMVZ $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). Move product register before multiplication to accumulator $acR, set low part of accumulator $acR.l to zero.

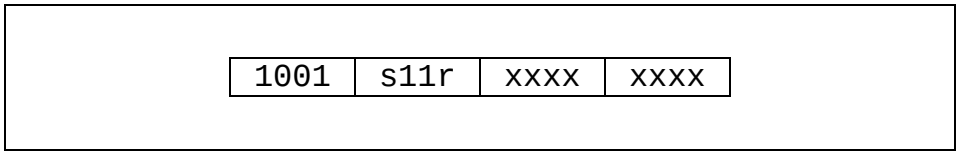
Operation:

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

See also:

$sr.AM bit affects multiply result

MULMV



Format:

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

Description:

Move 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).

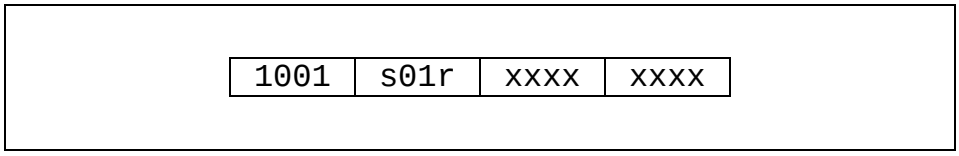
Operation:

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

See also:

$sr.AM bit affects multiply result

MULMVZ



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).

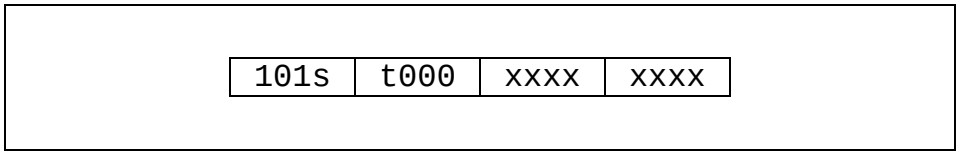
Operation:

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

See also:

$sr.AM bit affects multiply result

MULX



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.

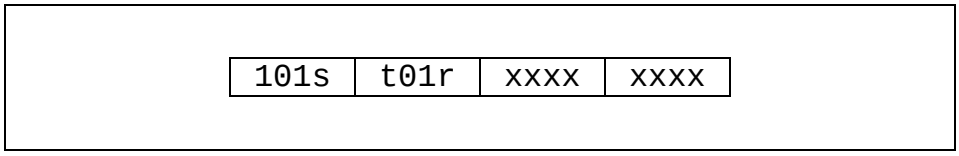
Operation:

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

See also:

$sr.AM bit affects multiply result

MULXAC



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.

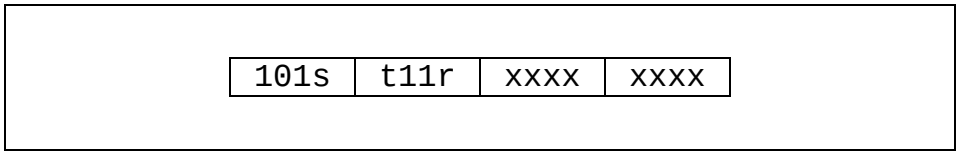
Operation:

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

See also:

$sr.AM bit affects multiply result

MULXMV



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.

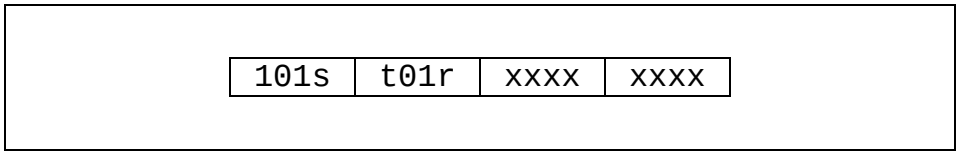
Operation:

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

See also:

$sr.AM bit affects multiply result

MULXMVZ



Format:

MULXMV $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.

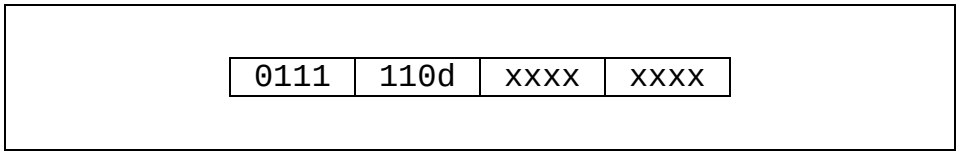
Operation:

```
$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

NEG



Format:

NEG $acD

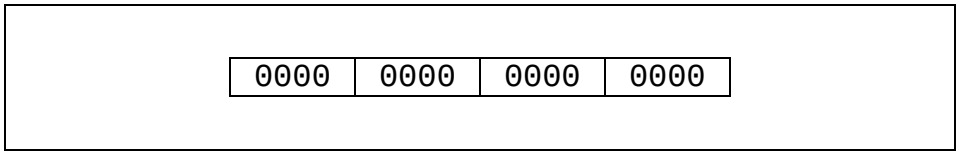
Description:

Negate accumulator $acD.

Operation:

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

NOP



Format:

NOP

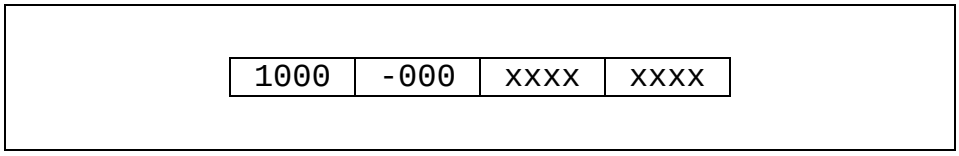
Description:

No operation.

Operation:

```
$pc++;
```

NX



Format:

NX

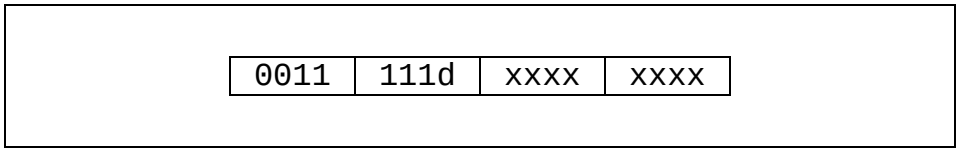
Description:

No operation, but can be extended with extended opcode.

Operation:

```
$pc++;
```

ORC



Format:

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

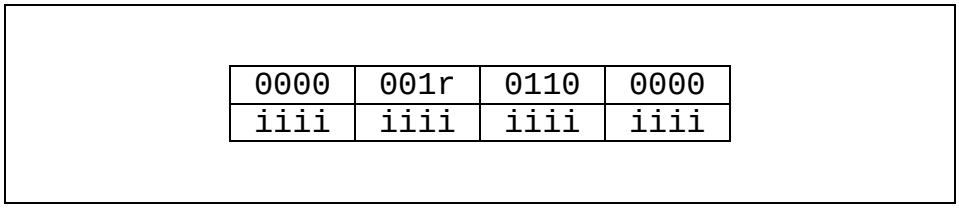
Description:

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

Operation:

```
$acD.m |= $ac(1-D).m
FLAGS($acD)
$pc++
```

ORI



Format:

ORI $acD.m, #I

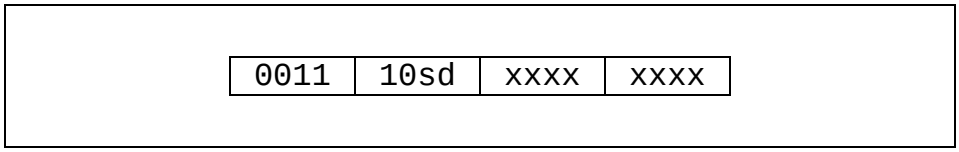
Description:

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

Operation:

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

ORR



Format:

ORR $acD.m, $axS.h

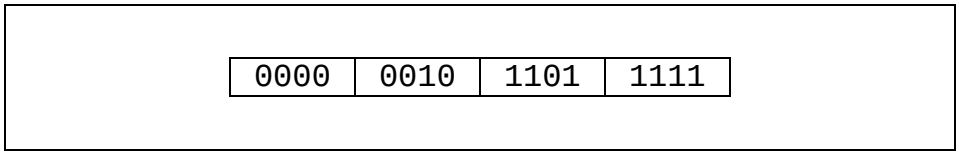
Description:

Logic OR middle part of accumulator $acD.m with hight part of secondary accumulator $axS.h.

Operation:

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

RET



Format:

RET

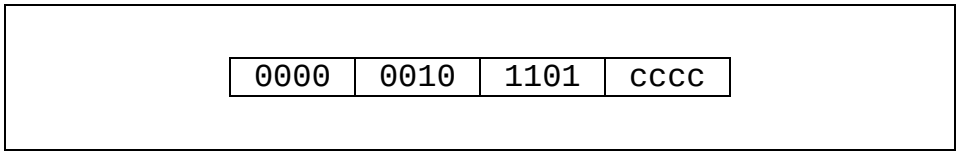
Description:

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

Operation:

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

RETcc



Format:

RETcc

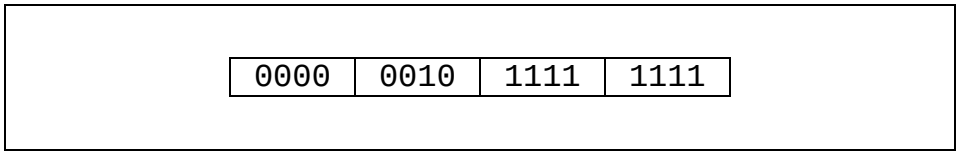
Description:

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)   $pc = POP_STACK($st0)
ELSE      $pc += 2
```

RTI



Format:

RTI

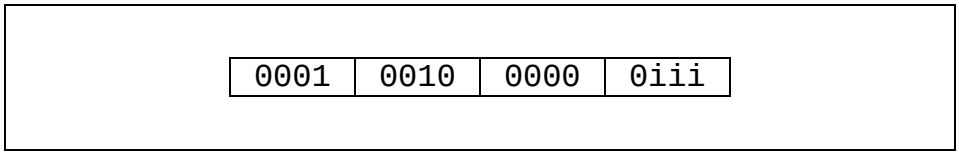
Description:

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)
```

SBSET



Format:

SBSET #I

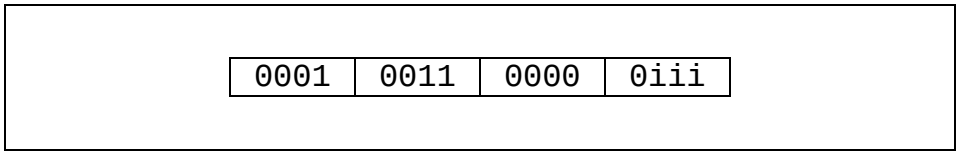
Description:

Set bit of status register $sr. Bit number is calculated by adding 6 to immediate value I.

Operation:

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

SBCLR



Format:

SBCLR #I

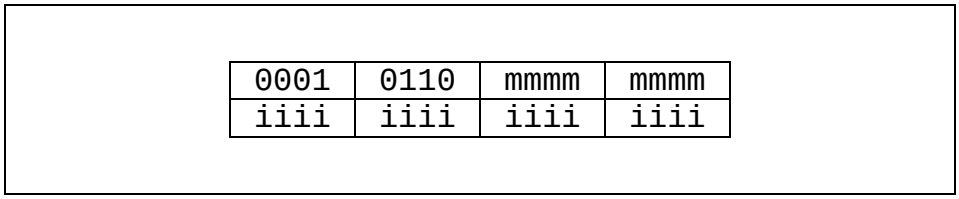
Description:

Clear bit of status register $sr. Bit number is calculated by adding 6 to immediate value I.

Operation:

```
$sr &= ~(I + 6)
$pc++
```

SI



Format:

SI @M, #I

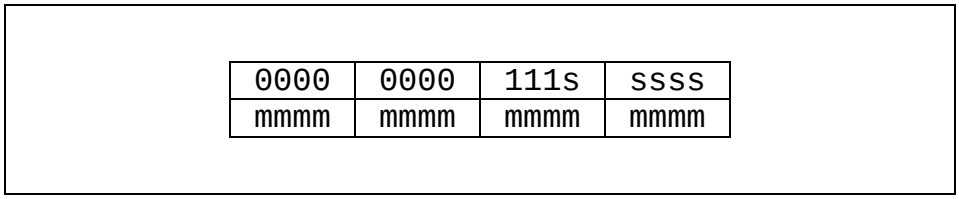
Description:

Store 16-bit immediate value I to a memory location pointed by address M (M is 8-bit value sign extended).

Operation:

```
MEM[M] = I
$pc += 2
```

SR



Format:

SR @M, $S

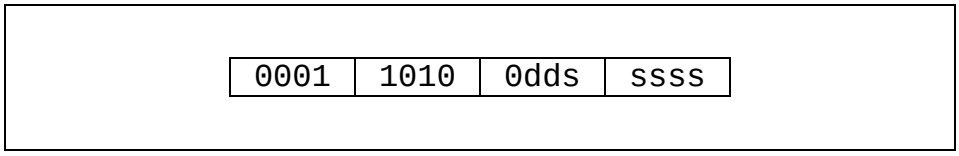
Description:

Store value from register $S to a memory pointed by address M. Perform additional operation depending on destination register.

Operation:

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

SRR



Format:

SRR @$D, $S

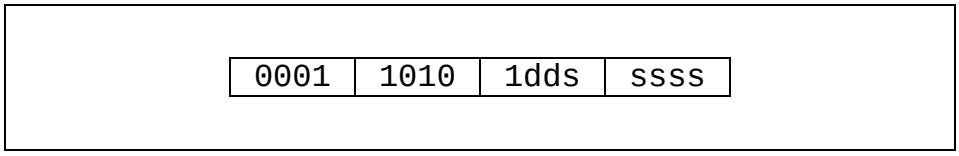
Description:

Store value from source register $S to a memory location pointed by addressing register $D. Perform additional operation depending on source register.

Operation:

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

SRRD



Format:

SRRD @$D, $S

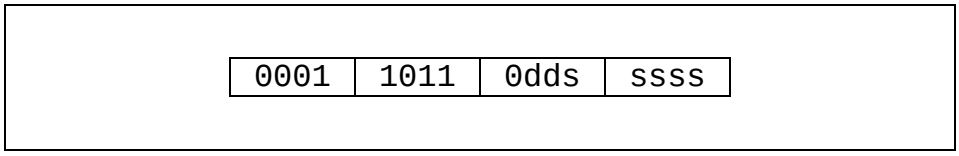
Description:

Store value from source register $S to a memory location pointed by addressing register $D. Decrement register $D. Perform additional operation depending on source register.

Operation:

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

SRRI



Format:

SRRI @$D, $S

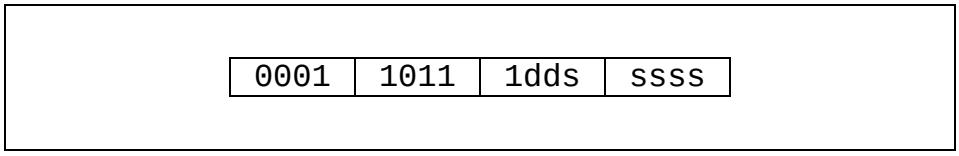
Description:

Store value from source register $S to a memory location pointed by addressing register $D. Increment register $D. Perform additional operation depending on source register.

Operation:

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

SRRN



Format:

SRRN @$D, $S

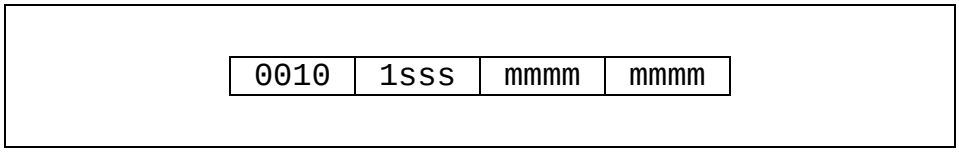
Description:

Store value from source register $S to a memory location pointed by addressing register $D. Add indexing register $(0x4+D) to register $D. Perform additional operation depending on source register.

Operation:

```
MEM[$D] = $S
$D += $(4+D)
$pc++
```

SRS



Format:

SRS @M, $(0x18+S)

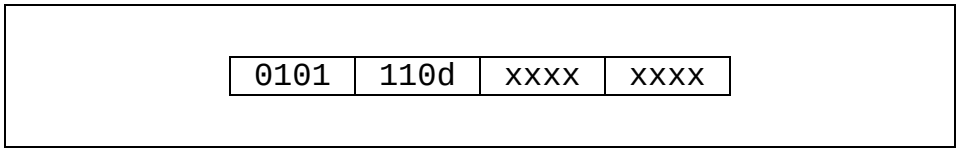
Description:

Store value from register $(0x18+S) to a memory pointed by address M. (8-bit sign extended). Perform additional operation depending on destination register.

Operation:

```
MEM[M] = $(0x18+S)
$pc += 2
```

SUB



Format:

SUB $acD, $ac(1-D)

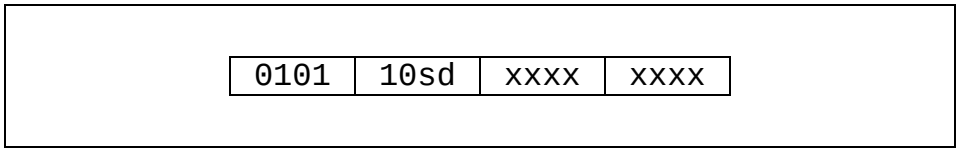
Description:

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

Operation:

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

SUBAX



Format:

SUBAX $acD, $axS

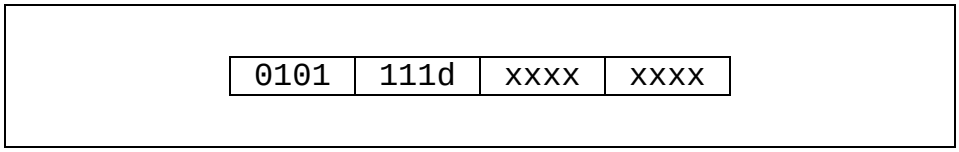
Description:

Subtracts secondary accumulator $axS from accumulator register $acD.

Operation:

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

SUBP



Format:

SUBP $acD

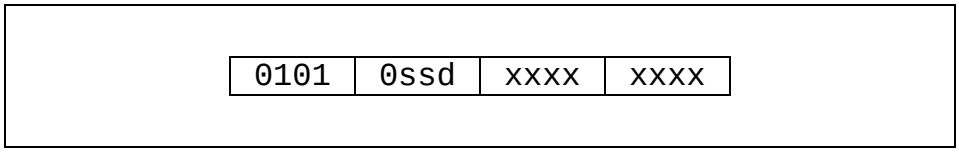
Description:

Subtracts product register from accumulator register.

Operation:

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

SUBR



Format:

SUBR $acD, $(0x18+S)

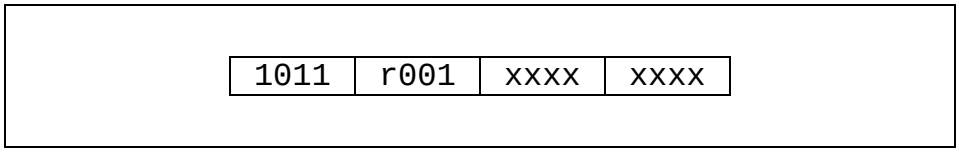
Description:

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

Operation:

```
$acD -= $(0x18+S)
FLAGS($acD)
$pc++
```

TST



Format:

TST $acR

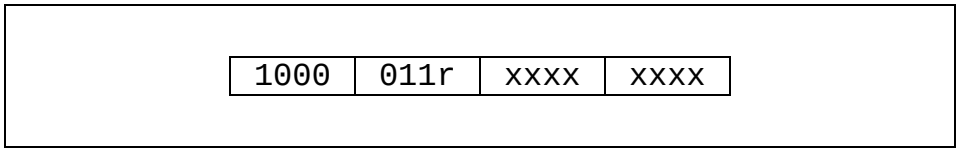
Description:

Test accumulator $acR

Operation:

```
FLAGS($acR)
$pc++
```

TSTAXH



Format:

TST $axR.h

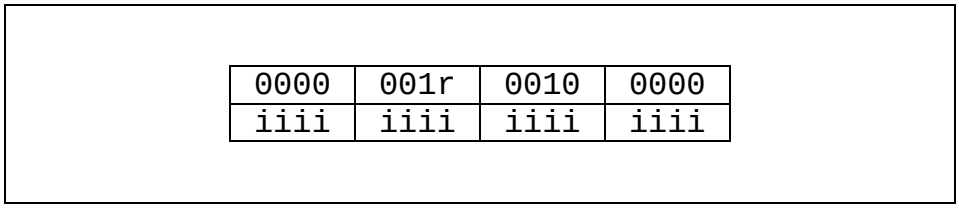
Description:

Test hight part of secondary accumulator $axR.h.

Operation:

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

XORI



Format:

XORI $acD.m, #I

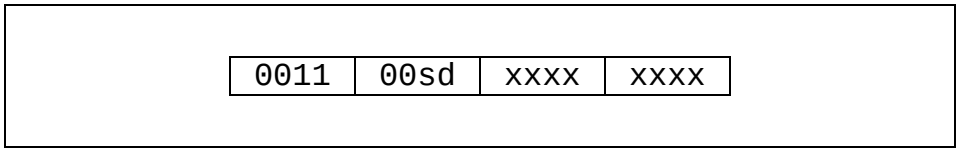
Description:

Logic exclusive or (XOR) of accumulator mid part $acD.m with immediate value I.

Operation:

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

XORR



Format:

XORR $acD.m, $axS.h

Description:

Logic XOR (exclusive or) middle part of accumulator $acD.m with hight part of secondary accumulator $axS.h.

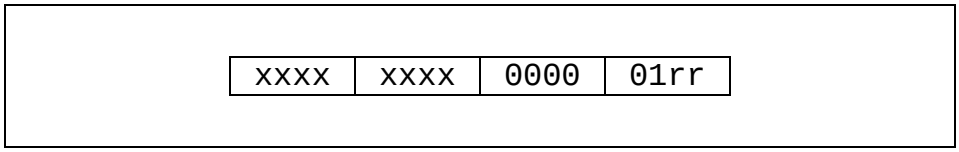
Operation:

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

6. Extended opcodes decoding

Extended opcodes do not exist on their own. These opcodes can only be attached to opcodes that allow extending (8 lower bits of opcode not used by opcode). Extended opcodes do not modify program counter $pc register.

‘DR



Format:

‘DR $arR

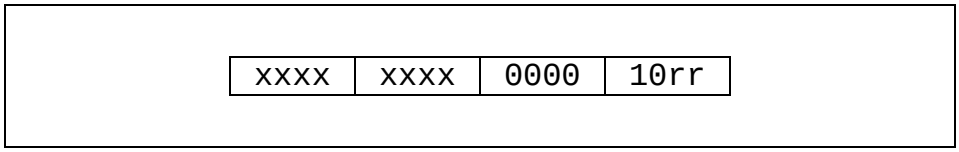
Description:

Decrement addressing register $arR.

Operation:

$arR—-

‘IR



Format:

‘IR $arR

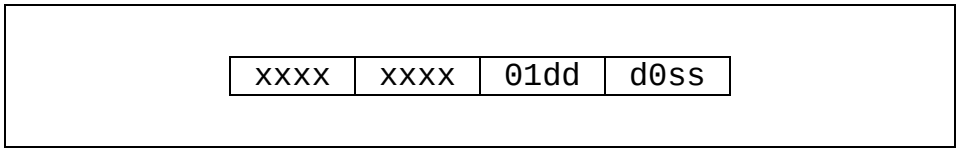
Description:

Increment addressing register $arR.

Operation:

$arR++

‘L



Format:

‘L $(0x18+D), @$S

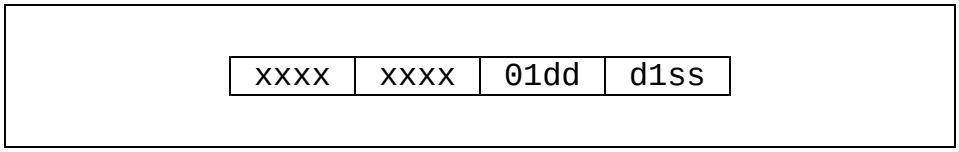
Description:

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

Operation:

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

‘LN



Format:

‘LN $(0x18+D), @$S

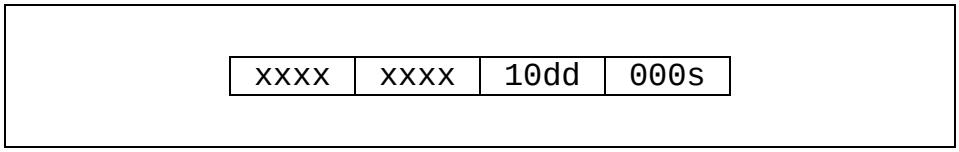
Description:

Load register $(0x18+D) with value from memory pointed by register $S. Add indexing register register $(0x04+S) to register $S.

Operation:

```
$(0x18+D) = MEM[$S]
$S += $(0x04+S)
```

‘LS



Format:

‘LS $(0x18+D), $acS.m

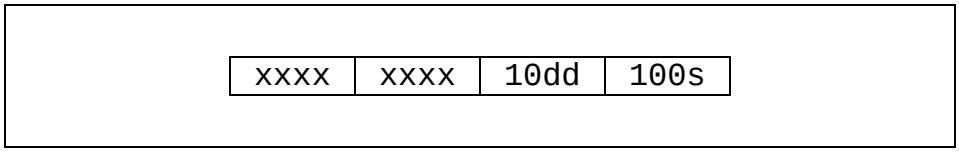
Description:

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.

Operation:

```
$(0x18+D) = MEM[$ar0]
MEM[$ar3] = $acS.m
$ar0++
$ar3++
```

‘LSM



Format:

‘LSM $(0x18+D), $acS.m

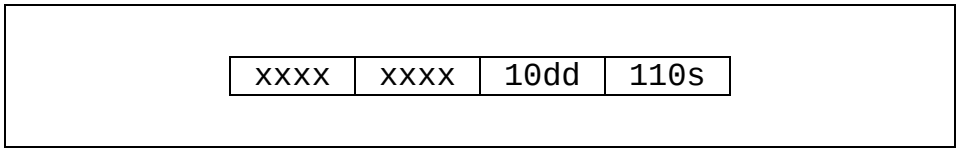
Description:

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.

Operation:

```
$(0x18+D) = MEM[$ar0]
MEM[$ar3] = $acS.m
$ar0++
$ar3 += $ix3
```

‘LSMN



Format:

‘LSMN $(0x18+D), $acS.m

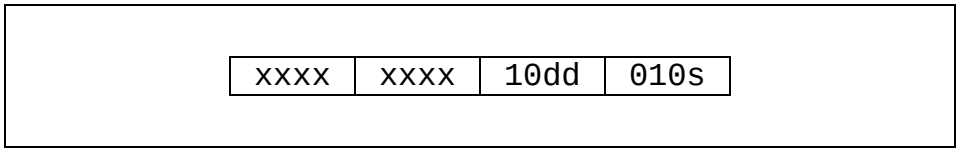
Description:

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.

Operation:

```
$(0x18+D) = MEM[$ar0]
MEM[$ar3] = $acS.m
$ar0 += $ix0
$ar3 += $ix3
```

‘LSN



Format:

‘LSN $(0x18+D), $acS.m

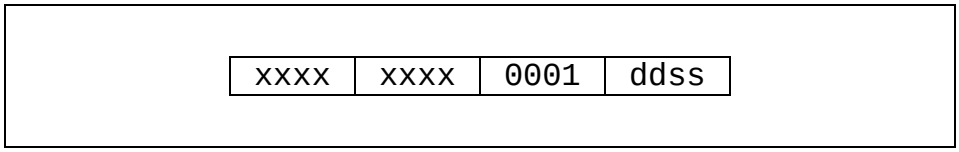
Description:

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.

Operation:

```
$(0x18+D) = MEM[$ar0]
MEM[$ar3] = $acS.m
$ar0 += $ix0
$ar3++
```

‘MV



Format:

‘MV $(0x18+D), $(0x1c+S)

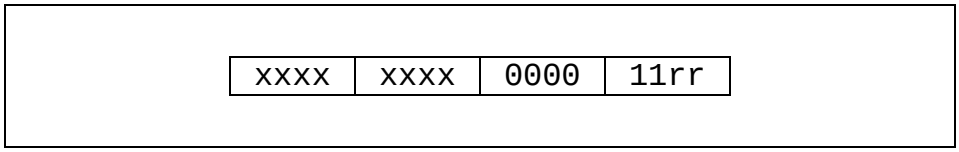
Description:

Move value of register $(0x1c+S) to the register $(0x18+D).

Operation:

```
$(0x18+D) = $(0x1c+S)
```

‘NR



Format:

‘NR $arR

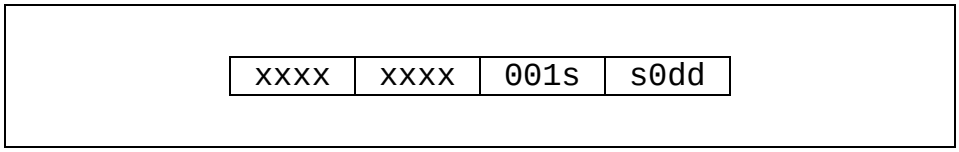
Description:

Add corresponding indexing register $ixR to addressing register $arR.

Operation:

```
$arR += $ixR
```

‘S



Format:

‘S @$D, $(0x1c+D)

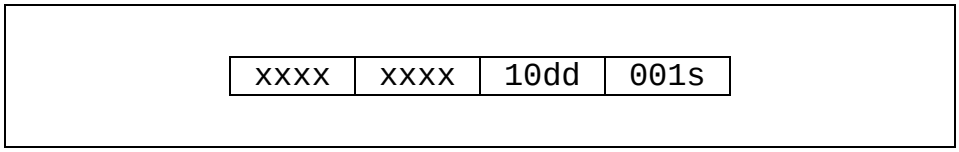
Description:

Store value of register $(0x1c+S) in the memory pointed by register $D. Post increment register $D.

Operation:

```
MEM[$D] = $(0x1c+D)
$S++
```

‘SL



Format:

‘SL $acS.m, $(0x18+D)

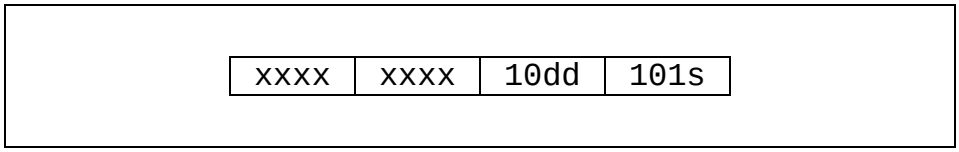
Description:

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.

Operation:

```
$(0x18+D) = MEM[$ar0]
MEM[$ar3] = $acS.m
$ar0++
$ar3++
```

‘SLM



Format:

‘SLM $acS.m, $(0x18+D)

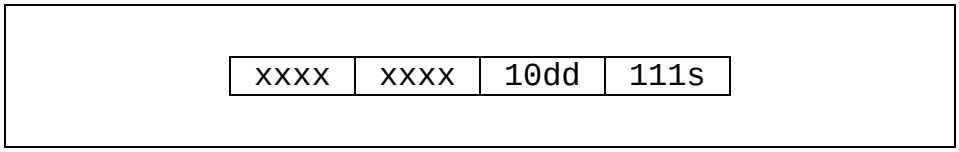
Description:

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.

Operation:

```
$(0x18+D) = MEM[$ar0]
MEM[$ar3] = $acS.m
$ar0++
$ar3 += $ix3
```

‘SLMN



Format:

‘SLMN $acS.m, $(0x18+D)

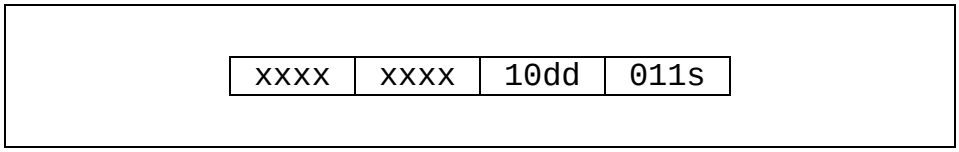
Description:

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.

Operation:

```
$(0x18+D) = MEM[$ar0]
MEM[$ar3] = $acS.m
$ar0 += $ix0
$ar3 += $ix3
```

‘SLN



Format:

‘SLN $acS.m, $(0x18+D)

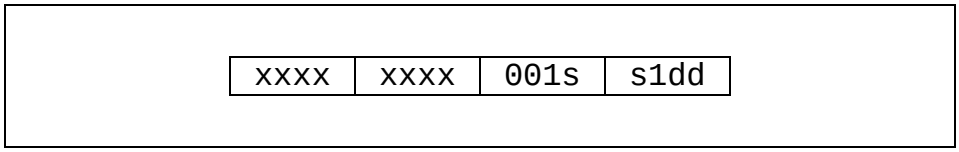
Description:

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.

Operation:

```
$(0x18+D) = MEM[$ar0]
MEM[$ar3] = $acS.m
$ar0 += $ix0
$ar3++
```

‘SN



Format:

‘SN @$D, $(0x1c+D)

Description:

Store value of register $(0x1c+S) in the memory pointed by register $D. Add indexing register register $(0x04+D) to register $D.

Operation:

```
MEM[$D] = $(0x1c+D)
$D += $(0x04+D)
```

7. Opcodes sorted by bit decoding

```
NOP                   *  0000 0000 0000 0000
DAR                   *  0000 0000 0000 01aa
IAR                   *  0000 0000 0000 10aa
XXX          NOT USED      0000 0000 0000 11xx
ADDARN                *  0000 0000 0001 bbaa
HALT                  *  0000 0000 0010 0001

LOOP                  *  0000 0000 010r rrrr
BLOOP                 *  0000 0000 011r rrrr

LRI                   *  0000 0000 100r rrrr iiii iiii iiii iiii
XXX          NOT USED      0000 0000 101x xxxx
LR                    *  0000 0000 110r rrrr mmmm mmmm mmmm mmmm
SR                    *  0000 0000 111r rrrr mmmm mmmm mmmm mmmm

IF cc                 *  0000 0010 0111 cccc
JMP cc                *  0000 0010 1001 cccc
CALL cc               *  0000 0010 1011 cccc
RET cc                *  0000 0010 1101 cccc

ADDI                  *  0000 001r 0000 0000 iiii iiii iiii iiii
XORI                  *  0000 001r 0010 0000 iiii iiii iiii iiii
ANDI                  *  0000 001r 0100 0000 iiii iiii iiii iiii
ORI                   *  0000 001r 0110 0000 iiii iiii iiii iiii
CMPI                  *  0000 001r 1000 0000 iiii iiii iiii iiii
ANDCF                 *  0000 001r 1010 0000 iiii iiii iiii iiii
ANDF                  *  0000 001r 1100 0000 iiii iiii iiii iiii

ILRR                  *  0000 001r 0001 mmaa

ADDIS                 *  0000 010d iiii iiii
CMPIS                 *  0000 011d iiii iiii
LRIS                  *  0000 1rrr iiii iiii

LOOPI                 *  0001 0000 iiii iiii aaaa aaaa aaaa aaaa
BLOOPI                *  0001 0001 iiii iiii aaaa aaaa aaaa aaaa
SBSET        bit set  *  0001 0010 ???? ?iii
SBCLR        bit clear*  0001 0011 ???? ?iii
LSL/LSR               *  0001 010r 0sss ssss
ASL/ASR               *  0001 010r 1sss ssss
SI                    *  0001 0110 iiii iiii mmmm mmmm mmmm mmmm
CALLR                 *  0001 0111 rrr1 1111
JMPR                  *  0001 0111 rrr0 1111
```

```
LRR(I|D|X)              *  0001 100x xaar rrrr
SRR(I|D|X)              *  0001 101x xaar rrrr
MRR                     *  0001 11dd ddds ssss

LRS                     *  0010 0rrr mmmm mmmm
SRS                     *  0010 1rrr mmmm mmmm

XORR                    *  0011 00sr xxxx xxxx
ANDR                    *  0011 01sr xxxx xxxx
ORR                     *  0011 10sr xxxx xxxx
ANDC                    *  0011 110r xxxx xxxx
ORC                     *  0011 111r xxxx xxxx

ADDR                    *  0100 0ssd xxxx xxxx
ADDAX                   *  0100 10sd xxxx xxxx
ADD                     *  0100 110d xxxx xxxx
ADDP                    *  0100 111d xxxx xxxx

SUBR                    *  0101 0ssd xxxx xxxx
SUBAX                   *  0101 10sd xxxx xxxx
SUB                     *  0101 110d xxxx xxxx
SUBP                    *  0101 111d xxxx xxxx

MOVR                    *  0110 0ssd xxxx xxxx
MOVAX                   *  0110 10sd xxxx xxxx
MOV                     *  0110 110d xxxx xxxx
MOVP                    *  0110 111d xxxx xxxx

ADDAXL                  *  0111 00sr xxxx xxxx
INCM                    *  0111 010r xxxx xxxx
INC                     *  0111 011r xxxx xxxx
DECM                    *  0111 100r xxxx xxxx
DEC                     *  0111 101r xxxx xxxx
NEG                     *  0111 110r xxxx xxxx
MOVNP                   *  0111 111r xxxx xxxx

NX                         1000 x000 xxxx xxxx
CLR                     *  1000 x001 xxxx xxxx
CMP                     *  1000 0010 xxxx xxxx
???          UNUSED        1000 0011 xxxx xxxx
CLRP                    *  1000 0100 xxxx xxxx
TSTAXH                  *  1000 011x xxxx xxxx
M0/M2                      1000 101x xxxx xxxx
CLR15/SET15                1000 110x xxxx xxxx
SET40/16                   1000 111x xxxx xxxx


MUL                     *  1001 a000 xxxx xxxx
```

```
ASR16                 *  1001 r001 xxxx xxxx
MULMVZ                *  1001 a01r xxxx xxxx
MULAC                 *  1001 a10r xxxx xxxx
MULMV                 *  1001 a11r xxxx xxxx

MULX                  *  101b a000 xxxx xxxx
???                      1010 r001 xxxx xxxx
TST                   *  1011 r001 xxxx xxxx
MULXMVZ               *  101b a01r xxxx xxxx
MULXAC                *  101b a10r xxxx xxxx
MULXMV                *  101b a11r xxxx xxxx


MULC                  *  110s a000 xxxx xxxx
CMP                   *  110x r001 xxxx xxxx
MULCMVZ               *  110s a01r xxxx xxxx
MULCAC                *  110s a10r xxxx xxxx
MULCMV                *  110s a11r xxxx xxxx

MADDX                 ** 1110 00st xxxx xxxx
MSUBX                 ** 1110 01st xxxx xxxx
MADDC                 ** 1110 10st xxxx xxxx
MSUBC                 ** 1110 11st xxxx xxxx

LSL16                 *  1111 000r xxxx xxxx
MADD                  *  1111 001s xxxx xxxx
LSR16                 *  1111 010r xxxx xxxx
MSUB                  *  1111 011s xxxx xxxx
ADDPAXZ               *  1111 10ar xxxx xxxx
CLRL                  *  1111 110r xxxx xxxx
MOVPZ                 *  1111 111r xxxx xxxx
```

Opcode Extensions

```
[D|I|N]R              *  xxxx xxxx 0000 nnaa
MV                    *  xxxx xxxx 0001 ddss
S[N]                  *  xxxx xxxx 001r rnaa
L[N]                  *  xxxx xxxx 01dd diss
LS[NM|M|N]            *  xxxx xxxx 10dd ba0r
SL[NM|M|N]            *  xxxx xxxx 10dd ba1r
LD[NM|M|N]               xxxx xxxx 11mn barr
LD2[NM|M|N]              xxxx xxxx 11rm ba11
```

IX. References

1. United States Patent No.: US 6,606,689 “Method and apparatus for pre-caching audio data”. Assigne: Nintendo Co., Ltd., Kyoto (JP). Inventors: Howard H. Cheng, Dan Shimizu, Genyo Takeda (http://www.uspto.gov)
2. Yet Another Gamecube Documentation by groepaz/hitmen (http://www.gc-linux.org/docs/yagcd.html)
3. LibOGC and DevkitPro by shagkur and WntrMute (http://sourceforge.net/projects/devkitpro)