

# HD 61810

High Performance Signal Processor (HSP)





HITACHI DIGITAL  
SIGNAL PROCESSOR(HSP) HD61810  
USER'S MANUAL



AD-E0091

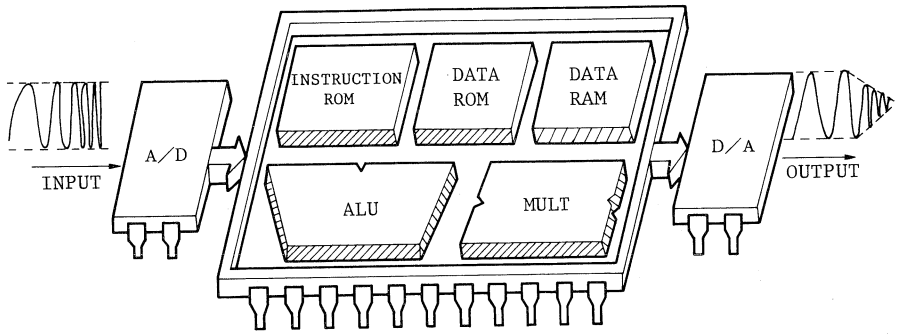
JAN. 1986 AD-ED091

When using this manual, the reader should keep the following in mind;

1. This manual may, wholly or partially, be subject to change without notice.
2. All rights reserved: No one is permitted to reproduce or duplicate, in any form, the whole or part of this manual without Hitachi's permission.
3. Hitachi will not be responsible for any damage to the user that may result from accidents or any other reasons during operation of his unit according to this manual.
4. This manual neither ensures the enforcement of any industrial properties or other rights, nor sanctions the enforcement right thereof.

Copyright © Hitachi, Ltd. 1985. All rights reserved.







## CONTENTS

1. GENERAL DESCRIPTIONS .....	1
1.1 HSP KEY FEATURES .....	3
1.2 APPLICATIONS .....	6
2. ARCHITECTURE .....	7
2.1 PIN FUNCTIONS .....	9
2.2 INTERNAL RESOURCES .....	14
2.3 MEMORY .....	17
2.3.1 Configuration .....	17
2.3.2 Instruction ROM .....	18
2.3.3 Data RAM .....	18
2.3.4 Data ROM .....	21
2.3.5 General Register (GR) .....	22
2.3.6 Memory Addressing Mode .....	23
2.3.7 Memory Data Format .....	24
2.3.8 Precaution in Using ROM/RAM Memory .....	25
2.4 REGISTERS .....	26
2.4.1 Program Counter (PC) .....	26
2.4.2 Stack Registers (STACK0 and STACK1) .....	26
2.4.3 Accumulators (ACCA and ACCB) .....	27
2.4.4 Condition Code Register (CCR) .....	29
2.4.5 Control Register (CTR) .....	30
2.4.6 Status Register (STR) .....	31
2.4.7 Repeat Counter (RC) .....	34
2.4.8 Address Pointers (RAM Pointer A/B, ROM Pointer) .....	35
2.4.9 Delay Register (DREG) .....	36
3. I/O INTERFACE .....	39
3.1 I/O INTERFACE .....	41
3.1.1 Parallel I/O Functions .....	41
3.1.2 Serial I/O Functions .....	44
3.2 INTERRUPT .....	47
3.3 DMA (DIRECT MEMORY ACCESS) .....	53
3.4 BIT I/O, TxRQ .....	60

4.	ARITHMETIC OPERATION .....	61
4.1	DATA FORMAT .....	63
4.2	FIXED POINT ARITHMETIC .....	65
4.2.1	Fixed Point ALU .....	65
4.2.2	Fixed Point Multiplication .....	67
4.3	FLOATING POINT ARITHMETIC .....	71
4.3.1	Floating Point ALU (FALU) .....	71
4.3.2	Floating Point Multiplier .....	77
4.4	DATA TRANSFORMATION .....	79
5.	INSTRUCTION .....	85
5.1	GENERAL DESCRIPTION .....	87
5.2	INSTRUCTION SET .....	88
5.2.1	Description of Operands .....	234
5.2.2	HSP Internal Data Flow .....	238
5.3	PIPELINE CONTROL .....	243
6.	PROGRAMMING TECHNIQUE .....	247
6.1	BIQUAD FILTER .....	249
6.2	TRANSVERSAL FILTER .....	251
7.	HSP APPLICATION .....	253
8.	ELECTRICAL CHARACTERISTICS AND PACKAGE OUTLINE .....	265
8.1	ABSOLUTE MAXIMUM RATING .....	267
8.2	ELECTRICAL CHARACTERISTICS .....	267
8.3	PACKAGE OUTLINE .....	273
	APPENDIX .....	275
1.	HSP REGISTER MODEL .....	277
2.	INSTRUCTION CODE .....	279
3.	INSTRUCTION SET .....	282
4.	ASSEMBLER SYNTAX .....	284
5.	HSP INSTRUCTION SUMMARY .....	287
6.	MEMORY MAP .....	288
7.	HSP ORDERING SPECIFICATION .....	289
8.	TEST PROGRAM .....	291

**SECTION 1**  
**GENERAL DESCRIPTIONS**



## 1. GENERAL DESCRIPTIONS

### 1.1 HSP KEY FEATURES

A High Performance Signal Processor (HSP) is a single chip processor with a stored program designed for a high speed digital signal processing. The HSP contains a high speed floating point arithmetic unit and performs an operation (addition/subtraction/multiplication) with an instruction in a single cycle of only 250 ns. Moreover, the HSP employs a CMOS process to realize a low power consumption.

- 3  $\mu$ m CMOS technology
- Arithmetic
  - Floating point arithmetic
  - Pipeline control
  - Horizontal microinstructions
- Large capacity memories
  - 200  $\times$  16-bit data RAM (2-port accessible)
  - 128  $\times$  16-bit data ROM
  - 512  $\times$  22-bit instruction ROM
- System bus compatible with 8/16-bit microcomputer
- DMA operation between the HSP and external memory
- Serial I/O interface for up to 16 bits
- Operation speed
  - Input clock ; 16 MHz
  - Internal clock ; 4 MHz
  - Instruction cycle; 250 ns
  - MULT, ALU ; 250 ns

(throughput with pipeline control)
- A single power supply of +5V
- Low power dissipation of 250 mW typ.
- Two levels of subroutine and interrupt
- Interrupt by three factors (end of three kinds of data I/O transfer)
- Package; DIC-40/DIP-40/PLCC-52

● What Is The Floating Point Arithmetic ?

The floating point arithmetic expresses a number with a mantissa and an exponent as follows;

$$n = a \cdot 2^b \text{ (a; mantissa, b; exponent)}$$

The floating point arithmetic allows a wide range of numbers to be expressed with less bits, and realizes an easy programming without digit adjustment.

The HSP provides 16 bits for mantissa and 4 bits for exponent and can always work in the maximum precision (16 bits).

In the ALU and an accumulator, the mantissa is 16 bits and the exponent is 4 bits, while in multiplier and memory, the mantissa is 12 bits and the exponent is 4 bits. These data formats allow the HSP to have the precision given by the hatches in Fig. 1.1.1 in the floating point operation. As the exponent is 4 bits, it is fixed to -8 when the data amplitude is low. Therefore, the effective bit length varies in proportion to the data amplitude. When the data exceeds  $2^{-8}$ , the mantissa is normalized and the exponent varies; the effective bit length is fixed to 16 bits maximum.

As described above, the HSP realizes a 32-bit dynamic range with 16 bits of the mantissa and 4 bits of the exponent.



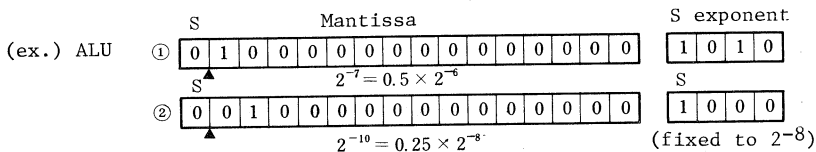
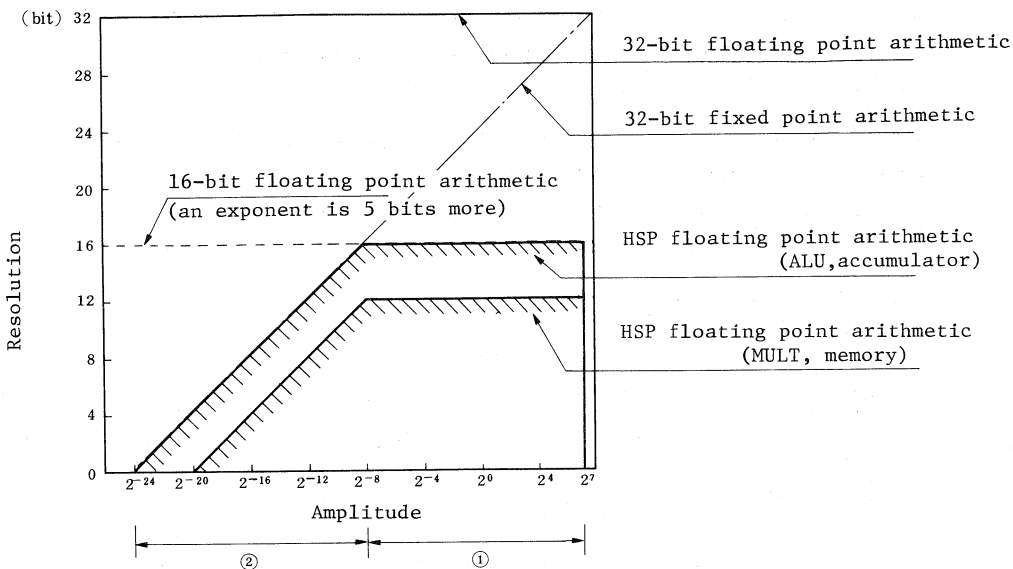


Fig. 1.1.1 HSP FLOATING POINT ARITHMETIC

● What Is The Pipeline Control ?

The HSP utilizes a pipeline control to realize a high-speed operation. This operation permits a full overlap of the instruction prefetch and execution as shown in Fig. 1.1.2. The results are saved for one instruction cycle and can be used in the next instruction cycle.

In the HSP, one instruction execution permits data read, arithmetic operation and data write, etc.

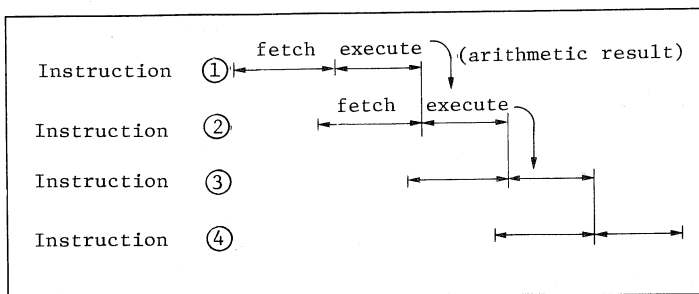


Fig. 1.1.2 PIPELINE CONTROL

The HSP realizes a high throughput with a highly pipelined internal operation and a horizontal microprogram. Fig. 1.1.3 shows how the instructions are executed.

Though the result of multiplication is used in the next instruction cycle, the program with pipeline control permits the simultaneous operations of the ALU and MULT in appearance, thereby the high-speed operation is possible.

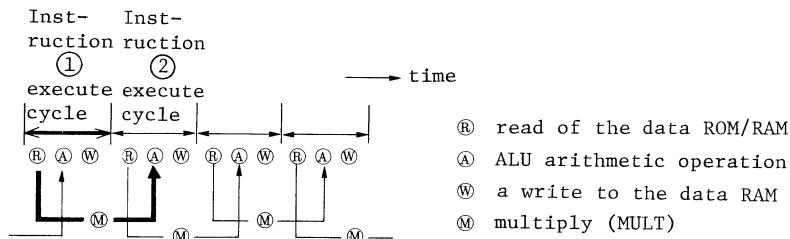


Fig. 1.1.3 ONE INSTRUCTION CYCLE

## 1.2 APPLICATIONS

The HSP (HD61810) is typically used for the following signal processing.

- Digital filter
- FFT
- Digital PLL
- Windowing
- Modulation/demodulation
- Waveform generation
- Correlation

Its applications are described as below.

1. TELECOMMUNICATIONS	<ul style="list-style-type: none"> <li>• High-speed modems</li> <li>• Data transfer (PB,MF,etc.)</li> <li>• Echo canceller</li> <li>• Modulation/demodulation</li> <li>• Adaptive equalizers</li> <li>• High-efficiency CODEC etc.</li> </ul>
2. SPEECH PROCESSING	<ul style="list-style-type: none"> <li>• Speech synthesis</li> <li>• Speech analysis</li> <li>• Speech recognition</li> <li>• Voice mail</li> <li>• Filter bank etc.</li> </ul>
3. IMAGE PROCESSING	<ul style="list-style-type: none"> <li>• Pattern recognition etc.</li> </ul>
4. HIGH-SPEED CONTROL	<ul style="list-style-type: none"> <li>• Servo links etc.</li> </ul>

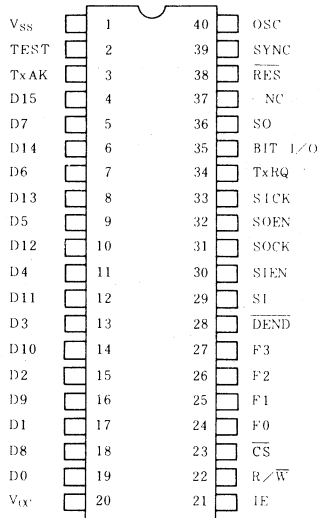
**SECTION 2**  
**ARCHITECTURE**



## 2. ARCHITECTURE

### 2.1 PIN FUNCTIONS

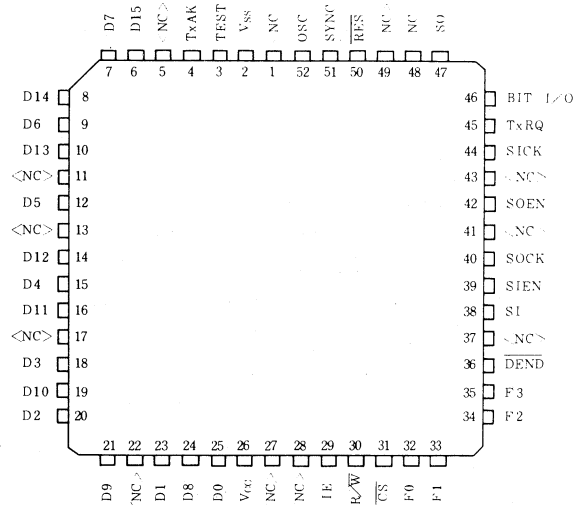
Fig. 2.1.1 and Fig. 2.1.2 show the HSP pin assignments.



(top view)

(NOTE) Any line should not be connected to <NC> pins.

Fig. 2.1.1 DIC/DIP PACKAGE PIN ASSIGNMENT



(top view)

(NOTE) Any line should not be connected to <NC> pins.

Fig. 2.1.2 PLCC PACKAGE PIN ASSIGNMENT





- The Handling of HSP Unused Pins

When the HSP operates in the stand-alone mode, pins for control and parallel I/O are not used. Moreover, there are some cases when the serial I/O functions and DMA function are not used.

Each unused pin should be handled as follows.

Table 2.1.1 THE HANDLING OF HSP UNUSED PINS

Functions	Pins	Pin No.	I/O	Handling of Pins
Serial Input	SI	29	I	Tie to Vcc/GND.
	SIEN	30	I	Tie to GND.
	SICK	33	I	Tie to Vcc.
Serial Output	SO	36	O	Pull up/down by a resistor.
	SOEN	32	I	Tie to GND.
	SOCK	31	I	Tie to Vcc.
DMA Operation	TxRQ*	34	O	Pull down by a resistor.
	TxAk	3	I	Tie to GND.
	DEND	28	I	Tie to Vcc.
Bus I/O	$\overline{CS}$	23	I	Tie to Vcc.
	$\overline{IE}$	21	I	Tie to GND.
	$\overline{R/\overline{W}}$	22	I	Tie to Vcc.
	$F_0 \sim F_3$	24-27	I	Tie to GND.
	$D_0 \sim D_7$		I/O	Pull up/down by a resistor.
	$D_8 \sim D_{15}$		I/O	Pull up/down by a resistor.  (Note) Even in the byte transfer mode, $D_8 \sim D_{15}$ should be pulled up/down by a resistor.
Others	BIT I/O*	35	I/O	Pull down by a resistor.
	TEST	2	I	Tie to GND. (Not for user)
	SYNC	39	O	Open

(Note 1) \* means open drain output. Connect a pull-up resistor to output data.

(Note 2) The pins tied to Vcc/GND can be connected to resistor.



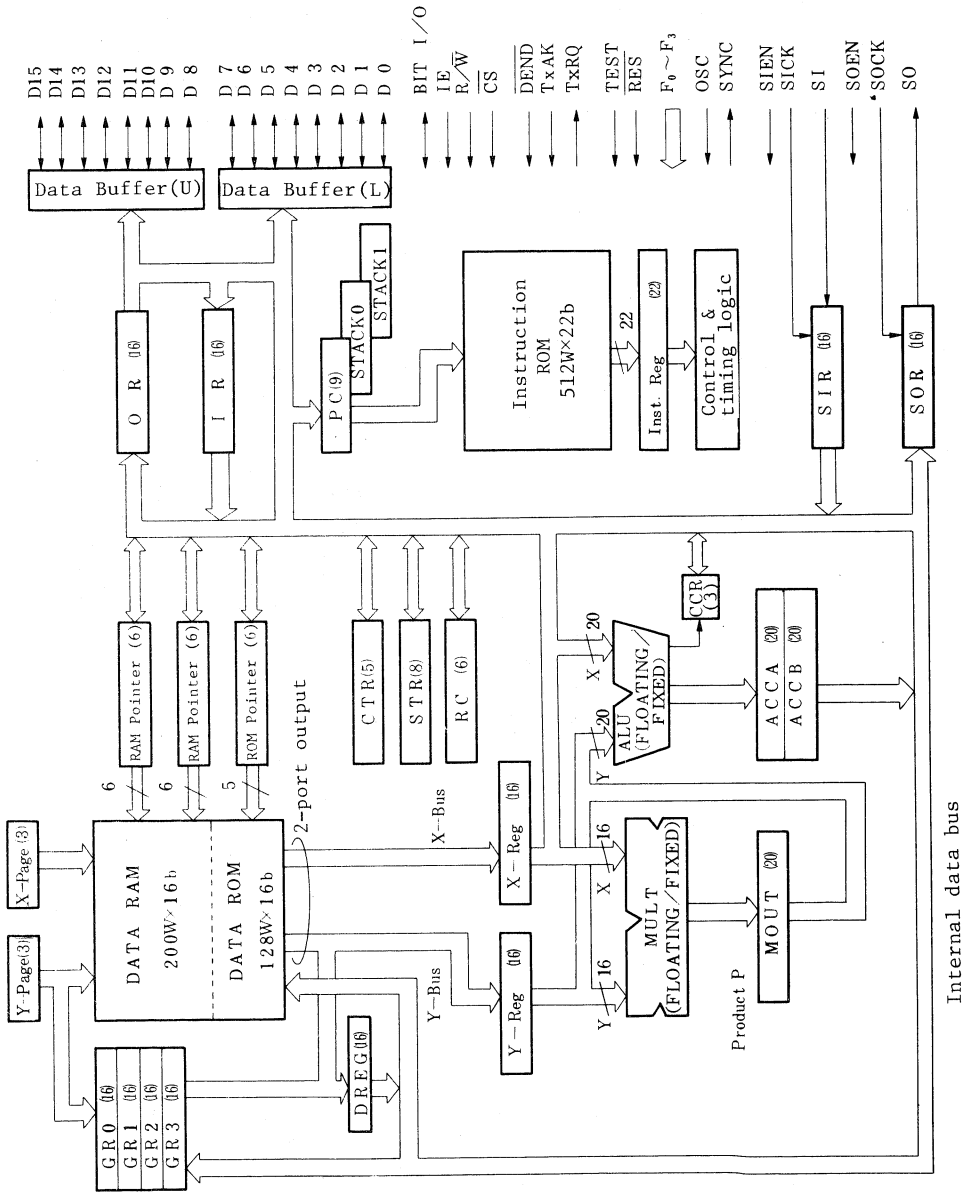


Fig. 2.2.1 HD61810 BLOCK DIAGRAM

## 2.2 INTERNAL RESOURCES

The HSP contains large capacity memories; a data RAM of 200 word  $\times$  16 bits, a data ROM of 128 words  $\times$  16 bits and an instruction ROM of 512 words  $\times$  22 bits, and contains dedicated multiplier and ALU which permit high-speed, high-accuracy floating point operation. The instruction ROM stores a comprehensive instruction set, which allows a wide range of applications.

The block diagram of the HSP is shown in Fig. 2.2.1. Each of the blocks is described in the following table.

Input/Output Registers		
Input Register	IR	16-bit register. Data is input to this register through the external data bus (D0-D15).
Output Register	OR	16-bit register. Data is output from this register to the external data bus (D0-D15).
Serial Input Register	SIR	16-bit shift register for serial data input. After a serial data is transferred from the SIR to an accumulator (ACCA or ACCB), the SIR is cleared.
Serial Output Register	SOR	16-bit shift register for serial data output. If data has been transferred to the SOR through the internal data bus, the data is output to peripherals on a bit basis.
Control Registers		
Condition Code Register	CCR	The CCR contains three flag bits; Carry (C), Negative (N), and Zero (Z). They indicate the results of ALU operation.
Status Register	STR	The STR flags are individually set or cleared depending on the status of the HSP. The contents of the STR can be transferred to an accumulator.
Control Register	CTR	The CTR is used to select the desired operating modes for the HSP. The CTR contents are determined by either the HSP instructions or the MPU.

Instruction Controllers		
Instruction ROM		A 512 word by 22 bit ROM. The instruction ROM stores instructions for the HSP. 22-bit instruction is transferred to the instruction register in parallel in a single instruction cycle.
Program Counter	PC	The PC is a 9-bit address counter that is used to address the Instruction ROM. The PC generates the instruction ROM addresses 0 through 511.
Stack Registers	STACK0 STACK1	The stack registers are 9-bit registers that are used to save the PC contents. The contents of the PC is pushed onto these registers when a subroutine jump or an interrupt occurs.
Repeat Counter	RC	6-bit down counter. The RC is used for repeated execution of an instruction and for the control of loops.
Instruction Register	Inst. Reg.	22-bit buffer register. This register temporarily stores the instruction transferred from the instruction ROM.
Internal Memory Controllers		
Data RAM		A 200 word by 16 bit RAM.
Data ROM		A 128 word by 16 bit ROM.
RAM Pointer A RAM Pointer B	RA RB	These are 6-bit address pointers which are used to generate the data RAM address, combining with the contents of page address.
ROM Pointer	RO	This is a 6-bit address pointer which is used to generate the data ROM address, combining with the contents of page address.
X and Y page Address Pointers	X/Y- Page	These are 3-bit buffer registers for a page address. The effective address for the data RAM or the data ROM consists of this address and the contents of the RAM/ROM pointer.
General Registers	GR0-3	16-bit general purpose registers. The GRs can be used as working registers. Data is transferred to or from the GRs through the Y-Bus.

Internal Memory Controllers		
Delay Register	DREG	16-bit register. The DREG holds the data to be transferred through the Y-bus for a single instruction cycle period.
Arithmetic Elements		
Accumulator A Accumulator B	ACCA ACCB	20-bit accumulators. The accumulators store the output from the ALU. Either the ACCA or ACCB is selected in response to the instructions.
Arithmetic Logic Unit	ALU	The ALU performs arithmetic and logical operations. Either the fixed point operation or the floating point operation is selected depending on the instructions.
Multiplier	MULT	The MULT performs a multiply operation. Either the fixed point operation or the floating point operation is selected depending on the instructions.
Multiplier Input X-Register	X-Reg.	16-bit register. The X-Reg. holds the data transferred from the X-bus or the internal data bus during a multiply operation.
Multiplier Input Y-Register	Y-Reg.	16-bit register. The Y-Reg. holds the data transferred from the Y-bus or the internal data bus during a multiply operation.
Multiplier Output Register	MOUT	This is a 20-bit buffer register which holds the output from the MULT for a single instruction cycle period. This register consists of a mantissa (16 bits) and an exponent (4 bits).

## 2.3 MEMORY

### 2.3.1 Configuration

The HSP has the following three data memories as shown in Fig. 2.3.1.

- o Data RAM : 200 W × 16b
- o Data ROM (for coefficient) : 128 W × 16b
- o General Registers : 4 W × 16b

The words data can be transferred from these memories to the multiplier (FMULT) and ALU (FALU) in parallel through two buses; X- and Y-Bus.

A data bus is also provided and the data is written into the data RAM or GR through this bus.

Program memory is a 512 W × 22b on-chip ROM and stores instructions.

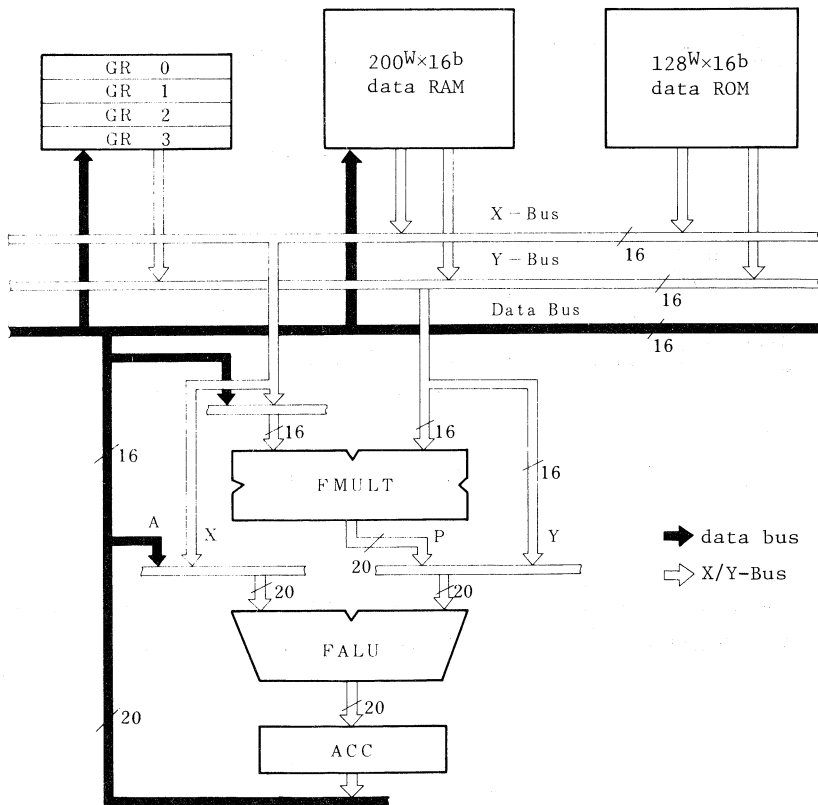


Fig. 2.3.1 DATA MEMORY CONFIGURATION

### 2.3.2 Instruction ROM

The instruction ROM consists of up to 512 words of 22-bit width. A 22-bit instruction is read out to the instruction register during each instruction cycle. The program starts from location \$0, but the execution of the program starts from location \$1. When jumping to location \$0 during instruction execution, the program starts from location \$0 and the instruction of location \$0 is also executed.

Location \$1FF (the last address) is a vector address for interrupt. Thus, store jump instruction here for jump to an interrupt service routine. Locations \$1E7 through \$1FE are reserved for LSI testing purposes, so they should not be used for the user's program.

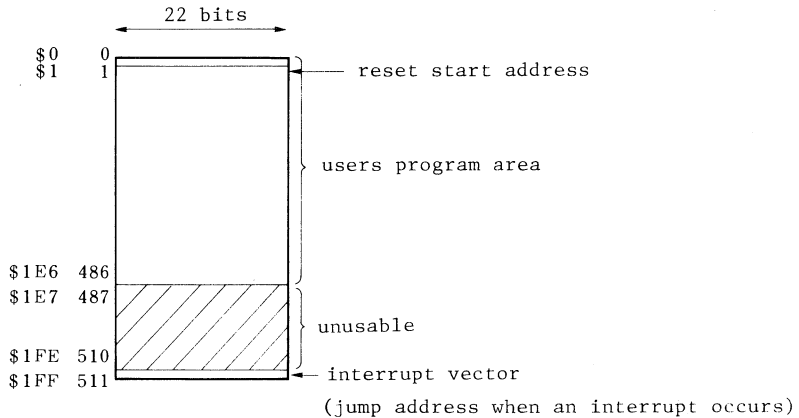


Fig. 2.3.2 INSTRUCTION ROM MEMORY MAP

### 2.3.3 Data RAM

Data RAM consists of up to 200 words of 16-bit width and is divided into four pages (page address; 0 through 3). Each page consists of 50 words.

The data in the data RAM can be accessed through two ports. Two data of the different pages (X-page and Y-page) can be read in parallel if the pointer addressing mode is employed.

In the pointer addressing mode, 6 bits of the RAM pointer and 3-bit page address (X-/Y-page address) in the instruction code can be used as the data memory address. When accessing two data, the pointer addresses should be identical. (RAM pointer A and RAM pointer B cannot be used at the same time.) Fig. 2.3.4 illustrates data RAM memory access in the pointer addressing mode.

The page address specified in the X-page part of an instruction code is the X-Bus and that specified in the Y-page part to the Y-Bus a write operation is performed through the data bus to the data RAM address consisted of RAM pointer address and Y-page address. The contents of an accumulator or the DREG are written to the data RAM.

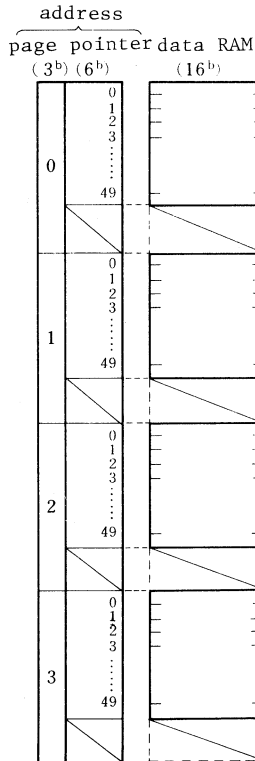


Fig. 2.3.3 DATA RAM MEMORY MAP

In the direct addressing mode, both the pointer address and the page address are contained in the instruction code. In this case, the data is output to Y-Bus.

See Section 2.4.8 for details of address pointers.

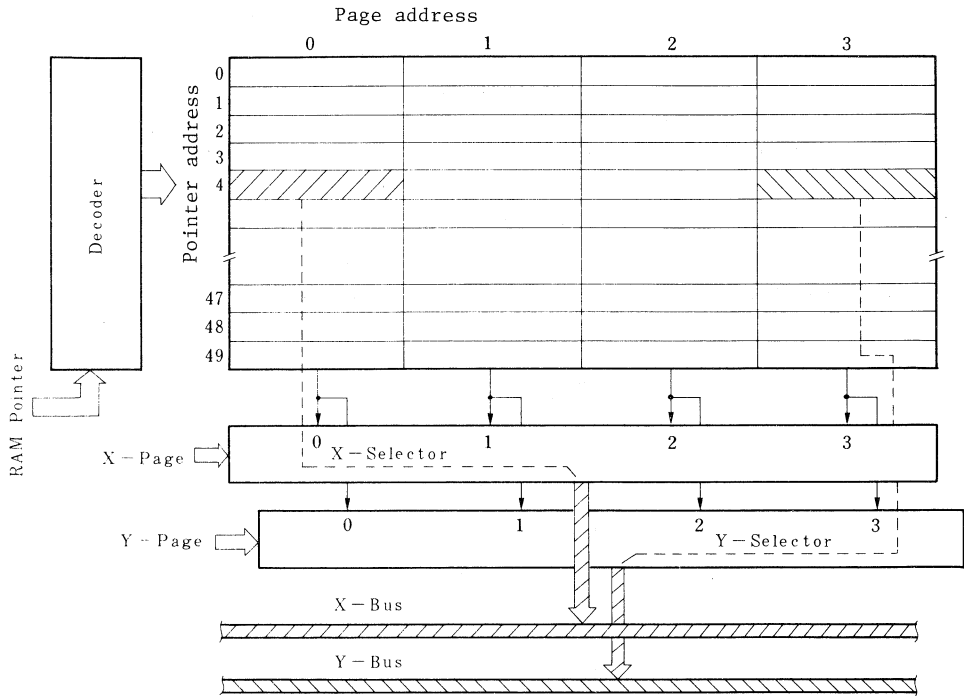


Fig. 2.3.4 MEMORY ACCESS OF DATA RAM (POINTER ADDRESSING MODE)



### 2.3.4 Data ROM

Data ROM consists of up to 128 words of 16-bit width and is divided into four pages (page address; 4 through 7). Each page consists of 32 words. If the pointer addressing mode is employed, 6 bits of the ROM pointer and 3-bit page address (X-/Y-page address) in the instruction code are used to make an effective address.

The data RAM has two ports for output, while the data ROM has only one port. However, as shown in Fig. 2.3.6, one data can be output to both X-Bus and Y-Bus. Thus, two-data output from the data ROM does not allow the difference between the X- and Y-page addresses.

In the direct addressing mode, both the pointer address and the page address are contained in the instruction code. In this case, the data is output to Y-Bus.

When reading the data ROM and the data RAM at the same time, two data can be read from the different pointer addresses, because each data memory has its own pointer address.

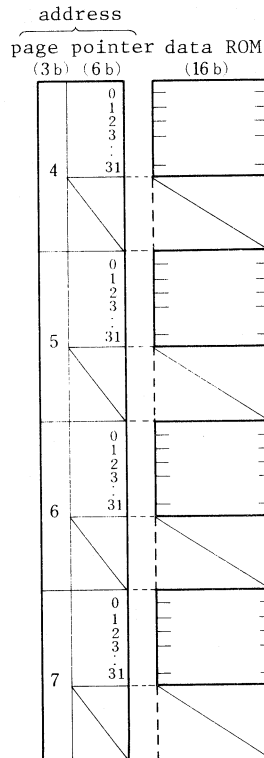


Fig. 2.3.5 DATA ROM MEMORY MAP

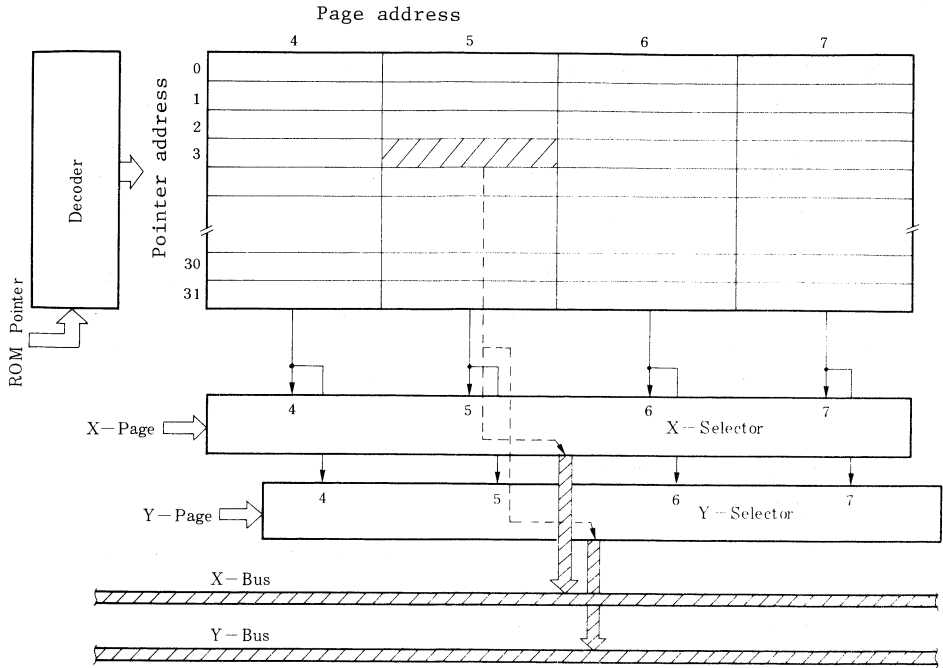


Fig. 2.3.6 MEMORY ACCESS OF DATA ROM (POINTER ADDRESSING MODE)

### 2.3.5 General Register (GR)

The HSP has four 16-bit general registers (GR0 to 3) which are used as working registers.

If the pointer addressing mode is employed, the GR and data ROM/RAM can be accessed at the same time.

The address of the GR is specified in the Y-page address part in the instruction code. The data in the GR is input/output through the Y-Bus.

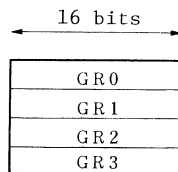


Fig. 2.3.7 GENERAL REGISTERS

### 2.3.6 Memory Addressing Mode

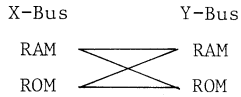
There are two modes of data memory addressing; pointer addressing and direct addressing.

#### (1) Pointer addressing mode

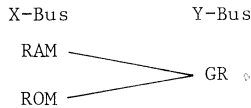
Pointer addressing uses the page address (0 through 7 page) and the pointer address as the data memory address. Page address is contained in the instruction code, and the pointer address is determined by each address pointer (RA/RB/RO).

In this mode, the X-Bus output and Y-Bus output are as follows;

- o The data RAM and data ROM are accessed.



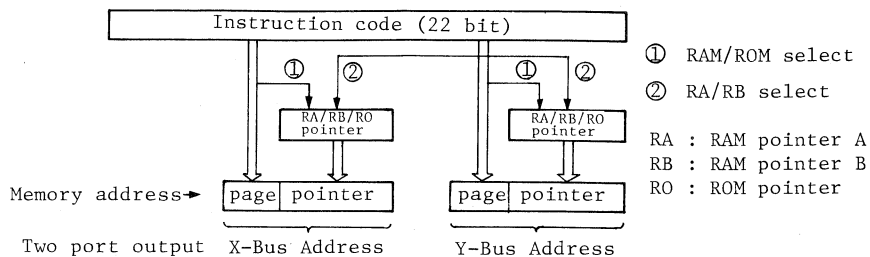
- o The data RAM/ROM and GR are accessed.



The pointer addressing mode is available for accessing two-word data during the product sum operation, reading data from sequential addresses, etc.

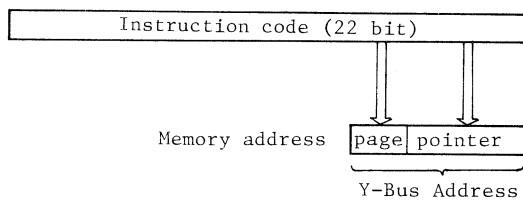
#### (2) Direct addressing mode

In direct addressing, nine bits of the instruction code specify data RAM/ROM address (page address and pointer address). The direct addressing mode is available for multiplying one-word data in the data ROM/RAM by data in an accumulator, or reading data from discrete addresses.



Pointer part is specified by ROM/RAM pointer indirectly.

(a) Pointer addressing mode



Memory address is specified directly.

(b) Direct addressing mode

Fig. 2.3.8 MEMORY ADDRESSING MODE

### 2.3.7 Memory Data Format

Fig. 2.3.9 details data formats in the data ROM/RAM and GR.

Note that in floating point data form the twelve leftmost bits of the 16-bit mantissa are transferred from an accumulator to the data memory.

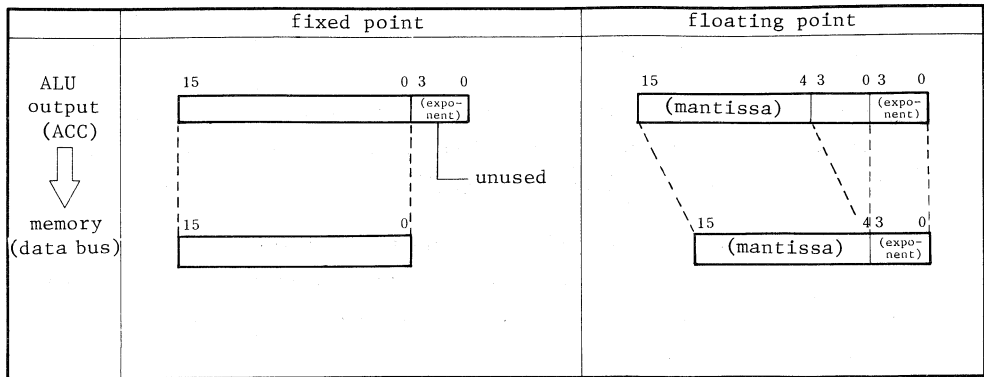


Fig. 2.3.9 DATA FORMAT

### 2.3.8 Precaution in Using the Data ROM/RAM

- (1) The data RAM (and registers, if necessary) should be cleared by software after power on to prevent the oscillation of the digital filter, etc. The data RAM cannot be cleared at reset.
- (2) Location 31 (in page 7) of the data ROM is reserved by the HSP emulator system for program development. Special attention is needed not to use this location when using the emulator system.

## 2.4 REGISTERS

### 2.4.1 Program Counter (PC)

The PC is a 9-bit register which contains the address of the next instruction to be executed. The PC can generate the instruction ROM addresses in the range from 0 through 511.

The MPU can set the instruction ROM addresses 0 through 255 in the PC through the external data bus. In this case, however, the instruction execution starts from the next address to the specified one.

### 2.4.2 Stack Registers (STACK0 and STACK1)

The STACK0 and the STACK1 are 9-bit registers. The contents of the PC are pushed onto the stack registers when a subroutine jump or an interrupt occurs.

The HSP provides two stack registers, allowing two nesting levels.

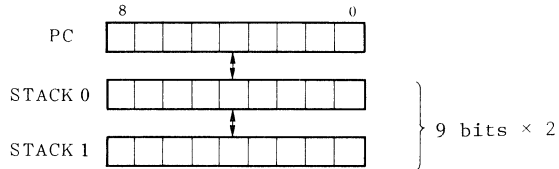


Fig. 2.4.1 ARRANGEMENT OF THE STACK REGISTERS

### 2.4.3 Accumulators (ACCA and ACCB)

The HSP has two 20-bit accumulators (ACCA and ACCB). Either of the accumulators is selected by the accumulator select bit in the instruction code.

The accumulators store the output from the ALU. Fig. 2.4.2 and Fig. 2.4.3 show the input/output of the accumulators for the fixed point operation and the floating point operation respectively. When a floating point data is transferred from an accumulator to the data RAM or the GRs, the lower four bits of the mantissa is truncated.

Since a floating point data consists of 20 bits, it occupies two words of the data RAM or two GRs. A transfer of a floating point data between an accumulator and the data RAM (or the GRs) is performed in the following procedures.

- o To save the contents of an accumulator in the data RAM or the GRs,
  - (1) transfer a mantissa (16 bits) of the floating point data from the accumulator to a data RAM (or GR) address ① in the fixed point representation.
  - (2) then transfer the same data into the next data RAM (or the next GR) address ② in the floating point representation.
- o To transfer the contents of the data RAM or the GRs to an accumulator,
  - (1) Transfer the contents of the data RAM (or GR) address ① to the accumulator.
  - (2) Converts the contents of the accumulator to a floating point data using the contents of the data RAM (or GR) address ② as a scaling constant.

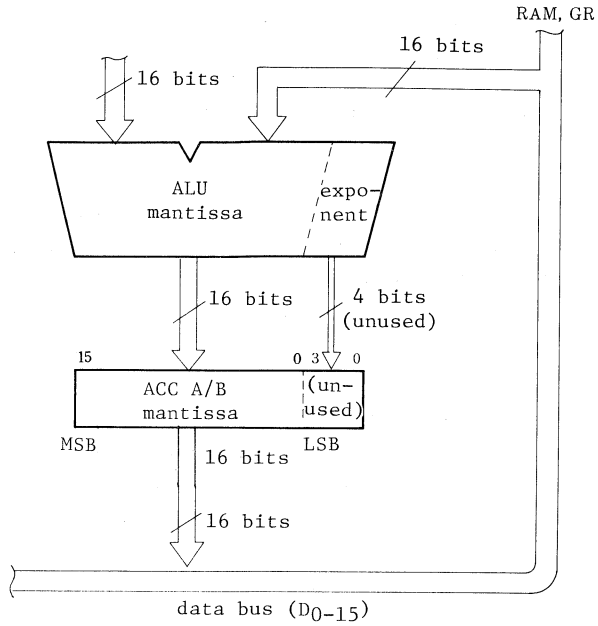


Fig. 2.4.2 INPUT/OUTPUT OF THE ACCUMULATORS FOR THE FIXED POINT ARITHMETIC OPERATION

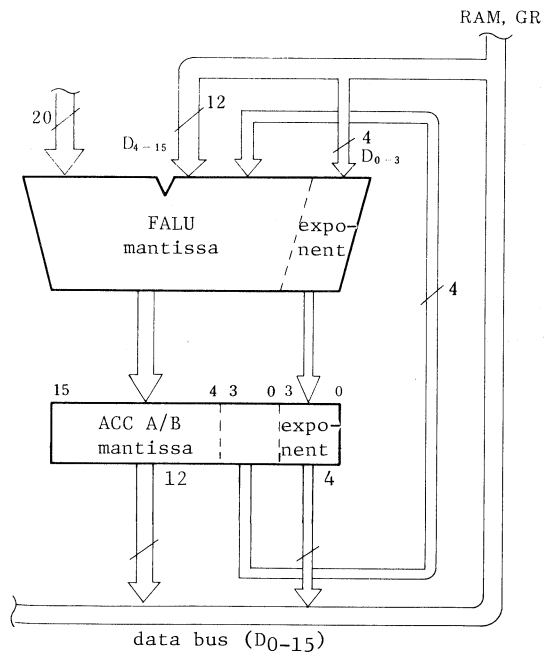


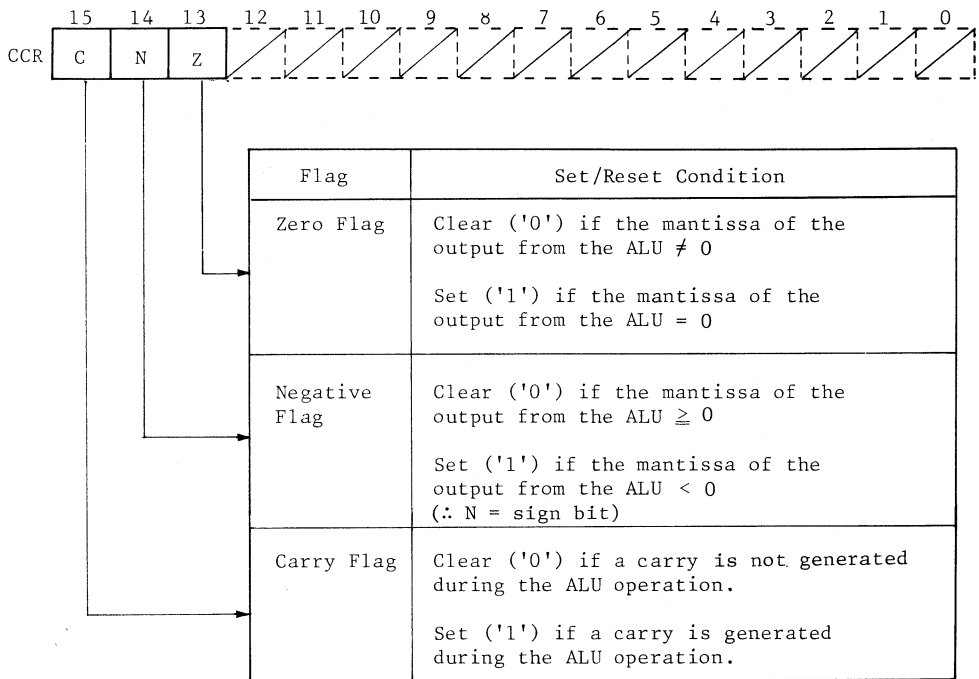
Fig. 2.4.3 INPUT/OUTPUT OF THE ACCUMULATORS FOR THE FLOATING POINT ARITHMETIC OPERATION



#### 2.4.4 Condition Code Register (CCR)

The CCR contains three flag bits; Carry (C), Negative (N) and Zero (Z). They indicate results of the ALU operation.

The CCR is connected to the D13 to D15 bits of the internal data bus, which allows the transfer of data between the CCR and the accumulators in response to the instructions.



- o All of the CCR bits are affected by the fixed point operation.
  - o Z and N are affected by the floating point operation.
- For details of the C flag bit, see 5.2 'INSTRUCTION SET'.

Fig. 2.4.4 CCR SET/RESET CONDITION

### 2.4.5 Control Register (CTR)

The CTR can be used to determine the desired operating mode for the HSP. Dedicated instructions transfer data from the accumulator to the CTR. The MPU can also input data to the CTR through parallel I/O pins (D0 - D15). During reset, the DMA bit and the TxRQ bit are cleared but the others are undefined.

The CTR is connected to the D0 to D7 of the internal data bus.

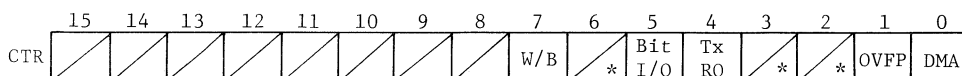


Table 2.4.1 FUNCTIONS OF THE CTR

Bit	Name	Function	Descriptions
0	DMA	Parallel I/O Data Transfer Mode Select bit	<ul style="list-style-type: none"> <li>Selection of the mode for data transfer through the parallel I/O ports.                             <ul style="list-style-type: none"> <li>1: DMA operation mode</li> <li>0: Non-DMA operation mode</li> </ul> </li> </ul> <p>This bit must be cleared during a usual data transfer. Even in the DMA operation mode, the transfer of data through the parallel I/O ports must be accomplished by the HSP instructions.</p>
1	OVFP	Overflow Protection bit	<ul style="list-style-type: none"> <li>Protection against overflows.</li> </ul> <p>If an overflow occurs during an ALU operation, the result is fixed to the maximum value.</p> <ul style="list-style-type: none"> <li>1: Performing a protection against overflow.</li> <li>0: Not performing a protection against overflow.</li> </ul>
4	TxRQ	DMA Operation Request bit	<ul style="list-style-type: none"> <li>Request for DMA operation.</li> </ul> <p>This bit is set to request a DMA transfer in the DMA operation mode. The TxAK input clears this bit and then it is set automatically. However, after the <u>DEND</u> input, this bit remains clear. In the non-DMA operation mode, the TxRQ can be used as a programmable output.</p>
5	BIT I/O	Bit I/O bit	<ul style="list-style-type: none"> <li>Input and output of the BIT I/O pin.</li> </ul> <p>This bit controls the input/output of the BIT I/O pin. When using the BIT I/O pin as an output, the user must write data directly to this register. When using the bit as an input, he must write a 1 and then data in this register.</p>

Bit	Name	Function	Descriptions
7	W/B	Word/Byte bit	<ul style="list-style-type: none"> <li>o Selection of the size of the data transferred through parallel I/O ports.</li> <li>1: Word (16-bit) data is used.</li> <li>0: Byte (8-bit) data is used.</li> </ul>

\* 0s must be written in the unused bits of the CTR, but 1s are read from these bits.

#### 2.4.6 Status Register (STR)

The STR indicates the current HSP information and controls the HSP operation.

The contents of this register are transferred to the accumulator by the TFR $\Delta$ STR,A and TFR $\Delta$ STR,B). After the transfer of data from the STR to the accumulator, the SOF, SIF and PF bits are cleared.

In addition, the contents of the accumulator can be transferred into the UF, I<sub>SO</sub>, I<sub>SI</sub>, I<sub>P</sub> and I<sub>M</sub> bits by the instructions (TFRAA, STR and TFRAB, STR).

The STR is connected to the low-order byte of the internal data bus (D0-D7). During reset, the I<sub>M</sub> bit is set, the PF, SIF and SOF are cleared and the other bits are undefined.

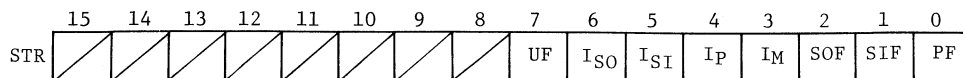


Table 2.4.2 FUNCTIONS OF THE STR

Bit	Name	Function	Descriptions
0	PF	Parallel I/O data transfer end flag	The PF bit is a read-only bit which is set by the $\overline{CS}$ input during data transfer through the D0-D15. If the data transfer is performed on a byte basis, the PF bit is set by the $\overline{CS}$ input during the transfer of the high-order byte. If interrupts are enabled, an interrupt occurs when this bit is set. The PF bit can be set even when interrupts are disabled. The data transfer from the STR to the accumulator clears the PF.
1	SIF	Serial input data transfer end flag	The SIF bit is a read-only bit which is set to indicate that a serial input has been completed. If interrupts are enabled, an interrupt occurs when this bit is set. This bit can be set even when interrupts are disabled. The data transfer from the STR to the accumulator clears the SIF.
2	SOF	Serial output data transfer end flag	The SOF is a read-only bit which is set to indicate that a serial output has been completed. If interrupts are enabled, an interrupt occurs when this bit is set. This bit can be set even when interrupts are disabled. The data transfer from the STR to the accumulator clears the SOF bit.

Table 2.4.2 FUNCTIONS OF THE STR (cont'd)

Bit	Name	Function	Descriptions
3	I <sub>M</sub>	Interrupt mask bit	When an interrupt occurs, the I <sub>M</sub> bit is automatically set to prevent additional interrupts. The I <sub>M</sub> bit is cleared by the RTI instruction. This bit is also set or cleared in response to the data transfer instructions of TFRΔA,STR and TFRΔB,STR.
4	I <sub>P</sub>	Parallel I/O interrupt mask bit	The I <sub>P</sub> bit is set to disable interrupts occurring after the completion of a data transfer through the parallel I/O ports. The I <sub>P</sub> bit is also set or cleared in response to the data transfer instructions of TFRΔA,STR and TFRΔB,STR.
5	I <sub>SI</sub>	Serial input interrupt mask bit	The I <sub>SI</sub> bit is set to disable interrupts occurring after the completion of serial input. This bit is set or cleared in response to the data transfer instructions of TFRΔA,STR and TFRΔB,STR.
6	I <sub>SO</sub>	Serial output interrupt mask bit	The I <sub>SO</sub> bit is set to disable interrupts occurring after the completion of serial output. This bit is set or cleared in response to the data transfer instructions of TFRΔA,STR and TFRΔB,STR.
7	UF	User's bit	The UF bit is an optional bit. This bit is set or cleared in response to the data transfer instructions of TFRΔA,STR and TFRΔB,STR.

### 2.4.7 Repeat Counter (RC)

The RC can be used for the repeated execution of the same program step in response to the repeat instructions or for the execution of loops in response to the jump instructions. This counter reduces the number of the program steps for repeated product/sum operations, which leads data processing time to be reduced.

The RC consists of six bits connected to D10 through D15 of the internal data bus. The user can set a value of 0 to 63 in the RC.

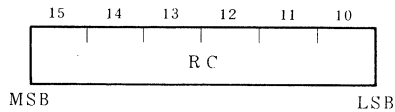


Fig. 2.4.5 REPEAT COUNTER

The following programs contain the repeated operations using the RC.

Example (1) Repeated operation by the repeat instructions  
Step 1 (RC) ← #n (n = the number of repetition)  
Step 2 Repeat instruction (The RC may be decremented)  
Step 3 Arithmetic instruction (RC) ← (RC-1)  
The arithmetic instruction is repeated until the RC content becomes 0. Therefore, the number of the repetition of the arithmetic operation is n+1.  
Repeat instruction ; FRPTA, FRPTB, RPTA, RPTB

Example (2) Repeated operation by the jump instructions  
Step 1 (RC) ← #n  
Step 2 Instruction 1  
Step 3 Instruction 2  
Step 4 If RC ≠ 0, jump to Step 2 decrementing the RC content.  
Step 5 .....  
The execution of instructions 1 and 2 will be repeated until the RC becomes 0, and then the execution of Step 5 will start.

The RC may be autodecremented when any of the RAM/ROM pointers is incremented by the pointer addressing mode instructions. The RC can also be decremented by the decrement instructions.

For details, see the descriptions of the HSP instructions.

#### 2.4.8 Address Pointers (RAM pointer A/B, ROM pointer)

The HSP provides three 6-bit address pointers: RAM pointer A, RAM pointer B and ROM pointer.

The contents of an address pointer (that is, pointer address) is combined with the contents of the page address part in the instruction code (either X page or Y page) to generate an effective address.

Fig. 2.4.6 shows the address pointers. Each of the address pointers is connected to the D10-D15 of the internal data bus.

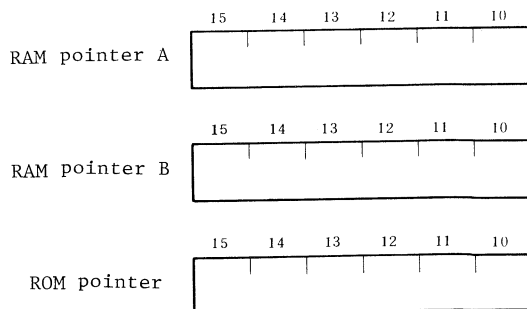
Fig. 2.4.7 shows how to generate an effective address of the data RAM or the data ROM using the address pointers.

By using two page addresses (X page and Y page) and a pointer address, two data located in different pages of the data RAM can be read at the same time. A write operation is performed to the data RAM address which is generated by the contents of a RAM pointer and a Y page address.

Since two different data cannot be read from the data ROM at the same time, the same data are output onto the X-Bus and Y-Bus.

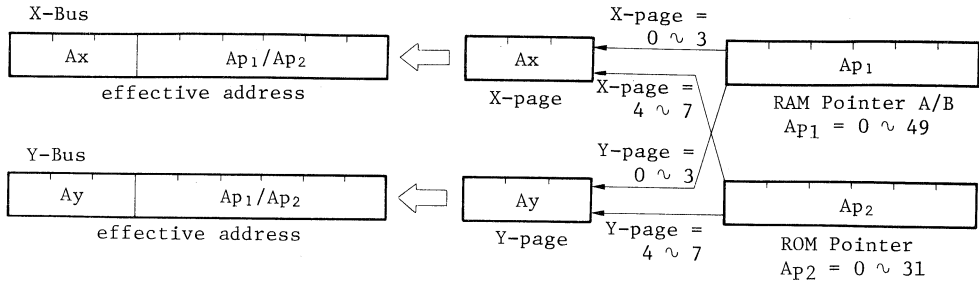
The HSP instructions select either the RAM pointer A or the RAM pointer B. The address pointers may be autodecremented after the instruction execution is completed.

Using two RAM pointers brings the efficient programmed complex number arithmetic such as FFT.



(Note) The highest bit of the ROM pointer must be 0.

Fig. 2.4.6 ADDRESS POINTERS



- o Either the RAM pointer or the ROM pointer is selected by page address.
- o Either the RAM pointer A or the RAM pointer B can be used.
- o If both of the effective addresses are generated with the contents of the ROM pointer, the contents of X-page must be the same as these of Y-page.

Fig. 2.4.7 GENERATION OF EFFECTIVE ADDRESS USING ADDRESS POINTERS

#### 2.4.9 Delay Register (DREG)

The arithmetic operation for signal processing is performed using the delay function, such as the transversal filter shown in Fig. 2.4.9. This function delays a string of input data for a sampling period on every sampling. (The delay is represented by  $Z^{-1}$ ).

The HSP provides the DREG connected to the output of the data RAM as shown in Fig. 2.4.10.

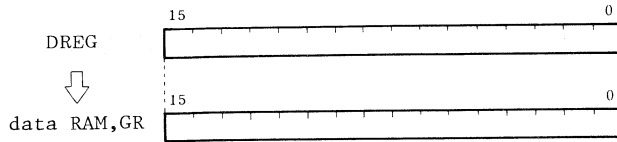
The DREG provides a two-stage latch circuit. This register holds the data read from the data RAM for a single instruction cycle, and then the data is written into the next data RAM address after reading the data of the address.

By repeating this sequence, the DREG shifts data of the data RAM to the next address.



The data transfer from the DREG to the data RAM is controlled by the instruction code. The format of the data transfer is described as below.

• Fixed point data



• Floating point data

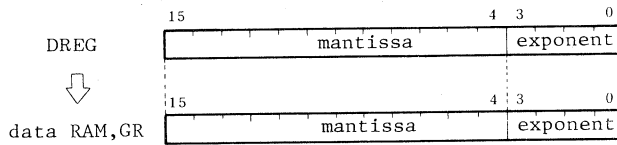


Fig. 2.4.3 THE DATA TRANSFER FROM THE DREG TO THE DATA RAM

The data transfer between the DREG and the GR can also be performed.

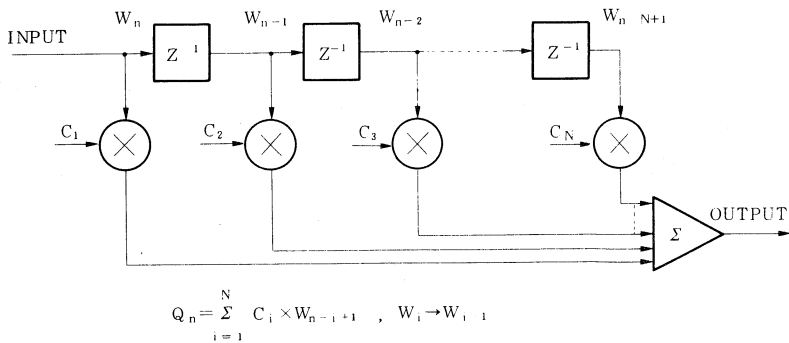


Fig. 2.4.9 TRANSVERSAL FILTER

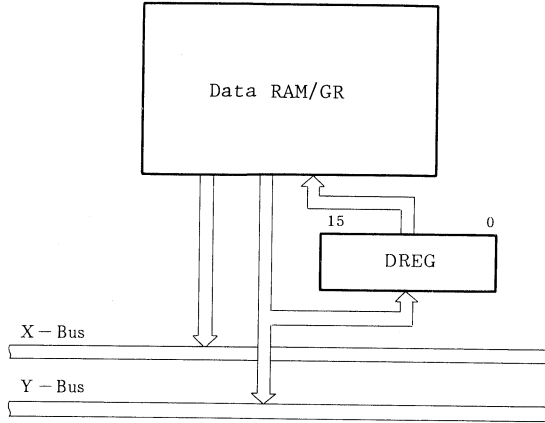


Fig. 2.4.10 THE POSITION OF THE DREG

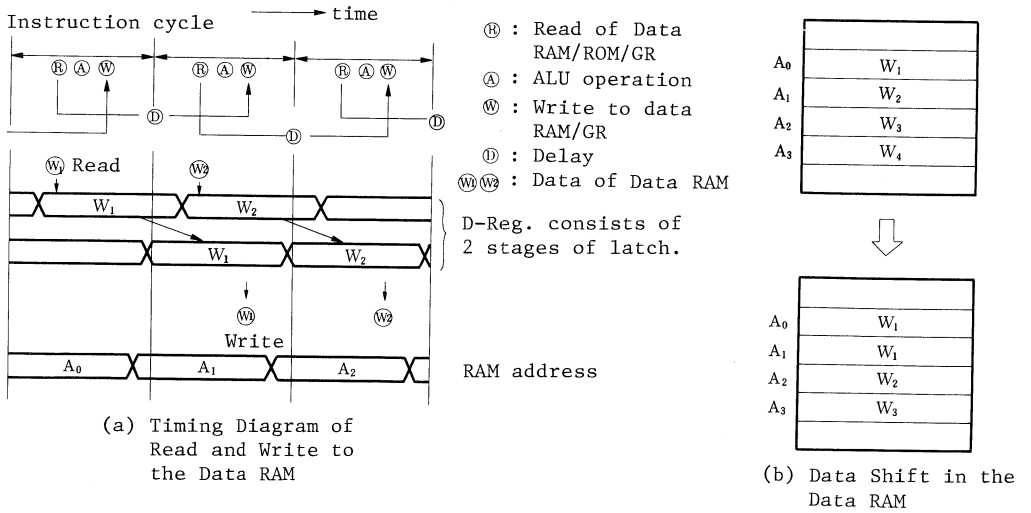


Fig. 2.4.11 THE DATA SHIFT IN THE DATA RAM USING THE DREG

**SECTION 3**  
**I/O INTERFACE**



### 3. I/O INTERFACE

#### 3.1 I/O INTERFACE

The HSP has parallel and serial I/O functions. The parallel I/O pins can be directly connected with the data buses of the 8-bit micro-computer HMCS6800.

##### 3.1.1 Parallel I/O Functions

The HSP provides bidirectional data buses (D0 through D15) for parallel I/O function. The parallel data is transferred between the HSP and the external device through parallel I/O ports (D0 through D15) as follows.

- (1) Word (16-bit) data transfer (D0-D15 ↔ IR, OR)

16-bit data transfer between the parallel I/O lines (D0 through D15) and the 16-bit input register (IR) or the 16-bit output register (OR).

- (2) Byte (8-bit) data transfer (D0-D7 ↔ IR, OR)

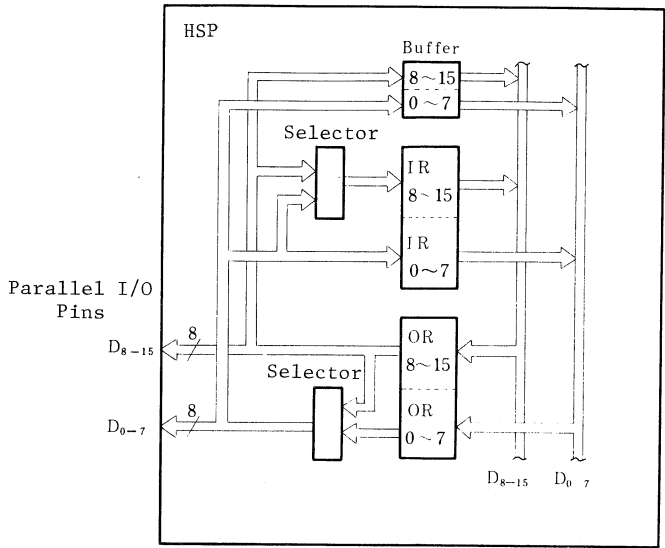
8-bit data transfer between the parallel I/O lines (D0 through D7) and the IR or OR. A word data in the IR/OR is divided into two bytes for transfer; the high-order byte (bit 8 through bit 15) and the low-order byte (bit 0 through bit 7).

First, data in the low-order byte is sent, and then the high-order byte is sent. Note that this is a reverse of the order in the microcomputer.

- (3) CTR/PC data transfer (D0-D15 → CTR, PC)

The contents of the CTR/PC can be changed externally.

In this case, a written data is transferred from the MPU through the parallel I/O pins D0 - D15. This function can be used to restart the HSP program from the optional address (address 1 - 256) during instruction execution. The start address is (PC) + 1. In this case, the I/O ports (D0 - D15) are directly connected to the internal data bus (D0 - D15) without through the IR or OR. During a transfer operation, the HSP is placed in the halt state because the internal data bus is totally dedicated. The data is set in response to the negative edge of IE.



The selector determines the size of transferred data (word/byte)

Fig. 3.1.1 PARALLEL I/O DATA TRANSFER

Parallel I/O data transfer is controlled with the  $\overline{CS}$ , IE,  $R/\overline{W}$ , F0-F3.  
 The function depends on the contents of function control pins F0-F3.  
 The function control pins F0-F3 can be used only if the  $\overline{CS}$  is active.

Table 3.1.1 PARALLEL I/O FUNCTIONS

Control		Operations	HSP Operations	Interrupt
Chip Select	Function Control			
$\overline{CS}$	F3 F2 F1 F0			
1	* * * *	No I/O operation	Active	—
0	0 0 0 0	No I/O operation	Active	—
	0 0 1 0	<u>Data transfer</u> (CTR(W/B)=0) <ul style="list-style-type: none"> <li>● low order byte</li> <li>Read : D0-7 ← OR0-7</li> <li>Write: D0-7 → IR0-7</li> </ul>	Active	No interrupt
	0 0 1 1	<u>Data transfer</u> (1) Byte transfer mode (CTR(W/B)=0) <ul style="list-style-type: none"> <li>● high order byte</li> <li>Read : D0-7 ← OR8-15</li> <li>Write: D0-7 → IR8-15</li> </ul> (2) Word transfer mode (CTR(W/B)=1) <ul style="list-style-type: none"> <li>● word (16 bits)</li> <li>Read : D0-15 ← OR0-15</li> <li>Write: D0-15 → IR0-15</li> </ul>	Active	Interrupt occurs
	0 1 0 0	<u>CTR data transfer</u> Write: D0-7 → CTR0-7	Halt	No interrupt
	1 0 0 0	<u>PC data transfer</u> Write: D0-7 → PC0-7 <ul style="list-style-type: none"> <li>● The value of PC should be in the range of 0 through 255. (D10=1, D9=0, D8=0)</li> <li>● The interrupt mask bit <math>I_M</math> (CTR) is set.              The contents of other registers may be changed.              The contents of the data RAM remain unchanged.</li> </ul>	Halt	No interrupt

(Note) Other contents of F0-F3 than the above are inhibited when the the  $\overline{CS}$  is 0.  
 The start address is (PC) + 1 after the transfer of PC data.

When writing data into the CTR or PC, the F0 - F3 and  $\overline{CS}$  input aborts the internal clock and the current instruction execution is stopped; the HSP is placed in the halt state.

The  $\overline{CS}$  goes to high after the end of CTR/PC data transfer, which makes the HSP leave the halt state and resume the program execution. Special attention is needed not to place the HSP in the halt state for more than 10  $\mu$ s because of the HSP dynamic operation.

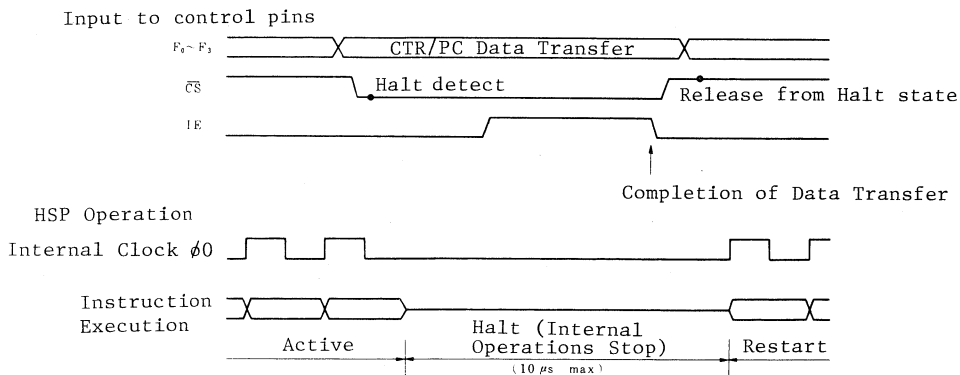


Fig. 3.1.2 HALT OPERATION DURING CTR/PC DATA TRANSFER

### 3.1.2 Serial I/O Functions

Serial I/O mainly interfaces the HSP to A/D and D/A converters. The HSP permits up to 16-bit serial I/O. Table 3.1.2 gives a general description of serial I/O.

#### (1) Serial input

The HSP provides the SICK, SIEN and SI pins for serial inputs. Fig. 3.1.3 is a timing diagram of the serial input. SI is a serial data input pin and SICK is a serial input clock. Data is input to the serial input register (16 bits) from the Most Significant Bit (MSB). If the input data is less than 16 bits, it is shifted automatically with an internal counter so that the MSB of the data goes to the MSB of the SIR. In this case, the serial data shifts synchronously with SICK. Thus the shift operation needs at least 16 clocks of SICK. Usually, a continuous clock signals must be input to the SICK pin.

The rising of SIEN permits the serial input. Once SIEN is enabled, the serial data is completely input independently of the falling of SIEN.



No data transfer to an accumulator is allowed during the serial input. The SIR is cleared when data is transferred to the accumulator.

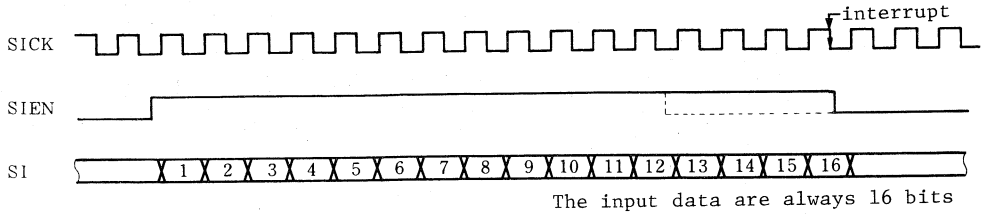


Fig. 3.1.3 SERIAL INPUT TIMING

(2) Serial output

The HSP provides the SOCK, SOEN and SO pins for serial output. The basic timing of the serial output is the same as the serial input timing. SO is a serial data output pin and SOCK is a clock pin for serial output.

Usually, a continuous clock must be input to SOCK. The rising of SOEN enables the serial output. The serial data starts to be output at the rising edge of SOCK. When sock goes to high, data stops to be output and SO goes to the high impedance state. Thus note that the transition to the high impedance state (Hi-Z) is not synchronized with a falling edge of the SOEN. The SOR is cleared after the completion of the serial data output.

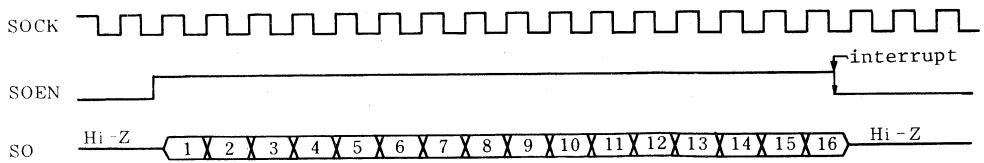


Fig. 3.1.4 SERIAL OUTPUT TIMING

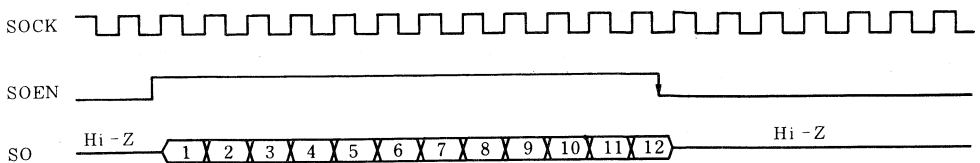


Fig. 3.1.5 SERIAL OUTPUT TIMING (WHEN DATA IS LESS THAN 16 BITS)

Table 3.1.2 SERIAL INPUT/OUTPUT

	Serial Input	Serial Output
Block Diagram	<p>16 bit Internal Data Bus</p> <p>Input Shift Reg (SIR) 16 bits</p> <p>SI</p> <p>SICK</p> <p>Enable</p> <p>Bit Counter 4bits</p> <p>Carry</p> <p>Control circuit</p> <p>SIEN</p> <p>Clock</p> <p>to Interrupt Block</p>	<p>16 bit Internal Data Bus</p> <p>Output Shift Reg (SOR) 16 bits</p> <p>SO</p> <p>SOCK</p> <p>SOEN</p> <p>to Interrupt Block</p>
I/O Data Bit Numbers	<p>MSB</p> <p>LSB</p> <p><math>2^{15}, 2^{14}, \dots, 2^1, 2^0</math></p> <p>SI</p> <p>Input from the MSB.</p>	<p>LSB</p> <p>MSB</p> <p><math>2^0, 2^1, \dots, 2^{14}, 2^{15}</math></p> <p>SO</p> <p>Output from the MSB.</p>
Input/Output Bit Count Control	<ul style="list-style-type: none"> <li>● Even if more than 16 clocks are input while SIEN is active, only the first 16 bits are input and the following bits are disabled.</li> <li>● If there are less than 16 SICK clocks while SIEN is active, the input data is shifted to the higher bits.</li> </ul>	<ul style="list-style-type: none"> <li>● Even if more than 16 clocks are input while SOEN is active, the SO outputs '0's for bits following the first 16 bits.</li> <li>● If the output data are less than 16 bits, SOEN goes inactive when the specified bits are all output.</li> </ul>
Interrupt	The HSP counts 16 clocks after the rising edge of SIEN and enables an interrupt generation circuit when the 16-bit data is all input to the SIR.	The HSP enables an interrupt generation circuit in response to the negative edge of SOEN.
Shift Register Set/Reset	The shift register is cleared (all bits are '0's) after a transfer instruction (SIR → ACC) has been executed.	Data is set in the SOR with the TFR instruction (ACC → SOR).

### 3.2 INTERRUPT

The HSP can generate an interrupt when data is transferred to the HSP to realize the efficient arithmetic operation and data I/O. Fig. 3.2.1 shows a schematic of the interrupt circuit.

#### (1) Interrupt level and factors

The interrupt level is one level. An interrupt is generated by the following three factors:

- (a) the end of parallel I/O data transfer to the IR or from the OR
- (b) the end of serial input data transfer to the SIR
- (c) the end of serial output data transfer from the SOR

These factors are identified by software.

Fig. 3.2.2 shows an interrupt timing for parallel I/O. As  $\overline{CS}$  input enables interrupt, special attention is needed in designing the peripheral circuit in order not to set  $\overline{CS}$  to logic low unnecessarily. Moreover, in the user's program, a write to the OR or a read of the IR in the HSP must be performed after the completion of external data transfer. For the interrupt timing of the serial I/O, see 'Serial I/O Function'.

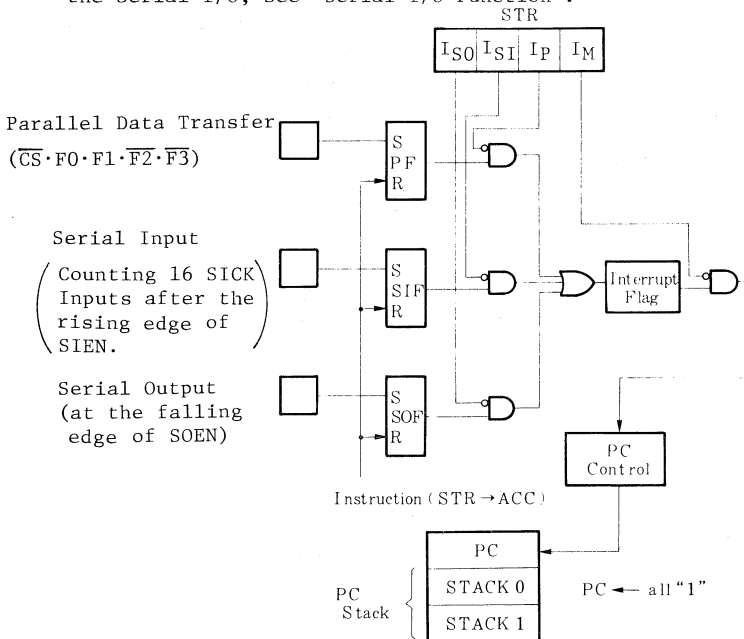
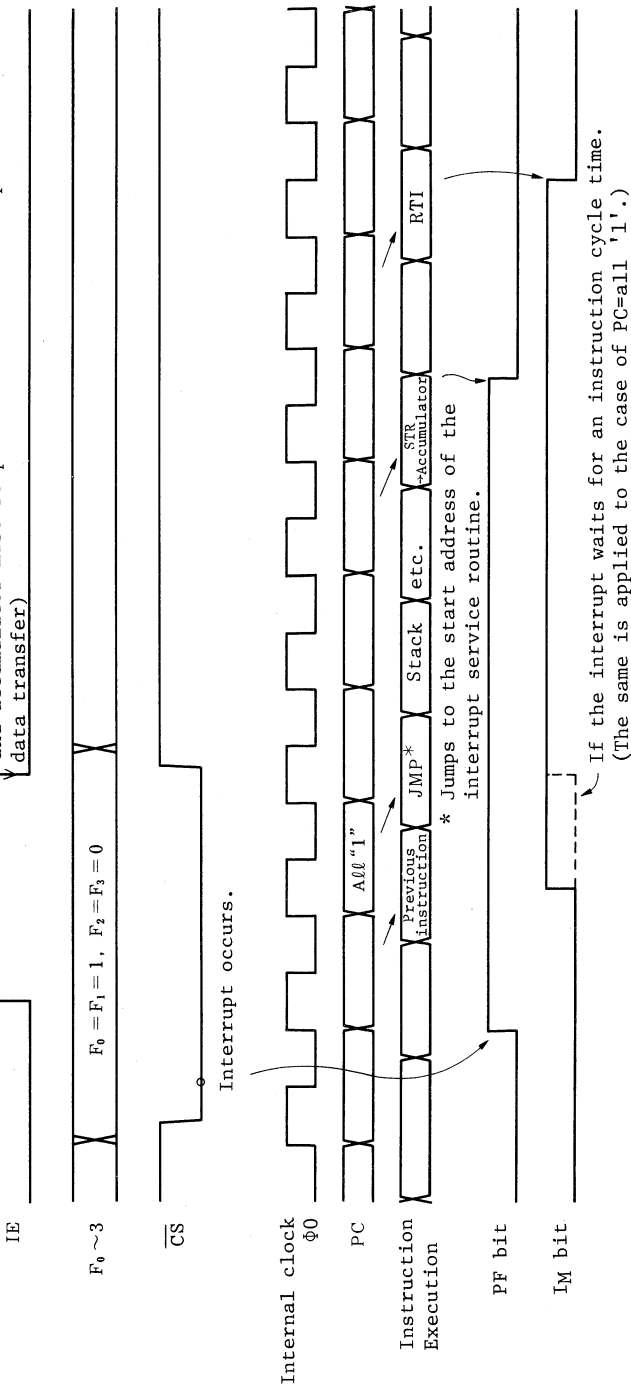


Fig. 3.2.1 SCHEMATIC OF INTERRUPT CIRCUIT

Completion of a data transfer (Data transfer between the IR/OR and accumulator must be performed after the completion of this data transfer)



(NOTE) The interrupt waits for an instruction cycle time if the previous instruction is JMP, etc.

Fig. 3.2.2 PARALLEL I/O INTERRUPT TIMING

(2) Mask

An interrupt is masked with  $I_M$ ,  $I_P$ ,  $I_{SI}$  or  $I_{SO}$  in the status register.

Table 3.2.1 THE FUNCTION OF INTERRUPT MASK BIT

$I_M$	Mask bit for all interrupts. This bit is set to '1' automatically to mask all interrupts when an interrupt occurs. It is cleared to '0' by the RTI instruction and the HSP leaves the interrupt-masked state. $I_M$ can be also set or cleared by the TFR instructions (accumulator $\rightarrow$ STR).
$I_P$	Mask bit for the interrupt at the completion of parallel data transfer through the external data bus. The interrupt is masked when $I_P$ is '1', and is not masked when it is '0'. This bit is set or cleared by the TFR instructions.
$I_{SI}$	Mask bit for the interrupt at the completion of serial data input through the external data bus. The interrupt is masked when $I_{SI}$ is '1', and is not masked when it is '0'. This bit is set and cleared by the TFR instructions.
$I_{SO}$	Mask bit for the interrupt at the completion of serial data output through the external data bus. The interrupt is masked when $I_{SO}$ is '1', and is not masked when it is '0'. This bit is set or cleared by the TFR instructions.

The PF, SIF and SOF bits in the status register can be set by an external input even under these interrupt-masked conditions.

(3) Stack

The program counter (PC) has two stacks. This allows a two-level nesting of interrupt and subroutine.

The contents of other registers are saved in the data RAM by software. In this case, note that the floating point data in an accumulator loses its four low-order bits when it is transferred to the data RAM.

(4) Wait for interrupt

An interrupt sequence is not executed during the execution of a repeat instruction or any instruction repeated by the repeat instruction, or after the execution of a jump instruction (only for jump operation) or return instructions (RTN, RTI). Therefore, the interrupt sequence is never executed with the jump instruction whose destination is the same address.

Repeat instructions : FRPTA, FRPTB, RPTA, RPTB

————— Jump instructions : JCS, JNS, JZS, JSR, JNE, JNZM, JMP —————

Return instructions : RTN, RTI

(5) Vectoring

When an interrupt occurs, \$1FF is set in the PC. This is the last address of the instruction ROM. Therefore, the jump instruction must be placed in this address to jump to the start address of the interrupt service routine. Fig. 3.2.3 gives an example of program sequence. If a program has no interrupt sequence, place the jump instruction in \$1FF to jump to the reset starting address (\$001) in the case of an interrupt error.

- (6) Since the output data of the MULT and the data in the DREG are latched for only one instruction cycle, an interrupt during a pipeline-based product sum operation makes these data ineffective. Therefore, the program must mask interrupts during any DREG/MULT operation.

o Example of Interrupt Service Routine ①

```

INT    NOPA  A,3 32    : Saves the ACCA contents to the data
                        RAM (address 32 of page 3).
      NOPB  A,3,33    : Saves the ACCB contents to the data
                        RAM (address 33 of page 3).
      TFR   STR,A     : Determines the cause of an interrupt.
                        (PF is cleared.)
      SRA   EE,0,00
      JCS   PF        : End of parallel I/O data transfer?
      SRA   EE,0,00
      JCS   SIF       : End of serial input ?

```

```

      .
      .
      .
      .
      JMP   RETN

```

Serial output operation  
(Transfers data to the SOR.)

```

SIF
      .
      .
      .
      .
      JMP   RETN

```

Serial input operation  
(Transfers data from the SIR)

```

PF
      .
      .
      .
      .

```

Parallel I/O operation.  
(Transfers data to the OR or from the IR.)

```

RETN   LDB   YX,EE,3,33: Returns the ACCB contents from the data
                        RAM.
      LDA   YX,EE,3,32: Returns the ACCA contents from the data
                        RAM.
      RTI                   : Returns from interrupt.
                        (Clears IM bit)
      ORG   $1FF          : Interrupt vector address
      JMP   INT           : Jumps to the interrupt service
                        routine.

```

Fig. 3.2.3 EXAMPLE OF INTERRUPT SERVICE ROUTINE

The CCR contents become ineffective in this routine. The contents of the accumulator are saved in the data RAM in the fixed point representation. If necessary, the contents of the STR, CTR, RC, RO, RA and RB must be saved. However, the data of the DREG and the product (P) cannot be saved in the data RAM. Therefore, if using the DREG and P in the main routine, interrupts should be masked.



o Example of Interrupt Service Routine ②

```

INT    TFR   CCR,B      : Saves the CCR contents in the data RAM
                               (address 49 of page 0).
      NOPB  A,0,49
      TFR   STR,B      : Dummy transfer (PF bit is cleared.)
      TFR   IR,B       : Data input
      NOPB  A,0,0      : Stores data in the data RAM
                               (address 00 of page 0)
      LDB   YX,EE,0,49: Returns the CCR contents from the data
                               RAM.
      TFR   B,CCR
      RTI                       : Returns from interrupt. (IM bit is
                               cleared)
      ORG   $1FF       : Interrupt vector address
      JMP   INT        : Jumps to the interrupt service routine
  
```

Fig. 3.2.4 EXAMPLE OF INTERRUPT SERVICE ROUTINE

Since the ACCB contents become ineffective in this routine, the main routine does not use the ACCB.

Interrupts are caused only by parallel I/O data transfer. The contents of the DREG and the product (P) cannot be restored. Therefore, if using the DREG and the P in the main routine, interrupts should be masked.

### 3.3 DMA (DIRECT MEMORY ACCESS)

The HSP can perform DMA operation through the parallel I/O ports D0 - D15. The DMAC (direct memory access controller) for an 8-bit micro-computer 6800 allows the direct transfer of data between the HSP and the memory.

#### o DMA Operation Mode

DMA operation mode can be activated by setting DMA bit in the CTR. In this mode, the data is directly transferred between the HSP and peripherals through the IR or OR and the data transfer between the IR/OR and the data RAM is performed by software.

```

CTR (DMA bit) = 1 ; DMA operation mode
CTR (DMA bit) = 0 ; Non-DMA operation mode
  
```

The HSP provides three pins for DMA operation; TxRQ, TxAK and  $\overline{\text{DEND}}$ . Among some DMA operation modes employed by the DMAC6844, the HSP uses the HALT burst mode. In this mode, the HSP performs a byte data transfer using an 8-bit HD6844 as the DMAC.

o DMA Operation

DMA operation mode is activated by a set of the DMA bit in the CTR. Once the DMA bit is set, the HSP remains in the DMA operation mode until the reset by software or an input of  $\overline{\text{DEND}}$ .

(1) Word transfer

A DMA word transfer is performed by setting W/B, DMA and TxRQ bits in the CTR. When TxAK input is received, TxRQ bit is cleared and PF bit in the status register is set. This generates an interrupt on the completion of parallel I/O data transfer unless interrupts are masked. Data is input to the HSP through the IR and transferred to an accumulator and to the data RAM in the interrupt service routine.

At the completion of the data transfer from the IR to an accumulator, TxRQ bit is set to '1' automatically, which requests a DMA service again. PF bit is set at the completion of the data output from the HSP and an interrupt service routine is started unless interrupts are masked. In this routine, the data is transferred from an accumulator to the OR. At the completion of the data transfer from the accumulator to the OR, TxRQ bit is set to '1' automatically, which requests a DMA service again. The DMA bit is cleared by an input of  $\overline{\text{DEND}}$  after TxRQ bit is set, which is the end of the DMA operation.

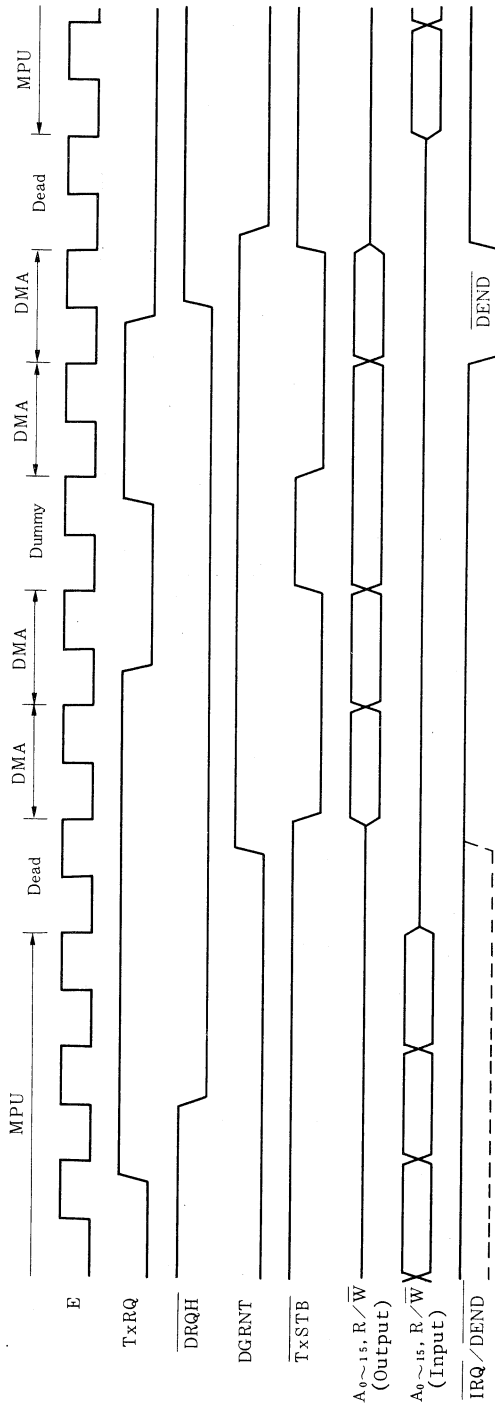
While interrupts are masked, the program monitors the PF bit to detect the completion of data transfer between the IR/OR and the external memory.

(2) Byte transfer

A DMA byte transfer is performed by setting the W/B bit to '0' and the DMA and TxRQ bits to '1'. In this case, the data is divided into two parts; a lower half and an upper half, and data transfer are performed in that order. In the HSP, data is transferred between an accumulator and the IR/OR after the completion of 2-byte data transfer. The DMA byte data transfer employs the same transfer sequence as the word transfer does.



DMAC (HD6844) Timing (HALT Burst Mode)



HSP (HD61810) Timing

$\bar{CS}$  is 'H' and R/ $\bar{W}$  (the direction of transfer through data bus) is reversed.

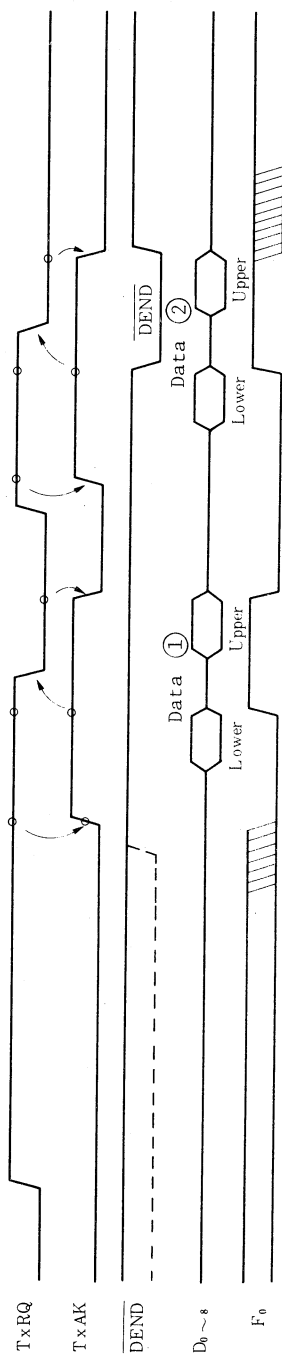
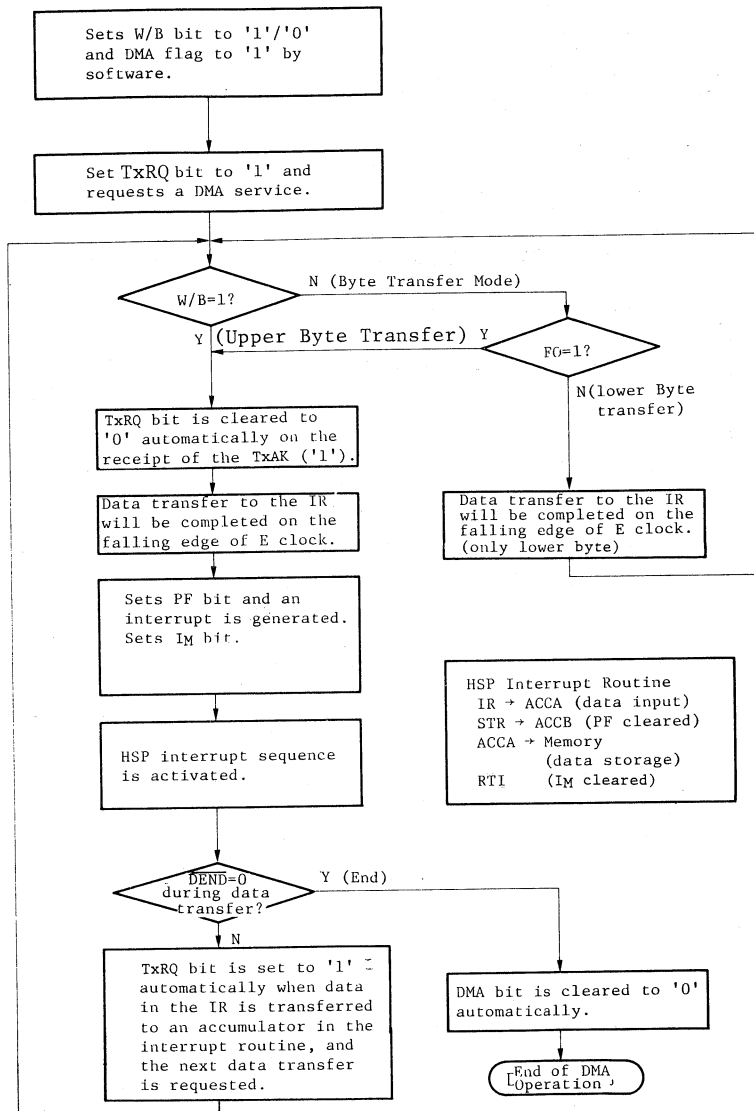
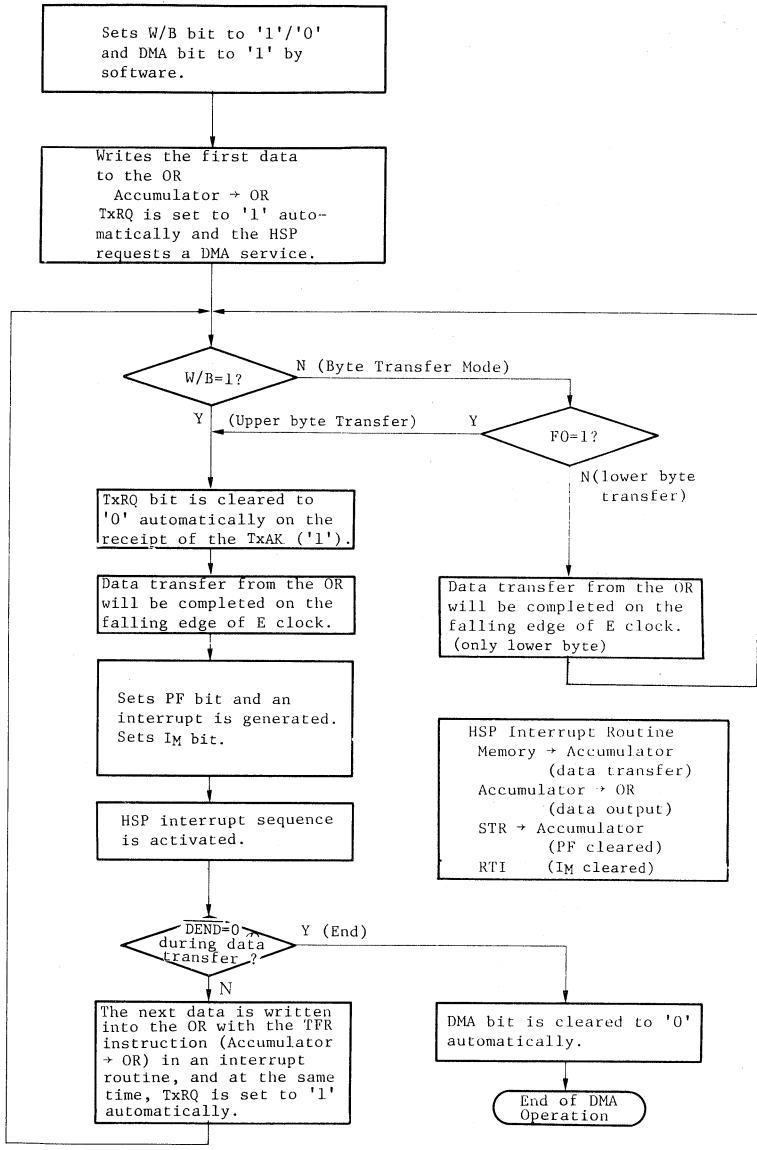


Fig. 3.3.2 DMA I/O TIMING (Byte data transfer)



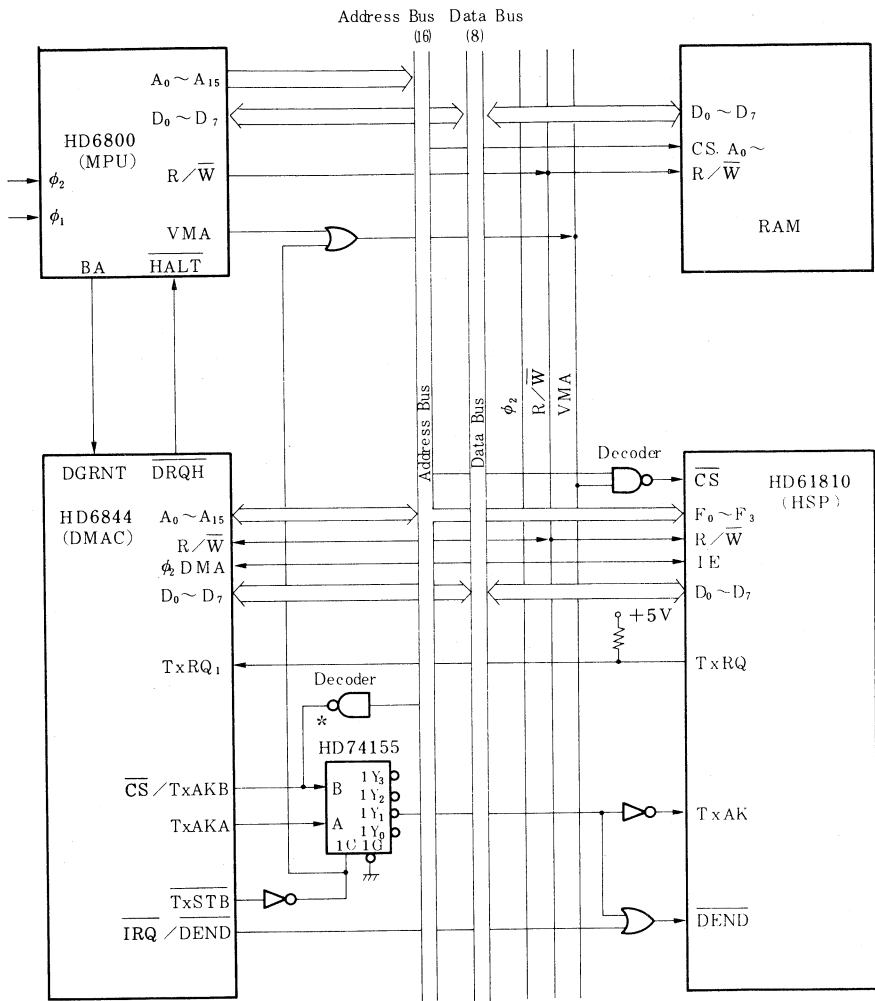
(Note) This flowchart provides minimum required operations. They can be altered depending on purpose. When an interrupt is masked, the HSP monitors the PF bit and latches data in response to setting of the PF bit.

Fig. 3.3.3 DMA OPERATION FLOWCHART (External memory → HSP)



(Note) This flowchart provides minimum required operations. They can be altered depending on purpose. When an interrupt is masked, the HSP monitors the PF bit and transfers the next data to the OR in response to the setting of the PF bit.

Fig. 3.3.4 DMA OPERATION FLOWCHART (HSP → External memory)



\* Open drain

Fig. 3.3.5 DMA SYSTEM CONFIGURATION

### 3.4 BIT I/O, TxRQ

1-bit data can be input and output using BIT I/O and TxRQ bit in the CTR.

The BIT I/O bit can be used as the I/O pin for 1-bit data as shown in Fig. 3.4.1.

#### (1) When used as an input pin

Write '1' into the BIT I/O bit beforehand with the TFR instruction (Accumulator  $\rightarrow$  CTR), and the output MOS transistor of BIT I/O is turned off to allow an external data to be entered. The input data is transferred to an accumulator through the data bus with the TFR instruction (CTR  $\rightarrow$  Accumulator). Note that it is necessary to hold the input data during instruction execution.

#### (2) When used as an output pin

As an output of the BIT I/O is open drain, a pull-up resistor must be connected externally. This output signal can control the HSP peripherals, and can also interrupt the MPU.

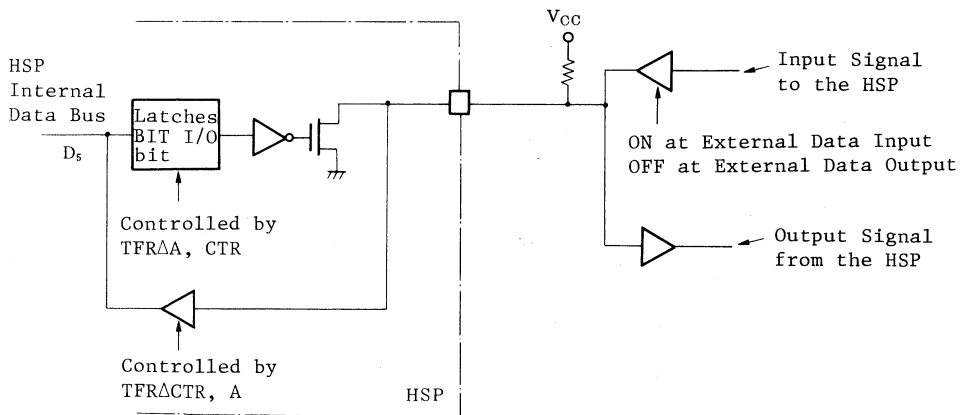


Fig. 3.4.1 FUNCTION OF BIT I/O

In the non-DMA operation mode, TxRQ can be used as a 1-bit output pin. Because of the open drain output, it also needs a pull-up resistor connected externally. However, TxRQ cannot be used as an input pin. When data is input with the TFR instruction (CTR  $\rightarrow$  Accumulator), the value written in TxRQ bit is read out independently of the external state ('H' or 'L').



**SECTION 4**  
**ARITHMETIC OPERATION**



## 4. ARITHMETIC OPERATION

The speech processing and the signal processing in telecommunications specially need high speed and high precision. The HSP provides a floating point arithmetic unit on a single chip to meet these needs.

### 4.1 DATA FORMAT

Data formats in fixed point and floating point arithmetics are illustrated in Fig. 4.1.1.

#### (1) ALU

##### (a) Fixed point data

(i) Binary representation : Represents the value of  $0 \sim 2^{16}-1$ .

(ii) Two's complement representation : The most significant bit corresponds to a sign bit.

##### (b) Floating point data

Floating point data consists of a mantissa part and an exponent part using two's complement representation. The most significant bit of each part corresponds to a sign bit. Decimal point is set between bit 15 (MSB) and bit 14 of a mantissa.

Mantissa : Represents the value of  $-1 \sim 1-2^{-15}$ .

Exponent : Represents the value of  $-8 \sim +7$ (integral number).

#### (2) Multiplier

##### (a) Fixed point data

Fixed point multiplication is performed using two's complement representation. The most significant bit corresponds to a sign bit. Input data is 12 bits and output data is 16 bits.

##### (b) Floating point data

Floating point multiplication is performed using two's complement representation. A mantissa of input data is 12 bits and of output data is 16 bits. An exponent part of input/output data is 4 bits.

Decimal point is set between bits 15 and 14.

Mantissa: Input data — Represents the value of  $-1 \sim 1-2^{-11}$ .

Output data — Represents the value of  $-1 \sim 1-2^{-15}$ .

Exponent: Represents the value of  $-8 \sim +7$ (integral number).

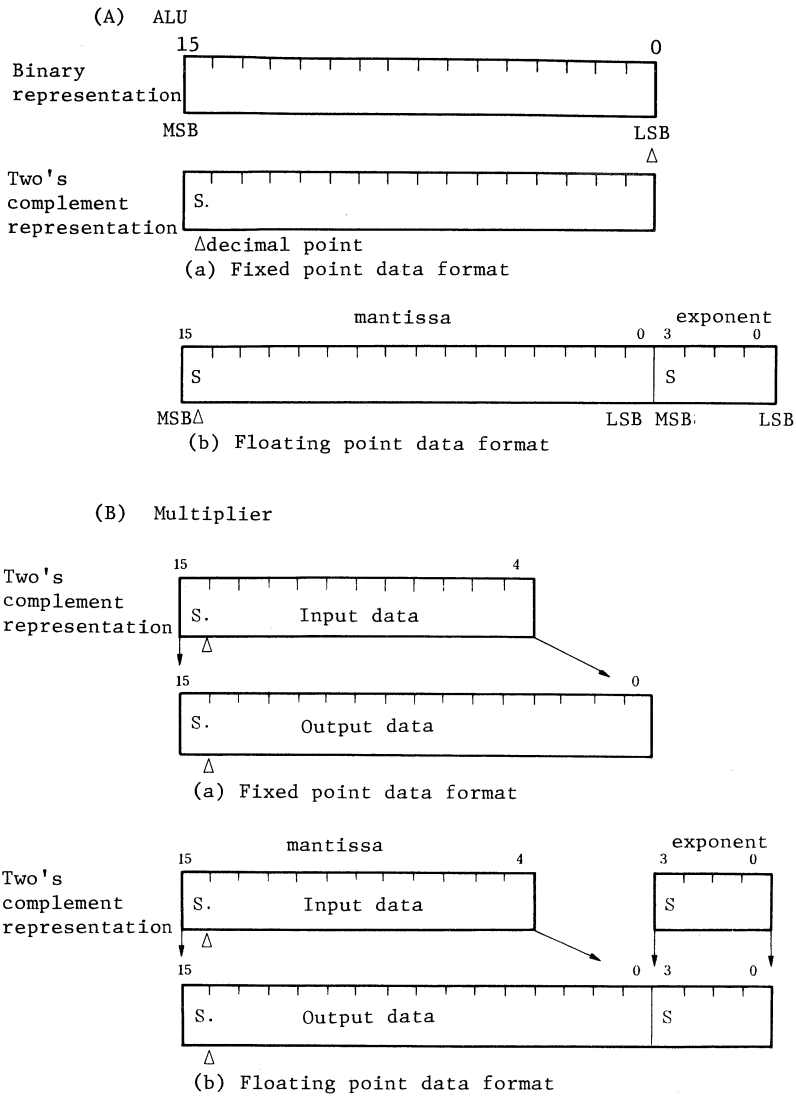


Fig. 4.1.1 DATA FORMAT

## 4.2 FIXED POINT ARITHMETIC

### 4.2.1 Fixed Point ALU

The HSP can perform ALU operations in either fixed point or floating point representation.

Fixed point data is always represented with 16 bits, and an exponent part used in floating point data representation is undefined. The HSP instructions select fixed/floating data representation. For details, see 'INSTRUCTION'. Fixed point ALU operation is performed in binary representation or two's complement representation.

#### (1) ALU input/output data format

The ALU input/output data is represented with 16 bits. Fig. 4.2.1 and 4.2.2 illustrate I/O data format.

(An undefined exponent part is also shown.)

- ALU input

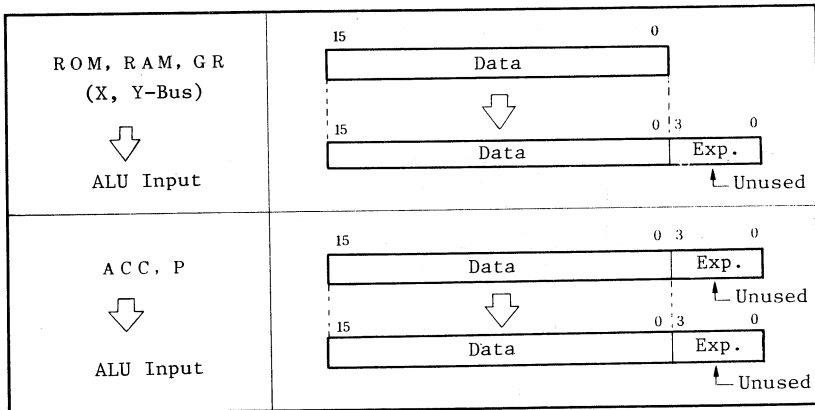


Fig. 4.2.1 ALU INPUT DATA FORMAT

- ALU output

A result of the ALU operation is output to an accumulator. When the result is stored into the RAM/GR, the data format is described in Fig. 4.2.2.

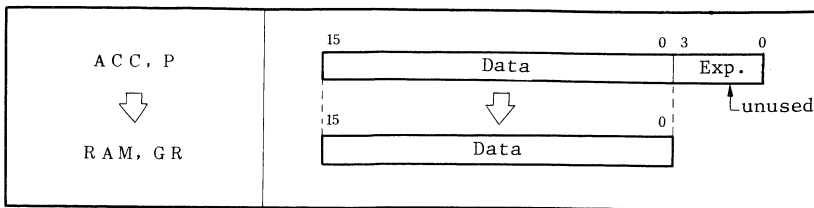
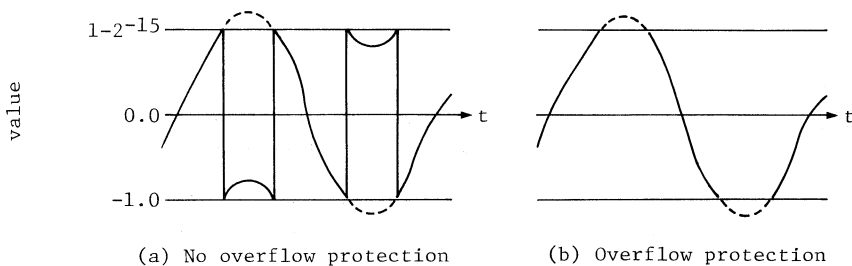


Fig. 4.2.2 ALU OUTPUT DATA FORMAT (RAM, GR)

(2) Overflow

An overflow occurs when an operation yields a result beyond the range of representable value. In this case, the OVFP bit in the CTR enables an overflow protection. If the OVFP bit is set to one, an overflow is protected. If set to zero, an overflow protection is cleared.

An overflow protection fixes the positive value more than  $1-2^{-15}$  to  $1-2^{-15}$ , and the negative value less than  $-1$  to  $-1$ . In this case, data is represented in two's complement.



- o A dotted line shows a result from the ALU, and a solid line shows an input of the accumulator.
- o This figure shows filter outputs every sampling period with time (t) for horizontal axis.

Fig. 4.2.3 OVERFLOW PROTECTION

#### 4.2.2 Fixed Point Multiplication

The HSP provides an exclusive multiplier (MULT) for a high-speed multiplication.

##### (1) Fixed point multiplication system

The MULT performs a multiplication in two's complement representation. The input data is 12 bits, and the output data is 16 bits, which produces a result including truncation error.

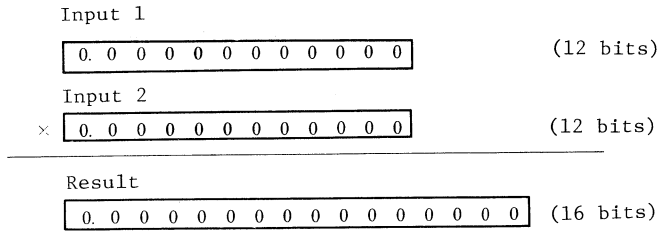


Fig. 4.2.4 MULTIPLIER RESOLUTION

Multiplication is performed according to quadratic Booth algorithm as shown in the following formula.

$$Z = X \cdot Y = \sum_{i=0}^5 (y_{2i+3} + y_{2i+4} - 2y_{2i+5}) \cdot X \cdot 2^{2i} = \sum_{i=0}^5 P_i \cdot 2^{2i}$$

The partial sum  $P_i$  is obtained by the blocks in Fig. 4.2.5.

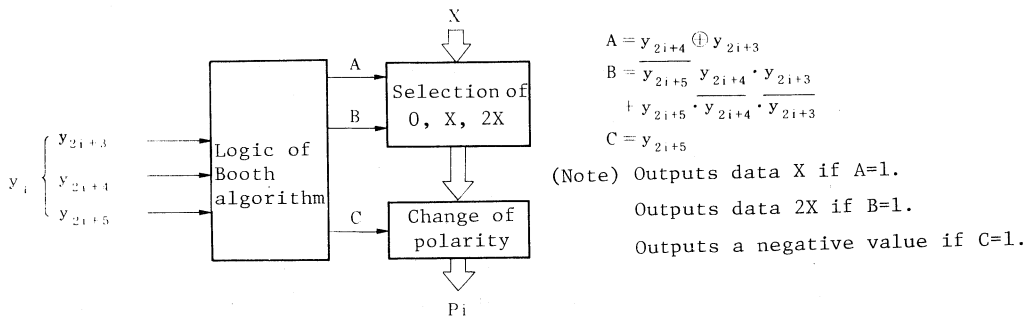


Fig. 4.2.5 PARTIAL SUM BLOCK DIAGRAM

The MULT operation is illustrated in Fig. 4.2.6.

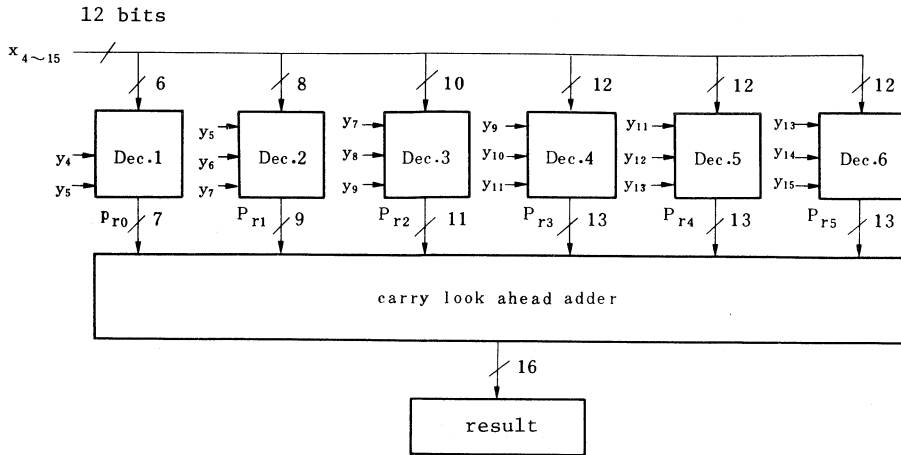


Fig. 4.2.6 MULT BLOCK DIAGRAM

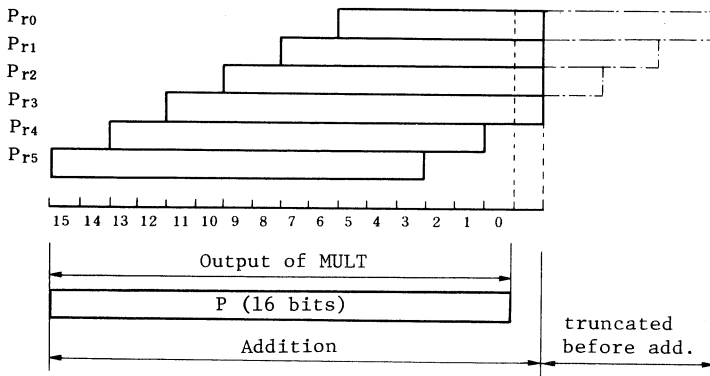


Fig. 4.2.7 ADDING OF PARTIAL PRODUCT IN MULT

Fig. 4.2.7 illustrates a sequence of adding of partial products. An truncation error is included in the result because the lower 6 bits are not added. As an output result is 16 bits, the LSB is neglected after the add operation. In this case, this result has a truncation error of max.  $\pm 2/2^{16}$ , compared with the case when counting fraction over 1/2 as one and disregarding the rest of the LSB.

If input data X is \$000 and Y is not \$000, the product may not be \$0000. If Y is also \$000, the product is \$0000.



(2) MULT input/output

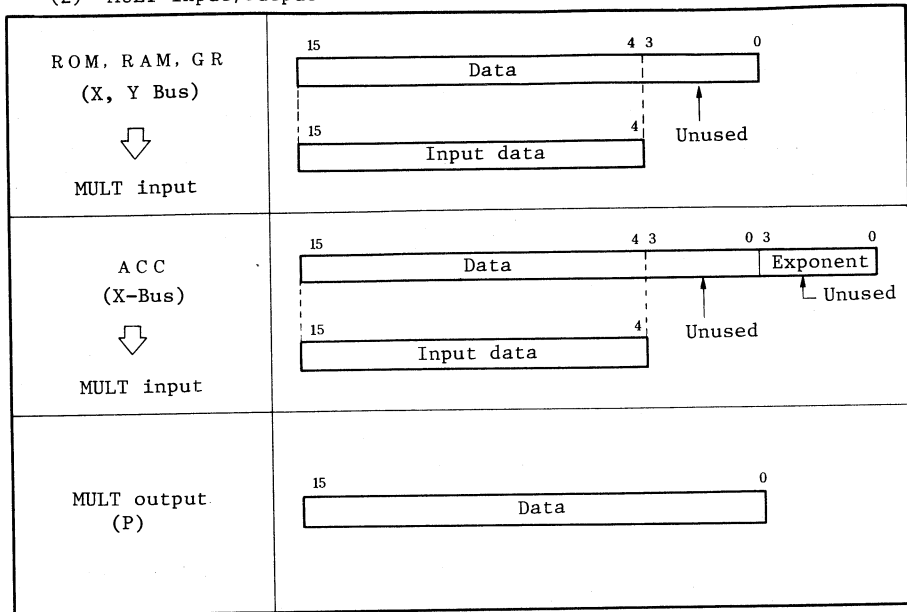


Fig. 4.2.8 MULT INPUT/OUTPUT DATA FORMAT

(3) Multiplication

The MULT always operates and receives data from both the X-Bus and the Y-Bus (free-running multiplier). However, in the HSP, the only ALU operation instructions can specify the X- and Y-Bus data. Therefore, the multiplication should be performed with the ALU operation instructions.

A product (P) of multiplication is used in the next instruction cycle. If the product is represented in the fixed point, the next instruction should be executed in the fixed point representation. Therefore, when the present instruction is a fixed point one, the next instruction, if (P) is used in it, should also be a fixed point one. The same applies in the floating point instructions.

Table 4.2.1 THE MULT INPUTS BY ALU OPERATION INSTRUCTIONS

Instruction Format		MULT Input		Note
Addressing Mode	Memory Output	X-Bus	Y-Bus	
Pointer addressing mode	XY(n, m)	ROM/RAM data (n page)	ROM/RAM data (m page)	n=m if X-Bus and Y-Bus data are read from data ROM.
	XG(n, ℓ)	ROM/RAM data (n page)	GR (ℓ)	
Direct addressing mode	—	ACC	ROM/RAM data (direct address)	

\* A product (P) is used in the next instruction.  
 For details of ALU operations, see 'INSTRUCTION SET'.

(4) Overflow

The MULT performs a multiplication in two's complement representation. When -1 and -1 are multiplied together, the result is +1 and an overflow occurs. In this case, the result is fixed to the maximum positive value ( $1-2^{-15}$ ).

### 4.3 FLOATING POINT ARITHMETIC

#### 4.3.1 Floating Point ALU (FALU)

The HSP provides a floating point ALU (FALU) to perform floating point operations. The FALU has two inputs (20 bits) which consists of a mantissa (16 bits) and an exponent (4 bits). A result is 20 bits and is delivered to an accumulator (ACC).

##### (1) FALU input/output

- FALU input

The data ROM/RAM and GR is 16-bit width, while the input to FALU should be 20-bit. Therefore, '0's are added for lower four bits for for an input to FALU. If a source register is an accumulator or the P register, 20-bit data is input to the FALU in the same bit length.

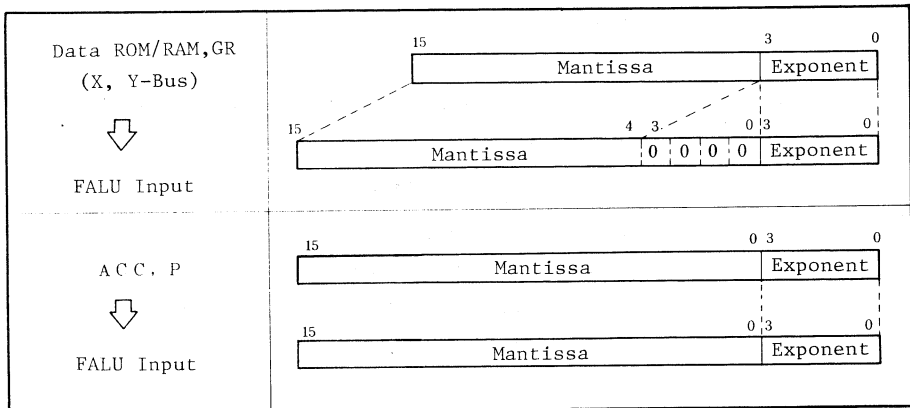


Fig. 4.3.1 FALU INPUT DATA FORMAT

- FALU output

The contents of the FALU are transferred to the ACCA or ACCB. This 20-bit data of the accumulator can be transferred to the FALU in the same bit length. However, if the 20-bit data is transferred to the data RAM or GR (16 bits), lower 4 bits of a mantissa is truncated.

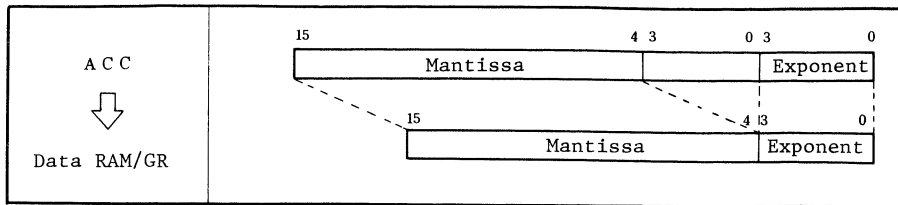


Fig. 4.3.2 ACCUMULATOR → DATA RAM/GR DATA TRANSFER FORMAT

(2) FALU operation

The FALU provides functions of digit adjustment and normalization in addition to the fixed point ALU to perform a floating point arithmetic operation.

The following formula represents a floating point arithmetic operation.

$$\text{input data } A_1 = a_1 \cdot 2^{e_1}, \text{ input data } A_2 = a_2 \cdot 2^{e_2}$$

( $a_1, a_2$  : mantissa,  $e_1, e_2$  : exponent)

In case of floating point add, data digit is adjusted with the exponent part as follows;

$$A_1 + A_2 = a_1 \cdot 2^{e_1} + a_2 \cdot 2^{e_2} = a_1 \cdot 2^{e_1} + a_2' \cdot 2^{e_1} \quad (e_1 \geq e_2)$$

In this case,  $a_2' = a_2 \cdot 2^{e_2 - e_1}$  and the value of  $a_2$  is shifted right ( $e_1 - e_2$ ). After that, the ALU performs an add operation with these data. The result is normalized, which means that a mantissa is fixed to the maximum value.

$$A_1 + A_2 = (a_1 + a_2') \cdot 2^{e_1} = a \cdot 2^e$$

Fig. 4.3.3 shows a sequence of a floating point ALU operation.

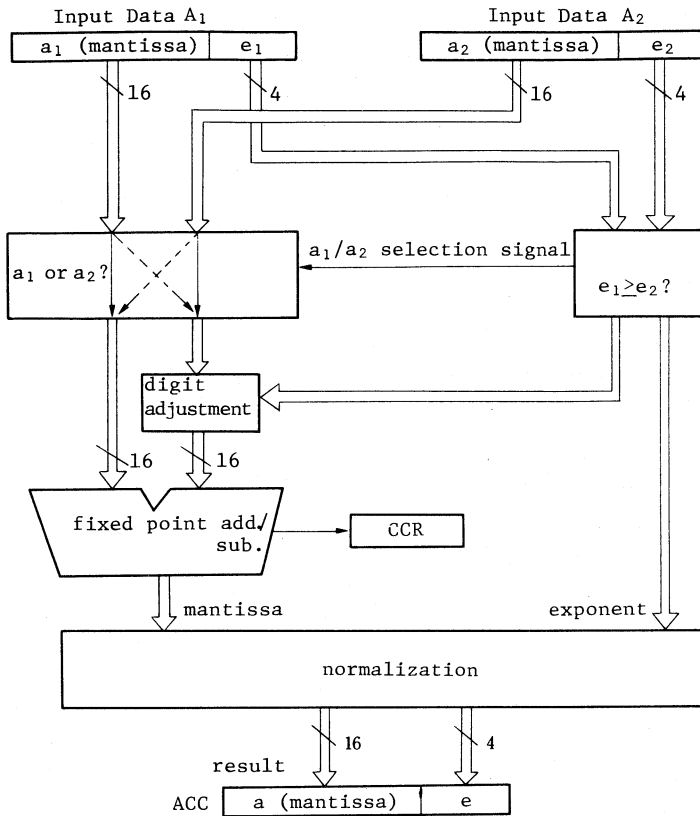


Fig. 4.3.3 FLOATING POINT ALU (FALU)

(3) Overflow, underflow

An overflow or underflow occurs when an operation yields a floating point result beyond the range of  $-8$  to  $+7$  of an exponent part.

(a) Overflow

The FALU must always perform an overflow protection during floating point arithmetic operation. (The OVFP bit of the control register must be set to '1' beforehand.)

When an operation yields a result beyond the range of negative and positive maximum value representable with 20 bits, an overflow is protected by fixing the data to the maximum value.

An overflow protection fixes a positive value to  $(1-2^{-15}) \cdot 2^7$  and a negative value to  $(-1) \cdot 2^7$ .

(b) Underflow

An underflow occurs when an exponent data is less than -8 ( $e < -8$ ). An underflow cannot permit normalization of data and fixes the exponent part to -8. Therefore, the full mantissa precision is not maintained.

Fig. 4.3.4 gives an example of an underflow in floating point subtraction.

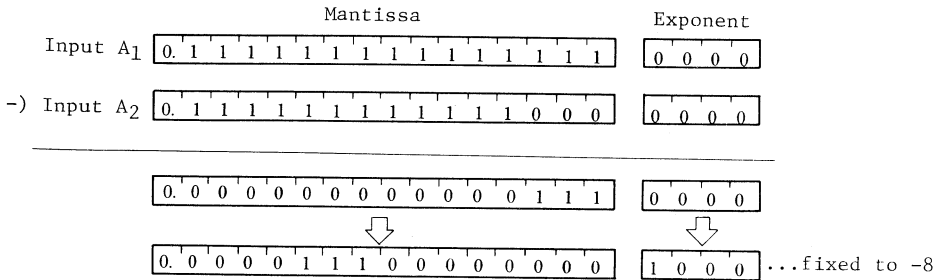
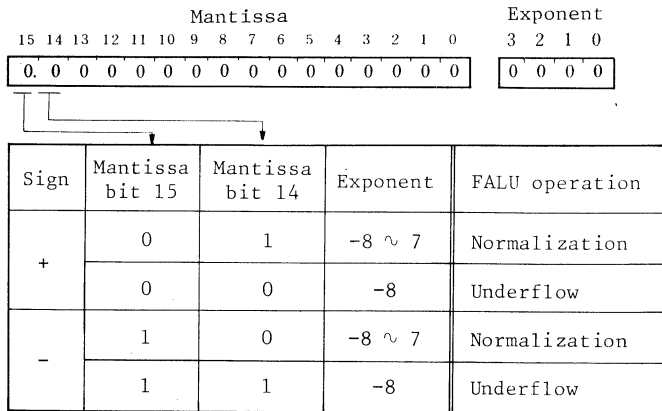


Fig. 4.3.4 AN UNDERFLOW REPRESENTATION

If both exponent parts of data are fixed to -8, the ALU performs a fixed point arithmetic operation in appearance. (except when an exponent part of a result is not -7.)

Fig. 4.3.5 shows results when an underflow occurs.



\* In this figure, an overflow is not taken into consideration.

Fig. 4.3.5 RESULTS WHEN UNDERFLOW OCCURS

(4) The contents of the condition code register (CCR)

The contents of the CCR depends on a mantissa value of a result after digit adjustment.

(5) FALU arithmetic errors

(a) Error in digit adjustment

If exponents of FALU two inputs differ in value, a mantissa of smaller data is shifted right by digit adjustment circuit and its lower bits are truncated, which is caused by 16-bit arithmetic operation.

The repetition of this operation causes an accumulation of errors.

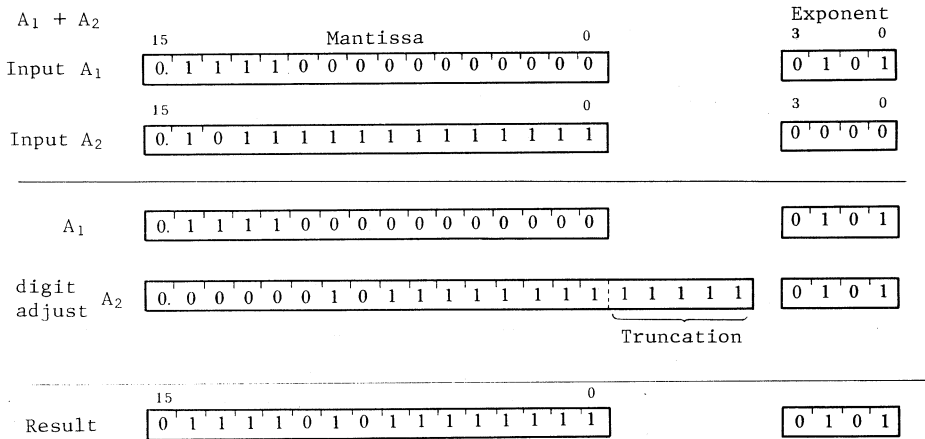


Fig. 4.3:6 AN EXAMPLE OF TRUNCATION AFTER DIGIT ADJUSTMENT

(b) An error in subtraction

Floating point subtraction can be performed as follows;

input data  $A_1 = a_1 \cdot 2^{e_1}$ , input data  $A_2 = a_2 \cdot 2^{e_2}$

( $a_1, a_2$ : mantissa,  $e_1, e_2$ : exponent)

$$A_1 - A_2 = a_1 \cdot 2^{e_1} - a_2 \cdot 2^{e_2} = \begin{cases} a_1 \cdot 2^{e_1} + a_2'' \cdot 2^{e_1} & (e_1 > e_2) \\ a_1' \cdot 2^{e_2} + a_2' \cdot 2^{e_2} & (e_1 \leq e_2) \end{cases}$$

Each mantissa part is;

$$a_1' = a_1 \cdot 2^{e_1 - e_2} \quad (\text{Shift a right } (e_2 - e_1) \text{ bit position})$$

$$a_1' = \bar{a}_2 + 2^{-15} \quad (\text{Two's complement of } a_2)$$

$$a_2'' = (\bar{a}_2 + 2^{-15}) \cdot 2^{e_2 - e_1} \quad (\text{Shift a right } (e_1 - e_2) \text{ bit position})$$

However, in case of  $e_1 > e_2$ ,

$$a_2'' \doteq (\bar{a}_2) \cdot 2^{e_2 - e_1} \quad (\bar{a}_2: \text{one's complement of } a_2)$$

Errors occur caused by this approximate value and digit adjustment described in (a).

(c) Accumulated errors

Errors are accumulated as arithmetic operation is performed with digit adjustment and approximation of value repeatedly. If the FALU input data is not normalized, the data must be once saved in an accumulator to normalize them. In case of using the product (P) as FALU input data, the contents of product must be normalized in order to decrease errors.

Errors from subtraction are prevented by the execution of the NEG instruction and addition.



### 4.3.2 Floating Point Multiplier

#### (1) Floating point multiplication system

The floating point multiplier (FMULT) consists of the MULT and an adder of exponent parts.

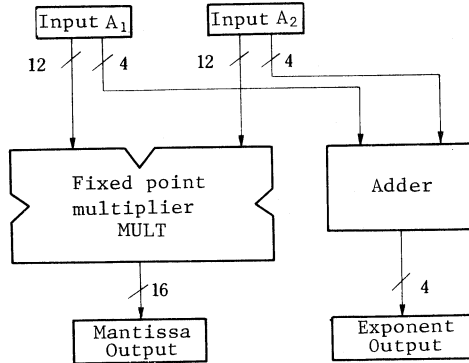


Fig. 4.3.7 BLOCK DIAGRAM OF THE FMULT

Input/output data format is as follows;

mantissa      12 bits × 12 bits → 16 bits  
 exponent      4 bits × 4 bits → 4 bits

The floating point multiplication is performed in the same way with the fixed point multiplication.

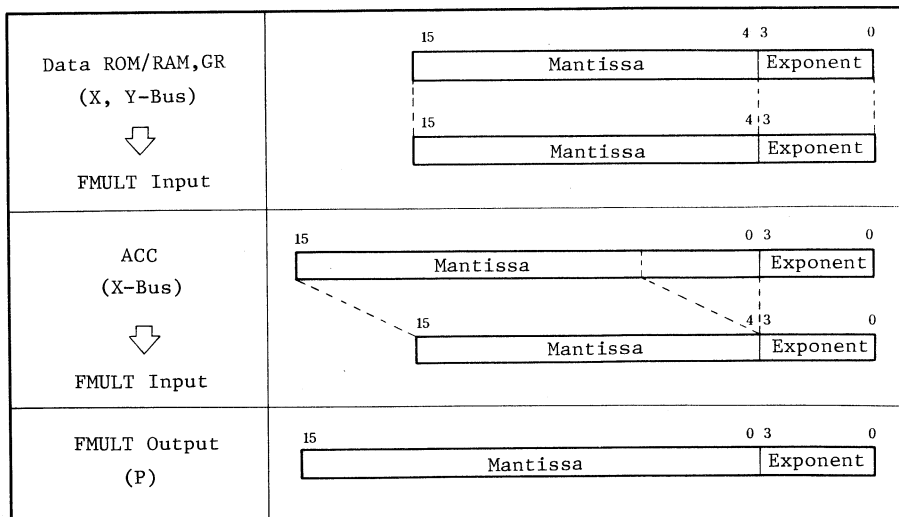


Fig. 4.3.8 FMULT INPUT/OUTPUT DATA FORMAT

(3) Overflow, underflow

The HSP protects an overflow/underflow on an exponent part and overflow on a mantissa part.

(a) An overflow of an exponent part

An overflow occurs on an exponent part ( $e$ ) when  $e \geq +8$  resulting from adding two exponent parts in floating point multiplication.

- ① If a mantissa of a result is not normalized and  $e = +8$ , the mantissa is shifted left 1 bit and the exponent part is fixed to +7.

If the exponent is more than +8, the result is fixed to the maximum value representable. (The sign does not change.)

- ② If a mantissa of a result is normalized, the mantissa is fixed to the maximum value representable and the exponent part is fixed to +7.

(b) An underflow of an exponent part

An underflow occurs when an exponent part of the result is less than -8 in floating point multiplication. An underflow protection fixes the exponent part to -8 and shifts a mantissa right  $(-8-n)$  bit position. This shift operation is executed in the FALU.

(c) An overflow of a mantissa

A result of multiplication of -1 and -1 is fixed to the maximum positive value  $(1-2^{-15})$  when only mantissa is effective.

(4) FMULT arithmetic errors

The FMULT utilizes the fixed point multiplier (MULT) for arithmetic operation of a mantissa. The FMULT input data must be normalized in order to minimize errors. However, usual input data to the FMULT are automatically normalized before input.

#### 4.4 DATA TRANSFORMATION

The HSP inputs or outputs data in the fixed point representation, while it performs operations in the floating point representation for higher precision. When a fixed point data is input from an A/D convertor to the HSP, the data is transformed to the floating point data and the HSP performs an arithmetic operation with these data. A result of the arithmetic operation is transformed to the fixed point data before the output to a D/A convertor. Transformation between the fixed point representation and the floating point representation can be efficiently performed with a single instruction using a constant in the data memory (data ROM/RAM) as a transformation scaling constant.

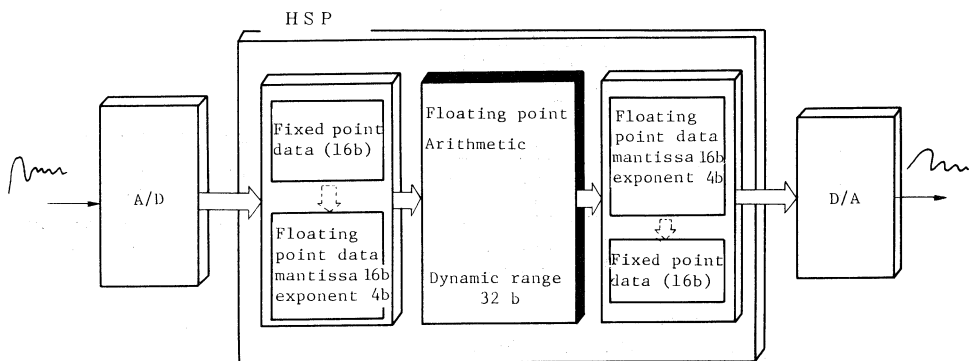


Fig. 4.4.1 DATA TRANSFORMATION

##### (i) fixed point data → floating point data

The fixed point data (16 bits) of an accumulator is transformed to the floating point data (20 bits) using an exponent part of Y-Bus data specified by an instruction as a scaling constant. The FLTA or FLTB instruction performs this transformation.

Normalization with a scaling factor is represented in the following formula.

$$A_1 \times 2^{n_1} \quad (\therefore A_1 \cdot 2^{n_1} = a \cdot 2^N)$$

A : fixed point input data

n : scaling constant

A<sub>1</sub>: a mantissa after transformation

n<sub>1</sub>: an exponent part after transformation

Floating point data ( $A \times 2^n$ ) is normalized, and the normalized data ( $A_1 \times 2^{n_1}$ ) is stored in an accumulator.

Fig. 4.4.2 shows a sequence of transformation from the fixed point data to the floating point data. An example of data transformation is given in Fig. 4.4.3.

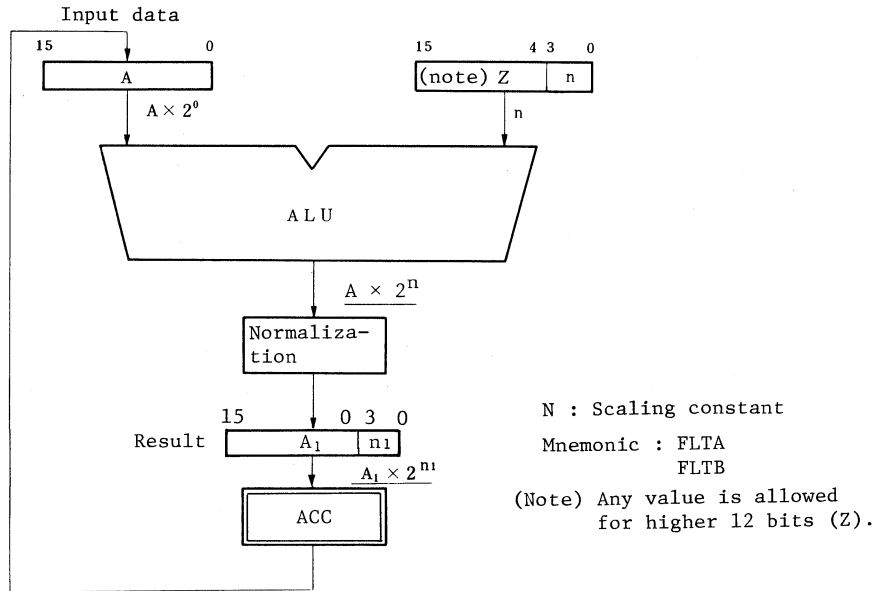
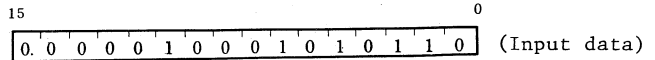


Fig. 4.4.2 DATA TRANSFORMATION  
(Fixed Point Data  $\rightarrow$  Floating Point Data)

Fixed point input  
data \$0456



Transformation Instruction  
(Using Y-Bus data  
\$0000 as a  
scaling constant)

FLTA EE 0, 00

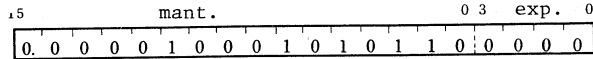
(Data in location  
(0,00) is \$0000.)

Floating point  
data

\$04560

\$4560C

\$4560 × 2<sup>-4</sup>



(Normalization)

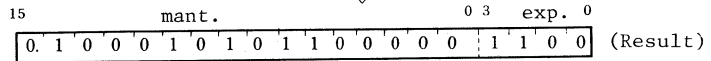


Fig. 4.4.3 AN EXAMPLE OF DATA TRANSFORMATION  
(Fixed Point Data → Floating Point Data)

(ii) floating point data → fixed point data

The floating point data (20 bits) of an accumulator is transformed to the fixed point data (16 bits) with an exponent part (4 bits) of Y-Bus data specified by an instruction as a scaling constant. The FIXA or FIXB instruction performs this transformation.

Normalization with a scaling factor is represented in the following formula.

$$B \times 2^m \rightarrow B_1 \times 2^l \quad (\therefore B_1 = B \cdot 2^{m-l})$$

- B : a mantissa of floating point input data
- m : an exponent part of floating point input data
- l : a scaling constant (an exponent part of Y-Bus output data)
- B<sub>1</sub>: fixed point data after transformation

Data  $B \times 2^m$  is normalized with a scaling factor  $2^l$ . If  $l \geq m$ , a mantissa is shifted right  $(l-m)$  bit position to be a  $B_1$ . If  $l < m$ , a mantissa is shifted left  $(m-l)$  bit position to be a  $B_1$ . If an overflow occurs, mantissa  $B_1$  is fixed to positive or negative maximum value.

A mantissa of Y-Bus data (12 bits) must be set to zero (\$000). After normalization,  $2^{\ell}$  is neglected and only  $B_1$  is stored in the accumulator. When  $\ell$  is zero, the floating point data equals the fixed point transformed data.

Fig. 4.4.4 shows a sequence of transformation from the floating point data to the fixed point data. An example of data transformation is given in Fig. 4.4.5.

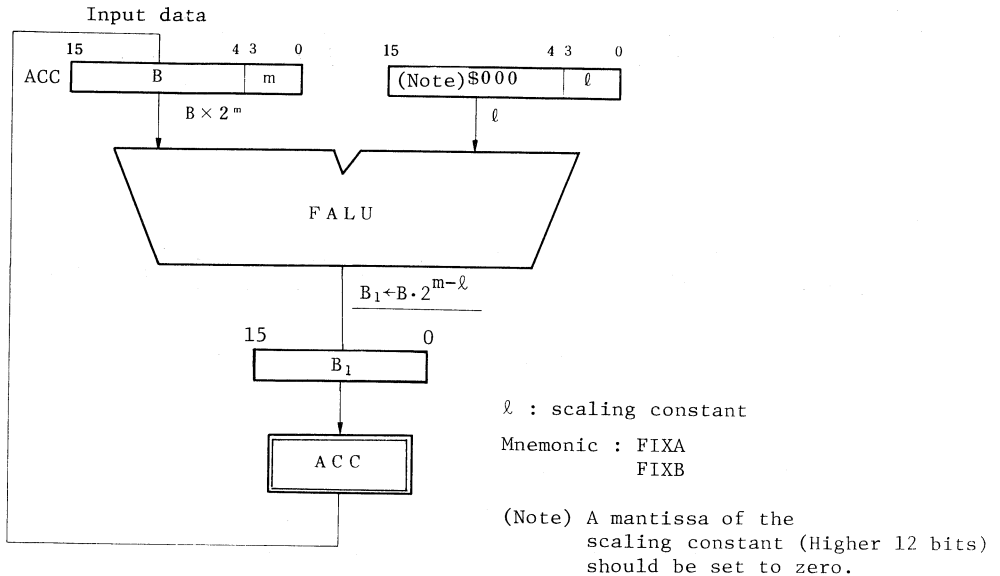
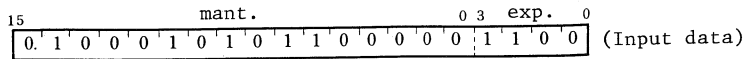


Fig. 4.4.4 DATA TRANSFORMATION  
(Floating Point Data  $\rightarrow$  Fixed Point Data)

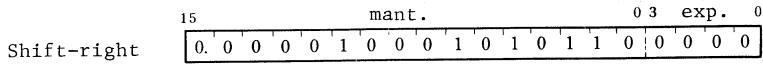
Floating point data  
 \$4560C  
 = \$4560 × 2<sup>-4</sup>



Transformation instruction  
 (Using Y-Bus data  
 (\$0000=\$000 × 2<sup>0</sup>)  
 as a scaling constant)

FIXA EE 0, 00

(Data in location)  
 (0,00) is \$0000.)



Fixed point data  
 \$0456

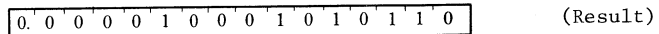
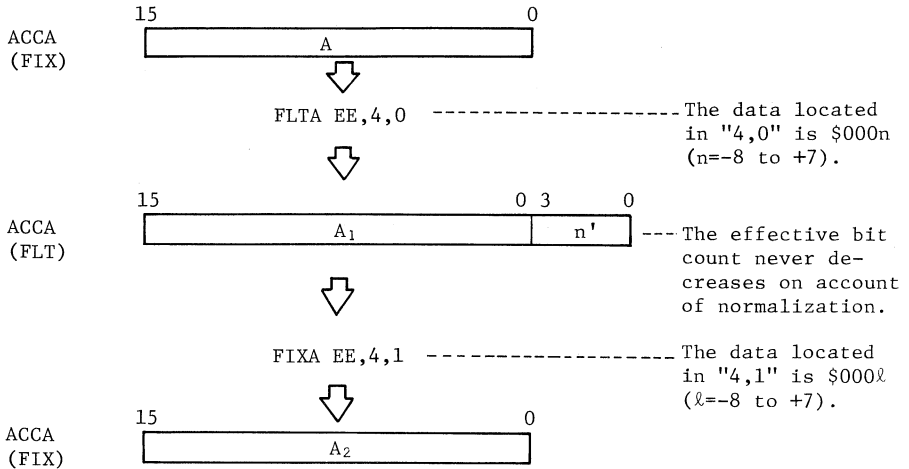


Fig. 4.4.5 AN EXAMPLE OF DATA TRANSFORMATION  
 (Floating Point Data → Fixed Point Data)

(iii) Data shift using data transformation

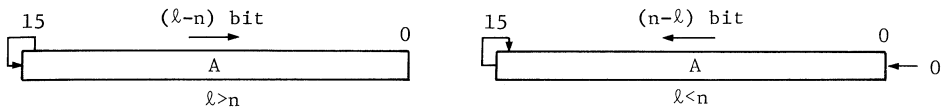
The HSP has no instruction to shift data  $n$  bits in high speed. However, the shift operation can be performed freely with data transformation between floating point and fixed point.

16-bit fixed point data in the ACCA can be shifted  $n$  bits arithmetically by transforming it to the floating point data and then back into the fixed point data as shown in the following.



shift bit count: shift right of  $(l-n)$

Equivalent bit manipulation



Data can be shifted in the range of  $-15$  to  $+15$  bits.

Keep it in mind that an overflow can be protected in case of the left shift. A left shift of  $(n-l)$  bits is equivalent to the arithmetic  $A_2 = A \times 2^{(n-l)}$ , and if the result is out of the range of  $-1.0$  to  $+1.0 \cdot 2^{-15}$ , it is fixed to the maximum value.

(ex.) o data  $\$3000$

a shift left of 1  $\rightarrow$   $\$6000$

a shift left of 2  $\rightarrow$   $\$7FFF$

o data  $\$F0F0$

a shift left of 3  $\rightarrow$   $\$8780$

a shift left of 4  $\rightarrow$   $\$8000$



**SECTION 5**  
**INSTRUCTION**



## 5. INSTRUCTION

### 5.1 GENERAL DESCRIPTION

The HSP provides 77 instructions. These instructions can execute the HSP arithmetics effectively in just 250ns cycle.

#### (1) Data format

The HSP provides fixed point instructions and floating point instructions. This allows the HSP to execute floating point arithmetics. The transformation between the fixed and floating data forms can be performed in a single step of transformation instructions. An overflow is automatically protected in ALU arithmetic operations.

#### (2) Horizontal microprogram

The HSP provides an efficient and easily programmed ALU instruction set. The high throughput is the result of a horizontal microprogram. This instruction allows the execution of some operations in parallel in a single instruction cycle.

An ALU operation instruction permits the following operations:

- (a) ALU operation
- (b) MULT operation
- (c) Read of memory
- (d) Write to memory
- (e) Autoincrement of the address pointers
- (f) Autodecrement of the repeat counter

#### (3) Multiplication

As the HSP multiplier always operates, the HSP has no MULT instruction. Once two data are input to the MULT, the multiplication is executed. The input data are from the data ROM/RAM, GR, or an accumulator. In this case, the data may be input to the ALU at the same time depending on the instruction. The result of the multiplication can be used as the P contents (product) in the next instruction cycle.

When the ALU operates in the floating point form, the multiplication is also executed in the floating point form. In the same way, when the ALU operates in the fixed point form, the multiplication is executed in the fixed point form. Therefore, if the next instruction uses the product of the multiplication, the arithmetic should be performed in the same form (FIX/FLT) as shown in the following examples of program.

- ex. (i) Correct description
- ```

FNOPA EE,XY(0,1),RA+ ; floating point multiplication
FADA PA,EE,XY(0,1),RA ; addition of product (floating
                        point addition)

```
- (ii) Incorrect description
- ```

FNOPA EE,XY(0,1),RA+ ; floating point multiplication
ADA PA,EE,XY(0,1),RA ; addition of product
                      (fixed point addition)

```

## 5.2 INSTRUCTION SET

The HSP instruction set is divided into six classifications:

- (I) ALU operation instructions
- (II) Immediate instructions
- (III) Jump instructions
- (IV) Register data transfer instructions
- (V) Register increment/decrement instructions
- (VI) Return instructions

This section contains descriptions of individual instructions. Assembler syntax expression and description format are explained as follows.  $\Delta$  shows a space.

- Assembler source statements
  - I. ALU operation instructions

Example:
$\underline{\text{LABEL}\Delta\text{FADA}\Delta\text{YA},\text{A},\text{XY}(1,3),\text{RA},\text{RO}+\Delta;\text{ACCA}\langle--\text{M}(\text{Y})+\text{ACCA}$
$\begin{array}{ccccccc} \uparrow & \uparrow & & \uparrow & & & \uparrow \\ \langle\text{label}\rangle & \langle\text{mnemonic}\rangle & \langle\text{operand}\rangle & & & & \langle\text{comment}\rangle \end{array}$

These instructions execute ALU arithmetic operations.

- <label> Label field in assembler expression
- <mnemonic> Operation field indicating ALU operation
- <operand> Specifies ALU input, addressing/reading/writing of data memory, autoincrement of the RAM/ROM pointers, and auto-decrement of the repeat counter. Operand expressions of each instruction are the same with one exception: a part of operand expressions depending on addressing mode. Common expression is described in <Operand (A) - (D)> at the end of this section.
- <comment> Optional field for comment

ALU operation instructions <mnemonics>

There is a difference in operand expression between I and I'.

I FADA, FADB, ADA, ADB, FSBA, FSBB, SBA, SBB, FLDA, FLDB, LDA, LDB, ANDA, ANDB, ORA, ORB, EORA, EORB

I' FABSA, FABSBB, ABSA, ABSB, FRPTA, FRPTB, RPTA, RPTB, FNEGA, FNEGB, NEGA, NEGB, INCA, INCB, DECA, DECB, SRA, SRB, SLA, SLB, FLTA, FLTB, FIXA, FIXB, FCLRA, FCLRB, CLRA, CLRB, FNOPA, FNOPB, NOPA, NOPB, FSGYA, FSGYB, SGYA, SGYB

## II. Immediate instructions

Example :
<u>LABL</u> Δ <u>LIA</u> Δ <u>\$3C5F</u> Δ <u>;ACCA&lt;-0.47165</u>
↑ ↑ ↑ ↑
<label><mnemonic><constant> <comment>

These instructions immediately set data to registers.

- <label> label field in assembler expression
- <mnemonic> operation field indicating immediate instruction
- <constant> value transferred to a register
- <comment> optional field for comment

Immediate instructions <mnemonics>

LIA, LIB, LIRA, LIRB, LIRO, LIRC

### III. Jump instructions

Example :
<u>LABL</u> Δ <u>JCS</u> Δ <u>LAB</u> Δ ;JUMP IF C=1
↑            ↑            ↑            ↑
<label><mnemonic><constant> <comment>

These instructions perform three kinds of jump operations: unconditional jump, conditional jump, and subroutine jump.

<label>        label field in assembler expression  
<mnemonic>    operation field indicating jump  
<constant>    specifies jump address  
<comment>     optional field for comment

Jump instructions <mnemonics>

JCS, JNS, JZS, JSR, JNZ, JNZM, JMP

### IV. Register data transfer instructions

Example :
<u>LABL</u> Δ <u>TFR</u> Δ <u>A</u> , <u>STR</u> Δ ;ACCA -> STR
↑            ↑            ↑            ↑            ↑
<label><mnemonic><register1><register2> <comment>

These instructions transfer data between registers.

<label>        label field in assembler expression  
<mnemonic>    TFR  
<register1>    specifies the source register  
<register2>    specifies the destination register  
<comment>     optional field for comment

Register data transfer instructions <mnemonics>

TFR

There are 38 transfer instructions (① to ③⑧) resulting from combination of the source register and the destination register.



- Description format

Each instruction is described in the following format.

Assembler Syntax	①
Example	②
Operation	③
Instruction Code	④
CCR OVFP	⑤

(Note) In this format, shows a space and [ ] can be omitted.

An ALU operation instruction has two addressing modes: pointer addressing mode and direct addressing mode. Therefore, two descriptions are given for an instruction.

- ① assembler syntax:  
Assembly language syntax format
- ② example:  
An example of full expression for an instruction.
- ③ operation:  
A description of operation. Common expression of operands is explained in <Operand ① - ④> at the end of this section.
- ④ instruction code:  
Machine code of each instruction consisting of 22 bits.  
For details of instruction code, see APPENDIX 2 'INSTRUCTION CODE'.
- ⑤ CCR, OVFP  
Shows the status of the CCR and the OVFP bit in the CTR. The contents of the CCR changes depending on instruction execution. The value of the OVFP bit is shown only when an instruction execution affects the OVFP bit. This bit must be set to 1 beforehand if necessary. If set to 0, the instruction may not be executed correctly.



Register data transfer instructions (TFR) are described in the following format.

Transfer No.

Expression	①
Operation	②
Instruction Code	③
CCR	④

① expression:

Assembler source statements of the TFR instructions with omitting label and comment fields.

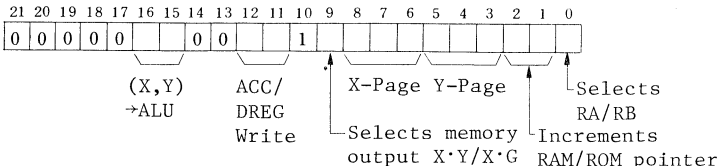
②, ③, ④ :

See ③, ④ and ⑤ in the former page.

The page of each instruction is listed in APPENDIX 3, 'HSP INSTRUCTION SUMMARY'.

# FADA

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔFADAΔ <operand ①>[Δ <comment>]										
Example	FADAΔYA, A, XY (1, 3), RA, RO + ① ②           ③           ④										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point add</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="333 581 1039 824"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (floating point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(20 bits)+ACCA(20 bits)→ACCA(20 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits)+ACCA(20 bits)→ACCA(20 bits)</td> </tr> <tr> <td>PX</td> <td>P(20 bits)+X(16 bits) →ACCA(20 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits)+X(16 bits) →ACCA(20 bits)</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <ul style="list-style-type: none"> <li>● Operand ②~④ For details of operand ②~④, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCA.</li> </ul>	Operand ①	ALU operation (floating point)	PA	P(20 bits)+ACCA(20 bits)→ACCA(20 bits)	YA	Y(16 bits)+ACCA(20 bits)→ACCA(20 bits)	PX	P(20 bits)+X(16 bits) →ACCA(20 bits)	YX	Y(16 bits)+X(16 bits) →ACCA(20 bits)
Operand ①	ALU operation (floating point)										
PA	P(20 bits)+ACCA(20 bits)→ACCA(20 bits)										
YA	Y(16 bits)+ACCA(20 bits)→ACCA(20 bits)										
PX	P(20 bits)+X(16 bits) →ACCA(20 bits)										
YX	Y(16 bits)+X(16 bits) →ACCA(20 bits)										
Instruction code											
CCR OVFP	<p>CCR C: Set if a carry is generated in fixed point add operation of digit-adjusted two mantissas. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is 0 after instruction execution.</p> <p>OVFP OVFP bit of the CTR must be set to 1 beforehand.</p>										

II. Direct addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔFADAΔ &lt;operand <b>ⓑ</b>&gt;[Δ &lt;comment&gt;]</p>																																													
<p>Example</p>	<p>FADA△PA, A, 0, 12                    ① ② ③</p>																																													
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation Floating point add</li> <li>● Operand <b>①</b> Operand <b>①</b> indicates input data of the ALU. The content of operand <b>①</b> is shown in the following table.</li> </ul> <table border="1" data-bbox="347 591 1057 831"> <thead> <tr> <th>Operand <b>①</b></th> <th>ALU operation (floating point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(20 bits)+ACCA(20 bits)→ACCA(20 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits)+ACCA(20 bits)→ACCA(20 bits)</td> </tr> <tr> <td>PX</td> <td>P(20 bits)+X(16 bits) →ACCA(20 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits)+X(16 bits) →ACCA(20 bits)</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle          X: X-Bus output    Y: Y-Bus output</p> <p>The contents of ACCA appear also on X-Bus. So the operations of PA and YA are the same as those of PX and YX respectively; however, there is a difference in the number of effective bits.</p> <ul style="list-style-type: none"> <li>● Operand <b>②~③</b> For details of operand <b>②~③</b>, see 5.2.1 'Operand <b>ⓑ</b>'. In 'Operand <b>ⓑ</b>', "ACC" means ACCA.</li> </ul>	Operand <b>①</b>	ALU operation (floating point)	PA	P(20 bits)+ACCA(20 bits)→ACCA(20 bits)	YA	Y(16 bits)+ACCA(20 bits)→ACCA(20 bits)	PX	P(20 bits)+X(16 bits) →ACCA(20 bits)	YX	Y(16 bits)+X(16 bits) →ACCA(20 bits)																																			
Operand <b>①</b>	ALU operation (floating point)																																													
PA	P(20 bits)+ACCA(20 bits)→ACCA(20 bits)																																													
YA	Y(16 bits)+ACCA(20 bits)→ACCA(20 bits)																																													
PX	P(20 bits)+X(16 bits) →ACCA(20 bits)																																													
YX	Y(16 bits)+X(16 bits) →ACCA(20 bits)																																													
<p>Instruction code</p>	<table border="1" data-bbox="347 1222 933 1282"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <p>(X,Y)      ACC/      (Page) (Pointer)          →ALU      DREG                      Write      direct address</p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	0	0	0	0			0	0			0	0									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																									
0	0	0	0	0	0			0	0			0	0																																	
<p>CCR OVFP</p>	<p>CCR C: Set if a carry is generated in fixed point add operation of digit-adjusted two mantissas.          N: Set if ACCA is negative after instruction execution.          Z: Set if ACCA is 0 after instruction execution.</p> <p>OVFP OVFP bit of the CTR must be set to 1 beforehand.</p>																																													

# FADB

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔFADBΔ <operand ①>[Δ <comment>]										
Example	FADBΔ <u>PA</u> , <u>D,XY</u> (2, 3), RA+ ①  ②      ③      ④										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point add</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="333 581 1039 824"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (floating point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(20 bits)+ACCB(20 bits)→ACCB(20 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits)+ACCB(20 bits)→ACCB(20 bits)</td> </tr> <tr> <td>PX</td> <td>P(20 bits)+X(16 bits) →ACCB(20 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits)+X(16 bits) →ACCB(20 bits)</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <ul style="list-style-type: none"> <li>● Operand ② ~ ④ For details of operand ② ~ ④, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCB.</li> </ul>	Operand ①	ALU operation (floating point)	PA	P(20 bits)+ACCB(20 bits)→ACCB(20 bits)	YA	Y(16 bits)+ACCB(20 bits)→ACCB(20 bits)	PX	P(20 bits)+X(16 bits) →ACCB(20 bits)	YX	Y(16 bits)+X(16 bits) →ACCB(20 bits)
Operand ①	ALU operation (floating point)										
PA	P(20 bits)+ACCB(20 bits)→ACCB(20 bits)										
YA	Y(16 bits)+ACCB(20 bits)→ACCB(20 bits)										
PX	P(20 bits)+X(16 bits) →ACCB(20 bits)										
YX	Y(16 bits)+X(16 bits) →ACCB(20 bits)										
Instruction code											
CCR OVFP	<p>CCR C: Set if a carry is generated in fixed point add operation of digit-adjusted two mantissas. N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is 0 after instruction execution.</p> <p>OVFP OVFP bit of the CTR must be set to 1 beforehand.</p>										

## II. Direct addressing mode

Assembler syntax	[<label>] ΔFADBA <operand <b>ⓑ</b> >[Δ <comment>]																																													
Example	FADB△ PA, D, 2, 49 ① ② ③																																													
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point add</li> <li>● Operand <b>①</b> Operand <b>①</b> indicates input data of the ALU. The content of operand <b>①</b> is shown in the following table.</li> </ul> <table border="1" data-bbox="337 597 1048 835"> <thead> <tr> <th>Operand <b>①</b></th> <th>ALU operation (floating point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(20 bits)+ACCB(20 bits)→ACCB(20 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits)+ACCB(20 bits)→ACCB(20 bits)</td> </tr> <tr> <td>PX</td> <td>P(20 bits)+X(16 bits) →ACCB(20 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits)+X(16 bits) →ACCB(20 bits)</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <p>The contents of ACCB appear also on X-Bus. So the operations of PA and YA are the same as those of PX and YX respectively; however, there is a difference in the number of effective bits.</p> <ul style="list-style-type: none"> <li>● Operand <b>② ~ ③</b> For details of operand <b>② ~ ③</b>, see 5.2.1 'Operand <b>ⓑ</b>'. In 'Operand <b>ⓑ</b>', "ACC" means ACCB.</li> </ul>	Operand <b>①</b>	ALU operation (floating point)	PA	P(20 bits)+ACCB(20 bits)→ACCB(20 bits)	YA	Y(16 bits)+ACCB(20 bits)→ACCB(20 bits)	PX	P(20 bits)+X(16 bits) →ACCB(20 bits)	YX	Y(16 bits)+X(16 bits) →ACCB(20 bits)																																			
Operand <b>①</b>	ALU operation (floating point)																																													
PA	P(20 bits)+ACCB(20 bits)→ACCB(20 bits)																																													
YA	Y(16 bits)+ACCB(20 bits)→ACCB(20 bits)																																													
PX	P(20 bits)+X(16 bits) →ACCB(20 bits)																																													
YX	Y(16 bits)+X(16 bits) →ACCB(20 bits)																																													
Instruction code	<table border="1" data-bbox="320 1236 908 1295"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td><td></td><td>0</td><td>1</td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <p>(X,Y)            ACC/ →ALU            DREG                   Write</p> <p>(Page) (Pointer)           direct address</p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	0	0	0	0			0	1			0	0									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																									
0	0	0	0	0	0			0	1			0	0																																	
CCR  OVFP	<p>CCR C: Set if a carry is generated in fixed point operation of digit-adjusted two mantissas. N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is 0 after instruction execution.</p> <p>OVFP OVFP bit of the CTR must be set to 1 beforehand.</p>																																													

# ADA

## I. Pointer addressing mode

Assembler syntax	[<label>] $\Delta$ ADA $\Delta$ <operand $\textcircled{A}$ >[ $\Delta$ <comment>]										
Example	$\text{ADA} \Delta \underset{\textcircled{1}}{\text{YA}}, \underset{\textcircled{2}}{\text{EE}}, \underset{\textcircled{3}}{\text{XG}} (\underset{\textcircled{4}}{0, 2}), \underset{\textcircled{4}}{\text{RA}}$										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point add</li> <li>● Operand <math>\textcircled{1}</math> Operand <math>\textcircled{1}</math> indicates input data of the ALU. The content of operand <math>\textcircled{1}</math> is shown in the following table.</li> </ul> <table border="1" data-bbox="337 576 1045 824"> <thead> <tr> <th>Operand <math>\textcircled{1}</math></th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td><math>P(16 \text{ bits}) + \text{ACCA}(16 \text{ bits}) \rightarrow \text{ACCA}(16 \text{ bits})</math></td> </tr> <tr> <td>YA</td> <td><math>Y(16 \text{ bits}) + \text{ACCA}(16 \text{ bits}) \rightarrow \text{ACCA}(16 \text{ bits})</math></td> </tr> <tr> <td>PX</td> <td><math>P(16 \text{ bits}) + X(16 \text{ bits}) \rightarrow \text{ACCA}(16 \text{ bits})</math></td> </tr> <tr> <td>YX</td> <td><math>Y(16 \text{ bits}) + X(16 \text{ bits}) \rightarrow \text{ACCA}(16 \text{ bits})</math></td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <ul style="list-style-type: none"> <li>● Operand <math>\textcircled{2} \sim \textcircled{4}</math> For details of Operand <math>\textcircled{2} \sim \textcircled{4}</math>, see 5.2.1 'Operand <math>\textcircled{A}</math>'. In 'Operand <math>\textcircled{A}</math>', "ACC" means ACCA.</li> </ul>	Operand $\textcircled{1}$	ALU operation (fixed point)	PA	$P(16 \text{ bits}) + \text{ACCA}(16 \text{ bits}) \rightarrow \text{ACCA}(16 \text{ bits})$	YA	$Y(16 \text{ bits}) + \text{ACCA}(16 \text{ bits}) \rightarrow \text{ACCA}(16 \text{ bits})$	PX	$P(16 \text{ bits}) + X(16 \text{ bits}) \rightarrow \text{ACCA}(16 \text{ bits})$	YX	$Y(16 \text{ bits}) + X(16 \text{ bits}) \rightarrow \text{ACCA}(16 \text{ bits})$
Operand $\textcircled{1}$	ALU operation (fixed point)										
PA	$P(16 \text{ bits}) + \text{ACCA}(16 \text{ bits}) \rightarrow \text{ACCA}(16 \text{ bits})$										
YA	$Y(16 \text{ bits}) + \text{ACCA}(16 \text{ bits}) \rightarrow \text{ACCA}(16 \text{ bits})$										
PX	$P(16 \text{ bits}) + X(16 \text{ bits}) \rightarrow \text{ACCA}(16 \text{ bits})$										
YX	$Y(16 \text{ bits}) + X(16 \text{ bits}) \rightarrow \text{ACCA}(16 \text{ bits})$										
Instruction code											
CCR OVFP	<p>CCR C: Set if a carry is generated after arithmetic operation. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is 0 after instruction execution.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit. 0: no overflow protection 1: overflow protection Overflow protection does not affect the status of carry flag.</p>										

## II. Direct addressing mode

Assembler syntax	[<label>] $\Delta$ ADA $\Delta$ <operand $\textcircled{B}$ >[ $\Delta$ <comment>]																																																																
Example	ADA $\Delta$ <u>YA</u> , <u>EE</u> , <u>4</u> , <u>24</u> $\textcircled{1}$ $\textcircled{2}$ $\textcircled{3}$																																																																
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point add</li> <li>● Operand <math>\textcircled{1}</math> Operand <math>\textcircled{1}</math> indicates input data of the ALU. The content of operand <math>\textcircled{1}</math> is shown in the following table.</li> </ul> <table border="1" style="margin: 10px auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th style="padding: 5px;">Operand <math>\textcircled{1}</math></th> <th style="padding: 5px;">ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">PA</td> <td style="padding: 5px;">P(16 bits)+ACCA(16 bits)<math>\rightarrow</math>ACCA(16 bits)</td> </tr> <tr> <td style="padding: 5px;">YA</td> <td style="padding: 5px;">Y(16 bits)+ACCA(16 bits)<math>\rightarrow</math>ACCA(16 bits)</td> </tr> <tr> <td style="padding: 5px;">PX</td> <td style="padding: 5px;">P(16 bits)+X(16 bits) <math>\rightarrow</math>ACCA(16 bits)</td> </tr> <tr> <td style="padding: 5px;">YX</td> <td style="padding: 5px;">Y(16 bits)+X(16 bits) <math>\rightarrow</math>ACCA(16 bits)</td> </tr> </tbody> </table> <p style="margin-left: 40px;">P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <p style="margin-left: 40px;">The contents of ACCA appear also on X-Bus. So the operations of PA and YA are the same as those of PX and YX respectively.</p> <ul style="list-style-type: none"> <li>● Operand <math>\textcircled{2} \sim \textcircled{3}</math> For details of operand <math>\textcircled{2} \sim \textcircled{3}</math>, see 5.2.1 'Operand <math>\textcircled{B}</math>'. In 'Operand <math>\textcircled{B}</math>', "ACC" means ACCA.</li> </ul>	Operand $\textcircled{1}$	ALU operation (fixed point)	PA	P(16 bits)+ACCA(16 bits) $\rightarrow$ ACCA(16 bits)	YA	Y(16 bits)+ACCA(16 bits) $\rightarrow$ ACCA(16 bits)	PX	P(16 bits)+X(16 bits) $\rightarrow$ ACCA(16 bits)	YX	Y(16 bits)+X(16 bits) $\rightarrow$ ACCA(16 bits)																																																						
Operand $\textcircled{1}$	ALU operation (fixed point)																																																																
PA	P(16 bits)+ACCA(16 bits) $\rightarrow$ ACCA(16 bits)																																																																
YA	Y(16 bits)+ACCA(16 bits) $\rightarrow$ ACCA(16 bits)																																																																
PX	P(16 bits)+X(16 bits) $\rightarrow$ ACCA(16 bits)																																																																
YX	Y(16 bits)+X(16 bits) $\rightarrow$ ACCA(16 bits)																																																																
Instruction code	<table style="margin: 0 auto; border-collapse: collapse; text-align: center;"> <tr> <td style="border: 1px solid black; padding: 2px;">21</td><td style="border: 1px solid black; padding: 2px;">20</td><td style="border: 1px solid black; padding: 2px;">19</td><td style="border: 1px solid black; padding: 2px;">18</td><td style="border: 1px solid black; padding: 2px;">17</td><td style="border: 1px solid black; padding: 2px;">16</td><td style="border: 1px solid black; padding: 2px;">15</td><td style="border: 1px solid black; padding: 2px;">14</td><td style="border: 1px solid black; padding: 2px;">13</td><td style="border: 1px solid black; padding: 2px;">12</td><td style="border: 1px solid black; padding: 2px;">11</td><td style="border: 1px solid black; padding: 2px;">10</td><td style="border: 1px solid black; padding: 2px;">9</td><td style="border: 1px solid black; padding: 2px;">8</td><td style="border: 1px solid black; padding: 2px;">7</td><td style="border: 1px solid black; padding: 2px;">6</td><td style="border: 1px solid black; padding: 2px;">5</td><td style="border: 1px solid black; padding: 2px;">4</td><td style="border: 1px solid black; padding: 2px;">3</td><td style="border: 1px solid black; padding: 2px;">2</td><td style="border: 1px solid black; padding: 2px;">1</td><td style="border: 1px solid black; padding: 2px;">0</td> </tr> <tr> <td style="border: 1px solid black; padding: 2px;">0</td><td style="border: 1px solid black; padding: 2px;">0</td><td style="border: 1px solid black; padding: 2px;">0</td><td style="border: 1px solid black; padding: 2px;">0</td><td style="border: 1px solid black; padding: 2px;">0</td><td style="border: 1px solid black; padding: 2px;">0</td><td style="border: 1px solid black; padding: 2px;">1</td><td style="border: 1px solid black; padding: 2px;">0</td><td style="border: 1px solid black; padding: 2px;"> </td><td style="border: 1px solid black; padding: 2px;"> </td><td style="border: 1px solid black; padding: 2px;"> </td><td style="border: 1px solid black; padding: 2px;">0</td><td style="border: 1px solid black; padding: 2px;">0</td><td style="border: 1px solid black; padding: 2px;"> </td><td style="border: 1px solid black; padding: 2px;"> </td><td style="border: 1px solid black; padding: 2px;"> </td><td style="border: 1px solid black; padding: 2px;"> </td><td style="border: 1px solid black; padding: 2px;"> </td><td style="border: 1px solid black; padding: 2px;"> </td><td style="border: 1px solid black; padding: 2px;"> </td><td style="border: 1px solid black; padding: 2px;"> </td><td style="border: 1px solid black; padding: 2px;"> </td> </tr> <tr> <td colspan="4" style="border: none; padding: 5px;">(X,Y)</td> <td colspan="4" style="border: none; padding: 5px;">ACC/ <math>\rightarrow</math>ALU</td> <td colspan="4" style="border: none; padding: 5px;">ACC/ DREG</td> <td colspan="4" style="border: none; padding: 5px;">(Page) (Pointer) direct address</td> <td colspan="4" style="border: none; padding: 5px;">Write</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	0	0	0	0	1	0				0	0										(X,Y)				ACC/ $\rightarrow$ ALU				ACC/ DREG				(Page) (Pointer) direct address				Write			
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																												
0	0	0	0	0	0	1	0				0	0																																																					
(X,Y)				ACC/ $\rightarrow$ ALU				ACC/ DREG				(Page) (Pointer) direct address				Write																																																	
CCR  OVFP	<p>CCR C: Set if a carry is generated after arithmetic operation. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is 0 after instruction execution.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit. 0: no overflow protection    1: overflow protection Overflow protection does not affect the status of carry flag.</p>																																																																

# ADB

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔADBΔ <operand $\textcircled{A}$ >[Δ <comment>]										
Example	ADB Δ <u>YA</u> , <u>EE</u> , <u>XG</u> (1, 3), <u>RA</u> ①   ②       ③       ④										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point add</li> <li>● Operand <math>\textcircled{1}</math> Operand <math>\textcircled{1}</math> indicates input data of the ALU. The content of operand <math>\textcircled{1}</math> is shown in the following table.</li> </ul> <table border="1" data-bbox="327 581 1039 824"> <thead> <tr> <th>Operand <math>\textcircled{1}</math></th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(16 bits)+ACCB(16 bits)→ACCB(16 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits)+ACCB(16 bits)→ACCB(16 bits)</td> </tr> <tr> <td>PX</td> <td>P(16 bits)+X(16 bits) →ACCB(16 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits)+X(16 bits) →ACCB(16 bits)</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <ul style="list-style-type: none"> <li>● Operand <math>\textcircled{2} \sim \textcircled{4}</math> For details of Operand <math>\textcircled{2} \sim \textcircled{4}</math>, see 5.2.1 'Operand <math>\textcircled{A}</math>'. In 'Operand <math>\textcircled{A}</math>', "ACC" means ACCB.</li> </ul>	Operand $\textcircled{1}$	ALU operation (fixed point)	PA	P(16 bits)+ACCB(16 bits)→ACCB(16 bits)	YA	Y(16 bits)+ACCB(16 bits)→ACCB(16 bits)	PX	P(16 bits)+X(16 bits) →ACCB(16 bits)	YX	Y(16 bits)+X(16 bits) →ACCB(16 bits)
Operand $\textcircled{1}$	ALU operation (fixed point)										
PA	P(16 bits)+ACCB(16 bits)→ACCB(16 bits)										
YA	Y(16 bits)+ACCB(16 bits)→ACCB(16 bits)										
PX	P(16 bits)+X(16 bits) →ACCB(16 bits)										
YX	Y(16 bits)+X(16 bits) →ACCB(16 bits)										
Instruction code											
CCR OVFP	<p>CCR C: Set if a carry is generated after arithmetic operation. N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is 0 after instruction execution.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit. 0: no overflow protection    1: overflow protection Overflow protection does not affect the status of carry flag.</p>										





# FSBA

## I. Pointer addressing mode

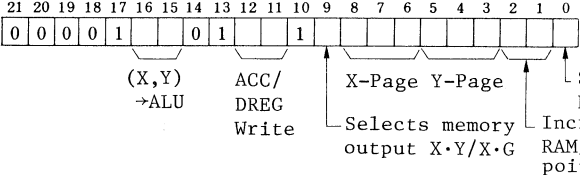
Assembler syntax	[<label>] ΔFSBAΔ <operand (A)>[Δ <comment>]										
Example	FSBAΔ <u>PX</u> , <u>EE</u> , <u>XY</u> (3, 4), <u>RA</u> ①    ②       ③       ④										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point subtract</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="335 574 1043 822"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (floating point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(20 bits)-ACCA(20 bits)→ACCA(20 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits)-ACCA(20 bits)→ACCA(20 bits)</td> </tr> <tr> <td>PX</td> <td>P(20 bits)-X(16 bits) →ACCA(20 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits)-X(16 bits) →ACCA(20 bits)</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <ul style="list-style-type: none"> <li>● Operand ②~④ For details of operand ②~④, see 5.2.1 'Operand (A)'. In 'Operand (A)', "ACC" means ACCA.</li> </ul>	Operand ①	ALU operation (floating point)	PA	P(20 bits)-ACCA(20 bits)→ACCA(20 bits)	YA	Y(16 bits)-ACCA(20 bits)→ACCA(20 bits)	PX	P(20 bits)-X(16 bits) →ACCA(20 bits)	YX	Y(16 bits)-X(16 bits) →ACCA(20 bits)
Operand ①	ALU operation (floating point)										
PA	P(20 bits)-ACCA(20 bits)→ACCA(20 bits)										
YA	Y(16 bits)-ACCA(20 bits)→ACCA(20 bits)										
PX	P(20 bits)-X(16 bits) →ACCA(20 bits)										
YX	Y(16 bits)-X(16 bits) →ACCA(20 bits)										
Instruction code											
CCR OVFP	<p>CCR C: Set if a borrow is generated in fixed point subtract operation of digit-adjusted two mantissas. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is 0 after instruction execution.</p> <p>OVFP OVFP bit of the CTR must be set to 1 beforehand.</p>										

## II. Direct addressing mode

<b>Assembler syntax</b>	[<label>] ΔFSBAΔ <operand <b>Ⓐ</b> >[Δ <comment>]																																												
<b>Example</b>	FSBAΔ <u>YA</u> , <u>EE</u> , <u>2, 39</u> ①   ②   ③																																												
<b>Operation</b>	<ul style="list-style-type: none"> <li>● ALU operation Floating point subtract</li> <li>● Operand <b>Ⓐ</b> Operand <b>Ⓐ</b> indicates input data of the ALU. The content of operand <b>Ⓐ</b> is shown in the following table.</li> </ul> <table border="1" data-bbox="333 590 1043 829"> <thead> <tr> <th>Operand <b>Ⓐ</b></th> <th>ALU operation (floating point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(20 bits)-ACCA(20 bits)→ACCA(20 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits)-ACCA(20 bits)→ACCA(20 bits)</td> </tr> <tr> <td>PX</td> <td>P(20 bits)-X(16 bits) →ACCA(20 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits)-X(16 bits) →ACCA(20 bits)</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <p>The contents of ACCA appear also on X-Bus. So the operations of PA and YA are the same as those of PX and YX respectively; however, there is a difference in the number of effective bits.</p> <ul style="list-style-type: none"> <li>● Operand <b>Ⓑ</b>~<b>Ⓒ</b> For details of operand <b>Ⓑ</b>~<b>Ⓒ</b>, see 5.2.1 'Operand <b>Ⓐ</b>'. In 'Operand <b>Ⓐ</b>', "ACC" means ACCA.</li> </ul>	Operand <b>Ⓐ</b>	ALU operation (floating point)	PA	P(20 bits)-ACCA(20 bits)→ACCA(20 bits)	YA	Y(16 bits)-ACCA(20 bits)→ACCA(20 bits)	PX	P(20 bits)-X(16 bits) →ACCA(20 bits)	YX	Y(16 bits)-X(16 bits) →ACCA(20 bits)																																		
Operand <b>Ⓐ</b>	ALU operation (floating point)																																												
PA	P(20 bits)-ACCA(20 bits)→ACCA(20 bits)																																												
YA	Y(16 bits)-ACCA(20 bits)→ACCA(20 bits)																																												
PX	P(20 bits)-X(16 bits) →ACCA(20 bits)																																												
YX	Y(16 bits)-X(16 bits) →ACCA(20 bits)																																												
<b>Instruction code</b>	<table border="1" data-bbox="312 1215 897 1275"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td></td><td>0</td><td>0</td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <p>(X,Y)            ACC/            (Page) (Pointer) →ALU            DREG            direct address Write</p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	0	0	1		0	0		0	0											
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
0	0	0	0	1		0	0		0	0																																			
<b>CCR</b>  <b>OVFP</b>	<p>CCR C: Set if a borrow is generated in fixed point subtract operation of digit-adjusted two mantissas. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is 0 after instruction execution.</p> <p>OVFP OVFP bit of the CTR must be set to 1 beforehand.</p>																																												

# FSBB

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔFSBBΔ <operand ①>[Δ <comment>]										
Example	FSBBΔ <u>PX</u> , <u>EE</u> , <u>XY(3,0)</u> , <u>RA</u> ①    ②        ③        ④										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point subtract</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="341 574 1049 817"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (floating point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(20 bits)-ACCB(20 bits)→ACCB(20 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits)-ACCB(20 bits)→ACCB(20 bits)</td> </tr> <tr> <td>PX</td> <td>P(20 bits)-X(16 bits) →ACCB(20 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits)-X(16 bits) →ACCB(20 bits)</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <ul style="list-style-type: none"> <li>● Operand ②~④ For details of operand ②~④, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCB.</li> </ul>	Operand ①	ALU operation (floating point)	PA	P(20 bits)-ACCB(20 bits)→ACCB(20 bits)	YA	Y(16 bits)-ACCB(20 bits)→ACCB(20 bits)	PX	P(20 bits)-X(16 bits) →ACCB(20 bits)	YX	Y(16 bits)-X(16 bits) →ACCB(20 bits)
Operand ①	ALU operation (floating point)										
PA	P(20 bits)-ACCB(20 bits)→ACCB(20 bits)										
YA	Y(16 bits)-ACCB(20 bits)→ACCB(20 bits)										
PX	P(20 bits)-X(16 bits) →ACCB(20 bits)										
YX	Y(16 bits)-X(16 bits) →ACCB(20 bits)										
Instruction code											
CCR OVFP	<p>CCR C: Set if a borrow is generated in fixed point subtract operation of digit-adjusted two mantissas. N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is 0 after instruction execution.</p> <p>OVFP OVFP bit of the CTR must be set to 1 beforehand.</p>										



# SBA

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔSBAΔ <operand ①>[Δ <comment>]										
Example	SBAΔ <u>YA</u> , <u>EE</u> , <u>XY(0,4)</u> , <u>RA</u> , <u>RO</u> ①   ②       ③       ④										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point subtract</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="341 574 1047 817"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(16 bits)-ACCA(16 bits)→ACCA(16 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits)-ACCA(16 bits)→ACCA(16 bits)</td> </tr> <tr> <td>PX</td> <td>P(16 bits)-X(16 bits) →ACCA(16 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits)-X(16 bits) →ACCA(16 bits)</td> </tr> </tbody> </table> <p style="margin-left: 40px;">P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <ul style="list-style-type: none"> <li>● Operand ②~④ For details of operand ②~④, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCA.</li> </ul>	Operand ①	ALU operation (fixed point)	PA	P(16 bits)-ACCA(16 bits)→ACCA(16 bits)	YA	Y(16 bits)-ACCA(16 bits)→ACCA(16 bits)	PX	P(16 bits)-X(16 bits) →ACCA(16 bits)	YX	Y(16 bits)-X(16 bits) →ACCA(16 bits)
Operand ①	ALU operation (fixed point)										
PA	P(16 bits)-ACCA(16 bits)→ACCA(16 bits)										
YA	Y(16 bits)-ACCA(16 bits)→ACCA(16 bits)										
PX	P(16 bits)-X(16 bits) →ACCA(16 bits)										
YX	Y(16 bits)-X(16 bits) →ACCA(16 bits)										
Instruction code											
CCR OVFP	<p>CCR C: Set if a borrow is generated after arithmetic operation. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is 0 after instruction execution.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit. 0: no overflow protection    1: overflow protection Overflow protection does not affect the status of carry flag.</p>										



# SBB

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔSBBΔ <operand (A)>[Δ <comment>]										
Example	SBBΔ <u>YA</u> , <u>EE</u> , <u>XG(4, 0)</u> , <u>RA</u> ①  ②          ③          ④										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point subtract</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="337 579 1043 822"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(16 bits)-ACCB(16 bits)→ACCB(16 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits)-ACCB(16 bits)→ACCB(16 bits)</td> </tr> <tr> <td>PX</td> <td>P(16 bits)-X(16 bits) →ACCB(16 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits)-X(16 bits) →ACCB(16 bits)</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <ul style="list-style-type: none"> <li>● Operand ②~④ For details of operand ②~④, see 5.2.1 "Operand (A)". In 'Operand (A)', "ACC" means ACCB.</li> </ul>	Operand ①	ALU operation (fixed point)	PA	P(16 bits)-ACCB(16 bits)→ACCB(16 bits)	YA	Y(16 bits)-ACCB(16 bits)→ACCB(16 bits)	PX	P(16 bits)-X(16 bits) →ACCB(16 bits)	YX	Y(16 bits)-X(16 bits) →ACCB(16 bits)
Operand ①	ALU operation (fixed point)										
PA	P(16 bits)-ACCB(16 bits)→ACCB(16 bits)										
YA	Y(16 bits)-ACCB(16 bits)→ACCB(16 bits)										
PX	P(16 bits)-X(16 bits) →ACCB(16 bits)										
YX	Y(16 bits)-X(16 bits) →ACCB(16 bits)										
Instruction code											
CCR OVFP	<p>CCR C: Set if a borrow is generated after arithmetic operation. N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is 0 after instruction execution.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit. 0: no overflow protection    1: overflow protection Overflow protection does not affect the status of carry flag.</p>										

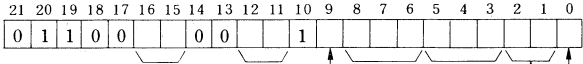


## II. Direct addressing mode

Assembler syntax	[<label>] ΔSBBΔ <operand <sup>ⓑ</sup> >[Δ <comment>]																																												
Example	SBBΔ <u>YA</u> , <u>EE</u> , <u>6</u> , <u>22</u> <sup>ⓑ</sup> <sup>ⓐ</sup> <sup>ⓓ</sup>																																												
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point subtract</li> <li>● Operand <sup>ⓑ</sup> Operand <sup>ⓑ</sup> indicates input data of the ALU. The content of operand <sup>ⓑ</sup> is shown in the following table.</li> </ul> <table border="1" data-bbox="337 591 1047 829"> <thead> <tr> <th>Operand <sup>ⓑ</sup></th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(16 bits)-ACCB(16 bits)→ACCB(16 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits)-ACCB(16 bits)→ACCB(16 bits)</td> </tr> <tr> <td>PX</td> <td>P(16 bits)-X(16 bits) →ACCB(16 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits)-X(16 bits) →ACCB(16 bits)</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <p>The contents of ACCB appear also on X-Bus. So the operations of PA and YA are the same as those of PX and YX respectively.</p> <ul style="list-style-type: none"> <li>● Operand <sup>ⓐ</sup>~<sup>ⓓ</sup> For details of operand <sup>ⓐ</sup>~<sup>ⓓ</sup>, see 5.2.1 'Operand <sup>ⓑ</sup>'. In 'Operand <sup>ⓑ</sup>', "ACC" means ACCB.</li> </ul>	Operand <sup>ⓑ</sup>	ALU operation (fixed point)	PA	P(16 bits)-ACCB(16 bits)→ACCB(16 bits)	YA	Y(16 bits)-ACCB(16 bits)→ACCB(16 bits)	PX	P(16 bits)-X(16 bits) →ACCB(16 bits)	YX	Y(16 bits)-X(16 bits) →ACCB(16 bits)																																		
Operand <sup>ⓑ</sup>	ALU operation (fixed point)																																												
PA	P(16 bits)-ACCB(16 bits)→ACCB(16 bits)																																												
YA	Y(16 bits)-ACCB(16 bits)→ACCB(16 bits)																																												
PX	P(16 bits)-X(16 bits) →ACCB(16 bits)																																												
YX	Y(16 bits)-X(16 bits) →ACCB(16 bits)																																												
Instruction code	<table border="1" data-bbox="322 1229 909 1286"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td></td><td></td><td></td><td>1</td><td>1</td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <p>(X,Y)                    ACC/ →ALU                    DREG                                  Write</p> <p>(Page) (Pointer) direct address</p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	0	0	1				1	1		0	0									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
0	0	0	0	1				1	1		0	0																																	
CCR  OVFP	<p>CCR C: Set if a borrow is generated after arithmetic operation. N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is 0 after instruction execution.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit. 0: no overflow protection    1: overflow protection Overflow protection does not affect the status of carry flag.</p>																																												

# FLDA

## I. Pointer addressing mode

Assembler syntax	[<label>] $\Delta$ FLDAA <operand $\textcircled{A}$ > [ $\Delta$ <comment>]										
Example	FLDA $\Delta$ <u>YA</u> , <u>EE</u> , <u>XY (4, 3)</u> , RB $\textcircled{1}$ $\textcircled{2}$ $\textcircled{3}$ $\textcircled{4}$										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point load</li> <li>● Operand <math>\textcircled{1}</math> Operand <math>\textcircled{1}</math> indicates input data of the ALU. The content of operand <math>\textcircled{1}</math> is shown in the following table.</li> </ul> <table border="1" data-bbox="337 579 1043 819"> <thead> <tr> <th>Operand <math>\textcircled{1}</math></th> <th>ALU operation (floating point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(20 bits) <math>\rightarrow</math> ACCA(20 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits) <math>\rightarrow</math> ACCA(20 bits)</td> </tr> <tr> <td>PX</td> <td>The same as PA</td> </tr> <tr> <td>YX</td> <td>The same as YA</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <p>The load data is normalized.</p> <ul style="list-style-type: none"> <li>● Operand <math>\textcircled{2} \sim \textcircled{4}</math> For details of operand <math>\textcircled{2} \sim \textcircled{4}</math>, see 5.2.1 'Operand <math>\textcircled{A}</math>'. In 'Operand <math>\textcircled{A}</math>', "ACC" means ACCA.</li> </ul>	Operand $\textcircled{1}$	ALU operation (floating point)	PA	P(20 bits) $\rightarrow$ ACCA(20 bits)	YA	Y(16 bits) $\rightarrow$ ACCA(20 bits)	PX	The same as PA	YX	The same as YA
Operand $\textcircled{1}$	ALU operation (floating point)										
PA	P(20 bits) $\rightarrow$ ACCA(20 bits)										
YA	Y(16 bits) $\rightarrow$ ACCA(20 bits)										
PX	The same as PA										
YX	The same as YA										
Instruction code											
CCR OVFP	CCR C: 0 N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is 0 after instruction execution.										



# FLDB

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔFLDBΔ <operand (A)>[Δ <comment>]										
Example	FLDBΔ <u>YA</u> , <u>EE</u> , <u>XY(4, 2)</u> , <u>RA+</u> ①    ②        ③          ④										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point load</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="335 579 1040 822"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (floating point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(20 bits) → ACCB(20 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits) → ACCB(20 bits)</td> </tr> <tr> <td>PX</td> <td>The same as PA</td> </tr> <tr> <td>YX</td> <td>The same as YA</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <p>The load data is normalized.</p> <ul style="list-style-type: none"> <li>● Operand ②~④ For details of operand ②~④, see 5.2.1 'Operand (A)'. In 'Operand (A)', "ACC" means ACCB.</li> </ul>	Operand ①	ALU operation (floating point)	PA	P(20 bits) → ACCB(20 bits)	YA	Y(16 bits) → ACCB(20 bits)	PX	The same as PA	YX	The same as YA
Operand ①	ALU operation (floating point)										
PA	P(20 bits) → ACCB(20 bits)										
YA	Y(16 bits) → ACCB(20 bits)										
PX	The same as PA										
YX	The same as YA										
Instruction code											
CCR OVFP	CCR C: 0 N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is 0 after instruction execution.										

<b>Assembler syntax</b>	[<label>] ΔFLDBΔ <operand ②>[Δ <comment>]																																												
<b>Example</b>	FLDB Δ YA, EE, 3.48 ①  ②  ③																																												
<b>Operation</b>	<ul style="list-style-type: none"> <li>● ALU operation Floating point load</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="345 591 1055 835"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (floating point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(20 bits) → ACCB(20 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits) → ACCB(20 bits)</td> </tr> <tr> <td>PX</td> <td>The same as PA</td> </tr> <tr> <td>YX</td> <td>The same as YA</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <ul style="list-style-type: none"> <li>● Operand ②~③ For details of operand ②~③, see 5.2.1 'Operand ②'. In 'Operand ②', "ACC" means ACCB.</li> </ul>	Operand ①	ALU operation (floating point)	PA	P(20 bits) → ACCB(20 bits)	YA	Y(16 bits) → ACCB(20 bits)	PX	The same as PA	YX	The same as YA																																		
Operand ①	ALU operation (floating point)																																												
PA	P(20 bits) → ACCB(20 bits)																																												
YA	Y(16 bits) → ACCB(20 bits)																																												
PX	The same as PA																																												
YX	The same as YA																																												
<b>Instruction code</b>	<table border="1" data-bbox="325 1220 912 1281"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td></td><td></td><td></td><td>0</td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <p>(X,Y)            ACC/ →ALU            DREG                   Write</p> <p>(Page) (Pointer) direct address</p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	1	0	0				0	1												
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
0	1	1	0	0				0	1																																				
<b>CCR</b>  <b>OVFP</b>	CCR C: 0 N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is 0 after instruction execution.																																												

# LDA

## I. Pointer addressing mode

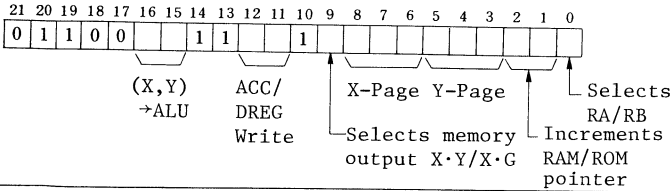
Assembler syntax	[<label>] ΔLDAΔ <operand (A)>[Δ <comment>]										
Example	LDAΔ <u>YA</u> , <u>EE</u> , <u>XY(0, 2)</u> , <u>RA</u> ①   ②           ③       ④										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point load</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="333 581 1039 824"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>PX</td> <td>The same as PA</td> </tr> <tr> <td>YX</td> <td>The same as YA</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <ul style="list-style-type: none"> <li>● Operand ②~④ For details of operand ②~④, see 5.2.1 'Operand (A)'. In 'Operand (A)', "ACC" means ACCA.</li> </ul>	Operand ①	ALU operation (fixed point)	PA	P(16 bits) → ACCA(16 bits)	YA	Y(16 bits) → ACCA(16 bits)	PX	The same as PA	YX	The same as YA
Operand ①	ALU operation (fixed point)										
PA	P(16 bits) → ACCA(16 bits)										
YA	Y(16 bits) → ACCA(16 bits)										
PX	The same as PA										
YX	The same as YA										
Instruction code											
CCR OVFP	CCR C: 0 N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is 0 after instruction execution.										

## II. Direct addressing mode

Assembler syntax	[<label>] ΔLDAΔ <operand <b>(B)</b> >[Δ <comment>]																																																	
Example	LDAΔYA,EE,5,26 ① ② ③																																																	
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point load</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="337 591 1043 835" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>PX</td> <td>The same as PA</td> </tr> <tr> <td>YX</td> <td>The same as YA</td> </tr> </tbody> </table> <p style="margin-left: 40px;">P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <ul style="list-style-type: none"> <li>● Operand ② ~ ③ For details of operand ② ~ ③, see 5.2.1 'Operand <b>(B)</b>'. In 'Operand <b>(B)</b>', "ACC" means ACCA.</li> </ul>	Operand ①	ALU operation (fixed point)	PA	P(16 bits) → ACCA(16 bits)	YA	Y(16 bits) → ACCA(16 bits)	PX	The same as PA	YX	The same as YA																																							
Operand ①	ALU operation (fixed point)																																																	
PA	P(16 bits) → ACCA(16 bits)																																																	
YA	Y(16 bits) → ACCA(16 bits)																																																	
PX	The same as PA																																																	
YX	The same as YA																																																	
Instruction code	<table border="1" data-bbox="316 1225 904 1286" style="margin-left: auto; margin-right: auto;"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td></td><td></td><td></td><td>1</td><td>0</td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <div style="margin-left: 100px; margin-top: 10px;"> <table style="border: none;"> <tr> <td style="text-align: center;">(X,Y) →ALU</td> <td style="text-align: center;">ACC/ DREG Write</td> <td style="text-align: center;">(Page) direct address</td> <td style="text-align: center;">(Pointer)</td> </tr> </table> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	1	0	0				1	0			0	0										(X,Y) →ALU	ACC/ DREG Write	(Page) direct address	(Pointer)
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																													
0	1	1	0	0				1	0			0	0																																					
(X,Y) →ALU	ACC/ DREG Write	(Page) direct address	(Pointer)																																															
CCR  OVFP	CCR C: 0 N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is 0 after instruction execution.																																																	

# LDB

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔLDBΔ <operand ①>[Δ <comment>]										
Example	LDBΔ <u>YA</u> , <u>EE</u> , <u>XG(0, 2)</u> , <u>RA</u> ①    ②        ③        ④										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point load</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="337 576 1043 822"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(16 bits) → ACCB(16 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits) → ACCB(16 bits)</td> </tr> <tr> <td>PX</td> <td>The same as PA</td> </tr> <tr> <td>YX</td> <td>The same as YA</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <ul style="list-style-type: none"> <li>● Operand ②~④ For details of operand ②~④, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCB.</li> </ul>	Operand ①	ALU operation (fixed point)	PA	P(16 bits) → ACCB(16 bits)	YA	Y(16 bits) → ACCB(16 bits)	PX	The same as PA	YX	The same as YA
Operand ①	ALU operation (fixed point)										
PA	P(16 bits) → ACCB(16 bits)										
YA	Y(16 bits) → ACCB(16 bits)										
PX	The same as PA										
YX	The same as YA										
Instruction code	 <p>The diagram shows a 22-bit instruction code field. Bits 21-18 are 0110. Bits 17-14 are blank. Bits 13-12 are 11. Bit 11 is blank. Bit 10 is 1. Bits 9-8 are blank. Bits 7-6 are blank. Bits 5-4 are blank. Bits 3-2 are blank. Bit 1 is blank. Bit 0 is blank. Labels below the bits indicate: bits 17-14 (X,Y) → ALU; bits 13-12 (ACC/DREG Write); bits 9-8 (X-Page); bits 7-6 (Y-Page); bits 5-4 (Selects memory output X·Y/X·G); bits 3-2 (Selects RA/RB); bit 1 (Increments RAM/ROM pointer).</p>										
CCR OVFP	CCR C: 0 N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is 0 after instruction execution.										



## II. Direct addressing mode

Assembler syntax	[<label>] ΔLDBΔ <operand ①>[Δ <comment>]																																															
Example	LDBΔ <u>YX</u> , <u>EE</u> , <u>2, 44</u> ①   ②   ③																																															
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point load</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="340 590 1052 833" style="margin-left: 40px;"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(16 bits) → ACCB(16 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits) → ACCB(16 bits)</td> </tr> <tr> <td>PX</td> <td>The same as PA</td> </tr> <tr> <td>YX</td> <td>The same as YA</td> </tr> </tbody> </table> <p style="margin-left: 40px;">P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output</p> <ul style="list-style-type: none"> <li>● Operand ②~③ For details of operand ②~③, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCB.</li> </ul>	Operand ①	ALU operation (fixed point)	PA	P(16 bits) → ACCB(16 bits)	YA	Y(16 bits) → ACCB(16 bits)	PX	The same as PA	YX	The same as YA																																					
Operand ①	ALU operation (fixed point)																																															
PA	P(16 bits) → ACCB(16 bits)																																															
YA	Y(16 bits) → ACCB(16 bits)																																															
PX	The same as PA																																															
YX	The same as YA																																															
Instruction code	<table border="1" data-bbox="321 1215 913 1275" style="margin-left: 40px; text-align: center;"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td></td><td></td><td></td><td>1</td><td>1</td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <div style="margin-left: 40px; margin-top: 10px;"> <table style="width: 100%; border: none;"> <tr> <td style="text-align: center; width: 33%;">(X,Y) →ALU</td> <td style="text-align: center; width: 33%;">ACC/ DREG Write</td> <td style="text-align: center; width: 33%;">(Page) (Pointer) direct address</td> </tr> </table> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	1	0	0				1	1			0	0									(X,Y) →ALU	ACC/ DREG Write	(Page) (Pointer) direct address
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																											
0	1	1	0	0				1	1			0	0																																			
(X,Y) →ALU	ACC/ DREG Write	(Page) (Pointer) direct address																																														
CCR  OVFP	CCR    C: 0 N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is 0 after instruction execution.																																															

# ANDA

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔANDAΔ <operand (A)>[Δ <comment>]										
Example	ANDAΔ <u>YX</u> , <u>EE</u> , <u>XY</u> (1, 8), <u>RA</u> ①    ②        ③        ④										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Logical arithmetic AND</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="331 578 1038 822"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(16 bits) ^ ACCA(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits) ^ ACCA(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>PX</td> <td>P(16 bits) ^ X(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits) ^ X(16 bits) → ACCA(16 bits)</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle            X: X-Bus output    Y: Y-Bus output            ^: Logical product (AND)</p> <ul style="list-style-type: none"> <li>● Operand ② ~ ④ For details of operand ② ~ ④, see 5.2.1 'Operand (A)'. In 'Operand (A)', "ACC" means ACCA.</li> </ul>	Operand ①	ALU operation (fixed point)	PA	P(16 bits) ^ ACCA(16 bits) → ACCA(16 bits)	YA	Y(16 bits) ^ ACCA(16 bits) → ACCA(16 bits)	PX	P(16 bits) ^ X(16 bits) → ACCA(16 bits)	YX	Y(16 bits) ^ X(16 bits) → ACCA(16 bits)
Operand ①	ALU operation (fixed point)										
PA	P(16 bits) ^ ACCA(16 bits) → ACCA(16 bits)										
YA	Y(16 bits) ^ ACCA(16 bits) → ACCA(16 bits)										
PX	P(16 bits) ^ X(16 bits) → ACCA(16 bits)										
YX	Y(16 bits) ^ X(16 bits) → ACCA(16 bits)										
Instruction code											
CCR OVFP	CCR C: Undefined. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is 0 after instruction execution.										

## II. Direct addressing mode

Assembler syntax	[<label>] ΔANDAΔ <operand (B)>[Δ <comment>]																																												
Example	ANDA Δ <u>YA</u> . <u>EE</u> , <u>4</u> , <u>24</u> ①   ②   ③																																												
Operation	<ul style="list-style-type: none"> <li>● ALU operation Logical arithmetic AND</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="341 591 1049 829"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(16 bits) ^ ACCA(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits) ^ ACCA(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>PX</td> <td>P(16 bits) ^ X(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits) ^ X(16 bits) → ACCA(16 bits)</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle            X: X-Bus output    Y: Y-Bus output            ^: Logical product (AND)</p> <ul style="list-style-type: none"> <li>● Operand ② ~ ③ For details of operand ② ~ ③, see 5.2.1 'Operand (B)'. In 'Operand (B)', "ACC" means ACCA.</li> </ul>	Operand ①	ALU operation (fixed point)	PA	P(16 bits) ^ ACCA(16 bits) → ACCA(16 bits)	YA	Y(16 bits) ^ ACCA(16 bits) → ACCA(16 bits)	PX	P(16 bits) ^ X(16 bits) → ACCA(16 bits)	YX	Y(16 bits) ^ X(16 bits) → ACCA(16 bits)																																		
Operand ①	ALU operation (fixed point)																																												
PA	P(16 bits) ^ ACCA(16 bits) → ACCA(16 bits)																																												
YA	Y(16 bits) ^ ACCA(16 bits) → ACCA(16 bits)																																												
PX	P(16 bits) ^ X(16 bits) → ACCA(16 bits)																																												
YX	Y(16 bits) ^ X(16 bits) → ACCA(16 bits)																																												
Instruction code	<table border="1" data-bbox="325 1220 910 1274"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td></td><td></td><td></td><td>1</td><td>0</td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <p>(X,Y)    ACC/    (Page) (Pointer)              →ALU    DREG    direct address                      Write</p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	1	1	0				1	0				0	0							
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
0	0	1	1	0				1	0				0	0																															
CCR  OVFP	CCR C: Undefined. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is 0 after instruction execution.																																												

# ANDB

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔANDBΔ <operand (A)>[Δ <comment>]										
Example	ANDB Δ <u>YX</u> , <u>EE</u> , <u>XY(1, 6)</u> , <u>RA, RO</u> ①   ②      ③      ④										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Logical arithmetic AND</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="332 579 1040 819"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(16 bits) ^ ACCB(16 bits) → ACCB(16 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits) ^ ACCB(16 bits) → ACCB(16 bits)</td> </tr> <tr> <td>PX</td> <td>P(16 bits) ^ X(16 bits) → ACCB(16 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits) ^ X(16 bits) → ACCB(16 bits)</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle X: X-Bus output    Y: Y-Bus output ^: Logical product (AND)</p> <ul style="list-style-type: none"> <li>● Operand ② ~ ④ For details of operand ② ~ ④, see 5.2.1 'Operand (A)'. In 'Operand (A)', "ACC" means ACCB.</li> </ul>	Operand ①	ALU operation (fixed point)	PA	P(16 bits) ^ ACCB(16 bits) → ACCB(16 bits)	YA	Y(16 bits) ^ ACCB(16 bits) → ACCB(16 bits)	PX	P(16 bits) ^ X(16 bits) → ACCB(16 bits)	YX	Y(16 bits) ^ X(16 bits) → ACCB(16 bits)
Operand ①	ALU operation (fixed point)										
PA	P(16 bits) ^ ACCB(16 bits) → ACCB(16 bits)										
YA	Y(16 bits) ^ ACCB(16 bits) → ACCB(16 bits)										
PX	P(16 bits) ^ X(16 bits) → ACCB(16 bits)										
YX	Y(16 bits) ^ X(16 bits) → ACCB(16 bits)										
Instruction code											
CCR OVFP	CCR C: Undefined. N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is 0 after instruction execution.										

## II. Direct addressing mode

Assembler syntax	[<label>] $\Delta$ ANDB $\Delta$ <operand $\textcircled{B}$ > [ $\Delta$ <comment>]																																															
Example	ANDB $\Delta$ YA, EE, 1, 45 <div style="display: flex; justify-content: center; gap: 20px; margin-top: 5px;"> <span><math>\textcircled{1}</math></span> <span><math>\textcircled{2}</math></span> <span><math>\textcircled{3}</math></span> </div>																																															
Operation	<ul style="list-style-type: none"> <li>● ALU operation Logical arithmetic AND</li> <li>● Operand <math>\textcircled{1}</math> Operand <math>\textcircled{1}</math> indicates input data of the ALU. The content of operand <math>\textcircled{1}</math> is shown in the following table.</li> </ul> <table border="1" style="margin: 10px auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th style="padding: 5px;">Operand <math>\textcircled{1}</math></th> <th style="padding: 5px;">ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">PA</td> <td style="padding: 5px;"><math>P(16 \text{ bits}) \wedge \text{ACCB}(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})</math></td> </tr> <tr> <td style="padding: 5px;">YA</td> <td style="padding: 5px;"><math>Y(16 \text{ bits}) \wedge \text{ACCB}(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})</math></td> </tr> <tr> <td style="padding: 5px;">PX</td> <td style="padding: 5px;"><math>P(16 \text{ bits}) \wedge X(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})</math></td> </tr> <tr> <td style="padding: 5px;">YX</td> <td style="padding: 5px;"><math>Y(16 \text{ bits}) \wedge X(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})</math></td> </tr> </tbody> </table> <p style="margin-top: 10px;">             P: Product of previous instruction cycle              X: X-Bus output    Y: Y-Bus output  <math>\wedge</math>: Logical product (AND)         </p> <ul style="list-style-type: none"> <li>● Operand <math>\textcircled{2} \sim \textcircled{3}</math> For details of operand <math>\textcircled{2} \sim \textcircled{3}</math>, see 5.2.1 'Operand <math>\textcircled{B}</math>'. In 'Operand <math>\textcircled{B}</math>', "ACC" means ACCB.</li> </ul>	Operand $\textcircled{1}$	ALU operation (fixed point)	PA	$P(16 \text{ bits}) \wedge \text{ACCB}(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})$	YA	$Y(16 \text{ bits}) \wedge \text{ACCB}(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})$	PX	$P(16 \text{ bits}) \wedge X(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})$	YX	$Y(16 \text{ bits}) \wedge X(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})$																																					
Operand $\textcircled{1}$	ALU operation (fixed point)																																															
PA	$P(16 \text{ bits}) \wedge \text{ACCB}(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})$																																															
YA	$Y(16 \text{ bits}) \wedge \text{ACCB}(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})$																																															
PX	$P(16 \text{ bits}) \wedge X(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})$																																															
YX	$Y(16 \text{ bits}) \wedge X(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})$																																															
Instruction code	<div style="text-align: center; margin-bottom: 5px;"> <span style="margin-right: 5px;">21</span><span style="margin-right: 5px;">20</span><span style="margin-right: 5px;">19</span><span style="margin-right: 5px;">18</span><span style="margin-right: 5px;">17</span><span style="margin-right: 5px;">16</span><span style="margin-right: 5px;">15</span><span style="margin-right: 5px;">14</span><span style="margin-right: 5px;">13</span><span style="margin-right: 5px;">12</span><span style="margin-right: 5px;">11</span><span style="margin-right: 5px;">10</span><span style="margin-right: 5px;">9</span><span style="margin-right: 5px;">8</span><span style="margin-right: 5px;">7</span><span style="margin-right: 5px;">6</span><span style="margin-right: 5px;">5</span><span style="margin-right: 5px;">4</span><span style="margin-right: 5px;">3</span><span style="margin-right: 5px;">2</span><span style="margin-right: 5px;">1</span><span style="margin-right: 5px;">0</span> </div> <table style="margin: 0 auto; border-collapse: collapse; text-align: center;"> <tr> <td style="border: 1px solid black; width: 15px; height: 15px;">0</td> <td style="border: 1px solid black; width: 15px; height: 15px;">0</td> <td style="border: 1px solid black; width: 15px; height: 15px;">1</td> <td style="border: 1px solid black; width: 15px; height: 15px;">1</td> <td style="border: 1px solid black; width: 15px; height: 15px;">0</td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;">1</td> <td style="border: 1px solid black; width: 15px; height: 15px;">1</td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;">0</td> <td style="border: 1px solid black; width: 15px; height: 15px;">0</td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> </tr> <tr> <td colspan="4" style="border: none; padding: 5px 0 5px 40px;"><math>(X, Y)</math></td> <td colspan="4" style="border: none; padding: 5px 0 5px 40px;"><math>\rightarrow</math>ALU</td> <td colspan="4" style="border: none; padding: 5px 0 5px 40px;">ACC/ DREG Write</td> <td colspan="4" style="border: none; padding: 5px 0 5px 40px;"><math>(\text{Page})</math></td> <td colspan="4" style="border: none; padding: 5px 0 5px 40px;"><math>(\text{Pointer})</math></td> <td colspan="4" style="border: none; padding: 5px 0 5px 40px;">direct address</td> </tr> </table>	0	0	1	1	0			1	1			0	0											$(X, Y)$				$\rightarrow$ ALU				ACC/ DREG Write				$(\text{Page})$				$(\text{Pointer})$				direct address			
0	0	1	1	0			1	1			0	0																																				
$(X, Y)$				$\rightarrow$ ALU				ACC/ DREG Write				$(\text{Page})$				$(\text{Pointer})$				direct address																												
CCR  OVFP	CCR    C: Undefined. N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is 0 after instruction execution.																																															

# ORA

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔORAAΔ <operand $\textcircled{A}$ >[Δ <comment>]										
Example	ORAAΔ <u>YA</u> , <u>EE</u> , <u>XY (1, 6)</u> , RA ①    ②    ③    ④										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Logical arithmetic OR</li> <li>● Operand <math>\textcircled{1}</math> Operand <math>\textcircled{1}</math> indicates input data of the ALU. The content of operand <math>\textcircled{1}</math> is shown in the following table.</li> </ul> <table border="1" data-bbox="340 572 1045 815"> <thead> <tr> <th>Operand <math>\textcircled{1}</math></th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(16 bits) ∨ ACCA(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits) ∨ ACCA(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>PX</td> <td>P(16 bits) ∨ X(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits) ∨ X(16 bits) → ACCA(16 bits)</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle            X: X-Bus output    Y: Y-Bus output            ∨: Logical sum (OR)</p> <ul style="list-style-type: none"> <li>● Operand <math>\textcircled{2} \sim \textcircled{4}</math> For details of operand <math>\textcircled{2} \sim \textcircled{4}</math>, see 5.2.1 for 'Operand <math>\textcircled{A}</math>'. In 'Operand <math>\textcircled{A}</math>', "ACC" means ACCA.</li> </ul>	Operand $\textcircled{1}$	ALU operation (fixed point)	PA	P(16 bits) ∨ ACCA(16 bits) → ACCA(16 bits)	YA	Y(16 bits) ∨ ACCA(16 bits) → ACCA(16 bits)	PX	P(16 bits) ∨ X(16 bits) → ACCA(16 bits)	YX	Y(16 bits) ∨ X(16 bits) → ACCA(16 bits)
Operand $\textcircled{1}$	ALU operation (fixed point)										
PA	P(16 bits) ∨ ACCA(16 bits) → ACCA(16 bits)										
YA	Y(16 bits) ∨ ACCA(16 bits) → ACCA(16 bits)										
PX	P(16 bits) ∨ X(16 bits) → ACCA(16 bits)										
YX	Y(16 bits) ∨ X(16 bits) → ACCA(16 bits)										
Instruction code											
CCR OVFP	CCR C: Undefined. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is 0 after instruction execution.										

## II. Direct addressing mode

Assembler syntax	[<label>] ΔORAΔ <operand <b>B</b> >[Δ <comment>]																																																								
Example	$\text{ORA} \Delta \underset{\textcircled{1}}{\text{YA}}, \underset{\textcircled{2}}{\text{EE}}, \underset{\textcircled{3}}{\text{6}}, \text{23}$																																																								
Operation	<ul style="list-style-type: none"> <li>● ALU operation Logical arithmetic OR</li> <li>● Operand <b>1</b> Operand indicates input data of the ALU. The content of operand <b>1</b> is shown in the following table.</li> </ul> <table border="1" style="margin: 10px auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th style="padding: 5px;">Operand <b>1</b></th> <th style="padding: 5px;">ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">PA</td> <td style="padding: 5px;">P(16 bits) ∨ ACCA(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td style="padding: 5px;">YA</td> <td style="padding: 5px;">Y(16 bits) ∨ ACCA(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td style="padding: 5px;">PX</td> <td style="padding: 5px;">P(16 bits) ∨ X(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td style="padding: 5px;">YX</td> <td style="padding: 5px;">Y(16 bits) ∨ X(16 bits) → ACCA(16 bits)</td> </tr> </tbody> </table> <p style="margin-left: 20px;">                     P: Product of previous instruction cycle                      X: X-Bus output    Y: Y-Bus output                      ∨: Logical sum (OR)                 </p> <ul style="list-style-type: none"> <li>● Operand <b>2</b> ~ <b>3</b> For details of operand <b>2</b> ~ <b>3</b>, see 5.2.1 'Operand <b>B</b>'. In 'Operand <b>B</b>', "ACC" means ACCA.</li> </ul>	Operand <b>1</b>	ALU operation (fixed point)	PA	P(16 bits) ∨ ACCA(16 bits) → ACCA(16 bits)	YA	Y(16 bits) ∨ ACCA(16 bits) → ACCA(16 bits)	PX	P(16 bits) ∨ X(16 bits) → ACCA(16 bits)	YX	Y(16 bits) ∨ X(16 bits) → ACCA(16 bits)																																														
Operand <b>1</b>	ALU operation (fixed point)																																																								
PA	P(16 bits) ∨ ACCA(16 bits) → ACCA(16 bits)																																																								
YA	Y(16 bits) ∨ ACCA(16 bits) → ACCA(16 bits)																																																								
PX	P(16 bits) ∨ X(16 bits) → ACCA(16 bits)																																																								
YX	Y(16 bits) ∨ X(16 bits) → ACCA(16 bits)																																																								
Instruction code	<div style="text-align: center;"> <table style="margin: 0 auto; border-collapse: collapse;"> <tr> <td style="padding: 0 5px;">21</td><td style="padding: 0 5px;">20</td><td style="padding: 0 5px;">19</td><td style="padding: 0 5px;">18</td><td style="padding: 0 5px;">17</td><td style="padding: 0 5px;">16</td><td style="padding: 0 5px;">15</td><td style="padding: 0 5px;">14</td><td style="padding: 0 5px;">13</td><td style="padding: 0 5px;">12</td><td style="padding: 0 5px;">11</td><td style="padding: 0 5px;">10</td><td style="padding: 0 5px;">9</td><td style="padding: 0 5px;">8</td><td style="padding: 0 5px;">7</td><td style="padding: 0 5px;">6</td><td style="padding: 0 5px;">5</td><td style="padding: 0 5px;">4</td><td style="padding: 0 5px;">3</td><td style="padding: 0 5px;">2</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td> </tr> <tr> <td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td> </tr> </table> <div style="margin-top: 5px; text-align: center;"> <table style="margin: 0 auto; border-collapse: collapse;"> <tr> <td style="padding: 0 10px;">(X,Y)</td> <td style="padding: 0 10px;">ACC/</td> <td style="padding: 0 10px;">(Page)</td> <td style="padding: 0 10px;">(Pointer)</td> </tr> <tr> <td style="padding: 0 10px;">→ALU</td> <td style="padding: 0 10px;">DREG</td> <td colspan="2" style="padding: 0 10px;">direct address</td> </tr> <tr> <td colspan="4" style="padding: 0 10px;">Write</td> </tr> </table> </div> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	(X,Y)	ACC/	(Page)	(Pointer)	→ALU	DREG	direct address		Write			
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																				
0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																				
(X,Y)	ACC/	(Page)	(Pointer)																																																						
→ALU	DREG	direct address																																																							
Write																																																									
CCR  OVFP	CCR C: Undefined. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is 0 after instruction execution.																																																								

# ORB

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔORBΔ <operand (A)>[Δ <comment>]										
Example	ORBΔ <u>YA</u> , <u>EE</u> , <u>XY(1,6)</u> , <u>RO</u> ①  ②          ③          ④										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Logical arithmetic OR</li> <li>● Operand ① Operand 1 indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="340 572 1045 815"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(16 bits) ∨ ACCB(16 bits) → ACCB(16 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits) ∨ ACCB(16 bits) → ACCB(16 bits)</td> </tr> <tr> <td>PX</td> <td>P(16 bits) ∨ X(16 bits) → ACCB(16 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits) ∨ X(16 bits) → ACCB(16 bits)</td> </tr> </tbody> </table> <p>P: Product of previous instruction cycle          X: X-Bus output    Y: Y-Bus output          ∨: Logical sum (OR)</p> <ul style="list-style-type: none"> <li>● Operand ② ~ ④ For details of operand ② ~ ④, see 5.2.1 'Operand (A)'. In 'Operand (A)', "ACC" means ACCB.</li> </ul>	Operand ①	ALU operation (fixed point)	PA	P(16 bits) ∨ ACCB(16 bits) → ACCB(16 bits)	YA	Y(16 bits) ∨ ACCB(16 bits) → ACCB(16 bits)	PX	P(16 bits) ∨ X(16 bits) → ACCB(16 bits)	YX	Y(16 bits) ∨ X(16 bits) → ACCB(16 bits)
Operand ①	ALU operation (fixed point)										
PA	P(16 bits) ∨ ACCB(16 bits) → ACCB(16 bits)										
YA	Y(16 bits) ∨ ACCB(16 bits) → ACCB(16 bits)										
PX	P(16 bits) ∨ X(16 bits) → ACCB(16 bits)										
YX	Y(16 bits) ∨ X(16 bits) → ACCB(16 bits)										
Instruction code											
CCR OVFP	<p>CCR C: Undefined.          N: Set if ACCB is negative after instruction execution.          Z: Set if ACCB is 0 after instruction execution.</p>										



## II. Direct addressing mode

<b>Assembler syntax</b>	$[\langle \text{label} \rangle] \Delta \text{ORBA} \langle \text{operand} \textcircled{\text{B}} \rangle [\Delta \langle \text{comment} \rangle]$																																													
<b>Example</b>	$\text{ORB} \triangle \underbrace{\text{YA}}_{\textcircled{1}}, \underbrace{\text{EE}}_{\textcircled{2}}, \underbrace{6, 23}_{\textcircled{3}}$																																													
<b>Operation</b>	<ul style="list-style-type: none"> <li>● ALU operation Logical arithmetic OR</li> <li>● Operand <math>\textcircled{1}</math> Operand <math>\textcircled{1}</math> indicates input data of the ALU. The content of operand <math>\textcircled{1}</math> is shown in the following table.</li> </ul> <table border="1" data-bbox="346 591 1057 829"> <thead> <tr> <th>Operand <math>\textcircled{1}</math></th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td><math>P(16 \text{ bits}) \vee \text{ACCB}(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})</math></td> </tr> <tr> <td>YA</td> <td><math>Y(16 \text{ bits}) \vee \text{ACCB}(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})</math></td> </tr> <tr> <td>PX</td> <td><math>P(16 \text{ bits}) \vee X(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})</math></td> </tr> <tr> <td>YX</td> <td><math>Y(16 \text{ bits}) \vee X(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})</math></td> </tr> </tbody> </table> <p> P: Product of previous instruction cycle  X: X-Bus output    Y: Y-Bus output  <math>\vee</math>: Logical sum (OR) </p> <ul style="list-style-type: none"> <li>● Operand <math>\textcircled{2} \sim \textcircled{3}</math> For details of operand <math>\textcircled{2} \sim \textcircled{3}</math>, see 5.2.1 'Operand <math>\textcircled{\text{B}}</math>'. In 'Operand <math>\textcircled{\text{B}}</math>', "ACC" means ACCB.</li> </ul>	Operand $\textcircled{1}$	ALU operation (fixed point)	PA	$P(16 \text{ bits}) \vee \text{ACCB}(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})$	YA	$Y(16 \text{ bits}) \vee \text{ACCB}(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})$	PX	$P(16 \text{ bits}) \vee X(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})$	YX	$Y(16 \text{ bits}) \vee X(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})$																																			
Operand $\textcircled{1}$	ALU operation (fixed point)																																													
PA	$P(16 \text{ bits}) \vee \text{ACCB}(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})$																																													
YA	$Y(16 \text{ bits}) \vee \text{ACCB}(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})$																																													
PX	$P(16 \text{ bits}) \vee X(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})$																																													
YX	$Y(16 \text{ bits}) \vee X(16 \text{ bits}) \rightarrow \text{ACCB}(16 \text{ bits})$																																													
<b>Instruction code</b>	<table border="1" data-bbox="341 1225 924 1277"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td></td><td></td><td></td><td>1</td><td>1</td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <p> (X,Y)            ACC/            (Page) (Pointer)  →ALU            DREG            direct address  Write </p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	1	0	0				1	1			0	0									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																									
0	0	1	0	0				1	1			0	0																																	
<b>CCR</b>  <b>OVFP</b>	CCR C: Undefined. N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is 0 after instruction execution.																																													

# EORA

## I. Pointer addressing mode

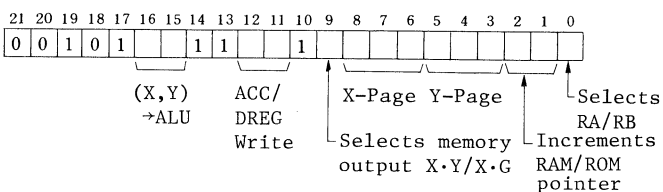
Assembler syntax	[<label>] ΔEORAA <OPERAND (A)>[Δ <comment>]										
Example	EORA Δ <u>YA</u> , <u>EE</u> , <u>XY (2, 7)</u> , <u>RA</u> ①    ②       ③       ④										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Logical arithmetic EOR</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="335 583 1043 822"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(16 bits) ⊕ ACCA(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits) ⊕ ACCA(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>PX</td> <td>P(16 bits) ⊕ X(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits) ⊕ X(16 bits) → ACCA(16 bits)</td> </tr> </tbody> </table> <p>           P: Product of previous instruction cycle            X: X-Bus output    Y: Y-Bus output            ⊕: Exclusive-OR operation         </p> <ul style="list-style-type: none"> <li>● Operand ② ~ ④ For details of operand ② ~ ④, see 5.2.1 for 'Operand (A)'. In 'Operand (A)', "ACC" means ACCA.</li> </ul>	Operand ①	ALU operation (fixed point)	PA	P(16 bits) ⊕ ACCA(16 bits) → ACCA(16 bits)	YA	Y(16 bits) ⊕ ACCA(16 bits) → ACCA(16 bits)	PX	P(16 bits) ⊕ X(16 bits) → ACCA(16 bits)	YX	Y(16 bits) ⊕ X(16 bits) → ACCA(16 bits)
Operand ①	ALU operation (fixed point)										
PA	P(16 bits) ⊕ ACCA(16 bits) → ACCA(16 bits)										
YA	Y(16 bits) ⊕ ACCA(16 bits) → ACCA(16 bits)										
PX	P(16 bits) ⊕ X(16 bits) → ACCA(16 bits)										
YX	Y(16 bits) ⊕ X(16 bits) → ACCA(16 bits)										
Instruction code											
CCR OVFP	CCR C: Undefined. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is 0 after instruction execution.										

## II. Direct addressing mode

Assembler syntax	[<label>] ΔEORAA <operand <b>Ⓐ</b> >[Δ <comment>]																																													
Example	EORA Δ <u>YA</u> , <u>EE</u> , <u>3</u> , <u>39</u> ①   ②       ③																																													
Operation	<ul style="list-style-type: none"> <li>● ALU operation Logical arithmetic EOR</li> <li>● Operand <b>Ⓐ</b> Operand <b>Ⓐ</b> indicates input data of the ALU. The content of operand <b>Ⓐ</b> is shown in the following table.</li> </ul> <table border="1" data-bbox="350 591 1057 829"> <thead> <tr> <th>Operand <b>Ⓐ</b></th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(16 bits) ⊕ ACCA(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits) ⊕ ACCA(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>PX</td> <td>P(16 bits) ⊕ X(16 bits) → ACCA(16 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits) ⊕ X(16 bits) → ACCA(16 bits)</td> </tr> </tbody> </table> <p>           P: Product of previous instruction cycle            X: X-Bus output    Y: Y-Bus output            ⊕: Exclusive-OR operation         </p> <ul style="list-style-type: none"> <li>● Operand <b>Ⓑ</b> ~ <b>Ⓒ</b> For details of Operand <b>Ⓑ</b> ~ <b>Ⓒ</b>, see 5.2.1 'Operand <b>Ⓐ</b>'. In 'Operand <b>Ⓐ</b>', "ACC" means ACCA.</li> </ul>	Operand <b>Ⓐ</b>	ALU operation (fixed point)	PA	P(16 bits) ⊕ ACCA(16 bits) → ACCA(16 bits)	YA	Y(16 bits) ⊕ ACCA(16 bits) → ACCA(16 bits)	PX	P(16 bits) ⊕ X(16 bits) → ACCA(16 bits)	YX	Y(16 bits) ⊕ X(16 bits) → ACCA(16 bits)																																			
Operand <b>Ⓐ</b>	ALU operation (fixed point)																																													
PA	P(16 bits) ⊕ ACCA(16 bits) → ACCA(16 bits)																																													
YA	Y(16 bits) ⊕ ACCA(16 bits) → ACCA(16 bits)																																													
PX	P(16 bits) ⊕ X(16 bits) → ACCA(16 bits)																																													
YX	Y(16 bits) ⊕ X(16 bits) → ACCA(16 bits)																																													
Instruction code	<table border="1" data-bbox="333 1216 918 1281"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td></td><td></td><td></td><td>1</td><td>0</td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <p>           (X,Y)            ACC/                      →ALU        DREG                                      Write         </p> <p>           (Page) (Pointer)                      direct address         </p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	1	0	1				1	0			0	0									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																									
0	0	1	0	1				1	0			0	0																																	
CCR  OVFP	CCR C: Undefined. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is 0 after instruction execution.																																													

# EORB

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔEORBΔ <operand A>[Δ <comment>]										
Example	EORB Δ YA, EE, XY(2.7), RA ①  ②      ③      ④										
Operation	<ul style="list-style-type: none"> <li>● ALU operation Logical arithmetic EOR</li> <li>● Operand ① Operand ① indicates input data of the ALU. The content of operand ① is shown in the following table.</li> </ul> <table border="1" data-bbox="327 572 1033 815"> <thead> <tr> <th>Operand ①</th> <th>ALU operation (fixed point)</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>P(16 bits) ⊕ ACCB(16 bits) → ACCB(16 bits)</td> </tr> <tr> <td>YA</td> <td>Y(16 bits) ⊕ ACCB(16 bits) → ACCB(16 bits)</td> </tr> <tr> <td>PX</td> <td>P(16 bits) ⊕ X(16 bits) → ACCB(16 bits)</td> </tr> <tr> <td>YX</td> <td>Y(16 bits) ⊕ X(16 bits) → ACCB(16 bits)</td> </tr> </tbody> </table> <p>           P: Product of previous instruction cycle            X: X-Bus output    Y: Y-Bus output            ⊕: Exclusive-OR operation         </p> <ul style="list-style-type: none"> <li>● Operand ② ~ ④ For details of operand ② ~ ④, see 5.2.1 'Operand A'. In 'Operand A', "ACC" means ACCB.</li> </ul>	Operand ①	ALU operation (fixed point)	PA	P(16 bits) ⊕ ACCB(16 bits) → ACCB(16 bits)	YA	Y(16 bits) ⊕ ACCB(16 bits) → ACCB(16 bits)	PX	P(16 bits) ⊕ X(16 bits) → ACCB(16 bits)	YX	Y(16 bits) ⊕ X(16 bits) → ACCB(16 bits)
Operand ①	ALU operation (fixed point)										
PA	P(16 bits) ⊕ ACCB(16 bits) → ACCB(16 bits)										
YA	Y(16 bits) ⊕ ACCB(16 bits) → ACCB(16 bits)										
PX	P(16 bits) ⊕ X(16 bits) → ACCB(16 bits)										
YX	Y(16 bits) ⊕ X(16 bits) → ACCB(16 bits)										
Instruction code	 <p>           Bit field diagram for instruction code (bits 21 to 0):            Bit 21: 0, Bit 20: 0, Bit 19: 1, Bit 18: 0, Bit 17: 1, Bits 16-14: (X,Y) → ALU, Bit 13: 1, Bit 12: 1, Bits 11-9: ACC/DREG Write, Bit 8: 1, Bits 7-6: X-Page, Bits 5-4: Y-Page, Bits 3-2: Selects RA/RB Increments, Bit 1: 1, Bit 0: 0.         </p>										
CCR OVFP	CCR C: Undefined. N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is 0 after instruction execution.										



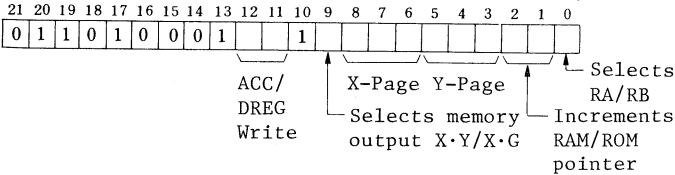


## II. Direct addressing mode

Assembler syntax	[<label>] ΔFABSΔΔ <operand ①>[Δ <comment>]																																																														
Example	$\text{FABSA} \triangle \text{EE}, \underset{\textcircled{2}}{0}, \underset{\textcircled{3}}{00}$																																																														
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <math> \text{ACCA}(20 \text{ bits})  \rightarrow \text{ACCA}(20 \text{ bits})</math> </div> <p>The obtained absolute value is stored in the ACCA. The negative maximum value generates an overflow when the absolute value is obtained. Therefore the absolute value is fixed to;</p> <p>Mantissa \$8000 → \$7FFF</p> <p>Exponent +7 → +7</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCA.</li> </ul>																																																														
Instruction code	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="padding: 0 5px;">21</td><td style="padding: 0 5px;">20</td><td style="padding: 0 5px;">19</td><td style="padding: 0 5px;">18</td><td style="padding: 0 5px;">17</td><td style="padding: 0 5px;">16</td><td style="padding: 0 5px;">15</td><td style="padding: 0 5px;">14</td><td style="padding: 0 5px;">13</td><td style="padding: 0 5px;">12</td><td style="padding: 0 5px;">11</td><td style="padding: 0 5px;">10</td><td style="padding: 0 5px;">9</td><td style="padding: 0 5px;">8</td><td style="padding: 0 5px;">7</td><td style="padding: 0 5px;">6</td><td style="padding: 0 5px;">5</td><td style="padding: 0 5px;">4</td><td style="padding: 0 5px;">3</td><td style="padding: 0 5px;">2</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td> </tr> <tr> <td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td> </tr> <tr> <td colspan="12" style="text-align: center;">ACC/ DREG Write</td> <td colspan="2" style="text-align: center;">(Page)</td> <td colspan="4" style="text-align: center;">(Pointer) direct address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ACC/ DREG Write												(Page)		(Pointer) direct address			
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																										
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																										
ACC/ DREG Write												(Page)		(Pointer) direct address																																																	
CCR  OVFP	<p>CCR C: 0 N: 0 Z: Set if a mantissa of ACCA is \$0000.</p> <p>OVFP OVFP bit of the CTR must be set to 1 beforehand.</p>																																																														

# FABSB

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔFABSBA <operand ©>[Δ <comment>]
Example	$\text{FABSB } \triangle \text{EE}, \underbrace{\text{XG}(1, 3)}_{\textcircled{3}}, \underbrace{\text{RA}}_{\textcircled{4}}$
Operation	<p>● ALU operation Floating point arithmetic</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <math> \text{ACCB}(20 \text{ bits})  \rightarrow \text{ACCB}(20 \text{ bits})</math> </div> <p>The obtained absolute value is stored in the ACCB. The negative maximum value generates an overflow when the absolute value is obtained. Therefore the absolute value is fixed to;</p> <p>Mantissa \$8000 → \$7FFF</p> <p>Exponent +7 → +7</p> <p>● Operand For details of operand, see 5.2.1 'Operand ©'. In 'Operand ©', "ACC" means ACCB.</p>
Instruction code	 <p>The diagram shows a 22-bit instruction code field. Bits 21-13 are 011010001. Bit 12 is ACC/DREG Write. Bit 11 is 1. Bits 10-9 are X-Page Y-Page. Bits 8-2 are 111111. Bit 1 is RA/RB. Bit 0 is increments RAM/ROM pointer.</p>
CCR OVFP	<p>CCR C: 0 N: 0</p> <p>OVFP Z: Set if a mantissa of ACCB is \$0000.</p> <p>OVFP OVFP bit of the CTR must be set to 1 beforehand.</p>



## II. Direct addressing mode

Assembler syntax	[<label>] ΔFABSBA <operand ①>[Δ <comment>]																																												
Example	$\text{FABSB } \triangle \text{EE, } \underset{\textcircled{2}}{0}, \underset{\textcircled{3}}{00}$																																												
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <math> \text{ACCB}(20 \text{ bits})  \rightarrow \text{ACCB}(20 \text{ bits})</math> </div> <p>The obtained absolute value is stored in the ACCB. The negative maximum value generates an overflow when the absolute value is obtained. Therefore the absolute value is fixed to;</p> <p>Mantissa \$8000 → \$7FFF</p> <p>Exponent +7 → +7</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCB.</li> </ul>																																												
Instruction code	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="padding: 0 5px;">21</td><td style="padding: 0 5px;">20</td><td style="padding: 0 5px;">19</td><td style="padding: 0 5px;">18</td><td style="padding: 0 5px;">17</td><td style="padding: 0 5px;">16</td><td style="padding: 0 5px;">15</td><td style="padding: 0 5px;">14</td><td style="padding: 0 5px;">13</td><td style="padding: 0 5px;">12</td><td style="padding: 0 5px;">11</td><td style="padding: 0 5px;">10</td><td style="padding: 0 5px;">9</td><td style="padding: 0 5px;">8</td><td style="padding: 0 5px;">7</td><td style="padding: 0 5px;">6</td><td style="padding: 0 5px;">5</td><td style="padding: 0 5px;">4</td><td style="padding: 0 5px;">3</td><td style="padding: 0 5px;">2</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td> </tr> <tr> <td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td> </tr> </table> <div style="margin-left: 100px; margin-top: 5px;"> <span style="margin-right: 100px;">ACC/ DREG Write</span> <span>(Page) (Pointer) direct address</span> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	1	0	1	0	0	0	1			0	0									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
0	1	1	0	1	0	0	0	1			0	0																																	
CCR  OVFP	<p>CCR C: 0 N: 0 Z: Set if a mantissa of ACCB is \$0000.</p> <p>OVFP OVFP bit of the CTR must be set to 1 beforehand.</p>																																												

# ABSA

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔABSΔ <operand ©>[Δ <comment>]
Example	$ABSA \triangle \underset{\textcircled{2}}{EE}, \underset{\textcircled{3}}{XG(1,3)}, \underset{\textcircled{4}}{RA}$
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <math> ACCA(16 \text{ bits})  \rightarrow ACCA(16 \text{ bits})</math> </div> <p>The obtained absolute value is stored in the ACCA. The exponent part (4 bits) of the result is undefined.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 "Operand ©". In "Operand ©", "ACC" means ACCA.</li> </ul>
Instruction code	<p>21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</p> <p>0 1 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0</p> <p>ACC/DREG Write (bits 12-10) X-Page Y-Page (bits 9-2) Selects RA/RB Increments RAM/ROM pointer (bit 1)</p>
CCR OVFP	<p>CCR C: 0 N: Set if ACCA=\$8000 before arithmetic operation and OVFP bit is 0 ; cleared otherwise. Z: Set if ACCA=\$0000.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit. 0: no overflow protection 1: overflow protection</p>

## II. Direct addressing mode

<b>Assembler syntax</b>	<code>[&lt;label&gt;] ΔABSAD &lt;operand ①&gt;[Δ &lt;comment&gt;]</code>																																																																		
<b>Example</b>	$ABSAD \frac{A}{②}, \frac{2,14}{③}$																																																																		
<b>Operation</b>	<ul style="list-style-type: none"> <li>● <b>ALU operation</b> Fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <math> ACCA(16 \text{ bits})  \rightarrow ACCA(16 \text{ bits})</math> </div> <p>The obtained absolute value is stored in the ACCA. The exponent part (4 bits) of the result is undefined.</p> <ul style="list-style-type: none"> <li>● <b>Operand</b> For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCA.</li> </ul>																																																																		
<b>Instruction code</b>	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="text-align: center;">21</td><td style="text-align: center;">20</td><td style="text-align: center;">19</td><td style="text-align: center;">18</td><td style="text-align: center;">17</td><td style="text-align: center;">16</td><td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td> </tr> <tr> <td colspan="12" style="text-align: center;">ACC/ DREG Write</td> <td colspan="10" style="text-align: center;">(Page) (Pointer) direct address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ACC/ DREG Write												(Page) (Pointer) direct address									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																														
0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																														
ACC/ DREG Write												(Page) (Pointer) direct address																																																							
<b>CCR</b>  <b>OVFP</b>	<p>CCR C: 0 N: Set if ACCA=\$8000 before arithmetic operation and OVFP bit is 0 ; cleared otherwise. Z: Set if ACCA=\$0000.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit. 0: no overflow protection 1: overflow protection</p>																																																																		



## II. Direct addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔABS B Δ &lt;operand ①&gt; [Δ &lt;comment&gt;]</p>																																																																																										
<p>Example</p>	<p>ABS B Δ <u>A</u>, <u>2, 14</u>                    ②     ③</p>																																																																																										
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation Fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <p> ACCB(16 bits)  → ACCB(16 bits)</p> </div> <p>The obtained absolute value is stored in the ACCB. The exponent part (4 bits) of the result is undefined.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCB.</li> </ul>																																																																																										
<p>Instruction code</p>	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="text-align: center;">21</td><td style="text-align: center;">20</td><td style="text-align: center;">19</td><td style="text-align: center;">18</td><td style="text-align: center;">17</td><td style="text-align: center;">16</td><td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td> </tr> <tr> <td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td style="text-align: center;">ACC/ DREG Write</td><td></td><td></td><td></td><td style="text-align: center;">(Page)</td><td></td><td></td><td></td><td></td><td></td><td></td><td style="text-align: center;">(Pointer)</td><td></td><td></td> </tr> <tr> <td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	1	0	1	0	0	1	1			0	0																			ACC/ DREG Write				(Page)							(Pointer)																									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																						
0	1	1	0	1	0	0	1	1			0	0																																																																															
									ACC/ DREG Write				(Page)							(Pointer)																																																																							
<p>CCR OVFP</p>	<p>CCR C: 0 N: Set if ACCB=\$8000 before arithmetic operation and OVFP bit is 0; cleared otherwise. Z: Set if ACCB=\$0000.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit. 0: no overflow protection 1: overflow protection</p>																																																																																										



## II. Direct addressing mode

Assembler syntax	[<label>] ΔFRPTAΔ <operand ①>[Δ <comment>]																																																																		
Example	FRPTA Δ <u>A</u> , <u>2</u> , <u>14</u> ②     ③																																																																		
Operation	<p>Repeat instruction</p> <ul style="list-style-type: none"> <li>● The FRPTA instruction repeats the operation of the next instruction.</li> <li>● Repeat count is specified by the RC. The actual repeat count is one more than the value loaded into the RC.</li> <li>● The other operations are the same as ones in pointer addressing mode.</li> </ul> <p>For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCA.</p> <p>Example</p> <table border="1"> <tr> <td>Repeat instruction</td> <td>RC=3 (The RC must be loaded with a value) FRPTAΔEE,0,00 ;starting operation FADAΔΔYA,EE,XY(0,0),RA+;addition RC decrement</td> </tr> <tr> <td>Equivalent instruction</td> <td>FNOPAΔEE,0,00 ;ineffective instruction FADAΔΔYA,EE,XY(0,0),RA+ } FADAΔΔYA,EE,XY(0,0),RA+ } four times FADAΔΔYA,EE,XY(0,0),RA+ } FADAΔΔYA,EE,XY(0,0),RA+ }</td> </tr> </table>	Repeat instruction	RC=3 (The RC must be loaded with a value) FRPTAΔEE,0,00 ;starting operation FADAΔΔYA,EE,XY(0,0),RA+;addition RC decrement	Equivalent instruction	FNOPAΔEE,0,00 ;ineffective instruction FADAΔΔYA,EE,XY(0,0),RA+ } FADAΔΔYA,EE,XY(0,0),RA+ } four times FADAΔΔYA,EE,XY(0,0),RA+ } FADAΔΔYA,EE,XY(0,0),RA+ }																																																														
Repeat instruction	RC=3 (The RC must be loaded with a value) FRPTAΔEE,0,00 ;starting operation FADAΔΔYA,EE,XY(0,0),RA+;addition RC decrement																																																																		
Equivalent instruction	FNOPAΔEE,0,00 ;ineffective instruction FADAΔΔYA,EE,XY(0,0),RA+ } FADAΔΔYA,EE,XY(0,0),RA+ } four times FADAΔΔYA,EE,XY(0,0),RA+ } FADAΔΔYA,EE,XY(0,0),RA+ }																																																																		
Instruction code	<table style="margin-left: auto; margin-right: auto;"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> <tr> <td colspan="11" style="text-align: center;">ACC/ DREG Write</td> <td colspan="3" style="text-align: center;">(Page)</td> <td colspan="8" style="text-align: center;">(Pointer) direct address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0									ACC/ DREG Write											(Page)			(Pointer) direct address							
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																														
1	1	1	1	0	0	0	0	0	0	0	0	0	0																																																						
ACC/ DREG Write											(Page)			(Pointer) direct address																																																					
CCR  OVFP	CCR C: } N: } not affected Z: }																																																																		





## II. Direct addressing mode

Assembler syntax	[<label>] ΔFRPTBΔ <operand ①>[Δ <comment>]																																												
Example	FRPTBΔA, 2, 14 <div style="display: flex; justify-content: center; gap: 20px; margin-top: 5px;"> <span>②</span> <span>③</span> </div>																																												
Operation	<p>Repeat instruction</p> <ul style="list-style-type: none"> <li>● The FRPTB instruction repeats the operation of the next instruction.</li> <li>● Repeat count is specified by the RC. The actual repeat count is one more than the value loaded into the RC.</li> <li>● The other operations are the same as ones in pointer addressing mode.</li> </ul> <p>For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCB.</p> <p>Example</p> <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 10px;"> <tr> <td style="width: 30%; padding: 5px;">Repeat instruction</td> <td style="padding: 5px;">RC=3 (The RC must be loaded with a value) FRPTBΔEE,0,00 ;starting operation FADBΔYA,EE,XY(0,0),RA+;addition RC decrement</td> </tr> <tr> <td style="padding: 5px;">Equivalent instruction</td> <td style="padding: 5px;"> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 80%;">                     FNOPBΔEE,0,00 ;ineffective instruction                      FADBΔYA,EE,XY(0,0),RA+                      FADBΔYA,EE,XY(0,0),RA+                      FADBΔYA,EE,XY(0,0),RA+                      FADBΔYA,EE,XY(0,0),RA+                 </td> <td style="width: 20%; vertical-align: middle; text-align: center;">                     }                      four times                 </td> </tr> </table> </td> </tr> </table>	Repeat instruction	RC=3 (The RC must be loaded with a value) FRPTBΔEE,0,00 ;starting operation FADBΔYA,EE,XY(0,0),RA+;addition RC decrement	Equivalent instruction	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 80%;">                     FNOPBΔEE,0,00 ;ineffective instruction                      FADBΔYA,EE,XY(0,0),RA+                      FADBΔYA,EE,XY(0,0),RA+                      FADBΔYA,EE,XY(0,0),RA+                      FADBΔYA,EE,XY(0,0),RA+                 </td> <td style="width: 20%; vertical-align: middle; text-align: center;">                     }                      four times                 </td> </tr> </table>	FNOPBΔEE,0,00 ;ineffective instruction FADBΔYA,EE,XY(0,0),RA+ FADBΔYA,EE,XY(0,0),RA+ FADBΔYA,EE,XY(0,0),RA+ FADBΔYA,EE,XY(0,0),RA+	} four times																																						
Repeat instruction	RC=3 (The RC must be loaded with a value) FRPTBΔEE,0,00 ;starting operation FADBΔYA,EE,XY(0,0),RA+;addition RC decrement																																												
Equivalent instruction	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 80%;">                     FNOPBΔEE,0,00 ;ineffective instruction                      FADBΔYA,EE,XY(0,0),RA+                      FADBΔYA,EE,XY(0,0),RA+                      FADBΔYA,EE,XY(0,0),RA+                      FADBΔYA,EE,XY(0,0),RA+                 </td> <td style="width: 20%; vertical-align: middle; text-align: center;">                     }                      four times                 </td> </tr> </table>	FNOPBΔEE,0,00 ;ineffective instruction FADBΔYA,EE,XY(0,0),RA+ FADBΔYA,EE,XY(0,0),RA+ FADBΔYA,EE,XY(0,0),RA+ FADBΔYA,EE,XY(0,0),RA+	} four times																																										
FNOPBΔEE,0,00 ;ineffective instruction FADBΔYA,EE,XY(0,0),RA+ FADBΔYA,EE,XY(0,0),RA+ FADBΔYA,EE,XY(0,0),RA+ FADBΔYA,EE,XY(0,0),RA+	} four times																																												
Instruction code	<table style="width: 100%; border-collapse: collapse; margin-bottom: 10px;"> <tr> <td style="text-align: center;">21</td><td style="text-align: center;">20</td><td style="text-align: center;">19</td><td style="text-align: center;">18</td><td style="text-align: center;">17</td><td style="text-align: center;">16</td><td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td> </tr> </table> <div style="display: flex; justify-content: space-around; margin-top: 5px;"> <div style="text-align: center;">ACC/ DREG Write</div> <div style="text-align: center;">(Page)</div> <div style="text-align: center;">(Pointer) direct address</div> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	0	0	0	1			0	0									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
1	1	1	1	0	0	0	0	1			0	0																																	
CCR  OVFP	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 15%;">CCR</td> <td style="width: 15%;">C:</td> <td rowspan="3" style="font-size: 3em; vertical-align: middle;">}</td> <td rowspan="3" style="vertical-align: middle;">not affected</td> </tr> <tr> <td></td> <td>N:</td> </tr> <tr> <td></td> <td>Z:</td> </tr> </table>	CCR	C:	}	not affected		N:		Z:																																				
CCR	C:	}	not affected																																										
	N:																																												
	Z:																																												









# FNEGA

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔFNEGAA <operand ②>[Δ <comment>]
Example	FNEGA Δ <sub>②</sub> EE, XG(1,3), RA <sub>④</sub> ③
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <math>-ACCA(20 \text{ bits}) \rightarrow ACCA(20 \text{ bits})</math> </div> <p>A two's complement of ACCA is stored in the ACCA.</p> <p>An overflow occurs when the mantissa and exponent of ACCA are \$8000 and \$7 respectively before this instruction is executed.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ②'. In 'Operand ④', "ACC" means ACCA.</li> </ul>
Instruction code	
CCR OVFP	<p>CCR C: Set if a mantissa of ACCA is \$0000 before instruction execution; cleared otherwise.</p> <p>N: Set if ACCA is negative after instruction execution.</p> <p>Z: Set if a mantissa of ACCA is \$0000 after instruction execution.</p> <p>OVFP OVFP bit of the CTR must be set to 1 beforehand.</p>

## II. Direct addressing mode

<b>Assembler syntax</b>	<code>[&lt;label&gt;] ΔFNEGAΔ &lt;operand ①&gt;[Δ &lt;comment&gt;]</code>																																												
<b>Example</b>	<code>FNEGA Δ EE, 0, 00</code> ②      ③																																												
<b>Operation</b>	<ul style="list-style-type: none"> <li>● ALU operation Floating point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <code>-ACCA(20 bits) → ACCA(20 bits)</code> </div> <p>A two's complement of ACCA is stored in the ACCA.</p> <p>An overflow occurs when the mantissa and exponent of ACCA are \$8000 and \$7 respectively before this instruction is executed.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCA.</li> </ul>																																												
<b>Instruction code</b>	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="padding: 0 5px;">21</td><td style="padding: 0 5px;">20</td><td style="padding: 0 5px;">19</td><td style="padding: 0 5px;">18</td><td style="padding: 0 5px;">17</td><td style="padding: 0 5px;">16</td><td style="padding: 0 5px;">15</td><td style="padding: 0 5px;">14</td><td style="padding: 0 5px;">13</td><td style="padding: 0 5px;">12</td><td style="padding: 0 5px;">11</td><td style="padding: 0 5px;">10</td><td style="padding: 0 5px;">9</td><td style="padding: 0 5px;">8</td><td style="padding: 0 5px;">7</td><td style="padding: 0 5px;">6</td><td style="padding: 0 5px;">5</td><td style="padding: 0 5px;">4</td><td style="padding: 0 5px;">3</td><td style="padding: 0 5px;">2</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td> </tr> <tr> <td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td> </tr> </table> <p style="text-align: center; margin-top: 10px;"> <span style="margin-right: 100px;">ACC/ DREG Write</span> <span>(Page) (Pointer) direct address</span> </p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	0	0	0	0	1	0	0			0	0									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
0	1	0	0	0	0	1	0	0			0	0																																	
<b>CCR</b>  <b>OVFP</b>	<p>CCR C: Set if a mantissa of ACCA is \$0000 before instruction execution; cleared otherwise.</p> <p>N: Set if ACCA is negative after instruction execution.</p> <p>Z: Set if a mantissa of ACCA is \$0000 after instruction execution.</p> <p>OVFP OVFP bit of the CTR must be set to 1 beforehand.</p>																																												





## II. Direct addressing mode

Assembler syntax	[<label>] ΔFNEGBΔ <operand ①>[Δ <comment>]																																																																		
Example	FNEGB Δ <sub>A</sub> , 2, 14 ②     ③																																																																		
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; margin: 10px auto; width: fit-content;">             -ACCB(20 bits) → ACCB(20 bits)           </div> <p style="margin-top: 10px;">A two's complement of ACCB is stored in the ACCB.</p> <p>An overflow occurs when the mantissa and exponent of ACCB are \$8000 and \$7 respectively before this instruction is executed.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCB.</li> </ul>																																																																		
Instruction code	<table style="margin: auto; border-collapse: collapse;"> <tr> <td style="text-align: center;">21</td><td style="text-align: center;">20</td><td style="text-align: center;">19</td><td style="text-align: center;">18</td><td style="text-align: center;">17</td><td style="text-align: center;">16</td><td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td> </tr> <tr> <td colspan="12" style="text-align: center;">ACC/ DREG Write</td> <td colspan="4" style="text-align: center;">(Page)</td> <td colspan="6" style="text-align: center;">(Pointer) direct address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	0	0	0	0	1	0	1				0	0									ACC/ DREG Write												(Page)				(Pointer) direct address					
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																														
0	1	0	0	0	0	1	0	1				0	0																																																						
ACC/ DREG Write												(Page)				(Pointer) direct address																																																			
CCR	CCR C: Set if a mantissa of ACCB is \$0000 before instruction execution; cleared otherwise.																																																																		
OVFP	N: Set if ACCB is negative after instruction execution. Z: Set if a mantissa of ACCB is \$0000 after instruction execution.																																																																		
	OVFP OVFP bit of the CTR must be set to 1 beforehand.																																																																		

# NEGA

## I. Pointer addressing mode

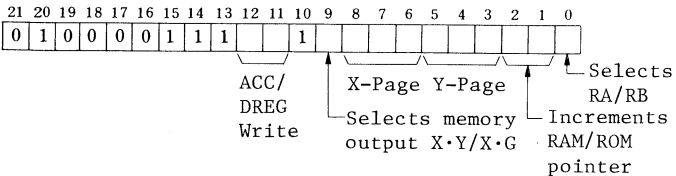
Assembler syntax	[<label>] ΔNEGAΔ <operand ©>[Δ <comment>]
Example	NEGA Δ $\overline{EE}$ , $\overline{XG}(1, 3)$ , $\overline{RA}$ ②                  ③                  ④
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <p>-ACCA(16 bits) → ACCA(16 bits)</p> </div> <p>A two's complement of ACCA is stored in the ACCA.</p> <p>An overflow occurs when the contents of ACCA is \$8000 before this instruction is executed.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ©'. In 'Operand ©', "ACC" means ACCA.</li> </ul>
Instruction code	
CCR OVFP	<p>CCR C: Set if ACCA is \$0000 before instruction execution; cleared otherwise. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is \$0000 after instruction execution.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit. 0: no overflow protection 1: overflow protection</p>

## II. Direct addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔNEGAΔ &lt;operand ①&gt;[Δ &lt;comment&gt;]</p>																																																																		
<p>Example</p>	<p>NEGA ΔEE, 0,00                    ②    ③</p>																																																																		
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation              Fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <p>-ACCA(16 bits) → ACCA(16 bits)</p> </div> <p>A two's complement of ACCA is stored in the ACCA.</p> <p>An overflow occurs when the contents of ACCA is \$8000 before this instruction is executed.</p> <ul style="list-style-type: none"> <li>● Operand              For details of operand, see 5.2.1 'Operand ①'.              In 'Operand ①', "ACC" meand ACCA.</li> </ul>																																																																		
<p>Instruction code</p>	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="text-align: center;">21</td><td style="text-align: center;">20</td><td style="text-align: center;">19</td><td style="text-align: center;">18</td><td style="text-align: center;">17</td><td style="text-align: center;">16</td><td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td> </tr> <tr> <td colspan="12" style="text-align: center;">ACC/ DREG Write</td> <td colspan="4" style="text-align: center;">(Page)</td> <td colspan="6" style="text-align: center;">(Pointer) direct address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ACC/ DREG Write												(Page)				(Pointer) direct address					
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																														
0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																														
ACC/ DREG Write												(Page)				(Pointer) direct address																																																			
<p>CCR  OVFP</p>	<p>CCR C: Set if ACCA is \$0000 before instruction execution; cleared otherwise.          N: Set if ACCA is negative after instruction execution.          Z: Set if ACCA is \$0000 after instruction execution.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit.          0: no overflow protection 1: overflow protection.</p>																																																																		

# NEGB

## I. Pointer addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔNEGBΔ &lt;operand ③&gt;[Δ &lt;comment&gt;]</p>
<p>Example</p>	<p>NEGB Δ <u>EE</u>, <u>XG(1,3)</u>, <u>RA</u>                    ②          ③          ④</p>
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation Fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <p>-ACCB(16 bits) → ACCB(16 bits)</p> </div> <p>A two's complement of ACCB is stored in the ACCB.</p> <p>An overflow occurs when the content of ACCB is \$8000 before this instruction is executed.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ③'. In 'Operand ③', "ACC" means ACCB.</li> </ul>
<p>Instruction code</p>	
<p>CCR OVFP</p>	<p>CCR C: Set if ACCB is \$0000 before instruction execution; cleared otherwise.          N: Set if ACCB is negative after instruction execution.          Z: Set if ACCB is \$0000 after instruction execution.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit.          0: no overflow protection    1: overflow protection</p>

## II. Direct addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔNEGBΔ &lt;operand <math>\textcircled{D}</math>&gt;[Δ &lt;comment&gt;]</p>																																																																	
<p>Example</p>	<p>NEGB <math>\triangle A, \underset{\textcircled{2}}{2}, \underset{\textcircled{3}}{14}</math></p>																																																																	
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation Fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <p>-ACCB(16 bits) → ACCB(16 bits)</p> </div> <p>A two's complement of ACCB is stored in the ACCB.</p> <p>An overflow occurs when the contents of ACCB is \$8000 before the instruction is executed.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand <math>\textcircled{D}</math>'. In 'Operand <math>\textcircled{D}</math>', "ACC" means ACCB.</li> </ul>																																																																	
<p>Instruction code</p>	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="padding: 0 5px;">21</td><td style="padding: 0 5px;">20</td><td style="padding: 0 5px;">19</td><td style="padding: 0 5px;">18</td><td style="padding: 0 5px;">17</td><td style="padding: 0 5px;">16</td><td style="padding: 0 5px;">15</td><td style="padding: 0 5px;">14</td><td style="padding: 0 5px;">13</td><td style="padding: 0 5px;">12</td><td style="padding: 0 5px;">11</td><td style="padding: 0 5px;">10</td><td style="padding: 0 5px;">9</td><td style="padding: 0 5px;">8</td><td style="padding: 0 5px;">7</td><td style="padding: 0 5px;">6</td><td style="padding: 0 5px;">5</td><td style="padding: 0 5px;">4</td><td style="padding: 0 5px;">3</td><td style="padding: 0 5px;">2</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td> </tr> <tr> <td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td> </tr> <tr> <td colspan="12" style="text-align: center;">ACC/ DREG Write</td> <td colspan="3" style="text-align: center;">(Page)</td> <td colspan="6" style="text-align: center;">(Pointer) direct address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	0	0	0	0	1	1	1			0	0										ACC/ DREG Write												(Page)			(Pointer) direct address					
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																													
0	1	0	0	0	0	1	1	1			0	0																																																						
ACC/ DREG Write												(Page)			(Pointer) direct address																																																			
<p>CCR  OVFP</p>	<p>CCR C: Set if ACCB is \$0000 before instruction execution; cleared otherwise. N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is \$0000 after instruction execution.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit. 0: no overflow protection 1: overflow protection</p>																																																																	

# INCA

## I. Pointer addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔINCAΔ &lt;operand ©&gt;[Δ &lt;comment&gt;]</p>
<p>Example</p>	<p>INCA Δ <u>EE</u>, <u>XY(0,0)</u>, <u>RA</u>                    (2)      (3)      (4)</p>
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation Fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>(ACCA)+1 → ACCA          (16 bits) (16 bits)</p> </div> <p>The contents of ACCA is dealt in the binary representation.          The value of the exponent is undefined.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ©'. In 'Operand ©', "ACC" means ACCA.</li> </ul>
<p>Instruction code</p>	<p style="text-align: center;">         21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0          0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0     </p> <p style="text-align: center;">         ACC/ DREG Write     </p> <p style="text-align: center;">         X-Page     </p> <p style="text-align: center;">         Y-Page Selects memory output X·Y/X·G     </p> <p style="text-align: center;">         Selects RAM/ROM pointer     </p> <p style="text-align: center;">         RA/RB Increments     </p>
<p>CCR</p> <p>OVFP</p>	<p>CCR C: Set if ACCA is \$FFFF before instruction execution; cleared otherwise.</p> <p>N: Set if ACCA is negative after instruction execution.</p> <p>Z: Set if ACCA is \$FFFF after instruction execution.</p>

## II. Direct addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔINCAΔ &lt;operand ①&gt;[Δ &lt;comment&gt;]</p>																																												
<p>Example</p>	<p>INCA Δ <u>EE</u>, <u>0,00</u>                    ②      ③</p>																																												
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation              Increment based on fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; margin: 10px auto; width: fit-content;"> <p>(ACCA)+1 → ACCA              (16 bits) (16 bits)</p> </div> <p>The contents of ACCA is dealt in the binary representation.              The value of the exponent is undefined.</p> <ul style="list-style-type: none"> <li>● Operand              For details of operand, see 5.2.1 'Operand ①'.              In 'Operand ①', "ACC" means ACCA.</li> </ul>																																												
<p>Instruction code</p>	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="text-align: center;">21</td><td style="text-align: center;">20</td><td style="text-align: center;">19</td><td style="text-align: center;">18</td><td style="text-align: center;">17</td><td style="text-align: center;">16</td><td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td> </tr> </table> <p style="text-align: center; margin-left: 100px;">ACC/ DREG Write</p> <p style="text-align: center; margin-left: 150px;">(Page) (Pointer) direct address</p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	0	0	1	0	0	1	0			0	0									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
0	1	0	0	1	0	0	1	0			0	0																																	
<p>CCR  OVFP</p>	<p>CCR C: Set if ACCA is \$FFFF before instruction execution;              cleared otherwise.              N: Set if ACCA is negative after instruction execution.              Z: Set if ACCA is \$FFFF after instruction execution.</p>																																												

# INCB

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔINCBΔ <operand ©>[Δ <comment>]
Example	$\text{INCB } \triangle \text{EE}, \text{XY}(\underline{0}, \underline{0}), \underline{\text{RA}}$ <p style="text-align: center;"> <span style="margin-right: 100px;">(2)</span> <span style="margin-right: 100px;">(3)</span> <span>(4)</span> </p>
Operation	<ul style="list-style-type: none"> <li>● ALU operation Increment based on fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <math display="block">\begin{matrix} (\text{ACCB})+1 &amp; \rightarrow &amp; \text{ACCB} \\ \text{(16 bits)} &amp; &amp; \text{(16 bits)} \end{matrix}</math> </div> <p>The contents of ACCB is dealt in the binary representation. The value of the exponent is undefined.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ©'. In 'Operand ©', "ACC" means ACCB.</li> </ul>
Instruction code	
CCR OVFP	<p>CCR C: Set if ACCB is \$FFFF before instruction execution; cleared otherwise.</p> <p>N: Set if ACCB is negative after instruction execution.</p> <p>Z: Set if ACCB is \$FFFF after instruction execution.</p>



## II. Direct addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔINCBΔ &lt;operand ①&gt;[Δ &lt;comment&gt;]</p>																																												
<p>Example</p>	<p>INCB Δ <u>EE</u>, <u>0,00</u>  <small>(2) (3)</small></p>																																												
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation Increment based on fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <p>(ACCB)+1 → ACCB          (16 bits) (16 bits)</p> </div> <p>The contents of ACCB is dealt in the binary representation. The value of the exponent is undefined.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCB.</li> </ul>																																												
<p>Instruction code</p>	<table style="border-collapse: collapse; margin: 0 auto;"> <tr> <td style="padding: 0 5px;">21</td><td style="padding: 0 5px;">20</td><td style="padding: 0 5px;">19</td><td style="padding: 0 5px;">18</td><td style="padding: 0 5px;">17</td><td style="padding: 0 5px;">16</td><td style="padding: 0 5px;">15</td><td style="padding: 0 5px;">14</td><td style="padding: 0 5px;">13</td><td style="padding: 0 5px;">12</td><td style="padding: 0 5px;">11</td><td style="padding: 0 5px;">10</td><td style="padding: 0 5px;">9</td><td style="padding: 0 5px;">8</td><td style="padding: 0 5px;">7</td><td style="padding: 0 5px;">6</td><td style="padding: 0 5px;">5</td><td style="padding: 0 5px;">4</td><td style="padding: 0 5px;">3</td><td style="padding: 0 5px;">2</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td> </tr> <tr> <td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td> </tr> </table> <p style="text-align: center; margin-top: 5px;"> <span style="margin-right: 100px;">ACC/ DREG Write</span> <span>(Page) (Pointer) direct address</span> </p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	0	0	1	0	0	1	1			0	0									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
0	1	0	0	1	0	0	1	1			0	0																																	
<p>CCR OVFP</p>	<p>CCR C: Set if ACCB is \$FFFF before instruction execution; cleared otherwise.          N: Set if ACCB is negative after instruction execution.          Z: Set if ACCB is \$FFFF after instruction execution.</p>																																												

# DECA

## I. Pointer addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔDECAΔ &lt;operand ©&gt;[Δ &lt;comment&gt;]</p>
<p>Example</p>	<p>DECA Δ<math>\overline{EE}</math>, <math>\overline{XY}(0,0)</math>, <math>\overline{RA}</math>                    ②          ③          ④</p>
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation Decrement based on fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>(ACCA)-1     →   ACCA          (16 bits)       (16 bits)</p> </div> <p>The contents of ACCA is dealt in the binary representation.          The value of the exponent is undefined.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ©'. In 'Operand ©', "ACC" means ACCA.</li> </ul>
<p>Instruction code</p>	
<p>CCR OVFP</p>	<p>CCR C: Set if ACCA is not \$0000 before instruction execution; cleared otherwise.          N: Set if ACCA is negative after instruction execution.          Z: Set if ACCA is \$0000 after instruction execution.</p>

## II. Direct addressing mode

Assembler syntax	[<label>] ΔDECAΔ <operand ①>[Δ <comment>]																																												
Example	DECA Δ <u>EE</u> , <u>0,00</u> ②     ③																																												
Operation	<ul style="list-style-type: none"> <li>● ALU operation Decrement based on fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <math display="block">\begin{array}{ccc} (\text{ACCA})-1 &amp; \rightarrow &amp; \text{ACCA} \\ \text{(16 bits)} &amp; &amp; \text{(16 bits)} \end{array}</math> </div> <p>The contents of ACCA is dealt in the binary representation. The value of the exponent is undefined.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCA.</li> </ul>																																												
Instruction code	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="padding: 0 5px;">21</td><td style="padding: 0 5px;">20</td><td style="padding: 0 5px;">19</td><td style="padding: 0 5px;">18</td><td style="padding: 0 5px;">17</td><td style="padding: 0 5px;">16</td><td style="padding: 0 5px;">15</td><td style="padding: 0 5px;">14</td><td style="padding: 0 5px;">13</td><td style="padding: 0 5px;">12</td><td style="padding: 0 5px;">11</td><td style="padding: 0 5px;">10</td><td style="padding: 0 5px;">9</td><td style="padding: 0 5px;">8</td><td style="padding: 0 5px;">7</td><td style="padding: 0 5px;">6</td><td style="padding: 0 5px;">5</td><td style="padding: 0 5px;">4</td><td style="padding: 0 5px;">3</td><td style="padding: 0 5px;">2</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td> </tr> <tr> <td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td> </tr> </table> <div style="margin-left: 100px; margin-top: 5px;"> <span style="margin-right: 100px;">ACC/ DREG Write</span> <span>(Page) (Pointer) direct address</span> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	1	1	1	0	1	0				0	0									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
0	0	1	1	1	0	1	0				0	0																																	
CCR  OVFP	CCR C: Set if ACCA is not \$0000 before instruction execution; cleared otherwise. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is \$0000 after instruction execution.																																												

# DECB

## I. Pointer addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔDECBΔ &lt;operand ③&gt;[Δ &lt;comment&gt;]</p>
<p>Example</p>	<p>DECB ΔEE, XY(0,0), RA                    ②      ③      ④</p>
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation Decrement based on fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <p>(ACCB)-1    →    ACCB          (16 bits)       (16 bits)</p> </div> <p>The contents of ACCB is dealt in the binary representation. The value of the exponent is undefined.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ③'. In 'Operand ③', "ACC" means ACCB.</li> </ul>
<p>Instruction code</p>	<p>21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</p> <p>0 0 1 1 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0</p> <p>ACC/DREG Write    X-Page Y-Page    Selects RA/RB    Selects memory output X·Y/X·G    Increments RAM/ROM pointer</p>
<p>CCR OVFP</p>	<p>CCR C: Set if ACCB is not \$0000 before instruction execution; cleared otherwise.          N: Set if ACCB is negative after instruction execution.          Z: Set if ACCB is \$0000 after instruction execution.</p>

## II. Direct addressing mode

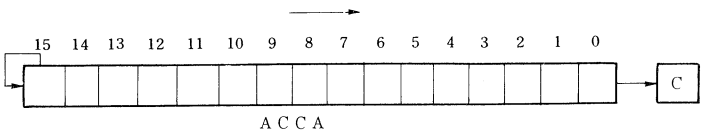
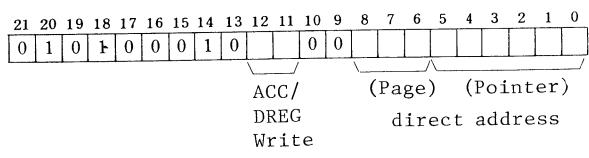
<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔDECBA &lt;operand ①&gt;[Δ &lt;comment&gt;]</p>																																																																			
<p>Example</p>	<p>DECB Δ <u>EE</u>, <u>0,00</u>                    ②      ③</p>																																																																			
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation              Decrement based on fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <p>(ACCB)-1      →      ACCB              (16 bits)      (16 bits)</p> </div> <p>The contents of ACCB is dealt in the binary representation.              The value of the exponent is undefined.</p> <ul style="list-style-type: none"> <li>● Operand              For details of operand, see 5.2.1 'Operand ①'.              In 'Operand ①', "ACC" means ACCB.</li> </ul>																																																																			
<p>Instruction code</p>	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="padding: 0 5px;">21</td><td style="padding: 0 5px;">20</td><td style="padding: 0 5px;">19</td><td style="padding: 0 5px;">18</td><td style="padding: 0 5px;">17</td><td style="padding: 0 5px;">16</td><td style="padding: 0 5px;">15</td><td style="padding: 0 5px;">14</td><td style="padding: 0 5px;">13</td><td style="padding: 0 5px;">12</td><td style="padding: 0 5px;">11</td><td style="padding: 0 5px;">10</td><td style="padding: 0 5px;">9</td><td style="padding: 0 5px;">8</td><td style="padding: 0 5px;">7</td><td style="padding: 0 5px;">6</td><td style="padding: 0 5px;">5</td><td style="padding: 0 5px;">4</td><td style="padding: 0 5px;">3</td><td style="padding: 0 5px;">2</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td> </tr> <tr> <td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td> </tr> <tr> <td colspan="12" style="text-align: center;">ACC/ DREG Write</td> <td colspan="11" style="text-align: center;">(Page) (Pointer) direct address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	1	1	1	0	0	1	1			0	0										ACC/ DREG Write												(Page) (Pointer) direct address										
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																															
0	0	1	1	1	0	0	1	1			0	0																																																								
ACC/ DREG Write												(Page) (Pointer) direct address																																																								
<p>CCR  OVFP</p>	<p>CCR C: Set if ACCB is not \$0000 before instruction execution;              cleared otherwise.              N: Set if ACCB is negative after instruction execution.              Z: Set if ACCB is \$0000 after instruction execution.</p>																																																																			

# SRA

## I. Pointer addressing mode

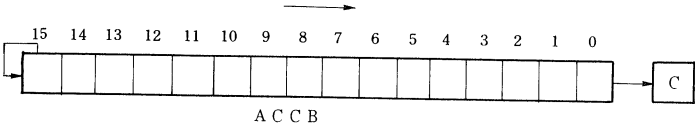
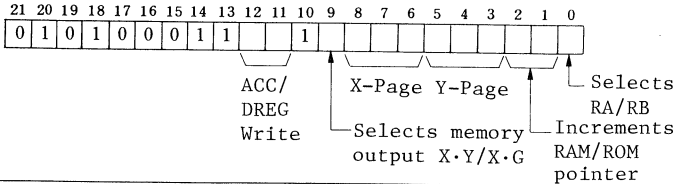
<p>Assembler syntax</p>	<p>[&lt;label&gt;]ΔSRAΔ &lt;operand ②&gt;[Δ &lt;comment&gt;]</p>
<p>Example</p>	<p>SRA Δ <u>A</u>, <u>XY(2,4)</u>, <u>RA</u></p> <p style="text-align: center;">②            ③            ④</p>
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation The contents of ACCA is shifted right 1 bit.</li> </ul> <div style="text-align: center;"> <p style="text-align: center;">ACCA</p> </div> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ②'. In 'Operand ②', "ACC" means ACCA.</li> <li>● Example of n bits shift</li> </ul> <pre> LIRC 6           ; Repeat count n=(6+1) RPTA EE,0,00    ; Repeat instruction SRA  EE,XY(0,0),RA ; Shift right 7 bits     </pre>
<p>Instruction code</p>	<div style="text-align: center;"> </div>
<p>CCR OVFP</p>	<p>CCR C: Set if the value of LSB is 1 before instruction execution. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is \$0000 after instruction execution.</p>

## II. Direct addressing mode

Assembler syntax	[<label>] ΔSRAΔ <operand ①>[Δ <comment>]
Example	SRA△EE, 0, 00 ②           ③
Operation	<p>● ALU operation The contents of ACCA is shifted right 1 bit.</p>  <p>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCA.</p>
Instruction code	
CCR OVFP	<p>CCR C: Set if the value of LSB is 1 before instruction execution. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is \$0000 after instruction execution.</p>

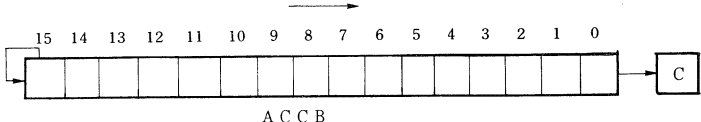
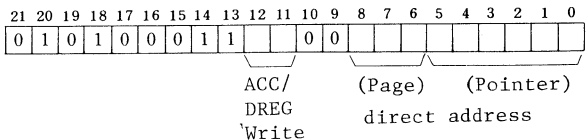
# SRB

## I. Pointer addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔSRBΔ &lt;operand ③&gt;[Δ &lt;comment&gt;]</p>
<p>Example</p>	<p>SRBΔ<sub>②</sub>A, XY(2,4), RA<sub>④</sub></p>
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation The contents of ACCB is shifted right 1 bit.</li> </ul>  <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ③'. In 'Operand ③', "ACC" means ACCB.</li> </ul>
<p>Instruction code</p>	
<p>CCR OVFP</p>	<p>CCR C: Set if the value of LSB is 1 before instruction execution. N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is \$0000 after instruction execution.</p>



## II. Direct addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔSRBΔ &lt;operand (D)&gt;[Δ &lt;comment&gt;]</p>
<p>Example</p>	<p>SRBΔEE, 0,00                    (2)  (3)</p>
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation              The contents of ACCB is shifted right 1 bit.</li> </ul>  <p style="text-align: center;">ACCB</p> <ul style="list-style-type: none"> <li>● Operand              For details of operand, see 5.2.1 'Operand (D)'.              In 'Operand (D)', "ACC" means ACCB.</li> </ul>
<p>Instruction code</p>	 <p style="text-align: center;">         ACC/          DREG          Write     </p> <p style="text-align: center;">         (Page) (Pointer)          direct address     </p>
<p>CCR OVFP</p>	<p>CCR C: Set if the value of LSB is 1 before instruction execution.          N: Set if ACCB is negative after instruction execution.          Z: Set if ACCB is \$0000 after instruction execution.</p>

# SLA

## I. Pointer addressing mode

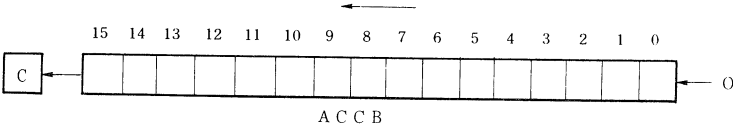
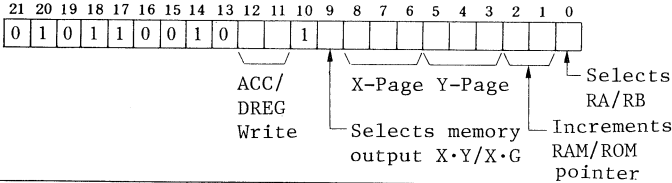
<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔSLAΔ &lt;operand (C)&gt;[Δ &lt;comment&gt;]</p>
<p>Example</p>	<p>SLA ΔA, XY(2,4), RA                    ②      ③      ④</p>
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation The contents of ACCA is shifted left 1 bit.</li> </ul> <div style="text-align: center;"> <p style="text-align: center;">ACCA</p> </div> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand (C)'. In 'Operand (C)', "ACC" means ACCA.</li> <li>● Example for n bit shift</li> </ul> <pre> LIRC 4           ; Number of repeat n=(4+1) RPTA EE,0,00    ; Repeat instruction SLA EE,XY(0,0),RA ; Shift left 5 bits     </pre>
<p>Instruction code</p>	<div style="text-align: center;"> </div>
<p>CCR OVFP</p>	<p>CCR C: Set if the value of MSB is 1 before instruction execution.          N: Set if ACCA is negative after instruction execution.          Z: Set if ACCA is \$0000 after instruction execution.</p>

## II. Direct addressing mode

Assembler syntax	[<label>] ΔSLAΔ <operand ①>[Δ <comment>]																																																																																																																			
Example	$SLA \triangle EE, 0, 00$ <p style="text-align: center;">②      ③</p>																																																																																																																			
Operation	<ul style="list-style-type: none"> <li>● ALU operation The contents of ACCA is shifted left 1 bit.</li> </ul> <div style="text-align: center; margin: 10px 0;"> <p style="text-align: center;">ACCA</p> </div> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCA.</li> </ul>																																																																																																																			
Instruction code	<div style="text-align: center; margin-bottom: 10px;"> <table border="1" style="border-collapse: collapse; width: 100%; text-align: center;"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> </div> <div style="text-align: center;"> <table style="width: 100%; border: none;"> <tr> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> <td style="border: none; width: 10%;"></td> </tr> <tr> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> </tr> <tr> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> <td style="border: none;"></td> </tr> </table> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	0	1	1	0	0	1	0			0	0																																																																																
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																															
0	1	0	1	1	0	0	1	0			0	0																																																																																																								
CCR OVFP	<p>CCR C: Set if the value of MSB is 1 before instruction execution. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is \$0000 after instruction execution.</p>																																																																																																																			

# SLB

## I. Pointer addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔSLBΔ &lt;operand ©&gt;[Δ &lt;comment&gt;]</p>
<p>Example</p>	<p>SLB Δ <u>A</u> , <u>XY (2,4)</u> , <u>RA</u>                    ②          ③          ④</p>
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation              The contents of ACCB is shifted left 1 bit.</li> </ul>  <ul style="list-style-type: none"> <li>● Operand              For details of operand, see 5.2.1 'Operand ©'.              In 'Operand ©', "ACC" means ACCB.</li> </ul>
<p>Instruction code</p>	
<p>CCR OVFP</p>	<p>CCR C: Set if the value of MSB is 1 before instruction execution.              N: Set if ACCB is negative after instruction execution.              Z: Set if ACCB is \$0000 after instruction execution.</p>

## II. Direct addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔSLBΔ &lt;operand ①&gt;[Δ &lt;comment&gt;]</p>																																												
<p>Example</p>	<p>SLB Δ <u>EE</u> , <u>0,00</u>  <small>②            ③</small></p>																																												
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation The contents of ACCB is shifted left 1 bit.</li> </ul> <div data-bbox="337 534 1089 668" style="text-align: center;"> </div> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCB.</li> </ul>																																												
<p>Instruction code</p>	<div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <p style="margin-left: 100px;">ACC/ DREG Write</p> <p style="margin-left: 150px;">(Page) (Pointer) direct address</p> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	0	1	1	0	0	1	0			0	0									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
0	1	0	1	1	0	0	1	0			0	0																																	
<p>CCR OVFP</p>	<p>CCR C: Set if the value of MSB is 1 before instruction execution.          N: Set if ACCB is negative after instruction execution.          Z: Set if ACCB is \$0000 after instruction execution.</p>																																												

# FLTA

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔFLTAΔ <operand ©>[Δ <comment>]
Example	FLTA△EE,XY(2,7),RA ②      ③      ④
Operation	<p>● ALU operation Transformation of data representation (fixed point → floating point)</p> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;">             ACCA(16 bits, fixed point) → ACCA(20 bits, floating point)           </div> <p>This instruction transforms the fixed point data in ACCA to the floating point data, using an exponent part (4 bits) of Y-Bus output data (16 bits) specified by operand as a scaling constant.</p> <div style="text-align: center;"> <p>Fixed point data (16 bits)    Scaling constant (Y-Bus data)</p> <p>ACCA 15 0    ROM 15 4 3 0 RAM    don't care GR    exp.</p> <p>mant.    exp.</p> <p>ALU*    * with normalization</p> <p>15 0 3 0 mant.    exp.</p> <p>Floating point data (20 bits)</p> </div> <p>(note) A mantissa (12 bits) of Y-Bus data can be any value.</p> <p>● Operand For details of operand, see 5.2.1 'Operand ©'. In 'Operand ©', "ACC" means ACCA.</p>
Instruction code	
CCR OVFP	CCR C: 0 N: Set if ACCA is negative. Z: Set if ACCA is \$0000.

## II. Direct addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔFLTAΔ &lt;operand ①&gt;[Δ &lt;comment&gt;]</p>																																												
<p>Example</p>	<p>FLTA△EE, 6, 30                    ②      ③</p>																																												
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation Transformation of data representation (fixed point → floating point)</li> </ul> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>ACCA(16 bits, fixed point) → ACCA(20 bits, floating point)</p> </div> <p>This instruction transforms the fixed point data in ACCA to the floating point data, using an exponent part (4 bits) of Y-Bus output data (16 bits) specified by operand as a scaling constant.</p> <div style="text-align: center;"> </div> <p>(note) A mantissa (12 bits) of Y-Bus data can be any value.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCA.</li> </ul>																																												
<p>Instruction code</p>	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <p style="text-align: center;"> <span style="margin-right: 100px;">ACC/ DREG Write</span> <span>(Page) (Pointer) direct address</span> </p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	1	1	1	0	1	0	0				0	0								
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
0	1	1	1	1	0	1	0	0				0	0																																
<p>CCR  OVFP</p>	<p>CCR C: 0          N: Set if ACCA is negative.          Z: Set if ACCA is \$0000.</p>																																												





## II. Direct addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔFLTBΔ &lt;operand ①&gt;[Δ &lt;comment&gt;]</p>																																																																		
<p>Example</p>	<p>FLTB ΔEE, 1, 44  <small>②      ③</small></p>																																																																		
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation Transformation of data representation (fixed point → floating point)</li> </ul> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>ACCB(16 bits, fixed point) → ACCB(20 bits, floating point)</p> </div> <p>This instruction transforms the fixed point data in ACCB to the floating point data, using an exponent part (4 bits) of Y-Bus output data (16 bits) specified by operand as a scaling constant.</p> <div style="text-align: center;"> <p>Fixed point data (16 bits)    Scaling constant (Y-Bus data)</p> <p style="text-align: right;">* with normalization</p> </div> <p>(note) A mantissa (12 bits) of Y-Bus data can be any value.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCB.</li> </ul>																																																																		
<p>Instruction code</p>	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="padding: 0 5px;">21</td><td style="padding: 0 5px;">20</td><td style="padding: 0 5px;">19</td><td style="padding: 0 5px;">18</td><td style="padding: 0 5px;">17</td><td style="padding: 0 5px;">16</td><td style="padding: 0 5px;">15</td><td style="padding: 0 5px;">14</td><td style="padding: 0 5px;">13</td><td style="padding: 0 5px;">12</td><td style="padding: 0 5px;">11</td><td style="padding: 0 5px;">10</td><td style="padding: 0 5px;">9</td><td style="padding: 0 5px;">8</td><td style="padding: 0 5px;">7</td><td style="padding: 0 5px;">6</td><td style="padding: 0 5px;">5</td><td style="padding: 0 5px;">4</td><td style="padding: 0 5px;">3</td><td style="padding: 0 5px;">2</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td> </tr> <tr> <td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td> </tr> <tr> <td colspan="12" style="text-align: center;">ACC/ DREG Write</td> <td colspan="10" style="text-align: center;">(Page) (Pointer) direct address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	1	1	1	0	1	0	1			0	0										ACC/ DREG Write												(Page) (Pointer) direct address									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																														
0	1	1	1	1	0	1	0	1			0	0																																																							
ACC/ DREG Write												(Page) (Pointer) direct address																																																							
<p>CCR  OVFP</p>	<p>CCR C: 0 N: Set if ACCB is negative. Z: Set if ACCB is \$0000.</p>																																																																		

# FIXA

## I. Pointer addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔFIXAΔ &lt;operand © &gt;[Δ &lt;comment&gt;]</p>
<p>Example</p>	<p>FIXA Δ EE , XY ( 2 , 7 ) , RA                    ②      ③      ④</p>
<p>Operation</p>	<p>● ALU operation          Transformation of data representation          (floating point → fixed point)</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <p>ACCA(20 bits, floating point) → ACCA(16 bits, fixed point)</p> </div> <p>This instruction transforms the floating point data in ACCA to the fixed point data using an exponent part (4 bits) of Y-Bus output data (16 bits) specified by operand as a scaling constant.</p> <p>Floating point data (20 bits)      Scaling constant (Y-Bus data)</p> <p style="text-align: center;">* mantissa part is shifted right (1-exp.) bits.</p> <p style="text-align: center;">Fixed point data (16 bits)</p> <p>(note) A mantissa (12 bits) of Y-Bus data must be all '0'.          If an overflow occurs, the mantissa is fixed to positive/negative maximum value.</p> <p>● Operand          For details of operand, see 5.2.1 'Operand ©'.          In 'Operand ©', "ACC" means ACCA.</p>
<p>Instruction code</p>	
<p>CCR OVFP</p>	<p>CCR C: 0          N: Set if ACCA is negative after instruction execution.          Z: Set if ACCA is \$0000 after instruction execution.</p>





II. Direct addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔFIXBΔ &lt;operand ①&gt;[Δ &lt;comment&gt;]</p>
<p>Example</p>	<p>FIXB Δ EE,6,30                    ②   ③</p>
<p>Operation</p>	<p>● ALU operation          Transformation of data representation          (floating point → fixed point)</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <p>ACCB(20 bits, floating point) → ACCB(16 bits, fixed point)</p> </div> <p>This instruction transforms the floating point data in ACCB to the fixed point data using an exponent part (4 bits) of Y-Bus output data (16 bits) specified by operand as a scaling constant.</p> <p>Floating point data (16 bits)    Scaling constant (Y-Bus data)</p> <div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: center;"> <p>15                    03 0</p> <div style="border: 1px solid black; padding: 2px; width: 150px; margin: 0 auto;"> <span style="float: left;">mant.</span> <span style="float: right;">exp.</span> </div> </div> <div style="text-align: center;"> <p>ROM 15                    43 0</p> <p>RAM \$000                    ℓ</p> </div> </div> <div style="text-align: center; margin: 10px 0;"> <div style="border: 1px solid black; padding: 5px; width: 100px; margin: 0 auto;">             ALU*         </div> <p>* mantissa part is shifted right (ℓ-exp.) bits.</p> </div> <div style="text-align: center;"> <p>15                    0</p> <div style="border: 1px solid black; padding: 2px; width: 150px; margin: 0 auto;">             ACCB         </div> <p>Fixed point data (16 bits)</p> </div>
<p>Instruction code</p>	<div style="text-align: center;"> <p>21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</p> <div style="border: 1px solid black; padding: 2px; width: 100%; margin: 0 auto;"> <span style="float: left;">0</span> <span style="float: left;">0</span> <span style="float: left;">0</span> <span style="float: left;">1</span> <span style="float: left;">0</span> <span style="float: left;">0</span> <span style="float: left;">1</span> <span style="float: left;">0</span> <span style="float: left;">1</span> <span style="float: left;">0</span> <span style="float: left;">1</span> <span style="float: left;">0</span> <span style="float: left;">0</span> <span style="float: left;">0</span> <span style="float: left;">0</span> <span style="float: left;">0</span> <span style="float: left;">0</span> <span style="float: left;">0</span> <span style="float: left;">0</span> <span style="float: left;">0</span> <span style="float: left;">0</span> <span style="float: left;">0</span> </div> <div style="display: flex; justify-content: space-around; margin-top: 5px;"> <div style="text-align: center;"> <p>ACC/ DREG Write</p> </div> <div style="text-align: center;"> <p>(Page) (Pointer) direct address</p> </div> </div> </div>
<p>CCR  OVFP</p>	<p>CCR C: 0          N: Set if ACCB is negative after instruction execution.          Z: Set if ACCB is \$0000 after instruction execution.</p>

# FCLRA

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔFCLRAΔ <operand ③>[Δ <comment>]																																													
Example	FCLRA Δ <u>EE</u> , <u>XY(2,7)</u> , <u>RA</u> ②      ③      ④																																													
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point arithmetic</li> </ul> <table border="1" data-bbox="337 473 895 572"> <tr> <td>Mantissa</td> <td>\$0000 → ACCA (16 bits)</td> </tr> <tr> <td>Exponent</td> <td>\$8 → ACCA ( 4 bits)</td> </tr> </table> <p>This instruction clears ACCA. ∴ ACCA = 0.0 × 2<sup>-8</sup></p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ③'. In 'Operand ③', "ACC" means ACCA.</li> </ul>	Mantissa	\$0000 → ACCA (16 bits)	Exponent	\$8 → ACCA ( 4 bits)																																									
Mantissa	\$0000 → ACCA (16 bits)																																													
Exponent	\$8 → ACCA ( 4 bits)																																													
Instruction code	<table border="1" data-bbox="308 1206 894 1258"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <p>ACC/DREG Write (bits 10-13)            X-Page Y-Page (bits 4-7)            Selects memory output X·Y/X·G (bits 8-9)            Selects RA/RB (bits 20-21)            Increments RAM/ROM pointer (bits 1-2)</p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	1	0	0	0				1										
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																									
1	1	1	1	0	1	0	0	0				1																																		
CCR OVFP	CCR C: Not affected. N: 0 Z: 1																																													

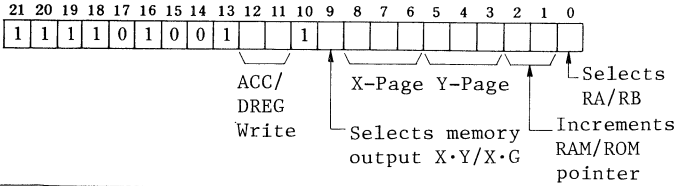
# FCLRA

## II. Direct addressing mode

Assembler syntax	[<label>] ΔFCLRAΔ <operand ①>[Δ <comment>]																																												
Example	FCLRA Δ <u>EE</u> , <u>0,00</u> ②      ③																																												
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point arithmetic</li> </ul> <table border="1" data-bbox="347 499 908 591"> <tr> <td>Mantissa</td> <td>\$0000 → ACCA (16 bits)</td> </tr> <tr> <td>Exponent</td> <td>\$8 → ACCA ( 4 bits)</td> </tr> </table> <p>This instruction clears ACCA. ∴ ACCA=0.0 × 2<sup>-8</sup></p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCA.</li> </ul>	Mantissa	\$0000 → ACCA (16 bits)	Exponent	\$8 → ACCA ( 4 bits)																																								
Mantissa	\$0000 → ACCA (16 bits)																																												
Exponent	\$8 → ACCA ( 4 bits)																																												
Instruction code	<table border="1" data-bbox="312 1229 895 1281"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <p style="text-align: center;">ACC/ DREG Write</p> <p style="text-align: center;">(Page) (Pointer) direct address</p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	1	0	0	0			0	0									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
1	1	1	1	0	1	0	0	0			0	0																																	
CCR  OVFP	CCR C: Not affected. N: 0 Z: 1																																												

# FCLRB

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔFCLRBΔ <operand ©>[Δ <comment>]				
Example	FCLRBΔ <u>EE</u> , <u>XY(2,7)</u> , <u>RA</u> ②      ③      ④				
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point arithmetic</li> </ul> <table border="1" data-bbox="325 489 884 586"> <tr> <td>Mantissa</td> <td>\$0000 → ACCB (16 bits)</td> </tr> <tr> <td>Exponent</td> <td>\$8 → ACCB ( 4 bits)</td> </tr> </table> <p data-bbox="332 618 934 649">This instruction clears ACCB. ∴ ACCB=0.0 × 2<sup>-8</sup></p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ©'. In 'Operand ©', "ACC" means ACCB.</li> </ul>	Mantissa	\$0000 → ACCB (16 bits)	Exponent	\$8 → ACCB ( 4 bits)
Mantissa	\$0000 → ACCB (16 bits)				
Exponent	\$8 → ACCB ( 4 bits)				
Instruction code					
CCR OVFP	CCR C: Not affected. N: 0 Z: 1				



## II. Direct addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔFCLRBΔ &lt;operand ①&gt;[Δ &lt;comment&gt;]</p>																																													
<p>Example</p>	<p>FCLRB Δ <u>EE,0.00</u>                            ②      ③</p>																																													
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation              Floating point arithmetic</li> </ul> <table border="1" data-bbox="333 499 892 593"> <tr> <td>Mantissa</td> <td>\$0000 → ACCB (16 bits)</td> </tr> <tr> <td>Exponent</td> <td>\$8 → ACCB ( 4 bits)</td> </tr> </table> <p>This instruction clears ACCB. ∴ ACCB = 0.0 × 2<sup>-8</sup></p> <ul style="list-style-type: none"> <li>● Operand              For details of operand, see 5.2.1 'Operand ①'.              In 'Operand ①', "ACC" means ACCB.</li> </ul>	Mantissa	\$0000 → ACCB (16 bits)	Exponent	\$8 → ACCB ( 4 bits)																																									
Mantissa	\$0000 → ACCB (16 bits)																																													
Exponent	\$8 → ACCB ( 4 bits)																																													
<p>Instruction code</p>	<table border="1" data-bbox="315 1232 900 1293"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <p style="text-align: center;"> <span style="margin-right: 100px;">ACC/ DREG Write</span> <span>(Page) (Pointer) direct address</span> </p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	1	0	0	1				0	0									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																									
1	1	1	1	0	1	0	0	1				0	0																																	
<p>CCR  OVFP</p>	<p>CCR C: Not affected.          N: 0          Z: 1</p>																																													



## II. Direct addressing mode

<b>Assembler syntax</b>	<code>[&lt;label&gt;] ΔCLRAΔ &lt;operand ①&gt;[Δ &lt;comment&gt;]</code>																																																																			
<b>Example</b>	<code>CLRA Δ <u>EE</u>, 0, 00</code> ②     ③																																																																			
<b>Operation</b>	<ul style="list-style-type: none"> <li>● ALU operation Fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <code>\$0000 → ACCA (16 bits)</code> </div> <p>This instruction clears ACCA. An exponent (4 bits) of the floating point data is undefined.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCA.</li> </ul>																																																																			
<b>Instruction code</b>	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="text-align: center;">21</td><td style="text-align: center;">20</td><td style="text-align: center;">19</td><td style="text-align: center;">18</td><td style="text-align: center;">17</td><td style="text-align: center;">16</td><td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td> </tr> <tr> <td colspan="10"></td> <td style="text-align: center;">ACC/ DREG Write</td> <td colspan="4" style="text-align: center;">(Page)</td> <td colspan="8" style="text-align: center;">(Pointer) direct address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0											ACC/ DREG Write	(Page)				(Pointer) direct address							
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																															
1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																															
										ACC/ DREG Write	(Page)				(Pointer) direct address																																																					
<b>CCR</b>  <b>OVFP</b>	<b>CCR</b> C: Not affected. N: 0 Z: 1																																																																			

# CLRB

## I. Pointer addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔCLRBA &lt;operand ©&gt;[Δ &lt;comment&gt;]</p>
<p>Example</p>	<p>CLRB Δ <u>EE</u>,<u>XY(2,7)</u>,<u>RA</u>                    ②      ③      ④</p>
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation Fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <p>\$0000 → ACCB (16 bits)</p> </div> <p>This instruction clears ACCB. An exponent (4 bits) of the floating point data is undefined.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ©'. In 'Operand ©', "ACC" means ACCB.</li> </ul>
<p>Instruction code</p>	<p>21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</p> <p>1 1 1 1 0 1 0 1 1 1 1 1 1 0 0 0 0 0 0 1 0</p> <p>ACC/DREG Write      X-Page Y-Page      Selects RA/RB      Selects memory output X·Y/X·G      Increments RAM/ROM Pointer</p>
<p>CCR OVFP</p>	<p>CCR C: Not affected. N: 0 Z: 1</p>

# CLRB

## II. Direct addressing mode

Assembler syntax	[<label>] ΔCLRBA <operand <sup>ⓓ</sup> >[Δ <comment>]																																																		
Example	CLRB Δ <u>EE</u> , <u>0, 00</u> ②      ③																																																		
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point arithmetic</li> </ul> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;">       \$0000 → ACCB (16 bits)     </div> <p>This instruction clears ACCB. An exponent (4 bits) of the floating point data is undefined.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand<sup>ⓓ</sup>'. In 'Operand<sup>ⓓ</sup>', "ACC" means ACCB.</li> </ul>																																																		
Instruction code	<div style="text-align: center;"> <table border="1" style="border-collapse: collapse; margin: auto;"> <tr> <td style="padding: 2px;">21</td><td style="padding: 2px;">20</td><td style="padding: 2px;">19</td><td style="padding: 2px;">18</td><td style="padding: 2px;">17</td><td style="padding: 2px;">16</td><td style="padding: 2px;">15</td><td style="padding: 2px;">14</td><td style="padding: 2px;">13</td><td style="padding: 2px;">12</td><td style="padding: 2px;">11</td><td style="padding: 2px;">10</td><td style="padding: 2px;">9</td><td style="padding: 2px;">8</td><td style="padding: 2px;">7</td><td style="padding: 2px;">6</td><td style="padding: 2px;">5</td><td style="padding: 2px;">4</td><td style="padding: 2px;">3</td><td style="padding: 2px;">2</td><td style="padding: 2px;">1</td><td style="padding: 2px;">0</td> </tr> <tr> <td style="padding: 2px;">1</td><td style="padding: 2px;">1</td><td style="padding: 2px;">1</td><td style="padding: 2px;">1</td><td style="padding: 2px;">0</td><td style="padding: 2px;">1</td><td style="padding: 2px;">0</td><td style="padding: 2px;">1</td><td style="padding: 2px;">1</td><td style="padding: 2px;"></td><td style="padding: 2px;"></td><td style="padding: 2px;">0</td><td style="padding: 2px;">0</td><td style="padding: 2px;"></td><td style="padding: 2px;"></td><td style="padding: 2px;"></td><td style="padding: 2px;"></td><td style="padding: 2px;"></td><td style="padding: 2px;"></td><td style="padding: 2px;"></td><td style="padding: 2px;"></td><td style="padding: 2px;"></td> </tr> </table> </div> <div style="text-align: center; margin-top: 5px;"> <table style="margin: auto;"> <tr> <td style="text-align: center;">ACC/ DREG Write</td> <td style="text-align: center;">(Page)</td> <td style="text-align: center;">(Pointer)</td> </tr> <tr> <td></td> <td colspan="2" style="text-align: center;">direct address</td> </tr> </table> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	1	0	1	1			0	0										ACC/ DREG Write	(Page)	(Pointer)		direct address	
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																														
1	1	1	1	0	1	0	1	1			0	0																																							
ACC/ DREG Write	(Page)	(Pointer)																																																	
	direct address																																																		
CCR  OVFP	CCR C: Not affected. N: 0 Z: 1																																																		

# FNOPA

## I. Pointer addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔFNOPAΔ &lt;operand ②&gt; [Δ &lt;comment&gt;]</p>
<p>Example</p>	<p>FNOPA Δ <u>A</u>, <u>XY(0, 1)</u>, <u>RA+</u></p> <p style="text-align: center;">②                    ③                    ④</p>
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation Floating point arithmetic</li>   <li>No operation is performed in the ALU. A read and write to the data ROM/RAM and multiplier input are performed in the floating point representation. ACCA is not affected.</li>   <li>● Operand For details of operand, see 5.2.1 'Operand ②'. In 'Operand ②', "ACC" means ACCA.</li> </ul>
<p>Instruction code</p>	<p style="text-align: center;">21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</p> <p style="text-align: center;">1 1 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0</p> <p style="text-align: center;">ACC/ DREG Write      X-Page Y-Page      Selects RA/RB Selects memory      Increments output X·Y/X·G      RAM/ROM pointer</p>
<p>CCR OVFP</p>	<p>CCR C: } N: } not affected Z: }</p>

# FNOPA

## II. Direct addressing mode

Assembler syntax	[<label>] ΔFNOPAΔ <operand ①>[Δ <comment>]																																																																		
Example	$FNOPA\Delta \underset{\textcircled{2}}{A}, \underset{\textcircled{3}}{2}, 40$																																																																		
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point arithmetic</li>   <li>No operation is performed in the ALU. A read and write to the data ROM/RAM and multiplier input are performed in the floating point representation. ACCA is not affected.</li>   <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCA.</li> </ul>																																																																		
Instruction code	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="text-align: center; padding: 0 5px;">21</td><td style="text-align: center; padding: 0 5px;">20</td><td style="text-align: center; padding: 0 5px;">19</td><td style="text-align: center; padding: 0 5px;">18</td><td style="text-align: center; padding: 0 5px;">17</td><td style="text-align: center; padding: 0 5px;">16</td><td style="text-align: center; padding: 0 5px;">15</td><td style="text-align: center; padding: 0 5px;">14</td><td style="text-align: center; padding: 0 5px;">13</td><td style="text-align: center; padding: 0 5px;">12</td><td style="text-align: center; padding: 0 5px;">11</td><td style="text-align: center; padding: 0 5px;">10</td><td style="text-align: center; padding: 0 5px;">9</td><td style="text-align: center; padding: 0 5px;">8</td><td style="text-align: center; padding: 0 5px;">7</td><td style="text-align: center; padding: 0 5px;">6</td><td style="text-align: center; padding: 0 5px;">5</td><td style="text-align: center; padding: 0 5px;">4</td><td style="text-align: center; padding: 0 5px;">3</td><td style="text-align: center; padding: 0 5px;">2</td><td style="text-align: center; padding: 0 5px;">1</td><td style="text-align: center; padding: 0 5px;">0</td> </tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td> </tr> <tr> <td colspan="12" style="text-align: center; padding-top: 5px;">ACC/ DREG Write</td> <td colspan="10" style="text-align: center; padding-top: 5px;">(Page) (Pointer) direct address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ACC/ DREG Write												(Page) (Pointer) direct address									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																														
1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																														
ACC/ DREG Write												(Page) (Pointer) direct address																																																							
CCR  OVFP	<table style="margin-left: auto; margin-right: auto;"> <tr> <td style="padding-right: 10px;">CCR</td> <td style="padding-right: 5px;">C:</td> <td rowspan="3" style="font-size: 2em; vertical-align: middle;">}</td> <td rowspan="3" style="padding-left: 10px;">not affected</td> </tr> <tr> <td></td> <td>N:</td> </tr> <tr> <td></td> <td>Z:</td> </tr> </table>	CCR	C:	}	not affected		N:		Z:																																																										
CCR	C:	}	not affected																																																																
	N:																																																																		
	Z:																																																																		

# FNOPB

## I. Pointer addressing mode

<p>Assembler syntax</p>	<p>[&lt;label&gt;] ΔFNOPBΔ &lt;operand ©&gt;[Δ &lt;comment&gt;]</p>
<p>Example</p>	<p>FNOPBΔ <u>A</u>, <u>XY(0, 1)</u>, <u>RA</u>                    ②      ③      ④</p>
<p>Operation</p>	<ul style="list-style-type: none"> <li>● ALU operation Floating point arithmetic</li>   <li>No operation is performed in the ALU. A read and write to the data ROM/RAM and multiplier input are performed in the floating point representation. ACCB is not affected.</li>   <li>● Operand For details of operand, see 5.2.1 'Operand ©'. In 'Operand ©', "ACC" means ACCB.</li> </ul>
<p>Instruction code</p>	
<p>CCR OVFP</p>	<p>CCR C: } N: } not affected OVFP Z: }</p>



# FNOPB

## II. Direct addressing mode

Assembler syntax	[<label>] ΔFNOPBΔ <operand ①>[Δ <comment>]																																												
Example	FNOPB Δ <u>A, 3, 40</u> ②    ③																																												
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point arithmetic</li>   <li>No operation is performed in the ALU. A read and write to the data ROM/RAM and multiplier input are performed in the floating point representation. ACCB is not affected.</li>   <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCB.</li> </ul>																																												
Instruction code	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="padding: 0 5px;">21</td><td style="padding: 0 5px;">20</td><td style="padding: 0 5px;">19</td><td style="padding: 0 5px;">18</td><td style="padding: 0 5px;">17</td><td style="padding: 0 5px;">16</td><td style="padding: 0 5px;">15</td><td style="padding: 0 5px;">14</td><td style="padding: 0 5px;">13</td><td style="padding: 0 5px;">12</td><td style="padding: 0 5px;">11</td><td style="padding: 0 5px;">10</td><td style="padding: 0 5px;">9</td><td style="padding: 0 5px;">8</td><td style="padding: 0 5px;">7</td><td style="padding: 0 5px;">6</td><td style="padding: 0 5px;">5</td><td style="padding: 0 5px;">4</td><td style="padding: 0 5px;">3</td><td style="padding: 0 5px;">2</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td> </tr> <tr> <td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">1</td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;">0</td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td><td style="padding: 0 5px;"></td> </tr> </table> <div style="margin-left: 100px;"> <span style="margin-right: 100px;">ACC/ DREG Write</span> <span>(Page) (Pointer) direct address</span> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	0	1	0	1			0	0									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
1	1	1	1	0	0	1	0	1			0	0																																	
CCR  OVFP	CCR C: } N: } not affected Z: }																																												



## II. Direct addressing mode

Assembler syntax	[<label>] ΔNOPAΔ <operandⓓ>[Δ <comment>]																																																																		
Example	$\text{NOPA} \triangle \frac{\text{A}}{\text{②}}, \frac{2, 39}{\text{③}}$																																																																		
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point arithmetic</li>   <li>No operation is performed in the ALU. A read and write to the data ROM/RAM and multiplier input are performed in the fixed point representation. ACCA is not affected.</li>   <li>● Operand For details of operand, see 5.2.1 'Operandⓓ'. In 'Operandⓓ', "ACC" means ACCA.</li> </ul>																																																																		
Instruction code	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="text-align: center; padding: 0 5px;">21</td><td style="text-align: center; padding: 0 5px;">20</td><td style="text-align: center; padding: 0 5px;">19</td><td style="text-align: center; padding: 0 5px;">18</td><td style="text-align: center; padding: 0 5px;">17</td><td style="text-align: center; padding: 0 5px;">16</td><td style="text-align: center; padding: 0 5px;">15</td><td style="text-align: center; padding: 0 5px;">14</td><td style="text-align: center; padding: 0 5px;">13</td><td style="text-align: center; padding: 0 5px;">12</td><td style="text-align: center; padding: 0 5px;">11</td><td style="text-align: center; padding: 0 5px;">10</td><td style="text-align: center; padding: 0 5px;">9</td><td style="text-align: center; padding: 0 5px;">8</td><td style="text-align: center; padding: 0 5px;">7</td><td style="text-align: center; padding: 0 5px;">6</td><td style="text-align: center; padding: 0 5px;">5</td><td style="text-align: center; padding: 0 5px;">4</td><td style="text-align: center; padding: 0 5px;">3</td><td style="text-align: center; padding: 0 5px;">2</td><td style="text-align: center; padding: 0 5px;">1</td><td style="text-align: center; padding: 0 5px;">0</td> </tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td> </tr> <tr> <td colspan="11" style="text-align: center; padding: 5px;">ACC/ DREG Write</td> <td colspan="11" style="text-align: center; padding: 5px;">(Page) (Pointer) direct address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	0	1	1	0														ACC/ DREG Write											(Page) (Pointer) direct address										
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																														
1	1	1	1	0	0	1	1	0																																																											
ACC/ DREG Write											(Page) (Pointer) direct address																																																								
CCR  OVFP	CCR C: } N: } not affected Z: }																																																																		

# NOPB

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔNOPBΔ <operand ©>[Δ <comment>]
Example	NOPB Δ <u>EE</u> , <u>XG(1, 3)</u> , <u>RA</u> ②                  ③                  ④
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point arithmetic</li>   <li>No operation is performed in the ALU. A read and write to the data ROM/RAM and multiplier input are performed in the fixed point representation. ACCB is not affected.</li>   <li>● Operand For details of operand, see 5.2.1 'Operand ©'. In 'Operand ©', "ACC" means ACCB.</li> </ul>
Instruction code	
CCR OVFP	CCR C: } N: } not affected Z: }

## II. Direct addressing mode

Assembler syntax	[<label>] ΔNOPBΔ <operand $\textcircled{D}$ >[Δ <comment>]																						
Example	NOPB Δ <u>A</u> , <u>1</u> , <u>44</u> <div style="display: flex; justify-content: space-around; width: 100px; margin-top: 5px;"> <span><math>\textcircled{2}</math></span> <span><math>\textcircled{3}</math></span> </div>																						
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point arithmetic</li>   <li>No operation is performed in the ALU. A read and write to the data ROM/RAM and multiplier input are performed in the fixed point representation. ACCB is not affected.</li>   <li>● Operand For details of operand, see 5.2.1 'Operand <math>\textcircled{D}</math>'. In 'Operand <math>\textcircled{D}</math>', "ACC" means ACCB.</li> </ul>																						
Instruction code	<div style="text-align: center; margin-bottom: 5px;"> <span style="margin-right: 5px;">21</span><span style="margin-right: 5px;">20</span><span style="margin-right: 5px;">19</span><span style="margin-right: 5px;">18</span><span style="margin-right: 5px;">17</span><span style="margin-right: 5px;">16</span><span style="margin-right: 5px;">15</span><span style="margin-right: 5px;">14</span><span style="margin-right: 5px;">13</span><span style="margin-right: 5px;">12</span><span style="margin-right: 5px;">11</span><span style="margin-right: 5px;">10</span><span style="margin-right: 5px;">9</span><span style="margin-right: 5px;">8</span><span style="margin-right: 5px;">7</span><span style="margin-right: 5px;">6</span><span style="margin-right: 5px;">5</span><span style="margin-right: 5px;">4</span><span style="margin-right: 5px;">3</span><span style="margin-right: 5px;">2</span><span style="margin-right: 5px;">1</span><span style="margin-right: 5px;">0</span> </div> <table style="margin: auto; border-collapse: collapse;"> <tr> <td style="border: 1px solid black; width: 15px; text-align: center;">1</td> <td style="border: 1px solid black; width: 15px; text-align: center;">1</td> <td style="border: 1px solid black; width: 15px; text-align: center;">1</td> <td style="border: 1px solid black; width: 15px; text-align: center;">0</td> <td style="border: 1px solid black; width: 15px; text-align: center;">0</td> <td style="border: 1px solid black; width: 15px; text-align: center;">1</td> <td style="border: 1px solid black; width: 15px; text-align: center;">1</td> <td style="border: 1px solid black; width: 15px; text-align: center;">1</td> <td style="border: 1px solid black; width: 15px; text-align: center;"> </td> <td style="border: 1px solid black; width: 15px; text-align: center;"> </td> <td style="border: 1px solid black; width: 15px; text-align: center;">0</td> <td style="border: 1px solid black; width: 15px; text-align: center;">0</td> <td style="border: 1px solid black; width: 15px; text-align: center;"> </td> <td style="border: 1px solid black; width: 15px; text-align: center;"> </td> <td style="border: 1px solid black; width: 15px; text-align: center;"> </td> <td style="border: 1px solid black; width: 15px; text-align: center;"> </td> <td style="border: 1px solid black; width: 15px; text-align: center;"> </td> <td style="border: 1px solid black; width: 15px; text-align: center;"> </td> <td style="border: 1px solid black; width: 15px; text-align: center;"> </td> <td style="border: 1px solid black; width: 15px; text-align: center;"> </td> <td style="border: 1px solid black; width: 15px; text-align: center;"> </td> <td style="border: 1px solid black; width: 15px; text-align: center;"> </td> </tr> </table> <div style="display: flex; justify-content: center; margin-top: 5px;"> <div style="text-align: center; margin-right: 20px;">             ACC/ DREG Write         </div> <div style="text-align: center;">             (Page) (Pointer) direct address         </div> </div>	1	1	1	0	0	1	1	1			0	0										
1	1	1	0	0	1	1	1			0	0												
CCR  OVFP	CCR C: } N: } not affected Z: }																						

# FSGYA

## I. Pointer addressing mode

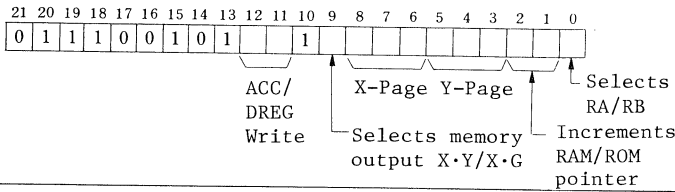
Assembler syntax	[<label>] ΔFSGYAΔ <operand ③>[Δ <comment>]								
Example	FSGYA Δ <u>EE</u> , <u>XG(1, 3)</u> , <u>RA</u> ②          ③          ④								
Operation	<p>● ALU operation Floating point arithmetic</p> <table border="1" data-bbox="341 487 986 630"> <tr> <td>ACCA (20 bits)</td> <td>→</td> <td>ACCA (20bits)</td> <td>If sign ACCA=sign Y</td> </tr> <tr> <td>-ACCA (20bits)</td> <td>→</td> <td>ACCA (20bits)</td> <td>If sign ACCA≠sign Y</td> </tr> </table> <p>If sign bit value of Y-Bus output data specified in operand equals that of ACCA, the ACCA is not affected. Otherwise, -ACCA → ACCA.</p> <p>● Operand For details of operand, see 5.2.1 'Operand ③'. In 'Operand ③', "ACC" means ACCA.</p>	ACCA (20 bits)	→	ACCA (20bits)	If sign ACCA=sign Y	-ACCA (20bits)	→	ACCA (20bits)	If sign ACCA≠sign Y
ACCA (20 bits)	→	ACCA (20bits)	If sign ACCA=sign Y						
-ACCA (20bits)	→	ACCA (20bits)	If sign ACCA≠sign Y						
Instruction code									
CCR OVFP	<p>CCR C: Set if sign ACCA≠sign Y and a mantissa of the ACCA is \$0000 before instruction execution; cleared otherwise. N: Set if ACCA is negative after instruction execution. Z: Set if a mantissa of ACCA is \$0000 after instruction execution.</p> <p>OVFP OVFP bit of the CTR must be set to 1 beforehand.</p>								

## II. Direct addressing mode

Assembler syntax	[<label>] ΔFSGYAΔ <operand ①>[Δ <comment>]																																												
Example	$\text{FSGY} \Delta \underset{\textcircled{2}}{A}, \underset{\textcircled{3}}{2}, 14$																																												
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point arithmetic</li> </ul> <table border="1" data-bbox="346 494 995 633"> <tr> <td>ACCA (20bits)</td> <td>→</td> <td>ACCA (20bits)</td> <td>If sign ACCA=sign Y</td> </tr> <tr> <td>-ACCA (20bits)</td> <td>→</td> <td>ACCA (20bits)</td> <td>If sign ACCA≠sign Y</td> </tr> </table> <p>If sign bit value of data on the operand ③ memory location (Y-Bus output data) equals that of ACCA, the ACCA is not affected. Otherwise, -ACCA → ACCA.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCA.</li> </ul>	ACCA (20bits)	→	ACCA (20bits)	If sign ACCA=sign Y	-ACCA (20bits)	→	ACCA (20bits)	If sign ACCA≠sign Y																																				
ACCA (20bits)	→	ACCA (20bits)	If sign ACCA=sign Y																																										
-ACCA (20bits)	→	ACCA (20bits)	If sign ACCA≠sign Y																																										
Instruction code	<table border="1" data-bbox="327 1223 913 1284"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <p style="text-align: center;"> <span style="margin-right: 100px;">ACC/ DREG Write</span> <span>(Page) (Pointer) direct address</span> </p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	1	1	0	0	1	0	0				0	0								
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
0	1	1	1	0	0	1	0	0				0	0																																
CCR  OVFP	<p>CCR C: Set if sign ACCA≠sign Y and a mantissa of the ACCA is \$0000 before instruction execution; cleared otherwise. N: Set if ACCA is negative after instruction execution. Z: Set if a mantissa of ACCA is \$0000 after instruction execution.</p> <p>OVFP OVFP bit of the CTR must be set to 1 beforehand.</p>																																												

# FSGYB

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔFSGYBΔ <operand ©>[Δ <comment>]								
Example	FSGYBΔ <u>EE</u> , <u>XG(1,3)</u> , <u>RA</u> ②          ③          ④								
Operation	<p>● ALU operation Floating point arithmetic</p> <table border="1" data-bbox="327 486 976 633"> <tr> <td>ACCB (20bits)</td> <td>→</td> <td>ACCB (20bits)</td> <td>If sign ACCB=sign Y</td> </tr> <tr> <td>-ACCB (20bits)</td> <td>→</td> <td>ACCB (20bits)</td> <td>If sign ACCB≠sign Y</td> </tr> </table> <p>If sign bit value of Y-Bus output data specified in operand equals that of ACCB, the ACCB is not affected. Otherwise, -ACCB → ACCB.</p> <p>● Operand For details of operand, see 5.2.1 'Operand ©'. In 'Operand ©', "ACC" means ACCB.</p>	ACCB (20bits)	→	ACCB (20bits)	If sign ACCB=sign Y	-ACCB (20bits)	→	ACCB (20bits)	If sign ACCB≠sign Y
ACCB (20bits)	→	ACCB (20bits)	If sign ACCB=sign Y						
-ACCB (20bits)	→	ACCB (20bits)	If sign ACCB≠sign Y						
Instruction code									
CCR OVFP	<p>CCR C: Set if sign ACCB≠sign Y and a mantissa of the ACCB is \$0000 before instruction execution; cleared otherwise. N: Set if ACCB is negative after instruction execution. Z: Set if a mantissa of ACCB is \$0000 after instruction execution.</p> <p>OVFP OVFP bit of the CTR must be set to 1 beforehand.</p>								

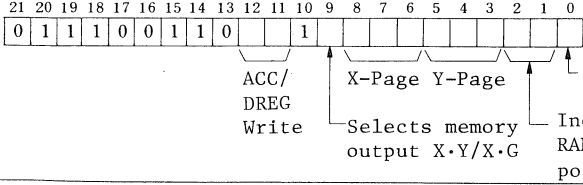


## II. Direct addressing mode

Assembler syntax	[<label>] ΔFSGYBΔ <operand ①>[Δ <comment>]																																												
Example	FSGYB△ <u>A</u> , <u>2</u> , <u>14</u> ②      ③																																												
Operation	<ul style="list-style-type: none"> <li>● ALU operation Floating point arithmetic</li> </ul> <table border="1" data-bbox="331 499 980 638"> <tr> <td>ACCB (20bits)</td> <td>→</td> <td>ACCB (20bits)</td> <td>If sign ACCB=sign Y</td> </tr> <tr> <td>-ACCB (20bits)</td> <td>→</td> <td>ACCB (20bits)</td> <td>If sign ACCB≠sign Y</td> </tr> </table> <p>If sign bit value of data on the operand ③ memory location (Y-Bus output data) equals that of ACCB, the ACCB is not affected. Otherwise, -ACCB → ACCB.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCB.</li> </ul>	ACCB (20bits)	→	ACCB (20bits)	If sign ACCB=sign Y	-ACCB (20bits)	→	ACCB (20bits)	If sign ACCB≠sign Y																																				
ACCB (20bits)	→	ACCB (20bits)	If sign ACCB=sign Y																																										
-ACCB (20bits)	→	ACCB (20bits)	If sign ACCB≠sign Y																																										
Instruction code	<table border="1" data-bbox="316 1234 904 1298"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table> <p style="text-align: center;"> <span style="margin-right: 100px;">ACC/ DREG Write</span> <span>(Page) (Pointer) direct address</span> </p>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	1	1	0	0	1	0	1			0	0									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
0	1	1	1	0	0	1	0	1			0	0																																	
CCR  OVFP	<p>CCR C: Set if sign ACCB≠sign Y and a mantissa of the ACCB is \$0000 before instruction execution; cleared otherwise.</p> <p>N: Set if ACCB is negative after instruction execution.</p> <p>Z: Set if a mantissa of ACCB is \$0000 after instruction execution.</p> <p>OVFP OVFP bit of the CTR must be set to 1 beforehand.</p>																																												

# SGYA

## I. Pointer addressing mode

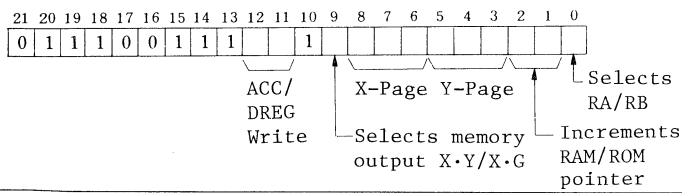
Assembler syntax	[<label>] ΔSGYAΔ <operand ③>[Δ <comment>]								
Example	SGYAΔ <u>EE</u> , <u>XG(1, 3)</u> , <u>RA</u> ②          ③          ④								
Operation	<p>● ALU operation Fixed point arithmetic</p> <table border="1" data-bbox="341 487 990 626"> <tr> <td>ACCA (16bits)</td> <td>→</td> <td>ACCA (16bits)</td> <td>If sign ACCA=sign Y</td> </tr> <tr> <td>-ACCA (16bits)</td> <td>→</td> <td>ACCA (16bits)</td> <td>If sign ACCA≠sign Y</td> </tr> </table> <p>If sign bit value of Y-Bus output data specified in operand equals that of ACCA, the ACCA is not affected. Otherwise, -ACCA → ACCA.</p> <p>● Operand For details of operand, see 5.2.1 'Operand ③'. In 'Operand ③', "ACC" means ACCA.</p>	ACCA (16bits)	→	ACCA (16bits)	If sign ACCA=sign Y	-ACCA (16bits)	→	ACCA (16bits)	If sign ACCA≠sign Y
ACCA (16bits)	→	ACCA (16bits)	If sign ACCA=sign Y						
-ACCA (16bits)	→	ACCA (16bits)	If sign ACCA≠sign Y						
Instruction code									
CCR OVFP	<p>CCR C: Set if sign ACCA≠sign Y and the ACCA is \$0000 before instruction execution; cleared otherwise. N: Set if ACCA is negative after instruction execution. Z: Set if ACCA is \$0000 after instruction execution.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit. 0: no overflow protection 1: overflow protection</p>								

## II. Direct addressing mode

Assembler syntax	[<label>] ΔSGYAΔ <operand ①>[Δ <comment>]																																																															
Example	$SGYA \triangle \underset{\textcircled{2}}{A}, \underset{\textcircled{3}}{2}, 14$																																																															
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point arithmetic</li> </ul> <table border="1" style="margin: 10px auto; border-collapse: collapse; text-align: center;"> <tr> <td style="padding: 5px;">ACCA (16bits)</td> <td style="padding: 5px;">→</td> <td style="padding: 5px;">ACCA (16bits)</td> <td style="padding: 5px;">If sign ACCA=sign Y</td> </tr> <tr> <td style="padding: 5px;">-ACCA (16bits)</td> <td style="padding: 5px;">→</td> <td style="padding: 5px;">ACCA (16bits)</td> <td style="padding: 5px;">If sign ACCA≠sign Y</td> </tr> </table> <p style="margin-top: 10px;">If sign bit value of data on the operand ③ memory location (Y-Bus output data) equals that of ACCA, the ACCA is not affected. Otherwise, -ACCA → ACCA.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCA.</li> </ul>	ACCA (16bits)	→	ACCA (16bits)	If sign ACCA=sign Y	-ACCA (16bits)	→	ACCA (16bits)	If sign ACCA≠sign Y																																																							
ACCA (16bits)	→	ACCA (16bits)	If sign ACCA=sign Y																																																													
-ACCA (16bits)	→	ACCA (16bits)	If sign ACCA≠sign Y																																																													
Instruction code	<table style="margin: 0 auto; border-collapse: collapse;"> <tr> <td style="text-align: center; padding: 0 5px;">21</td><td style="text-align: center; padding: 0 5px;">20</td><td style="text-align: center; padding: 0 5px;">19</td><td style="text-align: center; padding: 0 5px;">18</td><td style="text-align: center; padding: 0 5px;">17</td><td style="text-align: center; padding: 0 5px;">16</td><td style="text-align: center; padding: 0 5px;">15</td><td style="text-align: center; padding: 0 5px;">14</td><td style="text-align: center; padding: 0 5px;">13</td><td style="text-align: center; padding: 0 5px;">12</td><td style="text-align: center; padding: 0 5px;">11</td><td style="text-align: center; padding: 0 5px;">10</td><td style="text-align: center; padding: 0 5px;">9</td><td style="text-align: center; padding: 0 5px;">8</td><td style="text-align: center; padding: 0 5px;">7</td><td style="text-align: center; padding: 0 5px;">6</td><td style="text-align: center; padding: 0 5px;">5</td><td style="text-align: center; padding: 0 5px;">4</td><td style="text-align: center; padding: 0 5px;">3</td><td style="text-align: center; padding: 0 5px;">2</td><td style="text-align: center; padding: 0 5px;">1</td><td style="text-align: center; padding: 0 5px;">0</td> </tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td><td style="border: 1px solid black; width: 15px; height: 15px;"></td> </tr> <tr> <td colspan="11" style="text-align: center; padding-top: 5px;">ACC/ DREG Write</td> <td colspan="4" style="text-align: center; padding-top: 5px;">(Page) direct address</td> <td colspan="4" style="text-align: center; padding-top: 5px;">(Pointer)</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	1	1	0	0	1	1	0			0	0										ACC/ DREG Write											(Page) direct address				(Pointer)			
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																											
0	1	1	1	0	0	1	1	0			0	0																																																				
ACC/ DREG Write											(Page) direct address				(Pointer)																																																	
CCR  OVFP	<p>CCR C: Set if sign ACCA≠sign Y and the ACCA is \$0000 before instruction execution; cleared otherwise.</p> <p>N: Set if ACCA is negative after instruction execution.</p> <p>Z: Set if ACCA is \$0000 after instruction execution.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit. 0: no overflow protection    1: overflow protection</p>																																																															

# SGYB

## I. Pointer addressing mode

Assembler syntax	[<label>] ΔSGYBΔ <operand ①>[Δ <comment>]				
Example	SGYBΔ <u>EE</u> , <u>XG(1, 3)</u> , <u>RA</u> ②          ③          ④				
Operation	<p>● ALU operation Fixed point arithmetic</p> <table border="1" data-bbox="340 486 989 633"> <tr> <td>ACCB → ACCB (16bits) (16bits)</td> <td>If sign ACCB=sign Y</td> </tr> <tr> <td>-ACCB → ACCB (16bits) (16bits)</td> <td>If sign ACCB≠sign Y</td> </tr> </table> <p>If sign bit value of Y-Bus output data specified in operand equals that of ACCB, the ACCB is not affected. Otherwise, -ACCB → ACCB.</p> <p>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCB.</p>	ACCB → ACCB (16bits) (16bits)	If sign ACCB=sign Y	-ACCB → ACCB (16bits) (16bits)	If sign ACCB≠sign Y
ACCB → ACCB (16bits) (16bits)	If sign ACCB=sign Y				
-ACCB → ACCB (16bits) (16bits)	If sign ACCB≠sign Y				
Instruction code	 <p>The diagram shows a 22-bit instruction code field. Bits 21-10 are labeled 'ACC/DREG Write' and contain the binary value 011100111. Bits 9-2 are labeled 'X-Page Y-Page' and are empty. Bit 1 is labeled 'Selects RA/RB' and contains '1'. Bit 0 is labeled 'Increments RAM/ROM pointer' and contains '0'.</p>				
CCR OVFP	<p>CCR C: Set if sign ACCB≠sign Y and the ACCB is \$0000 before instruction execution; cleared otherwise. N: Set if ACCB is negative after instruction execution. Z: Set if ACCB is \$0000 after instruction execution.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit. 0: no overflow protection 1: overflow protection</p>				

## II. Direct addressing mode

Assembler syntax	[<label>] ΔSGYBΔ <operand ①>[Δ <comment>]																																																																		
Example	$\text{SGYB} \triangle \frac{\text{A}}{\text{②}}, \frac{2, 14}{\text{③}}$																																																																		
Operation	<ul style="list-style-type: none"> <li>● ALU operation Fixed point arithmetic</li> </ul> <table border="1" style="margin: 10px auto; border-collapse: collapse; text-align: center;"> <tr> <td style="padding: 5px;">ACCB (16 bits)</td> <td style="padding: 5px;">→</td> <td style="padding: 5px;">ACCB (16bits)</td> <td style="padding: 5px;">If sign ACCB=sign Y</td> </tr> <tr> <td style="padding: 5px;">-ACCB (16bits)</td> <td style="padding: 5px;">→</td> <td style="padding: 5px;">ACCB (16bits)</td> <td style="padding: 5px;">If sign ACCB≠sign Y</td> </tr> </table> <p style="margin-top: 10px;">If sign bit value of data on the operand ③ memory location (Y-Bus output data) equals that of ACCB, the ACCB is not affected. Otherwise, -ACCB → ACCB.</p> <ul style="list-style-type: none"> <li>● Operand For details of operand, see 5.2.1 'Operand ①'. In 'Operand ①', "ACC" means ACCB.</li> </ul>	ACCB (16 bits)	→	ACCB (16bits)	If sign ACCB=sign Y	-ACCB (16bits)	→	ACCB (16bits)	If sign ACCB≠sign Y																																																										
ACCB (16 bits)	→	ACCB (16bits)	If sign ACCB=sign Y																																																																
-ACCB (16bits)	→	ACCB (16bits)	If sign ACCB≠sign Y																																																																
Instruction code	<table style="margin: 0 auto; border-collapse: collapse;"> <tr> <td style="text-align: center; padding: 0 5px;">21</td><td style="text-align: center; padding: 0 5px;">20</td><td style="text-align: center; padding: 0 5px;">19</td><td style="text-align: center; padding: 0 5px;">18</td><td style="text-align: center; padding: 0 5px;">17</td><td style="text-align: center; padding: 0 5px;">16</td><td style="text-align: center; padding: 0 5px;">15</td><td style="text-align: center; padding: 0 5px;">14</td><td style="text-align: center; padding: 0 5px;">13</td><td style="text-align: center; padding: 0 5px;">12</td><td style="text-align: center; padding: 0 5px;">11</td><td style="text-align: center; padding: 0 5px;">10</td><td style="text-align: center; padding: 0 5px;">9</td><td style="text-align: center; padding: 0 5px;">8</td><td style="text-align: center; padding: 0 5px;">7</td><td style="text-align: center; padding: 0 5px;">6</td><td style="text-align: center; padding: 0 5px;">5</td><td style="text-align: center; padding: 0 5px;">4</td><td style="text-align: center; padding: 0 5px;">3</td><td style="text-align: center; padding: 0 5px;">2</td><td style="text-align: center; padding: 0 5px;">1</td><td style="text-align: center; padding: 0 5px;">0</td> </tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td> </tr> <tr> <td colspan="11" style="text-align: center; padding-top: 5px;">ACC/ DREG Write</td> <td colspan="4" style="text-align: center; padding-top: 5px;">(Page)</td> <td colspan="7" style="text-align: center; padding-top: 5px;">(Pointer) direct address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	1	1	0	0	0	1	1	1	1	0	0										ACC/ DREG Write											(Page)				(Pointer) direct address						
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																														
0	1	1	1	0	0	0	1	1	1	1	0	0																																																							
ACC/ DREG Write											(Page)				(Pointer) direct address																																																				
CCR  OVFP	<p>CCR C: Set if sign ACCB≠sign Y and the ACCB is \$0000 before instruction execution; cleared otherwise.</p> <p>N: Set if ACCB is negative after instruction execution.</p> <p>Z: Set if ACCB is \$0000 after instruction execution.</p> <p>OVFP Overflow protection is controlled by the status of OVFP bit. 0: no overflow protection    1: overflow protection</p>																																																																		

# LIA

Assembler syntax	[<label>] ΔLIAΔ <constant> [Δ <comment>]
Example	LIA Δ30
Operation	<p>The constant value (16 bits) is stored in ACCA. Immediate data must be used for a constant.</p>
Instruction code	
CCR	<p>CCR C: Not affected. N: Set if ACCA is negative. Z: Set if ACCA is 0.</p>

# LIB

Assembler syntax	[<label>] ΔLIBΔ <constant> [Δ <comment>]
Example	LIB Δ\$01FA+2
Operation	<p>The constant value (16 bits) is stored in ACCB. Immediate data must be used for a constant.</p>
Instruction code	
CCR	<p>CCR C: Not affected. N: Set if ACCB is negative. Z: Set if ACCB is 0.</p>

# LIRA

Assembler syntax	[<label>] ΔLIRAΔ <constant> [Δ <comment>]
Example	LIRA Δ29
Operation	<p>The constant value (6 bits) is transferred to the RAM pointer A. Immediate data must be used for a constant.</p> <p>Immediate data (6 bits)      The value is selectable from 0 to 63.</p>
Instruction code	
CCR	CCR C: } N: } Not affected. Z: }

# LIRB

Assembler syntax	[<label>] ΔLIRBΔ <constant> [Δ <comment>]
Example	LIRB Δ41
Operation	<p>The constant value (6 bits) is transferred to the RAM pointer B. Immediate data must be used for a constant.</p> <p>Immediate data (6 bits)      The value is selectable from 0 to 63.</p>
Instruction code	
CCR	CCR C: } N: } Not affected. Z: }

# LIRO

Assembler syntax	[<label>] ΔLIROΔ <constant> [Δ <comment>]																																													
Example	LIRO Δ13																																													
Operation	<p>The constant value (6 bits) is transferred to the ROM pointer. Immediate data must be used for a constant.</p> <p>Immediate data (6 bits)                      The value is selectable from 0 to 63.</p> <div style="text-align: center;"> <p>ROM pointer</p> <p>MSB                      LSB ▲...Decimal point</p> </div>																																													
Instruction code	<div style="text-align: center;"> <table border="1" style="border-collapse: collapse; width: 100%;"> <tr> <td style="width: 2.5%;">21</td><td style="width: 2.5%;">20</td><td style="width: 2.5%;">19</td><td style="width: 2.5%;">18</td><td style="width: 2.5%;">17</td><td style="width: 2.5%;">16</td><td style="width: 2.5%;">15</td><td style="width: 2.5%;">14</td><td style="width: 2.5%;">13</td><td style="width: 2.5%;">12</td><td style="width: 2.5%;">11</td><td style="width: 2.5%;">10</td><td style="width: 2.5%;">9</td><td style="width: 2.5%;">8</td><td style="width: 2.5%;">7</td><td style="width: 2.5%;">6</td><td style="width: 2.5%;">5</td><td style="width: 2.5%;">4</td><td style="width: 2.5%;">3</td><td style="width: 2.5%;">2</td><td style="width: 2.5%;">1</td><td style="width: 2.5%;">0</td> </tr> <tr> <td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td> </tr> </table> <p>OP code    Immediate data</p> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	0	0	1	0	0							0	0	0	0	0	0	0	0	0	0	0
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																									
1	0	0	1	0	0							0	0	0	0	0	0	0	0	0	0	0																								
CCR	CCR C: } N: } Not affected. Z: }																																													

# LIRC

Assembler syntax	[<label>] ΔLIRCA <constant> [Δ <comment>]																																													
Example	LIRC Δ15																																													
Operation	<p>The constant value (6 bits) is transferred to the RC. Immediate data must be used for a constant.</p> <p>Immediate data (6 bits)                      The value is selectable from 0 to 63.</p> <div style="text-align: center;"> <p>RC</p> <p>MSB                      LSB ▲... Decimal point</p> </div>																																													
Instruction code	<div style="text-align: center;"> <table border="1" style="border-collapse: collapse; width: 100%;"> <tr> <td style="width: 2.5%;">21</td><td style="width: 2.5%;">20</td><td style="width: 2.5%;">19</td><td style="width: 2.5%;">18</td><td style="width: 2.5%;">17</td><td style="width: 2.5%;">16</td><td style="width: 2.5%;">15</td><td style="width: 2.5%;">14</td><td style="width: 2.5%;">13</td><td style="width: 2.5%;">12</td><td style="width: 2.5%;">11</td><td style="width: 2.5%;">10</td><td style="width: 2.5%;">9</td><td style="width: 2.5%;">8</td><td style="width: 2.5%;">7</td><td style="width: 2.5%;">6</td><td style="width: 2.5%;">5</td><td style="width: 2.5%;">4</td><td style="width: 2.5%;">3</td><td style="width: 2.5%;">2</td><td style="width: 2.5%;">1</td><td style="width: 2.5%;">0</td> </tr> <tr> <td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td> </tr> </table> <p>OP cedo    Immediate data</p> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	0	0	1	0	1							0	0	0	0	0	0	0	0	0	0	0
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																									
1	0	0	1	0	1							0	0	0	0	0	0	0	0	0	0	0																								
CCR	CCR C: } N: } Not affected. Z: }																																													



# JCS

Assembler syntax	[<label>] ΔJCSΔ <constant> [Δ <comment>]																																																														
Example	JCS Δ <u>LABL1</u> ①																																																														
Operation	<p>This instruction produces a jump if a carry flag of the CCR is set.            ① : label or address of the destination (address=0 ~ 486)</p> <p>Even if an interrupt is requested under a jump condition, it is not accepted before the next instruction is completely executed.            (Interrupt wait state)</p>																																																														
Instruction code	<table style="border-collapse: collapse; margin: auto;"> <tr> <td style="text-align: center;">21</td><td style="text-align: center;">20</td><td style="text-align: center;">19</td><td style="text-align: center;">18</td><td style="text-align: center;">17</td><td style="text-align: center;">16</td><td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td> </tr> <tr> <td colspan="4" style="text-align: center;">OP code</td> <td colspan="4" style="text-align: center;">CCR</td> <td colspan="10" style="text-align: center;">Jump address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	OP code				CCR				Jump address									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																										
1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																										
OP code				CCR				Jump address																																																							
CCR	CCR C: } N: } Not affected. Z: }																																																														

# JNS

Assembler syntax	[<label>] ΔJNSΔ <constant> [Δ <comment>]																																																														
Example	JNS Δ <u>LABL2</u> ①																																																														
Operation	<p>This instruction produces a jump if a negative flag of the CCR is set.            ① : label or address of the destination (address=0 ~ 486)</p> <p>Even if an interrupt is requested under a jump condition, it is not accepted before the next instruction is completely executed.            (Interrupt wait state)</p>																																																														
Instruction code	<table style="border-collapse: collapse; margin: auto;"> <tr> <td style="text-align: center;">21</td><td style="text-align: center;">20</td><td style="text-align: center;">19</td><td style="text-align: center;">18</td><td style="text-align: center;">17</td><td style="text-align: center;">16</td><td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td> </tr> <tr> <td colspan="4" style="text-align: center;">OP code</td> <td colspan="4" style="text-align: center;">CCR</td> <td colspan="10" style="text-align: center;">Jump address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	OP code				CCR				Jump address									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																										
1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0																																										
OP code				CCR				Jump address																																																							
CCR	CCR C: } N: } Not affected. Z: }																																																														

# JZS

Assembler syntax	[<label>] ΔJZSΔ <constant> [Δ <comment>]																																																														
Example	JZS Δ <u>LBL3</u> ①																																																														
Operation	<p>This instruction produces a jump if a zero flag of the CCR is set.            ① : label or address of the destination (address=0 ~ 486)</p> <p>Even if an interrupt is requested under a jump condition, it is not accepted before the next instruction is completely executed. (Interrupt wait state)</p>																																																														
Instruction code	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td> </tr> <tr> <td colspan="4" style="text-align: center;">OP code</td> <td colspan="4" style="text-align: center;">CCR</td> <td colspan="10" style="text-align: center;">Jump address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	OP code				CCR				Jump address									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																										
1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0																																										
OP code				CCR				Jump address																																																							
CCR	CCR C: } N: } Not affected. Z: }																																																														

# JSR

Assembler syntax	[<label>] ΔJSRΔ <constant> [Δ <comment>]																																																														
Example	JSR Δ <u>LBL4</u> ①																																																														
Operation	<p>This instruction produces a jump to subroutine.            The contents of the PC is pushed onto the stack and the jump address is transferred to the PC.</p> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <span style="border: 1px solid black; padding: 2px 10px;">Jump address</span> → <span style="border: 1px solid black; padding: 2px 10px;">PC (9 bits)</span> → <span style="border: 1px solid black; padding: 2px 10px;">STACK 0</span> → <span style="border: 1px solid black; padding: 2px 10px;">STACK 1</span> </div> <p>① : label or address of the destination (address=0 ~ 486)</p> <p>Even if an interrupt is requested during execution of this instruction, it is not accepted before the next instruction is completely executed. (Interrupt wait state)</p>																																																														
Instruction code	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td> </tr> <tr> <td colspan="4" style="text-align: center;">OP code</td> <td colspan="4" style="text-align: center;">CCR</td> <td colspan="10" style="text-align: center;">Jump address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	OP code				CCR				Jump address									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																										
1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																										
OP code				CCR				Jump address																																																							
CCR	CCR C: } N: } Not affected. Z: }																																																														

# JNZ

Assembler syntax	[<label>] ΔJNZΔ <constant> [Δ <comment>]																																																														
Example	JNZ Δ <u>LABL5</u> ①																																																														
Operation	<p>This instruction produces a jump if value in the RC is not 0.</p> <p>① : label or address of the destination (address=0 ~ 486)</p> <p>Even if an interrupt is requested under a jump condition, it is not accepted before the next instruction is completely executed. (Interrupt wait state)</p>																																																														
Instruction code	<table style="border-collapse: collapse; margin: auto;"> <tr> <td style="text-align: center;">21</td><td style="text-align: center;">20</td><td style="text-align: center;">19</td><td style="text-align: center;">18</td><td style="text-align: center;">17</td><td style="text-align: center;">16</td><td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td> </tr> <tr> <td colspan="4" style="text-align: center;">OP code</td> <td colspan="4" style="text-align: center;">CCR</td> <td colspan="10" style="text-align: center;">Jump address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0									OP code				CCR				Jump address									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																										
1	0	1	0	1	1	0	0	0	0	0	0	0	0																																																		
OP code				CCR				Jump address																																																							
CCR	CCR C: } N: } Not affected. Z: }																																																														

# JNZM

Assembler syntax	[<label>] ΔJNZMΔ <constant> [Δ <comment>]																																																														
Example	JNZM Δ <u>LABL6</u> ①																																																														
Operation	<p>This instruction produces a jump if value in the RC is not 0, and decrements the RC simultaneously.</p> <p>① : label or address of the destination (address=0 ~ 486)</p> <p>Even if an interrupt is requested under a jump condition, it is not accepted before the next instruction is completely executed. (Interrupt wait state)</p>																																																														
Instruction code	<table style="border-collapse: collapse; margin: auto;"> <tr> <td style="text-align: center;">21</td><td style="text-align: center;">20</td><td style="text-align: center;">19</td><td style="text-align: center;">18</td><td style="text-align: center;">17</td><td style="text-align: center;">16</td><td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td><td style="text-align: center;"> </td> </tr> <tr> <td colspan="4" style="text-align: center;">OP code</td> <td colspan="4" style="text-align: center;">CCR</td> <td colspan="10" style="text-align: center;">Jump address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	0	1	0	1	1	1	0	0	0	0	0	0										OP code				CCR				Jump address									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																										
1	0	1	0	1	1	1	0	0	0	0	0	0																																																			
OP code				CCR				Jump address																																																							
CCR	CCR C: } N: } Not affected. Z: }																																																														

# JMP

Assembler syntax	[<label>] ΔJMPΔ <constant> [Δ <comment>]																																																														
Example	JMP Δ <u>LABL7</u> ①																																																														
Operation	<p>This is an unconditional jump instruction.          ① : label or address of the destination (address=0 ~ 486)</p> <p>Even if an interrupt is requested during execution of this instruction, it is not accepted before the next instruction is completely executed. (Interrupt wait state)          Therefore, the instruction which always wait for an interrupt cannot be used as Interrupt wait instruction.</p> <p>ex. LABLΔJMPΔLABL</p>																																																														
Instruction code	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="text-align: center;">21</td><td style="text-align: center;">20</td><td style="text-align: center;">19</td><td style="text-align: center;">18</td><td style="text-align: center;">17</td><td style="text-align: center;">16</td><td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td><td style="border: 1px solid black; text-align: center;"> </td> </tr> <tr> <td colspan="4" style="text-align: center;">OP code</td> <td colspan="4" style="text-align: center;">CCR</td> <td colspan="10" style="text-align: center;">Jump address</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0									OP code				CCR				Jump address									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																										
1	0	1	0	0	0	0	0	0	0	0	0	0	0																																																		
OP code				CCR				Jump address																																																							
CCR	CCR C: } N: } Not affected. Z: }																																																														

# TFR

Assembler syntax	[<label>] ΔTFRΔ <register 1>, <register 2> [Δ <comment>]
Example	TFR Δ <u>A</u> , <u>STR</u> ① ②
Operation	<p>This instruction transfers data between register.</p> <p>① : Source register ② : Destination register</p> <p>Data is transferred from register ① to register ②. If data is transferred to an accumulator, an exponent data in the accumulator may be changed.</p> <p>38 kinds of transfer instructions (① ~ ③⑧) are described from the following pages.</p>

## Transfer ①

Expression	TFR ΔA,STR																																												
Operation	<p>The contents of bit positions 3 through 7 of ACCA are transferred to the corresponding bit position of the STR to enable or disable an interrupt. The contents of ACCA remain unchanged.</p> <div style="text-align: center;"> </div>																																												
Instruction code	<div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td> </tr> </table> <p style="text-align: center;"> <span style="margin-right: 40px;">OP code</span> <span style="margin-right: 40px;">Selects ACCA/B</span> <span style="margin-right: 40px;">Unused</span> <span>Selects RAM pointer A/B</span> </p> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																								
CCR	CCR C: } N: } Not affected. Z: }																																												

## Transfer ②

Expression	TFR ΔB,STR																																												
Operation	<p>The contents of bit positions 3 through 7 of ACCB are transferred to the corresponding bit position of the STR to enable or disable an interrupt. The contents of ACCB remain unchanged.</p> <div style="text-align: center;"> </div>																																												
Instruction code	<div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td> </tr> </table> <p style="text-align: center;"> <span style="margin-right: 40px;">OP code</span> <span style="margin-right: 40px;">Selects ACCA/B</span> <span style="margin-right: 40px;">Unused</span> <span>Selects RAM pointer A/B</span> </p> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0																								
CCR	CCR C: } N: } Not affected. Z: }																																												

### Transfer ③

Expression	TFR ΔA,CTR
Operation	<p>The contents of bit positions 0 through 7 of ACCA are transferred to the corresponding bit position of the CTR. This instruction determines a data transfer mode and controls the status of the BIT I/O and TxRQ pins. The contents of ACCA remain unchanged.</p> <div style="text-align: center;"> </div> <p>Unused bits of the CTR (bit 2,3,6) must be set to 0.</p>
Instruction code	<div style="text-align: center;"> </div>
CCR	CCR C: } N: } Not affected. Z: }

### Transfer ④

Expression	TFR ΔB,CTR
Operation	<p>The contents of bit positions 0 through 7 of ACCB are transferred to the corresponding bit position of the CTR. This instruction determines a data transfer mode and controls the status of the BIT I/O and TxRQ pins. The contents of ACCB remain unchanged.</p> <div style="text-align: center;"> </div> <p>Unused bits of the CTR (bit 2,3,6) must be set to 0.</p>
Instruction code	<div style="text-align: center;"> </div>
CCR	CCR C: } N: } Not affected. Z: }

## Transfer ⑤

Expression	TFR ΔA,RC
Operation	<p>The contents of bit positions 10 through 15 of ACCA are transferred to the corresponding bit position of the RC. The contents of ACCA remain unchanged.</p>
Instruction code	
CCR	CCR C: } N: } Not affected. Z: }

## Transfer ⑥

Expression	TFR ΔB,RC
Operation	<p>The contents of bit positions 10 through 15 of ACCB are transferred to the corresponding bit position of the RC. The contents of ACCB remain unchanged.</p>
Instruction code	
CCR	CCR C: } N: } Not affected. Z: }



## Transfer ⑦

Expression	TFR ΔA,OR																																												
Operation	<p>The contents of bit positions 0 through 15 of ACCA are transferred to the corresponding bit position of OR. The contents of ACCA remain unchanged.</p> <div style="text-align: center;"> <p>Mantissa</p> <p>ACCA</p> <p>OR</p> </div>																																												
Instruction code	<div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td> </tr> </table> <p style="text-align: center;"> <span style="margin-right: 40px;">OP code</span> <span style="margin-right: 40px;">Selects ACCA/B</span> <span style="margin-right: 40px;">Unused</span> <span>Selects RAM pointer A/B</span> </p> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																								
CCR	CCR C: } N: } Not affected. Z: }																																												

## Transfer ⑧

Expression	TFR ΔB,OR																																												
Operation	<p>The contents of bit positions 0 through 15 of ACCB are transferred to the corresponding bit position of the OR. The contents of ACCB remain unchanged.</p> <div style="text-align: center;"> <p>Mantissa</p> <p>ACCB</p> <p>OR</p> </div>																																												
Instruction code	<div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td> </tr> </table> <p style="text-align: center;"> <span style="margin-right: 40px;">OP code</span> <span style="margin-right: 40px;">Selects ACCA/B</span> <span style="margin-right: 40px;">Unused</span> <span>Selects RAM pointer A/B</span> </p> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0																								
CCR	CCR C: } N: } Not affected Z: }																																												

## Transfer ⑨

Expression	TFR ΔA,RO
Operation	<p>The contents of bit positions 10 through 15 of ACCA are transferred to the corresponding bit position of the ROM pointer. The contents of ACCA remain unchanged. An unused MSB of the ROM pointer must be set to 0.</p>
Instruction code	
CCR	<p>CCR C: }          N: } Not affected.          Z: }</p>

## Transfer ⑩

Expression	TFR ΔB,RO
Operation	<p>The contents of bit positions 10 through 15 of ACCB are transferred to the corresponding bit position of the ROM pointer. The contents of ACCB remain unchanged. An unused MSB of the ROM pointer must be set to 0.</p>
Instruction code	
CCR	<p>CCR C: }          N: } Not affected.          Z: }</p>

## Transfer ⑪

Expression	TFR ΔA,RA
Operation	<p>The contents of bit positions 10 through 15 of ACCA are transferred to the corresponding bit position of the RAM pointer A. The contents of ACCA remain unchanged.</p>
Instruction code	
CCR	<p>CCR C: } Not affected.          N: }          Z: }</p>

## Transfer ⑫

Expression	TFR ΔB,RA
Operation	<p>The contents of bit positions 10 through 15 of ACCB are transferred to the corresponding bit position of the RAM pointer A. The contents of ACCB remain unchanged.</p>
Instruction code	
CCR	<p>CCR C: } Not affected.          N: }          Z: }</p>

## Transfer ⑬

Expression	TFR ΔA, RB
Operation	<p>The contents of bit positions 10 through 15 of ACCA are transferred to the corresponding bit position of the RAM pointer B. The contents of ACCA remain unchanged.</p>
Instruction code	
CCR	<p>CCR C: } Not affected.          N: }          Z: }</p>

## Transfer ⑭

Expression	TFR ΔB, RB
Operation	<p>The contents of bit positions 10 through 15 of ACCB are transferred to the corresponding bit position of the RAM pointer B. The contents of ACCB remain unchanged.</p>
Instruction code	
CCR	<p>CCR C: } Not affected.          N: }          Z: }</p>

## Transfer ⑮

Expression	TFR Δ A,CCR
Operation	<p>The contents of bit positions 13 through 15 of ACCA are transferred to the corresponding bit position of the CCR. The contents of ACCA remain unchanged.</p>
Instruction code	
CCR	<p>CCR C: }          N: } The contents of ACCA          Z: }</p>

## Transfer ⑯

Expression	TFR Δ B,CCR
Operation	<p>The contents of bit positions 13 through 15 of ACCB are transferred to the corresponding bit position of the CCR. The contents of ACCB remain unchanged.</p>
Instruction code	
CCR	<p>CCR C: }          N: } The contents of ACCB          Z: }</p>

## Transfer ⑰

Expression	TFR ΔA, SOR
Operation	<p>The contents of bit positions 0 through 15 of ACCA are transferred to the corresponding bit position of the SOR. The contents of ACCA remain unchanged.</p>
Instruction code	
CCR	<p>CCR C: } Not affected.          N: }          Z: }</p>

## Transfer ⑱

Expression	TFR ΔB, SOR
Operation	<p>The contents of bit positions 0 through 15 of ACCB are transferred to the corresponding bit position of the SOR. The contents of ACCB remain unchanged.</p>
Instruction code	
CCR	<p>CCR C: } Not affected.          N: }          Z: }</p>

## Transfer ①9

Expression	TFR Δ STR,A
Operation	<p>The contents of the STR (bit 0 through 7) are transferred to the corresponding bit position of ACCA.</p> <p>This instruction reads the status of the I/O data transfer end flag. At the completion of data transfer, PF, SIF and SOF are cleared.</p> <div style="text-align: center;"> </div> <p style="text-align: center;"> </p>
Instruction code	<div style="text-align: center;"> </div>
CCR	<p>CCR C: Not affected.</p> <p>N: 1</p> <p>Z: 0</p>

## Transfer ②0

Expression	TFR Δ STR,B
Operation	<p>The contents of the STR (bit 0 through 7) are transferred to the corresponding bit position of ACCB.</p> <p>This instruction reads the status of the I/O data transfer end flag. At the completion of data transfer, PF, SIF and SOF are cleared.</p> <div style="text-align: center;"> </div> <p style="text-align: center;"> </p>
Instruction code	<div style="text-align: center;"> </div>
CCR	<p>CCR C: Not affected.</p> <p>N: 1</p> <p>Z: 0</p>

## Transfer ②1

Expression	TFR ΔCTR,A
Operation	<p>The contents of the CTR (bit 0 through 7) are transferred to the corresponding bit position of ACCA.</p> <p>This instruction reads the status of CTR bits. A read of bit 5 means an input from BIT I/O pin.</p> <div style="text-align: center;"> <p>CTR: 7 0 w/B BIT TX OVP/DMA I/O RQ</p> <p>ACCA: 15 7 0 1 1 1 1 1 1 1 * 1 1 1</p> <p>Mantissa Exponent</p> </div> <p>* When using the HSP emulator, the content of bit 6 may be changed.</p>
Instruction code	<div style="text-align: center;"> <p>21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</p> <p>1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0</p> <p>OP code Selects Unused Selects ACCA/B RAM pointer A/B</p> </div>
CCR	<p>CCR C: Not affected.</p> <p>N: 1</p> <p>Z: 0</p>

## Transfer ②2

Expression	TFR ΔCTR,B
Operation	<p>The contents of the CTR (bit 0 through 7) are transferred to the corresponding bit position of ACCB.</p> <p>This instruction reads the status of CTR bits. A read of bit 5 means an input from BIT I/O pin.</p> <div style="text-align: center;"> <p>CTR: 7 0 w/B BIT TX OVP/DMA I/O RQ</p> <p>ACCB: 15 7 0 1 1 1 1 1 1 1 * 1 1 1</p> <p>Mantissa Exponent</p> </div> <p>* When using the HSP emulator, the content of bit 6 may be changed.</p>
Instruction code	<div style="text-align: center;"> <p>21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</p> <p>1 1 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0</p> <p>OP code Selects Unused Selects ACCA/B RAM pointer A/B</p> </div>
CCR	<p>CCR C: Not affected.</p> <p>N: 1</p> <p>Z: 0</p>





## Transfer ②⑤

Expression	TFR Δ IR,A
Operation	<p>The contents of the IR (bit 0 through 15) are transferred to the corresponding bit position of ACCA.</p>
Instruction code	
CCR	<p>CCR C: Not affected.          N: The content of the MSB of the IR          Z: Set if the IR is \$0000.</p>

## Transfer ②⑥

Expression	TFR Δ IR,B
Operation	<p>The contents of the IR (bit 0 through 15) are transferred to the corresponding bit position of ACCB.</p>
Instruction code	
CCR	<p>CCR C: Not affected.          N: The content of the MSB of the IR.          Z: Set if the IR is \$0000.</p>

## Transfer ②7

Expression	TFR ΔRO,A
Operation	<p>The contents of the ROM pointer (bit 10 through 15) are transferred to the corresponding bit position of ACCA. The content of the MSB of ROM pointer (bit 15) is also transferred to ACCA, though unused.</p> <p>ROM pointer: 15 10</p> <p>ACCA: 15 0</p> <p>Mantissa: 1 1 1 1 1 1 1 1 1 1</p> <p>Exponent: Undefined (bits 3-0)</p>
Instruction code	<p>21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</p> <p>1 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0</p> <p>OP code      Selects ACCA/B      Unused      Selects RAM pointer A/B</p>
CCR	<p>CCR C: Not affected. N: The content of bit 15 of ROM pointer. Z: 0</p>

## Transfer ②8

Expression	TFR ΔRO,B
Operation	<p>The contents of the ROM pointer (bit 10 through 15) are transferred to the corresponding bit position of ACCB. The content of the MSB of ROM pointer (bit 15) is also transferred to ACCB, though unused.</p> <p>ROM pointer: 15 10</p> <p>ACCB: 15 0</p> <p>Mantissa: 1 1 1 1 1 1 1 1 1 1</p> <p>Exponent: Undefined (bits 3-0)</p>
Instruction code	<p>21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</p> <p>1 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0</p> <p>OP code      Selects ACCA/B      Unused      Selects RAM pointer A/B</p>
CCR	<p>CCR C: Not affected. N: The content of bit 15 of ROM pointer. Z: 0</p>



## Transfer ①

Expression	TFR ΔRB,A
Operation	<p>The contents of the RAM pointer B (bit 10 through 15) are transferred to the corresponding bit position of ACCA.</p>
Instruction code	
CCR	<p>CCR C: Not affected.          N: The content of MSB of RAM pointer B.          Z: 0</p>

## Transfer ②

Expression	TFR ΔRB,B
Operation	<p>The contents of the RAM pointer B (bit 10 through 15) are transferred to the corresponding bit position of ACCB.</p>
Instruction code	
CCR	<p>CCR C: Not affected.          N: The content of MSB of RAM pointer B          Z: 0</p>

### Transfer ③③

Expression	TFR ΔCCR,A
Operation	<p>The contents of the CCR (bit 10 through 15) are transferred to the corresponding bit position of ACCA.</p> <p>CCR: 15 13 C N Z</p> <p>↓ ↓ ↓</p> <p>ACCA: 15 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 Mantissa</p> <p>3 0 Undefined Exponent</p>
Instruction code	<p>21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</p> <p>1 1 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0</p> <p>OP code      Selects ACCA/B      Unused      Selects RAM pointer A/B</p>
CCR	<p>CCR C: Not affected. N: The content of a carry flag. Z: 0</p>

### Transfer ③④

Expression	TFR ΔCCR,B
Operation	<p>The contents of the CCR (bit 10 through 15) are transferred to the corresponding bit position of ACCB.</p> <p>CCR: 15 13 C N Z</p> <p>↓ ↓ ↓</p> <p>ACCB: 15 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 Mantissa</p> <p>3 0 Undefined Exponent</p>
Instruction code	<p>21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</p> <p>1 1 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0</p> <p>OP code      Selects ACCA/B      Unused      Selects RAM pointer A/B</p>
CCR	<p>CCR C: Not affected. N: The content of a carry flag. Z: 0</p>

## Transfer ③⑤

Expression	TFR Δ SIR,A																																																															
Operation	<p>The contents of the SIR (bit 0 through 15) are transferred to the corresponding bit position of ACCA.</p> <p>The SIR is cleared after data transfer from the SIR to ACCA.</p>																																																															
Instruction code	<table style="border-collapse: collapse; margin: auto;"> <tr> <td style="text-align: center;">21</td><td style="text-align: center;">20</td><td style="text-align: center;">19</td><td style="text-align: center;">18</td><td style="text-align: center;">17</td><td style="text-align: center;">16</td><td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td> </tr> <tr> <td colspan="4" style="text-align: center;">OP code</td> <td colspan="4" style="text-align: center;">Selects ACCA/B</td> <td colspan="6" style="text-align: center;">Unused</td> <td colspan="4" style="text-align: center;">Selects RAM pointer A/B</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	OP code				Selects ACCA/B				Unused						Selects RAM pointer A/B			
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																											
1	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																										
OP code				Selects ACCA/B				Unused						Selects RAM pointer A/B																																																		
CCR	<p>CCR C: Not affected.          N: The content of the MSB of the SIR.          Z: Set if the IR is \$0000.</p>																																																															

## Transfer ③⑥

Expression	TFR Δ SIR,B																																																															
Operation	<p>The contents of the SIR (bit 0 through 15) are transferred to the corresponding bit position of ACCB.</p> <p>The SIR is cleared after data transfer from the SIR to ACCB.</p>																																																															
Instruction code	<table style="border-collapse: collapse; margin: auto;"> <tr> <td style="text-align: center;">21</td><td style="text-align: center;">20</td><td style="text-align: center;">19</td><td style="text-align: center;">18</td><td style="text-align: center;">17</td><td style="text-align: center;">16</td><td style="text-align: center;">15</td><td style="text-align: center;">14</td><td style="text-align: center;">13</td><td style="text-align: center;">12</td><td style="text-align: center;">11</td><td style="text-align: center;">10</td><td style="text-align: center;">9</td><td style="text-align: center;">8</td><td style="text-align: center;">7</td><td style="text-align: center;">6</td><td style="text-align: center;">5</td><td style="text-align: center;">4</td><td style="text-align: center;">3</td><td style="text-align: center;">2</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td> </tr> <tr> <td colspan="4" style="text-align: center;">OP code</td> <td colspan="4" style="text-align: center;">Selects ACCA/B</td> <td colspan="6" style="text-align: center;">Unused</td> <td colspan="4" style="text-align: center;">Selects RAM pointer A/B</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	OP code				Selects ACCA/B				Unused						Selects RAM pointer A/B			
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																											
1	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																										
OP code				Selects ACCA/B				Unused						Selects RAM pointer A/B																																																		
CCR	<p>CCR C: Not affected.          N: The content of the MSB of the SIR.          Z: Set if the IR is \$0000.</p>																																																															







# INCRO

Assembler syntax	[<label>] ΔINCRO [Δ <comment>]																																												
Operation	<p>This instruction increments ROM pointer.</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <math>(\text{ROM pointer}) + 1 \rightarrow (\text{ROM pointer})</math> </div> <p>Note: ROM pointer Model</p> <div style="margin-left: 40px;"> <table style="border-collapse: collapse;"> <tr> <td style="padding-right: 10px;">RO</td> <td style="border: 1px solid black; padding: 2px 5px;">15</td> <td style="border: 1px solid black; padding: 2px 5px;">14</td> <td style="border: 1px solid black; padding: 2px 5px;">13</td> <td style="border: 1px solid black; padding: 2px 5px;">12</td> <td style="border: 1px solid black; padding: 2px 5px;">11</td> <td style="border: 1px solid black; padding: 2px 5px;">10</td> </tr> <tr> <td></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> </tr> </table> <div style="margin-left: 100px;">▲... Decimal point</div> <div style="margin-left: 40px; margin-top: 5px;"> <span style="margin-right: 100px;">MSB</span> <span>LSB</span> </div> </div>	RO	15	14	13	12	11	10																																					
RO	15	14	13	12	11	10																																							
Instruction code	<table style="margin-left: 40px; border-collapse: collapse;"> <tr> <td style="padding-right: 5px;">21</td><td style="padding-right: 5px;">20</td><td style="padding-right: 5px;">19</td><td style="padding-right: 5px;">18</td><td style="padding-right: 5px;">17</td><td style="padding-right: 5px;">16</td><td style="padding-right: 5px;">15</td><td style="padding-right: 5px;">14</td><td style="padding-right: 5px;">13</td><td style="padding-right: 5px;">12</td><td style="padding-right: 5px;">11</td><td style="padding-right: 5px;">10</td><td style="padding-right: 5px;">9</td><td style="padding-right: 5px;">8</td><td style="padding-right: 5px;">7</td><td style="padding-right: 5px;">6</td><td style="padding-right: 5px;">5</td><td style="padding-right: 5px;">4</td><td style="padding-right: 5px;">3</td><td style="padding-right: 5px;">2</td><td style="padding-right: 5px;">1</td><td style="padding-right: 5px;">0</td> </tr> <tr> <td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td> </tr> </table> <div style="margin-left: 40px; margin-top: 5px;"> <span style="margin-right: 100px;">OP code</span> <span>Unused</span> <span style="margin-left: 100px;">Selects RAM pointer A/B</span> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0																								
CCR	CCR C: } N: } Not affected. Z: }																																												

# DECRA

Assembler syntax	[<label>] ΔDECRA [Δ <comment>]																																												
Operation	<p>This instruction decrements RAM pointer A.</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <math>(\text{RAM pointer A}) - 1 \rightarrow (\text{RAM pointer A})</math> </div> <p>Note: RAM pointer A Model</p> <div style="margin-left: 40px;"> <table style="border-collapse: collapse;"> <tr> <td style="padding-right: 10px;">RA</td> <td style="border: 1px solid black; padding: 2px 5px;">15</td> <td style="border: 1px solid black; padding: 2px 5px;">14</td> <td style="border: 1px solid black; padding: 2px 5px;">13</td> <td style="border: 1px solid black; padding: 2px 5px;">12</td> <td style="border: 1px solid black; padding: 2px 5px;">11</td> <td style="border: 1px solid black; padding: 2px 5px;">10</td> </tr> <tr> <td></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> </tr> </table> <div style="margin-left: 100px;">▲... Decimal point</div> <div style="margin-left: 40px; margin-top: 5px;"> <span style="margin-right: 100px;">MSB</span> <span>LSB</span> </div> </div>	RA	15	14	13	12	11	10																																					
RA	15	14	13	12	11	10																																							
Instruction code	<table style="margin-left: 40px; border-collapse: collapse;"> <tr> <td style="padding-right: 5px;">21</td><td style="padding-right: 5px;">20</td><td style="padding-right: 5px;">19</td><td style="padding-right: 5px;">18</td><td style="padding-right: 5px;">17</td><td style="padding-right: 5px;">16</td><td style="padding-right: 5px;">15</td><td style="padding-right: 5px;">14</td><td style="padding-right: 5px;">13</td><td style="padding-right: 5px;">12</td><td style="padding-right: 5px;">11</td><td style="padding-right: 5px;">10</td><td style="padding-right: 5px;">9</td><td style="padding-right: 5px;">8</td><td style="padding-right: 5px;">7</td><td style="padding-right: 5px;">6</td><td style="padding-right: 5px;">5</td><td style="padding-right: 5px;">4</td><td style="padding-right: 5px;">3</td><td style="padding-right: 5px;">2</td><td style="padding-right: 5px;">1</td><td style="padding-right: 5px;">0</td> </tr> <tr> <td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">1</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td><td style="border: 1px solid black; text-align: center;">0</td> </tr> </table> <div style="margin-left: 40px; margin-top: 5px;"> <span style="margin-right: 100px;">OP code</span> <span>Unused</span> <span style="margin-left: 100px;">Selects RAM pointer A/B</span> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																								
CCR	CCR C: } N: } Not affected. Z: }																																												

# DECRB

Assembler syntax	[<label>] ΔDECRB [Δ <comment>]
Operation	<p>This instruction decrements RAM pointer B.</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <math>(\text{RAM pointer B}) - 1 \rightarrow (\text{RAM pointer B})</math> </div> <p>Note: RAM pointer B Model</p> <div style="display: flex; align-items: center; margin-top: 10px;"> <div style="margin-right: 10px;">RB</div> <div style="display: flex; flex-direction: column; align-items: center;"> <div style="text-align: center; margin-bottom: 5px;">15 14 13 12 11 10</div> <div style="border: 1px solid black; width: 60px; height: 20px; display: flex; justify-content: space-around;"> <span style="font-size: 8px;">[ ]</span> <span style="font-size: 8px;">[ ]</span> <span style="font-size: 8px;">[ ]</span> <span style="font-size: 8px;">[ ]</span> <span style="font-size: 8px;">[ ]</span> <span style="font-size: 8px;">[ ]</span> </div> <div style="margin-top: 5px;">▲... Decimal Point</div> </div> <div style="margin-left: 10px; text-align: center;"> <div style="display: flex; justify-content: space-between; width: 60px;"> <span>MSB</span> <span>LSB</span> </div> </div> </div>
Instruction code	<div style="text-align: center; margin-bottom: 5px;">             21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0           </div> <div style="display: flex; justify-content: center; align-items: center; gap: 5px;"> <div style="border: 1px solid black; padding: 2px;">1</div> <div style="border: 1px solid black; padding: 2px;">1</div> <div style="border: 1px solid black; padding: 2px;">1</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">1</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">1</div> </div> <div style="display: flex; justify-content: space-around; margin-top: 5px;"> <div style="text-align: center; border-top: 1px solid black; width: 100px;">OP code</div> <div style="text-align: center; border-top: 1px solid black; width: 100px;">Unused</div> <div style="text-align: center; border-top: 1px solid black; width: 100px;">Selects RAM pointer A/B</div> </div>
CCR	CCR C: } N: } Not affected. Z: }

# DECRO

Assembler syntax	[<label>] ΔDECRO [Δ <comment>]
Operation	<p>This instruction decrements ROM pointer.</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <math>(\text{ROM pointer}) - 1 \rightarrow (\text{ROM pointer})</math> </div> <p>Note: ROM pointer Model</p> <div style="display: flex; align-items: center; margin-top: 10px;"> <div style="margin-right: 10px;">RO</div> <div style="display: flex; flex-direction: column; align-items: center;"> <div style="text-align: center; margin-bottom: 5px;">15 14 13 12 11 10</div> <div style="border: 1px solid black; width: 60px; height: 20px; display: flex; justify-content: space-around;"> <span style="font-size: 8px;">[ ]</span> <span style="font-size: 8px;">[ ]</span> <span style="font-size: 8px;">[ ]</span> <span style="font-size: 8px;">[ ]</span> <span style="font-size: 8px;">[ ]</span> <span style="font-size: 8px;">[ ]</span> </div> <div style="margin-top: 5px;">▲... Decimal Point</div> </div> <div style="margin-left: 10px; text-align: center;"> <div style="display: flex; justify-content: space-between; width: 60px;"> <span>MSB</span> <span>LSB</span> </div> </div> </div>
Instruction code	<div style="text-align: center; margin-bottom: 5px;">             21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0           </div> <div style="display: flex; justify-content: center; align-items: center; gap: 5px;"> <div style="border: 1px solid black; padding: 2px;">1</div> <div style="border: 1px solid black; padding: 2px;">1</div> <div style="border: 1px solid black; padding: 2px;">1</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">1</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">1</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> <div style="border: 1px solid black; padding: 2px;">0</div> </div> <div style="display: flex; justify-content: space-around; margin-top: 5px;"> <div style="text-align: center; border-top: 1px solid black; width: 100px;">OP code</div> <div style="text-align: center; border-top: 1px solid black; width: 100px;">Unused</div> <div style="text-align: center; border-top: 1px solid black; width: 100px;">Selects RAM pointer A/B</div> </div>
CCR	CCR C: } N: } Not affected. Z: }

# DECRC

Assembler syntax	[<label>] ΔDECRC [Δ <comment>]																																																														
Operation	<p>This instruction decrements the RC.</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;">(RC)-1 → (RC)</div> <p>Note: RC Model</p> <div style="text-align: center; margin: 10px auto;"> <table style="border-collapse: collapse;"> <tr> <td></td> <td style="text-align: center;">15</td> <td style="text-align: center;">14</td> <td style="text-align: center;">13</td> <td style="text-align: center;">12</td> <td style="text-align: center;">11</td> <td style="text-align: center;">10</td> <td></td> </tr> <tr> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> <td style="border: 1px solid black; width: 15px; height: 15px;"></td> </tr> <tr> <td style="text-align: left;">RC</td> <td colspan="6"></td> <td style="text-align: right;">▲... Decimal Point</td> </tr> <tr> <td></td> <td colspan="3" style="text-align: center;">MSB</td> <td colspan="4"></td> <td style="text-align: center;">LSB</td> </tr> </table> </div>		15	14	13	12	11	10										RC							▲... Decimal Point		MSB							LSB																													
	15	14	13	12	11	10																																																									
RC							▲... Decimal Point																																																								
	MSB							LSB																																																							
Instruction code	<table style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td style="width: 20px;">21</td><td style="width: 20px;">20</td><td style="width: 20px;">19</td><td style="width: 20px;">18</td><td style="width: 20px;">17</td><td style="width: 20px;">16</td><td style="width: 20px;">15</td><td style="width: 20px;">14</td><td style="width: 20px;">13</td><td style="width: 20px;">12</td><td style="width: 20px;">11</td><td style="width: 20px;">10</td><td style="width: 20px;">9</td><td style="width: 20px;">8</td><td style="width: 20px;">7</td><td style="width: 20px;">6</td><td style="width: 20px;">5</td><td style="width: 20px;">4</td><td style="width: 20px;">3</td><td style="width: 20px;">2</td><td style="width: 20px;">1</td><td style="width: 20px;">0</td> </tr> <tr> <td style="border: 1px solid black;">1</td><td style="border: 1px solid black;">1</td><td style="border: 1px solid black;">1</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">1</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">1</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td> </tr> <tr> <td colspan="8" style="text-align: center;">OP code</td> <td colspan="6" style="text-align: center;">Unused</td> <td colspan="4" style="text-align: center;">Selects RAM pointer A/B</td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	OP code								Unused						Selects RAM pointer A/B			
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																										
1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																										
OP code								Unused						Selects RAM pointer A/B																																																	
CCR	CCR C: } N: } Not affected. Z: }																																																														

# RTI

Assembler syntax	[<label>] ΔRTI [Δ <comment>]																																																														
Operation	<p>This instruction performs a return from interrupt. The PC data is recovered from STACK.</p> <div style="text-align: center; margin: 10px auto;"> <table style="border-collapse: collapse;"> <tr> <td style="border: 1px solid black; padding: 5px;">PC (9 bits)</td> <td style="width: 20px; text-align: center;">←</td> <td style="border: 1px solid black; padding: 5px;">STACK 0</td> <td style="width: 20px; text-align: center;">←</td> <td style="border: 1px solid black; padding: 5px;">STACK 1</td> </tr> </table> </div> <p>The RTI instruction clears Interrupt mask flag (I<sub>M</sub>) of the STR, which enables an interrupt. Even if an interrupt is requested during execution of this instruction, it is not accepted before the next instruction is completely executed. (Interrupt wait state)</p>	PC (9 bits)	←	STACK 0	←	STACK 1																																																									
PC (9 bits)	←	STACK 0	←	STACK 1																																																											
Instruction code	<table style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td style="width: 20px;">21</td><td style="width: 20px;">20</td><td style="width: 20px;">19</td><td style="width: 20px;">18</td><td style="width: 20px;">17</td><td style="width: 20px;">16</td><td style="width: 20px;">15</td><td style="width: 20px;">14</td><td style="width: 20px;">13</td><td style="width: 20px;">12</td><td style="width: 20px;">11</td><td style="width: 20px;">10</td><td style="width: 20px;">9</td><td style="width: 20px;">8</td><td style="width: 20px;">7</td><td style="width: 20px;">6</td><td style="width: 20px;">5</td><td style="width: 20px;">4</td><td style="width: 20px;">3</td><td style="width: 20px;">2</td><td style="width: 20px;">1</td><td style="width: 20px;">0</td> </tr> <tr> <td style="border: 1px solid black;">1</td><td style="border: 1px solid black;">1</td><td style="border: 1px solid black;">1</td><td style="border: 1px solid black;">1</td><td style="border: 1px solid black;">1</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td><td style="border: 1px solid black;">0</td> </tr> <tr> <td colspan="8" style="text-align: center;">OP code</td> <td colspan="6" style="text-align: center;">Unused</td> <td colspan="4"></td> </tr> </table>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	OP code								Unused									
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																										
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																										
OP code								Unused																																																							
CCR	CCR C: } N: } Not affected. Z: }																																																														

# RTN

Assembler syntax	[<label>] ΔRTN [Δ <comment>]																																												
Operation	<p>This instruction performs a return from subroutine. The PC data is recovered from STACK.</p> <div data-bbox="341 378 931 427" style="text-align: center;"> <pre> graph LR     S1[STACK 1] --&gt; S0[STACK 0]     S0 --&gt; PC[PC (9 bits)]             </pre> </div> <p>Even if an interrupt is requested during execution of this instruction, it is not accepted before the next instruction is completely executed. (Interrupt wait state)</p>																																												
Instruction code	<div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr> <td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td> </tr> </table> </div> <div style="text-align: center; margin-top: 5px;"> <span style="margin-right: 100px;">OP code</span> <span>Unused</span> </div>	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0																								
CCR	CCR C: } N: } Not affected. Z: }																																												

### 5.2.1 Description of Operand

<Operand (A)> Pointer Addressing Mode

Expression	<mnemonic> $\Delta$ <operand (A)>																																																	
Example	$\frac{FADA}{\text{mnemonic}} \Delta \frac{YA}{\text{①}}, \frac{A}{\text{②}}, \frac{XY(1,3)}{\text{③}}, \frac{RA,RO+}{\text{④}}$																																																	
Function	<p>① ALU operation (selection of inputs)</p> <table border="1"> <thead> <tr> <th rowspan="2">Symbol</th> <th rowspan="2">ALU Operation</th> <th rowspan="2">Note</th> <th colspan="2">Assigned Bits</th> </tr> <tr> <th>16</th> <th>15</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>(P, ACC) <math>\rightarrow</math> ACC</td> <td>Either ACCA or ACCB is selected.</td> <td>0</td> <td>0</td> </tr> <tr> <td>YA</td> <td>(Y, ACC) <math>\rightarrow</math> ACC</td> <td>Either ACCA or ACCB is selected.</td> <td>0</td> <td>1</td> </tr> <tr> <td>PX</td> <td>(P, X) <math>\rightarrow</math> ACC</td> <td>Either ACCA or ACCB is selected.</td> <td>1</td> <td>0</td> </tr> <tr> <td>YX</td> <td>(Y, X) <math>\rightarrow</math> ACC</td> <td>Either ACCA or ACCB is selected.</td> <td>1</td> <td>1</td> </tr> </tbody> </table> <p>( ) : Arithmetic operation  P : multiplier product with pipeline delay of previous instruction cycle (<math>P_{n+1}</math>)  X : X-Bus output  Y : Y-Bus output</p> <p>For details of arithmetic operation, see description of each instruction.</p> <p>② Write to the data RAM (selection of source register)</p> <table border="1"> <thead> <tr> <th rowspan="2">Symbol</th> <th rowspan="2">Write Operation</th> <th rowspan="2">Note</th> <th colspan="2">Assigned Bits</th> </tr> <tr> <th>12</th> <th>11</th> </tr> </thead> <tbody> <tr> <td>EE</td> <td>No write</td> <td></td> <td>0</td> <td>0</td> </tr> <tr> <td>A</td> <td>ACC <math>\rightarrow</math> M(Y)</td> <td>The accumulator contents is the result of the previous ALU operation. The same accumulator as ① is selected.</td> <td>0</td> <td>1</td> </tr> <tr> <td>D</td> <td>DREG <math>\rightarrow</math> M(Y)</td> <td>The contents of the DREG is the output from the Y-Bus during the previous instruction.</td> <td>1</td> <td>1</td> </tr> </tbody> </table> <p>Y: Memory address specified by ③ and ④.</p> <p>Data is written to the data RAM through the Y-Bus.</p>	Symbol	ALU Operation	Note	Assigned Bits		16	15	PA	(P, ACC) $\rightarrow$ ACC	Either ACCA or ACCB is selected.	0	0	YA	(Y, ACC) $\rightarrow$ ACC	Either ACCA or ACCB is selected.	0	1	PX	(P, X) $\rightarrow$ ACC	Either ACCA or ACCB is selected.	1	0	YX	(Y, X) $\rightarrow$ ACC	Either ACCA or ACCB is selected.	1	1	Symbol	Write Operation	Note	Assigned Bits		12	11	EE	No write		0	0	A	ACC $\rightarrow$ M(Y)	The accumulator contents is the result of the previous ALU operation. The same accumulator as ① is selected.	0	1	D	DREG $\rightarrow$ M(Y)	The contents of the DREG is the output from the Y-Bus during the previous instruction.	1	1
Symbol	ALU Operation				Note	Assigned Bits																																												
		16	15																																															
PA	(P, ACC) $\rightarrow$ ACC	Either ACCA or ACCB is selected.	0	0																																														
YA	(Y, ACC) $\rightarrow$ ACC	Either ACCA or ACCB is selected.	0	1																																														
PX	(P, X) $\rightarrow$ ACC	Either ACCA or ACCB is selected.	1	0																																														
YX	(Y, X) $\rightarrow$ ACC	Either ACCA or ACCB is selected.	1	1																																														
Symbol	Write Operation	Note	Assigned Bits																																															
			12	11																																														
EE	No write		0	0																																														
A	ACC $\rightarrow$ M(Y)	The accumulator contents is the result of the previous ALU operation. The same accumulator as ① is selected.	0	1																																														
D	DREG $\rightarrow$ M(Y)	The contents of the DREG is the output from the Y-Bus during the previous instruction.	1	1																																														

③ Read and Write to the Data ROM/RAM/GR  
(Specify the page addresses)

Symbol	Read		Write	Assigned Bit			
	X-Bus	Y-Bus	Y-Bus	10	9	876	543
XY(n,m)	Output of ROM or RAM → X-Bus	Output of ROM or RAM → Y-Bus	Y-Bus → RAM	1	0	n	m
XG(n,l)	Output of ROM or RAM → X-Bus	Output of GR → Y-Bus	Y-Bus → GR	1	1	n	l

(Note) The data in the data ROM/RAM/GR are read out to two buses simultaneously, while a write into data RAM/GR is performed through Y-Bus only.

n : page address of X-Bus output data (RAM: 0-3, ROM: 4-7)

m : page address of Y-Bus output data (RAM: 0-3, ROM: 4-7)

l : the GR address (0-3)

n must be equal to m if both of them specify the ROM address (4-7) concurrently.

④ ROM/RAM Pointers

Effective memory addresses are generated by ③ and ④.

Symbol	Selection of RAM Pointer	Autoincrement	Autodecrement	Assigned Bit		
				2	1	0
RA[,RO]	A	-	-	0	0	0
RB[,RO]	B	-	-	0	0	1
RA+[,RO]	A	Increment of RAM pointer A	Decrement of RC	1	0	0
RB+[,RO]	B	Increment of RAM pointer B	Decrement of RC	1	0	1
RA,RO+	A	Increment of ROM pointer	Decrement of RC	0	1	0
RB,RO+	B	Increment of ROM pointer	Decrement of RC	0	1	1
RA+,RO+	A	Increment of RAM Pointer A & ROM pointer	Decrement of RC	1	1	0
RB+,RO+	B	Increment of RAM Pointer B & ROM pointer	Decrement of RC	1	1	1

(Note) The RC is autodecremented depending on the result of logical OR of bit 2 and 1.

$2^2V2^1 = 0$  : not decremented

$2^2V2^1 = 1$  : decremented

Precaution

A write to the data RAM ( ② ) is performed after a read of the memory address specified by ③ and ④.

<Operand (B) > Direct Addressing Mode

Expression	<mnemonic>Δ<operand (B) >																																																									
Example	<u>FADA</u> Δ <u>PA</u> , <u>A</u> , <u>0</u> , <u>12</u> ① ② ③																																																									
Function	<p>① ALU operation (selection of ALU inputs)</p> <table border="1"> <thead> <tr> <th rowspan="2">Symbol</th> <th rowspan="2">ALU Operation</th> <th rowspan="2">Note</th> <th colspan="2">Assigned Bits</th> </tr> <tr> <th>16</th> <th>15</th> </tr> </thead> <tbody> <tr> <td>PA</td> <td>(P, ACC) → ACC</td> <td>Either ACCA or ACCB is selected.</td> <td>0</td> <td>0</td> </tr> <tr> <td>YA</td> <td>(Y, ACC) → ACC</td> <td>Either ACCA or ACCB is selected.</td> <td>0</td> <td>1</td> </tr> <tr> <td>PX</td> <td>(P, X) → ACC</td> <td>Either ACCA or ACCB is selected.</td> <td>1</td> <td>0</td> </tr> <tr> <td>YX</td> <td>(Y, X) → ACC</td> <td>Either ACCA or ACCB is selected.</td> <td>1</td> <td>1</td> </tr> </tbody> </table> <p>② Write to the Data RAM (selection of source register)</p> <table border="1"> <thead> <tr> <th rowspan="2">Symbol</th> <th rowspan="2">Write Operation</th> <th rowspan="2">Note</th> <th colspan="2">Assigned Bits</th> </tr> <tr> <th>12</th> <th>11</th> </tr> </thead> <tbody> <tr> <td>EE</td> <td>No write</td> <td></td> <td>0</td> <td>0</td> </tr> <tr> <td>A</td> <td>ACC → M(n,m)</td> <td>The accumulator contents is the result of the previous ALU operation. The same accumulator as ① is selected.</td> <td>0</td> <td>1</td> </tr> <tr> <td>D</td> <td>DREG → M(n,m)</td> <td>The contents of the DREG is the output from the Y-Bus during the previous instruction.</td> <td>1</td> <td>1</td> </tr> </tbody> </table> <p>n and m are specified by ③. Data is written to the Data RAM through the Y-Bus.</p> <p>③ Read and Write to the Data ROM/RAM (specify the ROM/RAM address)</p> <table border="1"> <thead> <tr> <th rowspan="2">Symbol</th> <th>Read</th> <th>Write</th> </tr> <tr> <th>Output of X-Bus and Y-Bus</th> <th>Y-Bus</th> </tr> </thead> <tbody> <tr> <td>n,m</td> <td>Output of ROM or RAM → Y-Bus Output of the accumulators → X-Bus</td> <td>Y-Bus → RAM</td> </tr> </tbody> </table> <p>n: page address (RAM: 0-3, ROM: 4-7) m: pointer address (RAM: 0-49, ROM: 0-31)</p> <p>This type of instructions do not affect the contents of the RAM pointers (RA, RB) and the ROM pointer (RO).</p>	Symbol	ALU Operation	Note	Assigned Bits		16	15	PA	(P, ACC) → ACC	Either ACCA or ACCB is selected.	0	0	YA	(Y, ACC) → ACC	Either ACCA or ACCB is selected.	0	1	PX	(P, X) → ACC	Either ACCA or ACCB is selected.	1	0	YX	(Y, X) → ACC	Either ACCA or ACCB is selected.	1	1	Symbol	Write Operation	Note	Assigned Bits		12	11	EE	No write		0	0	A	ACC → M(n,m)	The accumulator contents is the result of the previous ALU operation. The same accumulator as ① is selected.	0	1	D	DREG → M(n,m)	The contents of the DREG is the output from the Y-Bus during the previous instruction.	1	1	Symbol	Read	Write	Output of X-Bus and Y-Bus	Y-Bus	n,m	Output of ROM or RAM → Y-Bus Output of the accumulators → X-Bus	Y-Bus → RAM
Symbol	ALU Operation				Note	Assigned Bits																																																				
		16	15																																																							
PA	(P, ACC) → ACC	Either ACCA or ACCB is selected.	0	0																																																						
YA	(Y, ACC) → ACC	Either ACCA or ACCB is selected.	0	1																																																						
PX	(P, X) → ACC	Either ACCA or ACCB is selected.	1	0																																																						
YX	(Y, X) → ACC	Either ACCA or ACCB is selected.	1	1																																																						
Symbol	Write Operation	Note	Assigned Bits																																																							
			12	11																																																						
EE	No write		0	0																																																						
A	ACC → M(n,m)	The accumulator contents is the result of the previous ALU operation. The same accumulator as ① is selected.	0	1																																																						
D	DREG → M(n,m)	The contents of the DREG is the output from the Y-Bus during the previous instruction.	1	1																																																						
Symbol	Read	Write																																																								
	Output of X-Bus and Y-Bus	Y-Bus																																																								
n,m	Output of ROM or RAM → Y-Bus Output of the accumulators → X-Bus	Y-Bus → RAM																																																								



<Operand (C)> Pointer Addressing Mode

Expression	<mnemonic> Δ <operand (C)>
Example	$\frac{\text{FABSA}}{\text{mnemonic}} \Delta \frac{\text{A}}{\text{(2)}}, \frac{\text{XY(1,2)}}{\text{(3)}}, \frac{\text{RA,RO}}{\text{(4)}}$
Function	Refer to (2), (3) and (4) in <Operand (A)> In the operand (C), ALU input data is limited to ACC contents. Therefore, the description of the operand (1) is omitted.

<Operand (D)> Direct Addressing Mode

Expression	<mnemonic> Δ <operand (D)>
Example	$\frac{\text{FNOPA}}{\text{mnemonic}} \Delta \frac{\text{A}}{\text{(2)}}, \frac{\text{0,00}}{\text{(3)}}$
Function	Refer to (2) and (3) in <Operand (B)> In the operand (D), ALU input data is limited to ACC contents. Therefore, the description of the operand (1) is omitted.

### 5.2.2 HSP Internal Data Flow

The HSP instructions permit a read or write of data memory (ROM/RAM/GR) and arithmetic operation in a single instruction cycle. The product and the ALU output are used in the next instruction cycle. Fig. 5.2.1 and Fig. 5.2.2 give examples of data flow for an easy understanding.

#### (1) Pointer addressing mode

ex. FADA  $\Delta$  YA , A , XY(1,3) , RA,RO+  
          ①    ②            ③            ④

① YA : Output of Y-Bus + ACCA  $\rightarrow$  ACCA

② A : Write

ACCA  $\rightarrow$  M(Y) Y is an address specified by the RAM pointer in page 3.

③ XY(1,3) : Read

RAM output (page 1)  $\rightarrow$  X-Bus

RAM output (page 3)  $\rightarrow$  Y-Bus

④ RA,RO+ : The RAM pointer A is selected.

The ROM pointer is incremented.

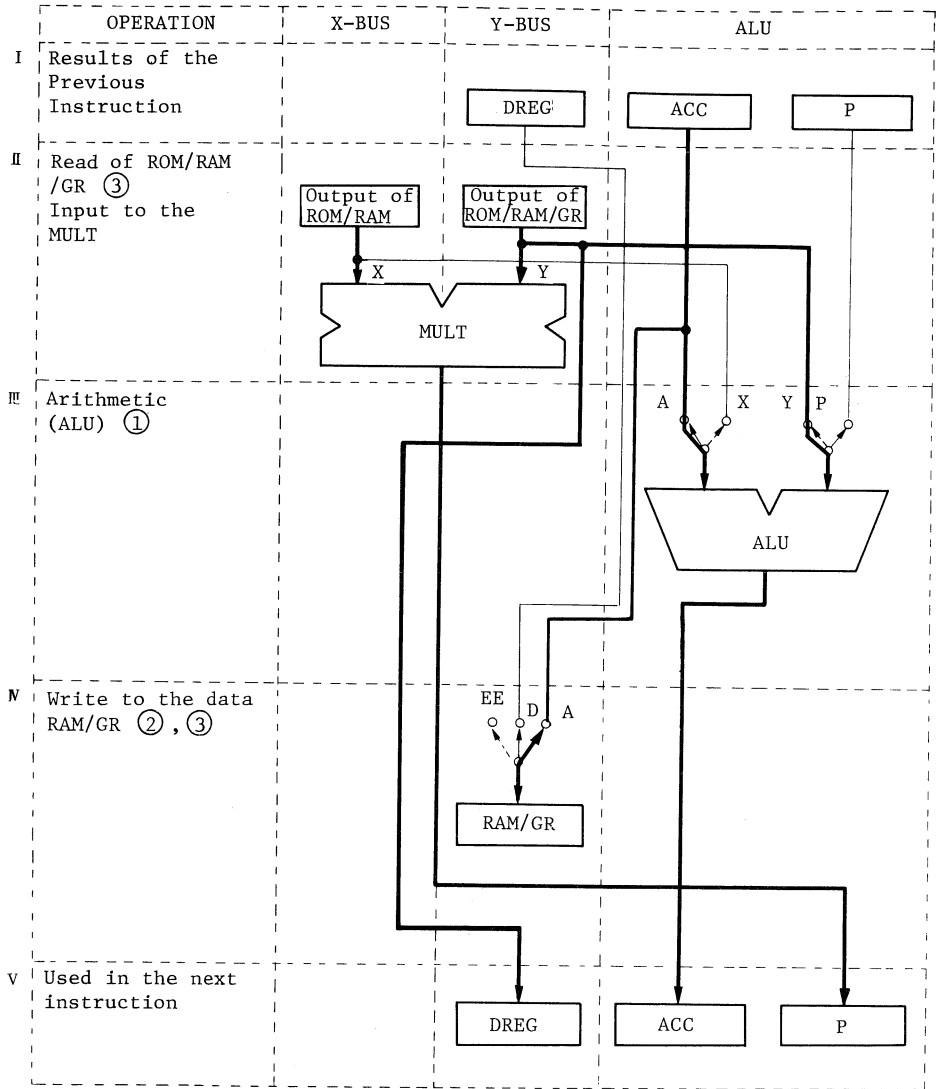
The RC is autodecremented.

#### <Explanation of Flowchart>

- I. The outputs of the MULT, the ALU and the Y-Bus of the previous instruction cycle are loaded in the P (product), an accumulator and the DREG respectively.
- II. Operand ③ (XY(1,3)) indicates that the data in page 1 is output to the X-Bus and the data in page 3 to the Y-Bus. In this case, the RAM pointer A is specified by operand ④. Data which is output onto the X-Bus and Y-Bus are then transmitted to the MULT and the product will be transferred to the P register. As the MULT always operates independently of instructions, two read-out data are inevitably multiplied. The data output to the Y-Bus is also transferred to the DREG.
- III. Operand ① (YA) indicates that the output from the Y-Bus and ACCA contents (the results of the previous instruction) are added in the ALU and a result is stored in ACCA.

The ALU has two inputs: either output of the accumulator or of the X-Bus is selected for one of ALU input, and either output of the P or the Y-Bus for the other. In this case, 'YA' selects and the ACCA output and the Y-Bus data for these two inputs. They are added and the result is stored in ACCA.

- IV. ② specifies a data to be written to the data RAM. The HSP has no store instructions in mnemonic, and operand ② controls a write to the data RAM. The data to be written is the accumulator or DREG contents of the previous instruction cycle. In this case, 'A' selects the accumulator as a source register. The data is written into the address of the data output to the Y-Bus specified in ③ for a read operation (page 3).
- V. The data written to the DREG, accumulator and P register in the above procedures can be used during the next instruction cycle.



The bold lines show the data flow described in the example.

①, ② and ③ are the operand numbers. The contents of the P and the DREG are reserved for a single instruction cycle.

Fig. 5.2.1 HSP DATA FLOW (IN POINTER ADDRESSING MODE)

(2) Direct addressing mode

ex. FADA  $\Delta$  PA, A, 0,12  
          ①   ②   ③

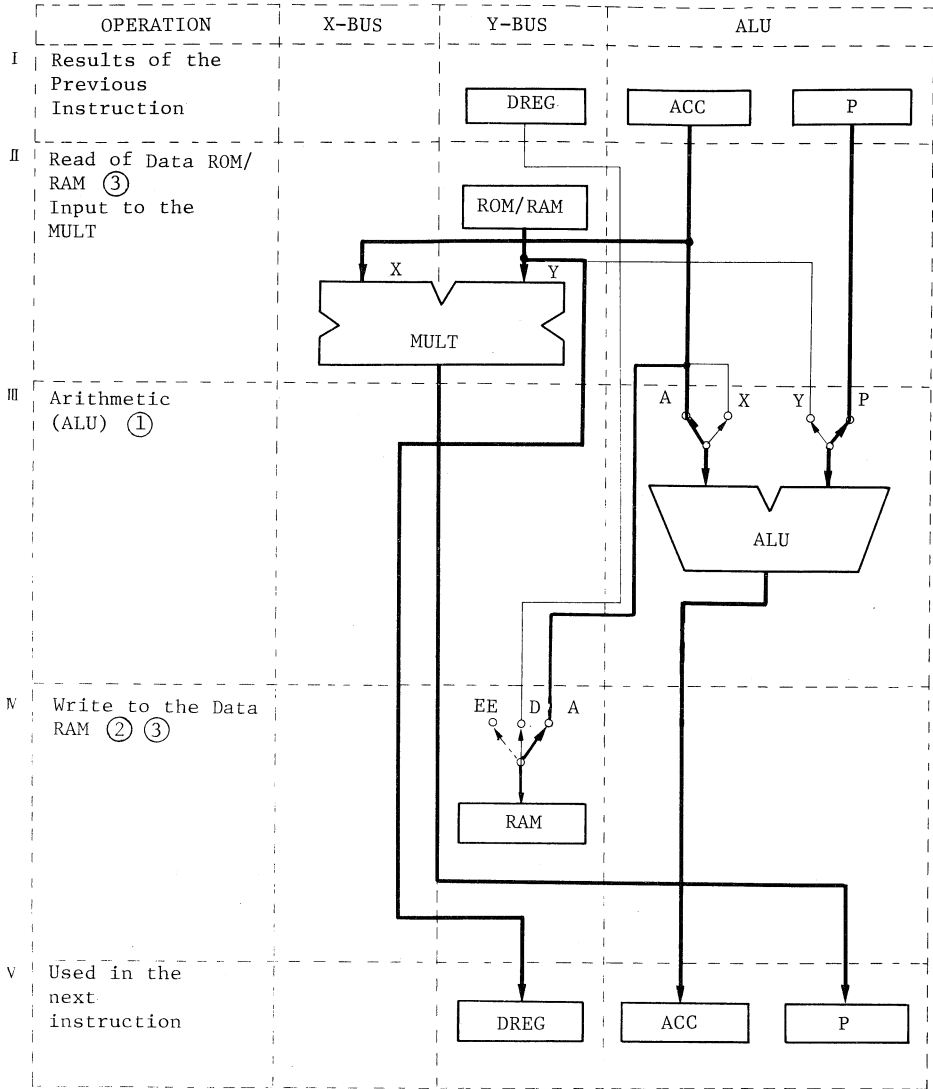
① PA : P (product) + ACCA  $\rightarrow$  ACCA

② A : Write  
      ACCA  $\rightarrow$  M(0,12)

③ 0,12 : Read  
      The data in location 12 of page 0  $\rightarrow$  Y-Bus

<Explanation of Flowchart>

- I. The product (output of the MULT) and the outputs of the ALU and the Y-bus of the previous instruction cycle are stored in the P register, accumulator and the DREG respectively.
- II. ③ (0,12) indicates that the data in location 12 of page 0 is output to the Y-bus. The contents of the accumulator appears on the X-bus. These X-bus data and Y-bus data are transferred to the MULT, which always operates independently of instructions, and then the product will be transferred to the P. The data output to the Y-bus is also transferred to the DREG.
- III. ① (PA) indicates that the contents of the P and ACCA (the results of the previous instruction) are added in the ALU and then a result is stored in ACCA. The ALU has two inputs: one is input from an accumulator, and either output of the P or of Y-Bus is selected for the other input. In this case, 'PA' selects the P output. These two inputs are added and then a result from the operation is stored in ACCA.
- IV. ② specifies a data to be written to the data RAM. The HSP has no store instructions in mnemonic, and operand ② controls a write to the data RAM. The data to be written is the accumulator or DREG contents of the previous instruction cycle. In this case, 'A' selects the accumulator contents. The data is written into the address of the data output to the Y-bus specified in ③ for a read operation (location 12 of page 0).
- V. The data written in the DREG and the P in the above procedures can be used during the next instruction cycle.



The bold lines show the data flow described in the example.

The contents of the P and the DREG are reserved for a single instruction cycle.

Fig. 5.2.2 HSP DATA FLOW (IN DIRECT ADDRESSING MODE)

### 5.3 PIPELINE CONTROL

The HSP is designed to support a high-speed product sum operation by employing the highly pipelined architecture. The detailed description of pipelined operation is given in the following example of program sequence.

Example:

Suppose that the following arithmetic is executed in the HSP.

$$Y_i = Y_{i-1} + C_{i-1} \cdot X_{i-1}, X_{i-1} \rightarrow X_i$$

Then its program is:

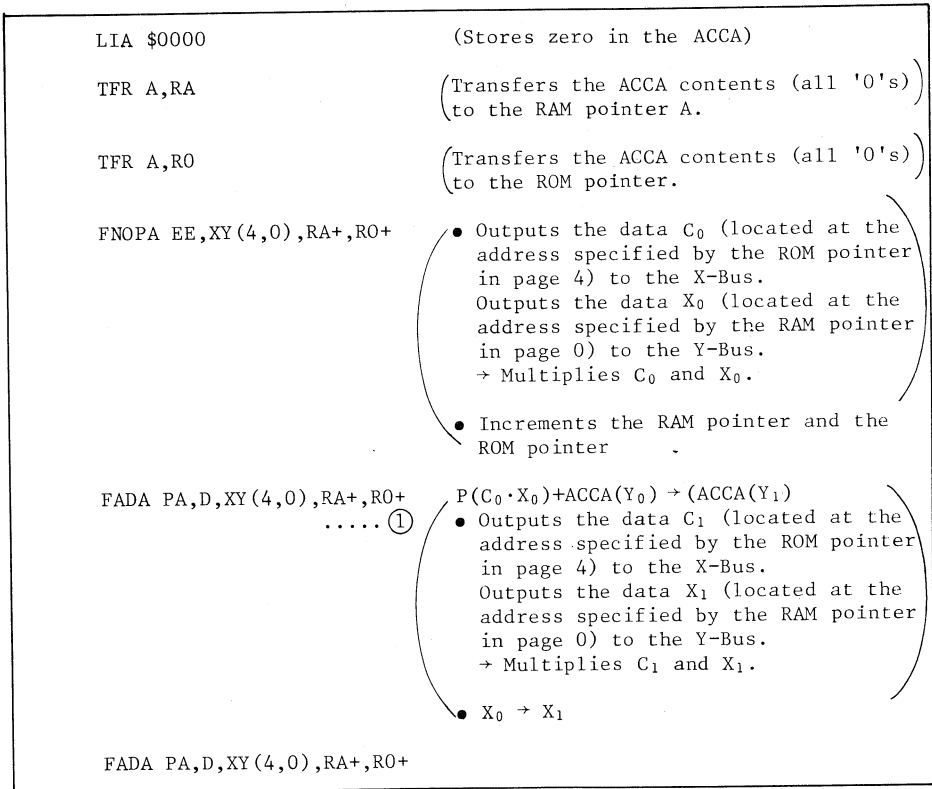


Fig. 5.3.1 shows the sequence of this example.

Instruction ① adds the previous product ( $C_0 \cdot X_0$ ) to the contents of the ACCA ( $Y_0$ ) and transfers its result to the ACCA ( $Y_1$ ), while data  $C_1$  and  $X_1$  are read from the data ROM or the data RAM on the X-Bus and the Y-Bus respectively, and are multiplied. Data  $X_1$  is also stored in the DREG and is written into the location of  $X_2$  which after  $X_2$  is read out.

As described above, the HSP appears to take only a single instruction cycle (250ns) for product-sum operation owing to the pipelined multiplication and ALU operation.

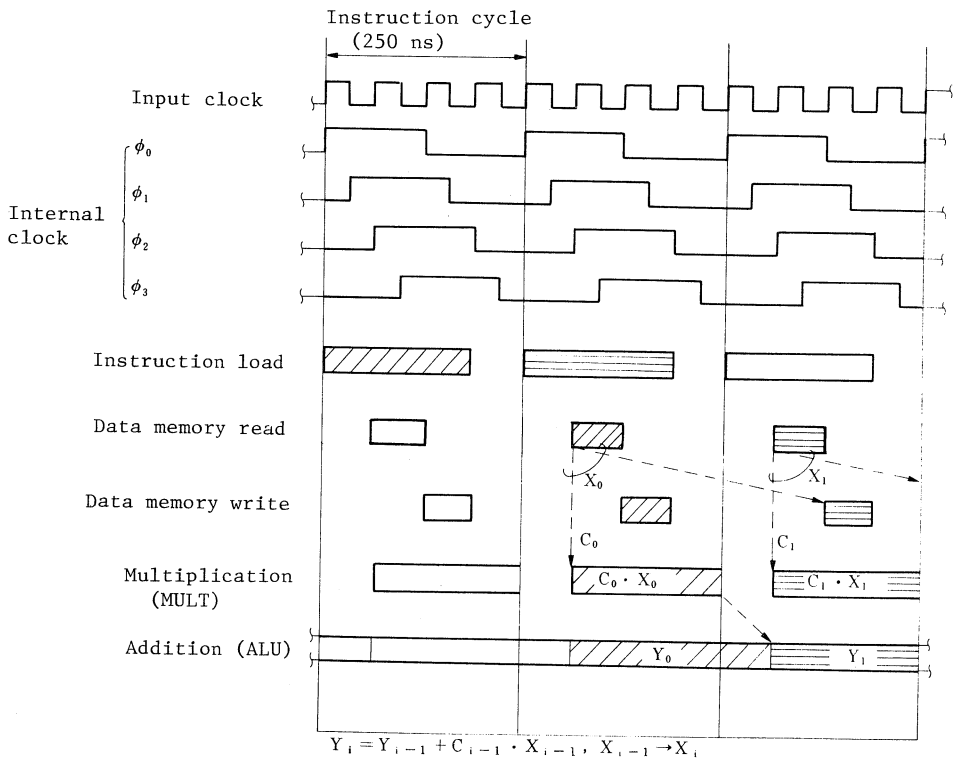


Fig. 5.3.1 PIPELINE CONTROL



Fig. 5.3.2 shows the sequence of the each register in a single instruction cycle.

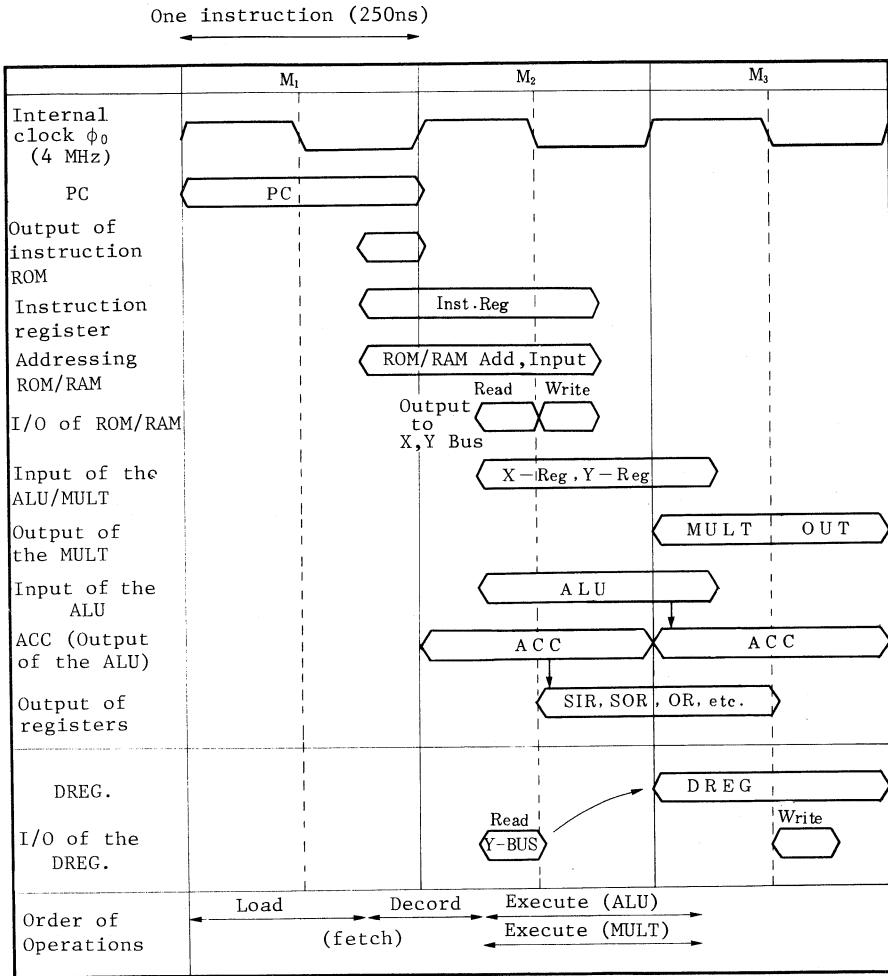


Fig. 5.3.2 PIPELINED OPERATION



**SECTION 6**  
**PROGRAMING TECHNIQUE**



## 6. PROGRAMMING TECHNIQUE

### 6.1 BIQUAD FILTER

This section describes an example of Biquad Filter.

In this example, a 16-bit two's complement data is output from the SIR and dealt in the arithmetic operation in the fixed point representation and finally set to the SOR. The program of Fig. 6.1.4 illustrates operations in a single sampling period and the filter output can be acquired by repeating this program every sampling period. In the Biquad Filter, one stage consists of 6 steps. As each step takes 250 ns in the HSP, one stage takes 1.5  $\mu$ s (250 ns  $\times$  6), which permits a max. 82 stages of filter in case of 8 kHz sampling. However, a single filtering stage needs four coefficients from the data ROM of 128 words, which limits the number of stages to 32.

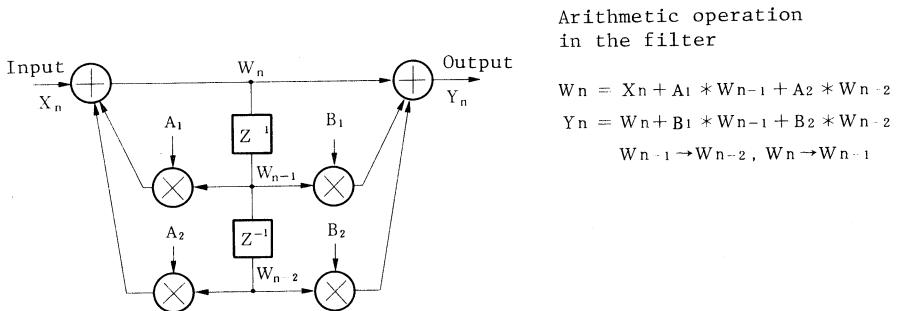


Fig. 6.1.1 A STAGE OF BIQUAD FILTER

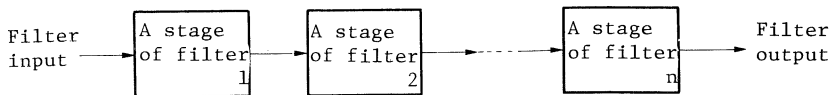


Fig. 6.1.2 BIQUAD FILTERS IN A SAMPLING PERIOD

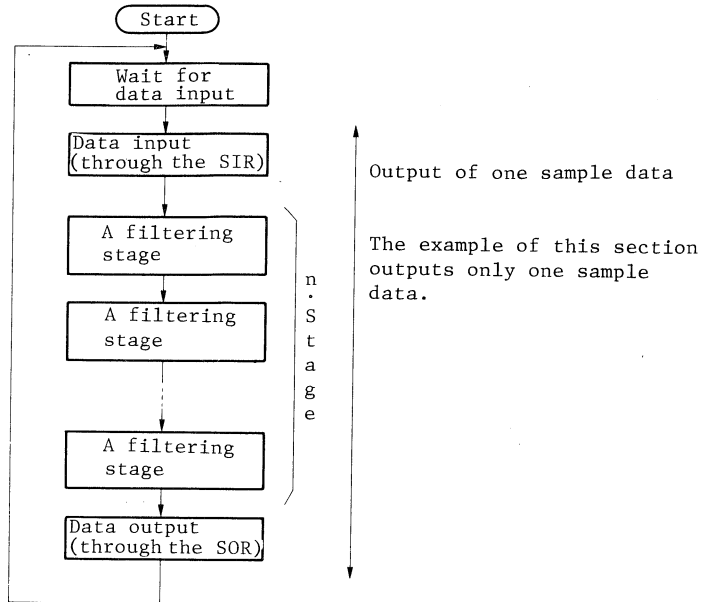


Fig. 6.1.3 A SAMPLING PROCESS WITH BIQUAD FILTER

```

LIRA      0      ; 0 → RAM pointer
LIRO      0      ; 0 → ROM pointer
TFR       SIR,A  ; SIR → ACCA
LIRC      n      ; n → RC
L1 NOPA EE,XY(4,0),RA ; A1*Wn-1
ADA PA,EE,XY(5,1),RA ; A2*Wn-2 ACCA+P → ACCA
ADA PA,EE,XY(7,1),RA ; B2*Wn-2 ACCA+P → ACCA
ADA PA,A,XY(6,0),RA ; B1*Wn-1 ACCA+P → ACCA,
                    Wn-1 → DREG, ACCA → Wn-1
ADA PA,D,XY(7,1),RA+,RO+ ; DREG → Wn-2' ACCA+P → ACCA,
                    RC DEC
                    RAM POINTER INC,ROM POINTER INC
JNZ       L1      ; Jump to L1 if (RC)≠0
TFR       A, SOR  ; ACCA → SOR
  
```

Fig. 6.1.4 AN EXAMPLE OF PROGRAM SEQUENCE OF BIQUAD FILTER

Table 6.1.1 MEMORY MAP

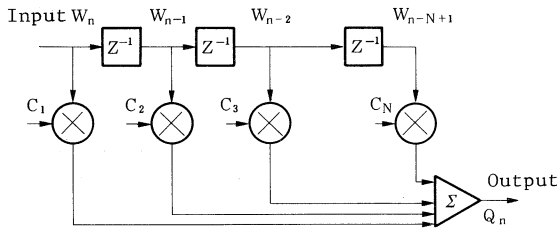
Data memory		Page address							
		0	1	2	3	4	5	6	7
Pointer address	00	$W_n - 1$	$W_n - 2$			$A_1$	$A_2$	$B_1$	$B_2$
	01	$W_n' - 1$	$W_n' - 2$			$A_1'$	$A_2'$	$B_1'$	$B_2'$
	02	$W_n'' - 1$	$W_n'' - 2$			$A_1''$	$A_2''$	$B_1''$	$B_2''$
	03								

## 6.2 TRANSVERSAL FILTER

This section describes a design of Transversal Filter.

In this example, a fixed point 16-bit two's complement data is output from the SIR and transformed to the floating point data to be dealt in the arithmetic operation and finally input to the SOR after transformed to the fixed pointer data.

A sampling operation containing 32 taps of Transversal Filter is performed in sampling period of 10.5  $\mu$ s.



Arithmetic operation in the filter

$$Q_n = \sum_{i=1}^N C_i * W_{n-i+1}$$

$$W_{n-i+1} \rightarrow W_{n-i} \quad (i=1 \sim N)$$

$$N = 32$$

Fig. 6.2.1 TRANSVERSAL FILTER

LIRO	0	; 0 $\rightarrow$ ROM POINTER
LIRA	0	; 0 $\rightarrow$ RAM POINTER
LIRC	31	; 31 $\rightarrow$ RC
TFR	SIR, A	; SIR $\rightarrow$ ACCA
FLTA	EE, 6, 00	; Fixed $\rightarrow$ Floating
FCLRA	A, 0, 00	; 0 $\rightarrow$ ACCA, ACCA $\rightarrow$ $W_n$
FRPTA	EE, 0, 00	; Next inst. repeat
FADA	PA, D, XY(4, 0), RA+, RO+	; ACCA + $C_i * W_{n-i+1}$ , $W_{n-i+1} \rightarrow W_{n-i}$ RC DEC
FADA	PA, EE, 0, 00	; ACCA + $C_{32} * W_{n-32} * W_{n-31}$
FIXA	EE, 6, 00	; Floating $\rightarrow$ Fixed
TFR	A, SOR	; ACCA $\rightarrow$ SOR

Fig. 6.2.2 AN EXAMPLE OF PROGRAM SEQUENCE OF TRANSVERSAL FILTER

Table 6.2.1 MEMORY MAP

Data memory		Page address							
		0	1	2	3	4	5	6	7
Pointer address	00	$W_n$				$C_1$		*	
	01	$W_{n-1}$				$C_2$			
	02	$W_{n-2}$				$C_3$			
	⋮	⋮				⋮			
	⋮	⋮				⋮			
	31	$W_{n-32+1}$				$C_{32}$			

\* Scaling constant



**SECTION 7**  
**HSP APPLICATION**

**7**



## 7. HSP APPLICATION

The HSP provides the instruction ROM, data ROM and data RAM, which permits stand-alone operation. Moreover, the HSP can be used as a peripheral device of an 8 or 16 bit microcomputer using parallel I/O ports which supports large systems.

This section describes two cases when the HSP is used as a stand-alone system and as a peripheral LSI of an 8-bit microcomputer.

- Stand-alone system

Stand-alone system is illustrated in Fig. 7.1.

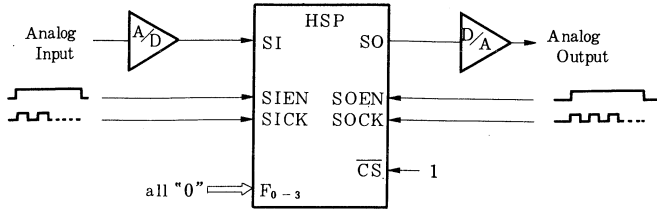
Stand-alone system permits filtering operation through the serial I/O ports. In this case,  $\overline{CS}$  must be set to high to disable parallel I/O ports. For details of unused pins in stand-alone operation, see 'The Handling of HSP Unused Pins' at the end of 2.1.

- Connection with an 8-bit microcomputer HD6800

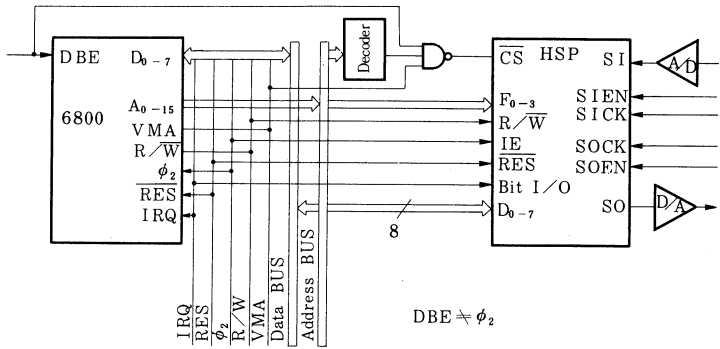
A connection of the HSP and an 8-bit microcomputer HD6800 is illustrated in Fig. 7.1. The HSP is designed to interface with the HD6800 through parallel I/O ports. An microcomputer permits the several HSPs to be operated at the same time.

While the  $\overline{CS}$  pin is active ( $\overline{CS}=0$ ), the status of F0 to F3 and  $R/\overline{W}$  must be fixed to high or low. The unfixed status of F0 to F3 and  $R/\overline{W}$  may cause a malfunction. The cycle time of a microcomputer should be 1  $\mu$ s ( $f=1$  MHz).

BIT I/O and TxRQ can be used as interrupt signal to a microcomputer.



(a) Stand-alone system



(b) Connection with an 8-bit microcomputer

Fig. 7.1 SYSTEM CONFIGURATION

• Connection with an 8-bit microcomputer HD6303

Fig. 7.4 gives an example of the system consisting of the MPU (HD6303R), the HSP (HD61810) and memory.

The HSP provides an interface with the MPU and memory through serial input ports and parallel I/O ports. The serial output port is not used.

• Bus timing of parallel I/O

The bus timing of the HD6303R is 1 MHz. The multiplex mode is used. The Chip Select ( $\overline{CS}$ ) signal of the HSP can be also used as the address strobe ( $\overline{AS}$ ) signal of the HD6303R. The data transfer of the PC and the CTR through the HSP parallel I/O ports takes 2 cycle time because  $\overline{CS}$  needs adequate time to set up.

● Bus timing of serial input

A 12-bit A/D convertor of 10 kHz sampling is shown in Fig. 7.4. Though Fig. 7.4 shows only an A/D convertor, the actual A/D convertor needs a sample and hold circuit additionally.

If the input is not two's complement data, it needs a transformation by software. Moreover, if the input data is less than 16 bits, the lower bits should be cleared to '0' by software.

Constant frequency 200 kHz must be provided for serial clock. SIEN signal is generated from the EOC (End of Conversion) signal.

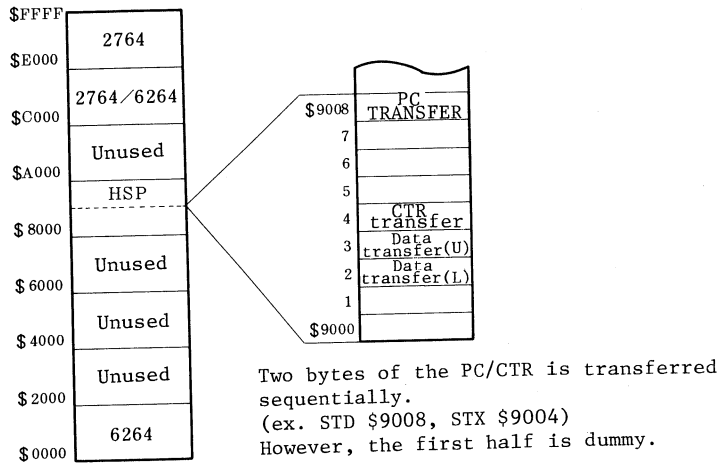
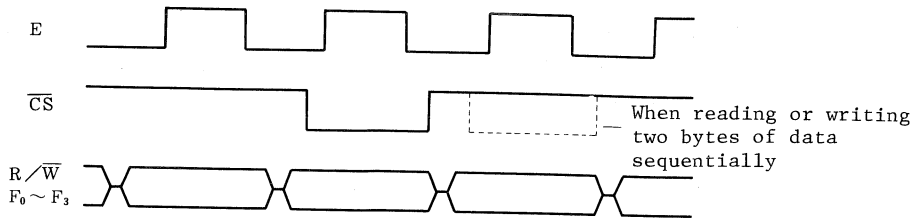
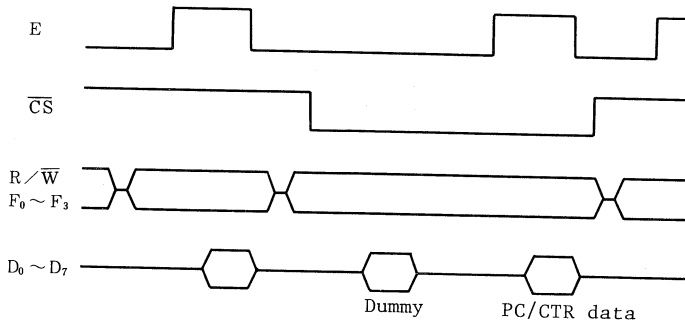


Fig. 7.2 HD6303 ADDRESS MAP IN THE APPLICATION CIRCUIT

When data is input/output



When the PC/CTR is written



The MPU writes two bytes of data in the PC/CTR sequentially (ex, STD, STX)

Fig. 7.3 BUS TIMING

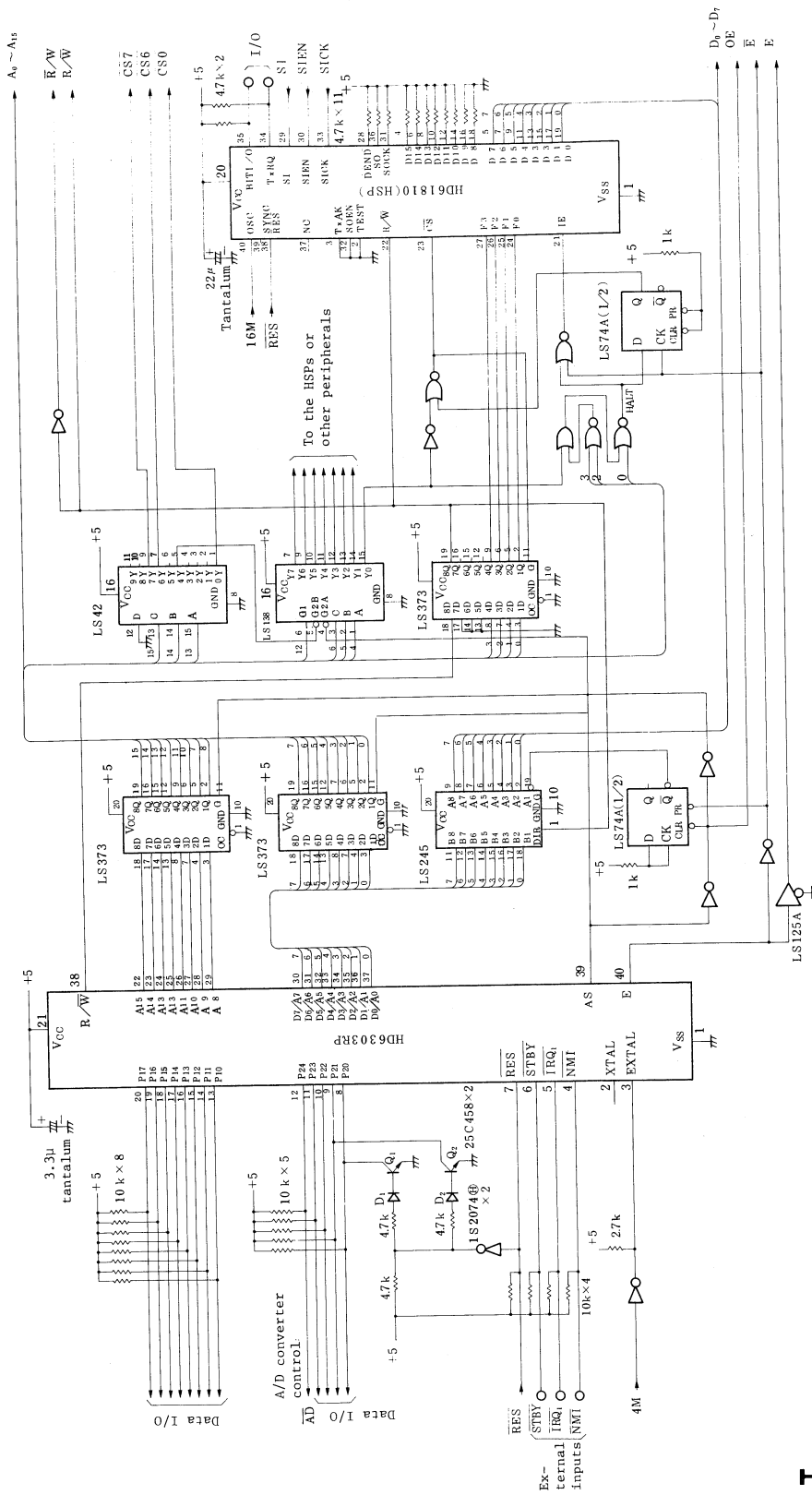
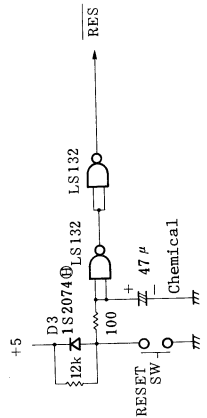


Fig. 7.4 APPLICATION CIRCUIT DIAGRAM

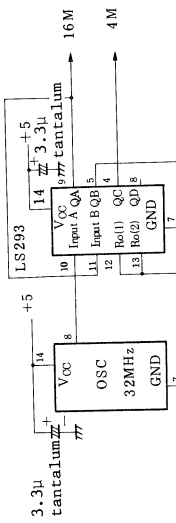




Reset circuit



Internal Oscillator



OSC: Kinseki Ltd. CXO-043B  
Nihon Denpa Kogyo Co., Ltd. TD308A 8M

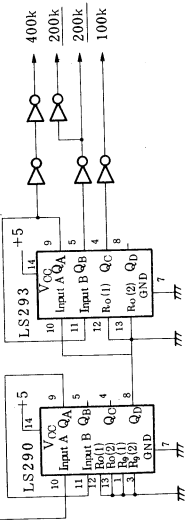
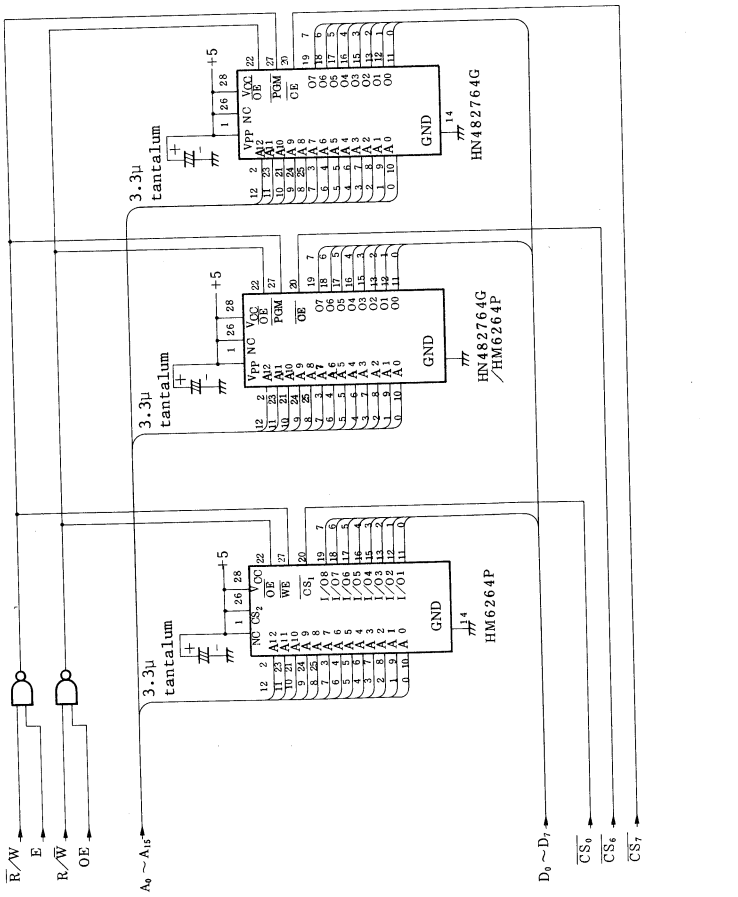


Fig. 7.4 APPLICATION CIRCUIT DIAGRAM (Cont'd)







**SECTION 8**  
**ELECTRICAL**  
**CHARACTERISTICS**



## 8. ELECTRICAL CHARACTERISTICS

### 8.1 ABSOLUTE MAXIMUM RATING

Item	Symbol	Value	Unit	Note
Supply Voltage	$V_{CC}$	-0.3 ~ 7.0	V	
Input Voltage	$V_{in}$	-0.3 ~ $V_{CC} + 0.3$	V	
Operating Temperature Range	$T_{opr}$	0 ~ +70	°C	
Storage Temperature Range	$T_{stg}$	-55 ~ +150	°C	DIC Package
		-55 ~ +125	°C	DIP, PLCC Package

### 8.2 ELECTRICAL CHARACTERISTICS

■ DC Characteristics ( $V_{CC}=5.0V\pm 5\%$ ,  $V_{SS}=0V$ ,  $T_a=0\sim 70^\circ C$  unless otherwise specified.)

Item		Symbol	Test Condition	Min	Typ	Max	Unit
Input "High" Voltage	OSC, IE, SICK, SOCK	$V_{IH}$		2.4	-	$V_{CC}+0.3$	V
	All others			2.2	-	$V_{CC}+0.3$	V
Input "Low" Voltage	OSC, IE, SICK, SOCK	$V_{IL}$		-0.3	-	0.4	V
	All others			-0.3	-	0.8	V
Input Leak Current	TEST, TxAK, IE, R/W, CS, $F_0\sim F_3$ , DEND, SI SIEN, SOCK, SOEN SICK, RES, OSC	$ I_{IN} $	$V_{IN}=0.4\sim 2.4V$	-	-	10	$\mu A$
Three State Current (OFF State)	$D_0\sim D_{15}$ , SO	$ I_{TSI} $	$V_{IN}=0.4\sim 2.4V$	-	-	10	$\mu A$
Open Drain Current (OFF State)	TxRQ, BIT I/O	$ I_{LOH} $	$V_{IN}=0.4\sim 2.4V$	-	-	10	$\mu A$
Output "High" Voltage	$D_0\sim D_{15}$ , SO, SYNC	$V_{CH}$	$-I_{OH}=400\mu A$	2.4	-	-	V
Output "Low" Voltage	All output pins	$V_{OL}$	$I_{OL}=1.6\text{ mA}$	-	-	0.8	V
Input Capacitance	All output pins	$C_{IN}$	$V_{IN}=0V$ , $f=1\text{MHz}$ , $T_a=25^\circ C$	-	-	12.5	pF
Current Dissipation		$I_{CC}$	Not port loading	-	50	100	mA

■ AC Characteristics

- System Clock ( $V_{CC}=5.0V\pm 5\%$ ,  $V_{SS}=0V$ ,  $T_a=0\sim 70^\circ C$  unless otherwise specified.)

Item	Symbol	Test Condition	Min	Typ	Max	Unit
Clock (OSC) cycle	$\phi_{cyc}$	Fig. 8-1	61.5	62.5	70.0	ns
Clock (OSC) Pulse Width	$\phi_{WH}$		20	-	-	ns
	$\phi_{WL}$		20	-	-	ns
Clock (OSC) Rise Time	$\phi_r$		-	-	10	ns
Clock (OSC) Fall Time	$\phi_f$		-	-	10	ns

- Serial I/O Timing ( $V_{CC}=5.0V\pm 5\%$ ,  $V_{SS}=0V$ ,  $T_a=0\sim 70^\circ C$  unless otherwise specified.)

Item	Symbol	Test Condition	Min	Typ	Max	Unit
Clock (SICK, SOCK) cycle	$S_{cyc}$	Fig. 8-2 Fig. 8-5	1.0	-	10.0	$\mu s$
Clock (SICK, SOCK) Pulse Width	$S_{WH}$		450	-	-	ns
	$S_{WL}$		450	-	-	ns
Clock (SICK, SOCK) Rise Time	$S_r$		-	-	25	ns
Clock (SICK, SOCK) Fall Time	$S_f$		-	-	25	ns
Serial Input Data Set-up Time	$t_{SDS}$		100	-	-	ns
Serial Input Data Hold Time	$t_{SDH}$		100	-	-	ns
Serial Output Data Delay Time	$t_{SDD}$		-	-	300	ns
Enable Delay Time	$t_{ED}$		50	-	-	ns
Enable Set-up Time	$t_{ES}$	100	-	-	ns	



- Parallel I/O (Bus Interface) Timing ( $V_{CC}=5.0V\pm 5\%$ ,  $V_{SS}=0V$ ,  $T_a=0\sim +70^\circ C$  unless otherwise specified.)

Item	Symbol	Test Condition	Min	Typ	Max	Unit
IE cycle	$t_{cyc}$	Fig. 8-3 Fig. 8-5	1.0	-	10.0	$\mu s$
IE Pulse Width	$t_{WH}$		450	-	-	ns
	$t_{WL}$		450	-	-	ns
IE Rise Time	$t_r$		-	-	25	ns
IE Fall Time	$t_f$		-	-	25	ns
$\overline{CS}$ Set-up Time	$t_{CS}$		140	-	-	ns
$\overline{CS}$ Hold Time	$t_{CH}$		10	-	-	ns
Address Set-up Time	$t_{AC}$		10	-	-	ns
Address Hold Time	$t_{CA}$		20	-	-	ns
Input Data Set-up Time	$t_{DSW}$		190	-	-	ns
Input Data Hold Time	$t_{DHW}$		10	-	-	ns
Output Data Delay Time	$t_{DDR}$		-	-	220	ns
Output Data Hold Time	$t_{DHR}$		10	-	-	ns

- DMA Interface Timing ( $V_{CC}=5.0V\pm 5\%$ ,  $V_{SS}=0V$ ,  $T_a=0\sim +70^\circ C$  unless otherwise specified.)

Item	Symbol	Test Condition	Min	Typ	Max	Unit
TxAk Set-up Time	$t_{AS}$	Fig. 8-4 Fig. 8-5	140	-	-	ns
TxAk Hold Time	$t_{AH}$		-	-	600	ns
TxRQ Delay Time	$t_{TR}$		-	-	470	ns

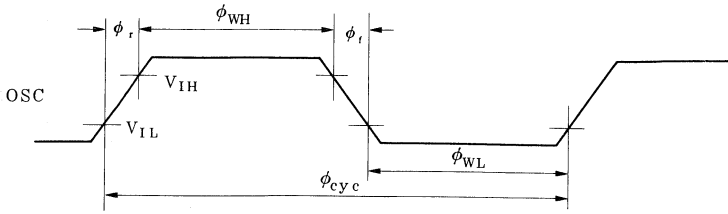
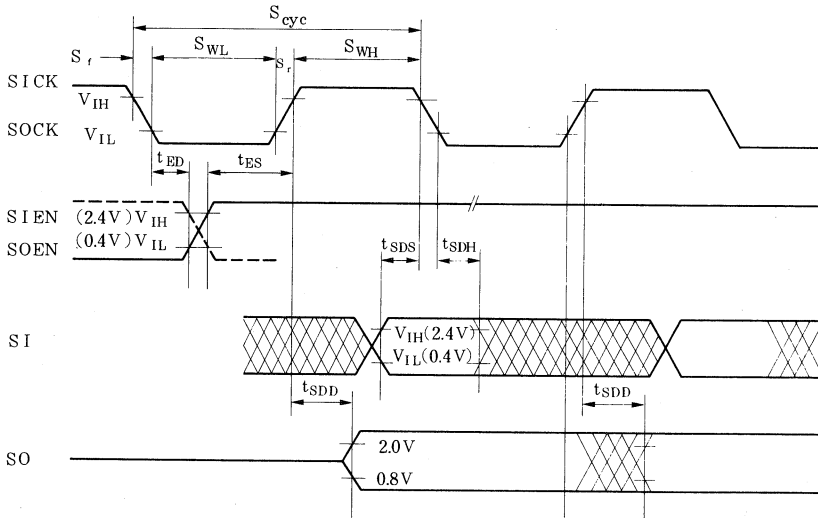
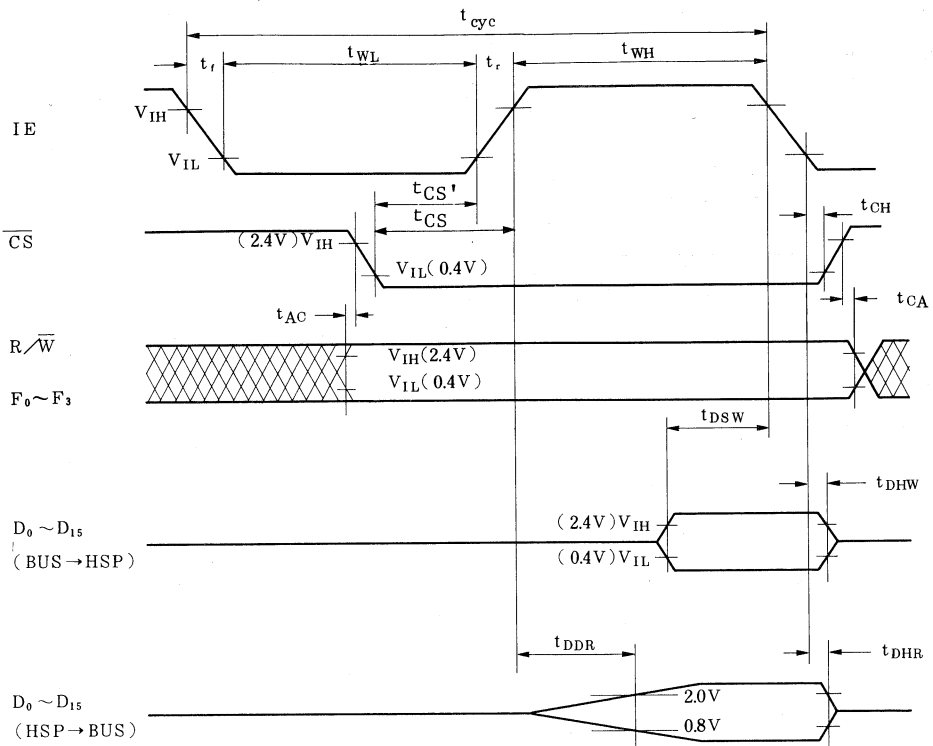


Fig. 8.1 SYSTEM CLOCK WAVEFORM



(Note) The SO pin goes to the high-impedance state by latching SOEN with SOCK.

Fig. 8.2 SERIAL I/O WAVEFORM



(Note 1) Keep  $t_{CS}$  (min. 140ns) with CTR/PC transfer instructions.

The data transfer instruction using the IR/OR takes  $t_{CS}'$  of more than 10 ns.

(Note 2) The data bus output in the byte transfer mode is 16 bit ( $D_0 \sim D_{15}$ ).

In this case, the upper half ( $D_8 \sim D_{15}$ ) is not valid. Therefore,  $D_8 \sim D_{15}$  should not be directly connected to  $V_{CC}/GND$ .

Fig. 8.3 PARALLEL I/O TIMING

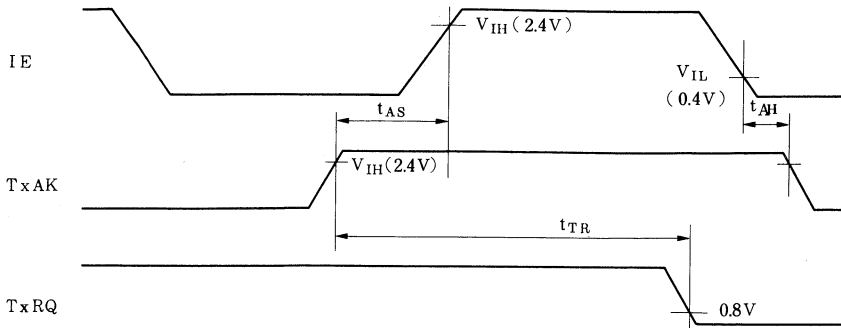
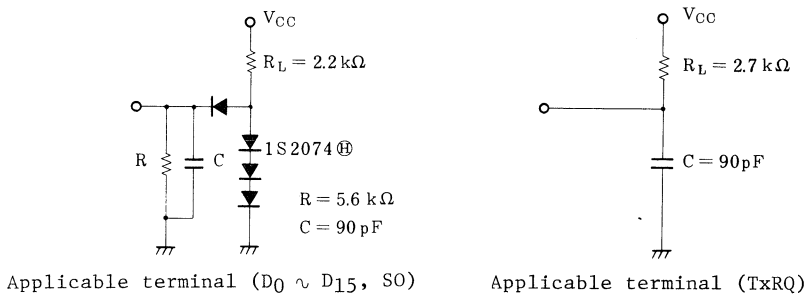
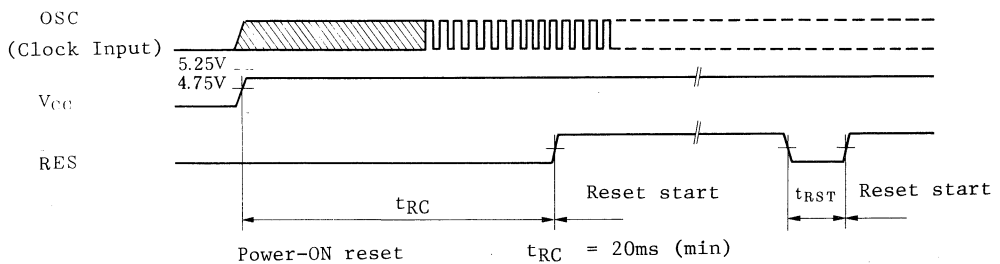


Fig. 8.4 DMA TIMING



C : includes probe and jig capacitance

Fig. 8.5 LOAD CIRCUIT (FOR TIMING TEST)

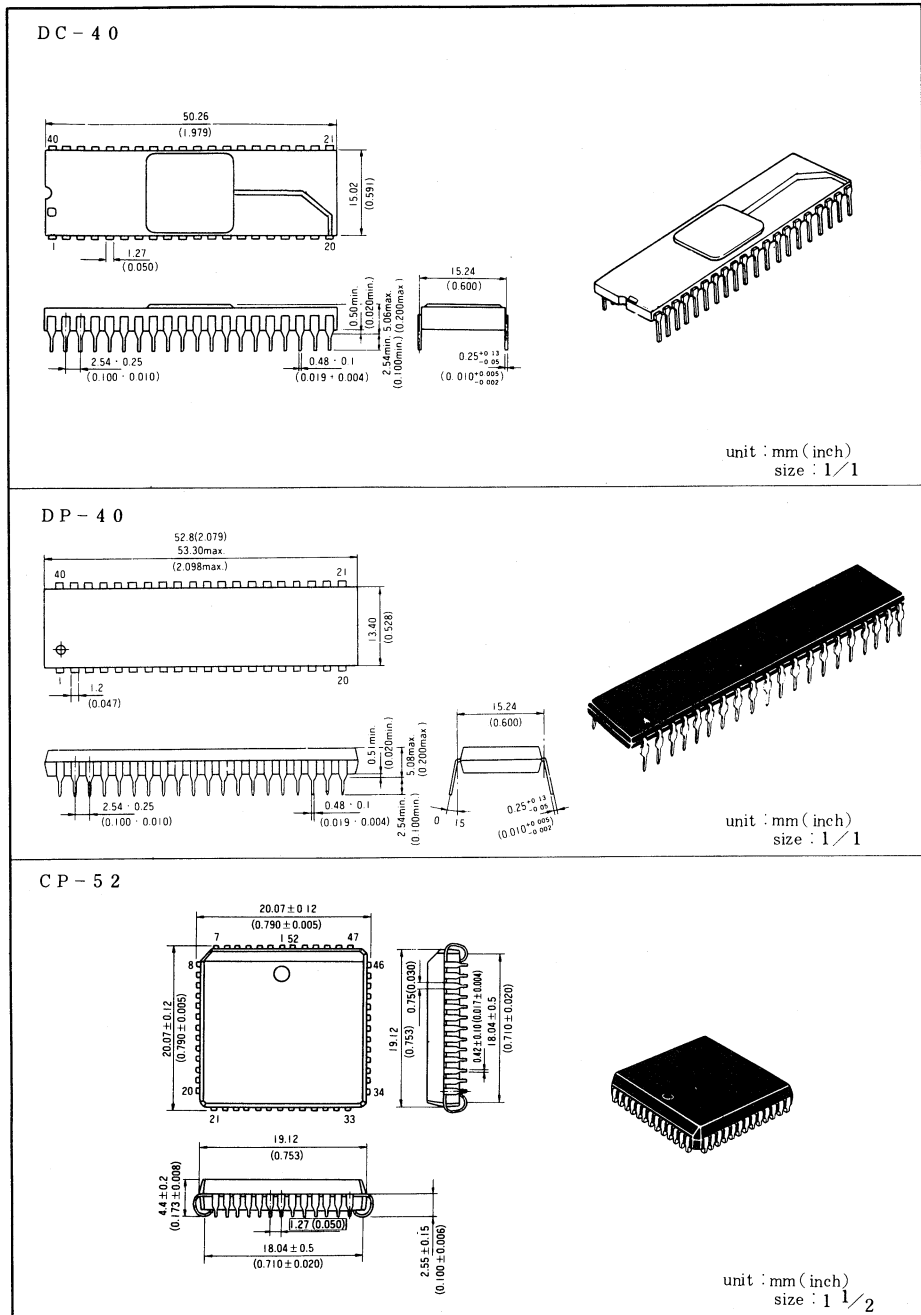


Power-ON reset  $t_{RC} = 20\text{ms (min)}$   
 Reset during operation  $t_{RST} = 1\mu\text{s (min)}$   
 (Note) Clock (16 MHz) should be input during  $t_{RST}$  or the power-ON rest start.

Fig. 8.6 RESET TIMING

### 8.3 PACKAGE OUTLINE

The HSP provides three kinds of packages as follows;



NOTE) Inch value indicated for your reference.



# APPENDIX

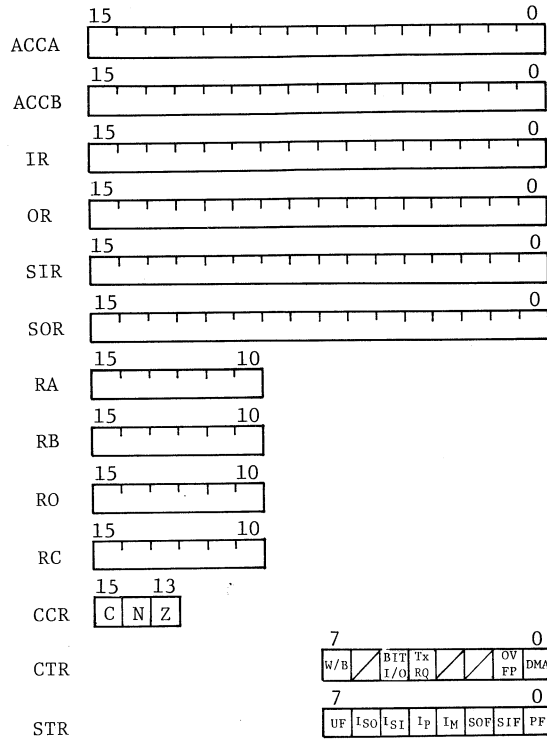
- 1 HSP REGISTER MODEL
- 2 INSTRUCTION CODE
- 3 INSTRUCTION SET
- 4 ASSEMBLER SYNTAX
- 5 HSP INSTRUCTION SUMMARY
- 6 MEMORY MAP
- 7 HSP ORDERING SPECIFICATION
- 8 TEST PROGRAM



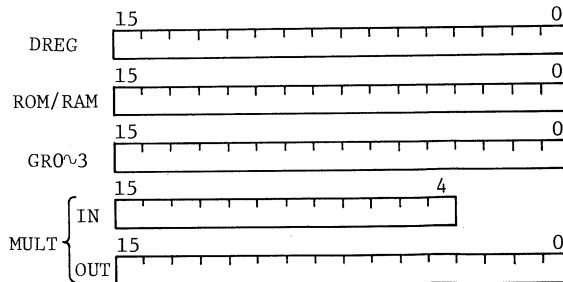


# 1. HSP REGISTER MODEL

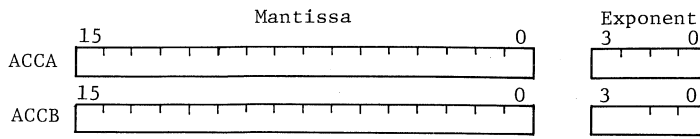
## (1) Fixed Point



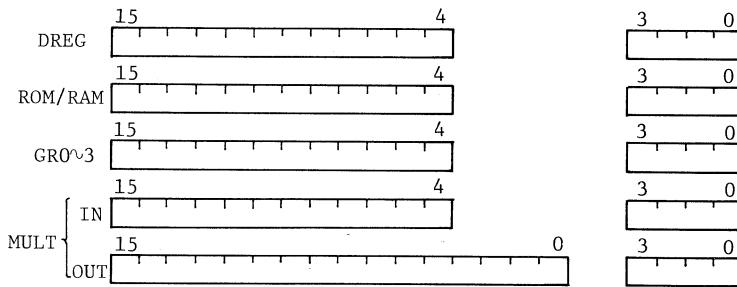
I/O register, General purpose register, ROM/RAM corresponding bit



(2) Floating point



I/O register, General purpose register, ROM/RAM corresponding bit



## 2. INSTRUCTION CODE

### I, I' ALU Operation Instruction (Pointer Addressing Mode)

OP																Address Control															
OP Code																ALU Input															
																Selects FL/FX															
0; ACC, 1; X																Selects ACC A/B															
0; P, 1; Y																Control of Memory Write															
0; FL, 1; FX																1															
0; ACCA, 1; ACCB																Control of memory I/O															
0; ACC-M(Y), 1; D-M(Y)																ROM/RAM Page															
0; Not write, 1; Write																ROM/RAM/GR Page															
0; X·Y, 1; X·G																RAM pointer Increment															
X-page (3 bits)																ROM pointer Increment															
Y-page (3 bits)																Selects RAM pointer A/B															
0; Not Inc. 1; Inc.																															
0; Not Inc. 1; Inc.																															
0; RA, 1; RB																															

→ M(Y) is ROM/RAM/GR

RC post decrement  
(bit 1 ∨ bit 2=1)

### ALU Operation Instruction (Direct Addressing Mode)

OP																Address Control															
OP Code																ALU Input															
																Selects FL/FX															
0; ACC, 1; X																Control of Memory Write															
0; P, 1; Y																0															
0; FL, 1; FX																0															
0; ACCA, 1; ACCB																Page Address (3 bits)															
0; ACC-M(Y), 1; D-M(Y)																Pointer Address (6 bits)															
0; Not write, 1; Write																															
0																															
0																															

→ M(Y) is ROM/RAM/GR.

## II. Immediate Instructions

21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP Code						Immediate Data for ACC A/B (16 bits)															

21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP Code						Immediate data for the RC, ROM/RAM pointer (6 bits)						/									

## III. Jump Instruction

21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP Code						Jump Condition						Jump Address									
						C	N	Z	/	/	/										/

- o Jumps if (jump condition)  $\wedge$  (CCR)  $\neq$  0
- o In case of unconditional jump, jump condition bits are all '0'.

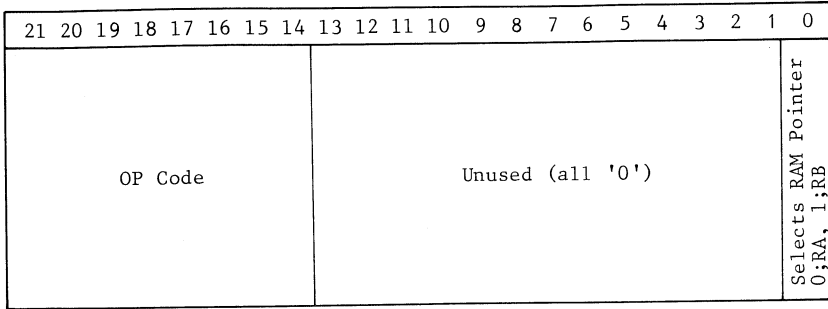
21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP Code						Jump Condition						Jump Address									
						RC*	/	/	/	/	/										/

- o When RC\* = 0, jumps if (RC)  $\neq$  0.
- o When RC\* = 1, jumps if (RC)  $\neq$  0. (RC)-1

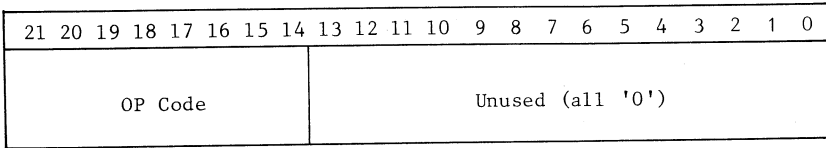
## IV. Register Data Transfer Instructions

21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
OP Code												Selects ACC A/B (destination) 0; ACCA, 1; ACCB			Selects ACC A/B (source) 0; ACCA, 1; ACCB			Unused (all '0')									Selects RAM pointer 0; RA, 1; RB	

V. Register Increment/Decrement Instructions



VI. Return Instructions



### 3. INSTRUCTION SET

MNE-MONIC	OPERATION	INSTRUCTION CODE																P	C	N	Z					
		21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6					5	4	3	2	1
I	FADA	P/Y+A/X→A (FLT)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	FADB	P/Y+B/X→B (FLT)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	ADA	P/Y+A/X→A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
	ADB	P/Y+B/X→B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
	FSBA	P/Y-A/X→A (FLT)	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	FSBB	P/Y-B/X→B (FLT)	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	SBA	P/Y-A/X→A	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
	SBB	P/Y-B/X→B	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
	FLDA	P/Y→A (FLT)	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	FLDB	P/Y→B (FLT)	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	LDA	P/Y→A	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
	LDB	P/Y→B	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
ANDA	P/Y∧A/X→A	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
ANDB	P/Y∧B/X→B	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
ORA	P/Y∨A/X→A	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
ORB	P/Y∨B/X→B	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
EORA	P/Y⊕A/X→A	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
EORB	P/Y⊕B/X→B	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
I'	FABSA	A →A (FLT)	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
	FABSB	B →B (FLT)	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
	ABSA	A →A	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
	ABSB	B →B	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
	FRPTA	Repeat next instruction (RC)+1 times	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
	FRPTB	Repeat next instruction (RC)+1 times	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
	RPTA	Repeat next instruction (RC)+1 times	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
	RPTB	Repeat next instruction (RC)+1 times	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
	FNEGA	-A→A (FLT)	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	FNEGB	-B→B (FLT)	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	NEGA	-A→A	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	NAGB	-B→B	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
INCA	A+1→A	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
INCB	B+1→B	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
DECA	A-1→A	0	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
DECB	B-1→B	0	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
SRA		0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
SRB		0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
SLA		0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
SLB		0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
FLTA	A(FIX)→A(FLT)⊕	0	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
FLTB	B(FIX)→B(FLT)⊕	0	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
FIXA	A(FLT)→A(FIX)⊕	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
FIXB	B(FLT)→B(FIX)⊕	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
FCLRA	00008→A	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	
FCLRB	00008→B	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	
CLRA	0000*→A	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	
CLRB	0000*→B	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	
FNOPA	{ All } (FLT)	1	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
FNOPB	{ no } (FLT)	1	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
NOPA	{ opera- } (FLT)	1	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
NOPB	{ tion } (FLT)	1	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
FSGYA	{ A/B→A/B } (FLT)	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
FSGYB	{ signA/B→signY } (FLT)	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
SGYA	{ A/B→A/B } (FLT)	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
SGYB	{ signA/B→signY } (FLT)	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
II	LIA	Immediate data→A	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	LIB	Immediate data→B	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	LIRA	Immediate data→RA	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	LIRB	Immediate data→RB	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	LIRO	Immediate data→RO	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	LIRC	Immediate data→RC	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
III	JCS	Jump if C=1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	JNS	Jump if N=1	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	JZS	Jump if Z=1	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
	JSR	Jump to subroutine	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	JNZ	Jump if RC=0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	JNZM	Jump if RC=0, RC-1→RC	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	JMP	Jump always	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	



#### 4. ASSEMBLER SYNTAX

##### I. ALU Operation Instructions (Pointer Addressing Mode)

<b>SYNTAX</b>
[ <label> ] Δ <mnemonic> Δ <operand (A)> [ Δ<comment> ]
<b>EXAMPLE</b>
FADA Δ $\frac{PA}{\textcircled{1}}$ , $\frac{D}{\textcircled{2}}$ , $\frac{XY(2,3)}{\textcircled{3}}$ , $\frac{RA+}{\textcircled{4}}$

[ ]: Can be omitted Δ: Space

<Operand (A)> Consists of  $\textcircled{1} \sim \textcircled{4}$ .

$\textcircled{1}$ ALU input select	$\textcircled{2}$ Memory write control	$\textcircled{3}$ ROM/RAM/GR page select	$\textcircled{4}$ ROM/RAM pointer select and increment
<u>PA</u> ...Product & ACC <u>YA</u> ...Y-Bus & ACC <u>PX</u> ...Product & X-Bus <u>YX</u> ...Y-Bus & X-Bus	<u>EE</u> ...Not write <u>A</u> ... ACC→RAM/GR <u>D</u> ... DREG→RAM/GR Destination address is determined by $\textcircled{3}$ and $\textcircled{4}$ .	<u>X,Y(n,m)</u> ...X-Bus:page n data Y-Bus:page m data  <u>X,G(n,l)</u> ...X-Bus:page n data Y-Bus:GR(l) data n = 0 ~ 7 m = 0 ~ 7 l = 0 ~ 3	<u>RA,RO</u> ...RA & RO select  <u>RB,RO</u> ...RB & RO select Autoincrement expression RA+ RB+ RO+ RO can be omitted.

##### I. ALU Operation Instructions (Direct Addressing Mode)

<b>SYNTAX</b>
[ <label> ] Δ <mnemonic> Δ <operand (B)> [ Δ<comment> ]
<b>EXAMPLE</b>
FADA Δ $\frac{PA}{\textcircled{1}}$ , $\frac{EE}{\textcircled{2}}$ , $\frac{3,18}{\textcircled{3}}$

<Operand (B)> Consists of  $\textcircled{1} \sim \textcircled{3}$

$\textcircled{1}$ ALU input select	$\textcircled{2}$ Memory write control	$\textcircled{3}$ Direct address
<u>PA</u> ...Product & ACC <u>YA</u> ...Y-Bus & ACC <u>PX</u> ...Product & X-Bus <u>YX</u> ...Y-Bus & X-Bus	<u>EE</u> ... Not write <u>A</u> ... ACC → RAM <u>D</u> ... D → RAM	<u>n,m</u> ... Specify the read/write address directly. n = page 0 ~ 7 m = address 0 ~ 49 (RAM) 0 ~ 31 (ROM)



### I' ALU Operation Instructions (Pointer Addressing Mode)

<b>SYNTAX</b>	[ <label> ] Δ <mnemonic> Δ <operand (C)> [ Δ <comment> ]
<b>EXAMPLE</b>	$\text{FABSA } \Delta \frac{\text{EE}}{\textcircled{2}}, \frac{\text{XG(1,3)}}{\textcircled{3}}, \frac{\text{RA}}{\textcircled{4}}$

<Operand (C)> consists of (2)~(4), which is same expression as <operand (A)>.

### I'. ALU Operation Instructions (Direct Addressing Mode)

<b>SYNTAX</b>	[ <label> ] Δ <mnemonic> Δ <operand (D)> [ Δ <comment> ]
<b>EXAMPLE</b>	$\text{FABSA } \Delta \frac{\text{A, 0, 12}}{\textcircled{2} \textcircled{3}}$

<Operand (D)> consists of (2)~(3), which is same expression as <operand (B)>.

### II. Immediate Instructions

<b>SYNTAX</b>	[ <label> ] Δ <mnemonic> Δ <constant> Δ [ <comment> ]
<b>EXAMPLE</b>	LIA Δ \$2FC7

<constant> LIA, LIB: \$0000 ~ \$FFFF, others : \$00 ~ \$3F

### III. Jump Instructions

<b>SYNTAX</b>	[ <label> ] Δ <mnemonic> Δ <constant> [ Δ <comment> ]
<b>EXAMPLE</b>	JCS Δ \$118

<constant> Jump address: \$000 ~ \$1E6

#### IV. Register Data Transfer Instructions

SYNTAX
[ <label> ] Δ <mnemonic> Δ <register 1>, <register 2> [Δ<comment>]
EXAMPLE
TFR Δ A, STR

<Register 1> : Source register: A, B, STR, CTR, RC, IR, RO, RA, RB, CCR, SIR

<Register 2> Destination  
register: A, B, STR, CTR, RC, OR, RO, RA, RB, CCR, SOR

#### V. Register Increment/Decrement Instructions

SYNTAX
[ <label> ] Δ <mnemonic> [ Δ <comment> ]
EXAMPLE
INCRA

#### VI. Return Instructions

SYNTAX
[ <label> ] Δ <mnemonic> [ Δ <comment> ]
EXAMPLE
RTN

## 5. HSP INSTRUCTION SUMMARY

\* In alphabetical order

Mnemonic	Page	Mnemonic	Page	TFR instruction	Page
ABSA	134	JCS	205	A, B	228
ABSB	136	JMP	208	A, CCR	217
ADA	98	JNS	205	A, CTR	211
ADB	100	JNZ	207	A, OR	213
ANDA	118	JNZM	207	A, RA	215
ANDB	120	JSR	206	A, RB	216
CLRA	182	JZS	206	A, RC	212
CLRB	184	LDA	114	A, RO	214
DECA	158	LDB	116	A, SOR	218
DECB	160	LIA	202	A, STR	210
DECRA	230	LIB	202	B, A	228
DECRB	231	LIRA	203	B, CCR	217
DECRC	232	LIRB	203	B, CTR	211
DECRO	231	LIRC	204	B, OR	213
EORA	126	LIRO	204	B, RA	215
EORB	128	NEGA	150	B, RB	216
FABSA	130	NEGB	152	B, RC	212
FABSB	132	NOPA	190	B, RO	214
FADA	94	NOPB	192	B, SOR	218
FADB	96	ORA	122	B, STR	210
FCLRA	178	ORB	124	CCR, A	226
FCLRB	180	RPTA	142	CCR, B	226
FIXA	174	RPTB	144	CTR, A	220
FIXB	176	RTI	232	CTR, B	220
FLDA	110	RTN	233	IR, A	222
FLDB	112	SBA	106	IR, B	222
FLTA	170	SBB	108	RA, A	224
FLTB	172	SGYA	198	RA, B	224
FNEGA	146	SGYB	200	RB, A	225
FNEGB	148	SLA	166	RB, B	225
FNOPA	186	SLB	168	RC, A	221
FNOPB	188	SRA	162	RC, B	221
FRPTA	138	SRB	164	RO, A	223
FRPTB	140	TFR	209	RO, B	223
FSBA	102			SIR, A	227
FSBB	104			SIR, B	227
FSGYA	194			STR, A	219
FSGYB	196			STR, B	219
INCA	154				
INCB	156				
INCRA	229				
INCRB	229				
INCRO	230				



## 7. HSP ORDERING SPECIFICATION

(1) General information Fill out or check the following items.

Family Name	HD61810	
Application *1		
Functions *2		
ROM Code		
Package Outline or Name	<input type="checkbox"/> 40 pin ceramic (DC-40)	<input type="checkbox"/> 40 pin plastic (DP-40)
	<input type="checkbox"/> 52 pin PLCC (CP-52)	
Special Specification		

\*1 Give the appliance using the HSP.  
(ex. cassette decks, VTRs, air conditioners)

\*2 Give the function of the HSP in the above appliance.  
(ex. automatic digital tuning, temperature control)

(2) Environmental check list

The following check list will be used for our reliability design, but it will not be used to determine the certified characteristics. Describe the environmental conditions for usual use.

Operating Ambient Temperature	Average	___ °C
	Range	___ °C - ___ °C
Operating Ambient Humidity	Average	___ %
	Range	___ % - ___ %
Power-on-time	Average	___ hours/day
Maximum Applied Voltage	Power Supply	___ V
	I/O pin	___ V
If there is any request on the HSP, please describe it.		

(3) Check list of ROM code

ROM Codes (2 sets)	<input type="checkbox"/> Attached	<input type="checkbox"/> Supplied
	<input type="checkbox"/> To be supplied (Date: ___ / ___ / ___)	
EPROM Type Number	( _____ )	

The following is for Hitachi's reference. Please leave it blank.

Type Number	_____
-------------	-------

Date	_____
Company	_____
Division	_____
Signature	_____
	_____
	_____

EPROM DATA FORMAT

INSTRUCTION ROM

HSP ADDRESS	EPROM ADDRESS			
	DUMMY	MSB ← → LSB		
0	003	002	001	000
1	007	006	005	004
2	00B	00A	009	008
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
1FF	7FF	7FE	7FD	7FC

Instruction Code Organization

21 1615 8 7 0

Dummy	00		
-------	----	--	--

22-bit effective data

DATA ROM

HSP ADDRESS		EPROM ADDRESS			
PAGE	POINTER	DUMMY		MSB ← → LSB	
4	0	803	802	801	800
	1	807	806	805	804
	.	.	.	.	.
	.	.	.	.	.
	1F	87F	87E	87D	87C
5	0	A03	A02	A01	A00
	1	A07	A06	A05	A04
	.	.	.	.	.
	.	.	.	.	.
	1F	A7F	A7E	A7D	A7C
6	0	C03	C02	C01	C00
	1	C07	C06	C05	C04
	.	.	.	.	.
	.	.	.	.	.
	1F	C7F	C7E	C7E	C7C
7	0	E03	E02	E01	E00
	1	E07	E06	E05	E04
	.	.	.	.	.
	.	.	.	.	.
	1F	E7F	E7E	E7D	E7C

Dummy data are all '\$FF'.

## 8. TEST PROGRAM

Locations \$1E7 through 1FE of the instruction ROM are reserved for the following test program.

```

*****
*                TEST    PROGRAM                *
*****
01E7 20 00 80      LIA      $B0
01E8 30 40 00      TFR      A, CTR
01E9 23 00 00      LIRB     0
01EA 25 C8 00      LIRC     50
01EB 34 E0 00      TFR      IR, B
01EC 3C ED C1      NOPB     A, XY(7, 0), RB
01ED 3C ED D1      NOPB     A, XY(7, 2), RB
01EE 01 C4 11      ADA      YX, EE, XY(0, 2), RB
01EF 00 24 11      FADB     PA, EE, XY(0, 2), RB
01F0 18 CD C9      LDA      YA, A, XY(7, 1), RB
01F1 31 00 00      TFR      A, OR
01F2 18 ED DD      LDB      YA, A, XY(7, 3), RB+
01F3 00 A0 13      FADB     YA, EE, 0, $13
01F4 31 10 00      TFR      B, OR
01F5 19 CE 40      LDA      YX, A, XG(1, 0), RA
01F6 19 CE C8      LDA      YX, A, XG(3, 1), RA
01F7 2B 01 EB      JNZ      $1EB
01F8 24 00 00      LIRO     0
01F9 25 84 00      LIRC     33
01FA 18 CD 20      LDA      YA, A, XY(4, 4), RA
01FB 18 CD 68      LDA      YA, A, XY(5, 5), RA
01FC 18 CD B0      LDA      YA, A, XY(6, 6), RA
01FD 18 CD FA      LDA      YA, A, XY(7, 7), RA, RQ+
01FE 2B 01 FA      JNZ      $1FA
*                END

```







