NEC

User's Manual

μPD77016 Family

Digital Signal Processor

Instructions

μPD77015
μPD77016
μPD77017
μPD77018
μPD77018A
μPD77019
μPD77110
μPD77111
μPD77112
μPD77113
μPD77114

Document No. U13116EJ2V0UM00 (2nd edition)
Date Published July 2000 N CP(K)

Printed in Japan

[MEMO]

NOTES FOR CMOS DEVICES

① PRECAUTION AGAINST ESD FOR SEMICONDUCTORS

Note:

Strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it once, when it has occurred. Environmental control must be adequate. When it is dry, humidifier should be used. It is recommended to avoid using insulators that easily build static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work bench and floor should be grounded. The operator should be grounded using wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with semiconductor devices on it.

② HANDLING OF UNUSED INPUT PINS FOR CMOS

Note:

No connection for CMOS device inputs can be cause of malfunction. If no connection is provided to the input pins, it is possible that an internal input level may be generated due to noise, etc., hence causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using a pull-up or pull-down circuitry. Each unused pin should be connected to V_{DD} or GND with a resistor, if it is considered to have a possibility of being an output pin. All handling related to the unused pins must be judged device by device and related specifications governing the devices.

③ STATUS BEFORE INITIALIZATION OF MOS DEVICES

Note:

Power-on does not necessarily define initial status of MOS device. Production process of MOS does not define the initial operation status of the device. Immediately after the power source is turned ON, the devices with reset function have not yet been initialized. Hence, power-on does not guarantee out-pin levels, I/O settings or contents of registers. Device is not initialized until the reset signal is received. Reset operation must be executed immediately after power-on for devices having reset function.

The export of these products from Japan is regulated by the Japanese government. The export of some or all of these products may be prohibited without governmental license. To export or re-export some or all of these products from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

License not needed:	μPD77016, μPD77019-013, μPD77110
The customer must judge the the need for licence:	μPD77015, μPD77017, μPD77018, μPD77018A, μPD77019, μPD77111, μPD77112, μPD77113, μPD77114

- **The information in this document is current as of February, 2000. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC's data sheets or data books, etc., for the most up-to-date specifications of NEC semiconductor products. Not all products and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without prior written consent of NEC. NEC assumes no responsibility for any errors that may appear in this document.
- NEC does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC semiconductor products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC endeavours to enhance the quality, reliability and safety of NEC semiconductor products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC semiconductor products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment, and anti-failure features.
- NEC semiconductor products are classified into the following three quality grades:
  "Standard", "Special" and "Specific". The "Specific" quality grade applies only to semiconductor products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of a semiconductor product depend on its quality grade, as indicated below. Customers must check the quality grade of each semiconductor product before using it in a particular application.
  "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots
  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support)
  "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.
  The quality grade of NEC semiconductor products is "Standard" unless otherwise expressly specified in NEC's data sheets or data books, etc. If customers wish to use NEC semiconductor products in applications not intended by NEC, they must contact an NEC sales representative in advance to determine NEC's willingness to support a given application.
  (Note)
  (1) "NEC" as used in this statement means NEC Corporation and also includes its majority-owned subsidiaries.
  (2) "NEC semiconductor products" means any semiconductor product developed or manufactured by or for NEC (as defined above).

M8E 00. 4

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC Electronics (Germany) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 02
Fax: 0211-65 03 490

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Italiana s.r.l.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (Germany) GmbH
Benelux Office
Eindhoven, The Netherlands
Tel: 040-2445845
Fax: 040-2444580

NEC Electronics (France) S.A.
Velizy-Villacoublay, France
Tel: 01-30-67 58 00
Fax: 01-30-67 58 99

NEC Electronics (France) S.A.
Spain Office
Madrid, Spain
Tel: 91-504-2787
Fax: 91-504-2860

NEC Electronics (Germany) GmbH
Scandinavia Office
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
United Square, Singapore 1130
Tel: 65-253-8311
Fax: 65-250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377
Fax: 02-2719-5951

NEC do Brasil S.A.
Electron Devices Division
Rodovia Presidente Dutra, Km 214
07210-902-Guarulhos-SP Brasil
Tel: 55-11-6465-6810
Fax: 55-11-6465-6829

J99.1

Major Revisions in This Edition

Page	Description
Throughout	Addition of mPD77110, 77111, 77112, 77113, and 77114 as target devices
p. 78	3.4 Load/Store Instructions Addition of description to Caution 1 of LSPA (parallel load/store instruction)
p. 84	3.4 Load/Store Instructions Addition of description to Caution 1 of LSSE (section load/store instruction)
p. 109	3.8 Hardware Loop Instructions Addition of Caution 2 of REP (repeat instruction)
p. 111	3.8 Hardware Loop Instructions Addition of Caution 2 and change of Caution 3 of LOOP (loop instruction)
p. 113	3.9 Control Instructions Change of description and addition of Caution 3 of STOP (stop instruction)

The mark ★ shows major revised points.

INTRODUCTION

Target Readers: This manual is intended for users who wish to understand the functions of the μPD77016 Family devices and to design and develop software/hardware application systems using these micocontrollers.

Purpose: This manual is intended to give users an understanding of the instruction functions provided in μPD77016 Family devices, and is designed to be used as a reference manual when developing software or hardware application systems using these microcontrollers.

Organization: This manual consists of the following sections:

- CHAPTER 1 OUTLINE
- CHAPTER 2 INSTRUCTION CATEGORIES AND INSTRUCTION FUNCTIONS
- CHAPTER 3 EXPLANATION OF INSTRUCTIONS
- APPENDIX A CLASSIFICATION OF INSTRUCTION WORDS
- APPENDIX B INSTRUCTION SETS
- APPENDIX C INDEX

How to Use This Manual: It is assumed that the reader of this manual has general knowledge in the fields of electrical engineering, logic circuits, and microcomputers.

The μPD77016 Family represents μPD7701x Family devices (μPD77016, 77015, 77017, 77018, 77018A, and 77019) and μPD77111 Family (μPD77110, 77111, 77112, 77113, and 77114). Unless there are differences in function or operation, read the μPD77016 Family as the corresponding products. If there are differences among family products, they are described under their respective names.

Legends:

Data significance:	Higher digits on the left and lower on the right
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remarks:	Supplementary information
Bold text:	Important items
Numerical representation:	Binary ... 0bXXXX
	Decimal ... XXXX
	Hexadecimal ... 0xXXXX
{ }:	Either of the items enclosed within { } can be selected.

Related Documents: The related documents listed below may include preliminary versions. However, preliminary versions are not marked as such.

Documents Related to Devices

Document Name / Product name	Pamphlet	Data Sheet	User's Manual		Application Note
			Architecture	Instructions	Basic software
μPD77016	U12395E	U10891E	U10503E	This document	U11958E
μPD77015		U10902E			
μPD77017					
μPD77018					
μPD77018A		U11849E			
μPD77019					
μPD77019-013		U13053E			
μPD77110		U12801E	U14623E		
μPD77111					
μPD77112					
μPD77113		U14373E			
μPD77114					

Documents Related to Development Tools

Document Name		Document No.
IE-77016-98, IE-77016-PC User's Manual	Hardware	U13044E
IE-11016-CM-LC User's Manual		U14139E
RX77016 User's Manual	Function	U14397E
	Configuration Tool	U14404E
RX77016 Application Note	HOST API	U14371E

Caution The documents listed above are subject to change without notice. Be sure to use the latest documents when designing.

CONTENTS

CHAPTER 1 OUTLINE 13
1.1 Assembly Description 13
1.1.1 Coding instructions 13
1.1.2 Parallel coding 13
1.1.3 Coding trinomial operations 14
1.1.4 Pointer operations 14
1.1.5 Coding conditional instructions 14
1.1.6 Coding hardware loop instructions 14
1.2 Conventions Used for Instruction Descriptions 16
1.2.1 Symbols and corresponding registers 16
1.2.2 General-purpose register partition format 17
1.2.3 Data pointer modification 18
1.3 Description Format 19

CHAPTER 2 INSTRUCTION CATEGORIES AND INSTRUCTION FUNCTIONS 21
2.1 Classification by Instruction Word 21
2.1.1 Classification by instruction word format (category classification) 21
2.1.2 Classification by instruction function (functional classification) 21

CHAPTER 3 EXPLANATION OF INSTRUCTIONS 23
3.1 Trinomial Operation Instructions 23
3.2 Binomial Operation Instructions 36
3.3 Monomial Operation Instructions 57
3.4 Load/Store Instructions 74
3.5 Inter-Register Transfer Instructions 93
3.6 Immediate Value Set Instruction 96
3.7 Branch Instructions 99
3.8 Hardware Loop Instructions 107
3.9 Control Instructions 114

APPENDIX A CLASSIFICATION OF INSTRUCTION WORDS 120
A.1 Three-Operand Instructions 121
A.2 Two-Operand Instructions 123
A.3 Immediate Value Operation Instructions 125
A.4 Load/Store Instructions 126
A.5 Inter-Register Transfer Instruction 129
A.6 Immediate Value Set Instruction 130
A.7 Branch Instructions 131
A.8 Hardware Loop Instructions 133
A.9 CPU Control Instructions 135
A.10 Conditional Instructions 136

APPENDIX B INSTRUCTION SETS 138

APPENDIX C INDEX 144

LIST OF FIGURES

Figure No.	Title	Page
1-1	Partition Formats of General-Purpose Registers	17
A-1	Three-Operand Instruction Format	122
A-2	Two-Operand Instruction Format	124
A-3	Immediate Value Operation Instruction Format	125
A-4	Load/Store Instruction Format	127
A-5	Inter-Register Transfer Instruction Format	129
A-6	Immediate Value Set Instruction Format	130
A-7	Branch Instruction Format	132
A-8	Hardware Loop Instruction Format	134
A-9	CPU Control Instruction Format	135
A-10	Conditional Instruction Format	137

LIST OF TABLES

Table No.	Title	Page
1-1	Symbols and Corresponding Registers	16
1-2	Data Pointer Modify	18
A-1	Formats of Instruction Words	120

CHAPTER 1 OUTLINE

An instruction word in the μPD77016 Family is composed of 32 bits. Each instruction word is partitioned logically, allowing plural functional instructions to be included in a single instruction word. All operation instructions and transfer instructions are executed in one instruction cycle, while loop instructions and branch instructions are executed in two or three instruction cycles. The μPD77016 has the following assembly instruction features.

- 32-bit instruction words
- Nine instruction function groups
- Plural instructions can be described in one instruction word.
- Flexible description of conditional operations by combining independent conditional instructions

1.1 Assembly Description

1.1.1 Coding instructions

The assembly language for the μPD77016 Family uses arithmetic symbols such as +, –, *, /, and =, instead of general mnemonics such as ADD and MOV, for improved program readability.

This assembly language requires a semicolon ";" to indicate the end of each line in the same manner as the C language. Note that this assembler treats all codes as one line until a semicolon, even though one instruction description may fill several lines. A description example is provided below:

Description example:

- R1 = R0; ..... Assigns the R0 register contents to the R1 register.
- R5 = *DP0; ..... Assigns the X memory contents to the R5 register by using the DP0 register as a pointer.
- R4 = R2 + R3; ..... Adds the R2 and R3 register contents, and assigns the result to the R4 register.

1.1.2 Parallel coding

To describe parallel operations of the μPD77016 Family, up to three instructions can be described in parallel: an operation instruction (trinomial, binomial, monomial operation instructions) and two transfer instructions for data movements via X and Y buses.

Description example:

R1 = R0 R2 = *DP0 *DP4 = R3H;

..... Assigns the R0 register contents to the R1 register, then assigns the X memory contents to the R2 register using the DP0 register as a pointer, and assigns the R3H register contents to the Y memory using the DP4 register as a pointer.

Remark In the example above, the μPD77016 Family first executes the operational instruction, then executes the transfer instructions.

1.1.3 Coding trinomial operations

The μPD77016 Family supports trinomial instructions that use three operands to describe the operation of the incorporated multiplier (the result obtained from multiplication by hardware is added to another register).

Description example:

R0 = R0 + R1L * R2L;

..... Multiplies the R1L register and the R2L register contents, then adds the result to the R0 register contents, and stores the sum in the R0 register.

1.1.4 Pointer operations

The μPD77016 Family is provided with instructions that add and subtract the pointer value after executing the transfer instructions explained in **1.1.2** and **1.1.3** above.

For details on pointer operations, refer to **μPD7701x Family User's Manual Architecture** or **μPD77111 Family User's Manual Architecture**.

Description example:

R1 = *DP0++ R2 = *DP4##;

..... Executes transfer instructions using the DP0 and DP4 registers as pointers, adds 1 to the DP0 contents, and adds the DN4 (supplement register) value to the DP4 contents.

1.1.5 Coding conditional instructions

The μPD77016 Family is provided with conditional instructions that can use a register value as the condition. These instructions are executed only when the register value matches the specified condition. By using these instructions, the number of branch (conditional) instructions required in conditional operation can be reduced. For details on the conditional instructions, refer to section **3.9 Conditional instruction of control instruction (COND)**.

Description example:

if (R0 < 0) R1L + = R2L;

..... Adds the R2 register contents to the R1 register contents when the R0 register value is negative.

1.1.6 Coding hardware loop instructions

The μPD77016 Family is provided with hardware loop instructions, which enable instructions consisting of 1 to 255 lines to be repeatedly executed 1 to 32767 times. To describe this operation, either the REPEAT or LOOP instruction is used depending on the number of repeated instruction lines. Nesting of loops is allowed (up to four levels).

Description example:

- **To execute one line of instruction two or more times:**

```
REPEAT  32
        R0 = R0 + R1L * R2L     R1 = *DP0++         R2 = *DP4++;
```

- **To execute 2 to 255 lines of instructions two or more times:**

```
LOOP    R1L {
        R0 = R0 + R1L * R2L;
        •
        •
        •
        R4 = R4 – 1;
};
```

- **To execute a loop within a loop function (nesting of loops):**

```
LOOP    64 {
        R0 = R0 + R1L * R2L;
                •
                •
                •
        LOOP    R1L  {
                R3 = R0 + R4;
                •
                •
                •
        };
        R4 = R4 – 1;
        R5 = R5 + 1;
        NOP;
};
```

1.2 Conventions Used for Instruction Descriptions

1.2.1 Symbols and corresponding registers

In the following sections, registers are described in a generalized style using symbols. The relationship between the symbols and the corresponding registers is shown in Table 1-1.

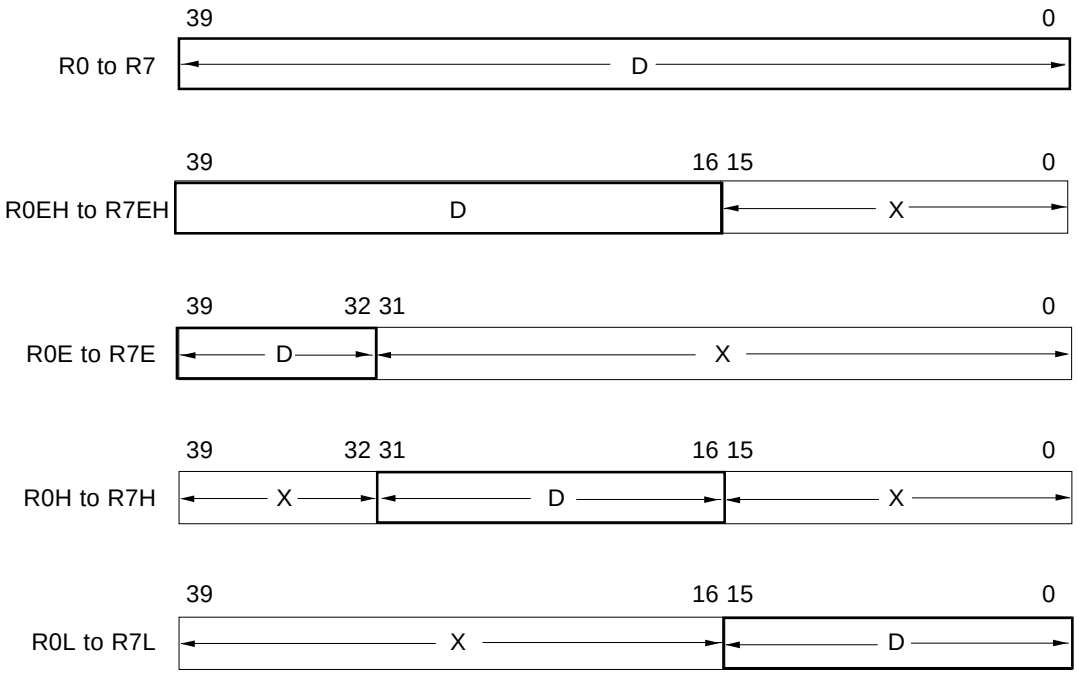
Table 1-1. Symbols and Corresponding Registers

Symbols	Corresponding Registers
ro, ro' ro''	R0 to R7
rl, rl'	R0L to R7L
rh, rh'	R0H to R7H
re	R0E to R7E
reh	R0EH to R7EH
dp	DP0 to DP7
dn	DN0 to DN7
dm	DMX and DMY
dpx	DP0 to DP3
dpy	DP4 to DP7
dpx_mod	DPn, DPn++, DPn– –, DPn##, DPn%%, !DPn## (n = 0 to 3)
dpy_mod	DPn, DPn++, DPn– –, DPn##, DPn%%, !DPn## (n = 4 to 7)
dp_imm	DPn##imm (n = 0 to 7)
* xxx	Memory contents with address xxx **Example** * DP0 indicates the contents of address 1000 when the contents of the DP0 register is 1000.

1.2.2 General-purpose register partition format

General-purpose registers can be partially accessed when an operation or transfer is executed. The partial access methods are classified into the five formats shown in Figure 1-1.

Figure 1-1. Partition Formats of General-Purpose Registers



Remark D: Numeral
X: Invalid

1.2.3 Data pointer modification

If indirect addressing using a pointer (DPn) is executed, the data pointer is, in most cases, modified after memory access. Table 1-2 shows the relationship between the descriptions of data pointer modify and the corresponding operations.

Table 1-2. Data Pointer Modify

Description	Operation
DPn	No operation (The value of DPn is not changed.)
DPn++	DPn ← DPn + 1
DPn– –	DPn ← DPn – 1
DPn##	DPn ← DPn + DNn (Adds DP0 through DP7 values to DN0 through DN7 values, respectively.) **Example:** DP0 ← DP0 + DN0
DPn%%	(n = 0 to 3) DPn = ((DP$_L$ + DNn) mod (DMX + 1)) + DP$_H$
	(n = 4 to 7) DPn = ((DP$_L$ + DNn) mod (DMY + 1)) + DP$_H$
!DPn##	Accesses memory after bit reverse for DPn. After memory access, DPn ← DPn + DNn
DPn##imm	DPn ← DPn + imm

1.3 Description Format

Instruction symbol

Instruction type

MADD MADD

Trinomial operation

ro = ro + rh * rh'

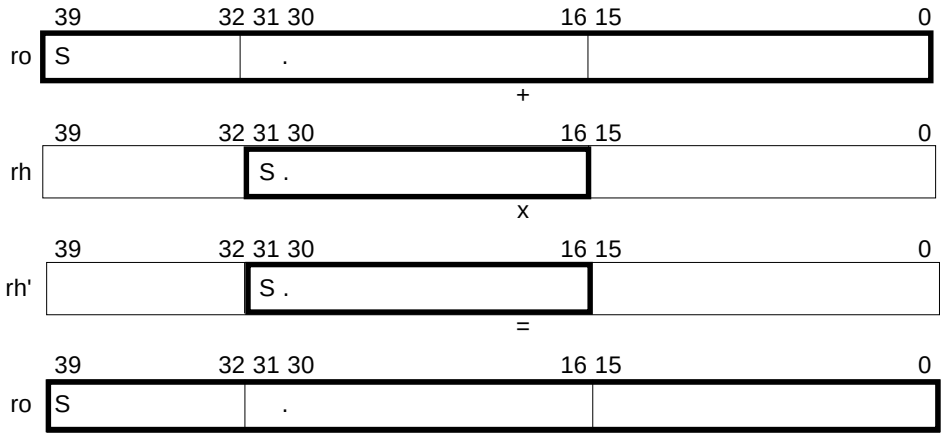
Name of instruction: Multiply add

Mnemonic: ro = ro + rh * rh'

Example R1 = R1 + R0H * R4H

Explanation ← ***Details on instruction operations are described. Operation target is marked with solid line.***

Instruction to add the product of 16-bit data x 16-bit data to 40-bit data.



Execution cycle 1 ← ***Number of instruction execution cycles***

Instruction that can be described concurrently ← ***Instructions marked with "Yes" can be described with this instruction.***

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

Caution ← ***Description that should be read carefully.***

[MEMO]

CHAPTER 2 INSTRUCTION CATEGORIES AND INSTRUCTION FUNCTIONS

An instruction is composed of 32 bits which are partitioned into several fields to concurrently manage various resources in the device. Instructions can be classified into various formats according to the field partition methods. These formats are further classified by function group into several categories to facilitate the use of instructions.

2.1 Classification by Instruction Word

This section describes two instruction classifications: category and functional classifications.

2.1.1 Classification by instruction word format (category classification)

Instructions can be classified by the format in which the instruction words are partitioned into fields. Such classification is called category classification. Under this classification, categories are mutually exclusive. In other words, an instruction which is described in a particular format cannot be used in any other format. If the entire program is described according to this classification, the programmer can easily trace which functions are concurrently performed by a one-word instruction execution. This also allows users to create bit patterns of instruction words by synthesizing element fields, and to analyze concurrently performed functions based on the bit pattern of an instruction word.

Although this classification is closely related to the architecture of language processors including assemblers, programmers do not necessarily have to be aware of these formats. The details of the category classification are provided in Appendix A.

2.1.2 Classification by instruction function (functional classification)

Classifying instructions according to their functions is called functional classification. This classification helps programmers understand overall instructions and is convenient when creating application programs. In this manual, instructions are classified into the following nine functional groups:

- Trinomial operation
- Binomial operation
- Monomial operation
- Load/store
- Inter-register transfer
- Immediate value set
- Branch
- Hardware loop
- Control

Remark: The classification indicated in 2.1.2 above does not imply that each instruction is exclusive within one word; plural instructions may be concurrently described in the same instruction word. For details, refer to **CHAPTER 3 EXPLANATION OF INSTRUCTIONS**.

[MEMO]

CHAPTER 3 EXPLANATION OF INSTRUCTIONS

This section describes instructions of the μPD77016 Family based on the functional classification. The following nine types of instructions are provided:

3.1 Trinomial Operation Instructions
3.2 Binomial Operation Instructions
3.3 Monomial Operation Instructions
3.4 Load/Store Instructions
3.5 Inter-Register Transfer Instruction
3.6 Immediate Value Set Instruction
3.7 Branch Instructions
3.8 Hardware Loop Instructions
3.9 Control Instructions

3.1 Trinomial Operation Instructions

Instruction for specifying operations with the multiply-accumulator. Any three registers can be specified for the operands (inputs) from the general-purpose register file, and any one of the registers specified for the operands can be specified for the output destination.

The following trinomial operation instructions are provided (symbol in parentheses is an abbreviation of each instruction).

Multiply add	(MADD)
Multiply sub	(MSUB)
Sign unsign multiply add	(SUMA)
Unsign unsign multiply add	(UUMA)
1-bit shift multiply add	(MAS1)
16-bit shift multiply add	(MAS16)

MADD

MADD

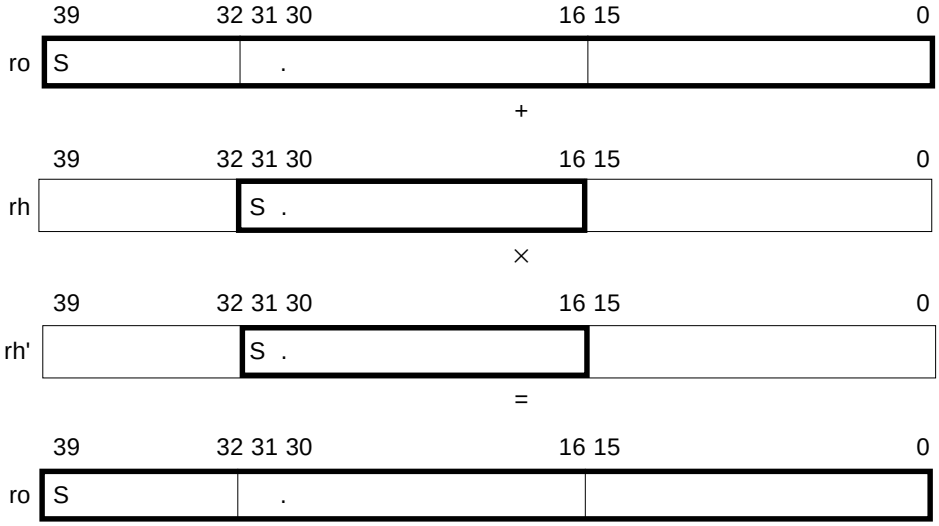
Trinomial operation

ro = ro + rh * rh'

Name of instruction:	**Multiply add**
Mnemonic:	**ro = ro + rh * rh'**

Example R1 = R1 + R0H * R4H

Explanation Instruction to add the product of 16-bit data x 16-bit data to 40-bit data.
The product of the value of bits 31 to 16 of the general-purpose register specified by rh multiplied by the value of bits 31 to 16 of the general-purpose register specified by rh' is added to the value of bits 39 to 0 of the general-purpose register specified by ro. The sum is stored in bits 39 to 0 of the general-purpose register specified by ro. The 16-bit values specified by rh and rh' and the 40-bit value specified by ro are data formats represented with two's complement.



On the assumption that a decimal point is located between bit 31 and bit 30 of the value of the general-purpose registers specified by rh and rh', the product to be added to the value of the general-purpose register specified by ro is a value with a decimal point located between bit 31 and bit 30, sign-extended to bits 39 to 32 and with bit 0 set to 0.

MADD

MADD

Trinomial operation

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

Caution **If the addition results in an overflow, the overflow flag ovf in the error status register ESR is set to 1.**

MSUB MSUB

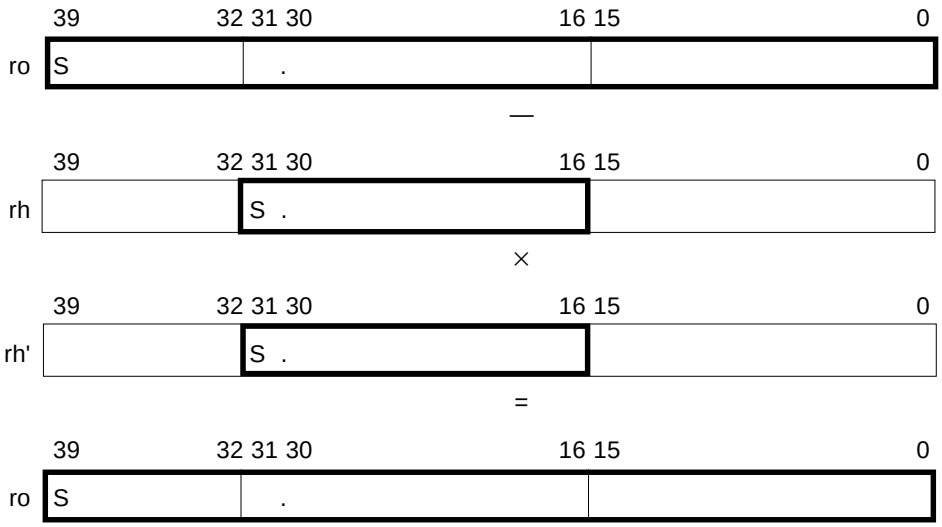
Trinomial operation

ro = ro – rh * rh'

Name of instruction:	**Multiply sub**
Mnemonic:	**ro = ro – rh * rh'**

Example R1 = R1 – R0H * R4H

Explanation Instruction to subtract the product of 16-bit data x 16-bit data from 40-bit data.
The product of bits 31 to 16 of register rh multiplied by bits 31 to 16 of register rh' is subtracted from bits 39 to 0 of register ro. The 16-bit values specified by rh and rh' and the 40-bit value specified by ro are data formats represented with two's complement.



On the assumption that a decimal point is located between bit 31 and bit 30 of the value of the general-purpose registers specified by rh and rh', the product to be added to the value of the general-purpose register specified by ro is a value with a decimal point located between bit 31 and bit 30, sign-extended to bits 39 to 32 and with bit 0 set to 0.

MSUB

MSUB

Trinomial operation

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

Caution **If the subtraction results in an overflow, the overflow flag ovf in the error status register ESR is set to 1.**

SUMA SUMA

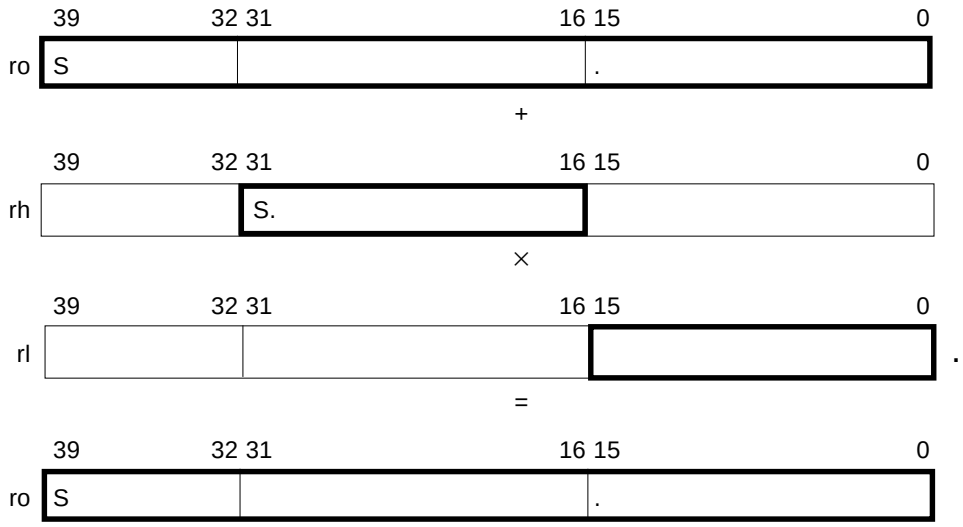
Trinomial operation

ro = ro + rh * rl

Name of instruction:	**Sign unsign multiply add**
Mnemonic:	**ro = ro + rh * rl**

Example R1 = R1 + R0H * R4L

Explanation Instruction to add the product of 16-bit data x 16-bit data to 40-bit data.
The product of the value of bits 31 to 16 of the general-purpose register specified by rh multiplied by the value of bits 15 to 0 of the general-purpose register specified by rl is added to the value of bits 39 to 0 of the general-purpose register specified by ro. The sum is stored in bits 39 to 0 of the general-purpose register specified by ro. The 16-bit value specified by rh and the 40-bit value specified by ro are represented with two's complement, and the 16-bit value specified by rl is a data format of a positive integer value.



On the assumption that a decimal point is located between bit 31 and bit 30 of the value of the general-purpose register specified by rh and another decimal point is located below bit 0 of the value of the general-purpose register specified by rl, the product to be added to the value of the general-purpose register specified by ro is a value with a decimal point located between bit 16 and bit 15, sign-extended to bits 39 to 32 and with bit 0 set to 0.

SUMA

SUMA

Trinomial operation

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

Caution **If the addition results in an overflow, the overflow flag ovf in the error status register ESR is set to 1.**

UUMA UUMA

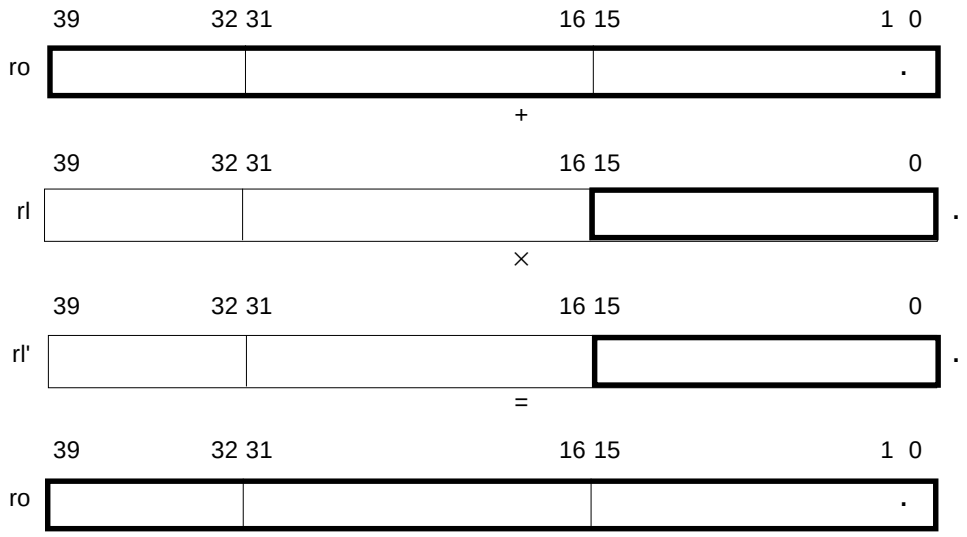
Trinomial operation

ro = ro + rl * rl'

Name of instruction:	**Unsign unsign multiply add**
Mnemonic:	**ro = ro + rl * rl'**

Example R1 = R1 + R0L * R4L

Explanation Instruction to add the product of 16-bit data x 16-bit data to 40-bit data.
The product of the value of bits 15 to 0 of the general-purpose register specified by rl multiplied by the value of bits 15 to 0 of the general-purpose register specified by rl' is added to the value of bits 39 to 0 of the general-purpose register specified by ro. The sum is stored into the general-purpose register specified by ro. The 16-bit values specified by rl and rl' are positive integer values, and the 40-bit value specified by ro is a data format represented with two's complement.



On the assumption that a decimal point is located below bit 0 of the values of the general-purpose registers specified by rl and rl', the product to be added to the value of the general-purpose register specified by ro is a value with a decimal point located between bit 1 and bit 0 with bits 39 to 32 and bit 0 set to 0.

UUMA

UUMA

Trinomial operation

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

Cautions

1. **The result is treated as two's complement data.**
2. **If the addition results in an overflow, the overflow flag ovf in the error status register ESR is set to 1.**

MAS1 MAS1

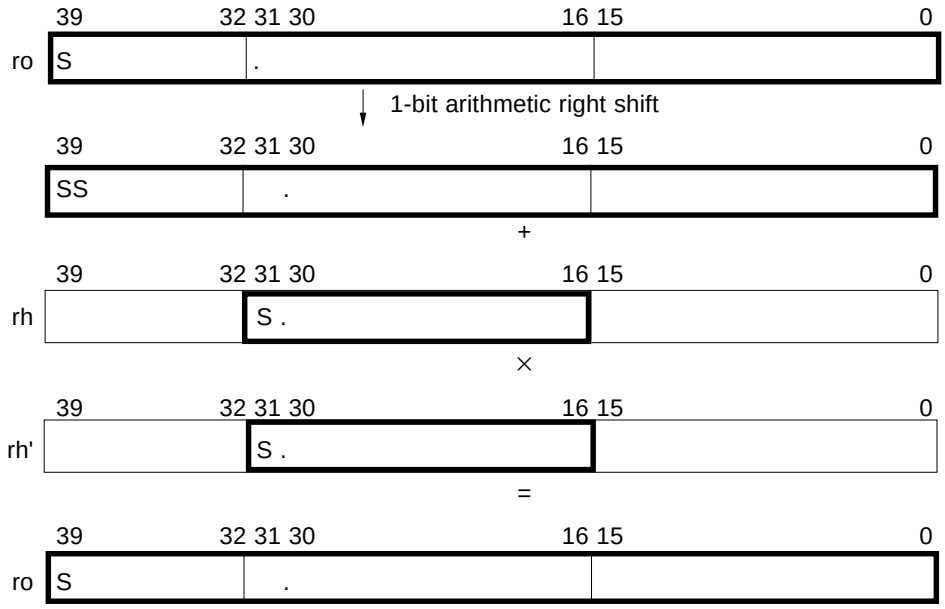
Trinomial operation

ro = (ro >> 1) + rh * rh'

Name of instruction:	**1-bit shift multiply add**
Mnemonic:	**ro = (ro >> 1) + rh * rh'**

Example R1 = (R1 >> 1) + R0H * R4H

Explanation Instruction to add the product of 16-bit data x 16-bit data to 40-bit data.
The product of the value of bits 31 to 16 of the general-purpose register specified by rh multiplied by the value of bit 31 to 16 of the general-purpose register specified by rh' is added to the value of bits 39 to 0 of the general-purpose register specified by ro arithmetically shifted to the right by one bit. The sum is stored in bits 39 to 0 of the general-purpose register specified by ro. The 16-bit values specified by rh and rh' and the 40-bit value specified by ro are data formats represented with two's complement.



On the assumption that a decimal point is located between bit 31 and bit 30 of the value of the general-purpose registers specified by rh and rh', the product to be added to the value of the general-purpose register specified by ro is a value with a decimal point located between bit 31 and bit 30, sign-extended to bits 39 to 32 and with bit 0 set to 0.

Remark This function is efficient for executing FFT at high speed.

MAS1

MAS1

Trinomial operation

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

Caution **If an arithmetic right shift results in an underflow, the value is not corrected.**
If 0xFF'FFFF'FFFF is arithmetically shifted to the right in the shift process, the value remains 0xFF'FFFF'FFFF (it is not set to 0x00'0000'0000).

MAS16 MAS16

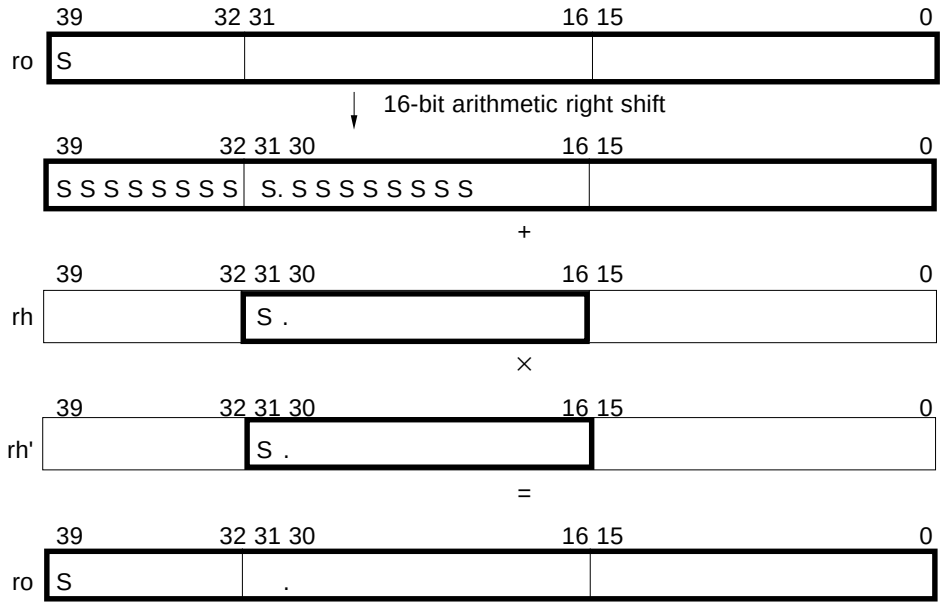
Trinomial operation

ro = (ro >> 16) + rh * rh'

Name of instruction:	**16-bit shift multiply add**
Mnemonic:	**ro = (ro >> 16) + rh * rh'**

Example R1 = (R1 >> 16) + R0H * R4H

Explanation Instruction to add the product of 16-bit data x 16-bit data to 40-bit data.
The product of the value of bits 31 to 16 of the general-purpose register specified by rh multiplied by the value of bits 31 to 16 of the general-purpose register specified by rh' is added to the value of bits 39 to 0 of the general-purpose register specified by ro arithmetically shifted to the right by 16 bits. The sum is stored in bits 39 to 0 of the general-purpose register specified by ro. The 16-bit values specified by rh and rh' and the 40-bit value specified by ro are data formats represented with two's complement.



On the assumption that a decimal point is located between bit 31 and bit 30 of the value of the specified general-purpose registers specified by rh and rh', the product to be added to the value of the general-purpose register specified by ro is a value with a decimal point located between bit 31 and bit 30, sign-extended to bits 39 to 32 and with bit 0 set to 0.

Remark This function is intended for high-speed double precision multiplication.

MAS16

MAS16

Trinomial operation

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

Caution **If an arithmetic right shift results in an underflow, the value is not corrected.**
If 0xFF'FFFF'FFFF is arithmetically shifted to the right in the shift process, the value remains 0xFF'FFFF'FFFF (it is not set to 0x00'0000'0000).

3.2 Binomial Operation Instructions

Instructions for specifying operations using a multiply-accumulator, ALU, or the barrel shifter. Any two registers can be specified for the operand (inputs) from the general register file. Also available are instructions to specify the immediate value for one input instead of the general-purpose registers. Any one register can be specified for the output destination from the general-purpose register file.

The following binomial operation instructions are provided:

Multiply	(MPY)
Add	(ADD)
Immediate add	(IADD)
Sub	(SUB)
Immediate sub	(ISUB)
Arithmetic right shift	(SRA)
Immediate arithmetic right shift	(ISRA)
Logical right shift	(SRL)
Immediate logical right shift	(ISRL)
Logical left shift	(SLL)
Immediate logical left shift	(ISLL)
AND	(AND)
Immediate AND	(IAND)
OR	(OR)
Immediate OR	(IOR)
Exclusive OR	(XOR)
Immediate exclusive OR	(IXOR)
Less than	(LT)

MPY

MPY

Binomial operation

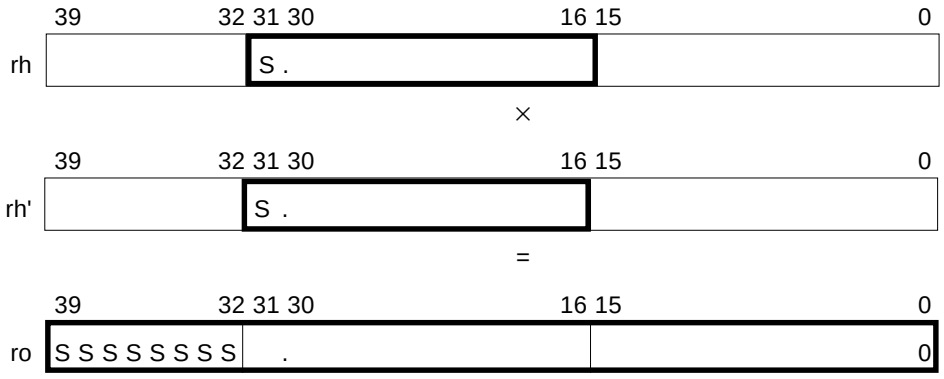
ro = rh * rh'

Name of instruction:	**Multiply**
Mnemonic:	**ro = rh * rh'**

Example R1 = R0H * R4H

Explanation Instruction to multiply two 16-bit data.

The value of bits 31 to 16 of the general-purpose register specified by rh is multiplied by the value of bits 31 to 16 of the general-purpose register specified by rh'. The product is stored in bits 39 to 0 of the general-purpose register specified by ro. The 16-bit values specified by rh and rh' and the 40-bit value specified by ro are data formats represented with two's complement.



On the assumption that a decimal point is located between bit 31 and bit 30 of the value of the general-purpose registers specified by rh and rh', the value to be stored to the general-purpose register specified by ro is a value with a decimal point located between bit 31 and bit 30, sign-extended to bits 39 to 32 and with bit 0 set to 0.

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

Caution **In the following case, bit 31 of the operation result is not a sign.**
0x8000 x 0x8000 = 0x00'8000'0000

ADD

ADD

Binomial operation

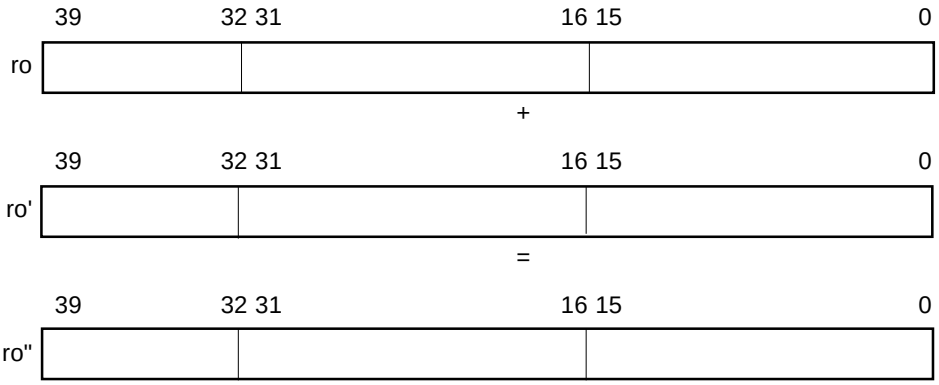
ro'' = ro + ro'

Name of instruction: **Add**

Mnemonic: **ro'' = ro + ro'**

Example R1 = R0 + R4

Explanation Instruction to add two 40-bit data. The value of bits 39 to 0 of the general-purpose register specified by ro is added to that of bits 39 to 0 of the general-purpose register specified by ro'. The sum is stored in bits 39 to 0 of the general-purpose register specified by ro''. The 40-bit values specified by ro, ro', and ro'' are data formats represented with two's complement.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

Caution **If the addition results in an overflow, the overflow flag ovf in the error status register ESR is set to 1, but the value is not corrected.**

IADD

IADD

Binomial operation

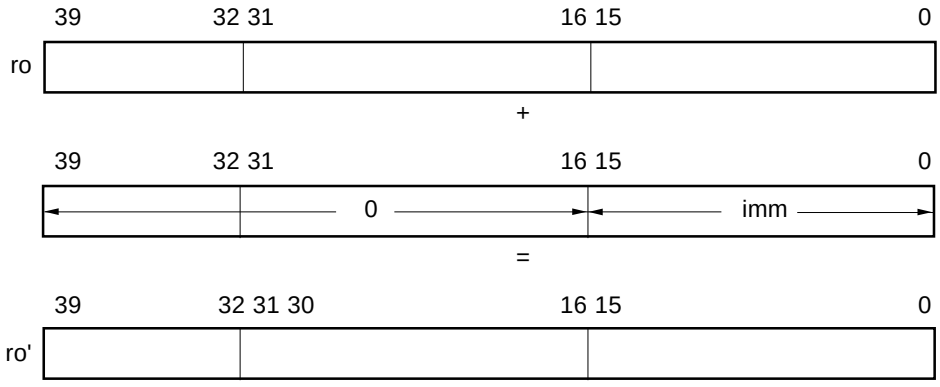
ro' = ro + imm

Name of instruction: **Immediate add**

Mnemonic: **ro' = ro + imm (imm = 0 to 0xFFFF (0 to 65535))**

Example R1 = R0 + 0x10

Explanation Instruction to add two 40-bit data.
The 40-bit immediate value, zero-extended to bits 39 to 16 by specifying the values of bits 15 to 0 using imm, is added to the value of bits 39 to 0 of the general-purpose register specified by ro. The sum is stored in bits 39 to 0 of the general-purpose register specified by ro'. The 40-bit values specified by ro and ro' are data formats represented with two's complement.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

Caution **If the addition results in an overflow, the overflow flag ovf in the error status register ESR is set to 1.**

SUB

SUB

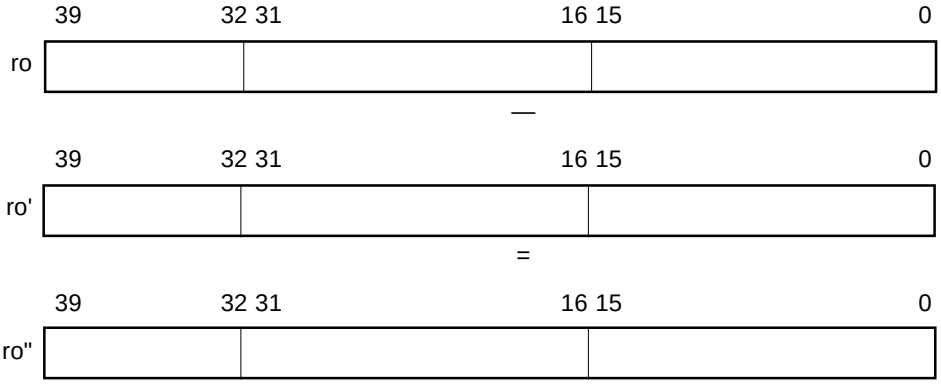
Binomial operation

ro'' = ro – ro'

Name of instruction:	**Sub**
Mnemonic:	**ro'' = ro – ro'**

Example R1 = R0 – R4

Explanation Instruction to subtract 40-bit data from 40-bit data.
The value of bits 39 to 0 of the general-purpose register specified by ro' is subtracted from the value of bits 39 to 0 of the general-purpose register specified by ro. The difference is stored in bits 39 to 0 of the general-purpose register specified by ro''. The 40-bit values specified by ro, ro', and ro'' are data formats represented with two's complement.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

Caution **If the subtraction results in an overflow, the overflow flag ovf in the error status register ESR is set to 1.**

ISUB

ISUB

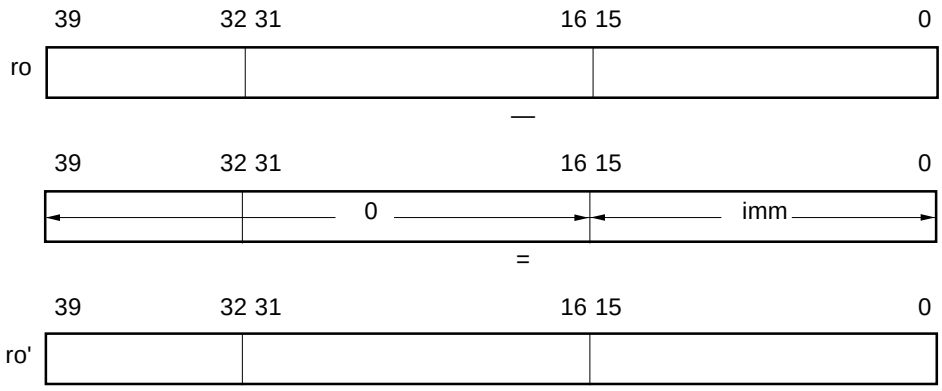
Binomial operation

ro' = ro – imm

Name of instruction:	**Immediate sub**
Mnemonic:	**ro' = ro – imm (imm = 0 to 0xFFFF (0 to 65535))**

Example R1 = R0 – 0x10

Explanation Instruction to subtract 40-bit data from 40-bit data.
The 40-bit immediate value, zero-extended to bits 39 to 16 by setting the value of bits 15 to 0 with imm, is subtracted from the value of bits 39 to 0 of the general-purpose register specified by ro. The difference is stored in bits 39 to 0 of the general-purpose register specified by ro'. The 40-bit values specified by ro and ro' are data formats represented with two's complement.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

Caution **If the subtraction results in an overflow, the overflow flag ovf in the error status register ESR is set to 1.**

SRA SRA

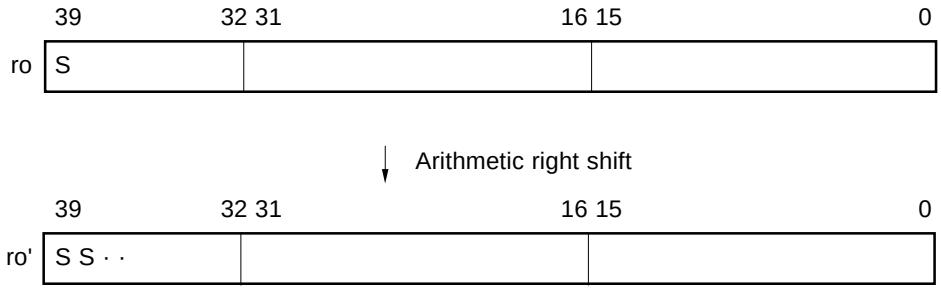
Binomial operation

ro' = ro SRA rl

Name of instruction:	**Arithmetic right shift**
Mnemonic:	**ro' = ro SRA rl**

Example R1 = R0 SRA R4L

Explanation Instruction to arithmetically shift 40-bit data to the right.
The value of bits 39 to 0 of the general-purpose register specified by ro is arithmetically shifted to the right by the value of bits 5 to 0 of the general-purpose register specified by rl. The result is stored in bits 39 to 0 of the general-purpose register specified by ro'. The values of 40 bits of the general-purpose registers specified by ro and ro' are represented with two's complement and the value of 6 bits of the general-purpose register specified by rl is a data format of a positive integer value.



An arithmetic right shift is a shift accompanied by sign expansion. Bits 39 to 6 of the general-purpose register specified by rl are ignored.

SRA

SRA

Binomial operation

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

Cautions

1. **If an arithmetic right shift results in an underflow, the value is not corrected.**
   For example, if 0xFF'FFFF'FFFF is arithmetically shifted to the right, the value remains 0xFF'FFFF'FFFF (it is not set to 0x00'0000'0000).
2. **If a 40-bit or higher bits are arithmetically shifted to the right, the data is sign-extended to bits 39 to 0.**

ISRA ISRA

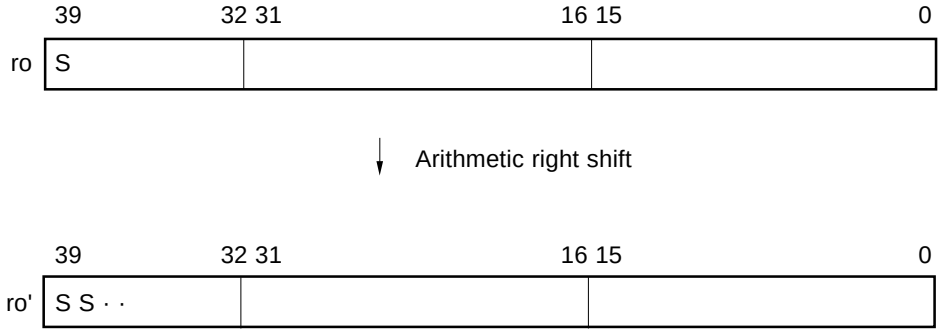
Binomial operation

ro' = ro SRA imm

Name of instruction:	**Immediate arithmetic right shift**
Mnemonic:	**ro' = ro SRA imm (imm = 0 to 0x27 (0 to 39))**

Example R1 = R0 SRA 0x10

Explanation Instruction to arithmetically shift 40-bit data to the right.
The value of bits 39 to 0 of the general-purpose register specified by ro is arithmetically shifted to the right by the 6-bit immediate value. An arithmetic right shift is a right shift accompanied by sign expansion.
Bits 5 to 0 of the immediate value are valid for shift and bits 15 to 6 are ignored.
The result is stored in bits 39 to 0 of the general-purpose register specified by ro'. The values of 40 bits in the general-purpose registers specified by ro and ro' are represented with two's complement and the 6-bit immediate value is a data format of a positive integer value.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

Caution **If an arithmetic right shift results in an underflow, the value is not corrected.**
For example, if 0xFF'FFFF'FFFF is arithmetically shifted to the right, the value remains 0xFF'FFFF'FFFF (it is not set to 0x00'0000'0000).

SRL

SRL

Binomial operation

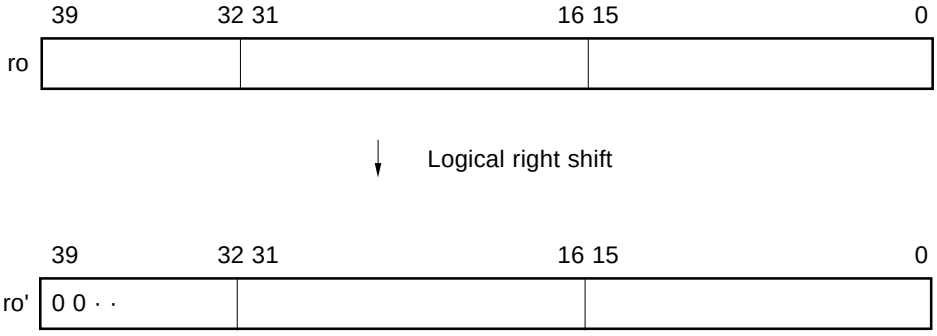
ro' = ro SRL rl

Name of instruction:	**Logical right shift**
Mnemonic:	**ro' = ro SRL rl**

Example R1 = R0 SRL R4L

Explanation Instruction to logically shift 40-bit data to the right.

The value of bits 39 to 0 of the general-purpose register specified by ro is logically shifted to the right by the value of bits 5 to 0 of the general-purpose register specified by rl. A logical right shift is a shift to insert 0 from the MSB by the shift amount. Bits 39 to 6 of the general-purpose register specified by rl are ignored. The result is stored in bits 39 to 0 of the general-purpose register specified by ro'. The values of 40 bits in the general-purpose registers specified by ro and ro' are logical values and the value of 6 bits of the general-purpose register specified by rl is a data format of a positive integer value.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

Caution **If a 40-bit or higher logical right shift is specified, 0x00’0000’0000 is set.**

ISRL ISRL

Binomial operation

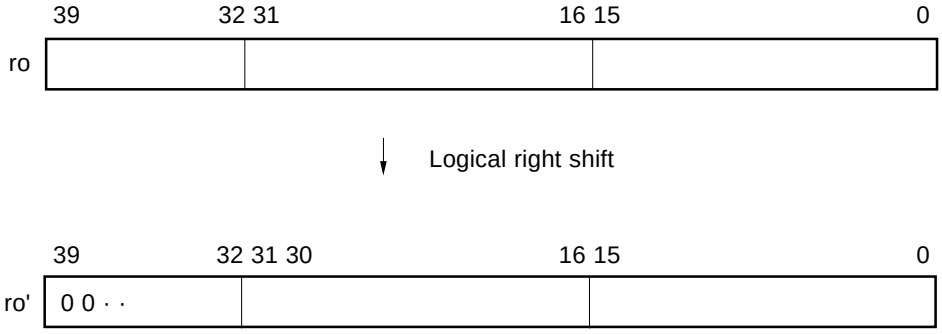
ro' = ro SRL imm

Name of instruction: **Immediate logical right shift**

Mnemonic: **ro' = ro SRL imm (imm = 0 to 0x27 (0 to 39))**

Example R1 = R0 SRL 0x10

Explanation Instruction to logically shift 40-bit data to the right.
The value of bits 39 to 0 of the general-purpose register specified by ro is logically shifted to the right by the 6-bit immediate value. A logical right shift is a right shift to insert 0 from the MSB by the shift amount.
Bits 5 to 0 of the immediate value are valid for shift and bits 15 to 6 are ignored. The result is stored in bits 39 to 0 of the general-purpose register specified by ro'. The values of 40 bits of the general-purpose registers specified by ro and ro' are logical values and the 6-bit immediate value is a data format of a positive integer value.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

SLL

SLL

Binomial operation

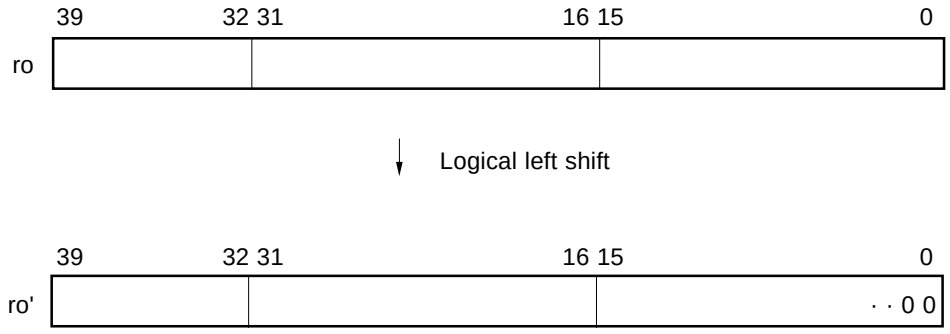
ro' = ro SLL rl

Name of instruction:	**Logical left shift**
Mnemonic:	**ro' = ro SLL rl**

Example R1 = R0 SLL R4L

Explanation Instruction to logically shift 40-bit data to the left.

The value of bits 39 to 0 of the general-purpose register specified by ro is logically shifted to the left by the value of bits 5 to 0 of the general-purpose register specified by rl. A logical left shift is a left shift to insert 0 from the LSB by the shift amount. Bits 39 to 6 of the general-purpose register specified by rl are ignored. The result is stored in bits 39 to 0 of the general-purpose register specified by ro'. The values of 40 bits of the general-purpose registers specified by ro and ro' are logical values and the value of 6 bits of the general-purpose register specified by rl is a data format of a positive integer value.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

Caution

1. **If a 40-bit or higher logical left shift is specified, 0x00’0000’0000 is set.**
2. **A logical left shift does not generate an overflow.**

ISLL ISLL

Binomial operation

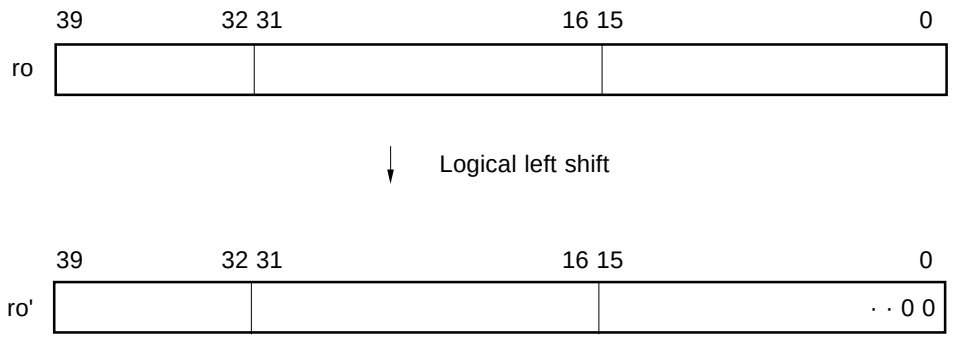
ro' = ro SLL imm

Name of instruction: **Immediate logical left shift**

Mnemonic: **ro' = ro SLL imm (0 to 0x27 (0 to 39))**

Example R1 = R0 SLL 0x10

Explanation Instruction to logically shift 40-bit data to the left.
The value of bits 39 to 0 of the general-purpose register specified by ro is logically shifted to the left by the 6-bit immediate value. A logical left shift is a left shift to insert 0 from the LSB by the shift amount.
Bits 5 to 0 of the immediate value are valid for shift and bits 15 to 6 are ignored. The result is stored in bits 39 to 0 of the general-purpose register specified by ro'. The values of 40 bits of the general-purpose registers specified by ro and ro' are logical values and the 6-bit immediate value is a data format of a positive integer value.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

Caution **A logical left shift does not generate an overflow.**

AND

AND

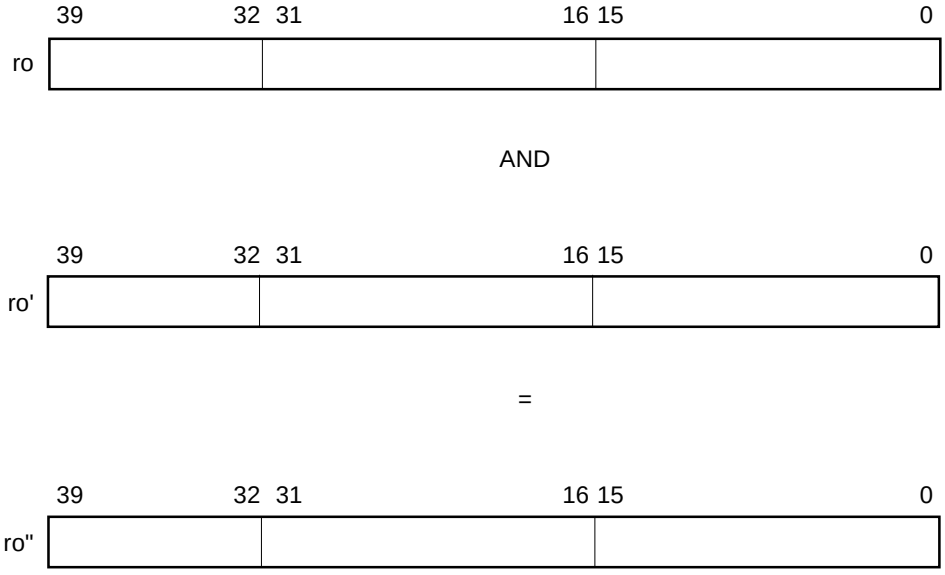
Binomial operation

ro" = ro & ro'

Name of instruction:	**AND**
Mnemonic:	**ro" = ro & ro'**

Example R1 = R0 & R4

Explanation Instruction to obtain a logical AND of 40-bit data.
The logical AND is obtained from the value of bits 39 to 0 of the general-purpose register specified by ro and the value of bits 39 to 0 of the general-purpose register specified by ro'. The result is stored in bits 39 to 0 of the general-purpose register specified by ro". The values of 40 bits of the general-purpose registers specified by ro, ro', and ro" are logical values.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

IAND

IAND

Binomial operation

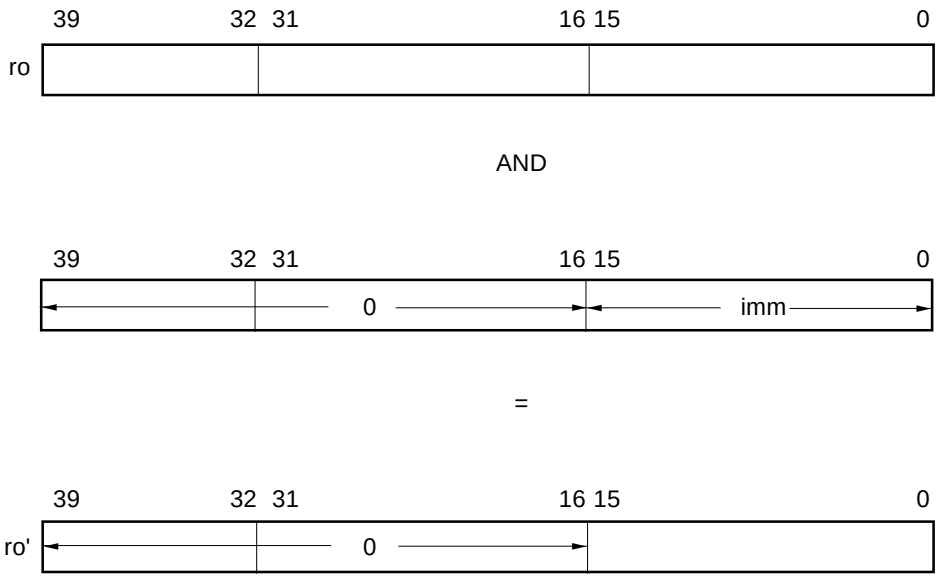
ro' = ro & imm

Name of instruction: **Immediate AND**

Mnemonic: **ro' = ro & imm (imm = 0 to 0xFFFF (0 to 65535))**

Example R1 = R0 & 0x10

Explanation Instruction to obtain a logical AND of 40-bit data.
The logical AND is obtained from the value of bits 39 to 0 of the general-purpose register specified by ro and the 40-bit immediate value, zero-extended to bits 39 to 16 by setting the value of bits 15 to 0 with an instruction. The result is stored in bits 39 to 0 of the general-purpose register specified by ro'. Bits 39 to 16 of the general-purpose register specified by ro' are set to 0. The value of the 40 bits specified by ro and ro' and the 16-bit immediate value are logical values.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

OR

OR

Binomial operation

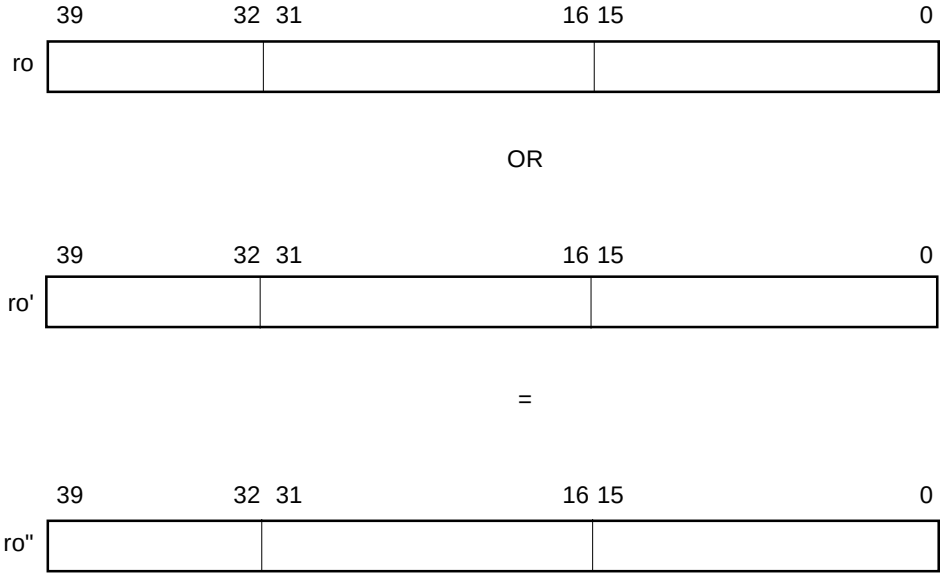
ro" = ro | ro'

Name of instruction: **OR**

Mnemonic: **ro" = ro | ro'**

Example R1 = R0 | R4

Explanation Instruction to obtain a logical inclusive OR of 40-bit data.
The logical inclusive OR is obtained from the value of bits 39 to 0 of the general-purpose register specified by ro and the value of bits 39 to 0 of the general-purpose register specified by ro'. The result is stored in bits 39 to 0 of the general-purpose register specified by ro". The values of 40 bits of the ro, ro', and ro" specified general-purpose registers are logical values.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

IOR IOR

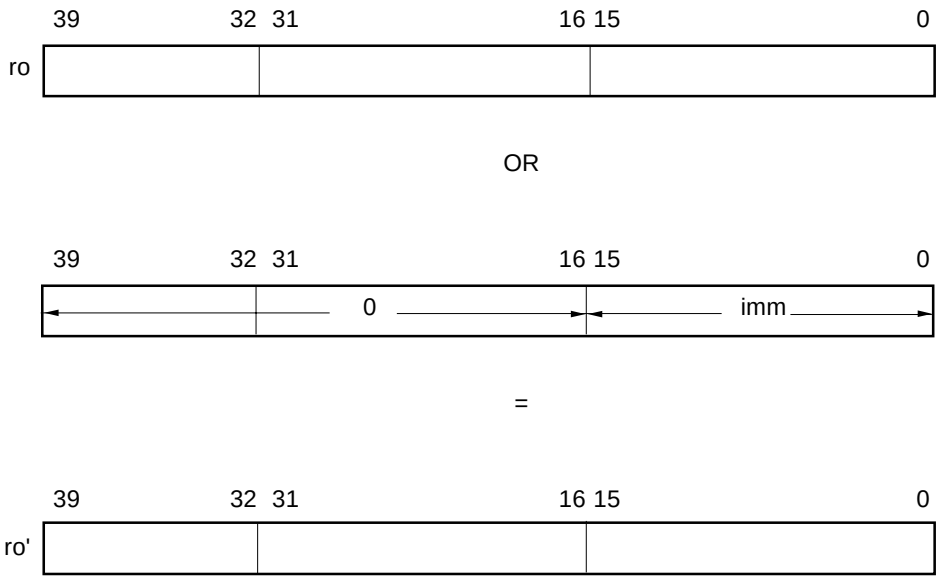
Binomial operation

ro' = ro | imm

Name of instruction:	**Immediate OR**
Mnemonic:	**ro' = ro	imm (imm = 0 to 0xFFFF (0 to 65535))**

Example R1 = R0 | 0x10

Explanation Instruction to obtain a logical inclusive OR of 40-bit data.
The logical inclusive OR is obtained from the value of bits 39 to 0 of the general-purpose register specified by ro and the 40-bit immediate value, zero-extended to bits 39 to 16 by setting the value of bits 15 to 0 with an instruction. The result is stored in bits 39 to 0 of the general-purpose register specified by ro'. Bits 39 to 16 of the general-purpose register specified by ro' become the same as those of the general-purpose register specified by ro. The values of the 40 bits specified by ro and ro' and the 16-bit immediate value are logical values.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

XOR

XOR

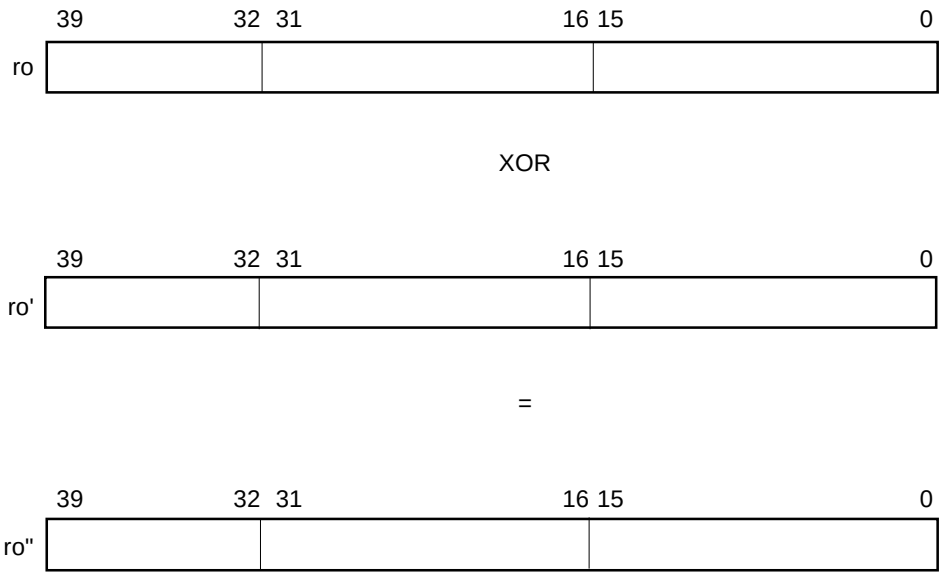
Binomial operation

ro" = ro /\ ro'

Name of instruction:	**Exclusive OR**
Mnemonic:	**ro" = ro /\ ro'**

Example R1 = R0 /\ R4

Explanation Instruction to obtain a logical exclusive OR of 40-bit data.
The logical exclusive OR is obtained from the value of bits 39 to 0 of the general-purpose register specified by ro and the value of bits 39 to 0 of the general-purpose register specified by ro'. The result is stored in bits 39 to 0 of the general-purpose register specified by ro". The values of 40 bits of the general-purpose registers specified by ro, ro', and ro" are logical values.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

IXOR IXOR

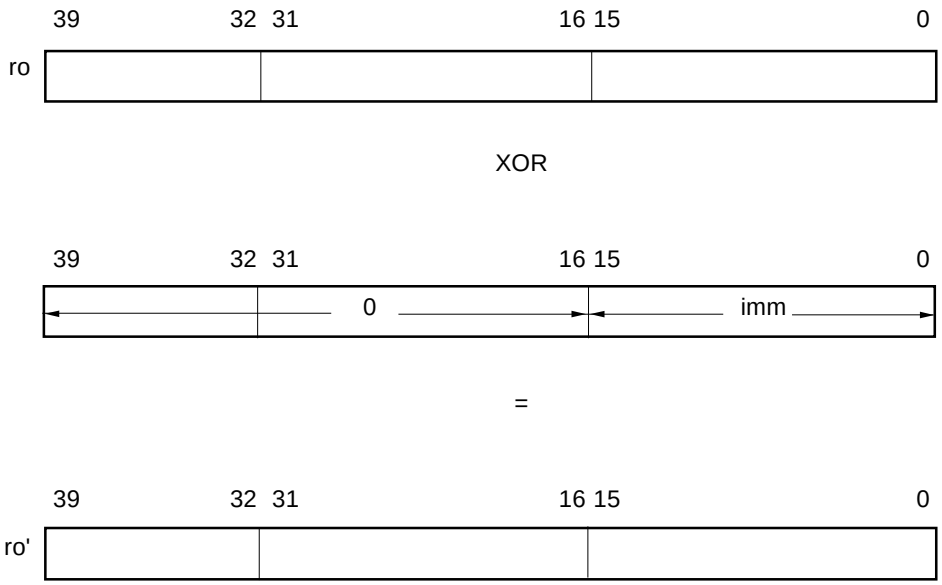
Binomial operation

ro' = ro /\ imm

Name of instruction:	**Immediate exclusive OR**
Mnemonic:	**ro' = ro /\ imm (imm = 0 to 0xFFFF (0 to 65535))**

Example R1 = R0 /\ 0x10

Explanation Instruction to obtain a logical exclusive OR of 40-bit data.
The logical exclusive OR is obtained from the value of bits 39 to 0 of the general-purpose register specified by ro and the 40-bit immediate value, zero-extended to bits 39 to 16 by setting the value of bits 15 to 0 with an instruction. The result is stored in bits 15 to 0 of the general-purpose register specified by ro'. Bits 39 to 16 of the general-purpose register specified by ro' become the same as those of the general-purpose register specified by ro. The values of the 40 bits specified by ro and ro' and the 16-bit immediate value are logical values.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

LT

LT

Binomial operation

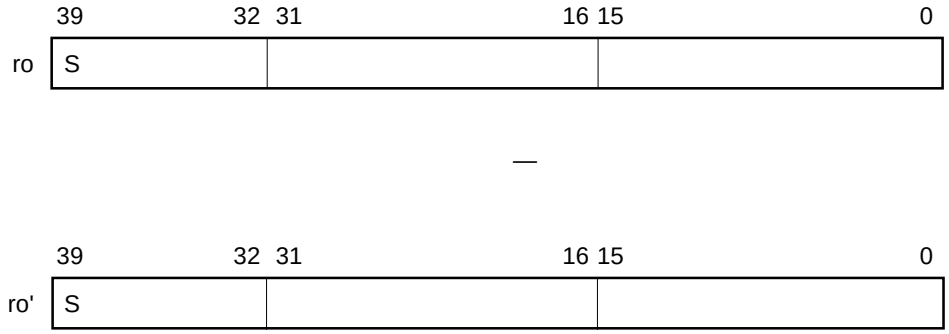
ro" = LT (ro, ro')

Name of instruction:	**Less than**
Mnemonic:	**ro" = LT (ro, ro')**

Example R2 = LT (R1, R4)

Explanation Instruction to compare two 40-bit data.

If the value of bits 39 to 0 of the general-purpose register specified by ro is less than the value of bits 39 to 0 of the register specified by ro', 0x00’0000’0001 is stored in bits 39 to 0 of the general-purpose register specified by ro". If the value of bits 39 to 0 of the general-purpose register specified by ro is greater than or equal to the value of bits 39 to 0 of the register specified by ro', 0x00’0000’0000 is stored in bits 39 to 0 of the general-purpose register specified by ro". The 40-bit values specified by ro and ro' are data formats represented with two’s complement, whereas the value of ro" can be taken as a boolean number format.



Arithmetic representation

```
if (ro < ro')   {ro" = 1;}
else            {ro" = 0;}
```

LT LT

Binomial operation

ro" = LT (ro, ro')

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	No

Caution **When ro' is subtracted from ro, comparison of the size of data where an overflow occurs is correctly performed.**
The overflow flag will not change.

3.3 Monomial Operation Instructions

These are instructions for specifying ALU operations. Any one register can be specified for the operand (input) from the general-purpose register file. Any one register can be specified for the output destination from the general-purpose register file.

The following monomial operation instructions are provided:

Clear	(CLR)
Increment	(INC)
Decrement	(DEC)
Absolute value	(ABS)
One’s complement	(NOT)
Two’s complement	(NEG)
Clip	(CLIP)
Round	(RND)
Exponent	(EXP)
Put	(PUT)
Accumulate add	(ACA)
Accumulate sub	(ACS)
Divide	(DIV)

CLR CLR

Monomial operation

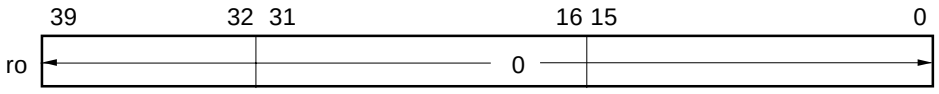
CLR (ro)

Name of instruction: **Clear**

Mnemonic: **CLR (ro)**

Example CLR (R0)

Explanation Instruction to clear all bits of the general-purpose register specified by ro to 0.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	Yes

INC

INC

Monomial operation

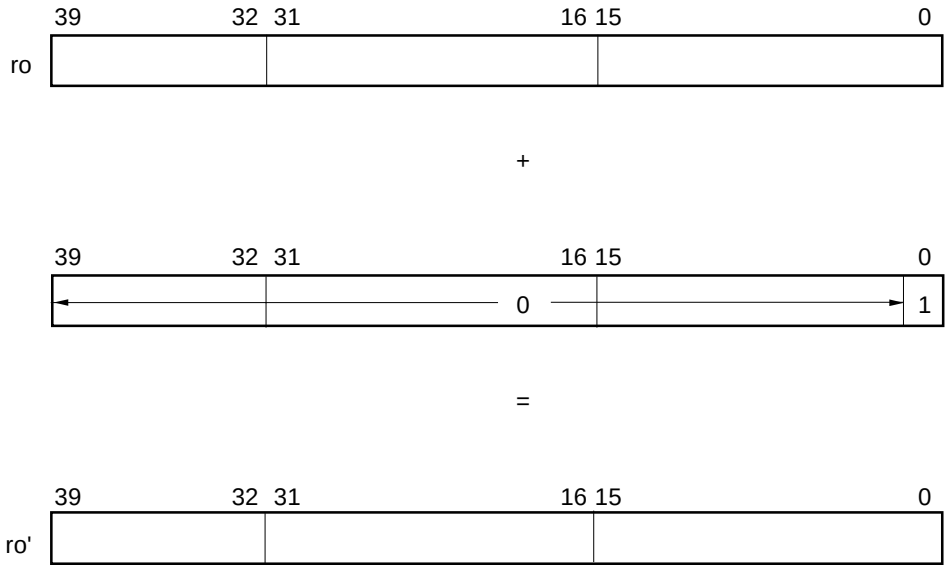
ro' = ro + 1

Name of instruction: **Increment**

Mnemonic: **ro' = ro + 1**

Example R1 = R1 + 1

Explanation Instruction to add 1 to 40-bit data.
The incremented result of the value of bits 39 to 0 of the general-purpose register specified by ro is stored in bits 39 to 0 of the general-purpose register specified by ro'. The 40-bit values specified by ro and ro' are data formats represented with two's complement.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	Yes

Caution **If the increment operation results in an overflow, the overflow flag ovf in the error status register ESR is set to 1.**
(0x7F'FFFF'FFFF + 1 → 0x80'0000'0000)

DEC DEC

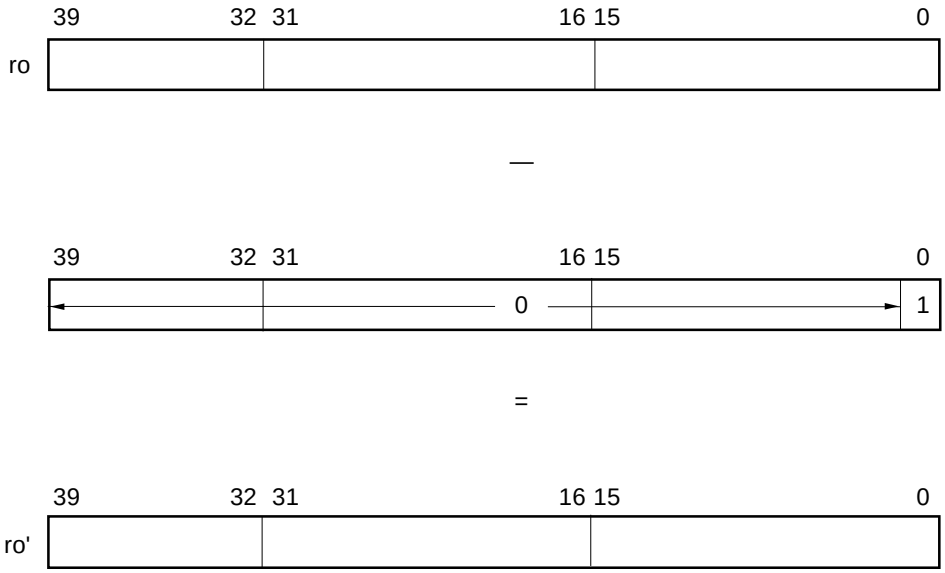
Monomial operation

ro' = ro – 1

Name of instruction:	**Decrement**
Mnemonic:	**ro' = ro – 1**

Example R1 = R1 – 1

Explanation Instruction to subtract 1 from 40-bit data.
The decremented result of the value of bits 39 to 0 of the general-purpose register specified by ro is stored in bits 39 to 0 of the general-purpose register specified by ro'. The 40-bit values specified by ro and ro' are data formats represented with two's complement.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	Yes

Caution **If the decrement operation results in an overflow, the overflow flag ovf in the error status register ESR is set to 1.**
(0x80'0000'0000 – 1 → 0x7F'FFFF'FFFF)

ABS

ABS

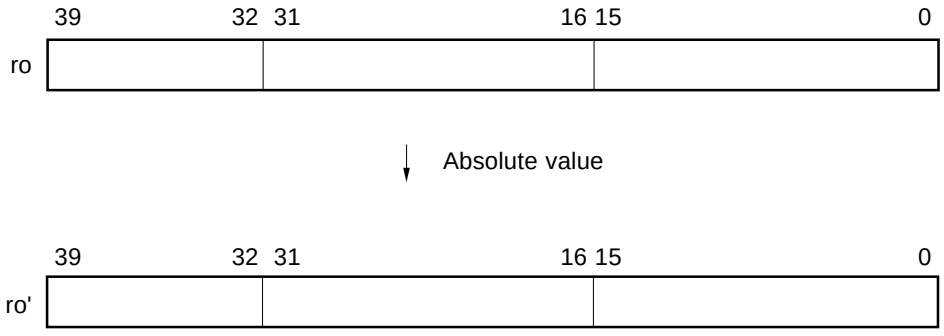
Monomial operation

ro' = ABS (ro)

Name of instruction:	**Absolute value**
Mnemonic:	**ro' = ABS (ro)**

Example R2 = ABS (R1)

Explanation Instruction to obtain the absolute value of 40-bit data.
The absolute value of bits 39 to 0 of the general-purpose register specified by ro is stored in bits 39 to 0 of the general-purpose register specified by ro'. The 40-bit values specified by ro and ro' are data formats represented with two's complement.



Arithmetic representation

```
if (ro < 0)   {ro' = –ro;}
else          {ro' = ro;}
```

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	Yes

Caution **When the ro specified general-purpose register value is 0x80'0000'0000, the overflow flag ovf in the error status register ESR is set to 1.**

NOT NOT

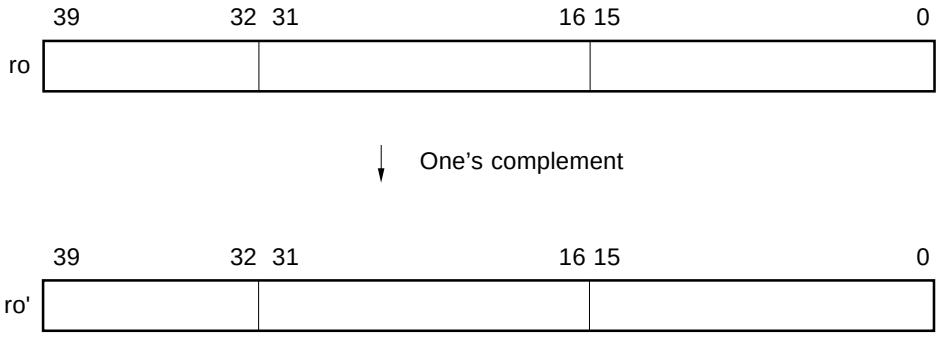
Monomial operation

ro' = ~ ro

Name of instruction:	**One's complement**
Mnemonic:	**ro' = ~ ro**

Example R2 = ~ R1

Explanation Instruction to obtain one's complement of 40-bit data.
One's complement of the value of bits 39 to 0 of the general-purpose register specified by ro (with 0 and 1 of each bit inverted) is stored in bits 39 to 0 of the general-purpose register specified by ro'.
The 40-bit values specified by ro and ro' are data formats represented with two's complement.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	Yes

NEG

NEG

Monomial operation

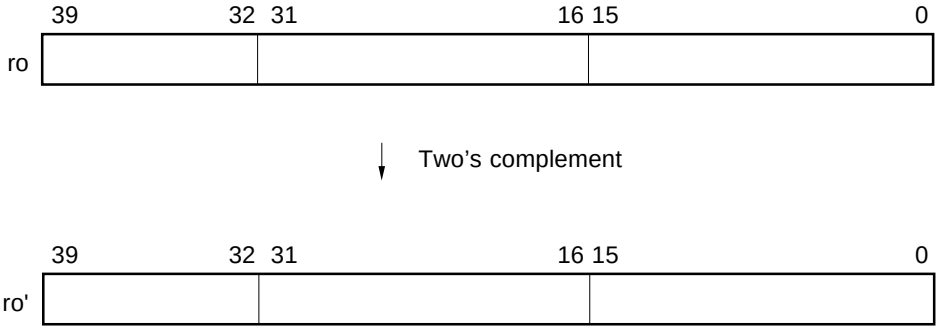
ro' = –ro

Name of instruction: **Two’s complement**

Mnemonic: **ro' = –ro**

Example R2 = –R1

Explanation Instruction to obtain two’s complement of 40-bit data.
Two’s complement (arithmetic negation) of the value of bits 39 to 0 of the general-purpose register specified by ro is stored in bits 39 to 0 of the general-purpose register specified by ro'. The 40-bit values specified by ro and ro' are data formats represented with two’s complement.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	Yes

Caution **When the value of the general-purpose register specified by ro is 0x80’0000’0000, the overflow flag ovf in the error status register ESR is set to 1.**

CLIP

CLIP

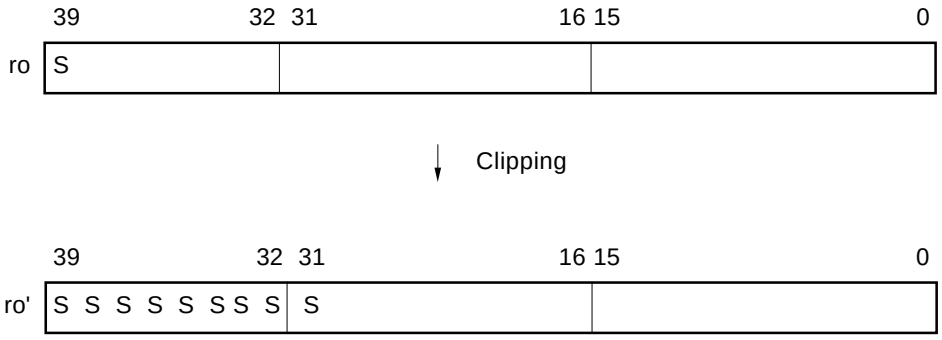
Monomial operation

ro' = CLIP (ro)

Name of instruction:	**Clip**
Mnemonic:	**ro' = CLIP (ro)**

Example R2 = CLP (R1)

Explanation Instruction to clip 40-bit data to 32 bits.
The value of bits 39 to 0 of the general-purpose register specified by ro and clipped to 32 bits (bits 31 to 0) is stored in bits 39 to 0 of the general-purpose register specified by ro'. The data is sign-extended to bits 39 to 32 of the general-purpose register specified by ro'. The 40-bit values specified by ro and ro' are data formats represented with two's complement.



Arithmetic representation

```
if      (ro > 0x00'7FFF'FFFF)    {ro' = 0x00'7FFF'FFFF;}
else if (ro < 0xFF'8000'0000)    {ro' = 0xFF'8000'0000;}
else    {ro' = ro;}
```

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	Yes

Caution **Even if saturation occurred due to CLIP execution, the overflow flag bit will not change.**

RND

RND

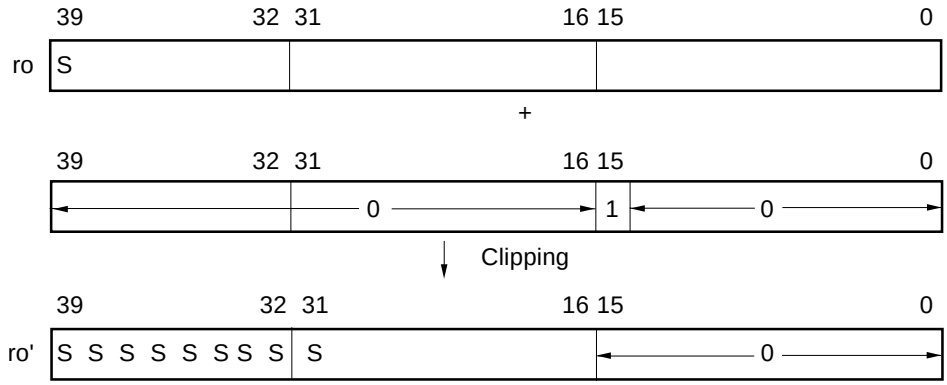
Monomial operation

ro' = ROUND (ro)

Name of instruction:	**Round**
Mnemonic:	**ro' = ROUND (ro)**

Example R2 = ROUND (R1)

Explanation Instruction to round and clip 40-bit data to 16-bit data.
Bits 15 to 0 of the general-purpose register specified by ro are rounded and clipped to 16 bits (bits 31 to 16) and then stored in bits 39 to 0 of the general-purpose register specified by ro'. The rounding of bits 15 to 0 means an addition of 1 to bit 15 only. The data is sign-extended to bits 39 to 32 of the general-purpose register specified by ro' and bits 15 to 0 are set to 0. The 40-bit values specified by ro and ro' are data formats represented with two's complement.



Arithmetic representation

```
if      (ro > 0x00'7FFF'0000)     {ro' = 0x00'7FFF'0000;}
else if (ro < 0xFF'8000'0000)     {ro' = 0xFF'8000'0000;}
else    {ro' = (ro + 0x8000) & 0xFF'FFFF'0000;}
```

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	Yes

Caution **Even if saturation occurred due to RND execution, the overflow flag bit will not change.**

EXP

EXP

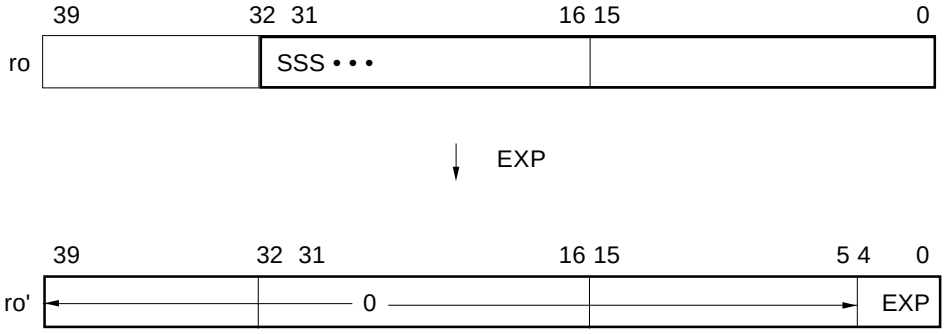
Monomial operation

ro' = EXP (ro)

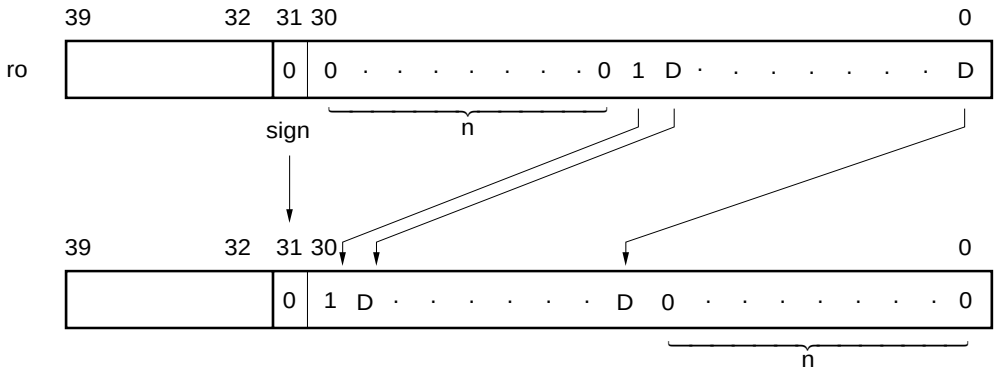
Name of instruction:	**Exponent**
Mnemonic:	**ro' = EXP (ro)**

Example R2 = EXP (R1)

Explanation Instruction to obtain a left shift amount to normalize to 32 bits.
The left shift amount (0 to 31) is obtained to normalize the value of bits 31 to 0 of the general-purpose register specified by ro. The result is stored in bits 39 to 0 of the general-purpose register specified by ro'. Bits 39 to 5 of the general-purpose register specified by ro' are set to 0. The 32-bit value specified by ro and the 40-bit value specified by ro' are data formats represented with two's complement.



Here, normalization means the following operation.



Count the number of bits whose value is the same as the sign bit from bit 30 and lower to find the first bit whose value is different from the sign bit. This number of bits is stored in ro' as n. After this, shift to the left by n. This operation is called "normalization". This keeps the value high-precision if its value is small in case multiplication.
The n is set to ro' when EXP instruction is executed.

EXP

EXP

Monomial operation

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	Yes

Cautions

1. **If the value of the general-purpose register specified by ro is 0x00’8000’0000 or more and 0xFF’7FFF’FFFF or less, the ro' value is not guaranteed.**
2. **If the value of the general-purpose register specified by ro is 0x00’0000’0000, the ro' value is set to 0x00’0000’0000.**
3. **If the value of the general-purpose register specified by ro is 0xFF’FFFF’FFFF, the ro' value is set to 0x1F.**
4. **Bits 31 to 0 of the general-purpose register specified by ro are evaluation targets.**

PUT

PUT

Monomial operation

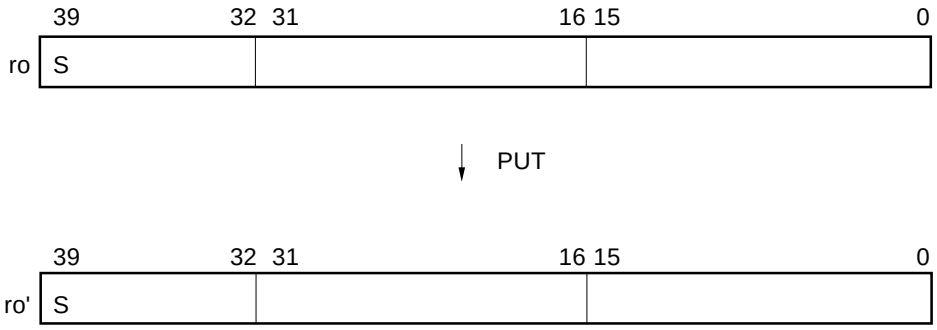
ro' = ro

Name of instruction: **Put**

Mnemonic: **ro' = ro**

Example R2 = R1

Explanation Instruction to put the 40-bit data of a register into another register. The value of bits 39 to 0 of the general-purpose register specified by ro are stored in bits 39 to 0 of the general-purpose register specified by ro'. The 40-bit values specified by ro and ro' are data formats represented with two's complement.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	Yes

ACA

ACA

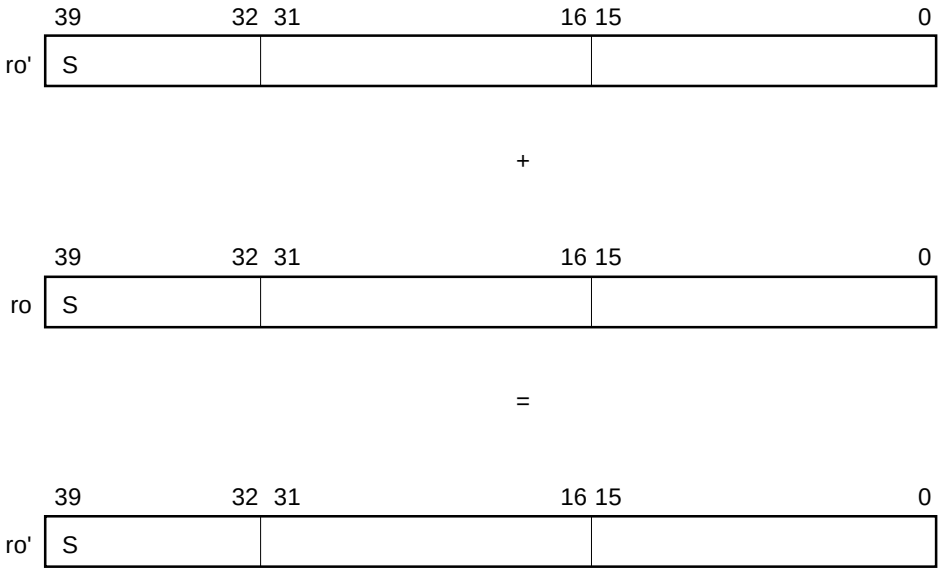
Monomial operation

ro' + = ro

Name of instruction:	**Accumulate add**
Mnemonic:	**ro' + = ro**

Example R2 + = R1

Explanation Instruction to add two 40-bit data.
The value of bits 39 to 0 of the general-purpose register specified by ro is added to the value of bits 39 to 0 of the general-purpose register specified by ro'. The sum is stored in bits 39 to 0 of the general-purpose register specified by ro'. The 40-bit values specified by ro and ro' are data formats represented with two's complement.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	Yes

Caution **If the addition results in an overflow, the overflow flag ovf in the error status register ESR is set to 1.**

ACS ACS

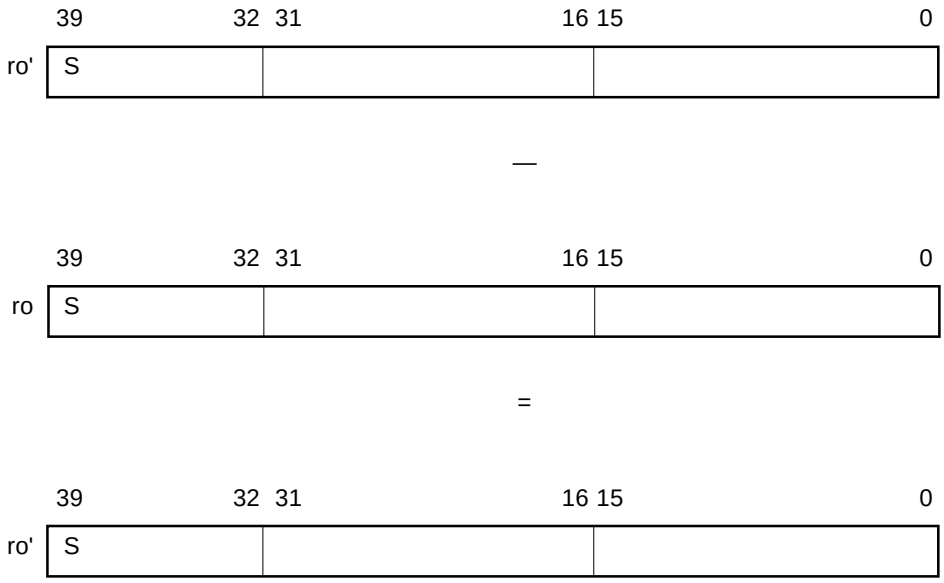
Monomial operation

ro' – = ro

Name of instruction:	**Accumulate subtract**
Mnemonic:	**ro' – = ro**

Example R2 – = R1

Explanation Instruction to subtract 40-bit data from 40-bit data.
The value of bits 39 to 0 of the general-purpose register specified by ro is subtracted from the value of bits 39 to 0 of the general-purpose register specified by ro'. The difference is stored in bits 39 to 0 of the general-purpose register specified by ro'. The 40-bit values specified by ro and ro' are data formats represented with two's complement.



ACS

ACS

Monomial operation

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	Yes

Caution **If the subtraction results in an overflow, the overflow flag ovf in the error status register ESR is set to 1.**

DIV DIV

Monomial operation

ro' / = ro

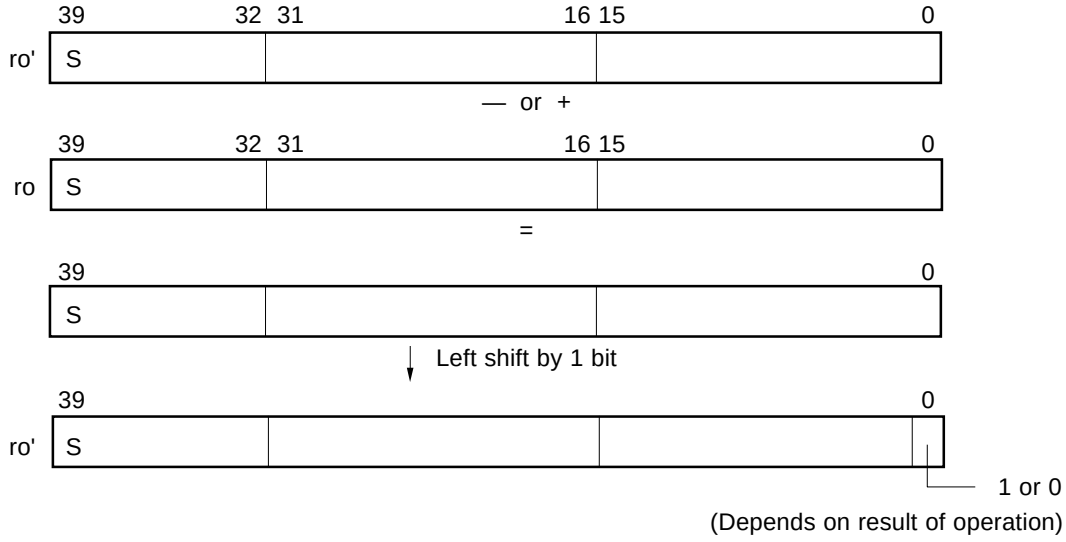
Name of instruction:	**Divide (1 bit)**
Mnemonic:	**ro' / = ro**

Example REP 16 (Example of division in 16-bit range)
R2 / = R1

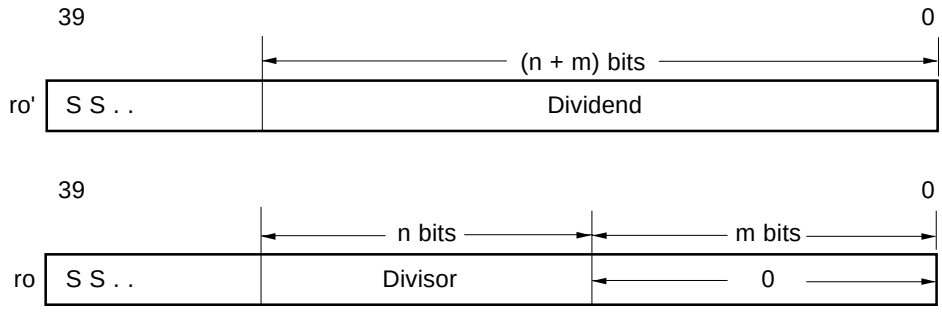
Explanation Divide instruction to obtain a quotient bit-wise in one operation.
The following operations are carried out using the values of bits 39 to 0 of the general-purpose registers specified by ro' and ro.

ro' and ro values with the same sign: ro' – ro
ro' and ro values with the different sign: ro' + ro

The calculated value is shifted to the left by one bit and one bit of the quotient is inserted into the LSB. The result is stored in bits 39 to 0 of the general-purpose register specified by ro'. The 40-bit values specified by ro and ro' are data formats represented with two's complement.



Set the ro' and ro values as follows:

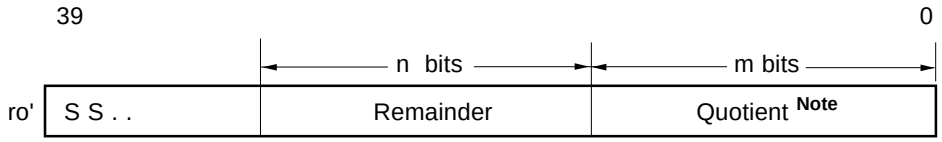


DIV

DIV

Monomial operation

Repeating the DIV instruction m times produces a quotient and remainder at the following position in the register specified by ro'.



Note $\text{Quotient} = \dfrac{\text{Dividend}}{|\ \text{Divisor}\ |}$

Remark The value of bits 39 to 0 of the general-purpose register specified by ro is the same as the value before operation.

Arithmetic representation :

```
if (sign (ro') == sign (ro))   {ro' = ro' – ro;}
else       {ro' = ro' + ro}
if (sign (ro') == 0)  {ro' = (ro' << 1) + 1;}
else       {ro' = ro' << 1;}
```

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	Yes	No	No	Yes

Caution **Specification of the same general-purpose register by ro and ro' is forbidden.**
The absolute value of the dividend should be less than that of devisor.

3.4 Load/Store Instructions

These are instructions to specify 16-bit data transfer between the memory and the general-purpose registers. Load/store operations consist of parallel load/store for simultaneously specifying an operation instruction, section load/store for specifying the load/store target general-purpose register range, direct addressing load/store for specifying addresses with instructions, and immediate index load/store for specifying the modify value using instructions. Any transfer target (input/output) can be specified from the general-purpose register file.

The following are provided as load/store instructions:

Parallel load/store	(LSPA) — Indirect addressing
Section load/store	(LSSE)
Direct addressing load/store	(LSDA)
Immediate value index load/store	(LSIM)

LSPA

LSPA

Load/Store

[ro = *dpx_mod] [ro' = *dpy_mod]
[*dpx_mod = rh] [*dpy_mod = rh']
[ro = *dpx_mod] [*dpy_mod = rh]
[*dpx_mod = rh] [ro = *dpy_mod]

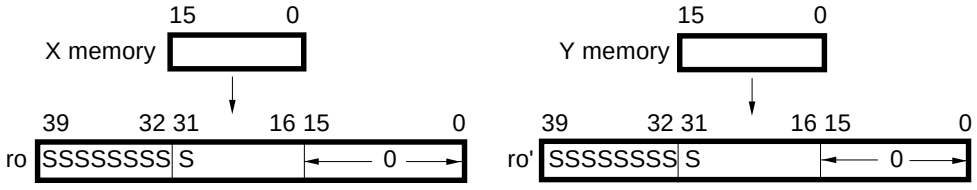
Name of instruction: Parallel load/store

Mnemonic: **[ro = *dpx_mod] [ro' = *dpy_mod]**
[*dpx_mod = rh] [*dpy_mod = rh']
[ro = *dpx_mod] [*dpy_mod = rh]
[*dpx_mod = rh] [ro = *dpy_mod]

Example R0 = *DP0++ R1 = *DP4++

Explanation Instruction to load/store 16-bit data between the indirectly addressed X/Y memories and a general-purpose register. It can be coded simultaneously with an operation instruction (except for the immediate value operation).
The following four types of coding methods are available:

(1) [ro = *dpx_mod] [ro' = *dpy_mod]
The contents of X memory with the address specified by dpx_mod are loaded to the register ro, and the contents of Y memory with the address specified by dpy_mod are loaded to the register ro'. When loading 16-bit data, the memory contents are stored in bits 31 to 16 of the general-purpose register. Bits 39 to 32 are sign-extended, and bits 15 to 0 are set to 0.



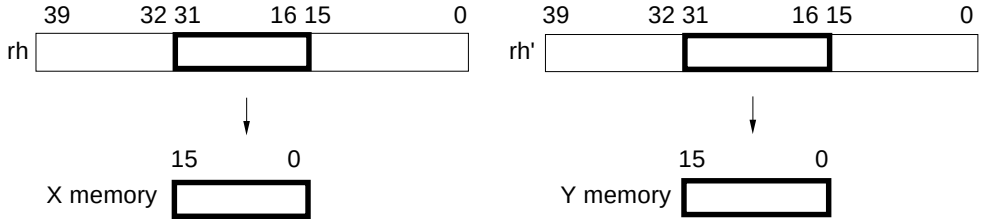
LSPA

LSPA

Load/Store

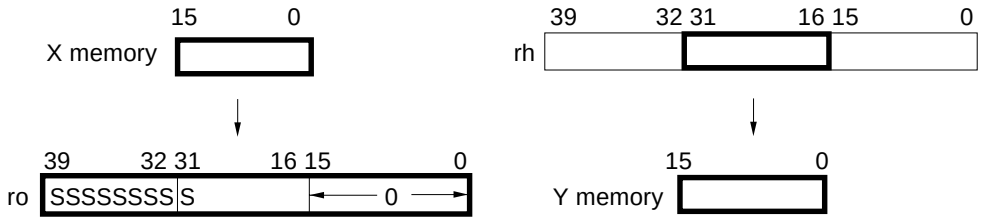
(2) [*dpx_mod = rh] [*dpy_mod = rh']

The contents of the register rh are loaded to X memory with the address specified by dpx_mod, and the contents of the register rh' are loaded to Y memory with the address specified by dpy_mod. When loading 16-bit data, the contents of the general-purpose register are stored in bits 15 to 0 of the X and Y memories. Bits 39 to 32 and bits 15 to 0 are ignored.



(3) [ro = * dpx_mod] [* dpy_mod = rh]

The contents of X memory with the address specified by dpx_mod are loaded to the register ro, and the contents of the register rh' are loaded to Y memory with the address specified by dpy_mod. When loading 16-bit data, the contents of X memory specified by ro are stored in bits 31 to 16 of the general-purpose register. Bits 39 to 32 are sign-extended, and bits 15 to 0 are set to 0.
In the same way, the contents of the general-purpose register specified by rh' are stored in bits 15 to 0 of the Y memory. Bits 39 to 32 and bits 15 to 0 are ignored.



LSPA

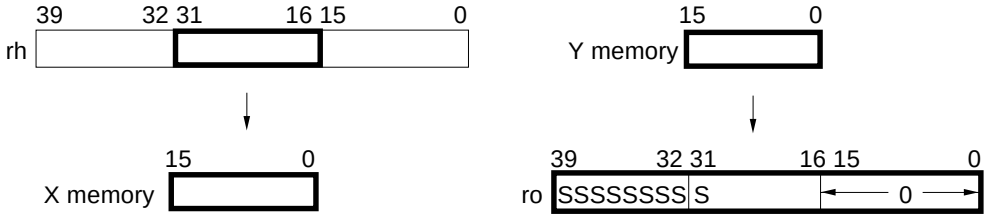
LSPA

Load/Store

(4) [*dpx_mod = rh] [ro = *dpy_mod]

When loading 16-bit data, the contents of the general-purpose register specified by rh are stored in bits 15 to 0 of the X memory. Bits 39 through 32 and bits 15 through 0 are ignored.

In the same way, the contents of Y memory specified by dpy_mod are stored in bits 31 to 16 of the general-purpose register specified by ro. Bits 39 to 32 are sign-extended, and bits 15 to 0 are set to 0.



Remarks 1. Load/store between the X memory and the general-purpose registers and load/store between the Y memory and the general-purpose registers can be specified independently or simultaneously. The data pointer and modifying method can be specified for X and Y independently.

2. The load/store target address is the data pointer value before modification, except for !DPn##. However, the data pointer set by the inter-register transfer instruction and the immediate value set instruction can be specified for the pointer from next-but-one instructions. It must not be specified for the pointer by the instruction just after the above instruction. The data pointer modification result is validated from the succeeding instruction onwards.

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial [Note]	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
Yes	Yes	Yes	No	No	No	No

Note Except for the immediate value operation

LSPA

LSPA

Load/Store

Cautions

1. When load/store between the X memory and a general-purpose register and between the Y memory and a general-purpose register is specified simultaneously, the following combinations of addrx and addry are forbidden.

★ **In the μPD7701x Family, if one of the following combinations occurs, the bus access error flag is set to 1. Note that this flag is not provided in μPD77111 Family devices.**

Forbidden Combinations
[μPD77016]
External area and external area
Peripheral area and peripheral area
[μPD77015, 77017, 77018, 77018A, 77019]
Internal ROM and internal ROM
Internal ROM and external area
External area and external area
Peripheral area and peripheral area
★ **[μPD77110, 77111, 77112, 77113, 77114]**
External area and external area
Peripheral area and peripheral area

2. The general-purpose register to be loaded from the X memory must not overlap the general-purpose register to be loaded from the Y memory.

3. The general-purpose register to write back the operation result of the simultaneously-described operation instruction must not overlap the general-purpose register to be loaded from the memory.

4. It is forbidden to load/store by instruction just after a value has been set to the data pointer with the same data pointer value set as an address.

Forbidden example 1: Inter-register transfer instruction

```
DP0 = R4L;
R0 = *DP0++;
```

Forbidden example 2: Immediate value set instruction

```
DP0 = 0x1234;
R0 = *DP0++;
```

LSSE LSSE

Load/Store

[dest = *dpx_mod] [dest' = *dpy_mod]
[dest = *dpx_mod] [*dpy_mod = source]
[*dpx_mod = source] [dest = *dpy_mod]
[*dpx_mod = source] [*dpy_mod = source']

Name of instruction: **Section load/store**

Mnemonic: **[dest = *dpx_mod] [dest' = *dpy_mod]**
[dest = *dpx_mod] [*dpy_mod = source]
[*dpx_mod = source] [dest = *dpy_mod]
[*dpx_mod = source] [*dpy_mod = source']

Registers that can be specified are as follows. Any one of these registers may be specified.

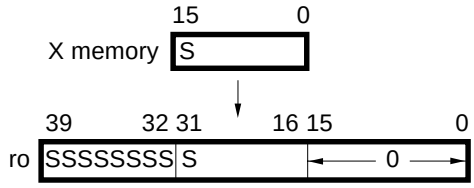
dest, dest' = {ro, reh, re, rh, rl}
source, source' = {re, rh, rl}

Example R0EH = *DP0++ R0L = *DP4++

Explanation Instruction to load/store 16-bit data between the indirectly addressed X/Y memory and a general-purpose register. The range of the general-purpose registers that are the target of the load/store instruction can be specified.

(1) There are five ways to load 16-bit data from the memory specified by dpx_mod or dpy_mod to the general-purpose register specified by dest or dest'.

(a) When R0 to R7 are specified for dest or dest', the value of the memory specified by dpx_mod or dpy_mod is loaded to bits 31 to 16 of the general-purpose register, the data is sign-extended to bits 39 to 32 and bits 15 to 0 are set to 0.

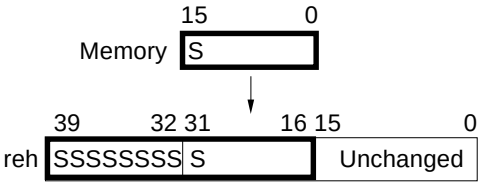


LSSE

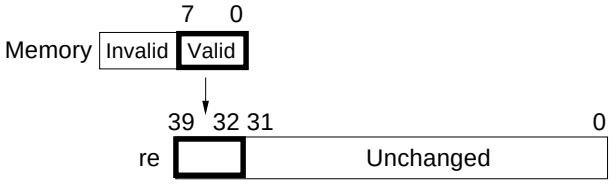
LSSE

Load/Store

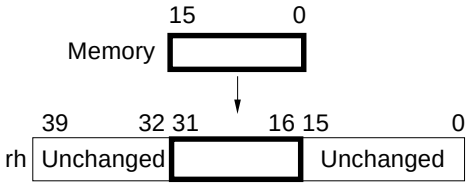
(b) When R0EH to R7EH are specified for dest or dest', the value of the memory specified by dpx_mod or dpy_mod is loaded to bits 31 to 16 of the general-purpose register and the data is sign-extended to bits 39 to 32. Bits 15 to 0 of the general-purpose register remain unchanged.



(c) When R0E to R7E are specified for dest or dest', the low-order 8 bits of the value of the memory specified by dpx_mod or dpy_mod are loaded to bits 39 to 32 of the general-purpose register. Bits 31 to 0 of the general-purpose register remain unchanged. The high-order 8 bits of the memory value are ignored.



(d) When R0H to R7H are specified for dest or dest', the value of the memory specified by dpx_mod or dpy_mod is loaded to bits 31 to 16 of the general-purpose register. Bits 39 to 32 and bits 15 to 0 of the general-purpose register remain unchanged.

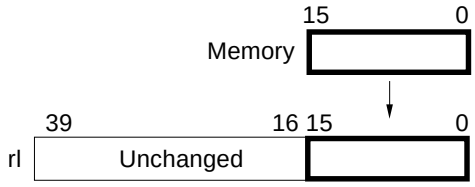


LSSE

LSSE

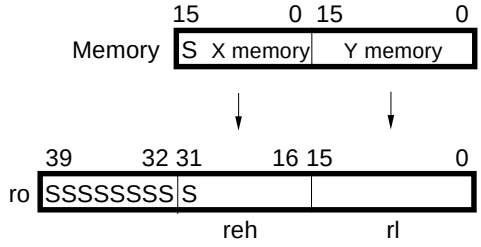
Load/Store

(e) When R0L to R7L are specified for dest or dest', the value of the memory specified by dpx_mod or dpy_mod is loaded at bits 15 to 0 of the general-purpose register. Bits 39 to 16 of the general-purpose register remain unchanged.

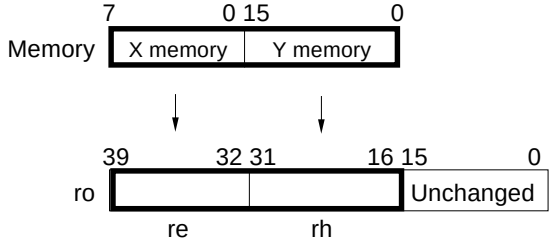


When 16-bit data is loaded from the Y memory to the same general-purpose register as that to which data is loaded from the X memory simultaneously, the general-purpose register value becomes as follows. X memory and Y memory in the source general-purpose register are interchangeable.

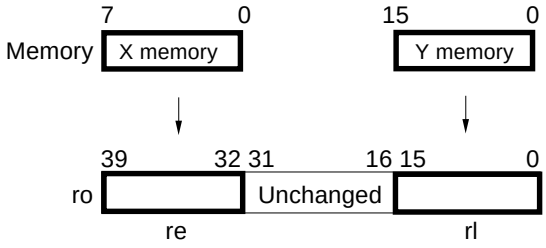
• RnEH and RnL; (example: [reh = *dpx_mod] [rl = *dpy_mod])



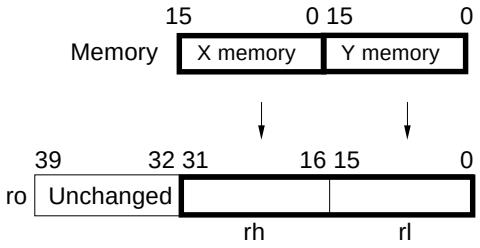
• RnE and RnH; (example: [re = *dpx_mod] [rh = *dpy_mod])



• **RnE and RnL; (example: [re = *dpx_mod] [rl = *dpy_mod])**



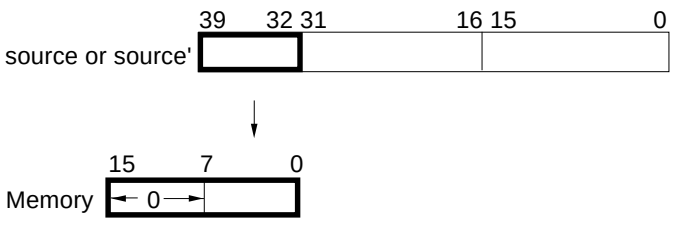
• **RnH and RnL; (example: [rh = *dpx_mod] [rl = *dpy_mod])**



Caution Other combinations are prohibited.

(2) There are three ways to store 16-bit data from the general-purpose register specified by source or source' to the memory specified by dpx_mod or dpy_mod.

(a) When R0E to R7E are specified for source or source', the value of bits 39 to 32 of the general-purpose register is stored in the low-order 8 bits of the memory specified by dpx_mod or dpy_mod. The high-order 8 bits of the memory are set to 0. Bits 31 to 0 of the general-purpose register are ignored.

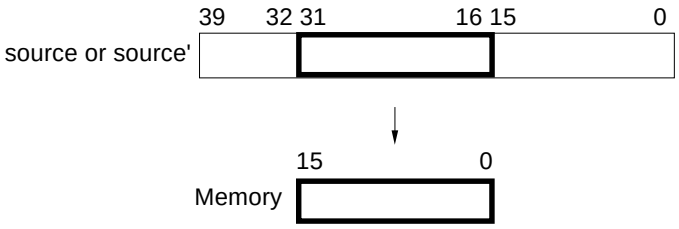


LSSE

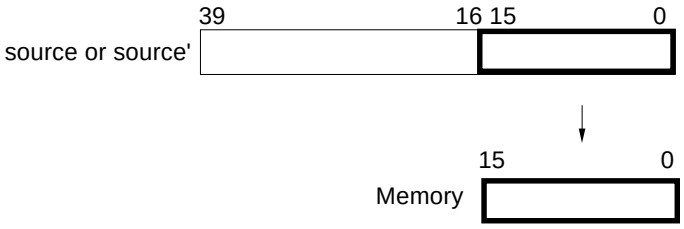
LSSE

Load/Store

(b) When R0H to R7H are specified for source or source', the value of bits 31 to 16 of the general register is stored in the memory specified by dpx_mod or dpy_mod. Bits 39 to 32 and bits 16 to 0 of the general-purpose register are ignored.



(c) When R0L to R7L are specified for source or source', the value of bits 15 to 0 of the general-purpose register is stored in the memory specified by dpx_mod or dpy_mod. Bits 39 to 16 of the general-purpose register are ignored.



Remarks 1. Load/store between the X memory and a general-purpose register and load/store between the Y memory and a general-purpose register can be specified independently or simultaneously.

2. The data pointer and its modifying method can be specified for dpx_mod and dpy_mod independently.

3. The load/store target address is the data pointer value before modification except for !DPn##. The data pointer modification result is validated from the succeeding instruction onward.

LSSE

LSSE

Load/Store

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

Cautions

1. When load/store between the X memory and a general-purpose register and between the Y memory and a general-purpose register is specified simultaneously, the following combinations of addrx and addry are forbidden.

★ **In the μPD7701x Family, if one of the following combinations occurs, the bus access error flag is set to 1. Note that this flag is not provided in μPD77111 Family devices.**

[Forbidden Combinations]
[μPD77016]
External area and external area
Peripheral area and peripheral area
[μPD77015, 77017, 77018, 77018A, 77019]
Internal ROM and internal ROM
Internal ROM and external area
External area and external area
Peripheral area and peripheral area
★ **[μPD77110, 77111, 77112, 77113, 77114]**
External area and external area
Peripheral area and peripheral area

2. The data pointer that was set with the inter-register transfer instruction or immediate value set instruction can be specified as a pointer with the next-but-one instruction from the above instruction. It must not be specified for the pointer by the instruction just after the above instruction. It is forbidden to load/store by instruction just after a value has been set to the data pointer with the same data pointer value set as an address.

Forbidden example 1: Inter-register transfer instruction

```
DP0 = R4L;
R0E = *DP0++;
```

Forbidden example 2: Immediate value set instruction

```
DP0 = 0x1234;
R0H = *DP0++;
```

LSDA

LSDA

Load/Store

dest = *addr
***addr = source**

Name of instruction: **Direct addressing load/store**

Mnemonic: **dest = *addr**
***addr = source**

Addresses that can be specified for addr (address):

X memory 0 to 0xFFFF (0 to 65535): X (X memory)
X memory 0 to 0xFFFF (0 to 65535): Y (Y memory)

Registers that can be specified are as follows. Any one of these registers may be specified.

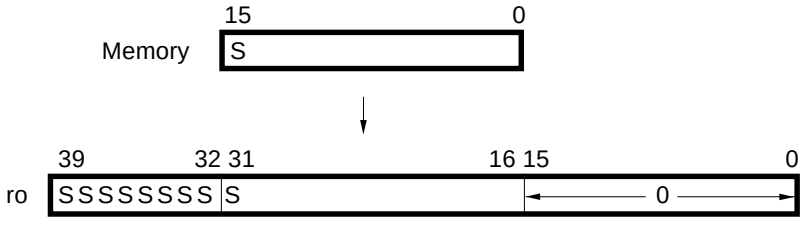
dest = {ro, reh, re, rh, rl}
source = {re, rh, rl}

Example R0H = *0x1234: X

Explanation Instruction to load/store 16-bit data between the directly addressed X or Y memory and a general-purpose register. The range of the general-purpose registers that are the target of the load/store instruction can be specified.

(1) There are five ways to load 16-bit data from the memory specified by addr to the general-purpose register specified by dest. In every case, the value loaded to the general-purpose register will be validated from the next instruction.

(a) When R0 to R7 are specified for dest, the value of the memory specified by addr is loaded to bits 31 to 16 of the general-purpose register, the data is sign-extended to bits 39 to 32, and bits 15 to 0 are set to 0.

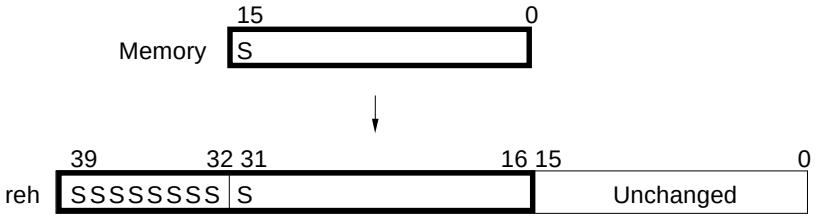


LSDA

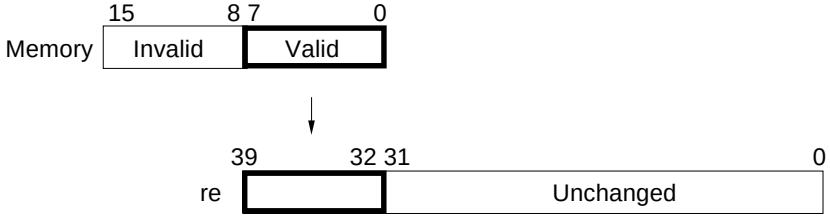
LSDA

Load/Store

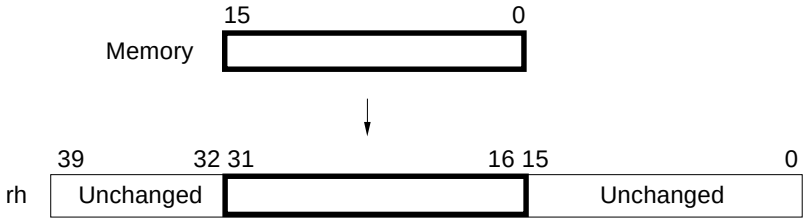
(b) When R0EH to R7EH are specified for dest, the value of the memory specified by addr is loaded to bits 31 to 16 of the general-purpose register and sign-extended to bits 39 to 32. Bits 15 to 0 of the general-purpose register remain unchanged.



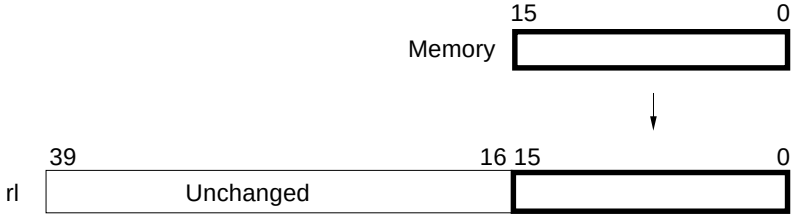
(c) When R0E to R7E are specified for dest, the low-order 8 bits of the value of the memory specified by addr are loaded to bits 39 to 32 of the general-purpose register. Bits 31 to 0 of the general-purpose register remain unchanged. The high-order 8 bits of the memory value are ignored.



(d) When R0H to R7H are specified for dest, the value of the memory specified by addr is loaded to bits 31 to 16 of the general-purpose register. Bits 39 to 32 and bits 15 to 0 of the general-purpose register remain unchanged.



(e) When R0L to R7L are specified for dest, the value of the memory specified by addr is loaded to bits 15 to 0 of the general-purpose register. Bits 39 to 16 of the general-purpose register remain unchanged.



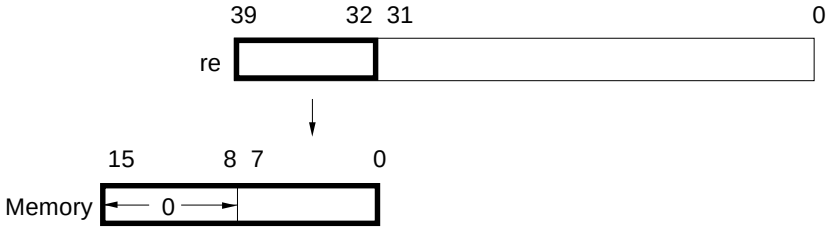
LSDA

LSDA

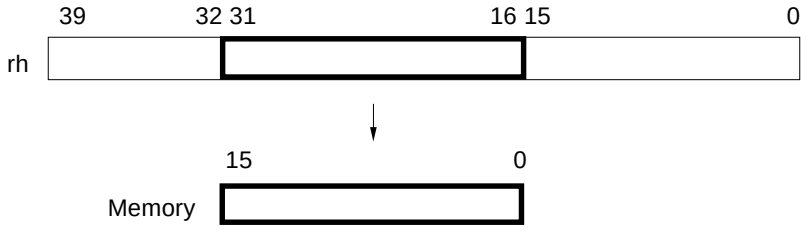
Load/Store

(2) There are three ways to store 16-bit data from the general-purpose register specified by source to the memory specified by addr.

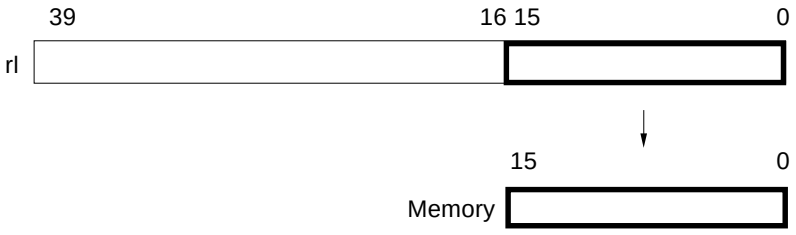
(a) When R0E to R7E are specified for source, the value of bits 39 to 32 of the general-purpose register is stored in the low-order 8 bits of the memory specified by addr. The high-order 8 bits of the memory are set to 0. Bits 31 to 0 of the general-purpose register are ignored.



(b) When R0H to R7H are specified for source, the value of bits 31 to 16 of the general-purpose register is stored into the memory specified by addr. Bits 39 to 32 and bits 15 to 0 of the general-purpose register are ignored.



(c) When R0L to R7L are specified for source, the values of bits 15 to 0 of the general-purpose register are stored in the memory specified by addr. Bits 39 to 16 of the general-purpose register are ignored.



LSDA

LSDA

Load/Store

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

LSIM

LSIM

Load/Store

dest = *dp_imm
***dp_imm = source**

Name of instruction: **Immediate value index load/store**

Mnemonic: **dest = *dp_imm**
***dp_imm = source (imm = 0 to 0xFFFF)**

Registers that can be specified are as follows. Any one of these registers may be specified.

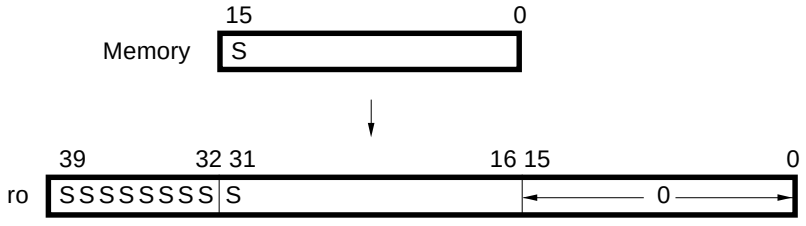
dest = {ro, reh, re, rh, rl}
source = {re, rh, rl}

Example R0H = *DP0## 0x10

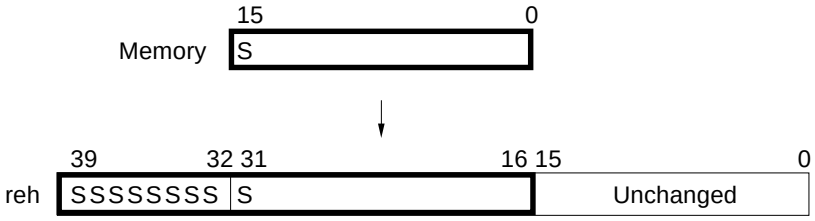
Explanation Instruction to load/store 16-bit data between the indirectly addressed X or Y memory and a general-purpose register. The range of the general-purpose registers that are the target of the load/store instructions can be specified.

(1) There are five ways to load 16-bit data from the memory specified by dp_imm to the general-purpose register specified by dest.

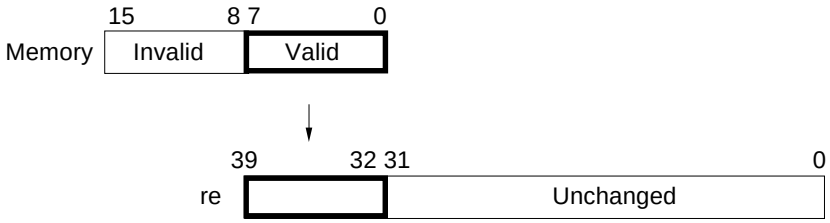
(a) When R0 to R7 are specified for dest, the value of the memory specified by dp_imm is loaded to bits 31 to 16 of the general-purpose register, the data is sign-extended to bits 39 to 32, and bits 15 to 0 are set to 0.



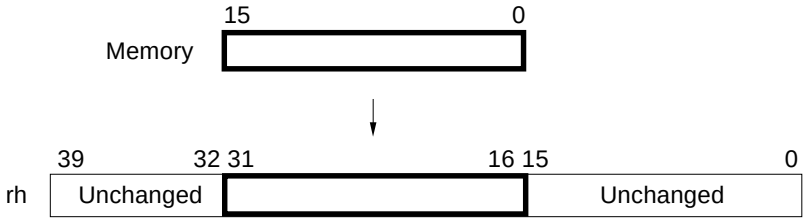
(b) When R0EH to R7EH are specified for dest, the value of the memory specified by dp_imm is loaded to bits 31 to 16 of the general-purpose register, and the data is sign-extended to bits 39 to 32. Bits 15 to 0 of the general-purpose register remain unchanged.



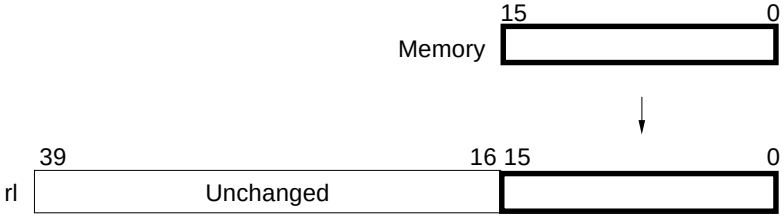
(c) When R0E to R7E are specified for dest, the low-order 8 bits of the value of the memory specified by dp_imm are loaded to bits 39 to 32 of the general-purpose register. Bits 31 to 0 of the general-purpose register remain unchanged. The high-order 8 bits of the memory value are ignored.



(d) When R0H to R7H are specified for dest, the value of the memory specified by dp_imm is loaded to bits 31 to 16 of the general-purpose register. Bits 39 to 32 and bits 15 to 0 of the general-purpose register remain unchanged.



(e) When R0L to R7L are specified for dest, the value of the memory specified by dp_imm is loaded in bits 15 to 0 of the general-purpose register. Bits 39 to 16 of the general-purpose register remain unchanged.



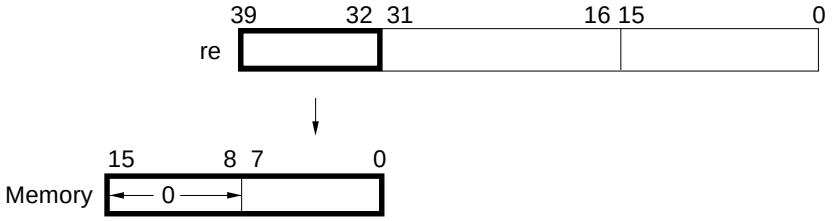
LSIM

LSIM

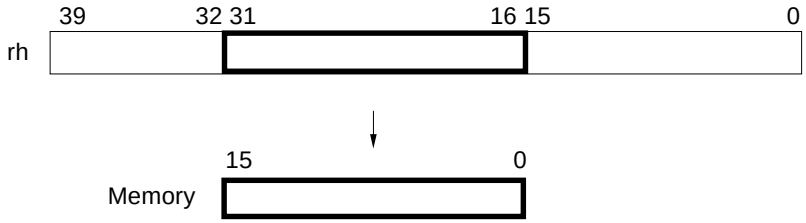
Load/Store

(2) There are three ways to store 16-bit data from the general-purpose register specified by source in the memory specified by dp_imm.

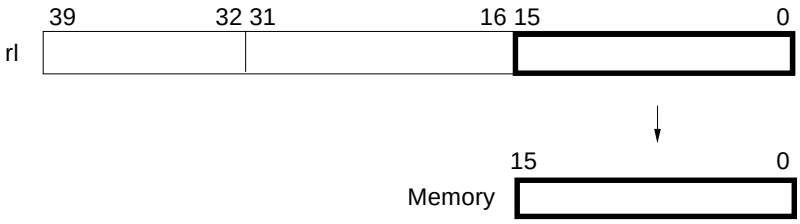
(a) When R0E to R7E are specified for source, the value of bits 39 to 32 of the general-purpose register is stored in the low-order 8 bits of the memory specified by dp_imm. The high-order 8 bits of the memory are set to 0. Bits 31 to 0 of the general-purpose register are ignored.



(b) When R0H to R7H are specified for source, the value of bits 31 to 16 of the general-purpose register is stored in the memory specified by dp_imm. Bits 39 to 32 and bits 15 to 0 of the general-purpose register are ignored.



(c) When R0L to R7L are specified for source, the value of bits 15 to 0 of the general-purpose register is stored in the memory specified by dp_imm. Bits 39 to 16 of the general-purpose register are ignored.



Remark The load/store target address is the data pointer value before modification (except for !DPn##). However, the data pointer set by the inter-register transfer instruction and the immediate value set instruction can be specified for the pointer from the next instruction. The data pointer modification result is validated from the succeeding instruction onwards.

LSIM

LSIM

Load/Store

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

Caution **It is forbidden to load/store by instruction just after a value has been set to the data pointer with the same data pointer value set as an address.**

Forbidden example 1: Inter-register transfer instruction

```
DP0 = R4L;
R0 = *DP0## 0xABCD;
```

Forbidden example 2: Immediate value set instruction

```
DP0 = 0x1234;
R0 = *DP0## 0xABCD;
```

3.5 Inter-Register Transfer Instructions

This is an instruction to transfer data between general-purpose registers and other registers via the main bus.

MOV

MOV

Inter-register transfer

dest = rl
rl = source

Name of instruction:	**Inter-register transfer**
Mnemonic:	**dest = rl** **rl = source**
	All registers other than general-purpose registers can be specified for dest and source.

Example DP0 = R0L

Explanation Instruction to transfer data between a general-purpose register and another register, via the main bus.
The value of the register specified by source is transferred bottom-justified to the register specified by dest via the main bus.

The following table shows the target registers of this instruction.

Register Name	Assembler-Reserved Name
General-purpose register	R0L to R7L (L part of R0 to R7)
Data pointer	DP0 to DP7
Index register	DN0 to DN7
Modulo register	DMX, DMY
Stack	STK
Stack pointer	SP
Loop counter[Note]	LC
Loop stack (LSTK)	LSR1, LSR2, LSR3
Loop stack pointer	LSP
Status register	SR
Interrupt enable flag stack register	EIR
Error status register	ESR

Note This register is not specified as dest.

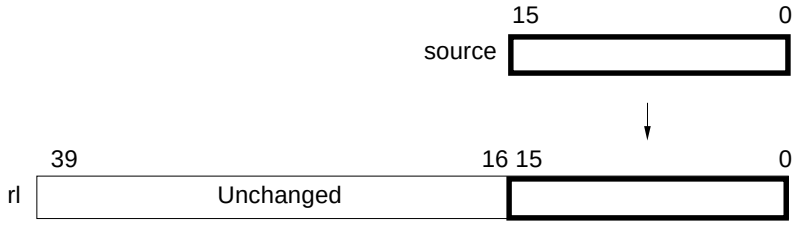
MOV

MOV

Inter-register transfer

(1) When rl is specified for dest, the value in bits 15 to 0 of the general-purpose register is transferred to the register specified by dest. When the valid bit length of the register specified by dest is m bits (m<16), bits 15 to m of the general-purpose register are ignored.

(2) When rl is specified for source, the value of the register specified by source is transferred to bits 15 to 0 of the general-purpose register. Bits 39 to 16 of the general-purpose register remain unchanged. When the bit length of the source register is n bits (n < 16) as in the ESR, bits 15 to n of the general-purpose register are undefined.



Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional[Note]
No	No	No	No	No	No	Yes

Note Limited to DP0 to DP7, DN0 to DN7, DMX, and DMY for dest and source.

Cautions

1. **If DP0 to DP7 are specified for dest, the transfer result can be specified for the load/store pointer from next-but-one instruction.**
2. **Modulo index addition should not be executed after a value in the address range 0x8000 to 0xFFFF is transferred to DMX or DMY.**
3. **When modulo index addition (DPn%%) is executed, transfer a value in the address range 1 to 0x7FFF. When a value of 0, or 0x8000 or higher is transferred, DPn%% does not operate correctly.**
4. **The value of bit 15 of LC is the loop flag (in/out loop). Do not change this flag.**
5. **Value between 0 and 0xF can be set for SP. Do not set this register to 0x10 to 0xFFFF.**
6. **Value between 0 and 4 can be set for LSP. Do not set this register to 0x5 to 0xFFFF.**
7. **Do not describe the RET or RETI instruction just after the MOV instruction to load from/ store in STK or SP.**

3.6 Immediate Value Set Instruction

This is an instruction to set the immediate value to the general-purpose registers and each address operation unit register.

LDI

LDI

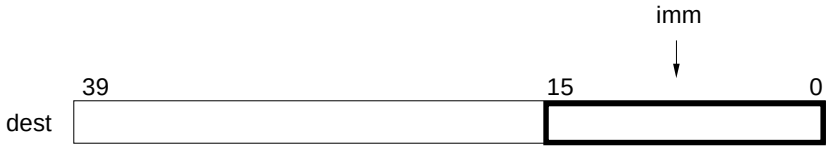
Immediate value set

dest = imm

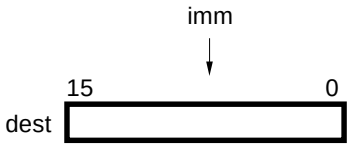
Name of instruction:	**Immediate value set**
Mnemonic:	**dest = imm (imm = 0 to 0xFFFF (0 to 65535))** **dest = {rl, dp, dn, dm}**

Example DP0 = 0x1234

Explanation Instruction to set the 16-bit immediate value bottom-justified to the register specified by dest. When R0L to R7L are specified for dest, the immediate value is set at bits 15 to 0 of the general-purpose register. Bits 39 to 16 remain unchanged.



When DP0 to DP7, DN0 to DN7, DMX, or DMY is specified for dest, the 16-bit immediate value becomes the register value.



LDI

LDI

Immediate value set

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

Cautions

1. **The set value is validated with the succeeding instruction. However, if DP0 to DP7 are specified for dest, the setting result can be specified for the load/store pointer from next-but-one instruction; it must not be specified for the succeeding load/store instruction pointer.**
2. **When executing modulo index addition, do not specify 0 or one of 0x8000 to 0xFFFF for DMX and DMY.**

3.7 Branch Instructions

These are instructions to specify program branches. Branch instructions consist of branch, jump, subroutine call, and return instructions.

Jump	(JMP)
Register indirect jump	(JREG)
Subroutine call	(CALL)
Register indirect subroutine call	(CREG)
Return	(RET)
Interrupt return	(RETI)

JMP

JMP

Branch

JMP imm

Name of instruction: **Jump**

Mnemonic: **JMP imm (imm = 0 to 0xFFFF (0 to 65535))**

Example JMP 0x1234

Explanation Instruction to branch to all addresses in the instruction memory space of the μPD77016 Family. A branch is executed in two cycles.
Note that no conditional branch instruction, which branches when a flag value is either 0 or 1, is provided for the μPD77016 Family. Therefore, when a conditional branch is required, a conditional instruction and a branch instruction should be combined as shown below. Consequently, in this case, the conditional branch is executed in three cycles when the specified condition is satisfied, and in one cycle when unsatisfied.

```
IF (R0 == 0)    JMP LABEL        or
IF (R2 > 0)     JMP DP0    (Concerning "JMP DP0", an explanation is given in the description
                            for the JREG instruction on the following page.)
```

Remarks The μPD77016 Family JMP and CALL instructions are actually instructions that branch with a relative value in the range of +/– 32 K words. However, the user does not need to consider the relative value because the assembler and linker performs relative address calculations. The user should describe absolute addresses (actual jump destination addresses) in the source program. If code with a relative address is required, put a “$” at the beginning of an operand of the JMP instruction as shown below:

```
JMP   $ + 2  .....  Jumps to the next-but-one address.
JMP   $ – 3  .....  Jumps three addresses before source address.
```

Execution cycle 2 (When used with a conditional instruction: 3 when condition satisfied and 1 when condition not satisfied.)

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	Yes

Caution **Do not describe the JMP instruction as a repeat target instruction or within three instructions of the loop end.**

JREG

JREG

Branch

JMP dp

Name of instruction: Register indirect jump

Mnemonic: **JMP dp**

Example JMP DP0

Explanation Instruction to branch to the address of the register value specified by dp (PC ← dp). The register indirect branch is executed in three cycles.
For the coding method of a conditional jump instruction by means of the JREG instruction, refer to the explanation for the JMP instruction. In this case, the conditional branch is executed in three cycles when the specified condition is satisfied, and in one cycle when unsatisfied. Unlike the JMP instruction, the execution cycles of the JREG instruction are unchanged regardless of use alone or together with a conditional instruction.

Execution cycle 3 (When used with a conditional instruction, 3 when condition is satisfied and 1 when unsatisfied.)

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	Yes

Caution **Do not describe the JREG instruction as a repeat target instruction or within three instruction of the loop end.**

CALL CALL

Branch

CALL imm

Name of instruction:	**Subroutine call**
Mnemonic:	**CALL imm (imm = 0 to 0xFFFF (0 to 65535))**

Example CALL 0x1234

Explanation Instruction that branches to a subroutine located at any address in the instruction memory space of the μPD77016 Family. A subroutine call is executed in two cycles.
This instruction execution increments the stack pointer and stores the address of the next instruction to the stack, then branches to the address specified by the operand (SP ← SP + 1, STACK ← PC + 1, PC ← PC + imm).
To execute a conditional branch, a conditional instruction and a branch instruction should be combined as shown below:

```
IF  (R0! = 0)  CALL LABEL
```

In the same way as the JREG instruction, the conditional branch is executed in three cycles when the specified condition is satisfied, and in one cycle when unsatisfied.

Remarks The μPD77016 JMP and CALL instructions are actually instructions that branch with a relative value in the range of +/– 32 K words. However, the user does not need to consider the relative value because the assembler and linker performs relative address calculations. The user should code absolute addresses (actual jump destination addresses) in the source program. If code with a relative address is required, put a "$" at the beginning of an operand of the CALL instruction as shown below:

```
CALL  $ + 2 .....  Jumps to the next-but-one address.
CALL  $ – 3 .....  Jumps three addresses before source address.
```

CALL

CALL

Branch

Execution cycle 2 (When used with a conditional instruction: 3 when condition satisfied, and 1 when condition not satisfied.)

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	Yes

Cautions

1. **Do not describe the CALL instruction as a repeat target instruction or within three instructions of the loop end.**
2. **Do not execute the CALL instruction when the SP value is within 0xF to 0x1F. If it is executed, a stack underflow or overflow occurs, the return address undefined, although the subroutine call is normally executed.**
3. **Do not describe the unconditional RET instruction or the unconditional RETI instruction, just after a CALL instruction that is combined with a conditional instruction.**

CREG CREG

Branch

CALL dp

Name of instruction: **Register indirect subroutine call**

Mnemonic: **CALL dp**

Example CALL DP0

Explanation Instruction that executes the same operation as that of the CALL instruction except that the subroutine call address is specified with dp, and three cycles are taken for execution (SP ← SP + 1, STACK ← PC + 1, PC ← reg).

Execution cycle 3 (When used with a conditional instruction: 3 when condition satisfied, and 1 when condition not satisfied.)

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	Yes

Cautions

1. **Do not describe the CALL instruction as a repeat target instruction or within three instructions of the loop end.**
2. **Do not execute the CALL instruction when the SP value is within 0xF to 0x1F. If it is executed, a stack underflow or overflow occurs, rendering the return address undefined, although the subroutine call is normally executed.**
3. **Do not describe the unconditional RET instruction or the unconditional RETI instruction just after a CREG instruction that is combined with a conditional instruction.**

RET

RET

Branch

RET

Name of instruction: **Return**

Mnemonic: **RET**

Example RET

Explanation Instruction that returns processing from the subroutine to the main routine. The processing jumps to the address specified by the stack value indicated by the stack pointer, and then the stack pointer is decremented by one (PC ← STACK, SP ← SP – 1). A return is executed in two cycles.
To execute a conditional return, a conditional instruction and the RET instruction should be combined in the same way as the JMP and CALL instructions, as shown below:

```
IF (R0 = = 0) RET;
```

Execution cycle 2 (When used with a conditional instruction: 3 when condition satisfied, and 1 when condition not satisfied.)

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	Yes

Cautions

1. **Executing the RET instruction for purposes other than to exit a subroutine also performs branch and decrements the stack pointer.**
2. **Do not execute the RET instruction when the SP value is within 0x10 to 0x1F. If it is executed, a stack underflow or overflow occurs, rendering the jump address undefined.**
3. **Do not describe the RET instruction just after the inter-register transfer instruction to load from/store in STK or SP.**
4. **Do not describe the RET instruction just after a CALL instruction or CREG instruction that is combined with a conditional instruction.**

RETI

RETI

Branch

RETI

Name of instruction: **Interrupt return**

Mnemonic: **RETI**

Example RETI

Explanation Instruction to exit interrupt servicing. The operation of this instruction is the same as that of the RET instruction except that the interrupt enable flag returns to the last condition (PC ← STACK, SP ← SP – 1, returns the interrupt enable flag to the last condition).

Execution cycle 2 (When used with a conditional instruction: 3 when condition satisfied and 1 when condition not satisfied.)

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	Yes

Cautions

1. **Do not describe the RETI instruction as a repeat target instruction or within three instructions of the loop end.**
2. **Executing RETI instruction for purposes other than interrupt servicing also performs branch, decrements the stack pointer, and changes the interrupt enable flag value (for bits 15 to 13 of SR, and left shifts the EIR contents).**
3. **Do not execute the RETI instruction when the SP value is 0 or within 0x10 to 0x1F. If it is executed, a stack underflow or overflow occurs, rendering the jump address undefined.**
4. **Do not describe the RETI instruction just after the inter-register transfer instruction to load from/store in STK or SP.**
5. **Do not describe the RETI instruction just after a CALL instruction or CREG instruction that is combined with a conditional instruction.**
6. **Do not describe the RETI instruction at the start address of each interrupt source in the vector area.**

3.8 Hardware Loop Instructions

These are instructions to specify repeated execution of one or several instructions. The following hardware loop instructions are provided:

Repeat	(REP)
Loop	(LOOP)
Loop pop	(LPOP)

REP

REP

Hardware loop

REP count

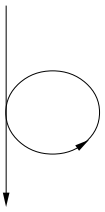
Name of instruction:	**Repeat**
Mnemonic:	**REP count** **count = {1 to 0x7FFF, rl}**

Example REP 0x1234

Explanation Instruction to repeat one instruction.
Repeats the succeeding instruction for the number of times specified by the count. During this time, the next instruction is repeatedly fetched.

- **Start**
  RC ← count
- **During repeat**
  PC ← PC
  RC ← RC – 1
- **End**
  PC ← PC + 1

Repeats the shaded instruction the count-specified times.



REP count

1 instruction word

When the repeat action is completed and the next instruction after the repeated instruction is executed, there will be no overhead.

REP

REP

Hardware loop

Execution cycle 2

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

Cautions

1. **A branch instruction (JMP, JREG, CALL, CREG, RET, RETI), or a REP, LOOP, LPOP, STOP, or HALT instruction must not be specified as a repeat target instruction.**
2. ★ **The loop instruction must not be described as a repeat target instruction.**
3. **During the execution and decoding cycles of the REP instruction and the repeated instruction, no interrupt is acknowledged.**
4. **The number of repetitions must not be set to 0 or 0x8000 or higher.**

LOOP

LOOP

Hardware loop

LOOP count {
2 to 255 instruction words
};

Name of instruction: **Loop**

Mnemonic: **LOOP count {**
2 to 255 instruction words
};
count = {1 to 0x7FFF, rl}

Remarks The " { " " } " should also be entered.

Example

```
LOOP 0x1234   {
  R1 = R0 + R4          R0 = *DP0++          R4 = *DP4++;
 *DP1++ = R1
                        };
```

Explanation Instruction to repeat two or more instructions. For one instruction repeat, use the REP instruction. Repeats the instructions from the succeeding instruction (loop starting address) to the instruction (loop ending address) just before the loop end pseudo-instruction (}) for the number of times specified by the count. The difference between the loop starting address and the loop ending address can be set within the range of 2 to 255. Operations during the LOOP instruction execution are as follows:

• Start

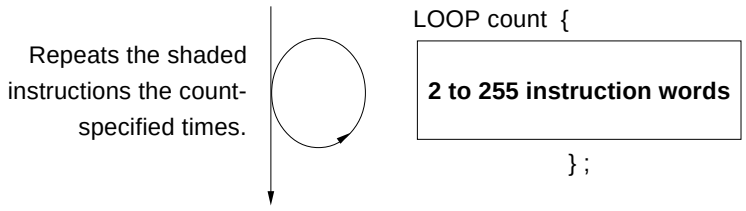
LSP ← LSP + 1
LSR1 ← LSA
LSR2 ← LEA
LSR3 ← LC
LSA ← PC + 1
LEA ← PC + RLE (loop end relative address)
LC ← count

• Loop end instruction fetch (during loop)

LC ← LC – 1
PC ← LSA

• Loop end instruction fetch (loop end)

PC ← PC + 1
LC ← LSR3
LEA ← LSR2
LSA ← LSR1
LSP ← LSP – 1



When branching from the loop ending address to the loop starting address, and when executing the instruction just behind the loop end pseudo-instruction after completion of the loop operation, there will be no overhead.
During the execution and decoding cycles of the LOOP instruction and the fetching cycle of the loop end instruction, no interrupt is acknowledged.

LOOP

LOOP

Hardware loop

Execution cycle 2

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

Cautions

1. Do not describe branch instructions (JMP, JREG, CALL, CREG, RET, RETI), or the REP, LOOP, LPOP, STOP, or HALT instruction, and a transfer instruction from LC within three instructions of the loop end. Care must be taken not to overlap the loop end with a loop end specified by another loop instruction.

Forbidden example 1:

```
LOOP  0x1234 {
          :
          :
      [              ]
      [              ]   } Branch instruction
      [              ]
                    };
```

Forbidden example 2:

```
LOOP  0x1234 {
          :
          :
      LOOP  0x2345 {
          :
          :
                    };
               };
```

★ **2. A transfer instruction from LC must not be described as the loop start.**

★ **3. Do not describe the repeat instruction within three instructions of the loop end.**

4. Do not execute the LOOP instruction when LSP is 4 to 7, or a loop stack underflow or overflow may occur.

5. The number of repetitions must not be set to 0 or 0x8000 or higher.

LPOP

LPOP

Hardware loop

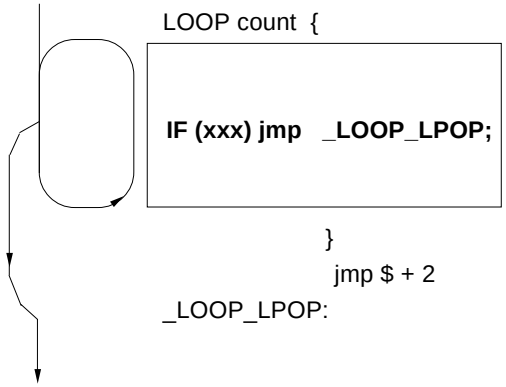
LPOP

Name of instruction:	**Loop pop**
Mnemonic:	**LPOP**

Example LPOP

Explanation Discards the present loop information and transfers the values of the loop stack indicated by the loop stack pointer to LC, LEA, and LSA. Decrements the value of the loop stack pointer. Operations during LOOP instruction execution are as follows:

LC ← LSR3
LEA ← LSR2
LSA ← LSR1
LSP ← LSP – 1



LPOP

LPOP

Hardware loop

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

Cautions

1. **Do not describe the LPOP instruction within three instructions of the loop end.**
2. **The LPOP instruction should not be described as a repeat target instruction.**
3. **Do not execute the LPOP instruction when LSP is 0 or 5 to 7. If it is executed, loop stack underflow or overflow occurs.**
4. **The LPOP instruction does not terminate a loop. It only decreases the loop stack pointer (LSP).**

3.9 Control Instructions

These are instructions to specify program control.
The following control instructions are provided:

No operation	(NOP)
Halt	(HALT)
Stop	(STOP)
Forget interrupt	(FINT)
Condition	(COND)

NOP

NOP

Control

NOP

Name of instruction: **No operation**

Mnemonic: **NOP**

Example NOP

Explanation Only updates the value of PC and executes nothing. This instruction is used to consume one instruction cycle for timing.

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

HALT

HALT

Control

HALT

Name of instruction: **Halt**

Mnemonic: **HALT**

Example HALT

Explanation Stops the operation of the μPD77016 Family.

Remarks 1. In HALT mode, the pins, registers, and internal memory of the device retain the statuses immediately before the HALT mode is set (refer to **μPD7701x Family User's Manual Architecture** or **μPD77111 Family User's Manual Architecture**).

2. This mode is released by using an external/internal interrupt (which is not masked) or hardware reset.

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

Caution **Do not describe the HALT instruction as a repeat target instruction or within three instructions of the loop end.**

★STOP

STOP

Control

STOP

Name of instruction: **Stop**

Mnemonic: **STOP**

Example STOP

Explanation Stops operation of the clock circuit and PLL of the μPD77016 Family (excluding the μPD77016).

Remarks 1. In the STOP mode, the device pins retain the statuses immediately before the STOP mode is set (refer to **μPD7701x Family User's Manual Architecture** or **μPD77111 Family User's Manual Architecture**).

2. The STOP mode can be released by means of hardware reset or $\overline{\text{WAKEUP}}$ pin input[Note].

Note Releasing the STOP mode by means of $\overline{\text{WAKEUP}}$ pin input is available only for the μPD77111 Family.

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

Cautions

1. **Do not describe the STOP instruction as a repeat target instruction or within three instruction of the loop end.**
2. **In STOP mode, the output pin statuses are initialized by hardware reset. The output pin statuses are not assured and go into a high-impedance state until the PLL becomes stable.**
3. **A NOP instruction must be inserted immediately before the STOP instruction for releasing the STOP mode by means of $\overline{\text{WAKEUP}}$ pin input in μPD77111 Family devices.**

Example)
nop; Required
stop; STOP instruction assuming release by means of $\overline{\text{WAKEUP}}$ pin input

FINT

FINT

Control

FINT

Name of instruction: **Forget interrupt**

Mnemonic: **FINT**

Example FINT

Explanation Discards all existing interrupt requests.
This instruction is used to synchronize the program with the generation of the specified interrupt factor when the servicing order of interrupts to be processed is limited.

Execution cycle 1

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	No	No	No	No	No

COND

COND

IF (ro cond)

Name of instruction: **Condition**

Mnemonic: **IF (ro cond)**

Conditions that can be described in cond (condition):

EVER:	Unconditional (top description not necessary)
==0:	= 0
!=0:	≠ 0
>0:	> 0
<0:	< 0
<=0:	≤ 0
>=0:	≥ 0
==EX:	Extension (bits 39 to 31 are a mixture of 0 and 1)
!=EX:	Without extension (bits 39 to 31 are all 0 or all 1)

Example IF (R1 >= 0) R0 = R0 + 1

Explanation Judges the condition.

The top specified general-purpose register is evaluated as 40-bit data represented with two's complement. Executes the instruction described in the same statement if the condition is true. If the condition is false, there is no operation.

Execution cycle Number of cycles of the instruction described simultaneously (when the condition was true), or 1 (when the condition was false)

Instructions that can be described concurrently

Trinomial	Binomial	Monomial	Parallel load/store	Inter-register transfer	Branch	Conditional
No	No	Yes	No	Yes	Yes	No

APPENDIX A CLASSIFICATION OF INSTRUCTION WORDS

This section describes the features of each instruction category. Table A-1 lists the formats of instruction words.

Table A-1. Formats of Instruction Words

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	op1			opcode				op2			op3			dx	dy	dpx			modix			rx (rxh)			dpy			modiy			ry (ryh)
0	1	1	1	opcode				op1			op2			dx	dy	dpx			modix			rx (rxh)			dpy			modiy			ry (ryh)
0	1	1	0	opcode				op1			op2			–	–	–	–	–	–	–	–	–	–	–	cond				top		
0	1	0	1	opcode				op1			op2			–	–	imm															
0	1	0	0	1	1	sufx			sufy			–	–	dx	dy	dpx			modix			regx			dpy			modiy			regy
0	1	0	0	1	0	reg			suf			xy	–	–	d	direct															
0	1	0	0	0	1	reg			suf			dp			d	imm															
0	0	1	1	1	1	source1			0	dest1					0	–	–	–	–	–	–	–	–	–	cond				top		
0	0	1	1	1	1	source1			1	dest1					0	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
0	0	1	1	1	1	dest2			0	source2					1	–	–	–	–	–	–	–	–	–	cond				top		
0	0	1	1	1	1	dest2			1	source2					1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
0	0	1	1	1	0	–	–	–	dest1						0	imm															
0	0	1	1	1	0	dest2			–	–	–	–	–	–	1	imm															
0	0	1	0	1	1	jc	–	–	Relative address																cond				top		
0	0	1	0	1	0	jc	–	–	–	–	–	dp			–	–	–	–	–	–	–	–	–	–	cond				top		
0	0	1	0	0	1	rt	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	cond				top		
0	0	0	1	1	1	–	–	–	–	–	–	–	–	–	–	0	imm (Repeated times)														
0	0	0	1	1	0	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	rl
0	0	0	1	0	1	–	–	Loop end relative address								0	imm (Repeated times)														
0	0	0	1	0	0	–	–	Loop end relative address								–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	rl
0	0	0	0																												

Remark –: Unused bits

A.1 Three-Operand Instructions

A three-operand instruction uses three operands when an operation is executed between the MAC and ALU. Trinomial and binomial instructions fall into this category, and are executed in the following formats:

- **Format of trinomial instruction**

  OP3 = OP3 opm op1 OP1 op2 OP2;

  where,

  OP1: 1st operand (operation operand)
  OP2: 2nd operand (operation operand)
  OP3: 3rd operand (operation operand and result operand)
  opm: Locally modifies OP3 under MSFT operation direction. Either "no direction", ">>1 (1-bit arithmetic right shift)", or ">>16 (16-bit arithmetic right shift)".
  op1: ALU operation direction. "+" or "–".
  op2: MAC operation direction. Always "*".

- **Format of binomial instruction**

  (1) OP3 = OP1 op OP2;
  (2) OP3 = oplt (OP1, OP2);

  where,

  OP1: 1st operand (operation operand)
  OP2: 2nd operand (operation operand)
  OP3: 3rd operand (result operand)
  op: ALU or MAC operation direction
  oplt: ALU operation direction. Always "LT".

  All binomial instructions, except LT (Less Than), are described in format (1). Only the LT instruction is described in format (2).

With a three-operand instruction, parallel load and store instructions can be described simultaneously, so that data can be exchanged between the X or Y memory and a general-purpose register while an operation of the ALU or MAC is executed.

Figure A-1 shows the instruction word format of three-operand instructions.

Figure A-1. Three-Operand Instruction Format

XY parallel load/store (operation register ⟷ memory) can be described simultaneously.

op [[rx = *dp (*dp++, *dp– –, *dp##, *dp%%, *!dp##)]
[*dp (*dp++, *dp– –, *dp##, *dp%%, *!dp##)] = rxh]] [[ry = *dp (*dp++, *dp– –, *dp##, *dp%%, *!dp##)]
[*dp (*dp++, *dp– –, *dp##, *dp%%, *!dp##)] = ryh]]

31	30–28	27–24	23–21	20–18	17	16	15–14	13–11	10–8	7–6	5–3	2–0
1	op1	opcode	op2	op3	dx	dy	dpx	modix	rx (rxh)	dpy	modiy	ry (ryh)

opcode

op3 = lt (op1,op2)
op3 = op1h*op2h
op3 = op3 + op1h*op2h
op3 = op3 – op1h*op2h
op3 = op3 + op1h*op2l
op3 = op3 + op1l*op2l
op3 = op3 >> 1 + op1h*op2h
op3 = op3 >> 16 + op1h*op2h
op3 = op1 + op2
op3 = op1 – op2
op3 = op1 & op2
op3 = op1 | op2
op3 = op1 ^ op2
op3 = op1 sra op2l
op3 = op1 srl op2l
op3 = op1 sll op2l

op1, op2, op3

r0 (r0h, r0l)
r1 (r1h, r1l)
r2 (r2h, r2l)
r3 (r3h, r3l)
r4 (r4h, r4l)
r5 (r5h, r5l)
r6 (r6h, r6l)
r7 (r7h, r7l)

A.2 Two-Operand Instructions

Only the monomial instructions of the ALU (including NOP) are classified as two-operand instructions.

- **Format of monomial instruction**

(1) opn
(2) OP2 = OP1 op 1;
(3) OP2 opu = OP1;
(4) OP2 = opl OP1;
(5) OP2 = opf (OP1);
where

OP1: 1st operand (operation operand)
OP2: 2nd operand (result operand)
opn: "NOP" only
op: Operation direction to ALU. "+" or "–".
opu: Operation instruction to ALU. Refer to the figure below.
opl: Operation direction to ALU. "–" or " ~ ".
opf: Operation direction to ALU. "CLIP", "ROUND", or "EXP".

Caution "NOP" is a direction to the ALU and does not mean that the entire instruction word is "NOP".

With a two-operand instruction, either of a parallel load/store instruction or a conditional instruction can be described simultaneously so that data can be exchanged between the X or Y memory and a general-purpose register while the ALU or MAC is executing an operation, or so that an ALU or MAC operation can be executed when a given condition is satisfied.
Figure A-2 shows the instruction word format of two-operand instructions.

Figure A-2. Two-Operand Instruction Format

XY parallel load/store (operation register ⟷ memory) can be described simultaneously.

op [[rx = *dp (*dp++, *dp– –, *dp##, *dp%%, *!dp##)]
[*dp (*dp++, *dp– –, *dp##, *dp%%, *!dp##)] = rxh]] [[ry = *dp (*dp++, *dp– –, *dp##, *dp%%, *!dp##)]
[*dp (*dp++, *dp– –, *dp##, *dp%%, *!dp##)] = ryh]]

31 28	27 24	23 21	20 18	17	16	15 14	13 11	10 8	7 6	5 3	2 0
0 1 1 1	opcode	op1	op2	dx	dy	dpx	modix	rx (rxh)	dpy	modiy	ry (ryh)

Condition can be described simultaneously.

[if (top cond)] op

31 28	27 24	23 21	20 18	17 7	6 3	2 0
0 1 1 0	opcode	op1	op2	– – – – – – – – – –	cond	top

opcode

nop
clr (op2)
op2 = op1 + 1
op2 = op1 – 1
op2 = abs (op1)
op2 = ~op1
op2 = –op1
op2 = clip (op1)
op2 = round (op1)
op2 = exp (op1)
op2 = op1
op2 / = op1
op2 + = op1
op2 – = op1

op1, op2

r0
r1
r2
r3
r4
r5
r6
r7

A.3 Immediate Value Operation Instructions

Only the immediate binomial operation instruction of the ALU falls into the category of immediate value operation instructions.

- Binomial operation instruction (immediate operation)

  OP2 = OP1 op imm;

  where,

  OP1: 1st operand (operation operand)

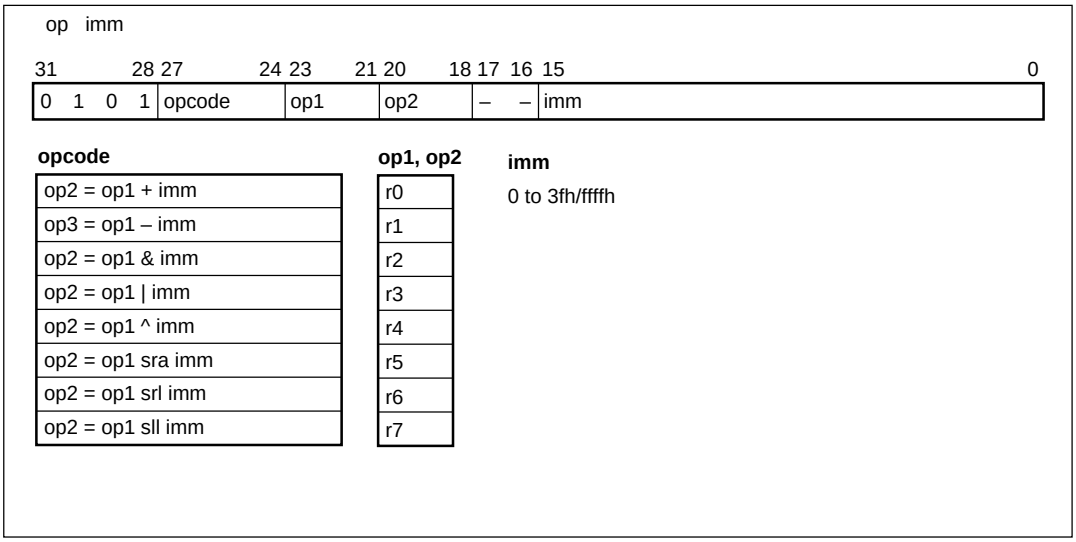
  OP2: 2nd operand (result operand)

  op: ALU operation direction. Refer to Figure A-3 below.

  imm: 16-bit immediate data

No other instruction can be described simultaneously with an instruction in this category. Figure A-3 shows the instruction word format of immediate value operation instructions.

Figure A-3. Immediate Value Operation Instruction Format



A.4 Load/Store Instructions

Load/store instructions can be classified into the following four types:

- **Parallel load/store instruction**

  This instruction loads or stores 16-bit data between X or Y memory and a general-purpose register. When accessing the memory with this instruction, only the indirect addressing mode using DPn (n = 0 to 7) is valid. The important point about this instruction is that it can be described with any of the following operation instructions:

  - Trinomial operation instructions
  - Binomial operation instructions (except immediate binomial operation)
  - Monomial operation instructions

  Therefore, preparing data in a general-purpose register for the next operation while executing an operation can be executed with a single instruction.

- **Partial load/store instruction**

  This instruction loads or stores 16-bit data between the X or Y memory and a general-purpose register. When accessing the memory with this instruction, only the indirect addressing mode using DPn (n = 0 to 7) is valid. The important points about this instruction are as follows:

  - Any part of a general-purpose register can be specified for loading or storing data.
  - 16-bit data can be loaded simultaneously from the X and Y memories (32-bit data load) to two parts of a general-purpose register.
  - No other instruction can be described with this instruction.

- **Direct addressing load/store instruction**

  This instruction directly specifies the type of data memory and an address in the instruction word. The important points about this instruction are as follows:

  - The type of data memory and an address can be directly described in the instruction.
  - Data can be loaded (from memory to a general-purpose register) to any one or two parts (Rn, RnEH: n = 0 to 7) of a general-purpose register.
  - Data can be stored (from a general-purpose register to memory) from any part of a general-purpose register.
  - The X and Y memories cannot be accessed at the same time.
  - No other instruction can be described simultaneously with this instruction.

- **Immediate modify load/store instruction**

  This instruction employs the indirect addressing mode using DPn. However, DPn is modified by immediate data, instead of DNn, after access. The important points about this instruction are as follows:

  - Data modifying DPn is directly described in the instruction.
  - Data can be loaded (from memory to a general-purpose register) to any one or two parts of a general-purpose register (Rn, RnEH: n = 0 to 7).
  - Data can be stored (from a general-purpose register to memory) in any part of a general-purpose register.
  - The X and Y memories cannot be accessed at the same time.
  - No other instruction can be described simultaneously with this instruction.

Figure A-4 shows the instruction word format of load/store instructions.

Figure A-4. Load/Store Instruction Format (1/2)

XY parallel load/store (operation register ⟷ memory)

[op] [x load/store nop
rx = *dp (*dp++, *dp– –, *dp##, *dp%%, *!dp##)
*dp (*dp++, *dp– –, *dp##, *dp%%, *!dp##) = rxh] [y load/store nop
ry = *dp (*dp++, *dp – –, *dp##, *dp%%, *!dp##)
*dp (*dp++, *dp– –, *dp##, *dp%%, *!dp##) = ryh]

31	30 28	27 24	23 21	20 18	17	16	15 14	13 11	10 8	7 6	5 3	2 0
1	op1	opcode	op2	op3	dx	dy	dpx	modix	rx (rxh)	dpy	modiy	ry (ryh)
0 1 1 1		opcode	op1	op2	dx	dy	dpx	modix	rx (rxh)	dpy	modiy	ry (ryh)

dx, dy

*dp = rxh (ryh)
rx (ry) =*dp

rx (rxh), ry (ryh)

r0 (r0h)
r1 (r1h)
r2 (r2h)
r3 (r3h)
r4 (r4h)
r5 (r5h)
r6 (r6h)
r7 (r7h)

dpx, dpy

dp0, dp4
dp1, dp5
dp2, dp6
dp3, dp7

modix, modiy

nop
*dpx (y)
*dpx (y)++
*dpx (y)– –
*dpx (y)##
*dpx (y)%%
*idpx (y)##

Partial load/store

[x load/store nop
regxs = *dp (*dp++, *dp– –, *dp##, *dp%%, *!dp##)
*dp (*dp++, *dp– –, *dp##, *dp%%, *!dp##) = regxs] [y load/store nop
regys = *dp (*dp++, *dp– –, *dp##, *dp%%, *!dp##)
*dp (*dp++, *dp– –, *dp##, *dp%%, *!dp##) = regys]

31 28	27 26	25 23	22 20	19 18	17	16	15 14	13 11	10 8	7 6	5 3	2 0
0 1 0 0	1 1	sufx	sufy	– –	dx	dy	dpx	modix	regx	dpy	modiy	regy

sufx + dx, sufy + dy

*dp = rx (y) l
*dp = rx (y) h
*dp = rx (y) e
rx (y) l = *dp
rx (y) h = *dp
rx (y) e = *dp
rx (y) eh = *dp
rx (y) = *dp

regx, regy

r0suf
r1suf
r2suf
r3suf
r4suf
r5suf
r6suf
r7suf

dpx, dpy

dp0, dp4
dp1, dp5
dp2, dp6
dp3, dp7

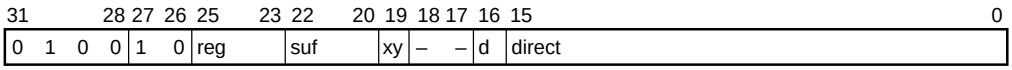
modix, modiy

nop
*dpx (y)
*dpx (y)++
*dpx (y)– –
*dpx (y)##
*dpx (y)%%
*idpx (y)##

Figure A-4. Load/Store Instruction Format (2/2)

Direct addressing load/store

[regs = *direct: x (y)
*direct: x (y) = regs]



suf + d
*dp = regl
*dp = regh
*dp = rege
regl = *dp
regh = *dp
rege = *dp
regeh = *dp
reg = *dp

reg
r0suf
r1suf
r2suf
r3suf
r4suf
r5suf
r6suf
r7suf

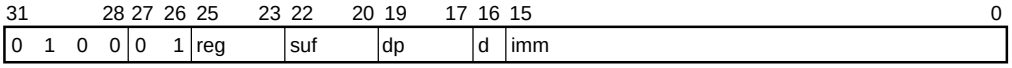
xy
x
y

direct

0 to ffffh

Immediate value modify load/store

[rs = *dp##imm
*dp##imm = rs]



suf + d
*dp##imm = regl
*dp##imm = regh
*dp##imm = rege
regl = *dp##imm
regh = *dp##imm
rege = *dp##imm
regeh = *dp##imm
reg = *dp##imm

reg
r0suf
r1suf
r2suf
r3suf
r4suf
r5suf
r6suf
r7suf

dp
dp0
dp1
dp2
dp3
dp4
dp5
dp6
dp7

im

0 to ffffh

A.5 Inter-Register Transfer Instruction

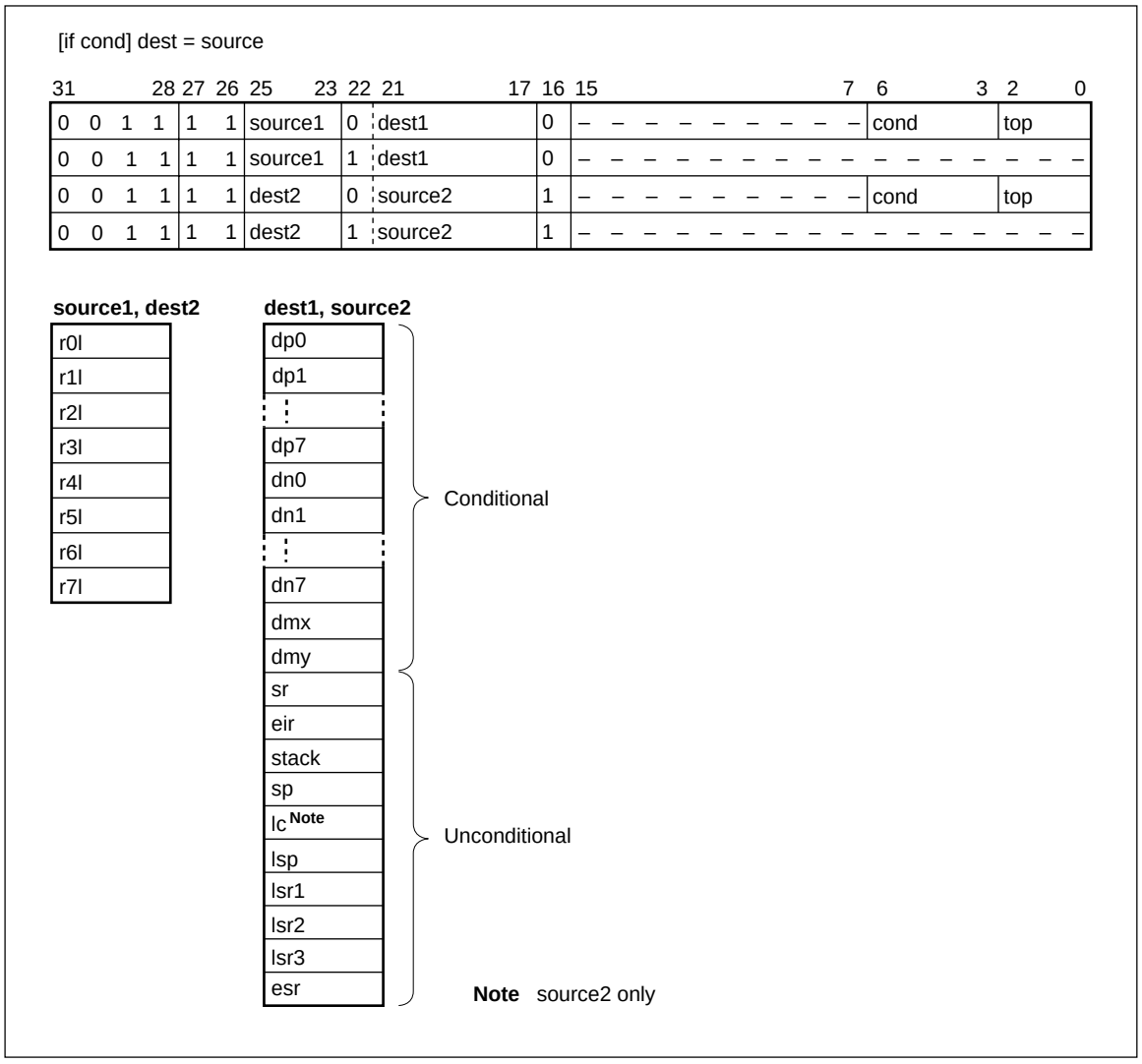
This instruction transfers data between a general-purpose register and a register connected to the main bus. The important points about this instruction are as follows:

- A general-purpose register must always be described at one end of transfer (i.e., as destination or source).
- Any register connected to the main bus can be specified as the destination or source.
- Because a conditional instruction can be described at the same time, conditional transfer can be executed.

Remark This instruction cannot be used to transfer data between two general-purpose registers. Use the monomial instruction PUT to transfer data between general-purpose registers.

Figure A-5 shows the instruction word format of the inter-register transfer instruction.

Figure A-5. Inter-Register Transfer Instruction Format



A.6 Immediate Value Set Instruction

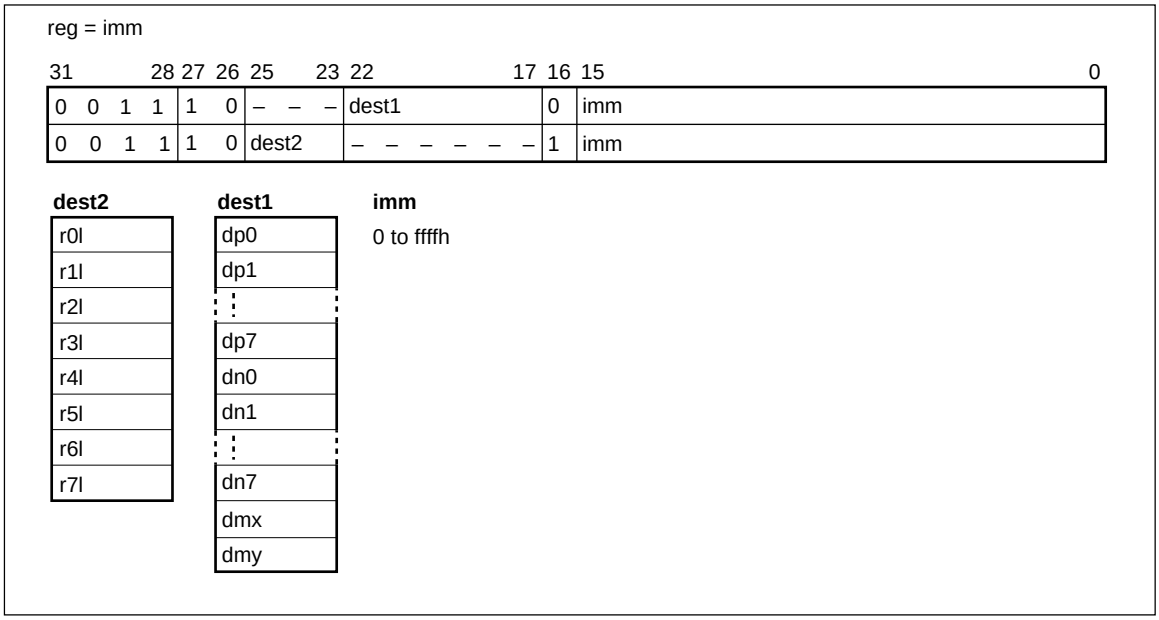
This instruction sets immediate data to a general-purpose register or addressing register.

The important points about using this instruction are as follows:

- Specify the L part (R7L to R0L), DPn, PNn, or DMX/Y of a general-purpose register.
- No other instruction can be described at the same time.

Figure A-6 shows the instruction word format of the immediate value setting instruction.

Figure A-6. Immediate Value Set Instruction Format



A.7 Branch Instructions

Branch instructions can be classified into the following three types:

- **Relative address jump/relative address call**

  Instructions of this type implement branching by adding (subtracting) the difference from the targeted address to (from) the current PC value. The difference is given as a 16-bit two's complement; therefore, execution can be branched over the entire 64-Kword instruction memory space. The important points about this instruction are as follows:

  - The operand is a 16-bit two's complement that indicates a difference from the branch destination address. (Note, however, that the numeric value of the targeted address is described directly or as a label in assembly language.)
  - A conditional instruction can be described at the same time.

- **Indirect jump/indirect call**

  The branch destination address is given by using a register (DPn). At this time, the specified value of DPn is directly set to the PC. The important points about this instruction are as follows:

  - The operand is DPn (n: 0 to 7).
  - A conditional instruction can be described at the same time.

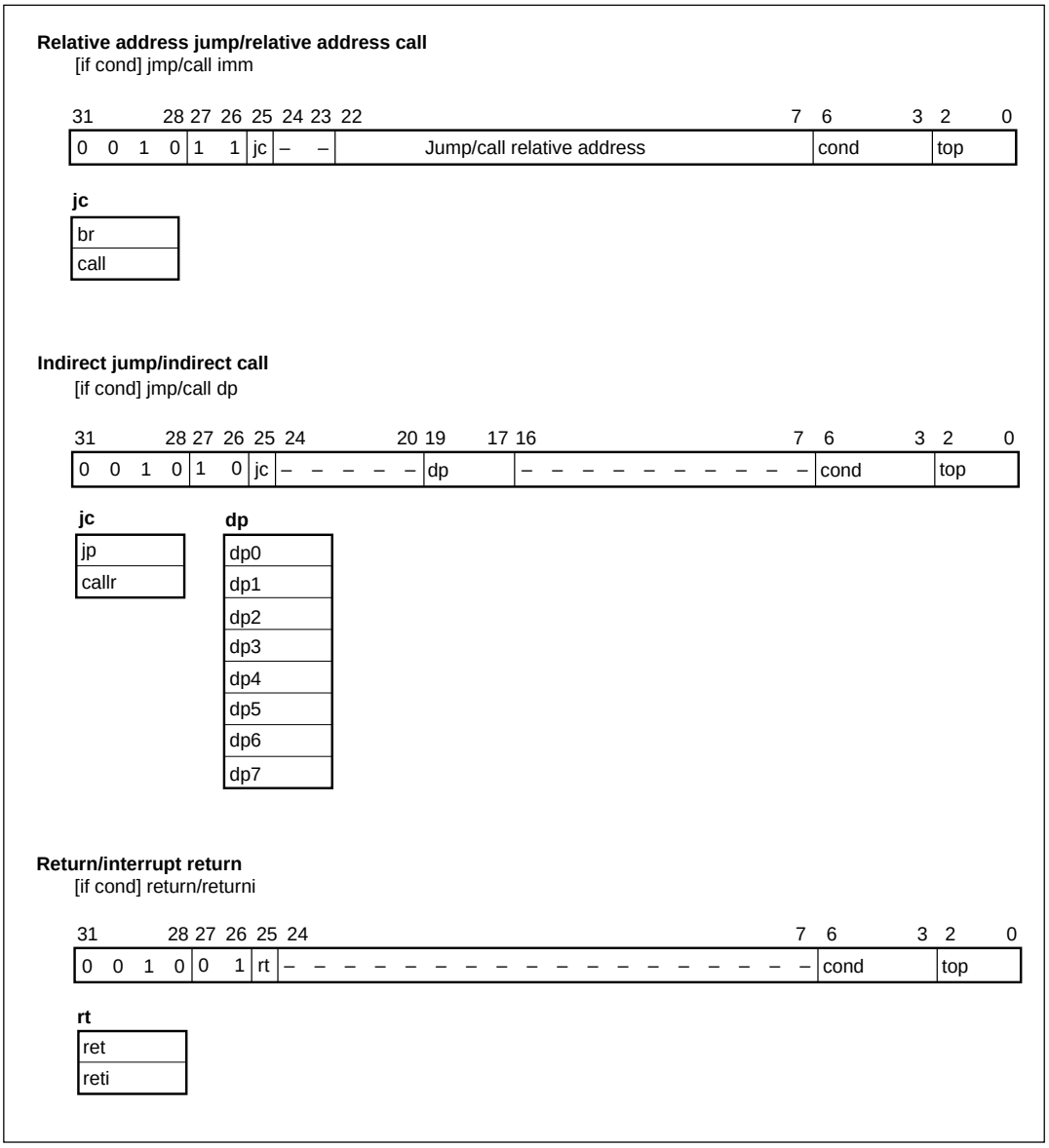
- **Return/interrupt return**

  The branch destination address is the difference from the data saved to STK (stack) and is added to (subtracted from) the current PC value.

  - There is no explicit operand, but STK is specified as an implicit operand. The value of STK is added to (subtracted from) the current PC value as a two's complement.
  - A conditional instruction can be described at the same time.

Figure A-7 shows the instruction word format of branch instructions.

Figure A-7. Branch Instruction Format



A.8 Hardware Loop Instructions

Hardware loop instructions can be classified into the following two types:

- **Repeat instruction**

  This instruction repeatedly executes a specified instruction[Note]. The important points about this instruction are as follows:

  - The number of times the specified instruction is to be repeated can be described directly in the repeat instruction, or as the L part (RnL: n = 0 to 7) of a general-purpose register.
  - No other instruction can be described at the same time.

- **Loop instruction**

  This instruction repeatedly executes a group of instructions. One group can consist of two to 255 instructions (see **Note**). The important points about this instruction are as follows:

  - The number of times the specified instructions are to be repeated can be described directly in the repeat instruction, or as the L part (RnL: n = 0 to 7) of a general-purpose register.
  - No other instruction can be described at the same time.

Note "Instruction" in this case is in units of one instruction word, and does not mean a function instruction, which is a field element in an instruction word.

Figure A-8 shows the instruction word format of hardware loop instructions.

Figure A-8. Hardware Loop Instruction Format

Repeat instruction

repeat count

31 28	27 26	25 16	15	14 0
0 0 0 1	1 1	– – – – – – – – – –	0	imm

imm

1to7fffh

repeat rl

31 28	27 26	25 3	2 0
0 0 0 1	1 0	– –	rl

rl

r0l
r1l
r2l
r3l
r4l
r5l
r6l
r7l

Loop instruction

loop imm { }

31 28	27 26	25 24	23 16	15	14 0
0 0 0 1	0 1	– –	Loop end relative address	0	imm

imm

1to7fffh

loop rl { }

31 28	27 26	25 24	23 16	15 3	2 0
0 0 0 1	0 0	– –	Loop end relative address	– – – – – – – – – – – – –	rl

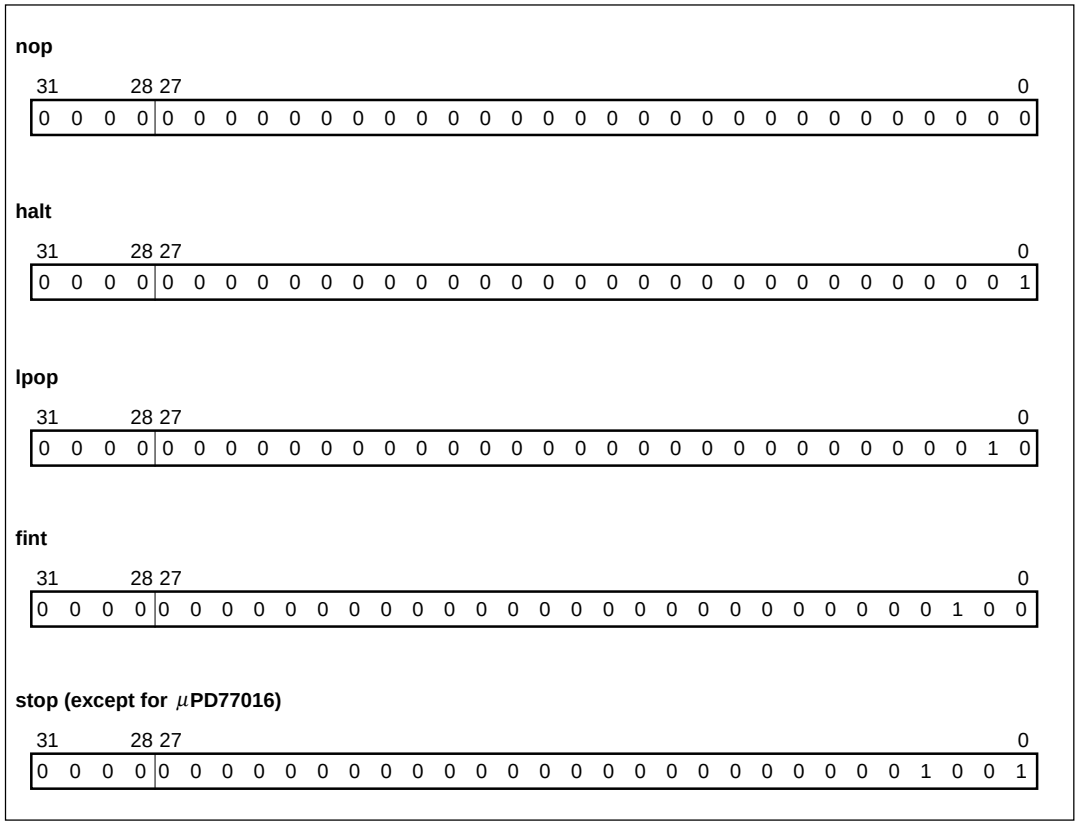
rl

r0l
r1l
r2l
r3l
r4l
r5l
r6l
r7l

A.9 CPU Control Instructions

Instructions of this category do not use an operand. No other instruction can be described with these instructions. Figure A-9 shows the instruction word format of CPU control instructions.

Figure A-9. CPU Control Instruction Format



A.10 Conditional Instructions

A conditional instruction is meaningful when used in combination with other function instructions.

- **Description format**

  IF (ro cond) other instruction;

  where

  ro: any general-purpose register (R7 to R0)

  cond: condition. The following conditions can be described:

  ==0: Judges =0

  !=0: Judges ≠0

  >0: Judges >0

  <0: Judges <0

  >=0: Judges ≥0

  <=0: Judges ≤0

  ==ex: Extension (bits 39 to 31 are mixed 0 and 1)

  !=ex: Without extension (bits 39 to 31 are all 0 or all 1)

- **Other instructions that can be described in combination**

  Any of the following instructions can be described in combination with the conditional instruction:

  - Monomial operation instruction
  - Inter-register transfer instruction
  - Branch instruction

Figure A-10 shows the instruction word format of conditional instructions.

Figure A-10. Conditional Instruction Format

Conditional operation

if (cond) op

31 28	27 24	23 21	20 18	17 7	6 3	2 0
0 1 1 0	opcode	op1	op2	– – – – – – – – – –	cond	top

Register-to-register transfer

if (cond) dest = source

31 28	27	26	25 23	22	21 17	16	15 7	6 3	2 0
0 0 1 1	1	1	source1	0	dest1	0	– – – – – – – –	cond	top
0 0 1 1	1	1	dest2	0	source2	1	– – – – – – – –	cond	top

Relative address jump/relative address call

if (cond) jmp/call imm

31 28	27	26	25	24 23	22 7	6 3	2 0
0 0 1 0	1	1	jc	– –	Branch/call relative address	cond	top

Indirect jump/indirect call

if (cond) jmp/call dp

31 28	27	26	25	24 20	19 17	16 7	6 3	2 0
0 0 1 0	1	0	jc	– – – – –	*dp*	– – – – – – – – – –	cond	top

Return/interrupt return

if (cond) return/returni

31 28	27	26	25	24 7	6 3	2 0
0 0 1 0	0	1	rt	– – – – – – – – – – – – – – – – – – – –	cond	top

top

r0
r1
r2
r3
r4
r5
r6
r7

cond

ever
==0
!=0
>0
<=0
>=0
<0
==ex
!=ex

APPENDIX B INSTRUCTION SETS

Group	Instruction Name	Mnemonic	Operation	Simultaneously Describable									Flag	Page
				Trinomial	Binomial	Monomial	Load/store	Transfer	Immediate	Branch	Loop	Condition	OV	
Trinomial	Multiply add	ro = ro + rh*rh'	ro ← ro + rh*rh'				✓						1	24
	Multiply subtract	ro = ro – rh*rh'	ro ← ro – rh*rh'				✓						1	26
	Sign-unsign multiply add	ro = ro + rh*rl (rl is positive integer)	ro ← ro + rh*rl				✓						1	28
	Unsign-unsign multiply add	ro = ro + rl*rl' (rl and rl' are positive integer)	ro ← ro + rl*rl'				✓						1	30
	1-bit shift multiply add	ro = (ro >> 1) + rh*rh'	ro ← $\frac{ro}{2}$ + rh*rh'				✓						1	32
	16-bit shift multiply add	ro = (ro >> 16) + rh*rh'	ro ← $\frac{ro}{2^{16}}$ + rh*rh'				✓						–	34
Binomial	Multiply	ro = rh*rh'	ro ← rh*rh'				✓						–	37
	Add	ro" = ro + ro'	ro" ← ro + ro'				✓						1	38
	Immediate add	ro' = ro + imm	ro' ← ro + imm (imm ≠ 1)										1	39
	Sub	ro" = ro – ro'	ro" ← ro – ro'				✓						1	40
	Immediate sub	ro' = ro – imm	ro' ← ro – imm (imm ≠ 1)										1	41
	Arithmetic right shift	ro' = ro SRA rl	ro' ← ro >> rl				✓						–	42
	Immediate arithmetic right shift	ro' = ro SRA imm	ro' ← ro >> imm										–	44
	Logical right shift	ro' = ro SRL rl	ro' ← ro >> rl				✓						–	45
	Immediate logical right shift	ro' = ro SRL imm	ro' ← ro >> imm										–	46
	Logical left shift	ro' = ro SLL rl	ro' ← ro << rl				✓						–	47
	Immediate logical left shift	ro' = ro SLL imm	ro' ← ro << imm										–	48
	AND	ro" = ro & ro'	ro" ← ro & ro'				✓						–	49
	Immediate AND	ro' = ro & imm	ro' ← ro & imm										–	50
	OR	ro" = ro | ro'	ro" ← ro | ro'				✓						–	51
	Immediate OR	ro' = ro | imm	ro' ← ro | imm										–	52
	Exclusive OR	ro" = ro ^ ro'	ro" ← ro ^ ro'				✓						–	53

Status of overflow flag (OV)

–: Not affected
1: Set to 1 when overflow occurs

Group	Instruction Name	Mnemonic	Operation	Simultaneously Describable									Flag	Page
				Trinomial	Binomial	Monomial	Load/store	Transfer	Immediate	Branch	Loop	Condition	OV	
Binomial	Immediate exclusive OR	ro' = ro ^ imm	ro' ← ro ^ imm										–	54
	Less than	ro" = LT (ro, ro')	if (ro < ro') {ro"← 0x0000000001} else {ro"← 0x0000000000}				✓						–	55
Monomial	Clear	CLR (ro)	ro ← 0x0000000000				✓					✓	–	58
	Increment	ro' = ro + 1	ro' ← ro + 1				✓					✓	1	59
	Decrement	ro' = ro – 1	ro' ← ro – 1				✓					✓	1	60
	Absolute value	ro' = ABS (ro)	if (ro < 0) {ro' ← – ro} else {ro' ← ro}				✓					✓	1	61
	One's complement	ro' = ~ ro	ro' ← ~ ro				✓					✓	–	62
	Two's complement	ro' = – ro	ro' ← – ro				✓					✓	1	63
	Clip	ro' = CLIP (ro)	if (ro > 0x007FFFFFFF) {ro'← 0x007FFFFFFF} else if (ro < 0xFF80000000) {ro'← 0xFF80000000} else {ro'← ro}				✓					✓	1	64
	Round	ro' = ROUND (ro)	if (ro > 0x007FFF0000) {ro'← 0x007FFF0000} else if (ro > 0xFF80000000) {ro'← 0xFF80000000} else {ro'← (ro + 0x8000) & 0xFFFFFF0000}				✓					✓	1	65
	Exponent	ro' = EXP (ro)	ro' ← $\log_2\left(\frac{1}{ro}\right)$				✓					✓	–	66
	Put	ro' = ro	ro' ← ro				✓					✓	–	68
	Accumulative add	ro' + = ro	ro' ← ro' + ro				✓					✓	1	69
	Accumulative subtract	ro' – = ro	ro' ← ro' – ro				✓					✓	1	70

Status of overflow flag (OV)

–: Not affected
1: Set to 1 when overflow occurs

Group	Instruction Name	Mnemonic	Operation	Simultaneously Describable									Flag	Page
				Trinomial	Binomial	Monomial	Load/store	Transfer	Immediate	Branch	Loop	Condition	OV	
Monomial	Divide	ro'/ = ro	if (sign (ro') = = sign (ro)) {ro' ← (ro' – ro) << 1} else {ro' ← (ro' + ro) << 1} if (sign (ro') = = 0) {ro' ← ro' + 1}				✓					✓	1	72
Load/store	Parallel load/store	ro = *dpx_mod ro' = *dpy_mod	ro ← *dpx, ro' ← *dpy	✓	✓	✓							–	75
		ro = *dpx_mod *dpy_mod = rh	ro ← *dpx, *dpy ← rh											
		*dpx_mod = rh ro = *dpy_mod	*dpx ← rh, ro ← *dpy											
		*dpx_mod = rh *dpy_mod = rh'	*dpx ← rh, *dpy ← rh'											
	Section load/store	dest = *dpx_mod dest' = *dpy_mod	dest ← *dpx, dest' ← *dpy										–	79
		dest = *dpx_mod *dpy_mod = source	dest ← *dpx, *dpy ← source											
		*dpx_mod = source dest = *dpy_mod	*dpx ← source, dest ← *dpy											
		*dpx_mod = source *dpy_mod = source'	*dpx ← source, dpy ← source'											
	Direct addressing load/store	dest = *addr	dest ← *addr										–	85
		*addr = source	*addr ← source											
	Immediate value index load/store	dest = *dp_imm	dest ← *dp										–	89
		*dp_imm = source	*dp ← source											
Transfer	Inter-register transfer	dest = rl	dest ← rl									✓	–	94
		rl = source	rl ← source											

Status of overflow flag (OV)

–: Not affected
1: Set to 1 when overflow occurs

Group	Instruction Name	Mnemonic	Operation	Simultaneously Describable									Flag	Page
				Trinomial	Binomial	Monomial	Load/store	Transfer	Immediate	Branch	Loop	Condition	OV	
Immediate set	Immediate value set	rl = imm (imm = 0-0xFFFF)	rl ← imm										–	97
		dp = imm (imm = 0-0xFFFF)	dp ← imm											
		dn = imm (imm = 0-0xFFFF)	dn ← imm											
		dm = imm (imm = 1-0xFFFF)	dm ← imm											
Branch	Jump	JMP imm	PC ← imm									✓	–	100
	Register indirect jump	JMP dp	PC ← dp									✓	–	101
	Subroutine call	CALL imm	SP ← SP + 1 STK ← PC + 1 PC ← imm									✓	–	102
	Register indirect subroutine call	CALL dp	SP ← SP + 1 STK ← PC + 1 PC ← dp									✓	–	104
	Return	RET	PC ← STK SP ← SP – 1									✓	–	105
	Interrupt return	RETI	PC ← STK STK ← SP – 1 Restores IE flag.									✓	–	106
Hardware loop	Repeat	REP count	Start RC ← count RF ← 0 Repeat PC ← PC RC ← RC – 1 End PC ← PC + 1 RF ← 1										–	108

Status of overflow flag (OV)

–: Not affected
1: Set to 1 when overflow occurs

Group	Instruction Name	Mnemonic	Operation	Simultaneously Describable									Flag	Page
				Trinomial	Binomial	Monomial	Load/store	Transfer	Immediate	Branch	Loop	Condition	OV	
Hardware loop	Loop	LOOP count (For repeating instructions of 2 or more lines)	Start RC ← count RF ← 0 Repeat PC ← PC RC ← RC – 1 End PC ← PC + 1 RF ← 1										–	110
	Loop pop	LPOP	LC ← LSR3 LEA ← LSR2 LSA ← LSR1 LSP ← LSP – 1										–	112
Control	No operation	NOP	PC ← PC + 1										–	115
	Halt	HALT	CPU stops.										–	116
	Stop	STOP	CPU, PLL, OSC stop.										–	117
	Forget interrupt	FINT	Discards interrupt requests.										–	118
Condition	Condition	IF (ro cond)	Conditional judge			✓		✓		✓			–	119

Status of overflow flag (OV)

–: Not affected
1: Set to 1 when overflow occurs

[MEMO]

APPENDIX C INDEX

[0]

1-bit shift multiply add 32
16-bit shift multiply add 34
One's complement 62
Two's complement 63

[A]

Absolute value 61
Accumulate add 69
Accumulate subtract 70
Add 38
AND 49
Arithmetic right shift 42

[C]

Clear 58
Clip 64
Condition 119

[D]

Decrement 60
Direct addressing load/store 85
Divide 72

[E]

Exclusive OR 53
Exponent 66

[F]

Forget interrupt 118

[H]

Halt 116

[I]

Immediate add 39
Immediate AND 50
Immediate arithmetic right shift 44
Immediate exclusive OR 54
Immediate logical left shift 48
Immediate logical right shift 46
Immediate OR 52
Immediate sub 41
Immediate value index load/store 89
Immediate value set 97
Increment 59
Inter-register transfer 94
Interrupt return 106

[J]

Jump 100

[L]

Less than 55
Logical left shift 47
Logical right shift 45
Loop pop 112
Loop 110

[M]

Multiply 37
Multiply add 24
Multiply sub 26

[N]

No operation 115

[O]

OR 51

[P]

Parallel load/store 75
Put 68

[R]

Register indirect jump 101
Register indirect subroutine call 104

Repeat 108
Return 105
Round 65

[S]

Section load/store 79
Sign unsign multiply add 28
Stop 117
Sub 40
Subroutine call 102

[U]

Unsign unsign multiply add 30

[MEMO]

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288
1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6465-6829

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: 02-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: 044-548-7900

I would like to report the following error/make the following suggestion:

Document title: _____

Document number: _____ Page number: _____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❑	❑	❑	❑
Technical Accuracy	❑	❑	❑	❑
Organization	❑	❑	❑	❑