To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

To all our customers

Regarding the change of names mentioned in the document, such as Mitsubishi Electric and Mitsubishi XX, to Renesas Technology Corp.

The semiconductor operations of Hitachi and Mitsubishi Electric were transferred to Renesas Technology Corporation on April 1st 2003. These operations include microcomputer, logic, analog and discrete devices, and memory chips other than DRAMs (flash memory, SRAMs etc.) Accordingly, although Mitsubishi Electric, Mitsubishi Electric Corporation, Mitsubishi Semiconductors, and other Mitsubishi brand names are mentioned in the document, these names have in fact all been changed to Renesas Technology Corp. Thank you for your understanding. Except for our corporate trademark, logo and corporate statement, no changes whatsoever have been made to the contents of the document, and these changes do not constitute any alteration to the contents of the document itself.

Note : Mitsubishi Electric will continue the business operations of high frequency & optical devices and power devices.

Renesas Technology Corp.
Customer Support Dept.
April 1, 2003

Software Manual

16

RENESAS

7751 Series

Software Manual

MITSUBISHI 16-BIT SINGLE-CHIP
MICROCOMPUTER
7700 FAMILY

Renesas Electronics
www.renesas.com

New publication, 1996.07

keep safety first in your circuit designs !

- Mitsubishi Electric Corporation puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage. Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of non-flammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

- These materials are intended as a reference to assist our customers in the selection of the Mitsubishi semiconductor product best suited to the customer’s application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Mitsubishi Electric Corporation or a third party.

- Mitsubishi Electric Corporation assumes no responsibility for any damage, or infringement of any third-party’s rights, originating in the use of any product data, diagrams, charts or circuit application examples contained in these materials.

- All information contained in these materials, including product data, diagrams and charts, represent information on products at the time of publication of these materials, and are subject to change by Mitsubishi Electric Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Mitsubishi Electric Corporation or an authorized Mitsubishi Semiconductor product distributor for the latest product information before purchasing a product listed herein.

- Mitsubishi Electric Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Mitsubishi Electric Corporation or an authorized Mitsubishi Semiconductor product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.

- The prior written approval of Mitsubishi Electric Corporation is necessary to reprint or reproduce in whole or in part these materials.

- If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
  Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.

- Please contact Mitsubishi Electric Corporation or an authorized Mitsubishi Semiconductor product distributor for further details on these materials or the products contained therein.

REVISION DESCRIPTION LIST	7751 Series Software Manual

Rev. No.	Revision Description	Rev. date
1.00	First Edition	970728
1.01	Thumbnails are created for all pages.	980731

Preface

This manual describes the software of the Mitsubishi CMOS 16-bit microcomputers 7751 SERIES. After reading this manual, the users will be able to understand the instruction set and the features, so that they can utilize their capabilities fully. This manual shows detailed descriptions of the instructions and the addressing modes for 7751 SERIES.

For details concerning the hardware and the development support tools (assembler, debugger, and others) of each product of 7751 series, refer to the respective user's manual.

Table of contents

CHAPTER 1. DESCRIPTION

CHAPTER 2. CENTRAL PROCESSING UNIT (CPU)

CHAPTER 3. ADDRESSING MODES

CHAPTER 4. INSTRUCTIONS

CHAPTER 5. NUMBER OF INSTRUCTION CYCLES

CHAPTER 6. CPU INSTRUCTION EXECUTION SEQUENCE FOR EACH ADDRESSING MODE

APPENDIX

CHAPTER 1
DESCRIPTION

DESCRIPTION

The 7751 series software offers the following features :

- The m and x flags to select between word and byte operation make it possible to execute each of most instructions with 1-byte operation code, so that the ROM size for application will be reduced.
- Powerful addressing modes, and a fast and compact instruction set are included.
- Direct page mapping function and memory oriented software system by direct paging are included.
- The entire 16 Mbytes of addressable memory space can be programed as a program memory without consideration of a 64-Kbyte boundary.
- A data memory can be accessed with a linear or a bank.
- Bit manipulation instructions and bit test and branch instructions can be used for memory accessing of the entire 16 Mbytes of addressable memory space.
- Block transfer instructions handling blocks up to 64 Kbytes are available.
- Decimal arithmetic instruction execution requires no software correction.
- Upward compatibility for the 7700 series is retained.
- A Repeat MultiPly and Accumulate, **RMPA**, instruction is available.

CHAPTER 2
CENTRAL PROCESSING UNIT (CPU)

2.1 Central processing unit

The CPU (Central Processing Unit) has ten registers as shown in Figure 2.1.1.

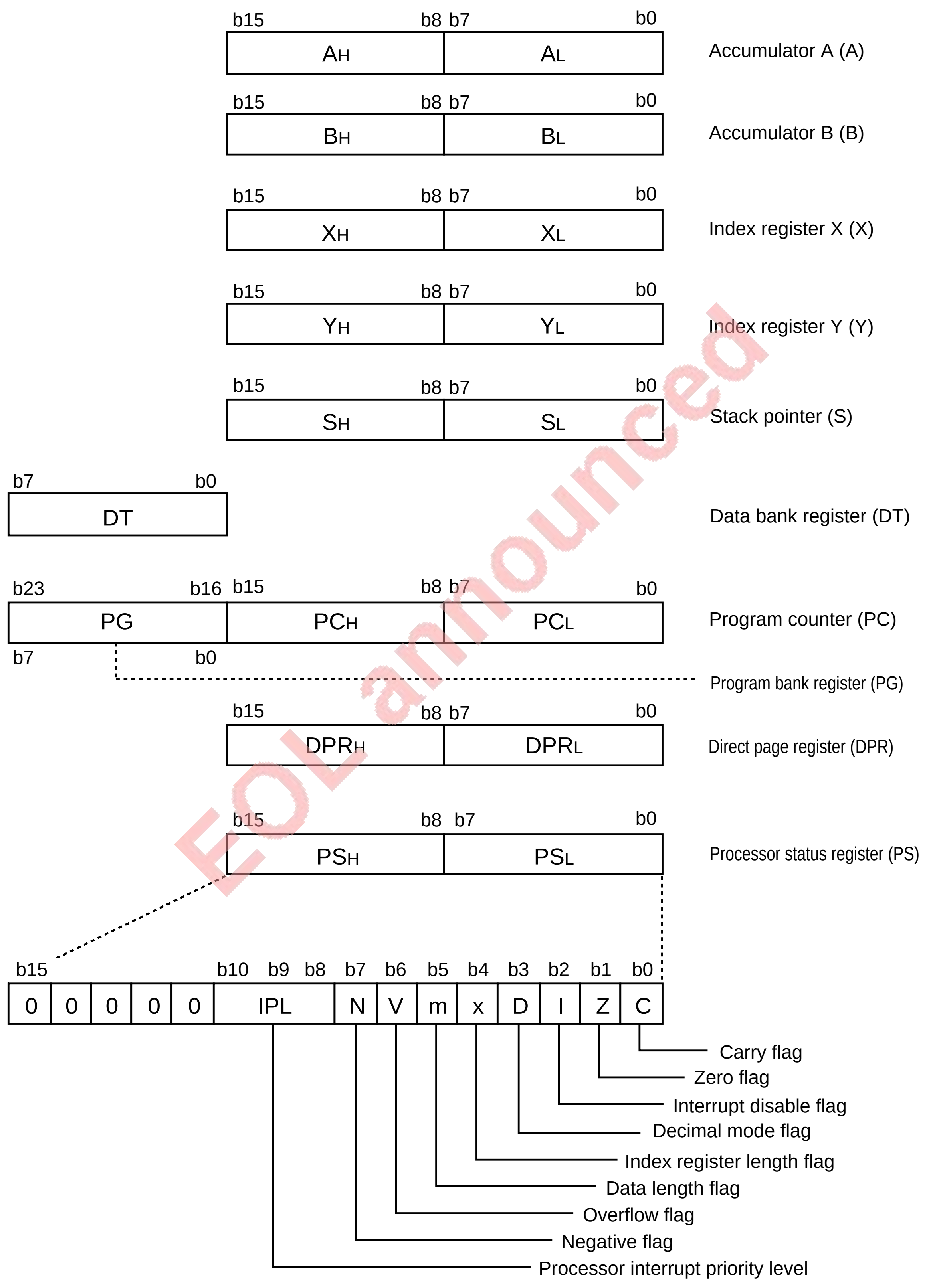


Fig. 2.1.1 CPU registers structure

2.1.1 Accumulator (Acc)

Accumulators A and B are available.

(1) Accumulator A (A)

Accumulator A is the main register of the microcomputer. The transaction of data such as calculation, data transfer, and input/output are performed mainly through accumulator A. It consists of 16 bits, and the low-order 8 bits can also be used separately. The data length flag (m) determines whether the register is used as a 16-bit register or as an 8-bit register. Flag m is a part of the processor status register which is described later. When an 8-bit register is selected, only the low-order 8 bits of accumulator A are used and the contents of the high-order 8 bits is unchanged.

(2) Accumulator B (B)

Accumulator B is a 16-bit register with the same function as accumulator A. Accumulator B can be used instead of accumulator A. The use of accumulator B, however except for some instructions, requires more instruction bytes and execution cycles than that of accumulator A. Accumulator B is also controlled by the data length flag (m) just as in accumulator A.

2.1.2 Index register X (X)

Index register X consists of 16 bits and the low-order 8 bits can also be used separately. The index register length flag (x) determines whether the register is used as a 16-bit register or as an 8-bit register. Flag x is a part of the processor status register which is described later. When an 8-bit register is selected, only the low-order 8 bits of index register X are used and the contents of the high-order 8 bits is unchanged. In an addressing mode in which index register X is used as an index register, the address obtained by adding the contents of this register to the operand's contents is accessed.

In the **MVP** or **MVN** instruction, a block transfer instruction, the contents of index register X indicates the low-order 16 bits of the source address. The third byte of the instruction is the high-order 8 bits of the source address.
In the **RMPA** instruction, a Repeat MultiPly and Accumulate instruction, the contents of index register X indicate the low-order 16 bits of address in which multiplicands are stored.

Note: Refer to **"CHAPTER 3. ADDRESSING MODES"** for addressing modes.

2.1.3 Index register Y (Y)

Index register Y is a 16-bit register with the same function as index register X. Just as in index register X, the index register length flag (x) determines whether this register is used as a 16-bit register or as an 8-bit register.
In the **MVP** or **MVN** instruction, a block transfer instruction, the contents of index register Y indicates the low-order 16 bits of the destination address. The second byte of the instruction is the high-order 8 bits of the destination address.
In the **RMPA** instruction, a Repeat MultiPly and Accumulate instruction, the contents of index register Y indicate the low-order 16 bits of address in which multipliers are stored.

2.1.4 Stack pointer (S)

The stack pointer (S) is a 16-bit register. It is used for a subroutine call or an interrupt. It is also used when addressing modes using the stack are executed. The contents of S indicate an address (stack area) for storing registers during subroutine calls and interrupts. Bank 0$_{16}$ is specified for the stack area. (Refer to **"2.1.6 Program bank register (PG)".**)

When an interrupt request is accepted, the microcomputer stores the contents of the program bank register (PG) at the address indicated by the contents of S and decrements the contents of S by 1. Then the contents of the program counter (PC) and the processor status register (PS) are stored. The contents of S after accepting an interrupt request is equal to the contents of S decremented by 5 before the accepting of the interrupt request. (Refer to **Figure 2.1.2.**)

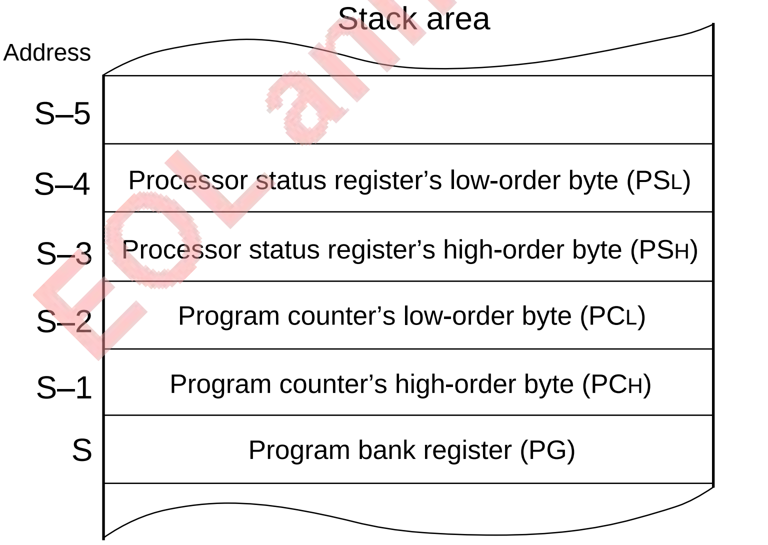
When completing the process in the interrupt routine and returning to the original routine, the contents of registers stored in the stack area are restored into the original registers in the reverse sequence (PS→PC→PG) by executing the **RTI** instruction. The contents of S is returned to the state before accepting an interrupt request.

The same operation is performed during a subroutine call, however, the contents of PS is not automatically stored. (The contents of PG may not be stored. This depends on the addressing mode.)

You should store registers other than those described above with software when you need them during interrupts or subroutine calls.

Additionally, initialize S at the beginning of the program because its contents are undefined at reset. The stack area changes when subroutines are nested or when multiple interrupt requests are accepted. Therefore, make sure of the subroutine's nesting depth not to destroy the necessary data.

Note: Refer to **"CHAPTER 3. ADDRESSING MODES"** for addressing modes.


Stack area
Address
S–5
S–4
Processor status register's low-order byte (PS$_L$)
S–3
Processor status register's high-order byte (PS$_H$)
S–2
Program counter's low-order byte (PC$_L$)
S–1
Program counter's high-order byte (PC$_H$)
S
Program bank register (PG)

● "S" is the initial address that the stack pointer (S) indicates at accepting an interrupt request.
The S's contents become "S–5" after storing the above registers.

Fig. 2.1.2 Contents of the stack area after accepting interrupt request

2.1.5 Program counter (PC)

The program counter is a 16-bit counter that indicates the low-order 16 bits of the address (24 bits) at which an instruction to be executed next (in other words, an instruction to be read out from an instruction queue buffer next) is stored. The contents of the high-order program counter (PC$_H$) become “FF$_{16}$,” and the low-order program counter (PC$_L$) becomes “FE$_{16}$” at reset. The contents of the program counter becomes the contents of the reset’s vector address (addresses FFFE$_{16}$, FFFF$_{16}$) just after reset.
Figure 2.1.3 shows the program counter and the program bank register.

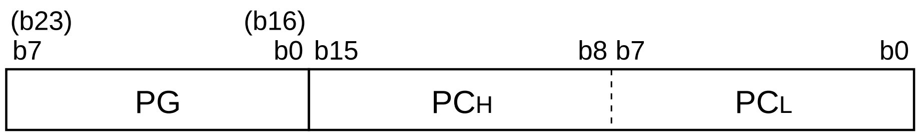


Fig. 2.1.3 Program counter and program bank register

2.1.6 Program bank register (PG)

The program bank register is an 8-bit register that indicates the high-order 8 bits of the address (24 bits) at which an instruction to be executed next (in other words, an instruction to be read out from an instruction queue buffer next) is stored. These 8 bits are called bank.
When a carry occurs after adding the contents of the program counter or adding the offset value to the contents of the program counter in the branch instruction and others, the contents of the program bank register is automatically incremented by 1. When a borrow occurs after subtracting the contents of the program counter, the contents of the program bank register is automatically decremented by 1. Accordingly, there is no need to consider bank boundaries in programming, usually.

In single-chip mode, make sure to prevent the program bank register from being set to a value other than “00$_{16}$” by executing the branch instructions and others. It is because only the internal area, within bank 0$_{16}$, can be accessed in single-chip mode.
This register is cleared to “00$_{16}$” at reset.

2.1.7 Data bank register (DT)

The data bank register is an 8-bit register. In the following addressing modes using the data bank register, the contents of this register is used as the high-order 8 bits (bank) of a 24-bit address to be accessed.

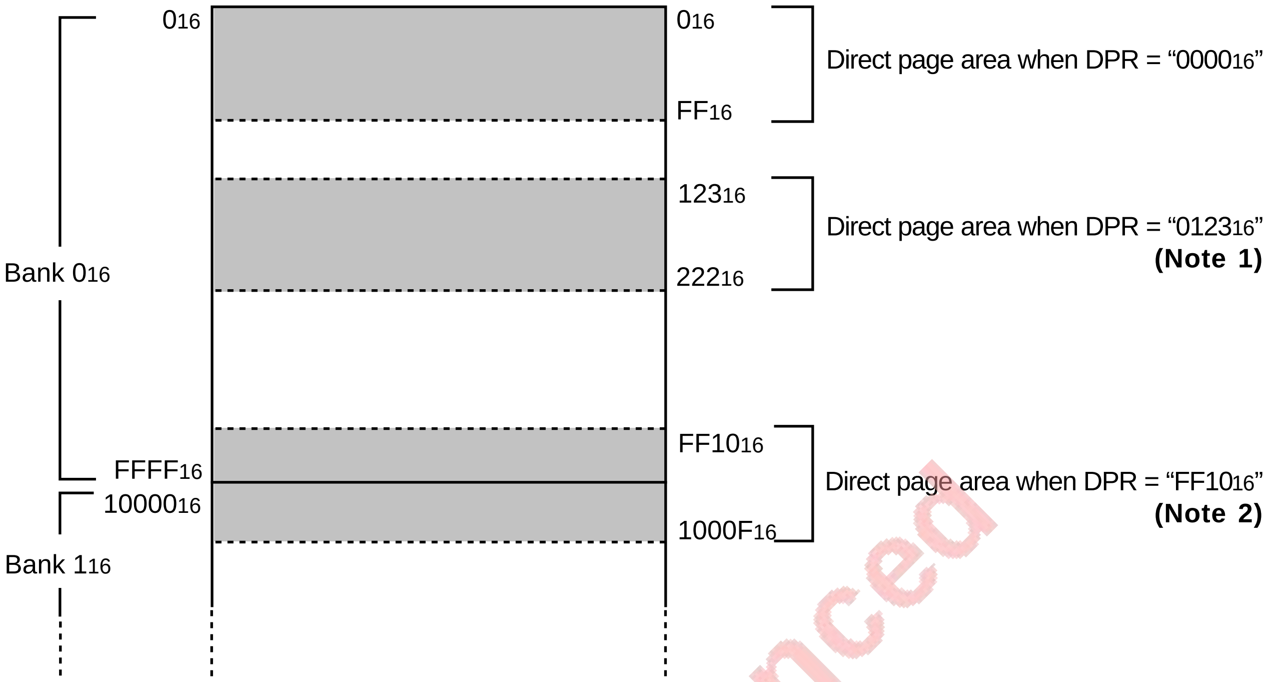
In single-chip mode, make sure to fix this register to “00_{16}.” It is because only the internal area, within bank 0_{16}, can be accessed in single-chip mode.
This register is cleared to “00_{16}” at reset.

●Addressing modes using data bank register
- Direct indirect
- Direct indexed X indirect
- Direct indirect indexed Y
- Absolute
- Absolute bit
- Absolute indexed X
- Absolute indexed Y
- Absolute bit relative
- Stack pointer relative indirect indexed Y
- Multiplied accumulation

2.1.8 Direct page register (DPR)

The direct page register is a 16-bit register. The contents of this register indicate the direct page area which is allocated in bank 0_{16} or in the space across banks 0_{16} and 1_{16}. The following addressing modes use the direct page register.
The contents of the direct page register indicates the base address (the lowest address) of the direct page area. The space which extends to 256 bytes above that address is specified as a direct page.
The direct page register can contain a value from 0000_{16} to $FFFF_{16}$. When it contains a value equal to or more than “$FF01_{16}$,” the direct page area spans the space across banks 0_{16} and 1_{16}.
When the contents of low-order 8 bits of the direct page register is “00_{16},” the number of cycles required to generate an address is 1 cycle smaller than the number when its contents are not “00_{16}.” Accordingly, the access efficiency can be enhanced in this case.
This register is cleared to “0000_{16}” at reset.
Figure 2.1.4 shows a setting example of the direct page area.

●Addressing modes using direct page register
- Direct
- Direct bit
- Direct indexed X
- Direct indexed Y
- Direct indirect
- Direct indexed X indirect
- Direct indirect indexed Y
- Direct indirect long
- Direct indirect long indexed Y
- Direct bit relative



Notes 1 : The number of cycles required to generate an address is 1 cycle smaller when the low-order 8 bits of the DPR are "00_{16}."

2 : The direct page area spans the space across banks 0_{16} and 1_{16} when the DPR is "$FF01_{16}$" or more.

Fig. 2.1.4 Setting example of direct page area

2.1.9 Processor status register (PS)

The processor status register is an 11-bit register.
Figure 2.1.5 shows the structure of the processor status register.

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
0	0	0	0	0	IPL			N	V	m	x	D	I	Z	C	Processor staus register (PS)

Note: Fix each of bits 15–11 to “0”.

Fig. 2.1.5 Processor status register structure

(1) Bit 0: Carry flag (C)

It retains a carry or a borrow generated in the arithmetic and logic unit (ALU) during an arithmetic operation. This flag is also affected by shift and rotate instructions. When the **BCC** or **BCS** instruction is executed, this flag’s contents determine whether the program causes a branch or not.
Use the **SEC** or **SEP** instruction to set this flag to “1”, and use the **CLC** or **CLP** instruction to clear it to “0”.

(2) Bit 1: Zero flag (Z)

It is set to “1” when the result of an arithmetic operation or data transfer is “0,” and cleared to “0” when otherwise. When the **BNE** or **BEQ** instruction is executed, this flag’s contents determine whether the program causes a branch or not.
Use the **SEP** instruction to set this flag to “1,” and use the **CLP** instruction to clear it to “0.”
Note: This flag is invalid in the decimal mode addition (the **ADC** instruction).

(3) Bit 2: Interrupt disable flag (I)

It disables all maskable interrupts (interrupts other than watchdog timer, the **BRK** instruction, and zero division). Interrupts are disabled when this flag is “1.” When an interrupt request is accepted, this flag is automatically set to “1” to avoid multiple interrupts. Use the **SEI** or **SEP** instruction to set this flag to “1,” and use the **CLI** or **CLP** instruction to clear it to “0.” This flag is set to “1” at reset.

(4) Bit 3: Decimal mode flag (D)

It determines whether addition and subtraction are performed in binary or decimal. Binary arithmetic is performed when this flag is “0.” When it is “1,” decimal arithmetic is performed with each word treated as two or four digits decimal (determined by the data length flag). Decimal adjust is automatically performed. Decimal operation is possible only with the **ADC** and **SBC** instructions. Use the **SEP** instruction to set this flag to “1,” and use the **CLP** instruction to clear it to “0.” This flag is cleared to “0” at reset.

(5) Bit 4: Index register length flag (x)

It determines whether each of index register X and index register Y is used as a 16-bit register or an 8-bit register. That register is used as a 16-bit register when this flag is “0,” and as an 8-bit register when it is “1.” Use the **SEP** instruction to set this flag to “1,” and use the **CLP** instruction to clear it to “0.” This flag is cleared to “0” at reset.

Note: When transferring data between registers which are different in bit length, the data is transferred with the length of the destination register, but except for the **TXA**, **TYA**, **TXB**, **TYB**, and **TXS** instructions. Refer to **“CHAPTER 4. INSTRUCTIONS”** for details.

(6) Bit 5: Data length flag (m)

It determines whether to use data as a 16-bit unit or as an 8-bit unit. A data is treated as a 16-bit unit when this flag is “0,” and as an 8-bit unit when it is “1.”
Use the **SEM** or **SEP** instruction to set this flag to “1,” and use the **CLM** or **CLP** instruction to clear it to “0.” This flag is cleared to “0” at reset.

Note: When transferring data between registers which are different in bit length, the data is transferred with the length of the destination register, but except for the **TXA**, **TYA**, **TXB**, **TYB**, and **TXS** instructions. Refer to **“CHAPTER 4. INSTRUCTIONS”** for details.

(7) Bit 6: Overflow flag (V)

It is used when adding or subtracting with a word regarded as signed binary. When the data length flag (m) is “0,” the overflow flag is set to “1” when the result of addition or subtraction exceeds the range between –32768 and +32767, and cleared to “0” in all other cases. When the data length flag (m) is “1,” the overflow flag is set to “1” when the result of addition or subtraction exceeds the range between –128 and +127, and cleared to “0” in all other cases.
The overflow flag is also set to “1” when the result of division exceeds the register length to be stored in the **DIV** or **DIVS** instruction, a division instruction with signed or unsigned; and when the result of addition exceeds the range between –2147483648 and +2147483647 in the **RMPA** instruction, a Repeat MultiPly and Accumulate instruction.
When the **BVC** or **BVS** instruction is executed, this flag’s contents determine whether the program causes a branch or not. Use the **SEP** instruction to set this flag to “1,” and use the **CLV** or **CLP** instruction to clear it to “0.”
Note: This flag is invalid in the decimal mode.

(8) Bit 7: Negative flag (N)

It is set to “1” when the result of arithmetic operation or data transfer is negative. (Bit 15 of the result is “1” when the data length flag (m) is “0,” or bit 7 of the result is “1” when the data length flag (m) is “1.”) It is cleared to “0” in all other cases. When the **BPL** or **BMI** instruction is executed, this flag determines whether the program causes a branch or not. Use the **SEP** instruction to set this flag to “1,” and use the **CLP** instruction to clear it to “0.”
Note: This flag is invalid in the decimal mode.

(9) Bits 10 to 8: Processor interrupt priority level (IPL)

These three bits can determine the processor interrupt priority level to one of levels 0 to 7. The interrupt is enabled when the interrupt priority level of a required interrupt, which is set in each interrupt control register, is higher than IPL. When an interrupt request is accepted, IPL is stored in the stack area, and IPL is replaced by the interrupt priority level of the accepted interrupt request.
There are no instruction to directly set or clear the bits of IPL. IPL can be changed by storing the new IPL into the stack area and updating the processor status register with the **PUL** or **PLP** instruction.
The contents of IPL is cleared to “000_2” at reset.

2.2 Bus interface unit

A bus interface unit (BIU) is built-in between the central processing unit (CPU) and memory•I/O devices. BIU's function and operation are described below.

2.2.1 Overview

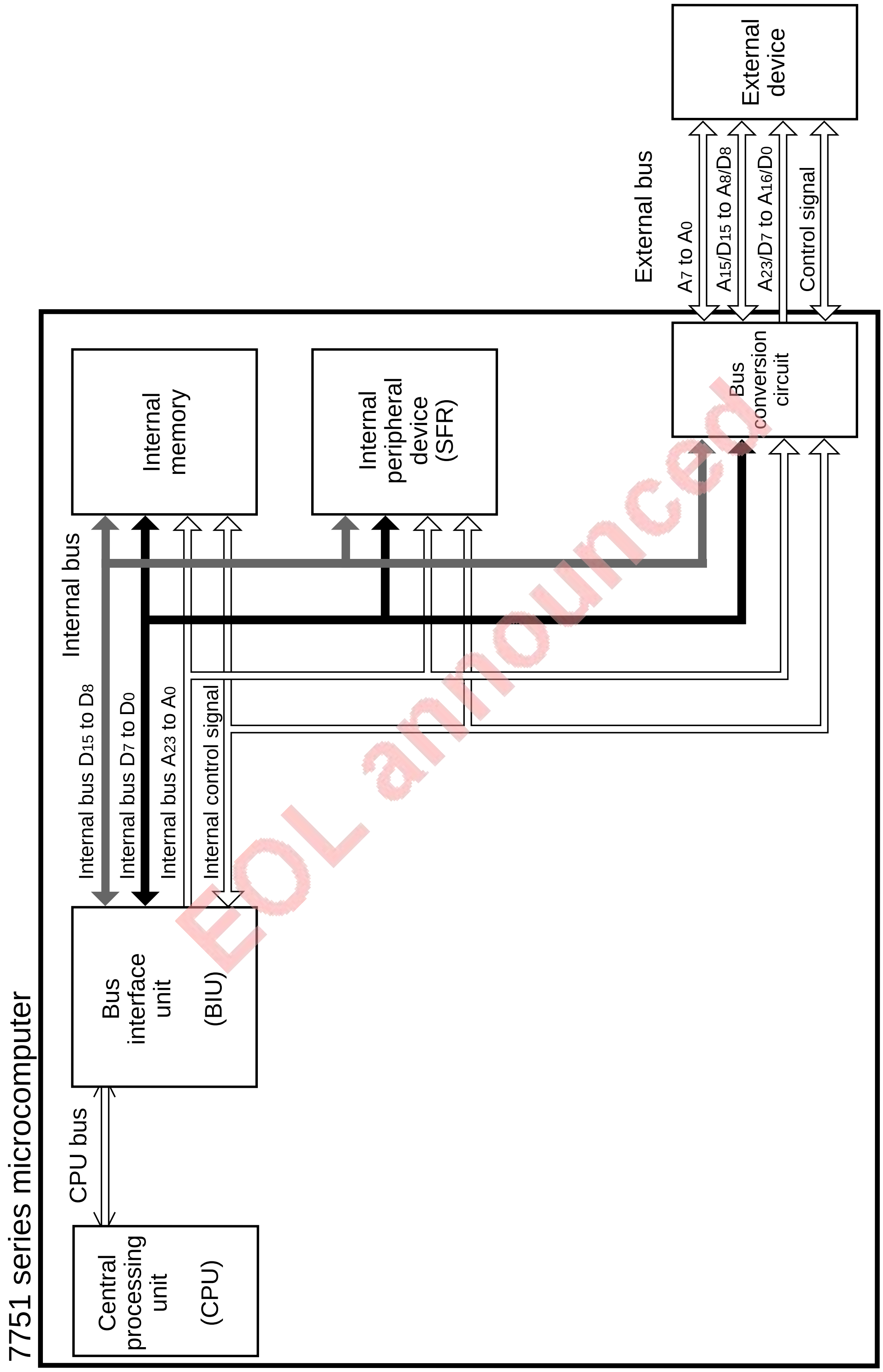
Transfer operation between the CPU and memory•I/O devices is always performed via the BIU.
Figure 2.2.1 shows the bus and bus interface unit (BIU).

① The BIU reads an instruction from the memory before the CPU executes it.

② When the CPU reads data from the memory • I/O device, the CPU first specifies the address from which data is read to the BIU. The BIU reads data from the specified address and passes it to the CPU.

③ When the CPU writes data to the memory • I/O device, the CPU first specifies the address to which data is written to the BIU and write data. The BIU writes the data to the specified address.

④ To perform the above operations ① to ③, the BIU inputs and outputs the control signals, and control the bus.



SFR : Special Function Register

Note : The CPU bus, internal bus, and external bus are independent of one another.

Fig. 2.2.1 Bus and bus interface unit (BIU)

2.2.2 Functions of bus interface unit (BIU)

The bus interface unit (BIU) consists of four registers shown in Figure 2.2.2. Table 2.2.1 shows the functions of each register.

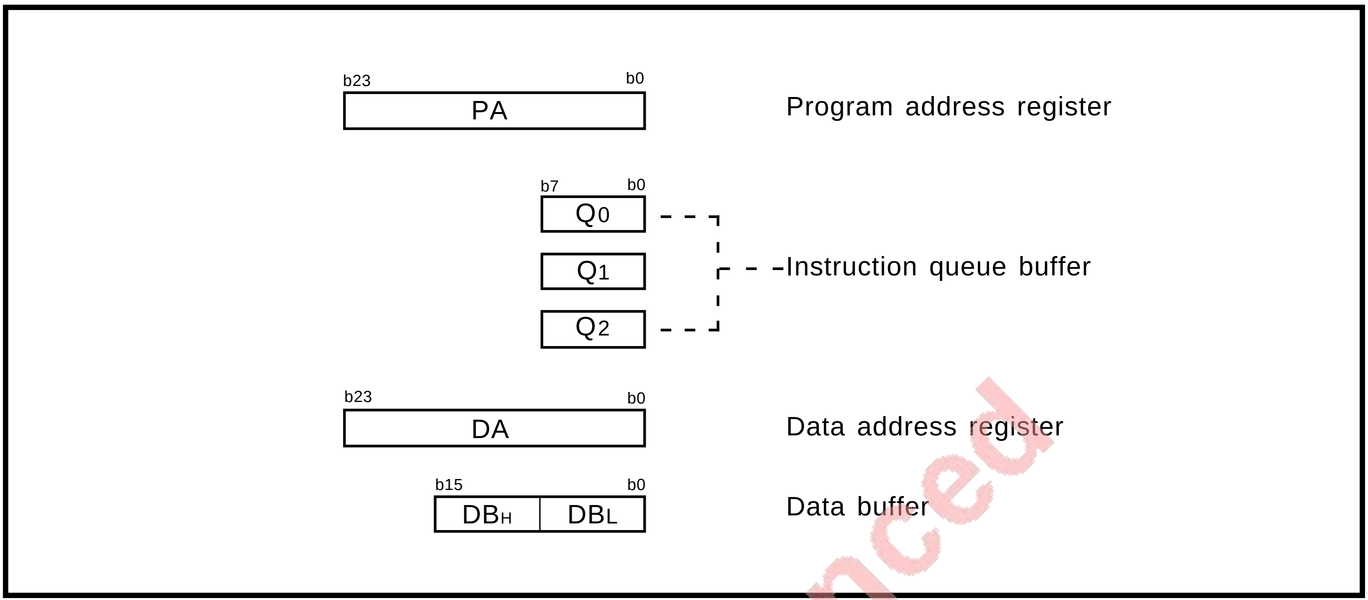


Fig. 2.2.2 Register structure of bus interface unit (BIU)

Table 2.2.1 Functions of each register

Name	Functions
Program address register	Indicates the storage address for the instruction which is next taken into the instruction queue buffer.
Instruction queue buffer	Temporarily stores the instruction which has been taken in.
Data address register	Indicates the address for the data which is next read from or written to.
Data buffer	Temporarily stores the data which is read from the memory•I/O device by the BIU or which is written to the memory•I/O device by the CPU.

The CPU and the bus send or receive data via BIU because each operates based on different clocks (Note). The BIU allows the CPU to operate at high speed without waiting for access to the memory • I/O devices that require a long access time.
The BIU's functions are described bellow.

Note: The CPU operates based on ϕCPU. The period of ϕCPU is normally the same as that of the internal clock ϕ. The internal bus operates based on the signal $\overline{E}$. The period of the signal $\overline{E}$ is normally twice that of the internal clock ϕ at a minimum.

(1) Reading out instruction (Instruction prefetch)

When the CPU does not require to read or write data, that is, when the bus is not in use, the BIU reads instructions from the memory and stores them in the instruction queue buffer. This is called instruction prefetch.
The CPU reads instructions from the instruction queue buffer and executes them, so that the CPU can operate at high speed without waiting for access to the memory which requires a long access time.
When the instruction queue buffer becomes empty or contains only 1 byte of an instruction, the BIU performs instruction prefetch. The instruction queue buffer can store instructions up to 3 bytes.
The contents of the instruction queue buffer is initialized when a branch or jump instruction is executed, and the BIU reads a new instruction from the destination address.
When instructions in the instruction queue buffer are insufficient for the CPU's needs, the BIU extends the pulse duration of clock ϕCPU in order to keep the CPU waiting until the BIU fetches required number of instructions or more.

(2) Reading data from memory•I/O device

The CPU specifies the storage address of data to be read to the BIU's data address register, and requires data. The CPU waits until data is ready in the BIU.
The BIU outputs the address received from the CPU onto the address bus, reads contents at the specified address, and takes it into the data buffer.
The CPU continues processing, using data in the data buffer.
However, if the BIU uses the bus for instruction prefetch when the CPU requires to read data, the BIU keeps the CPU waiting.

(3) Writing data to memory•I/O device

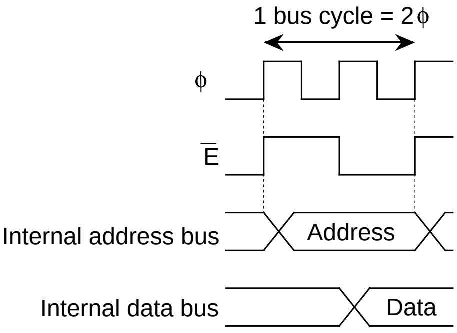
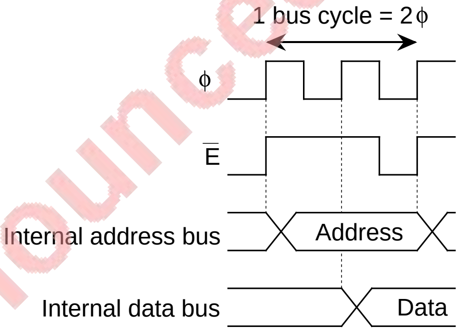
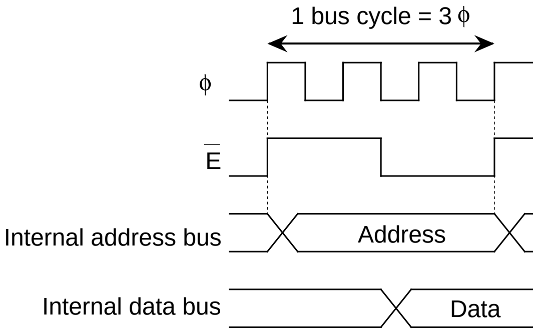
The CPU specifies the address of data to be written to the BIU's data address register. Then, the CPU writes data into the data buffer. The BIU outputs the address received from the CPU onto the address bus and writes data in the data buffer into the specified address.

The CPU advances to the next processing without waiting for completion of BIU's write operation. However, if the BIU uses the bus for instruction prefetch when the CPU requires to write data, the BIU keeps the CPU waiting.

(4) Bus control

To perform the above operations (1) to (3), the BIU inputs and outputs the control signals, and controls the address bus and the data bus. The cycle which the BIU controls the bus and accesses the memory•I/O device is called the bus cycle. Table 2.2.2 shows the bus cycle at accessing the internal area.

Table 2.2.2 Bus cycle at accessing internal area

	In low-speed running	In high-speed running
RAM	 1 bus cycle = 2ϕ ϕ $\overline{E}$ Internal address bus Address Internal data bus Data	 1 bus cycle = 2ϕ ϕ $\overline{E}$ Internal address bus Address Internal data bus Data
ROM		 1 bus cycle = 3 ϕ ϕ $\overline{E}$ Internal address bus Address Internal data bus Data
SFR		

2.2.3 Operation of bus interface unit (BIU)

Figure 2.2.3 shows the basic operating waveforms of the bus interface unit (BIU).

(1) When fetching instructions into the instruction queue buffer

① When the instruction which is next fetched is located at an even address, the BIU fetches 2 bytes of the instruction at a time with the timing of waveform (a).
However, when accessing an external device which is connected with the 8-bit external data bus width (BYTE = “H”), only 1 byte of the instruction is fetched.

② When the instruction which is next fetched is located at an odd address, the BIU fetches only 1 byte of the instruction with the timing of waveform (a). The data at the even address is not taken into the data buffer.

(2) When reading or writing data to and from the memory•I/O device

① When accessing a 16-bit data which begins at an even address, waveform (a) is applied. The 16 bits of data are accessed at a time.

② When accessing a 16-bit data which begins at an odd address, waveform (b) is applied. The 16 bits of data are accessed separately in 2 operations, 8 bits at a time. Invalid data is not fetched into the data buffer.

③ When accessing an 8-bit data at an even address, waveform (a) is applied. The data at the odd address is not fetched into the data buffer.

④ When accessing an 8-bit data at an odd address, waveform (a) is applied. The data at the even address is not fetched into the data buffer.

For instructions that are affected by the data length flag (m) and the index register length flag (x), operation ① or ② is applied when flag m or x = “0”; operation ③ or ④ is applied when flag m or x = “1.”

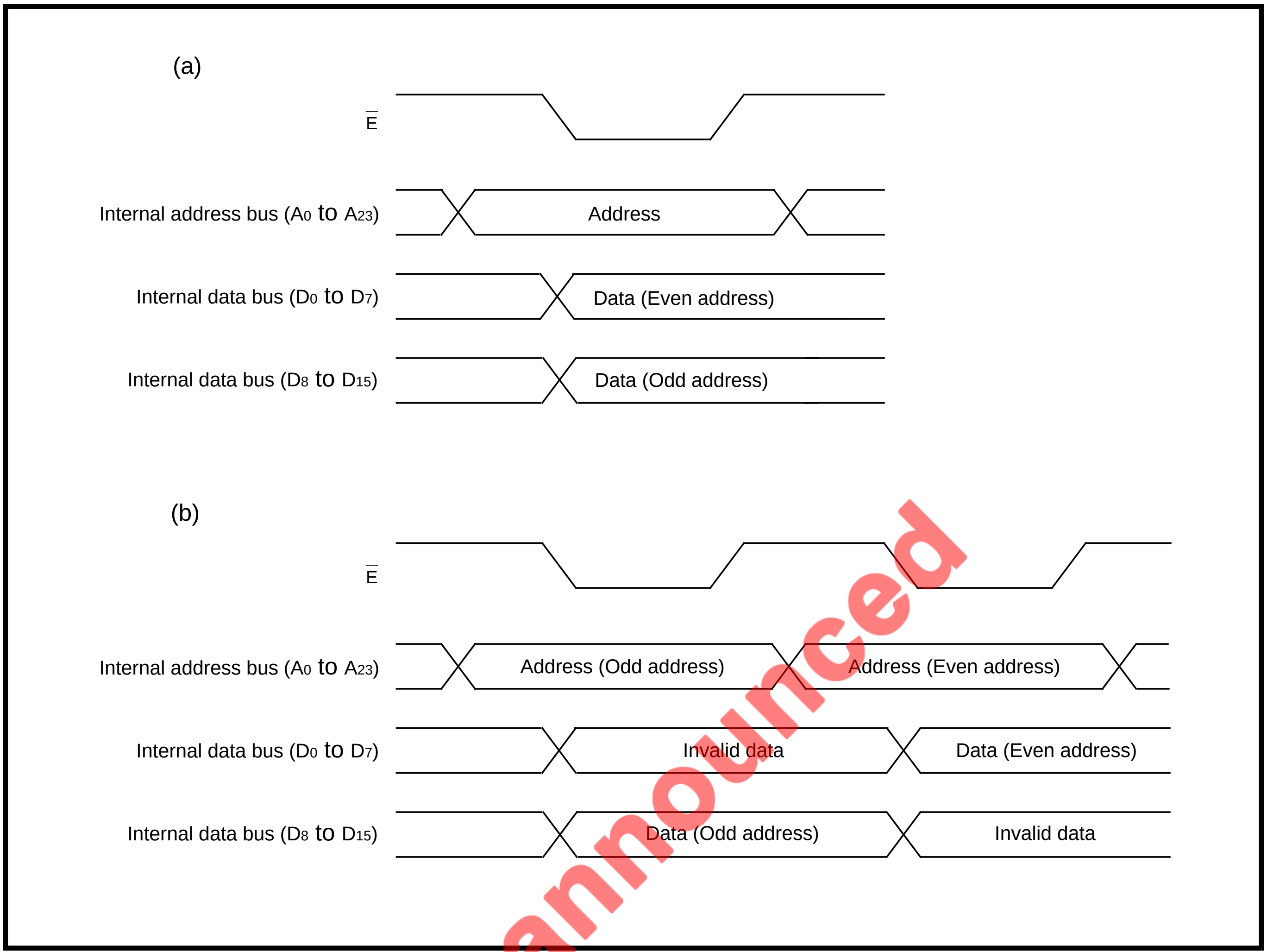

(a)
$\overline{E}$
Internal address bus (A_0 to A_{23})
Address
Internal data bus (D_0 to D_7)
Data (Even address)
Internal data bus (D_8 to D_{15})
Data (Odd address)
(b)
$\overline{E}$
Internal address bus (A_0 to A_{23})
Address (Odd address)
Address (Even address)
Internal data bus (D_0 to D_7)
Invalid data
Data (Even address)
Internal data bus (D_8 to D_{15})
Data (Odd address)
Invalid data

Fig. 2.2.3 Basic operating waveforms of bus interface unit (BIU)

CHAPTER 3
ADDRESSING MODES

3.1 Addressing modes

To execute an instruction, when the data required for arithmetic operation is retrieved from a memory or the result of arithmetic operation is stored to it, it is necessary to specify the address of the memory location in advance. Address specification is also necessary when the control is to jump to a certain memory address during program execution. Addressing refers to the method of specifying the memory address.

The memory access of the 7751 series microcomputers is reinforced with 29 different addressing modes.

3.2 Explanation of addressing modes

Each of the 29 addressing modes is explained on the pages indicated below:

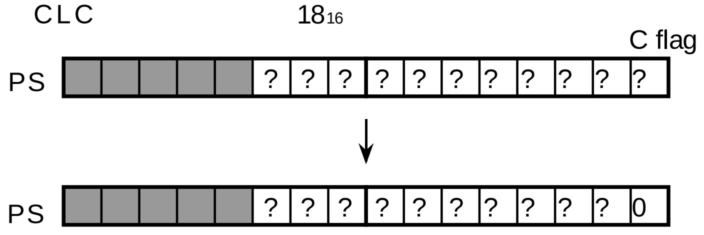
Implied

Mode : Implied addressing mode

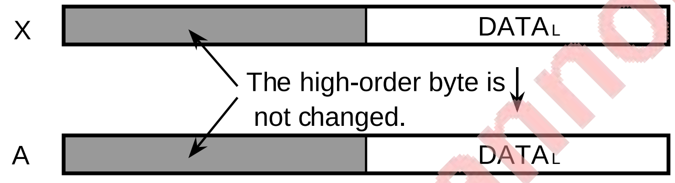
Function : Registers and others are worked with one instruction.

Instruction : BRK, CLC, CLI, CLM, CLV, DEX, DEY, INX, INY, NOP, RTI, RTL, RTS, SEC, SEI, SEM, STP, TAD, TAS, TAX, TAY, TBD, TBS, TBX, TBY, TDA, TDB, TSA, TSB, TSX, TXA, TXB, TXS, TXY, TYA, TYB, TYX, WIT, XAB

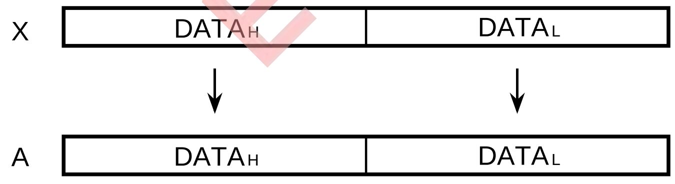
ex. : Mnemonic	Machine code
CLC	18_{16}



ex. : Mnemonic	Machine code
TXA (m=“1”, x=“1”)	$8A_{16}$



ex. : Mnemonic	Machine code
TXA (m=“0”, x=“0”)	$8A_{16}$



Immediate

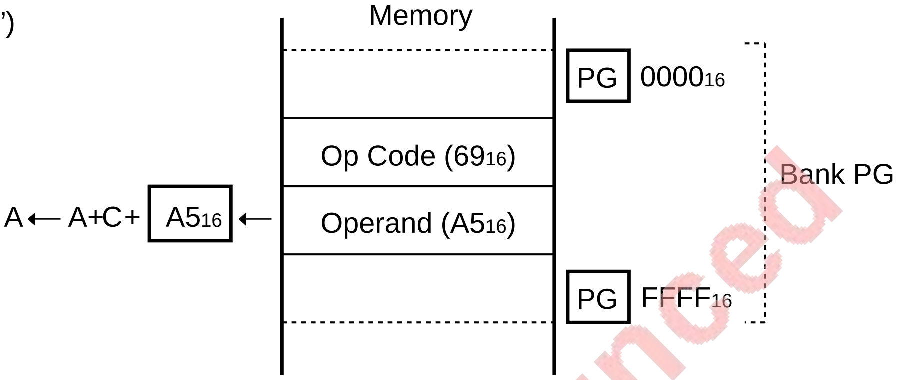
Mode : Immediate addressing mode

Function : A portion of the instruction is an actual data. Such instruction code can cross over the bank boundary.

Instruction : ADC, AND, CLP, CMP, CPX, CPY, DIV, DIVS, EOR, LDA, LDT, LDX, LDY, MPY, MPYS, ORA, RLA, SBC, SEP

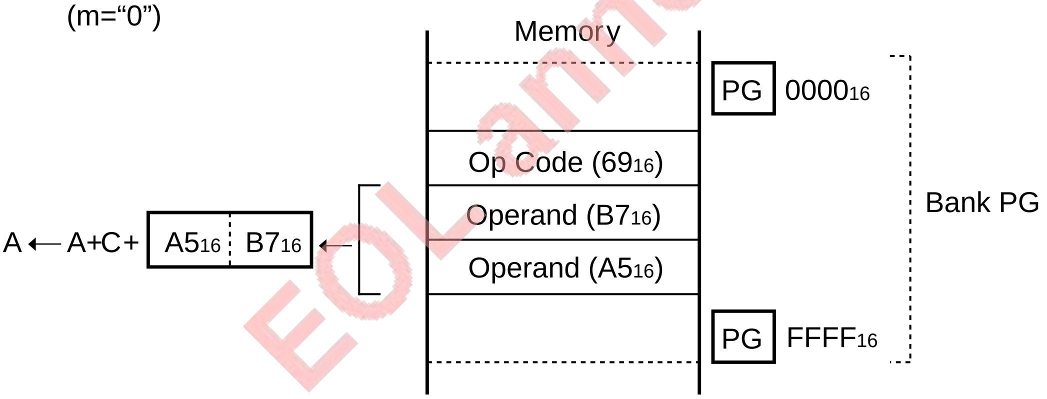
ex. : Mnemonic ADC A, #0A5H (m="1")

Machine code 69_{16} $A5_{16}$



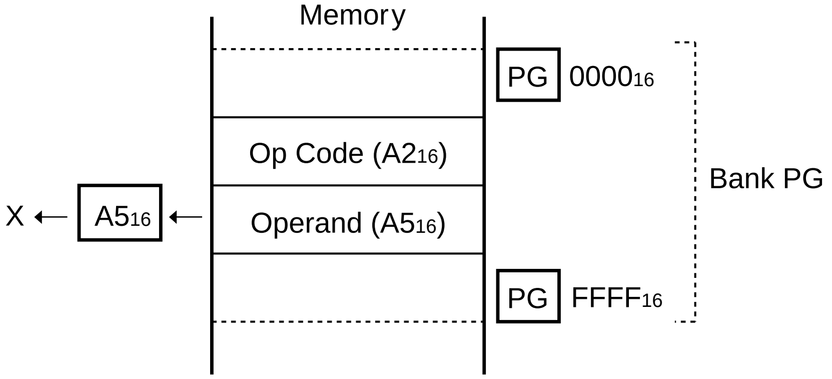
ex. : Mnemonic ADC A, #0A5B7H (m="0")

Machine code 69_{16} $B7_{16}$ $A5_{16}$



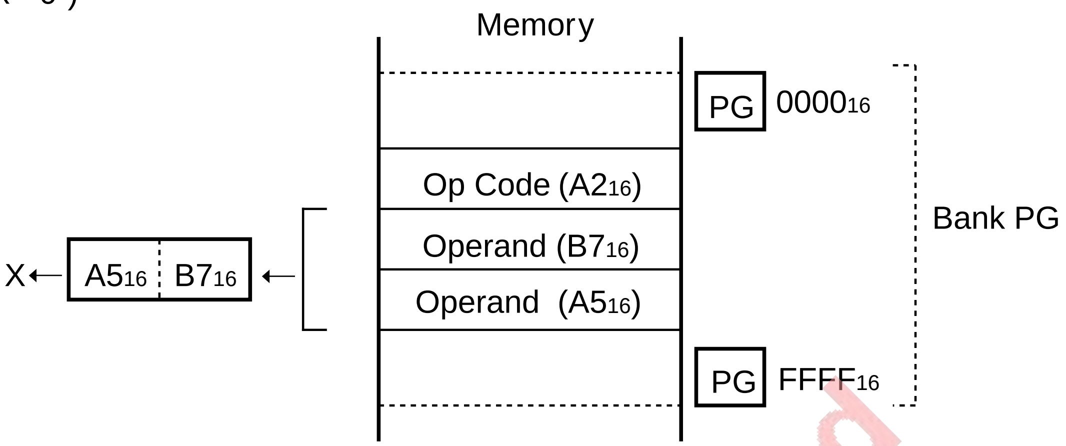
ex. : Mnemonic LDX #0A5H (x="1")

Machine code $A2_{16}$ $A5_{16}$



Immediate

ex. : Mnemonic	Machine code
LDX #0A5B7H (x="0")	$A2_{16}$ $B7_{16}$ $A5_{16}$



Accumulator

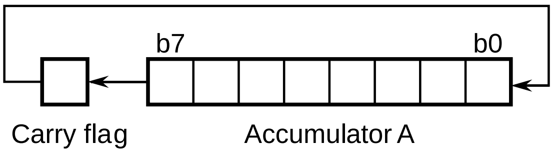
Mode : Accumulator addressing mode

Function : The contents of accumulator are an actual data.

Instruction : ASL, ASR, DEC, EXTS, EXTZ, INC, LSR, ROL, ROR

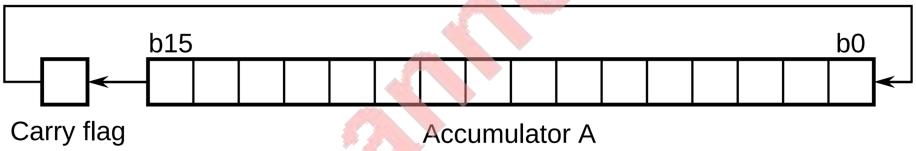
ex. : Mnemonic Machine code

ROL A $2A_{16}$
(m="1")



ex. : Mnemonic Machine code

ROL A $2A_{16}$
(m="0")



Direct

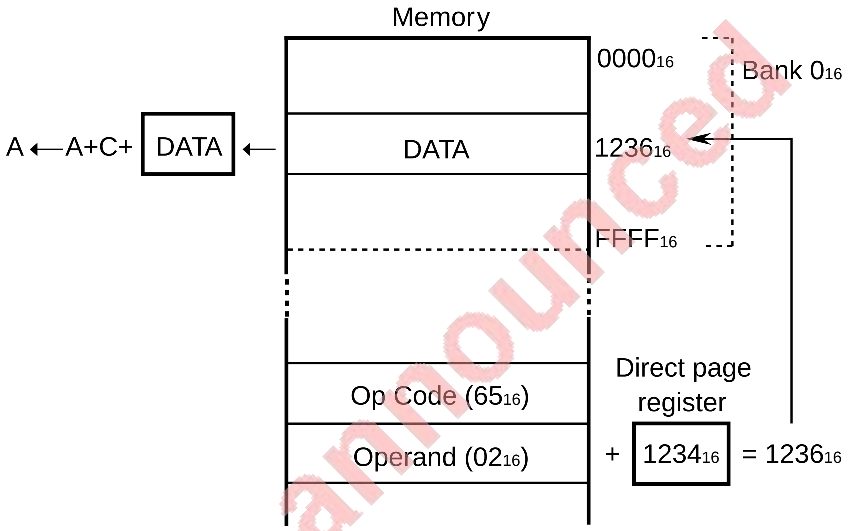
Mode : Direct addressing mode

Function : The contents of a memory in bank 0_{16} are an actual data. This memory location is specified by the result of adding the instruction's second byte to the contents of the direct page register. When, however, the result of adding of the instruction's second byte to the direct page register's contents exceeds bank 0_{16} range, the memory location in bank 1_{16} is specfied.

Instruction : ADC, AND, ASL, ASR, CMP, CPX, CPY, DEC, DIV, DIVS, EOR, INC,
LDA, LDM, LDX, LDY, LSR, MPY, MPYS, ORA, ROL, ROR, SBC, STA,
STX, STY

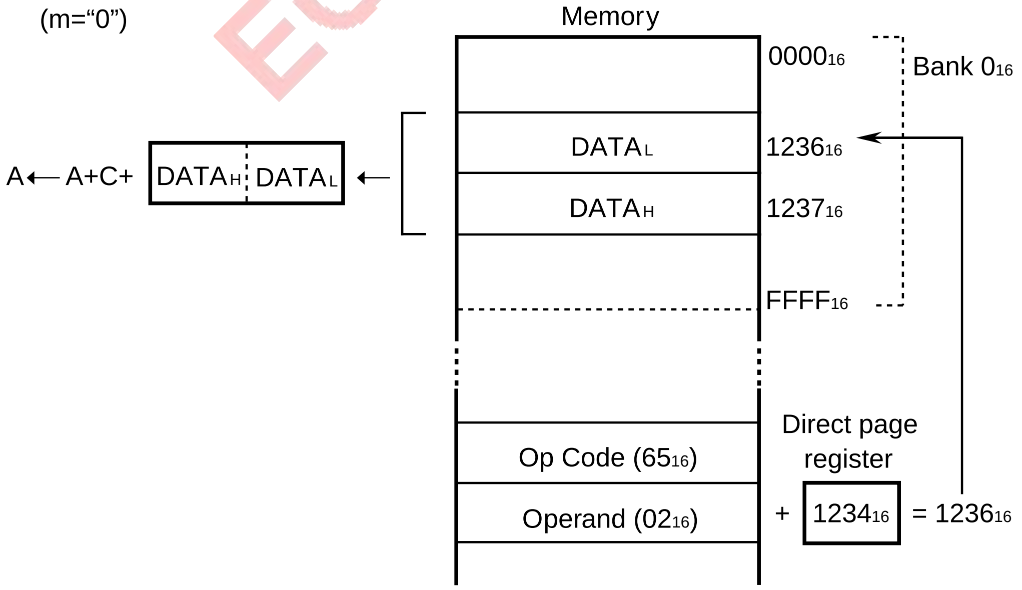
ex. : Mnemonic ADC A, 02H (m="1")

Machine code 65_{16} 02_{16}



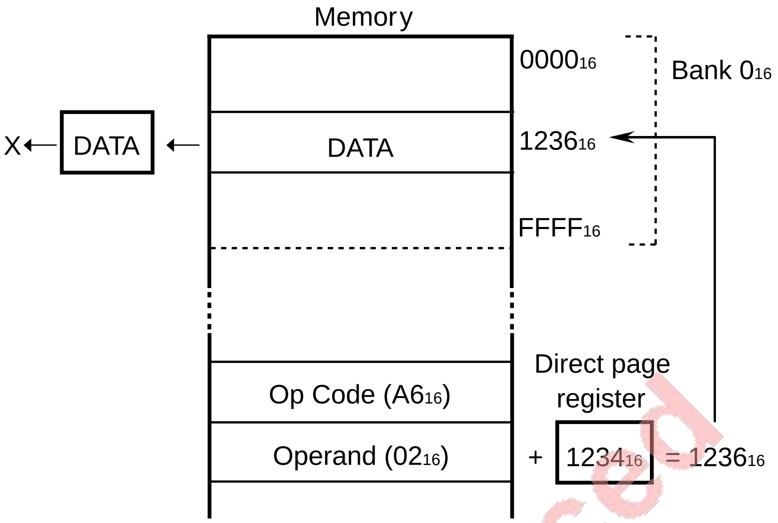
ex. : Mnemonic ADC A, 02H (m="0")

Machine code 65_{16} 02_{16}

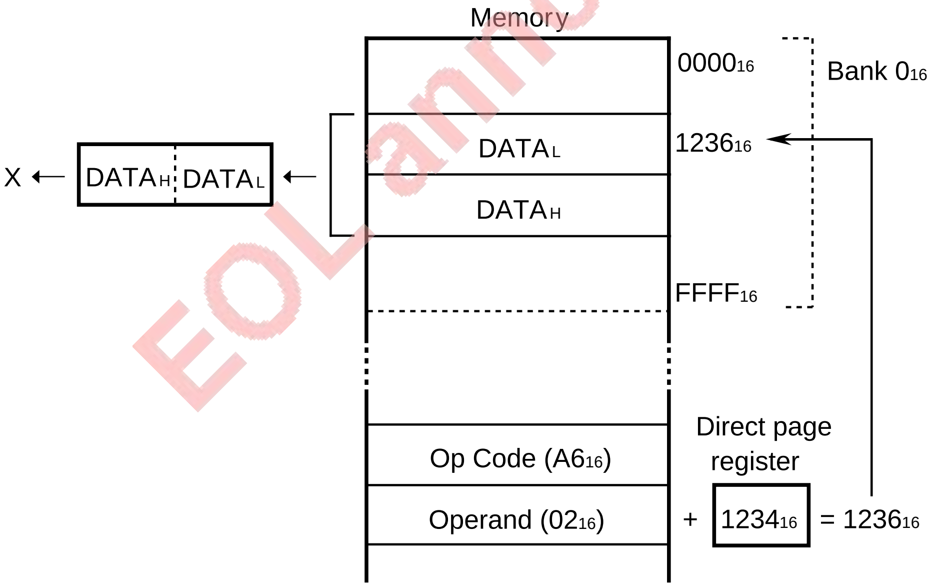


Direct

ex. : Mnemonic	Machine code
LDX 02H (x="1")	A6$_{16}$ 02$_{16}$



ex. : Mnemonic	Machine code
LDX 02H (x="0")	A6$_{16}$ 02$_{16}$



Direct Bit

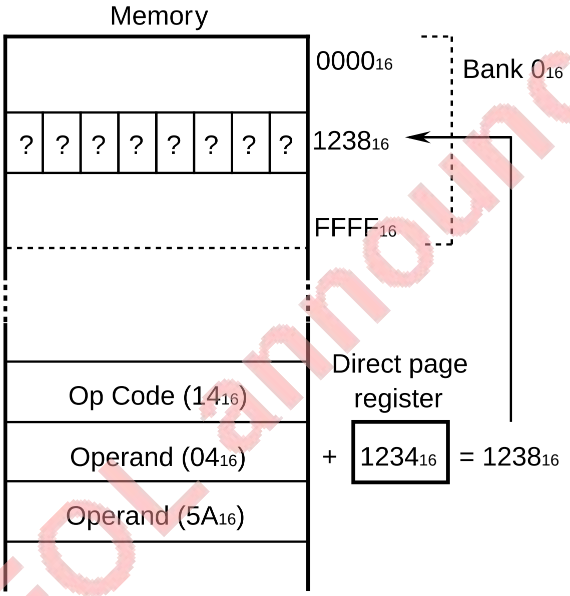
Mode : Direct bit addressing mode

Function : Specifies the memory location in bank 0_{16} by the result of adding the instruction's second byte to the direct page register's contents, and specifies the positions of multiple bits in the memory location by the bit pattern in the third and fourth bytes of the instruction (when the m flag is "1", the third byte only). When, however, the result of adding of the instruction's second byte to the direct page register's contents exceeds bank 0_{16} range, the memory location in bank 1_{16} is specified.

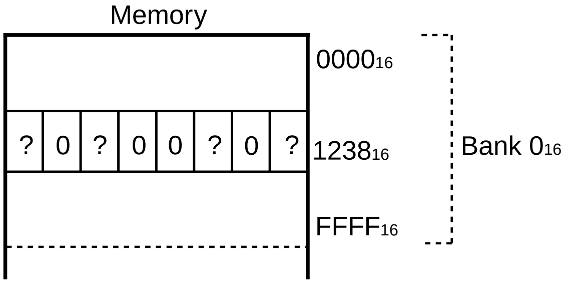
Instruction : CLB, SEB

ex. : Mnemonic	Machine code
CLB #5AH, 04H (m="1")	14_{16} 04_{16} $5A_{16}$

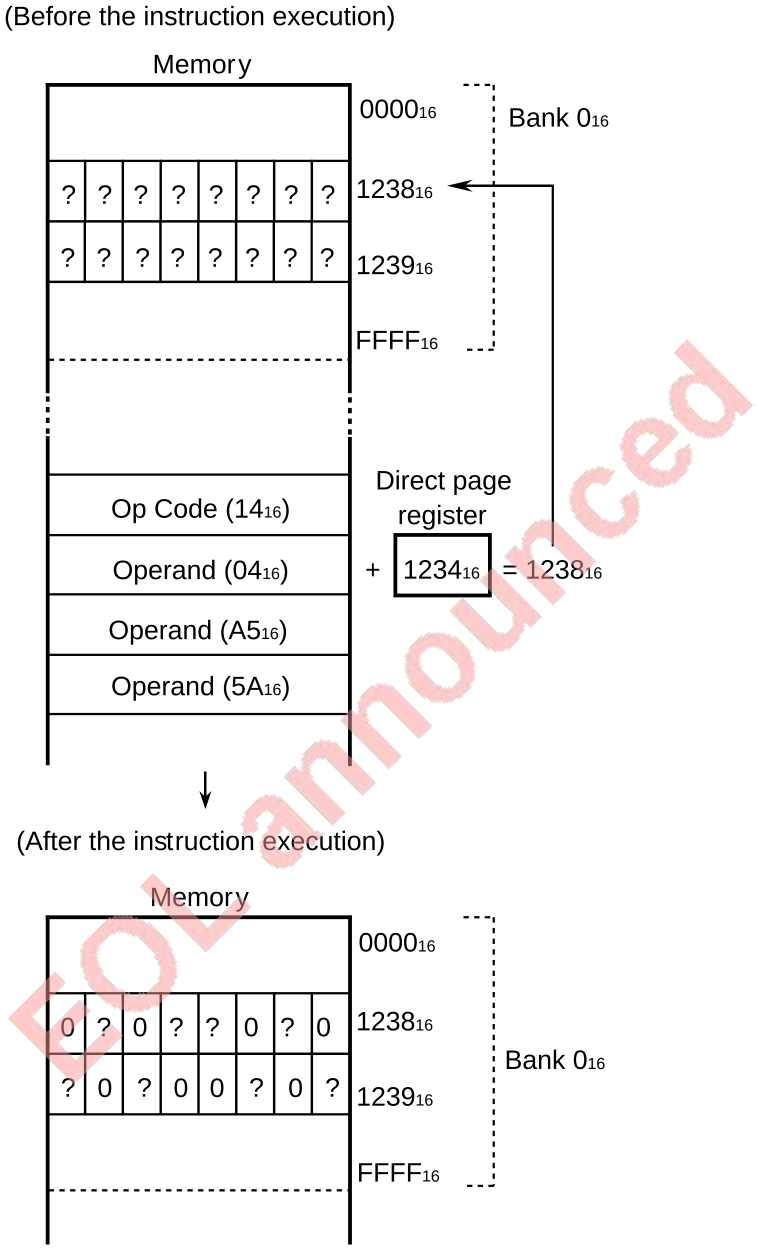
(Before the instruction execution)



(After the instruction execution)



ex. : Mnemonic	Machine code
CLB #5AA5H, 04H (m="0")	14_{16} 04_{16} $A5_{16}$ $5A_{16}$



Direct Indexed X

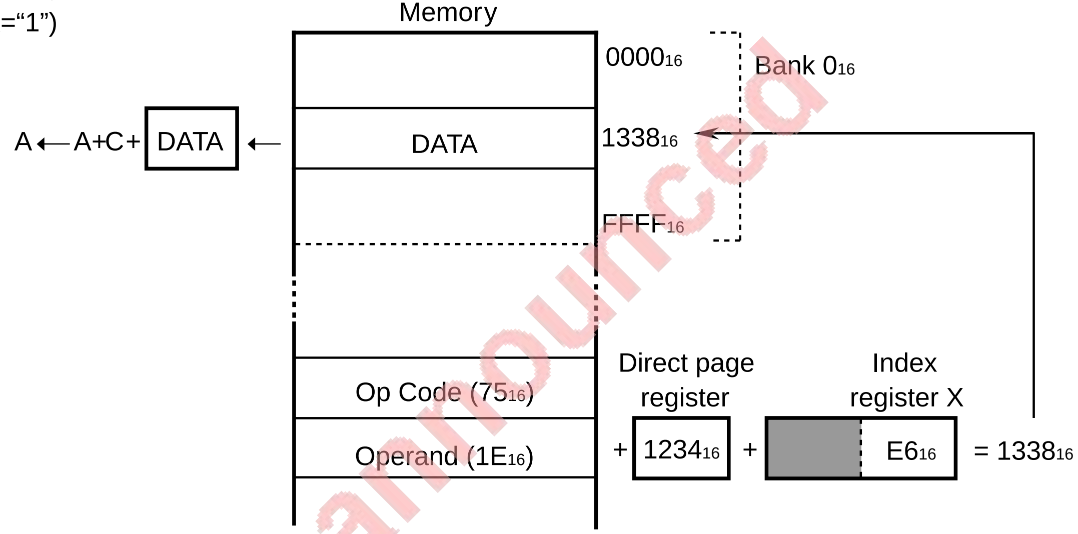
Mode : Direct indexed X addressing mode

Function : The contents of a memory in bank 0_{16} are an actual data. This memory location is specified by the result of adding the instruction's second byte, the direct page register's contents and the index register X's contents. When, however, the result of adding the instruction's second byte, the direct page register's contents and the index register X's contents exceeds bank 0_{16} or bank 1_{16} range, the memory location in bank 1_{16} or bank 2_{16} is specified.

Instruction : ADC, AND, ASL, ASR, CMP, DEC, DIV, DIVS, EOR, INC, LDA, LDM, LDY, LSR, MPY, MPYS, ORA, ROL, ROR, SBC, STA, STY

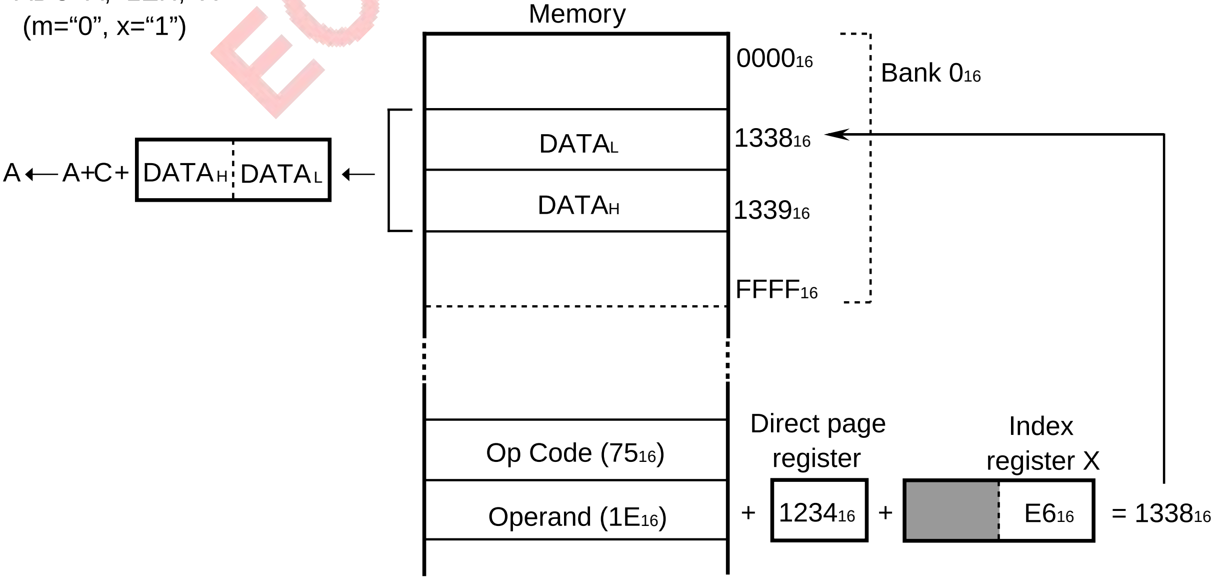
ex. : Mnemonic ADC A, 1EH, X (m="1", x="1")

Machine code 75_{16} $1E_{16}$



ex. : Mnemonic ADC A, 1EH, X (m="0", x="1")

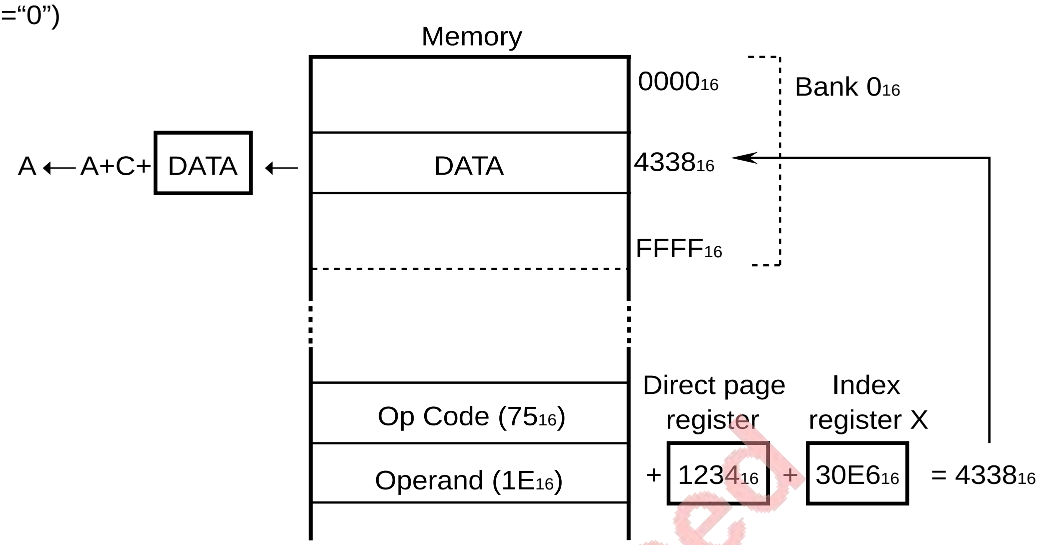
Machine code 75_{16} $1E_{16}$



Direct Indexed X

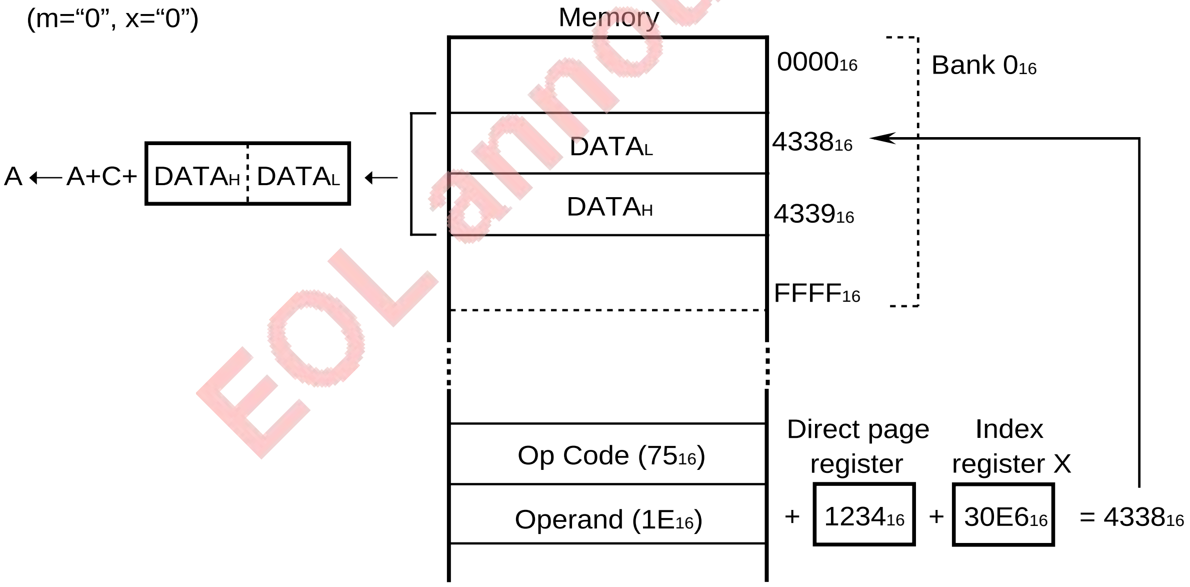
ex. : Mnemonic
ADC A, 1EH , X
(m="1", x="0")

Machine code
75_{16} $1E_{16}$



ex. : Mnemonic
ADC A, 1EH , X
(m="0", x="0")

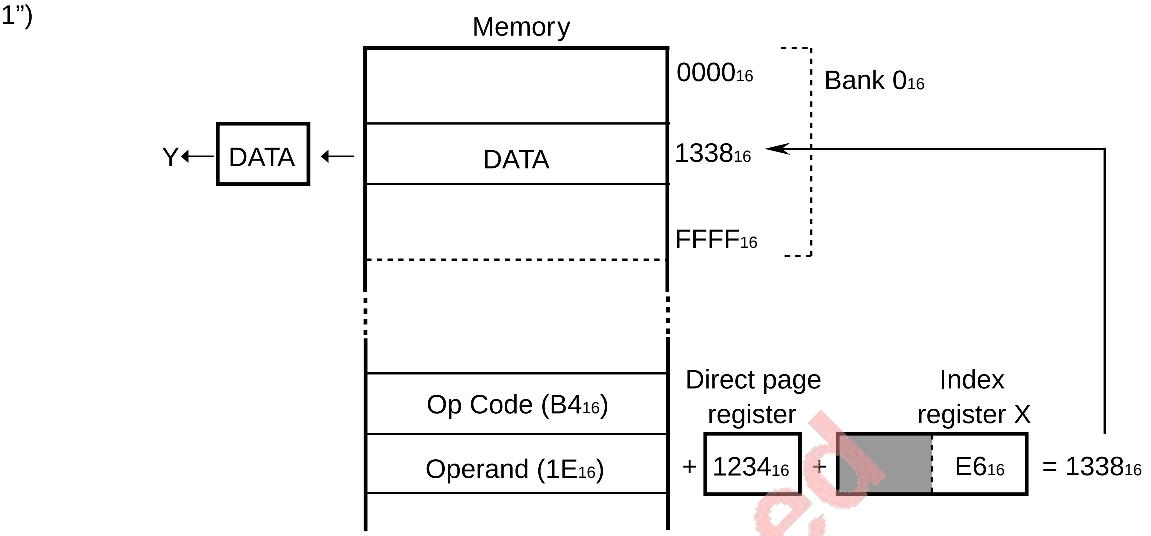
Machine code
75_{16} $1E_{16}$



Direct Indexed X

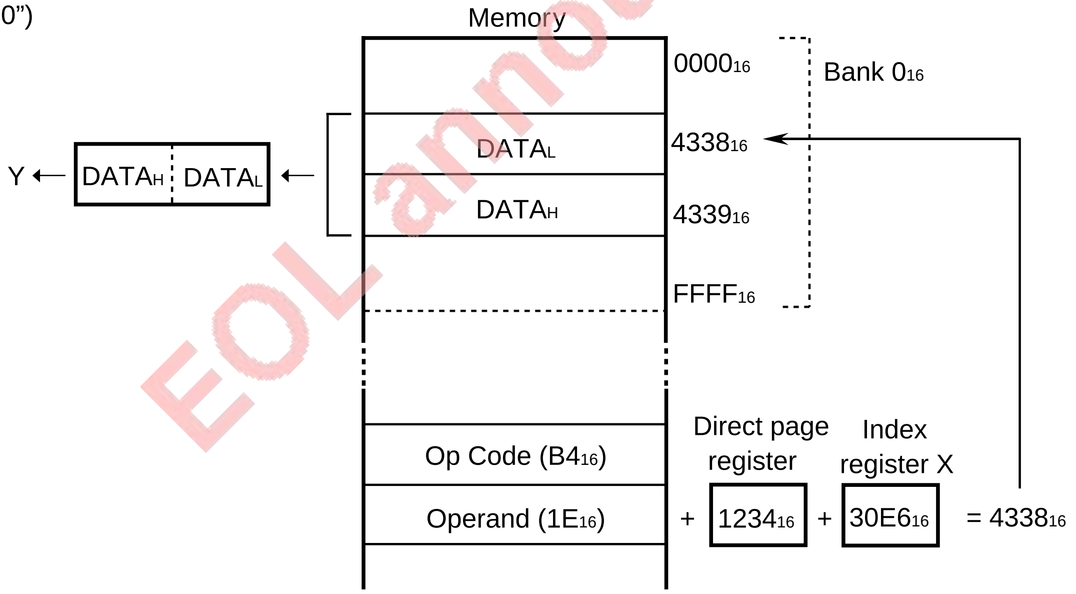
ex. : Mnemonic
LDY 1EH, X
(x="1")

Machine code
B4$_{16}$ 1E$_{16}$



ex. : Mnemonic
LDY 1EH, X
(x="0")

Machine code
B4$_{16}$ 1E$_{16}$



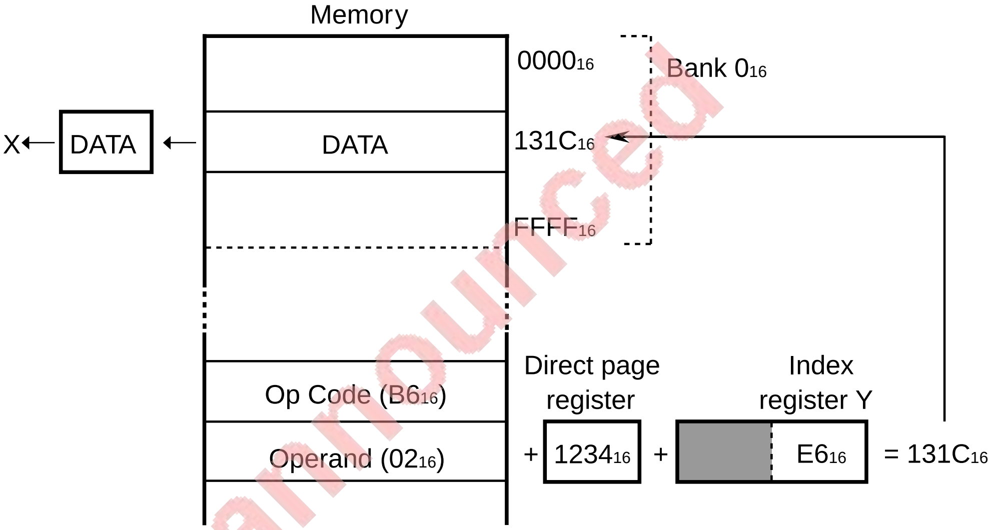
Direct Indexed Y

Mode : Direct indexed Y addressing mode

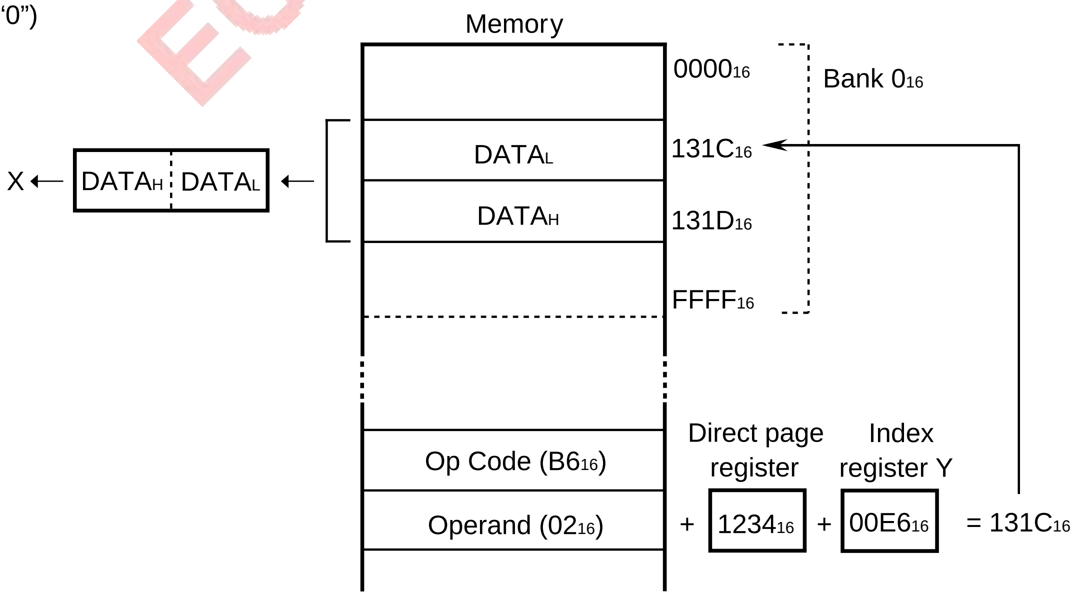
Function : The contents of a memory in bank 0_{16} are an actual data. This memory location is specified by the result of adding the instruction's second byte, the direct page register's contents and the index register Y's contents. When, however, the result of adding of the instruction's second byte, the direct page register's contents and the index register Y's contents exceeds bank 0_{16} or bank 1_{16} range, the memory location in bank 1_{16} or bank 2_{16} is specified.

Instruction : LDX, STX

ex. : Mnemonic LDX 02H , Y (x="1") Machine code $B6_{16}$ 02_{16}



ex. : Mnemonic LDX 02H , Y (x="0") Machine code $B6_{16}$ 02_{16}



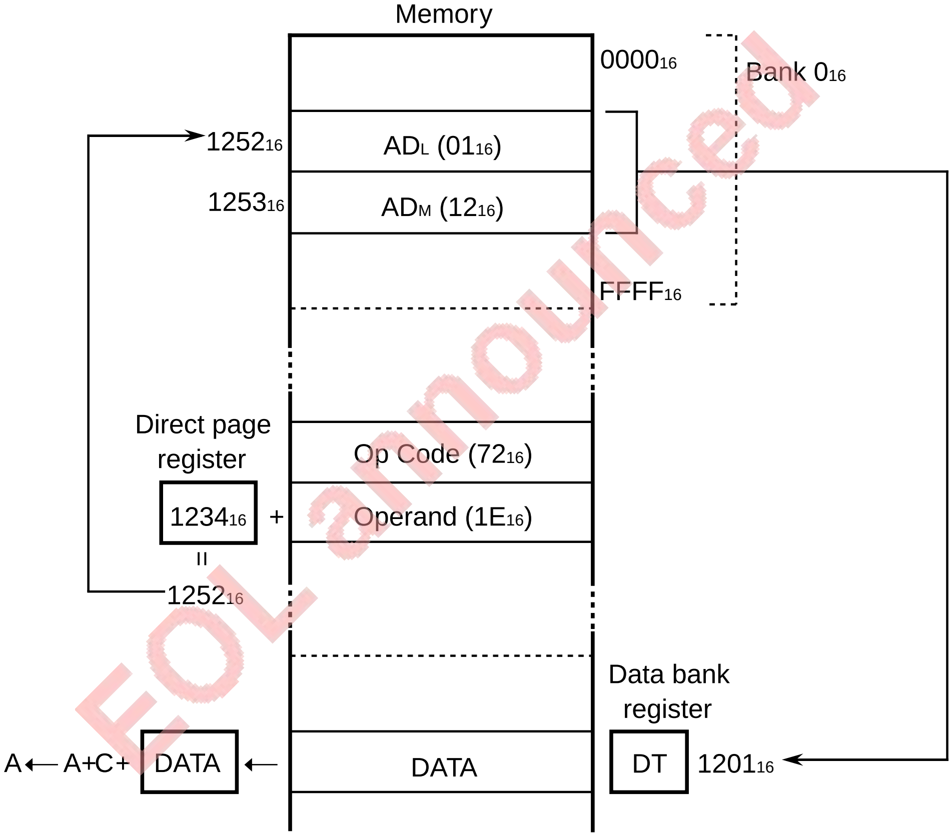
Direct Indirect

Mode : Direct indirect addressing mode

Function : Specifies a sequence of 2-byte memory in bank 0_{16} by the result of adding the instruction's second byte to the direct page register's contents. The contents of the memory location specified by these 2 bytes in bank DT (DT is the data bank register's contents) are an actual data. When, however, the result of adding the instruction's second byte to the direct page register's contents exceeds bank 0_{16} range, the memory location in bank 1_{16} is specified.

Instruction : ADC, AND, CMP, DIV, DIVS, EOR, LDA, MPY, MPYS, ORA, SBC, STA

ex. : Mnemonic	Machine code
ADC A, (1EH) (m="1")	72_{16} $1E_{16}$



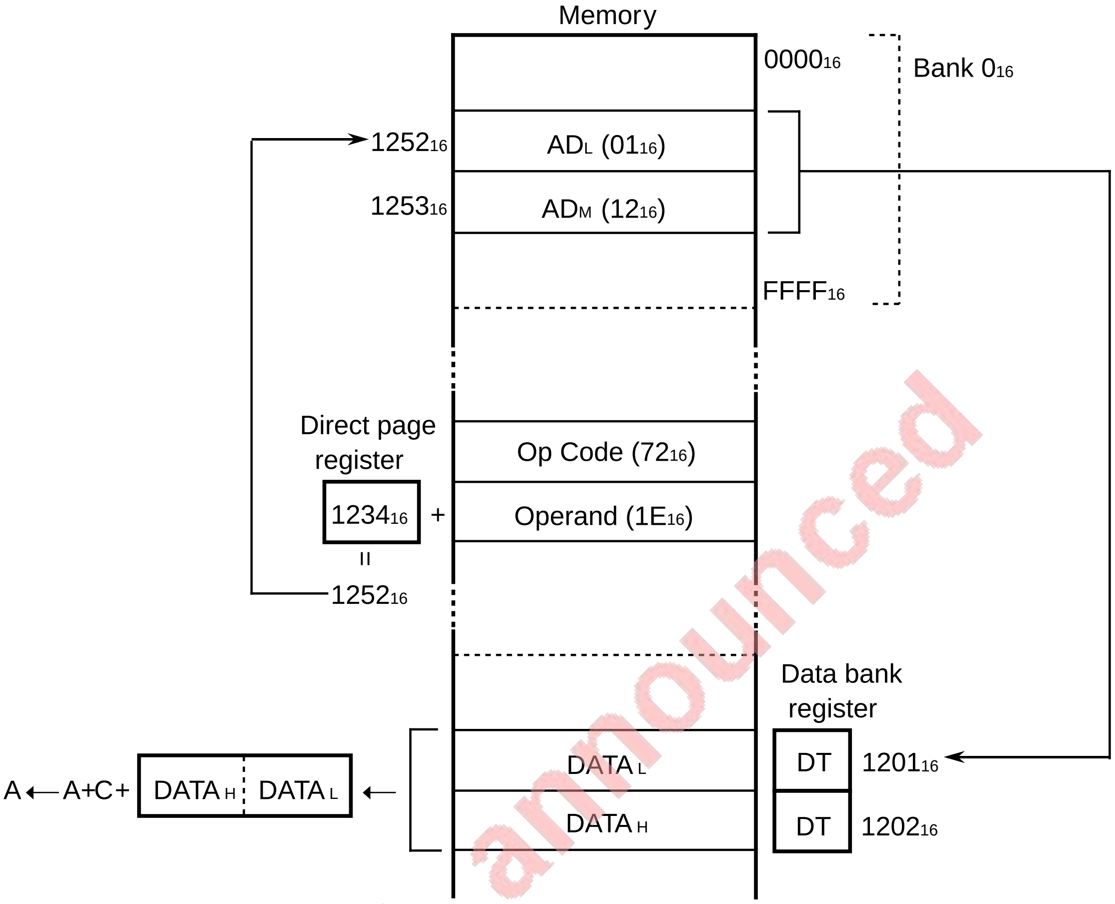
Direct Indirect

ex. : Mnemonic

ADC A, (1EH)
(m="0")

Machine code

72_{16} $1E_{16}$



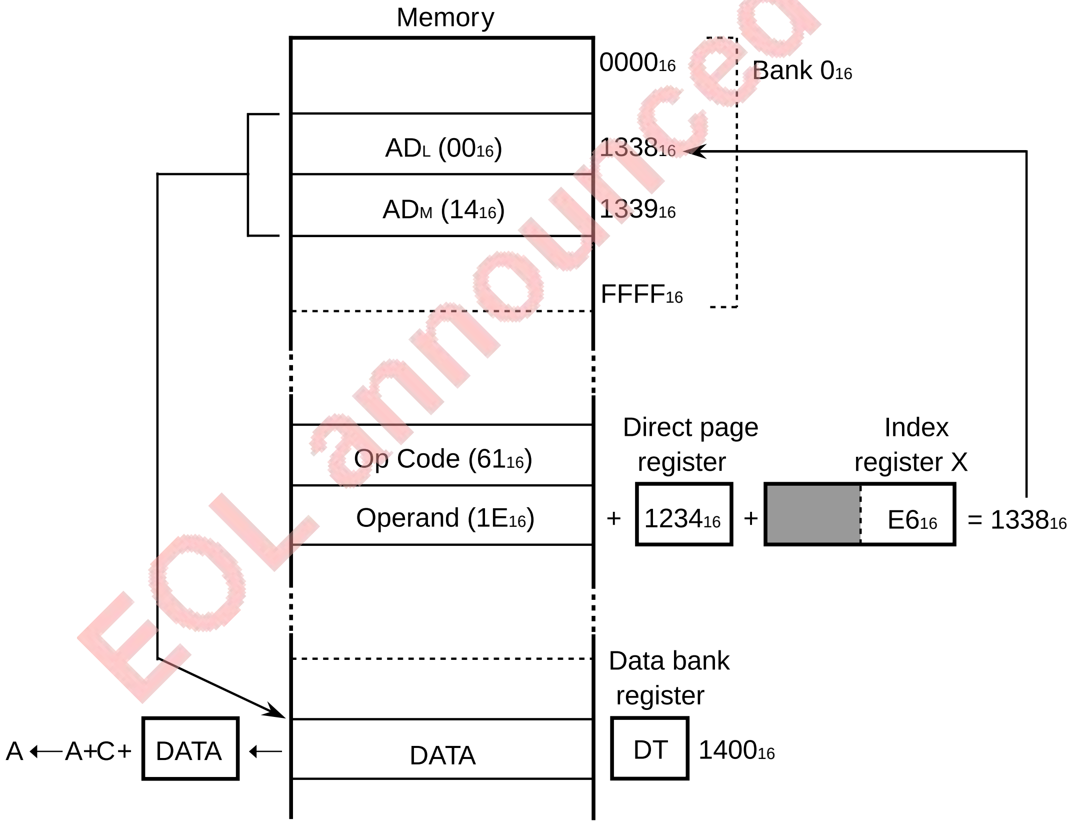
Direct Indexed X Indirect

Mode : Direct indexed X indirect addressing mode

Function : Specifies a sequence of 2-byte memory in bank 0_{16} by the result of adding the instruction's second byte, the direct page register's contents and the index register X's contents. The contents of the memory location specified by these bytes in bank DT (DT is the data bank register's contents) are an actual data. When, however, the result of adding the instruction's second byte, the direct page register's contents and the index register X's contents exceeds bank 0_{16} or bank 1_{16} range, the memory location in bank 1_{16} or bank 2_{16} is specified.

Instruction : ADC, AND, CMP, DIV, DIVS, EOR, LDA, MPY, MPYS, ORA, SBC, STA

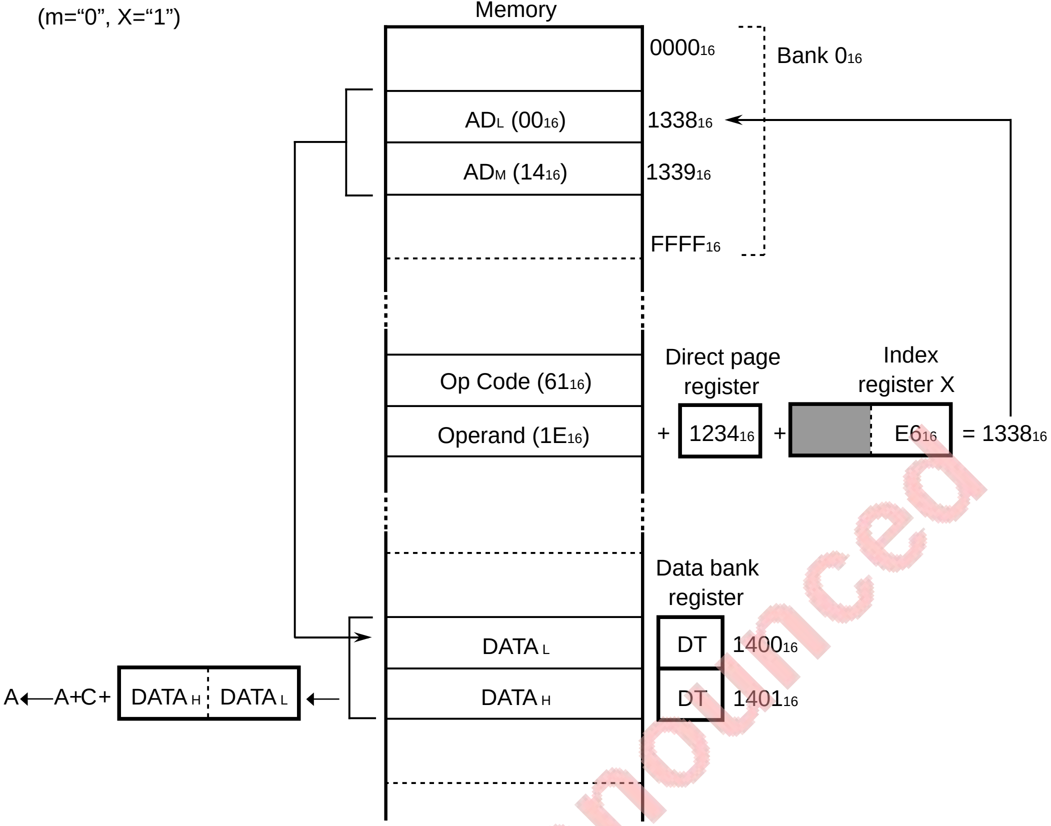
ex. : Mnemonic	Machine code
ADC A, (1EH, X) (m="1", x="1")	61_{16} $1E_{16}$



Direct Indexed X Indirect

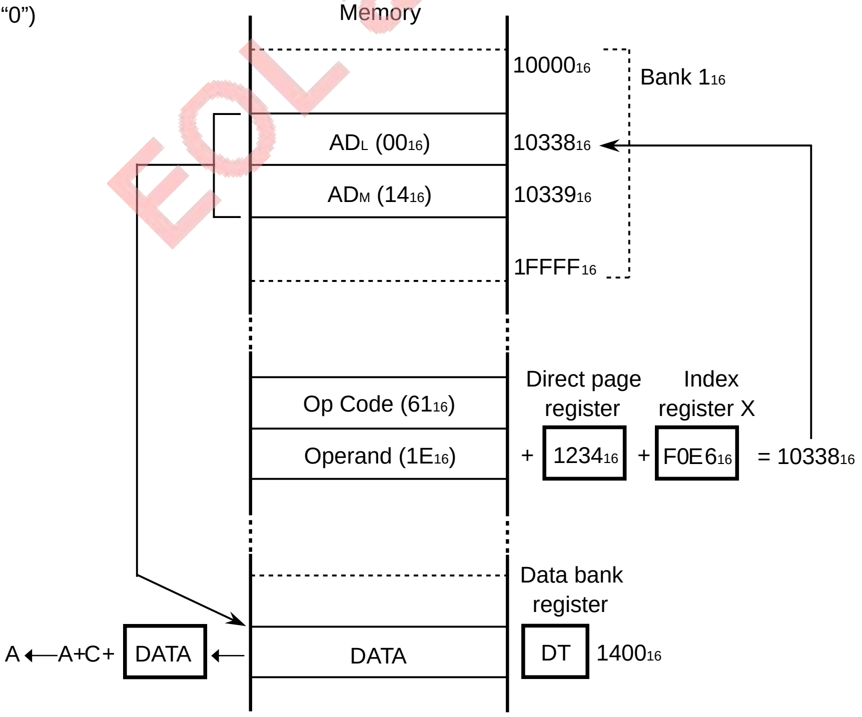
ex. : Mnemonic
ADC A, (1EH, X)
(m=“0”, X=“1”)

Machine code
61_{16} $1E_{16}$



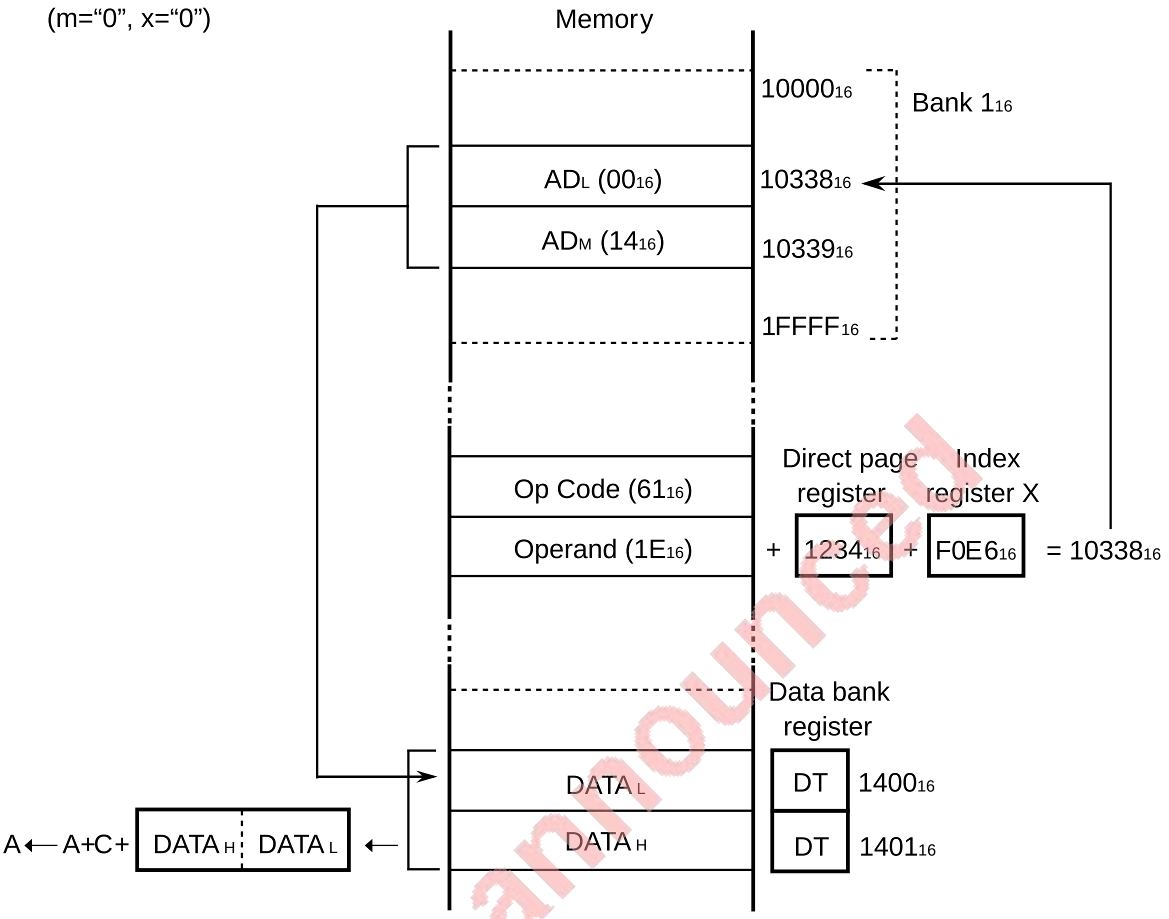
ex. : Mnemonic
ADC A, (1EH,X)
(m=“1”, x=“0”)

Machine code
61_{16} $1E_{16}$



Direct Indexed X Indirect

ex. : Mnemonic   Machine code
ADC A, (1EH, X) 61_{16} $1E_{16}$
(m="0", x="0")



Direct Indirect Indexed Y

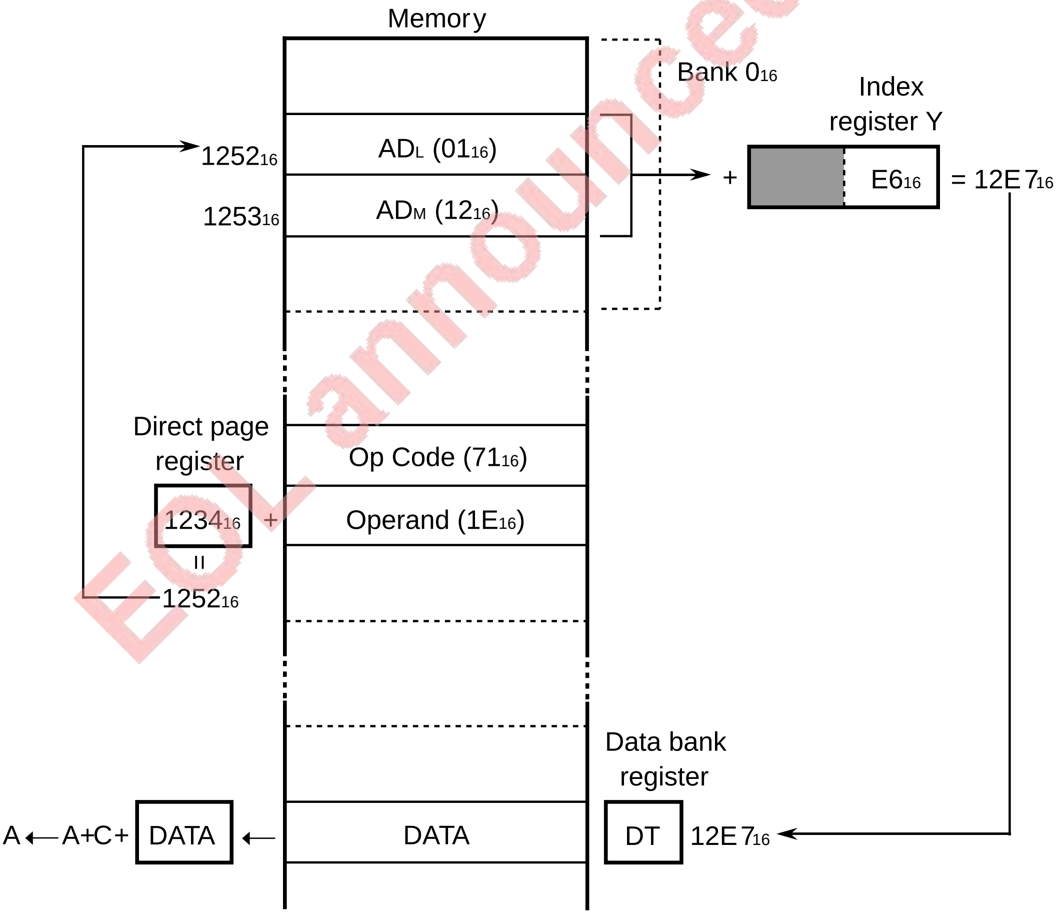
Mode : Direct indirect indexed Y addressing mode

Function : Specifies a sequence of 2-byte memory in bank 0_{16} by the result of adding the instruction's second byte to the direct page register's contents. The following is an actual data: the contents of the memory location specified by the result of adding the contents of these 2 bytes to the index register Y's contents and the contents of the data bank register. When, however, the result of adding the instruction's second byte to the direct page register's contents exceeds bank 0_{16} range, the memory location in bank 1_{16} is specified. Additionally, if the addition of the memory's contents and the index register Y's contents generates a carry, the bank which is 1 larger than the contents of the data bank register is used.

Instruction : ADC, AND, CMP, DIV, DIVS, EOR, LDA, MPY, MPYS, ORA, SBC, STA

ex. : Mnemonic
ADC A, (1EH), Y
(m="1", x="1")

Machine code
71_{16} $1E_{16}$

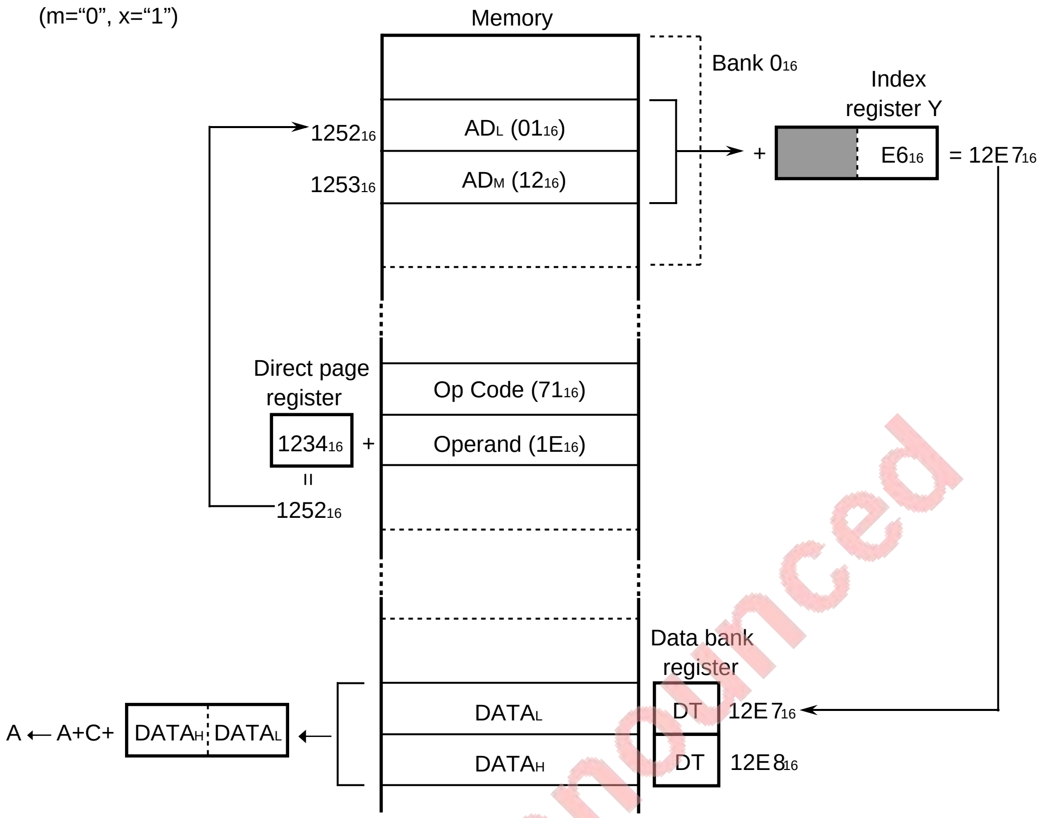


Direct Indirect Indexed Y

ex. : Mnemonic   Machine code

ADC A, (1EH), Y 71_{16} $1E_{16}$

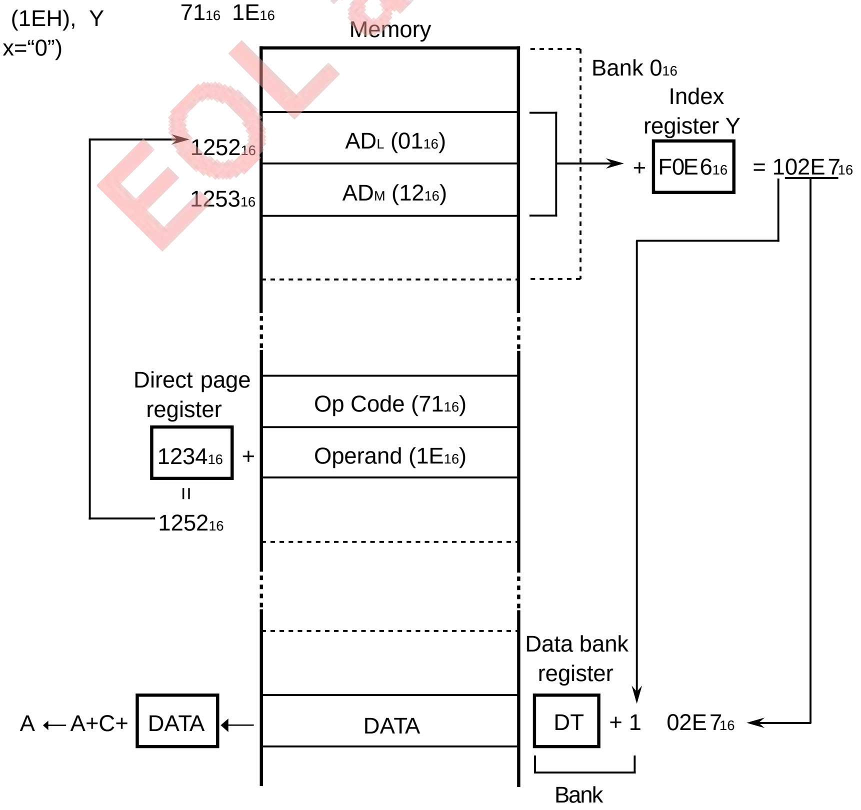
(m="0", x="1")



ex. : Mnemonic   Machine code

ADC A, (1EH), Y 71_{16} $1E_{16}$

(m="1", x="0")

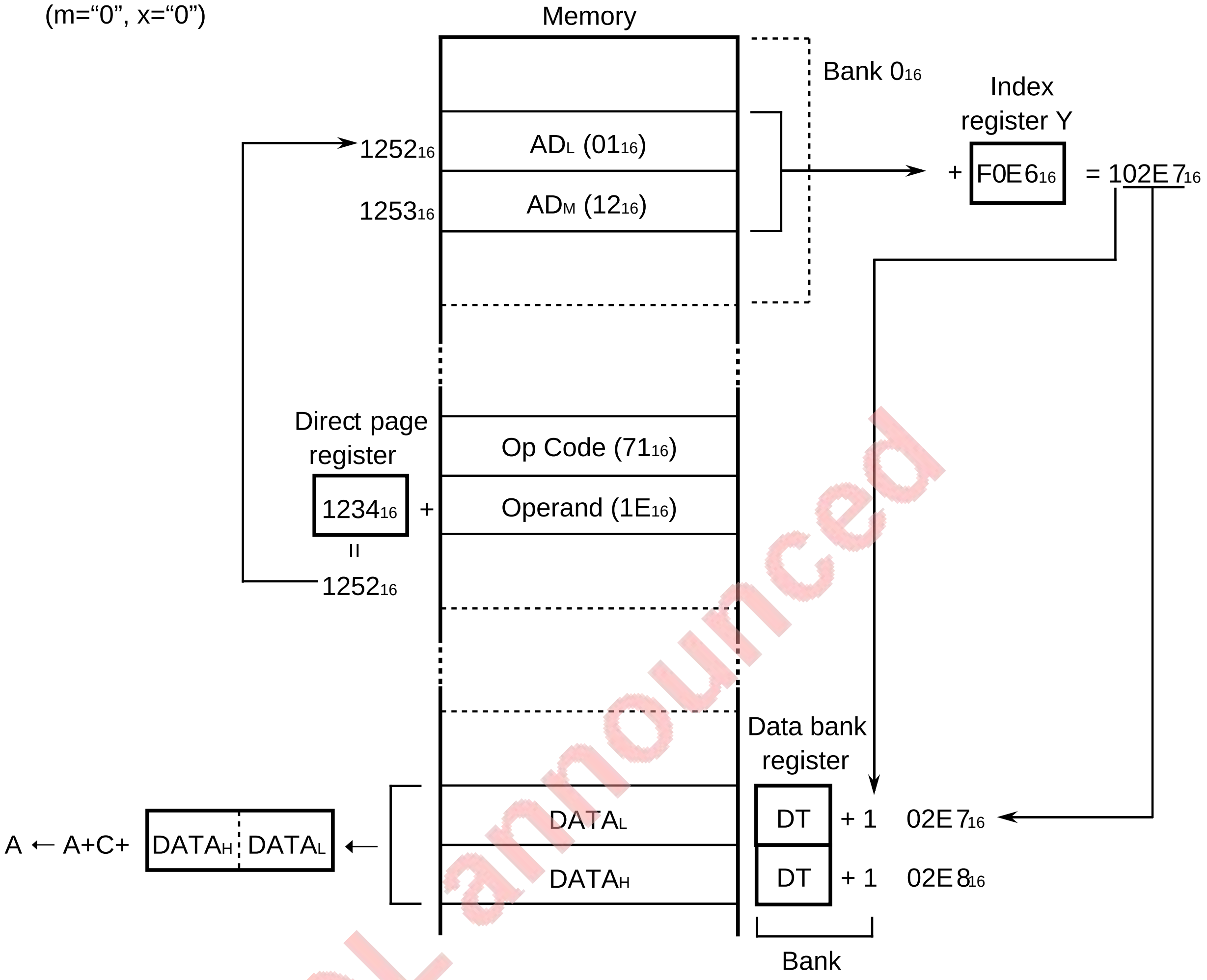


Direct Indirect Indexed Y

ex. : Mnemonic

ADC A, (1EH), Y
(m="0", x="0")

Machine code
71_{16} $1E_{16}$



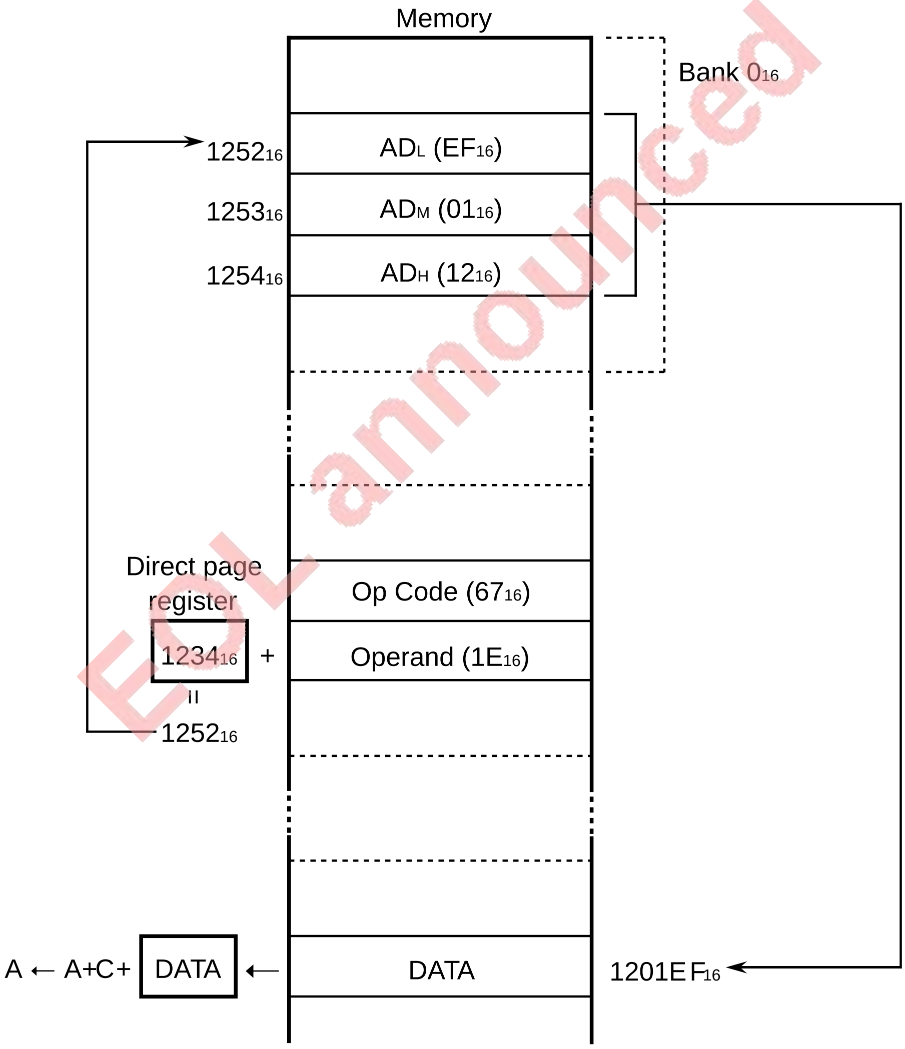
Direct Indirect Long

Mode : Direct indirect long addressing mode

Function : Specifies a sequence of 3-byte memory in bank 0_{16} by the result of adding the instruction's second byte to the direct page register's contents. The contents at the address specified by the contents of these 3 bytes are an actual data. When, however, the result of adding the instruction's second byte to the direct page register's contents exceeds bank 0_{16}, the memory location in bank 1_{16} is specified. A sequence of 3-byte memory can cross over the bank boundary.

Instruction : ADC, AND, CMP, DIV, DIVS, EOR, LDA, MPY, MPYS, ORA, SBC, STA

ex. : Mnemonic	Machine code
ADCL A, (1EH) (m="1")	67_{16} $1E_{16}$



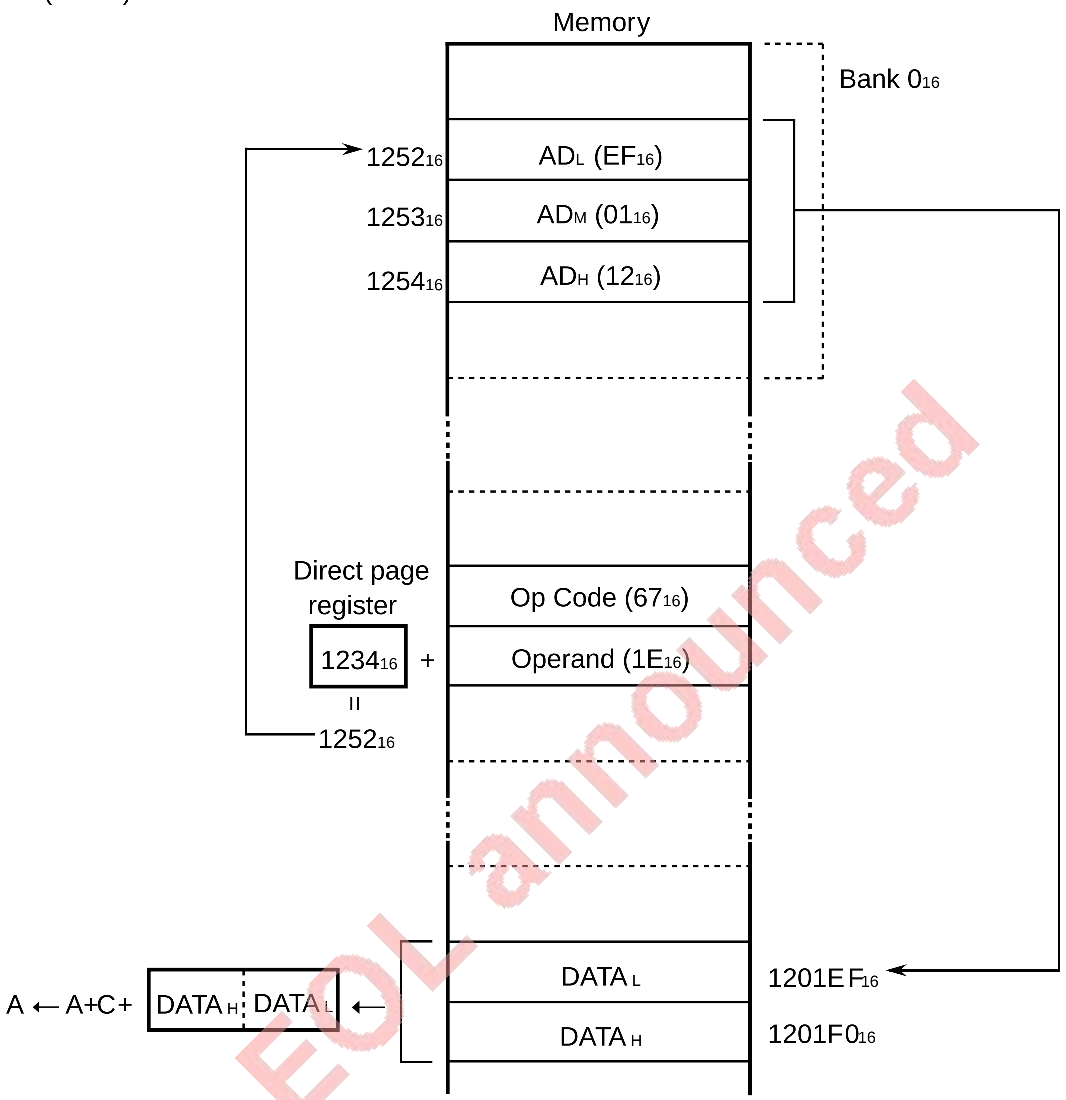
Direct Indirect Long

ex. : Mnemonic

ADCL A, (1EH)
(m="0")

Machine code

67_{16} $1E_{16}$



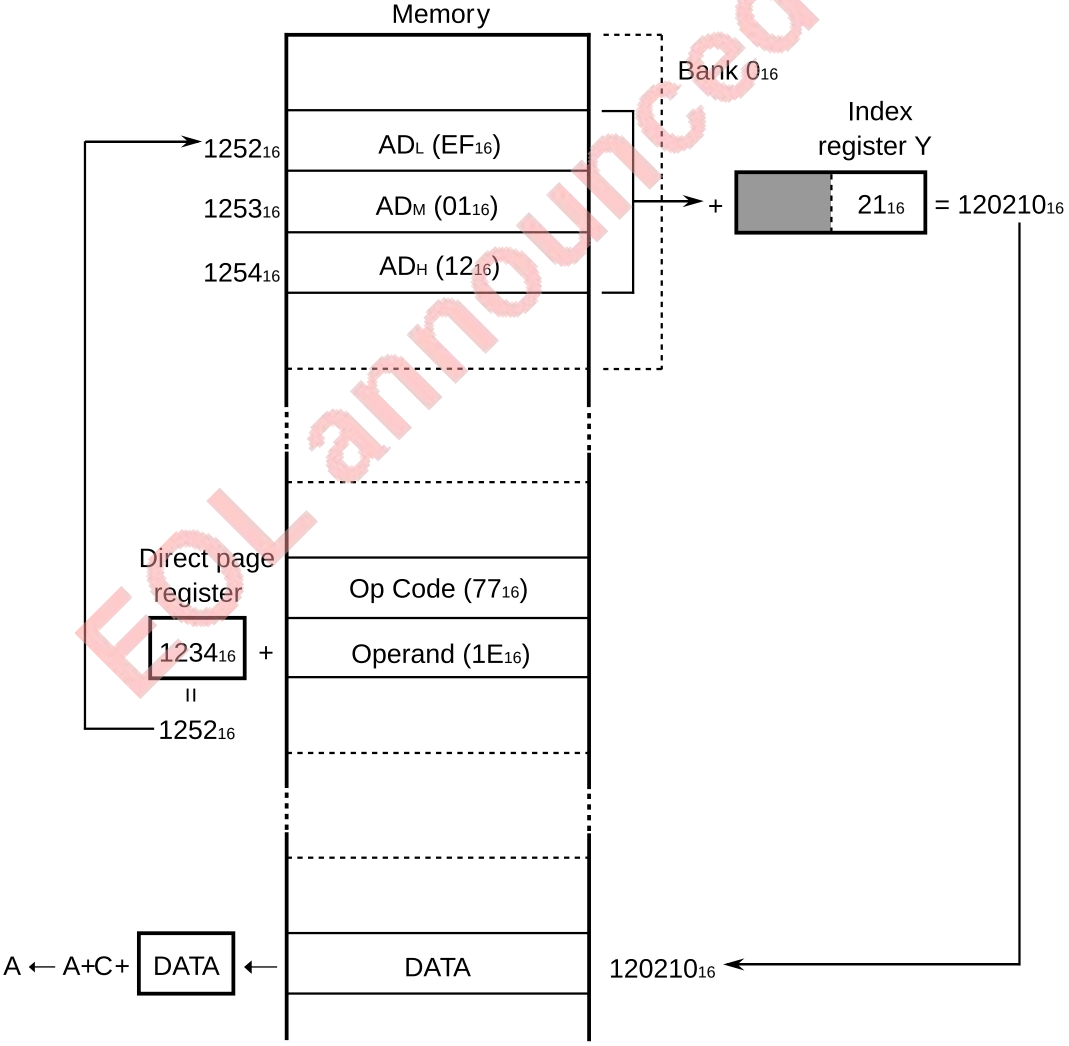
Direct Indirect Long Indexed Y

Mode : Direct indirect long indexed Y addressing mode

Function : Specifies a sequence of 3-byte memory in bank 0_{16} by the result of adding the instruction's second byte to the direct page register's contents. The contents at the address specified by the result of adding the contents of these 3 bytes to the index register Y's contents are an actual data. When, however, the result of adding the instruction's second byte to the direct page register's contents exceeds bank 0_{16} range, the memory location in bank 1_{16} is specified. A sequence of 3-byte memory can cross over the bank boundary.

Instruction : ADC, AND, CMP, DIV, DIVS, EOR, LDA, MPY, MPYS, ORA, SBC, STA

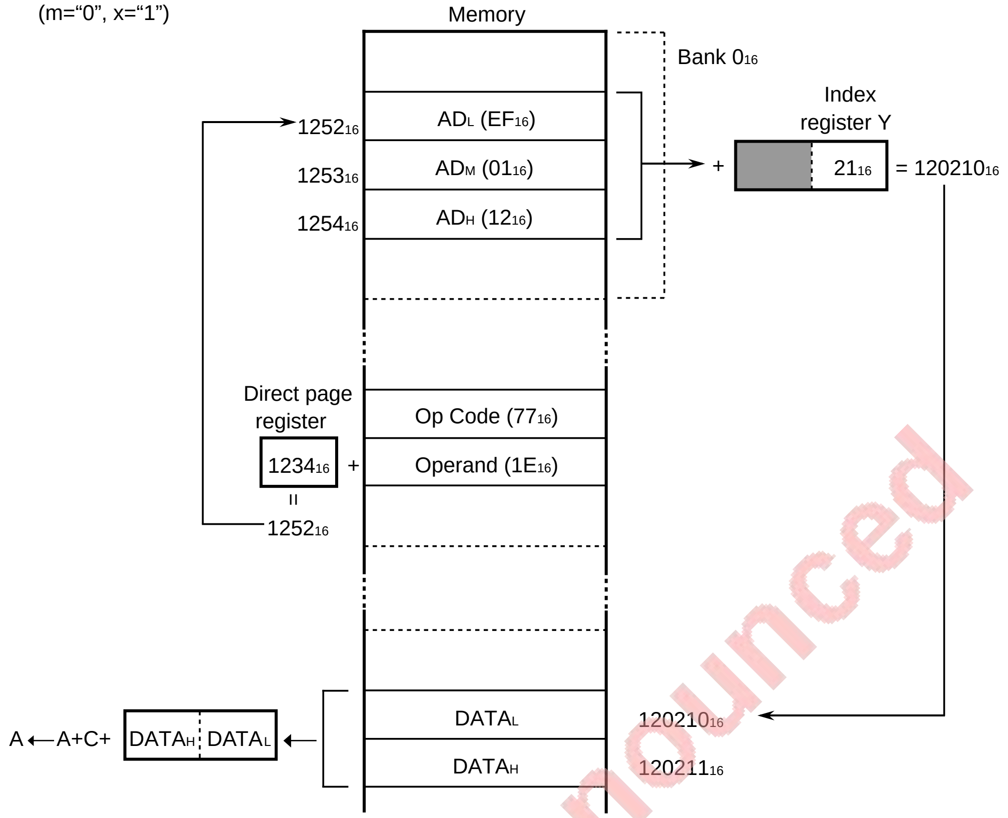
ex. : Mnemonic	Machine code
ADCL A, (1EH), Y (m="1", x="1")	77_{16} $1E_{16}$



Direct Indirect Long Indexed Y

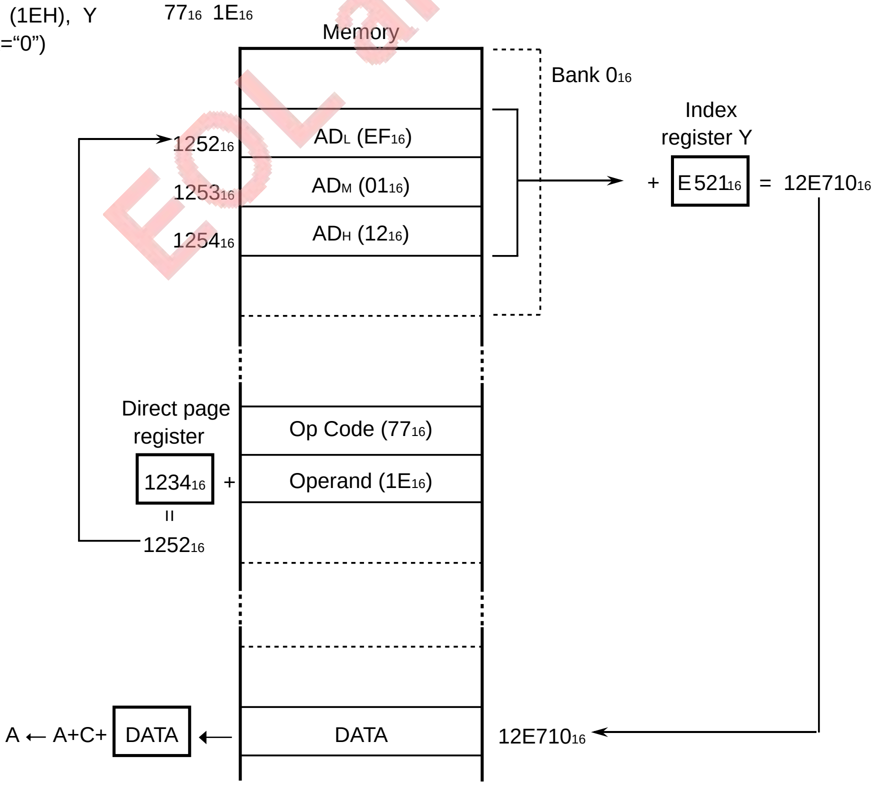
ex. : Mnemonic
ADCL A, (1EH), Y
(m="0", x="1")

Machine code
77_{16} $1E_{16}$



ex. : Mnemonic
ADCL A, (1EH), Y
(m="1", x="0")

Machine code
77_{16} $1E_{16}$



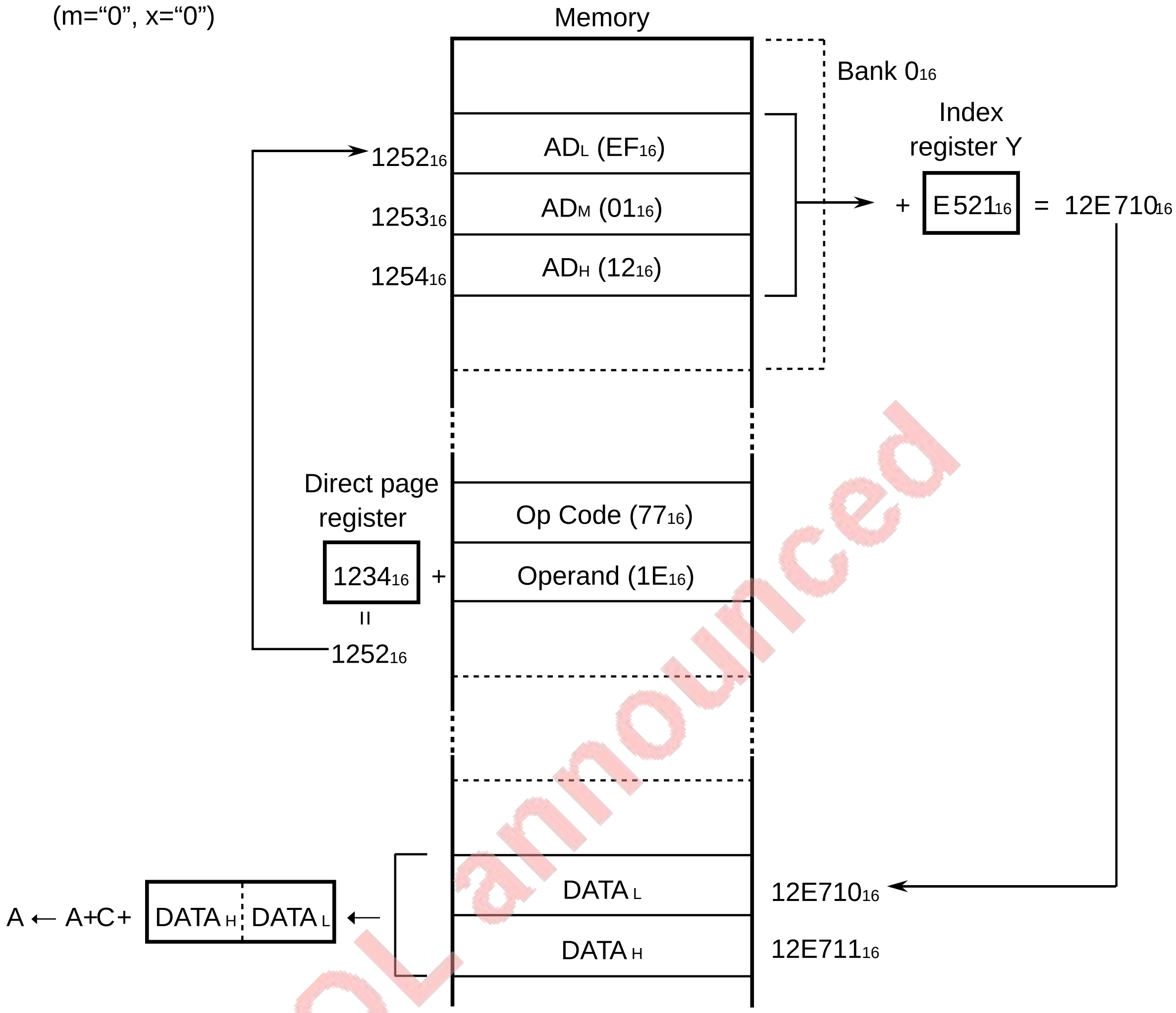
Direct Indirect Long Indexed Y

ex. : Mnemonic

ADCL A, (1EH), Y
(m=“0”, x=“0”)

Machine code

77_{16} $1E_{16}$



Absolute

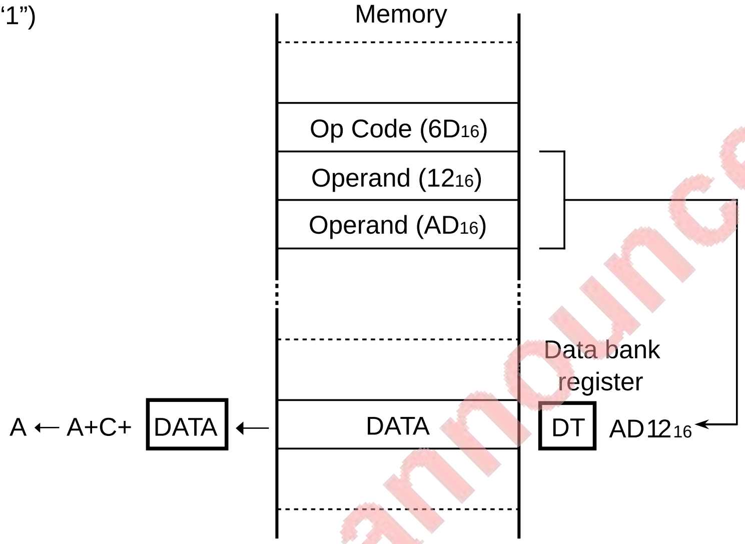
Mode : Absolute addressing mode

Function : The following is an actual data: the contents of the memory location specified by the instruction's second and third bytes and the contents of the data bank register. Note that, in the cases of the JMP and JSR instructions, the instructions' second and third bytes are transferred to the program counter.

Instruction : ADC, AND, ASL, ASR, CMP, CPX, CPY, DEC, DIV, DIVS, EOR, INC,
JMP, JSR, LDA, LDM, LDX, LDY, LSR, MPY, MPYS, ORA, ROL, ROR,
SBC, STA, STX, STY

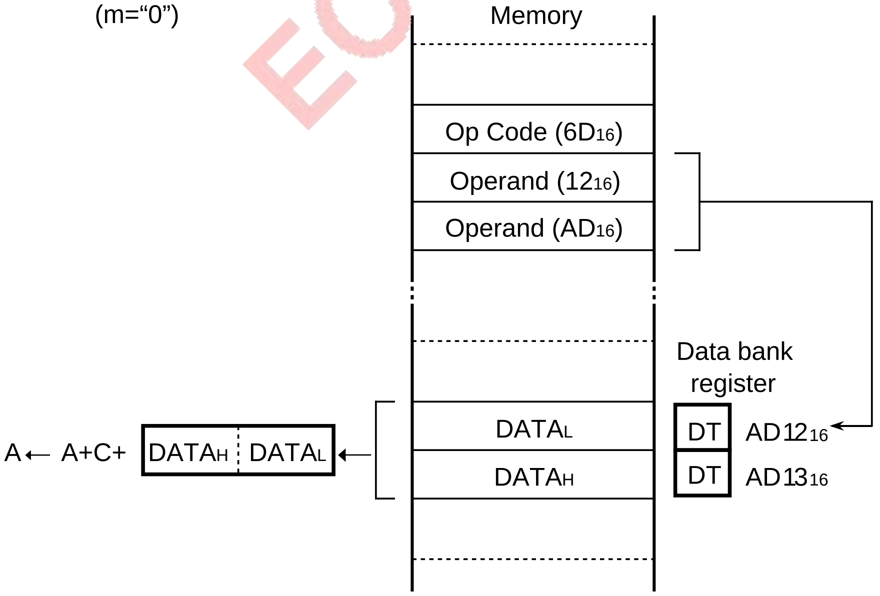
ex.: Mnemonic
ADC A, 0AD12H
(m="1")

Machine code
$6D_{16}$ 12_{16} AD_{16}



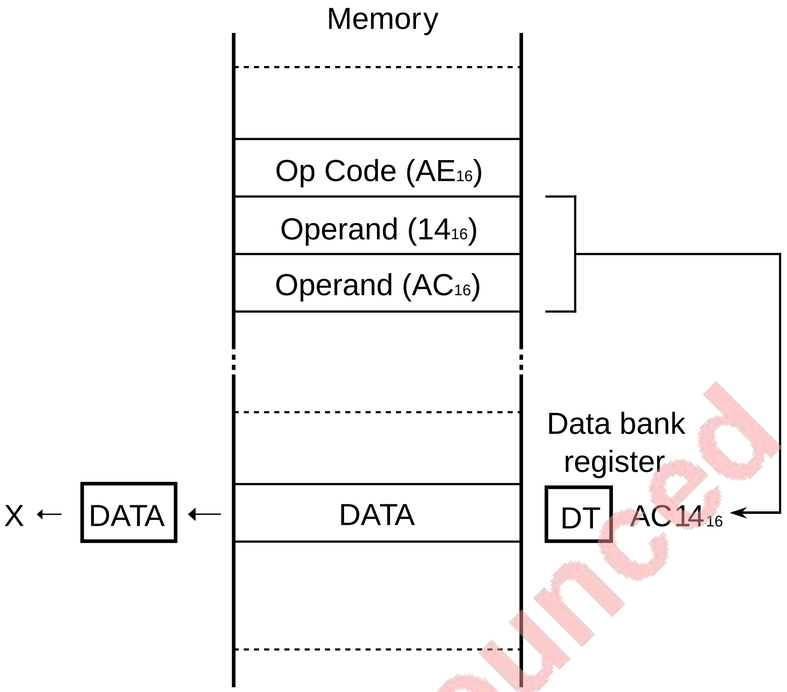
ex. : Mnemonic
ADC A, 0AD12H
(m="0")

Machine code
$6D_{16}$ 12_{16} AD_{16}



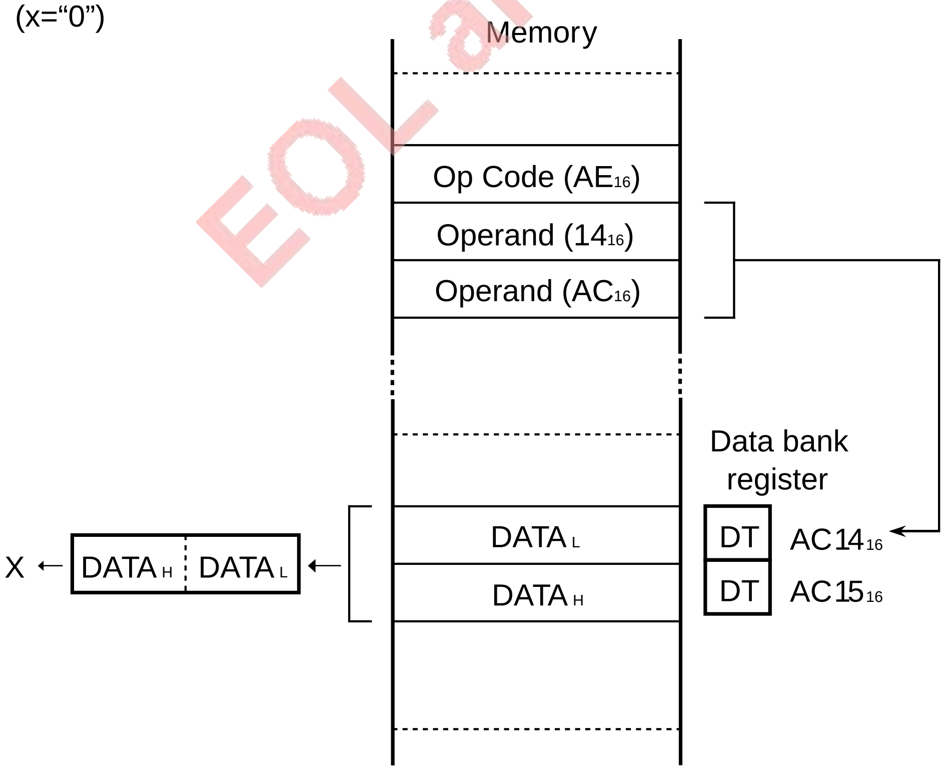
ex. : Mnemonic
LDX 0AC14H
(x="1")

Machine code
AE_{16} 14_{16} AC_{16}



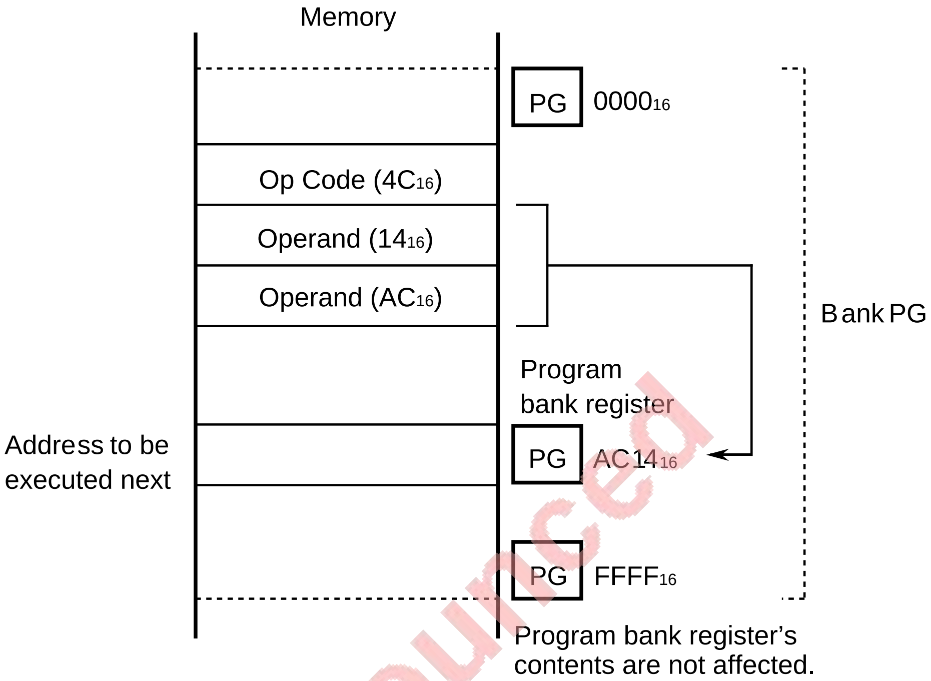
ex. : Mnemonic
LDX 0AC14H
(x="0")

Machine code
AE_{16} 14_{16} AC_{16}



ex. : Mnemonic JMP 0AC14H

Machine code $4C_{16}$ 14_{16} AC_{16}



Note : Note the branch destination bank when a JMP or a JSR instruction is located near a bank boundary.
⇒Refer to the description of a JMP instruction (Page 4-50).
Refer to the description of a JSR instruction (Page 4-51).

Absolute Bit

Mode : Absolute bit addressing mode

Function : Specifies the memory location by the instruction's second and third bytes and the contents of the data bank register; specifies the multiple bit positions in that memory by a bit pattern of the instruction's fourth and fifth bytes (when the m flag is "1", the fourth byte only).

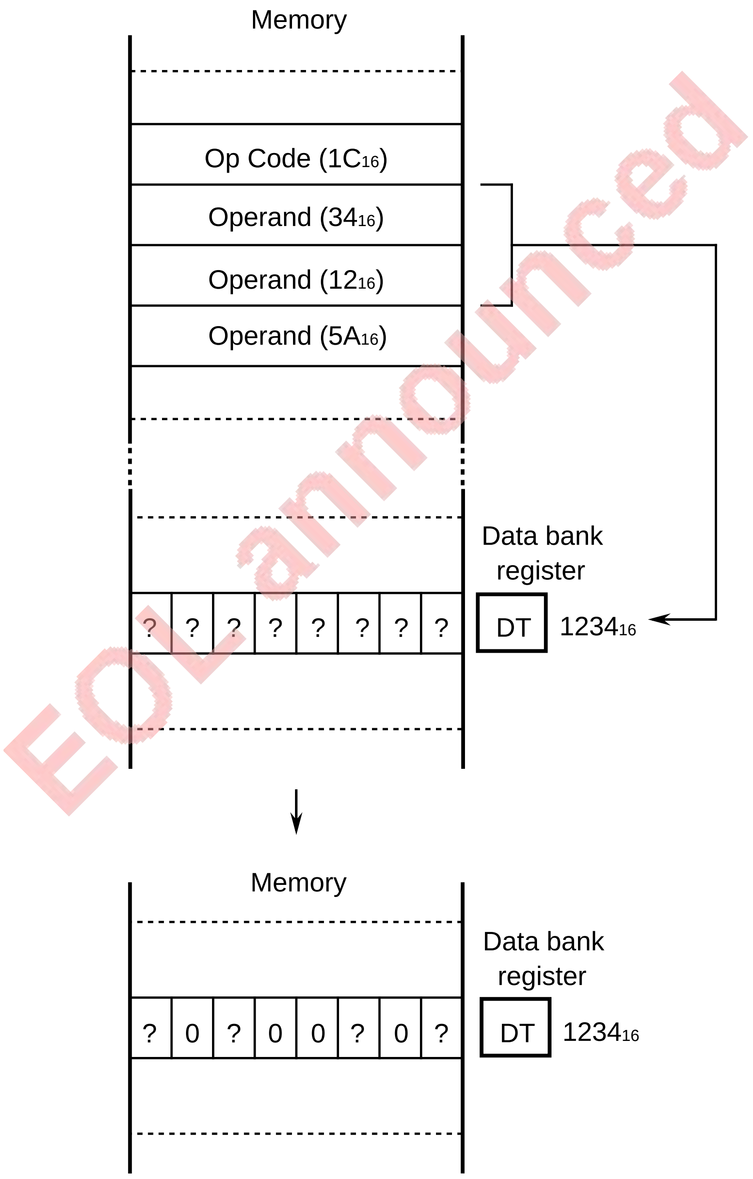
Instruction : CLB, SEB

ex. : Mnemonic

CLB #5AH , 1234H
(m="1")

Machine code

$1C_{16}$ 34_{16} 12_{16} $5A_{16}$

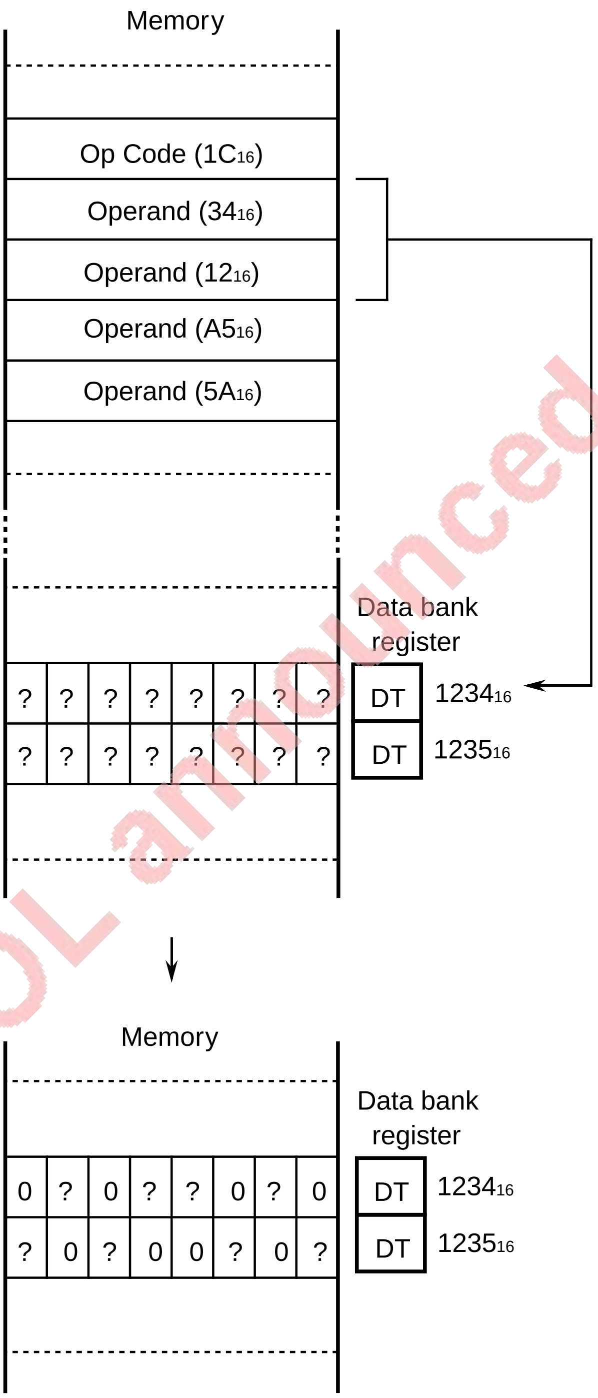


ex. : Mnemonic

CLB #5AA5H, 1234H
(m=“0”)

Machine code

$1C_{16}$ 34_{16} 12_{16} $A5_{16}$ $5A_{16}$



Absolute Indexed X

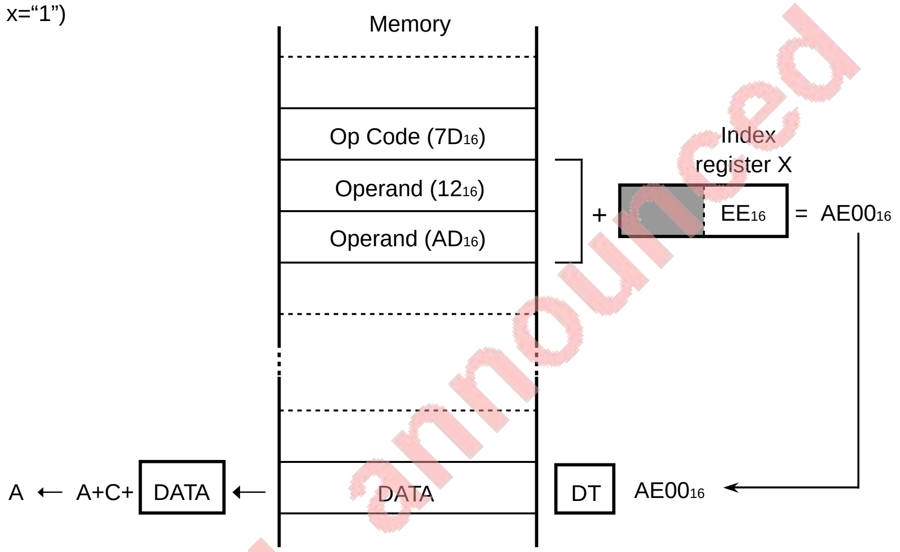
Mode : Absolute indexed X addressing mode

Function : The following is an actual data: the contents of the memory location specified by the result of adding a 16-bit length numerical value expressed with the instruction's second and third bytes to the index register X's contents, and the contents of the data bank register. If, however, the addition of the numerical value expressed with the instruction's second and third bytes, and the index register X's contents generates a carry; the bank which is 1 larger than the contents of the data bank register is used.

Instruction : ADC, AND, ASL, ASR, CMP, DEC, DIV, DIVS, EOR, INC, LDA, LDM,
LDY, LSR, MPY, MPYS, ORA, ROL, ROR, SBC, STA

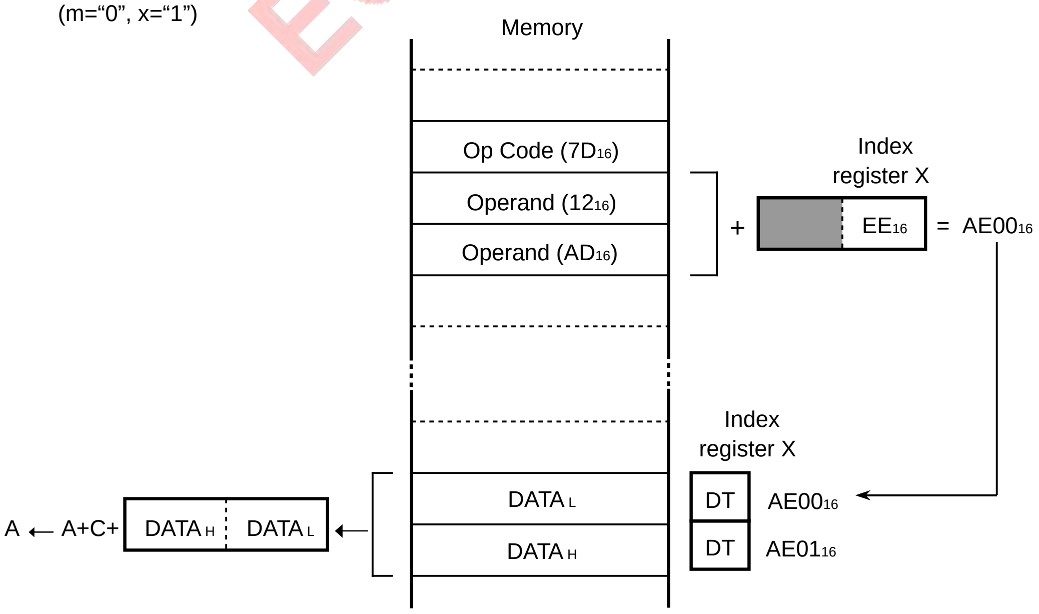
ex. : Mnemonic
ADC A, 0AD12H, X
(m="1", x="1")

Machine code
$7D_{16}$ 12_{16} AD_{16}



ex. : Mnemonic
ADC A, 0AD12H, X
(m="0", x="1")

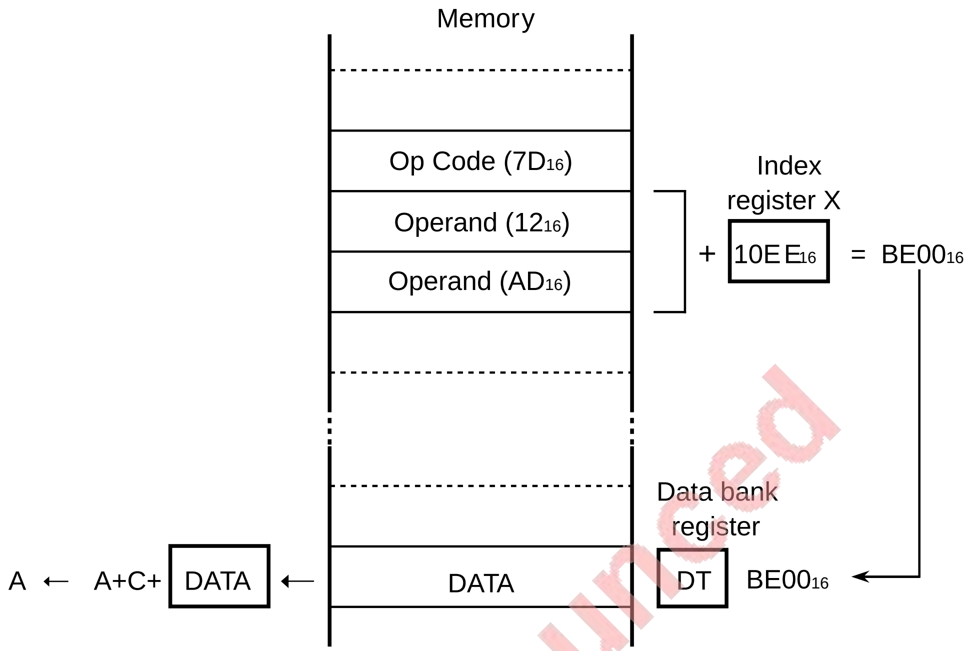
Machine code
$7D_{16}$ 12_{16} AD_{16}



Absolute Indexed X

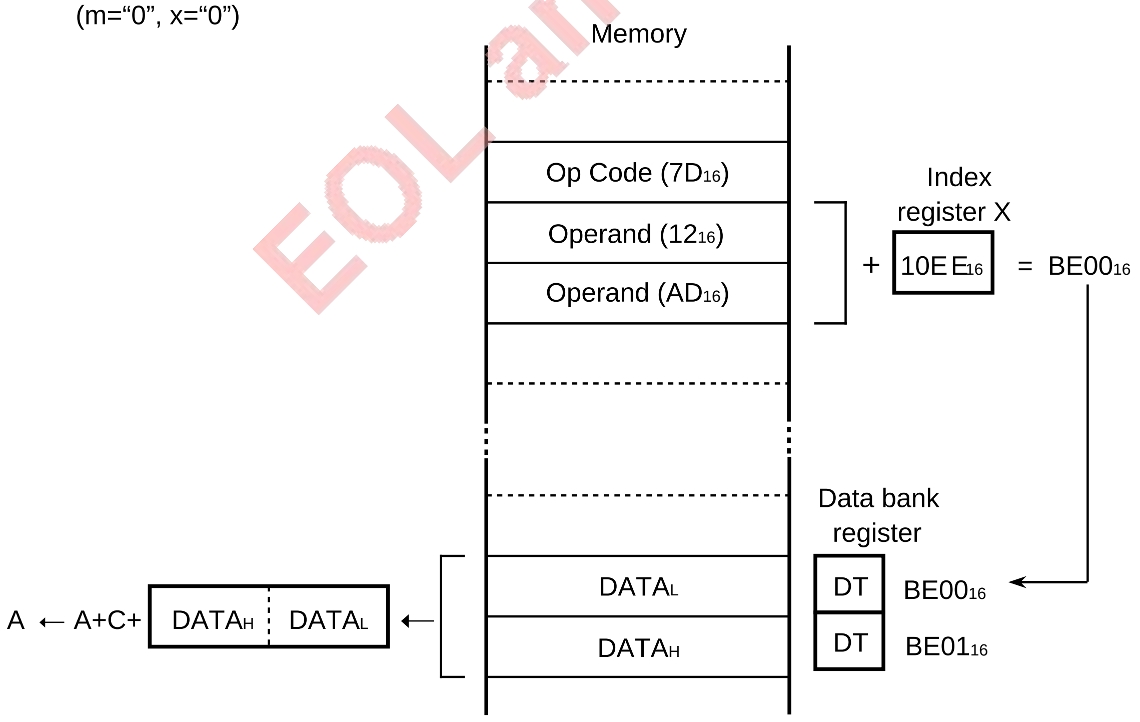
ex. : Mnemonic
ADC A, 0AD12H, X
(m=“1”, x=“0”)

Machine code
$7D_{16}$ 12_{16} AD_{16}



ex. : Mnemonic
ADC A, 0AD12H, X
(m=“0”, x=“0”)

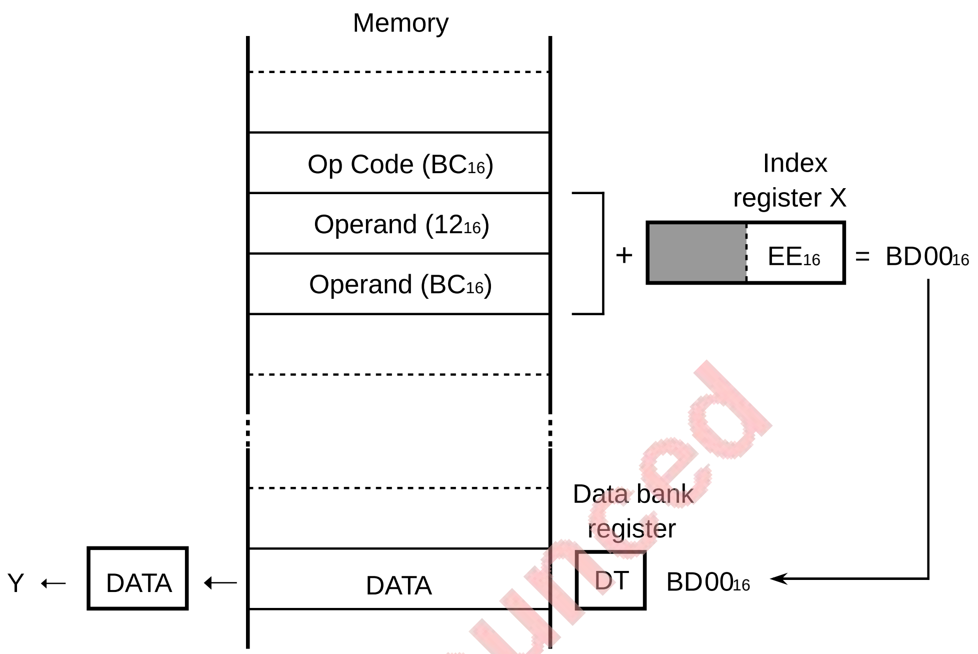
Machine code
$7D_{16}$ 12_{16} AD_{16}



Absolute Indexed X

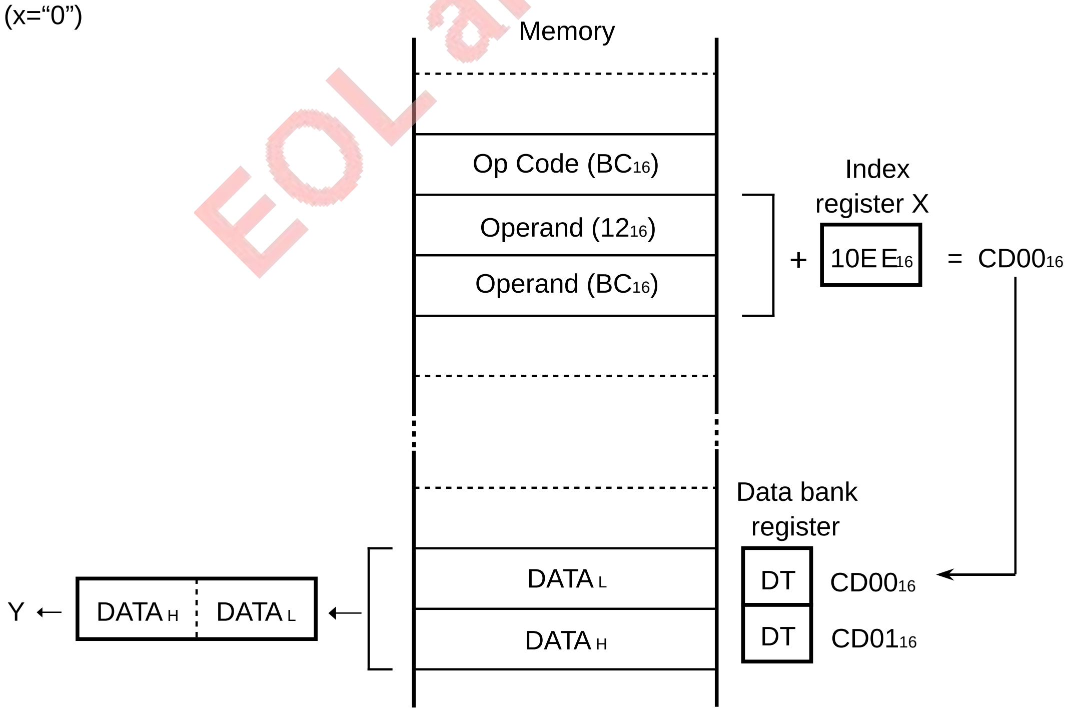
ex. : Mnemonic
LDY 0BC12H, X
(x="1")

Machine code
BC_{16} 12_{16} BC_{16}



ex. : Mnemonic
LDY 0BC12H, X
(x="0")

Machine code
BC_{16} 12_{16} BC_{16}



Absolute Indexed Y

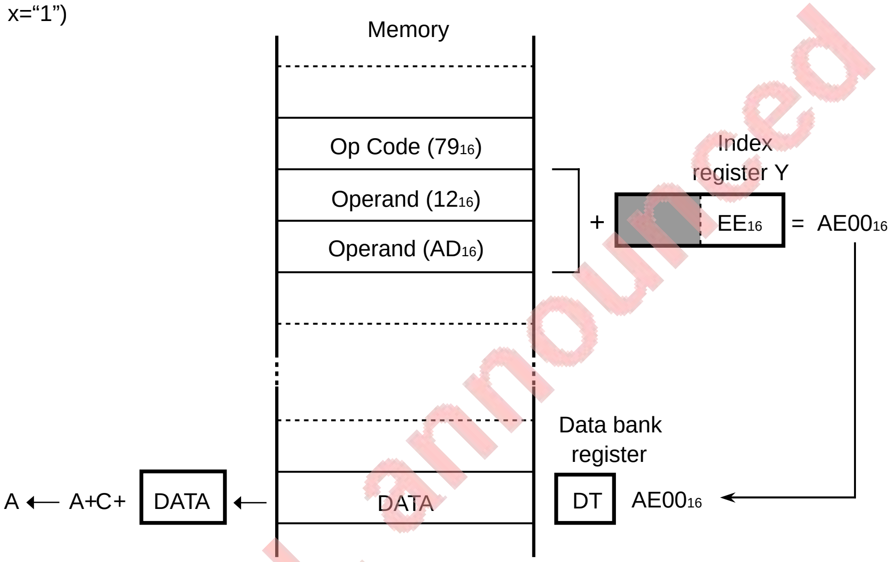
Mode : Absolute indexed Y addressing mode

Function : The following is an actual data: the contents of the memory location specified by the result of adding a 16-bit length numerical value expressed with the instruction's second and third bytes to the index register Y's contents, and the contents of the data bank register. If, however, the addition of the numerical value expressed with the instruction's second and third bytes to the the index register Y's contents generates a carry, the bank which is 1 larger than the contents of the data bank register is used.

Instruction : ADC, AND, CMP, DIV, DIVS, EOR, LDA, LDX, MPY, MPYS, ORA, SBC, STA

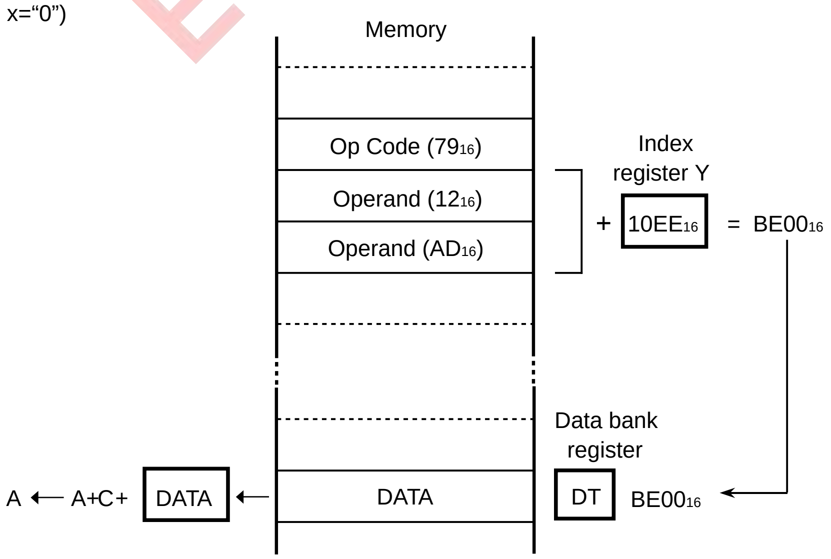
ex. : Mnemonic
ADC A, 0AD12H , Y
(m="1", x="1")

Machine code
79_{16} 12_{16} AD_{16}



ex. : Mnemonic
ADC A, 0AD12H , Y
(m="1", x="0")

Machine code
79_{16} 12_{16} AD_{16}



Absolute Indexed Y

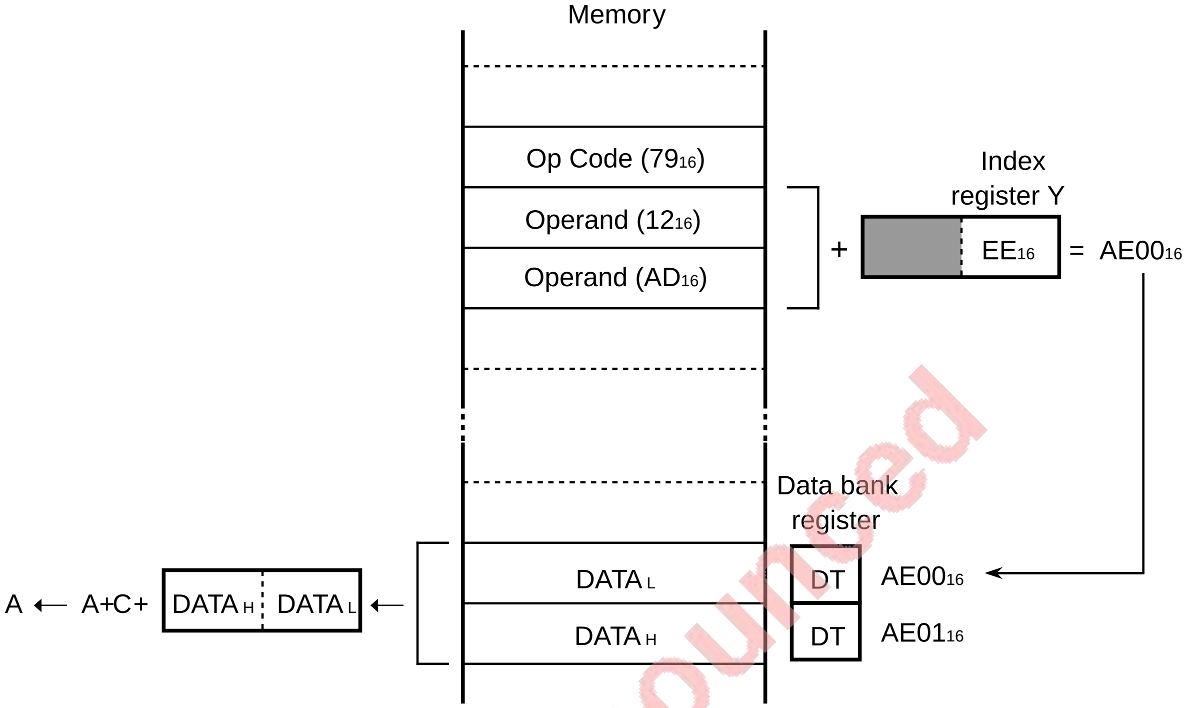
ex. : Mnemonic

ADC A, 0AD12H, Y

(m="0", x="1")

Machine code

79_{16} 12_{16} AD_{16}



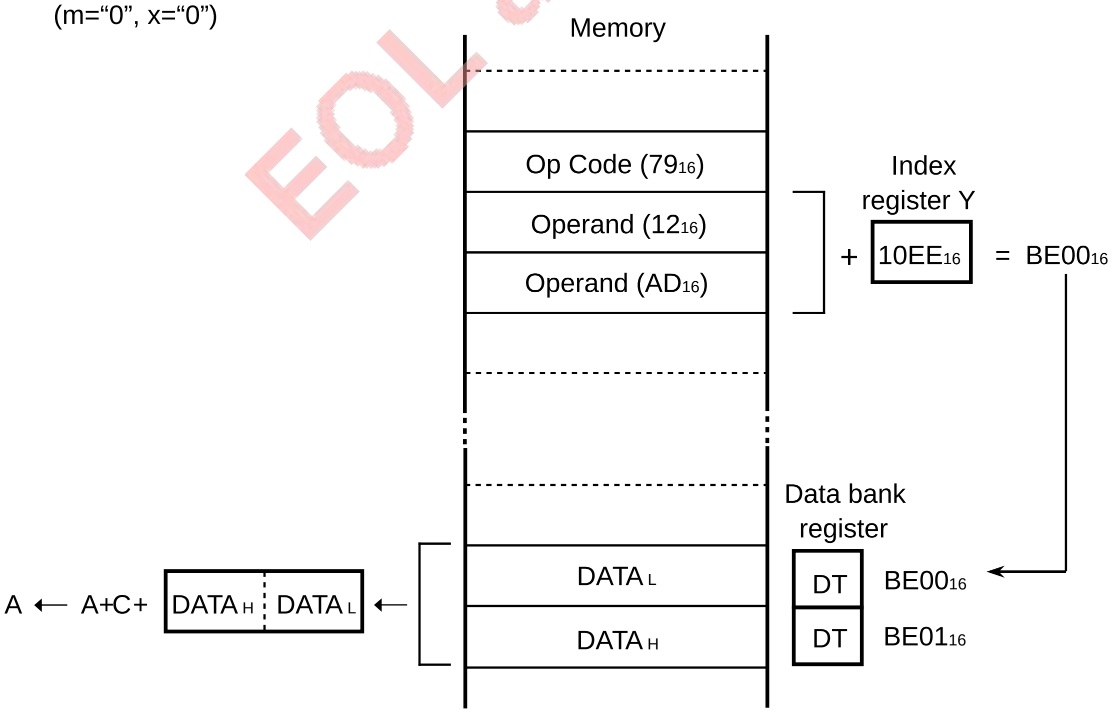
ex. : Mnemonic

ADC A, 0AD12H, Y

(m="0", x="0")

Machine code

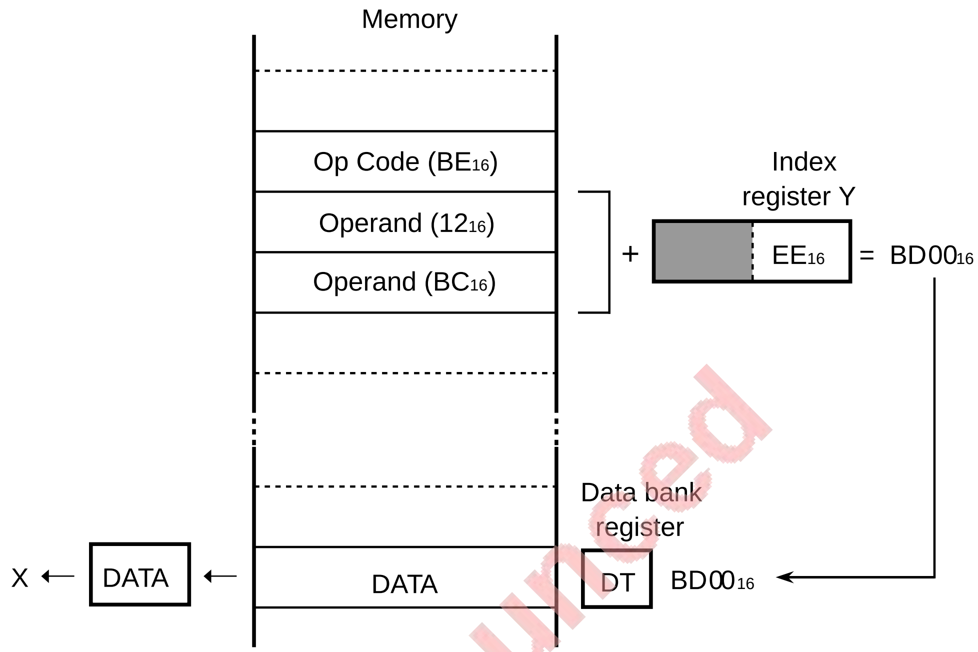
79_{16} 12_{16} AD_{16}



Absolute Indexed Y

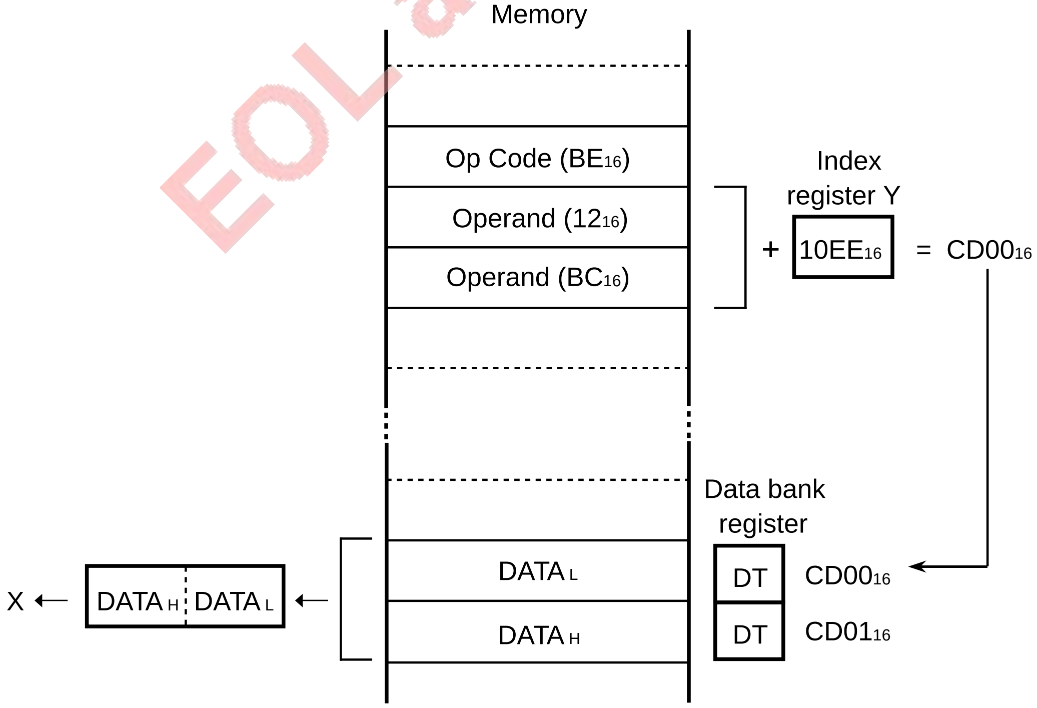
ex. : Mnemonic
LDX 0BC12H, Y
(x="1")

Machine code
BE$_{16}$ 12$_{16}$ BC$_{16}$



ex. : Mnemonic
LDX 0BC12H, Y
(x="0")

Machine code
BE$_{16}$ 12$_{16}$ BC$_{16}$



Absolute Long

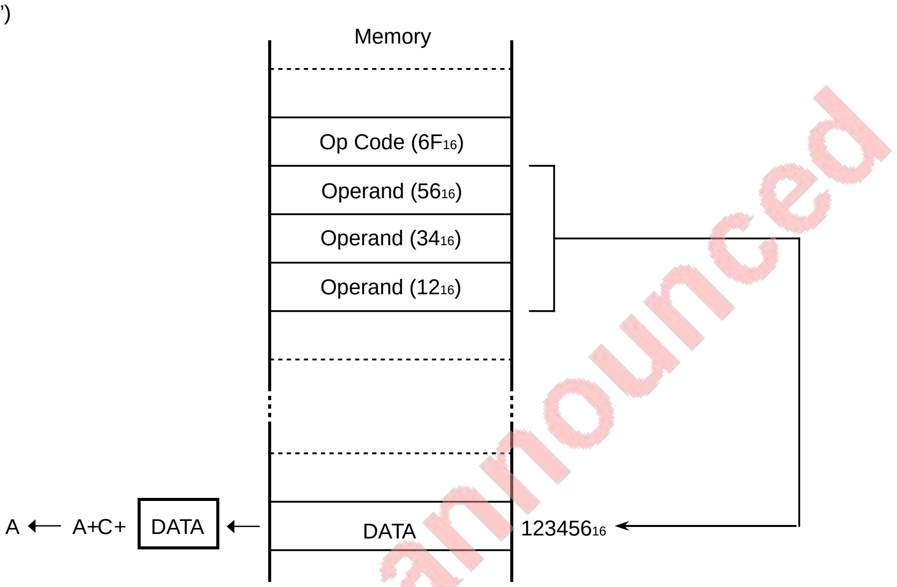
Mode : Absolute long addressing mode

Function : The contents of the memory location specified by the instruction's second, third and fourth bytes are an actual data. Note that, in the cases of the JMP and JSR instructions, the instruction's second and third bytes are transferred to the program counter and the fourth byte is transferred to the program bank register.

Instruction : ADC, AND, CMP, DIV, DIVS, EOR, JMP, JSR, LDA, MPY, MPYS, ORA, SBC, STA

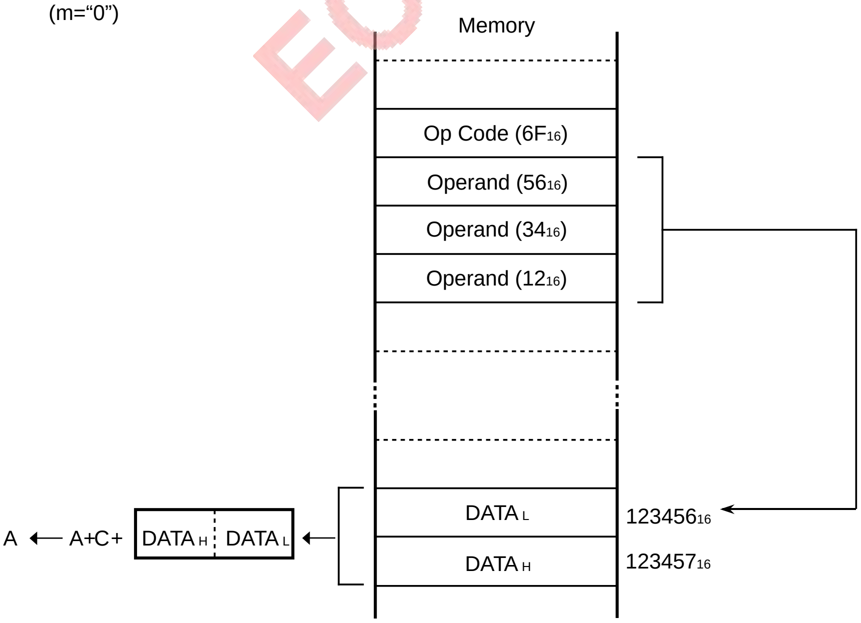
ex. : Mnemonic
ADC A, 123456H
(m="1")

Machine code
$6F_{16}$ 56_{16} 34_{16} 12_{16}



ex. : Mnemonic
ADC A, 123456H
(m="0")

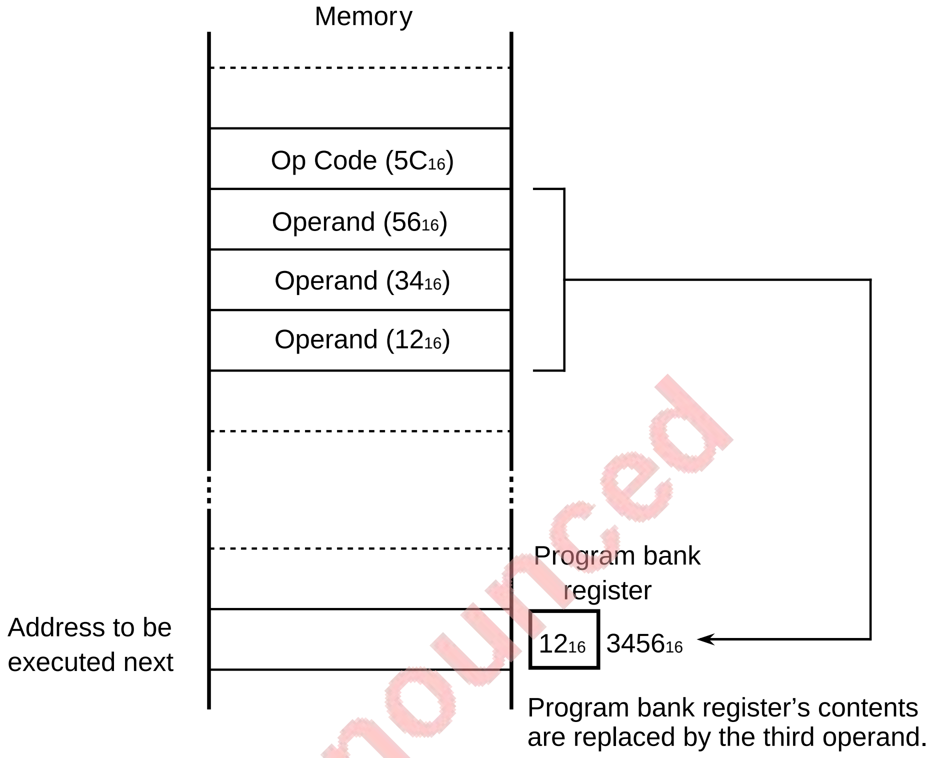
Machine code
$6F_{16}$ 56_{16} 34_{16} 12_{16}



Absolute Long

ex. : Mnemonic
JMPL 123456H

Machine code
$5C_{16}$ 56_{16} 34_{16} 12_{16}

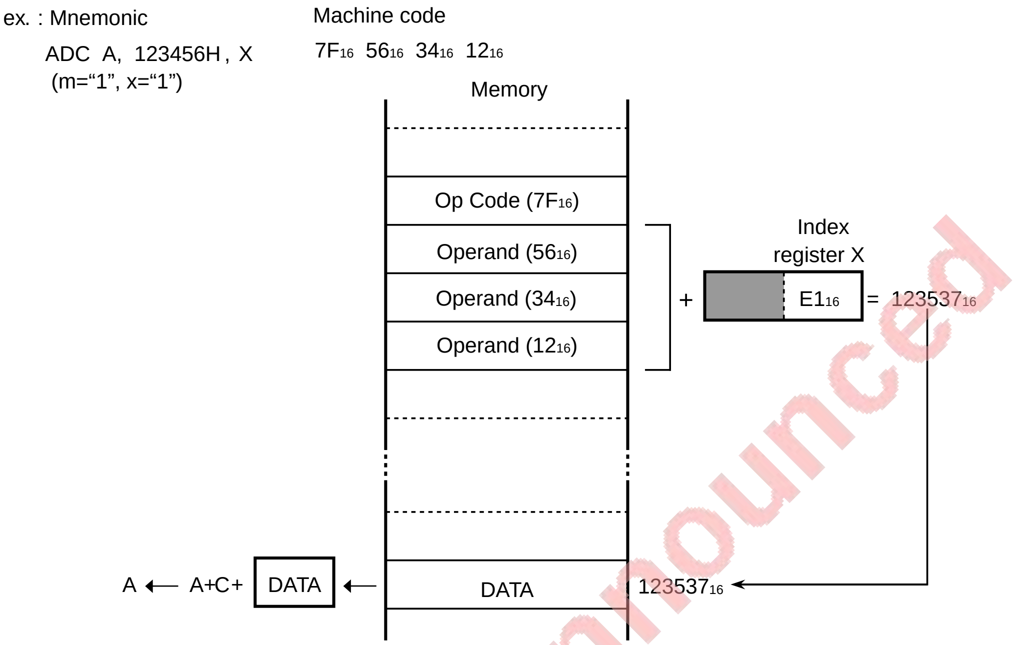


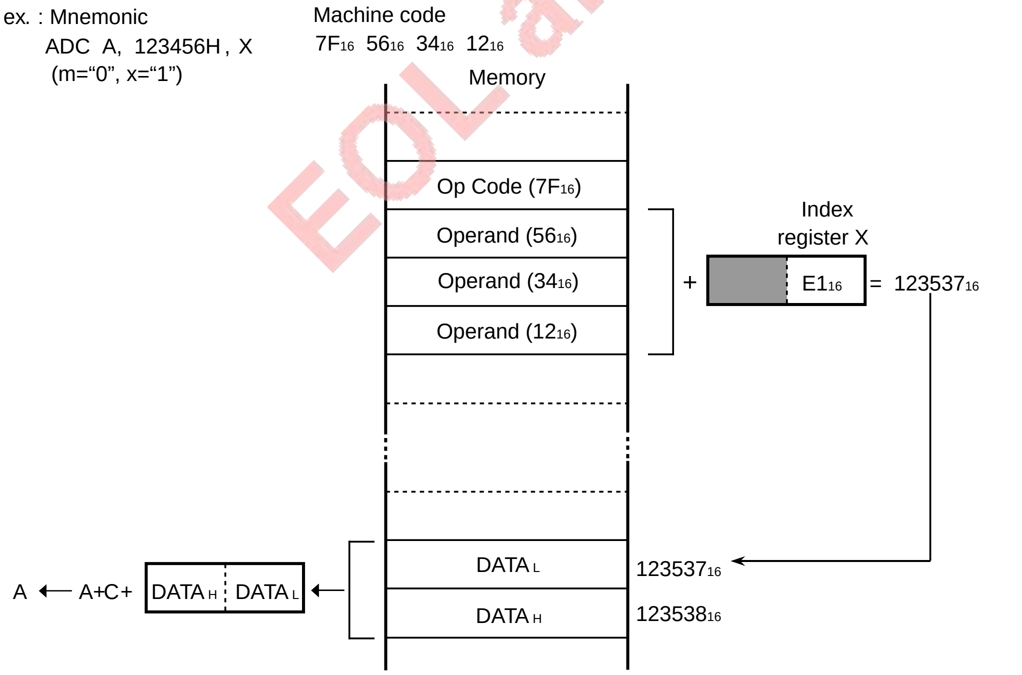
Absolute Long Indexed X

Mode : Absolute long indexed X addressing mode

Function : The following is an actual data: the contents of the memory location specified by the result of adding a numerical value expressed with the instruction’s second, third and forth bytes to the index register X’s contents.

Instruction : ADC, AND, CMP, DIV, DIVS, EOR, LDA, MPY, MPYS, ORA, SBC, STA

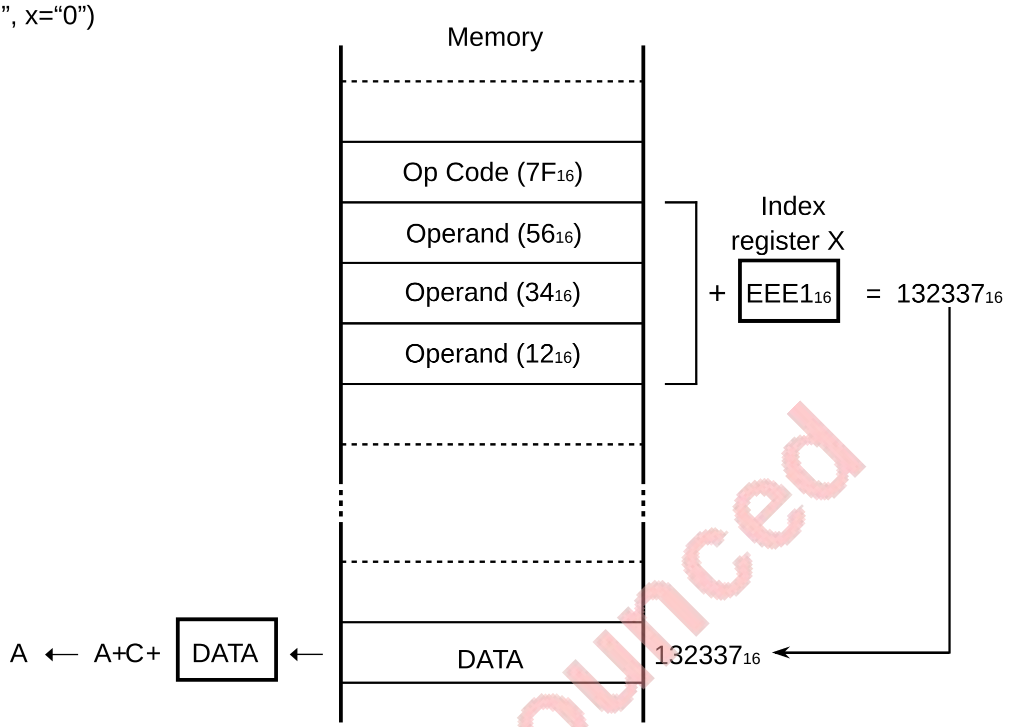




Absolute Long Indexed X

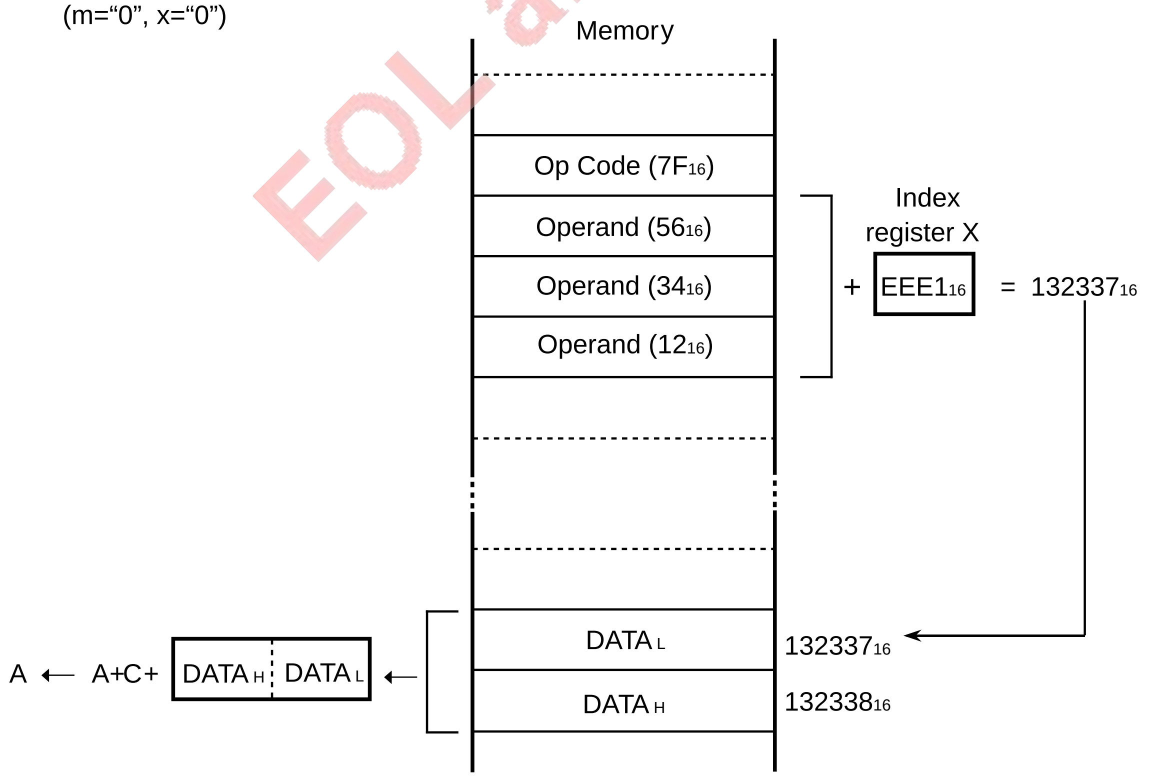
ex. : Mnemonic
ADC A, 123456H, X
(m=“1”, x=“0”)

Machine code
$7F_{16}$ 56_{16} 34_{16} 12_{16}



ex. : Mnemonic
ADC A, 123456H, X
(m=“0”, x=“0”)

Machine code
$7F_{16}$ 56_{16} 34_{16} 12_{16}



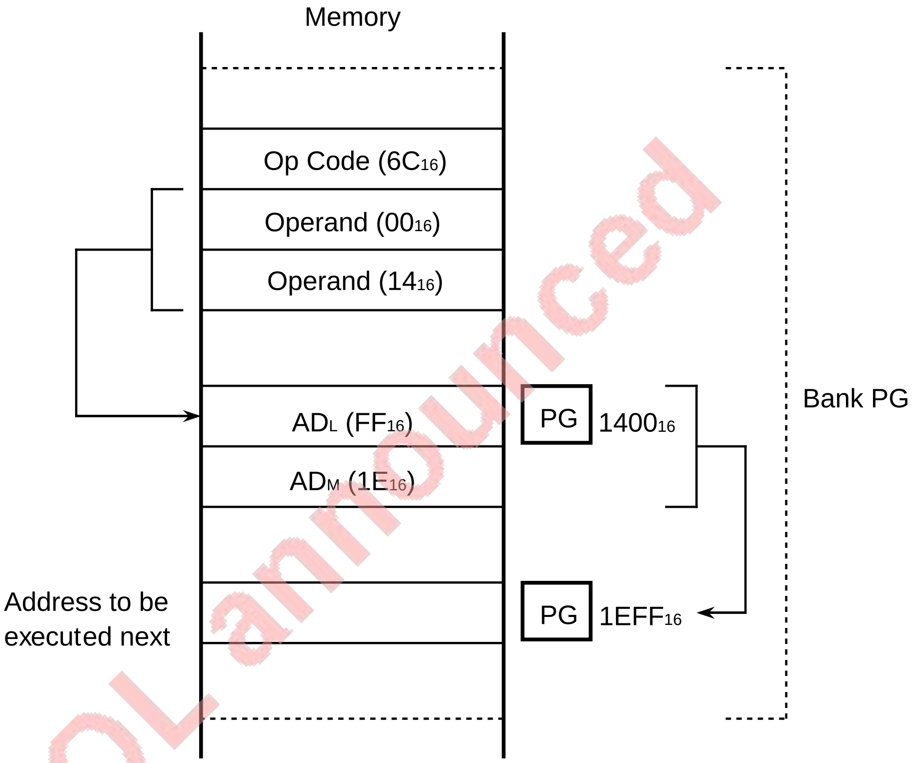
Absolute Indirect

Mode : Absolute indirect addressing mode

Function : A sequence of 2-byte memory is specified by the instruction's second and third bytes. The contents of these bytes specify the address within the same program bank to which a jump is to be made.

Instruction : JMP

ex. : Mnemonic	Machine code
JMP (1400H)	$6C_{16}$ 00_{16} 14_{16}



Note : Note the reference/branch destination bank when a JMP instruction or a reference destination is located near a bank boundary.
⇒Refer to the description of a JMP instruction (Page 4-50).

Absolute Indirect Long

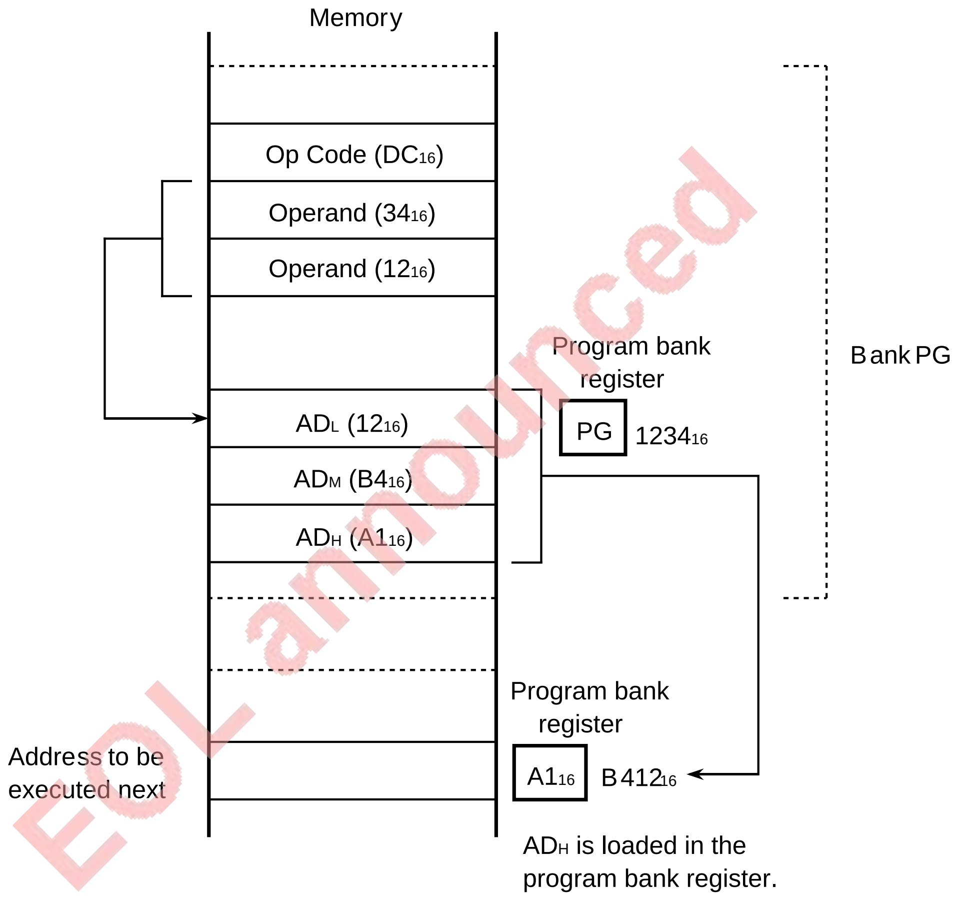
Mode : Absolute indirect long addressing mode

Function : A sequence of 3-byte memory is specified by the instruction's second and third bytes. The contents of these 3 bytes specify the address to which a jump is to be made.

Instruction : JMP

ex. : Mnemonic
JMPL (1234H)

Machine code
DC_{16} 34_{16} 12_{16}



Note : Note the reference destination bank when a JMP instruction is located near a bank boundary.
⇒Refer to the description of a JMP instruction (Page 4-50).

Absolute Indexed X Indirect

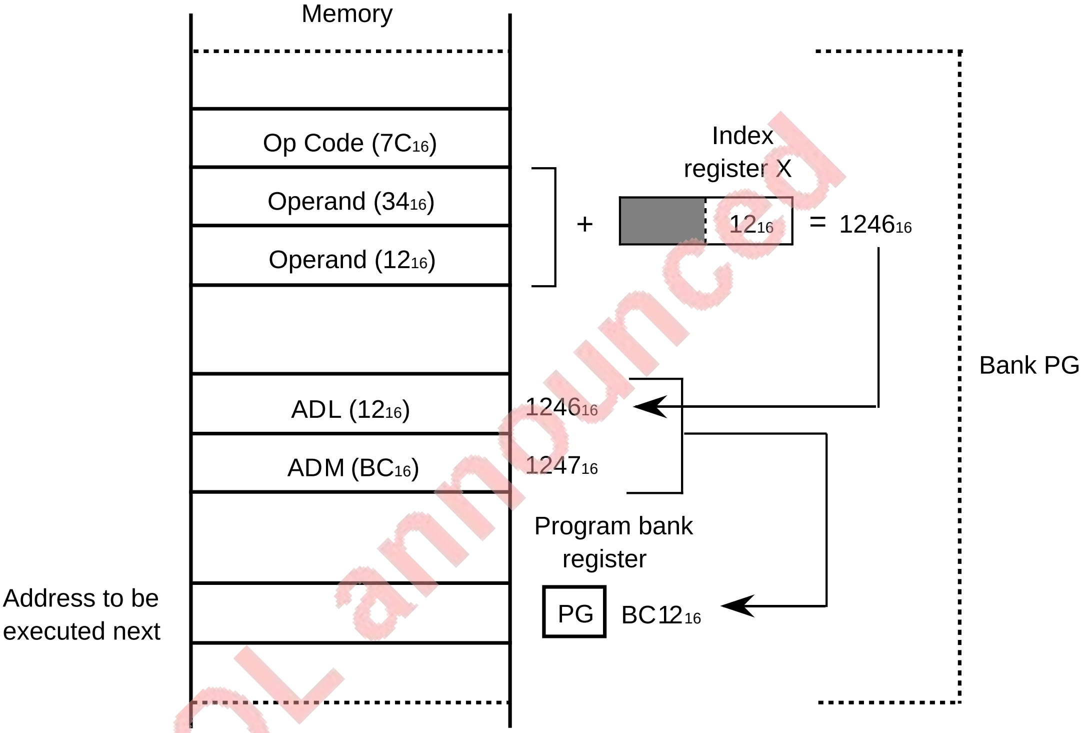
Mode : Absolute indexed X indirect addressing mode

Function : A sequence of 2-byte memory is specified by the result of adding a numerical value expressed with the instruction's second and third bytes to the index register X's contents. The contents of these bytes specify the address to which a jump is to be made.

Instruction : JMP, JSR

ex. : Mnemonic
JMP (1234H , X)
(x="1")

Machine code
$7C_{16}$ 34_{16} 12_{16}



Note : Note the reference/branch destination bank in the case of a JMP or a JSR instruction when the instruction or the branch destination address is located near a boundary.
⇒Refer to the description of a JMP instruction (Page 4-50).
Refer to the description of a JSR instruction (Page 4-51).

Stack

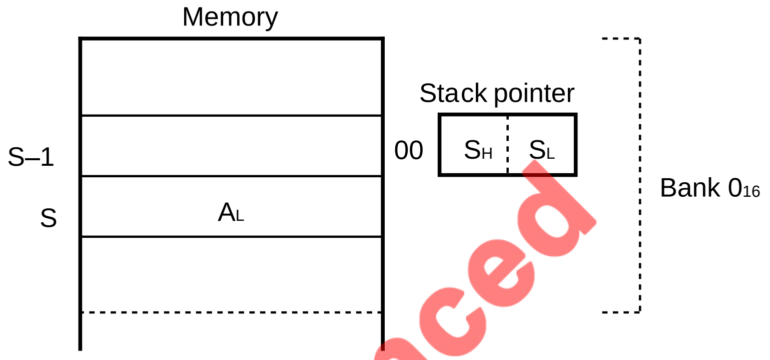
Mode : Stack addressing mode

Function : The contents of a register or others are stored to or restored from the memory location specified by the stack pointer. The stack register is set in bank 0_{16}.

Instruction : PEA, PEI, PER, PHA, PHB, PHD, PHG, PHP, PHT, PHX, PHY, PLA,
PLB, PLD, PLP, PLT, PLX, PLY, PSH, PUL

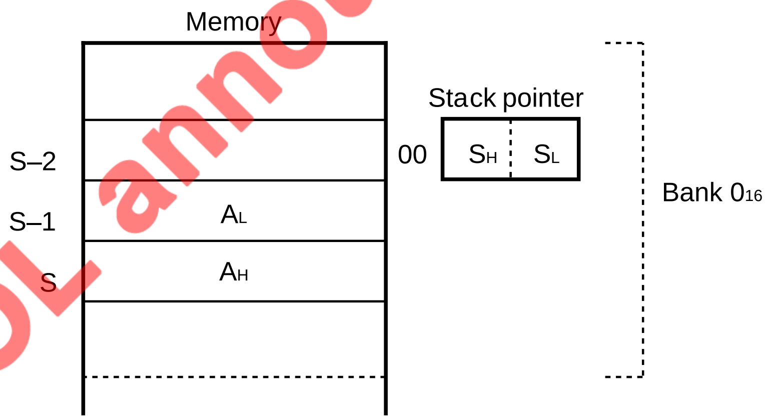
ex. : Mnemonic
PHA
(m="1")

Machine code
48_{16}



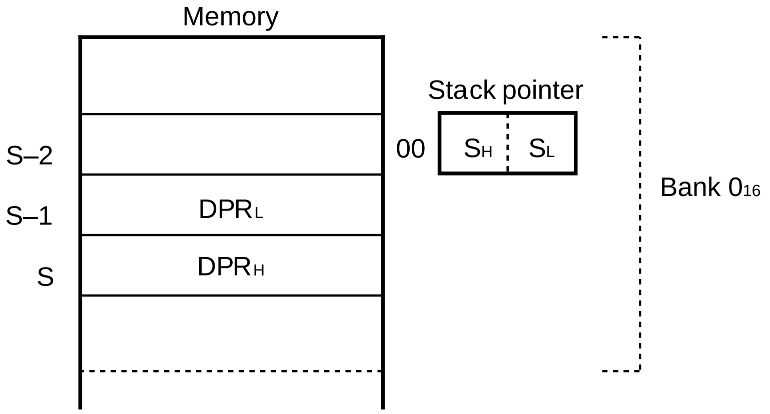
ex. : Mnemonic
PHA
(m="0")

Machine code
48_{16}



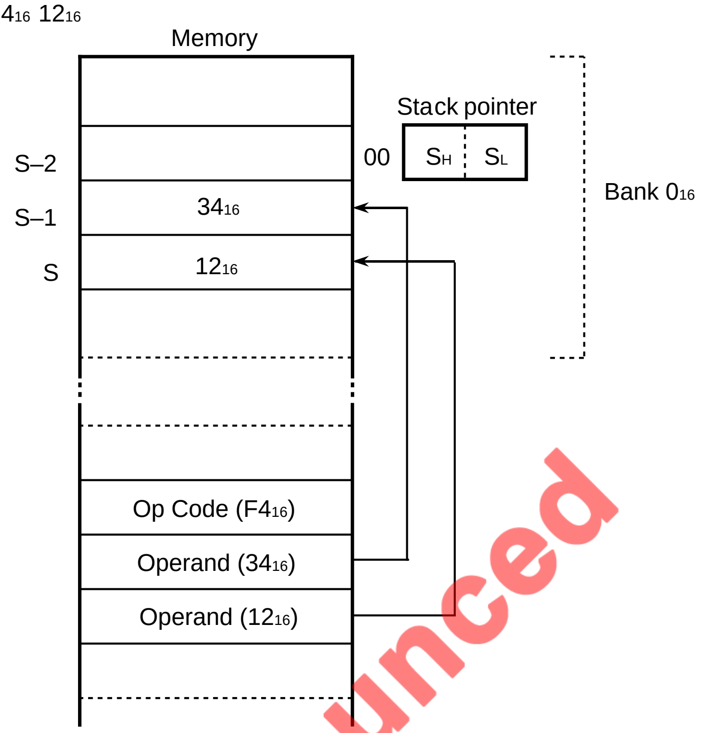
ex. : Mnemonic
PHD

Machine code
$0B_{16}$



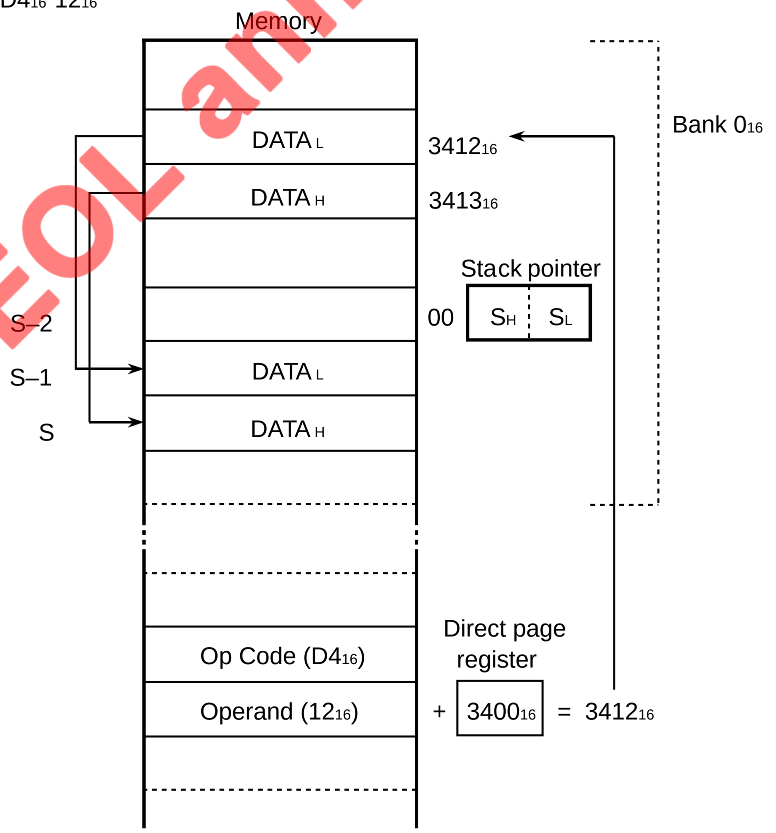
ex. : Mnemonic
PEA #1234H

Machine code
F4$_{16}$ 34$_{16}$ 12$_{16}$



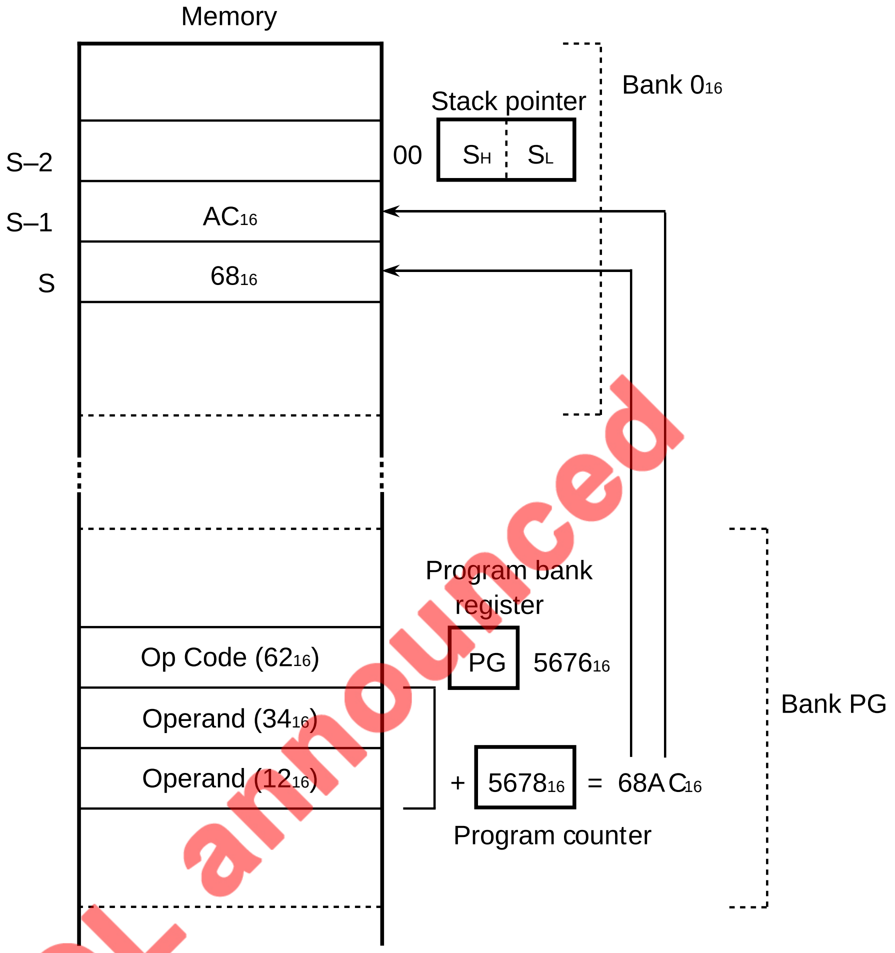
ex. : Mnemonic
PEI #12H

Machine code
D4$_{16}$ 12$_{16}$



ex. : Mnemonic PER #1234H

Machine code 62_{16} 34_{16} 12_{16}



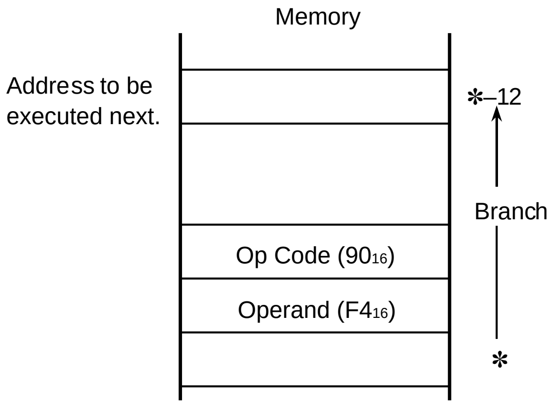
Relative

Mode : Relative addressing mode

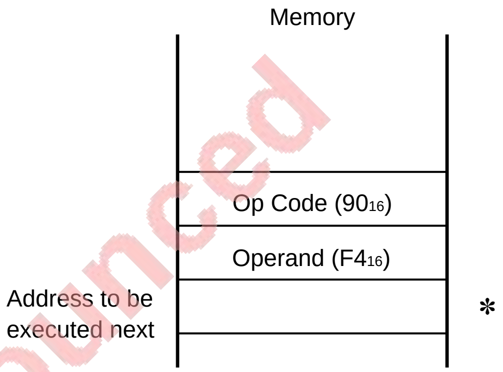
Function : Branches to the address specified by the result of adding the program counter's contents to the instruction's second byte. In the case of a long branch with the BRA instruction, the instruction's second and third bytes are added to the program counter's contents as a 15-bit signed numerical value. If the addition generates a carry or a borrow, 1 is added to or subtracted from the program bank register.

Instruction : BCC, BCS, BEQ, BMI, BNE, BPL, BRA, BVC, BVS

ex. : Mnemonic	Machine code
BCC ✻ –12	90_{16} $F4_{16}$

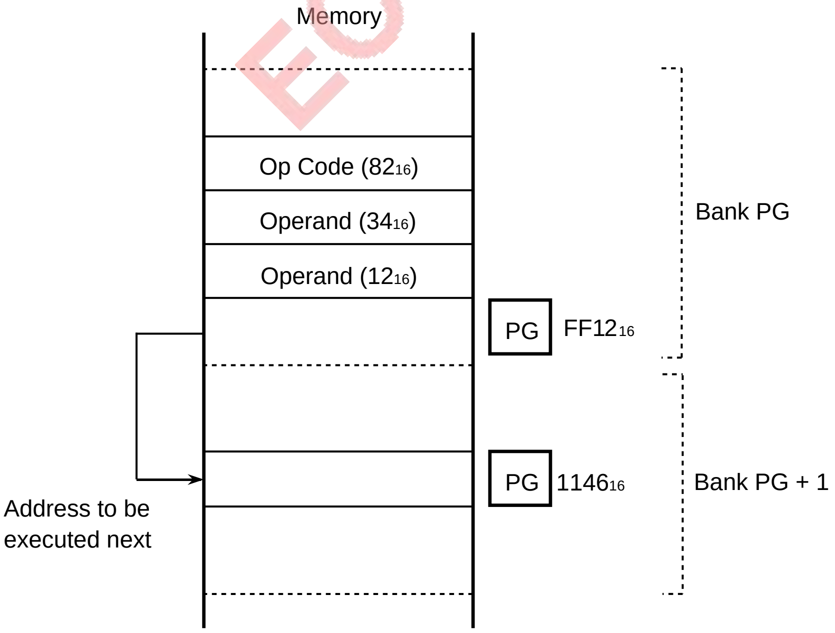


Branches to the address ✻ –12 when the carry flag (C) is "0".



Advances to the address ✻ when the carry flag (C) is "1".

ex. : Mnemonic	Machine code
BRAL 1234H	82_{16} 34_{16} 12_{16}



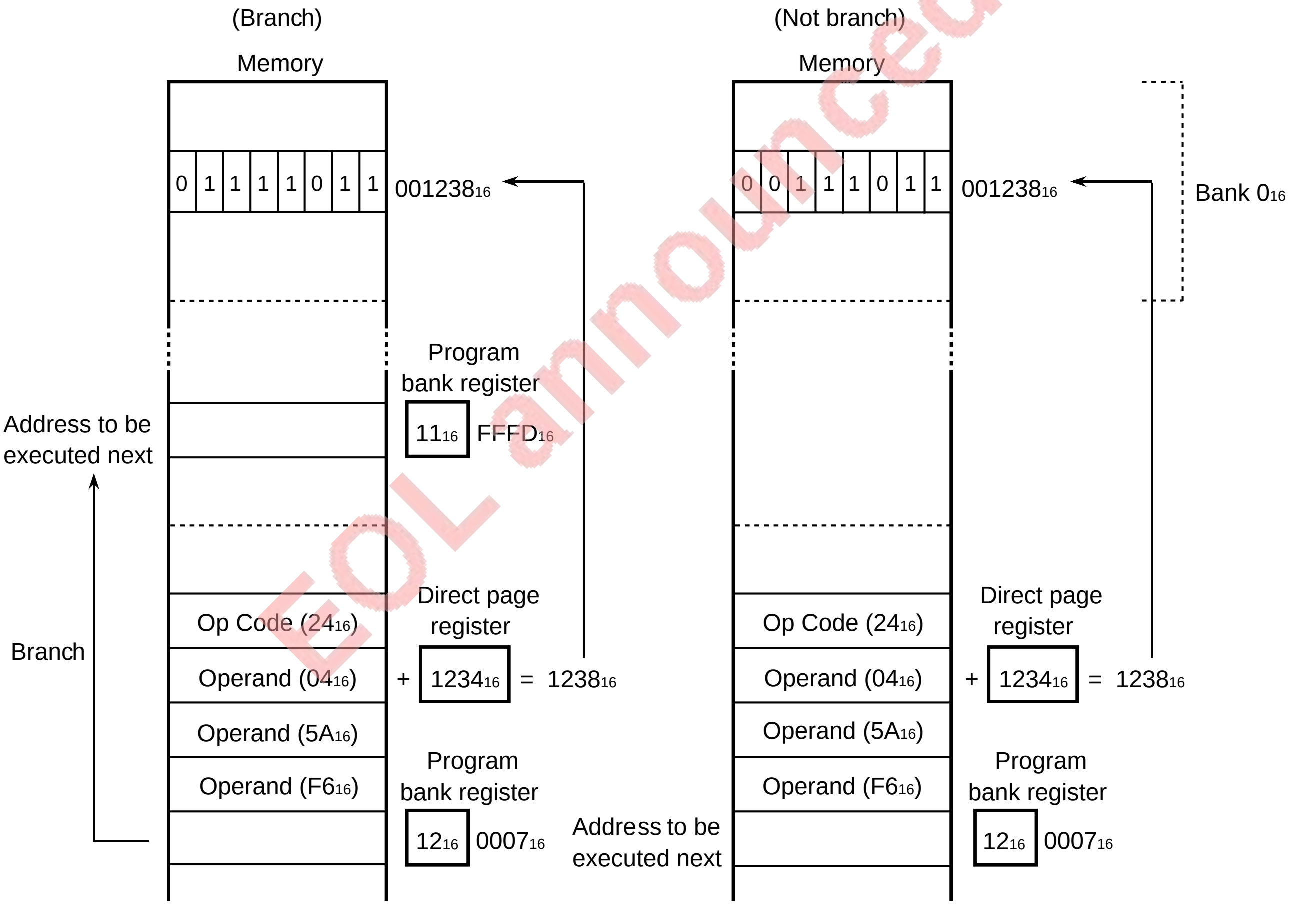
Direct Bit Relative

Mode : Direct bit relative addressing mode

Function : Specifies the memory location in bank 0₁₆ by the result of adding the instruction's second byte to the direct page register's contents; specifies the multiple bit positions in that memory by the bit pattern of the instruction's third and fourth bytes (when the m flag is "1", the third byte only). Then, when the specified bits all satisfy the branching conditions, branches to the address specified by the result of adding the instruction's fifth byte (or when the m flag is "1", the forth byte) as a signed numerical value to the program counter's contents. When, however, the result of adding the instruction's second byte to the direct page register's contents exceeds bank 0₁₆ range, the memory location in bank 1₁₆ is specified.

Instruction : BBC, BBS

ex. : Mnemonic	Machine code
BBS #5AH, 04H, 0F6H (m="1")	24_{16} 04_{16} $5A_{16}$ $F6_{16}$

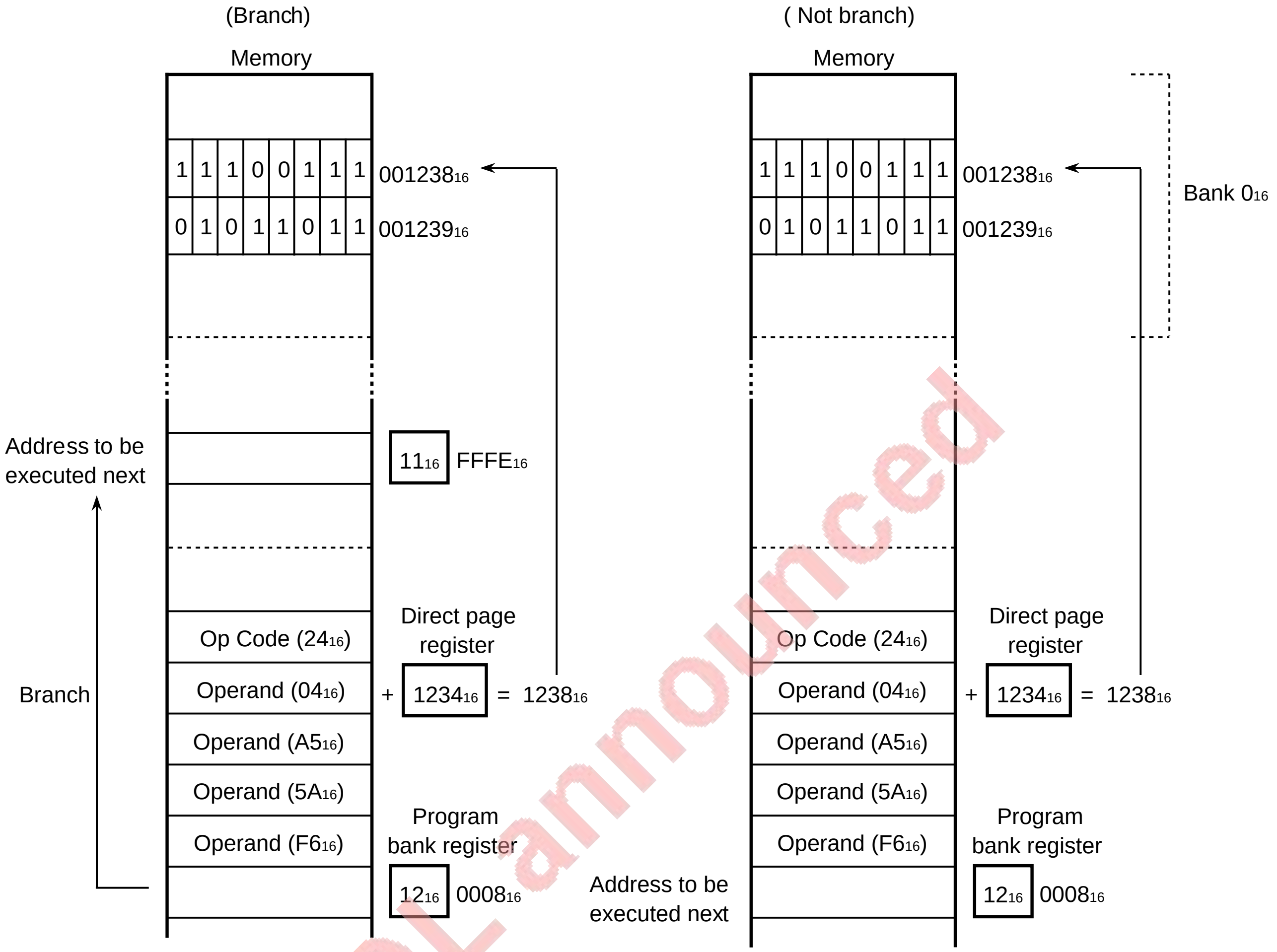


ex. : Mnemonic

BBS #5AA5H, 04H, 0F6H
(m="0")

Machine code

24_{16} 04_{16} $A5_{16}$ $5A_{16}$ $F6_{16}$



Absolute Bit Relative

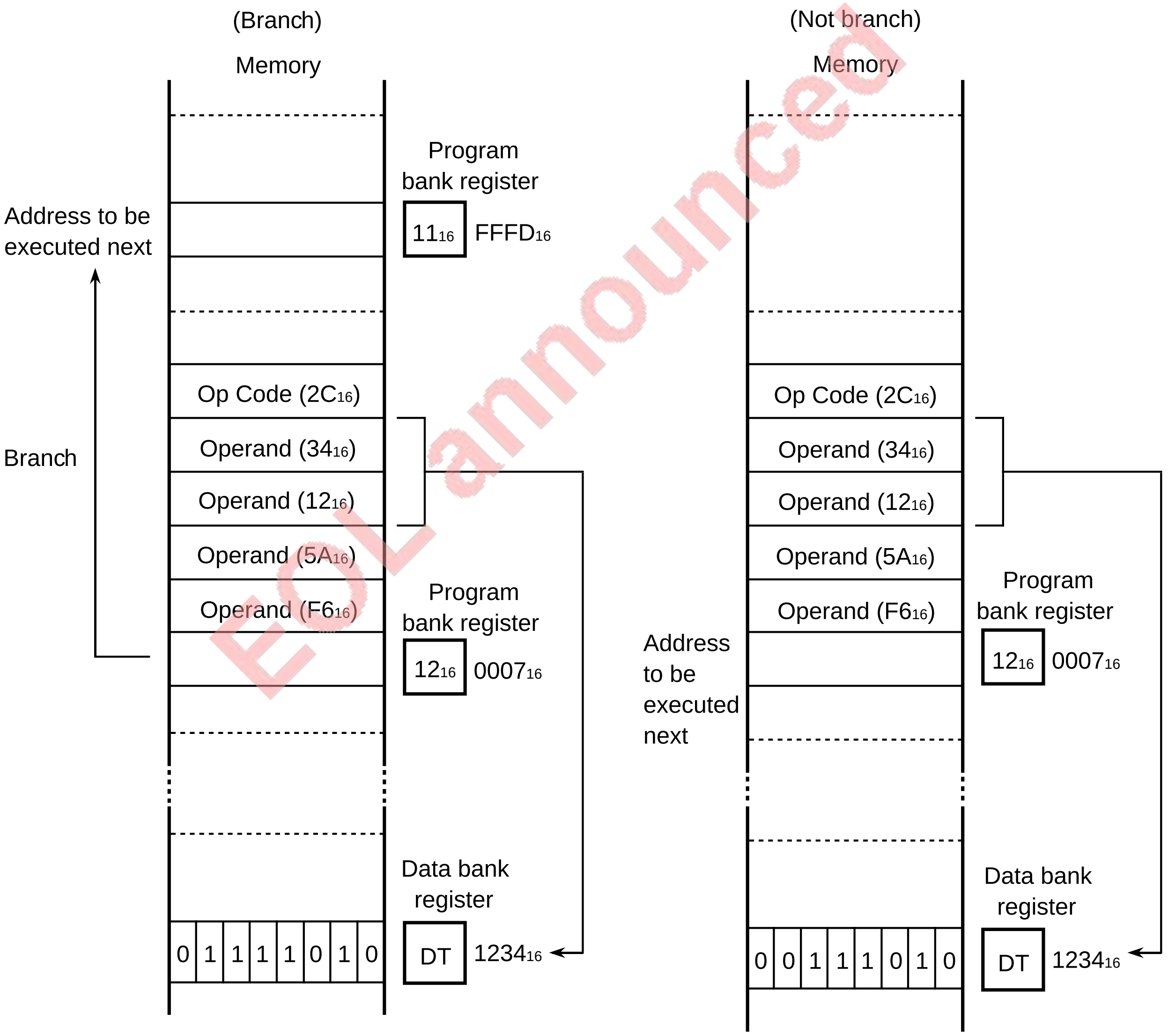
Mode : Absolute bit relative addressing mode

Function : Specifies the memory location by the instruction's second and third bytes and the contents of the data bank register; specifies the multiple bit positions in that memory by a bit pattern of the instruction's fourth and fifth bytes (when the m flag is "1", the fourth byte only). Then, when the specified bits all satisfy the branching conditions, branches to the address specified by the result of adding the instruction's sixth byte (or when the m flag is "1", the fifth byte) as a signed numerical value to the program counter's contents.

Instruction : BBC, BBS

ex. : Mnemonic BBS #5AH, 1234H, 0F6H (m="1")

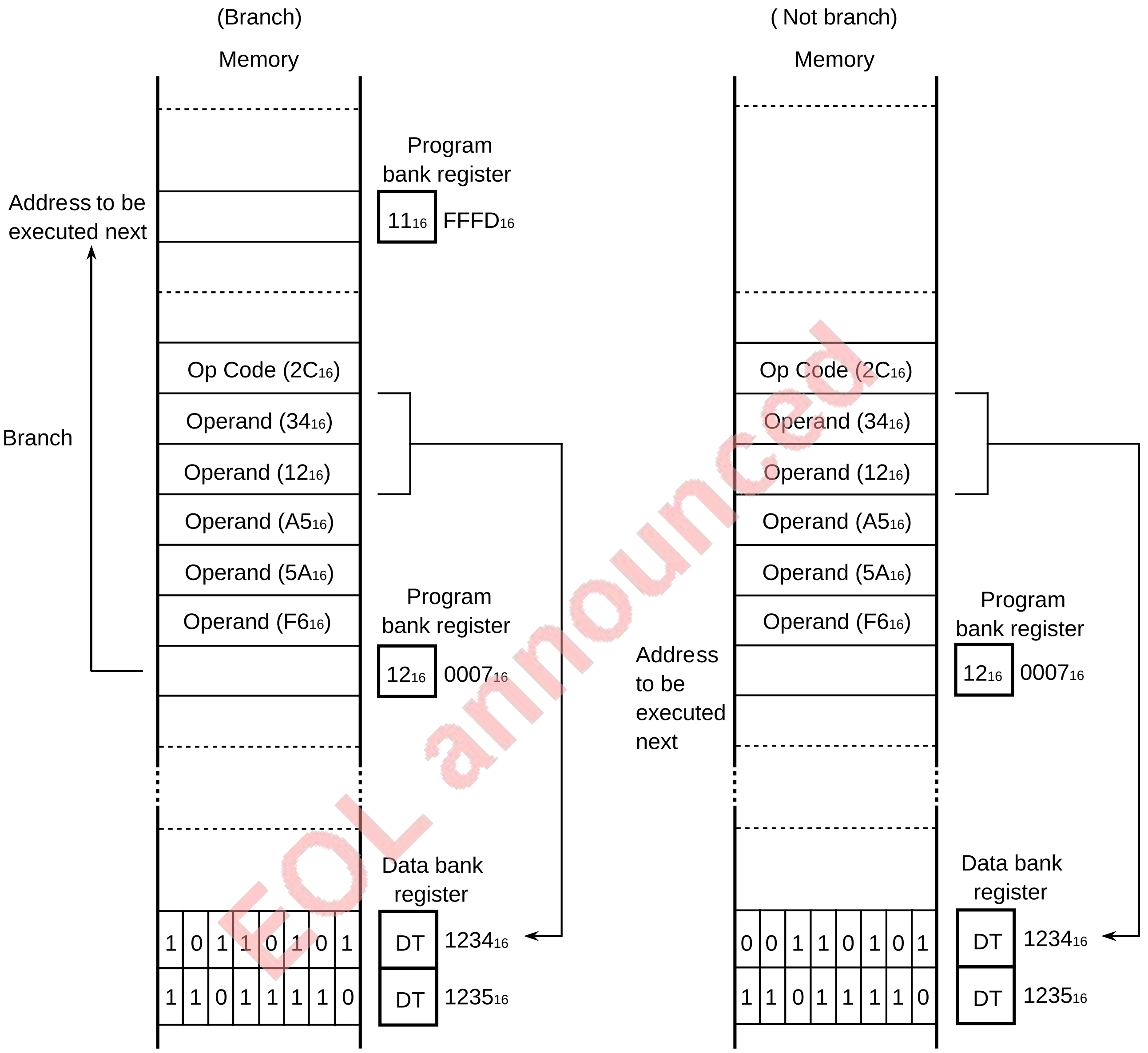
Machine code $2C_{16}$ 34_{16} 12_{16} $5A_{16}$ $F6_{16}$



Absolute Bit Relative

ex. : Mnemonic
BBS #5AA5H, 1234H, 0F6H
(m="0")

Machine code
$2C_{16}$ 34_{16} 12_{16} $A5_{16}$ $5A_{16}$ $F6_{16}$



Stack Pointer Relative

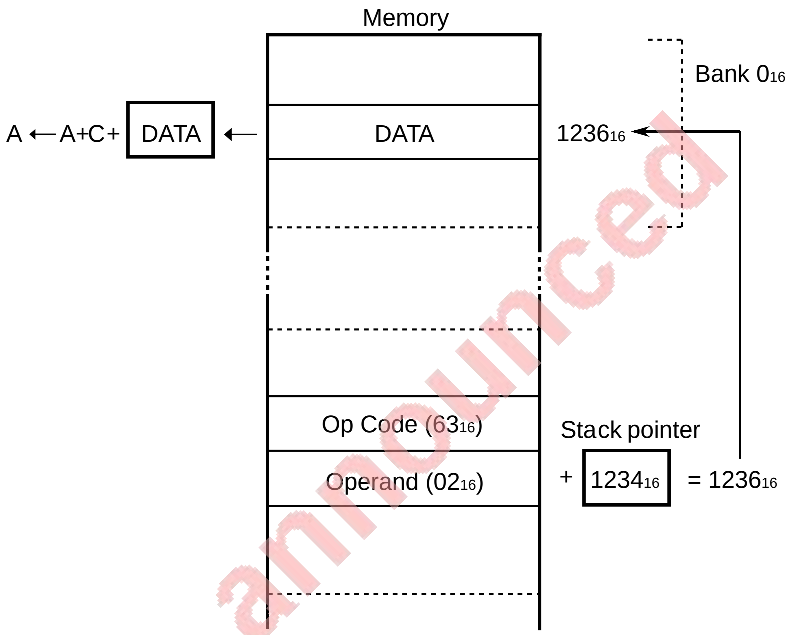
Mode : Stack pointer relative addressing mode

Function : The contents of the memory location in bank 0_{16} are an actual data. This memory is specified by the result of adding the instruction's second byte to the stack pointer's contents. When, however, the result of adding the instruction's second byte to the stack pointer's contents exceeds bank 0_{16} range, the memory location in bank 1_{16} is specified.

Instruction : ADC, AND, CMP, DIV, DIVS, EOR, LDA, MPY, MPYS, ORA, SBC, STA

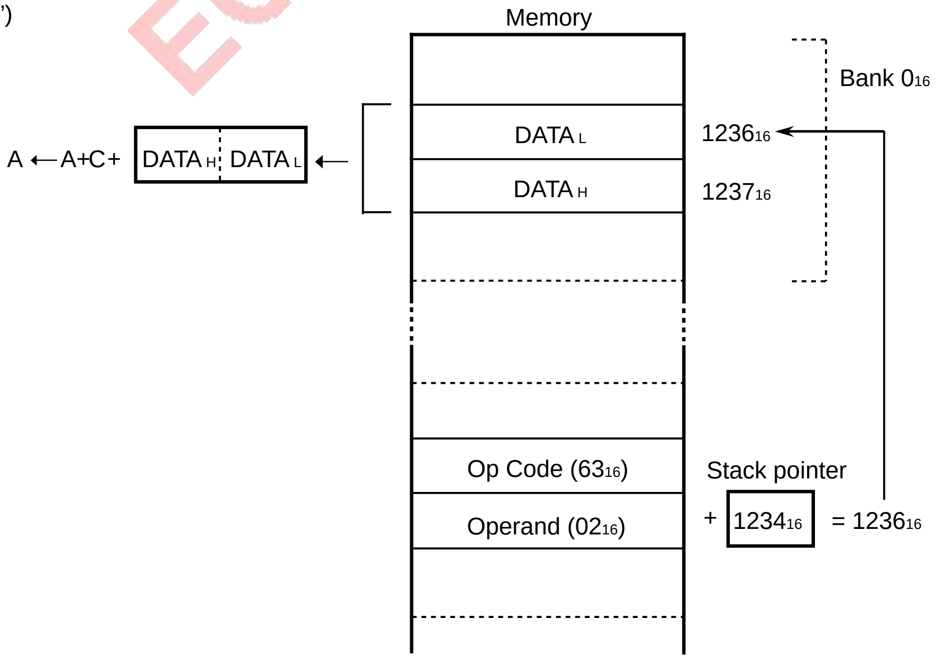
ex. : Mnemonic ADC A, 02H, S (m="1")

Machine code 63_{16} 02_{16}



ex. : Mnemonic ADC A, 02H , S (m="0")

Machine code 63_{16} 02_{16}



Stack Pointer Relative Indirect Indexed Y

Mode : Stack pointer relative indirect indexed Y addressing mode

Function : Specifies a sequence of 2-byte memory by the result of adding the instruction's second byte to the stack pointer's contents. The following is an actual data: the contents of the memory location specified by the result of adding the contents of these bytes to the index register Y's contents, and the data bank register's contents. If, however, the result of adding the contents of that sequence of 2-byte memory to the index register Y's contents generates a carry, the bank which is 1 larger than the contents of the data bank register is used.

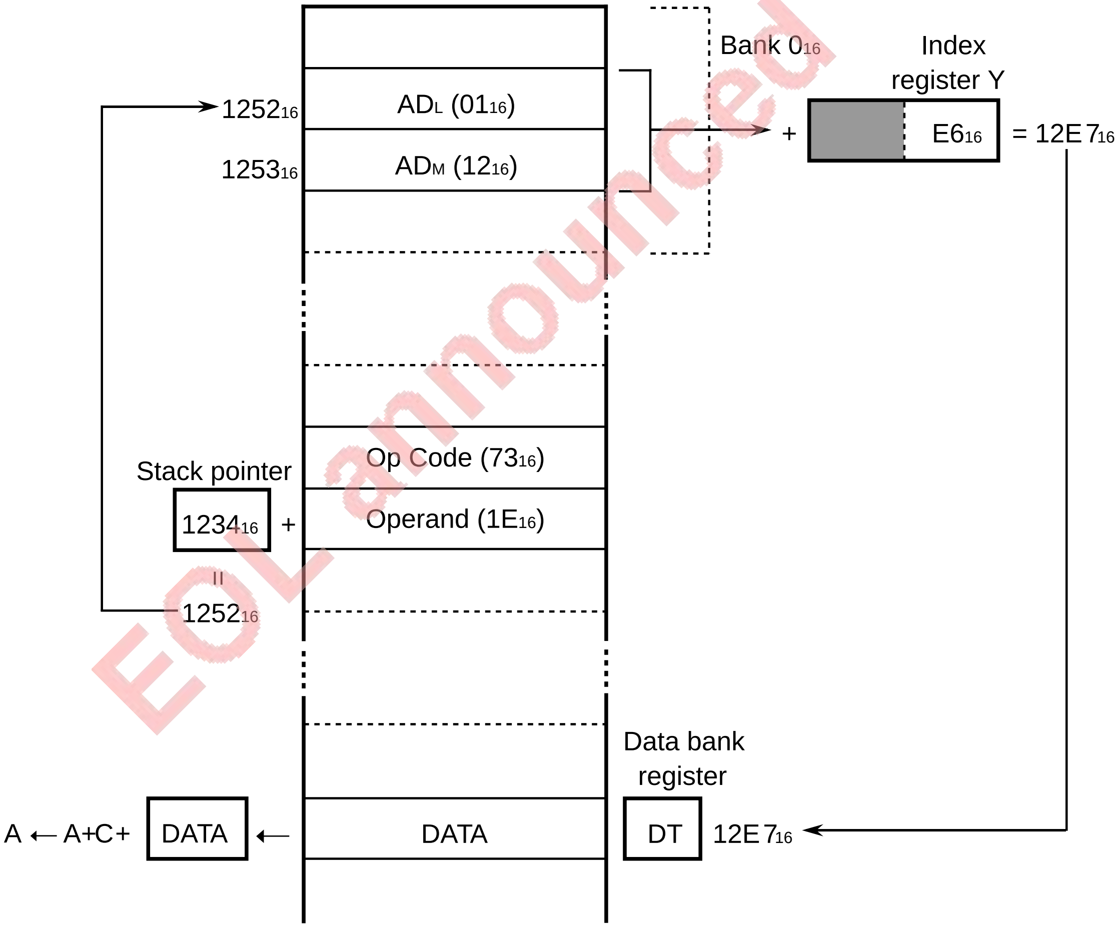
Instruction : ADC, AND, CMP, DIV, DIVS, EOR, LDA, MPY, MPYS, ORA, SBC, STA

ex. : Mnemonic

ADC A, (1EH, S), Y
(m="1", x="1")

Machine code

73_{16} $1E_{16}$

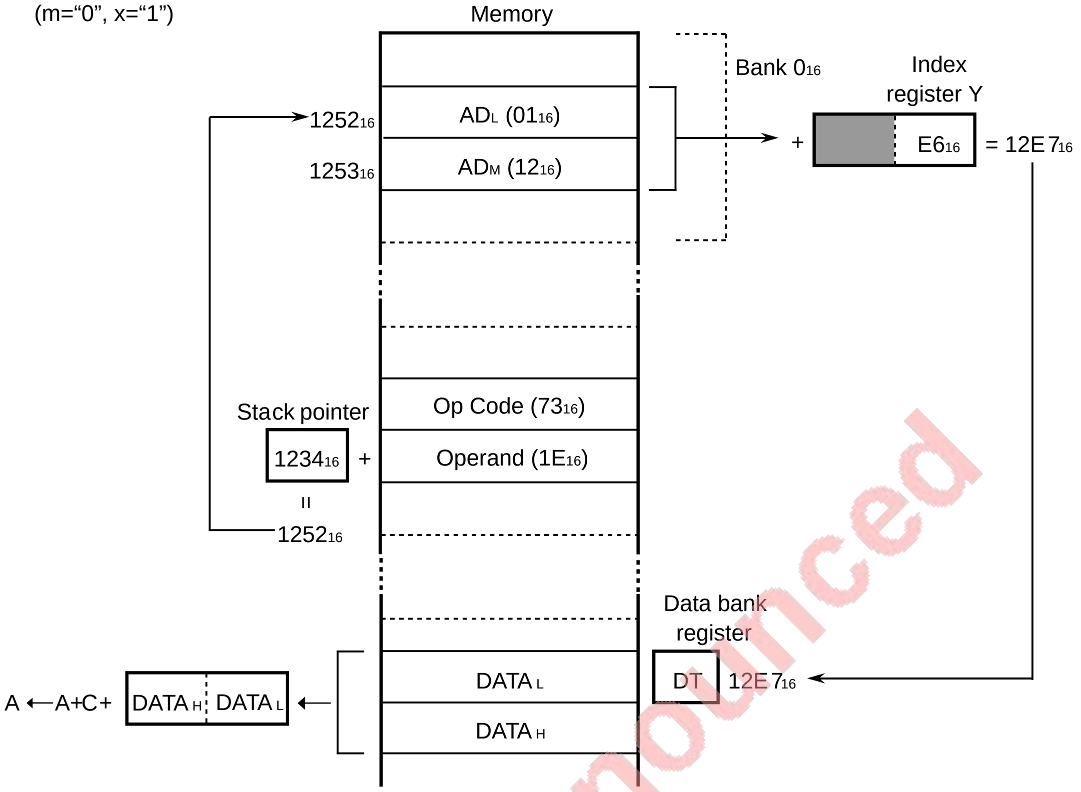


Stack Pointer Relative Indirect Indexed Y

ex. : Mnemonic

ADC A, (1EH, S), Y
(m="0", x="1")

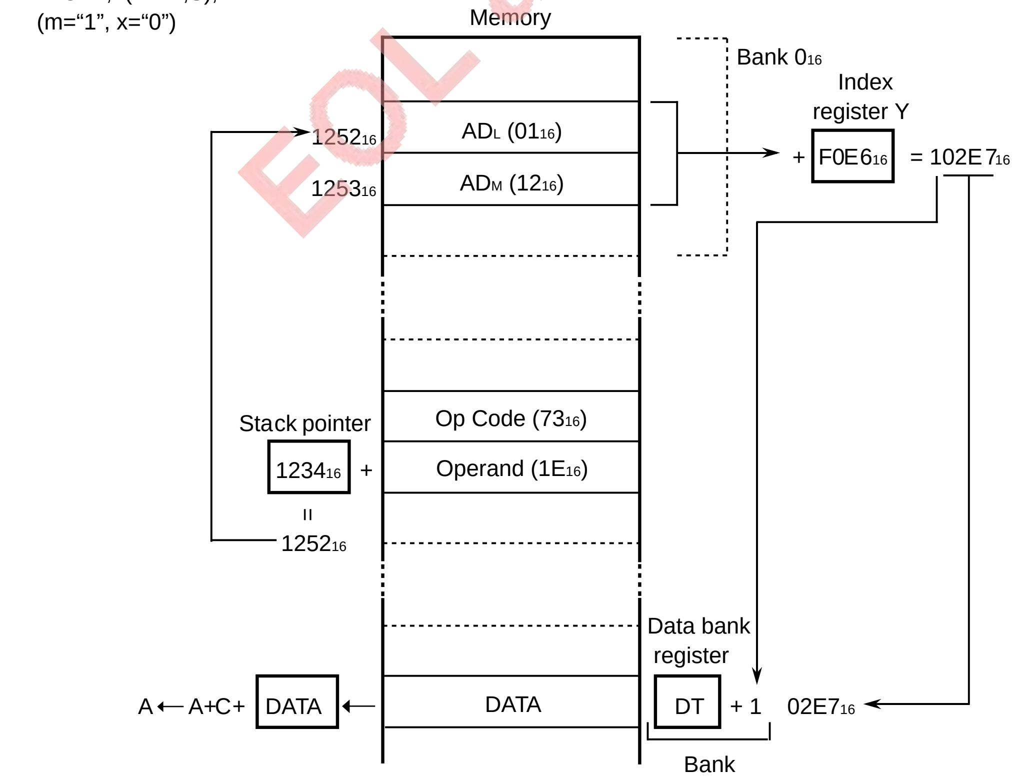
Machine code
73_{16} $1E_{16}$

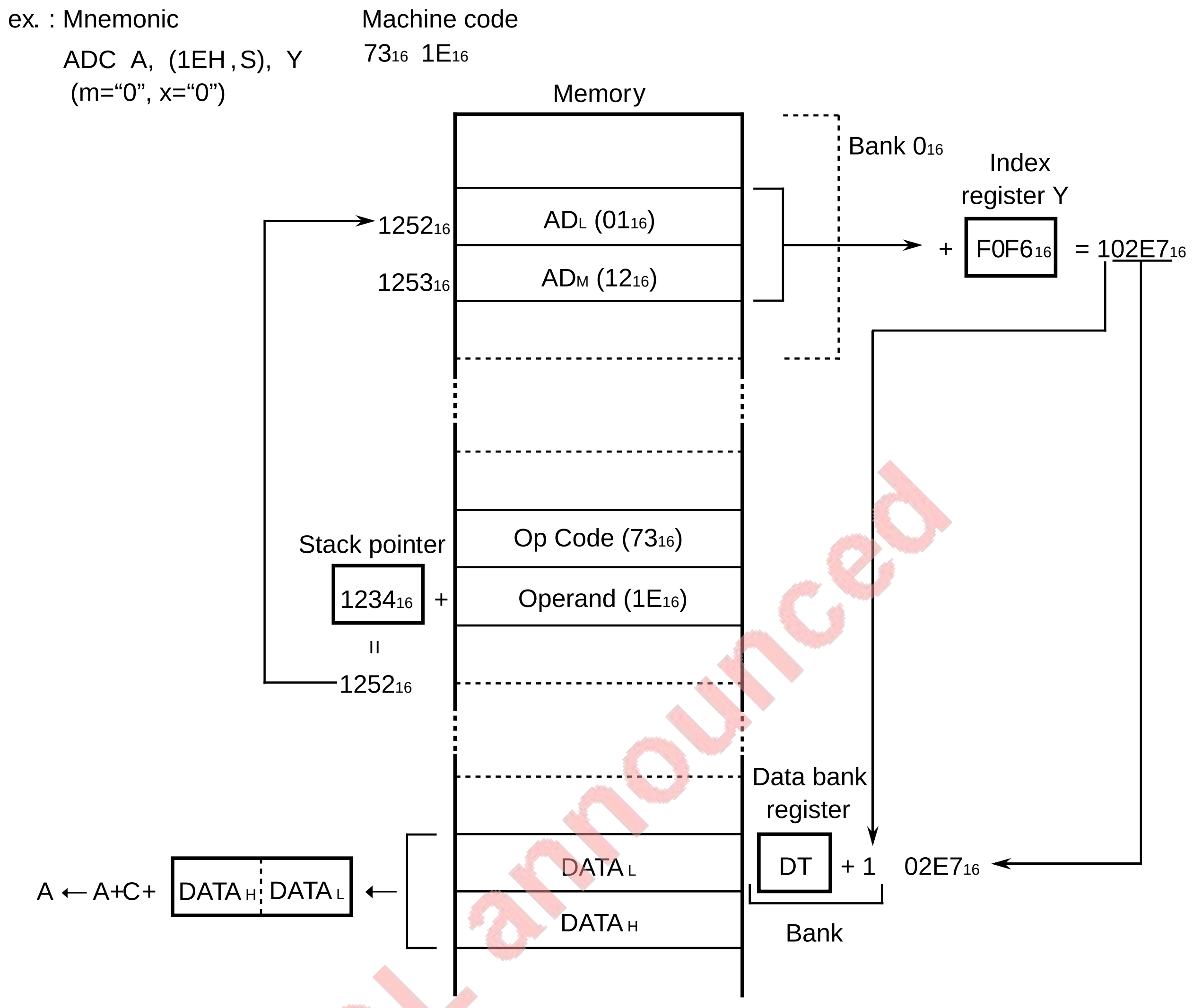


ex. : Mnemonic

ADC A, (1EH,S), Y
(m="1", x="0")

Machine code
73_{16} $1E_{16}$




ex. : Mnemonic
ADC A, (1EH,S), Y
(m="0", x="0")
Machine code
73₁₆ 1E₁₆
Memory
Bank 0₁₆
Index register Y
1252₁₆
AD_L (01₁₆)
1253₁₆
AD_M (12₁₆)
+ F0F6₁₆ = 102E7₁₆
Stack pointer
Op Code (73₁₆)
1234₁₆ +
Operand (1E₁₆)
=
1252₁₆
Data bank register
DT + 1 02E7₁₆
Bank
DATA_L
DATA_H
A ← A+C+ DATA_H DATA_L ←

Block Transfer

Mode : Block transfer addressing mode

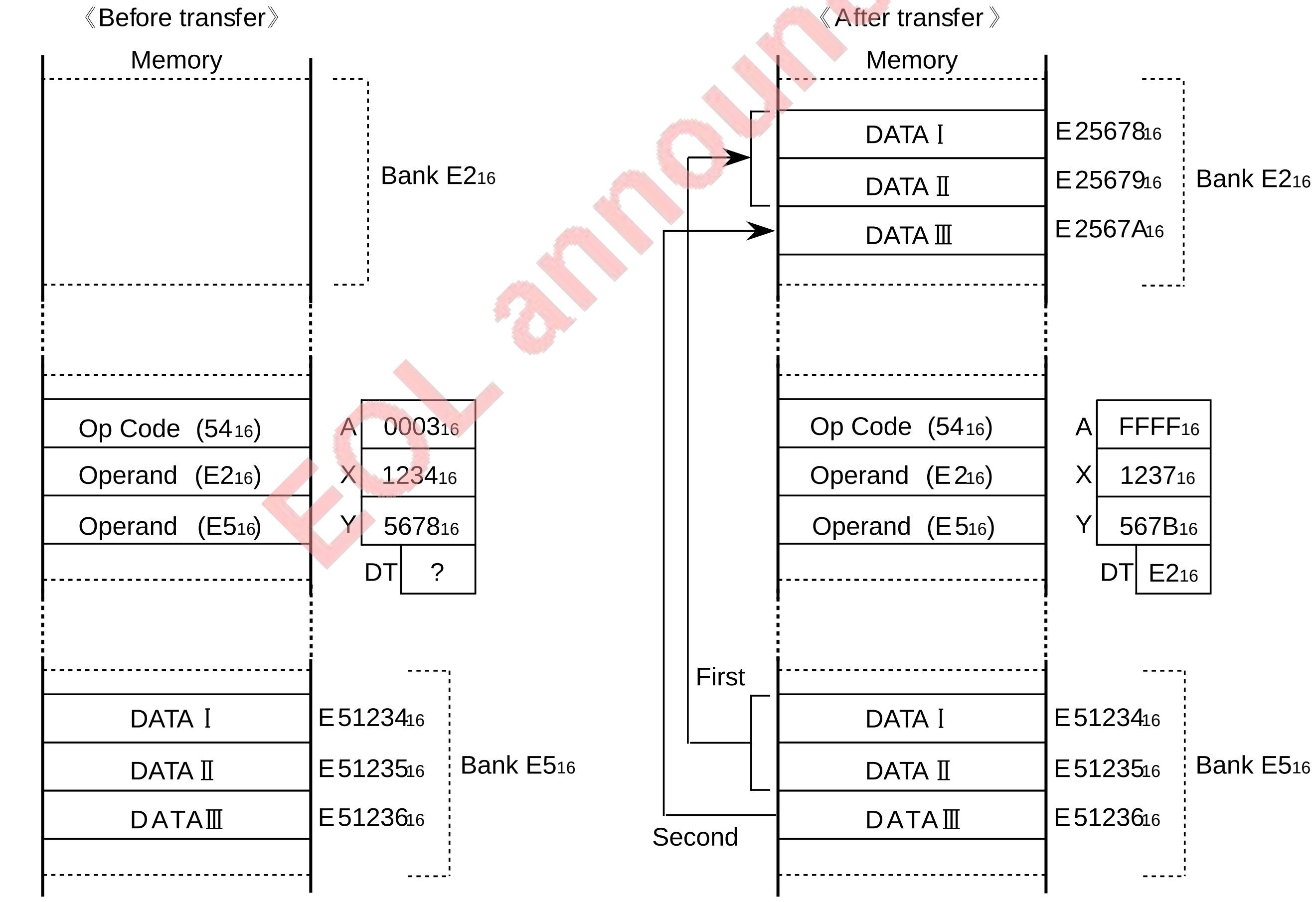
Function : Specifies the transfer-to data bank by the instruction's second byte, and specifies the transfer-to address whithin the data bank by the index register Y's contents. Specifies the transfer-from data bank by the instruction's third byte, and specifies the address of transfer data within the data bank by the index register X's contents. The accumulator A's contents are the number of bytes to be transferred. At termination of transfer, the data bank register's contents specify the transfer-to data bank. The MVN instruction is used for transfer toward lower addresses. In this case, the contents of the index registers X and Y are incremented each time data is transferred. The MVP instruction is used for transfer toward higher addresses. In this case, the contents of the index registers X and Y are decremented each time data is transferred. The transfer data can cross over the bank boundary.

Instruction : MVN, MVP

ex. : Mnemonic — Machine code

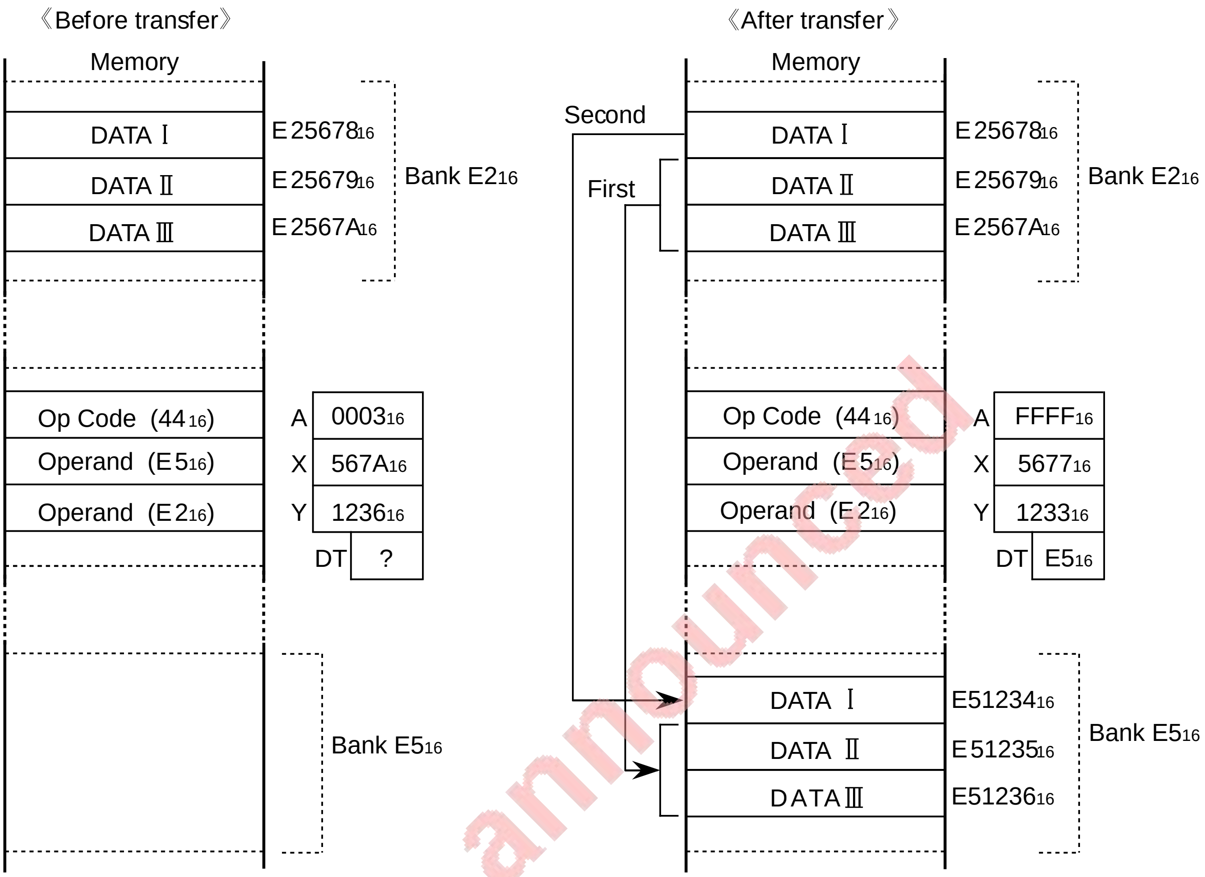
MVN 0E2H ,0E5H — 54_{16} $E2_{16}$ $E5_{16}$

(m="0", x="0")



ex. : Mnemonic　　　　　　Machine code

MVP 0E5H, 0E2H　　　44_{16} $E5_{16}$ $E2_{16}$
(m=“0”, x=“0”)



Note : For block transfer instructions, the number of bytes to be transferred and the range of transfer source/destination address change with the state of the m and x flags. However, the transfer unit is unaffected. The transfer unit is word (16 bits). However, only 1 byte is transferred when transferring the last byte at odd bytes transfer.

Multiplied accumulation

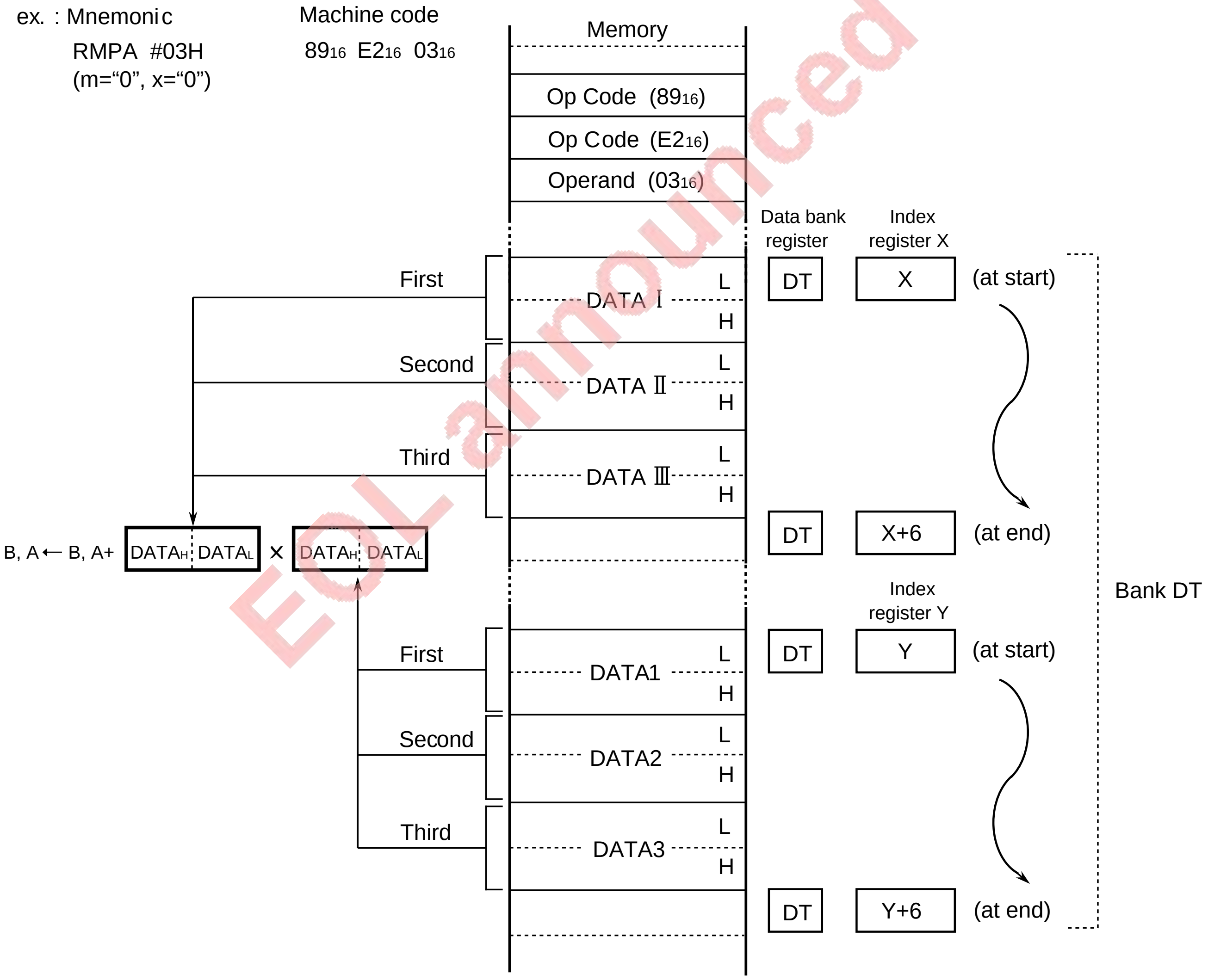
Mode : Multiplied accumulation addressing mode

Function : The following is a multiplicand and a multiplier: the contents of the memory location specified by the contents of index registers X and Y, and the data bank register's contents. The instruction's third byte is the repeat number of arithmetic operation. The contents of index registers X and Y are incremented each time the addition of the contents of accumulators B and A to the multiplication result finishes. Accordingly, the contents of index registers X and Y specify the next address where the multiplicand and the multiplier are read at last.

Allocate a multiplicand and a multiplier within the same bank and do not cross them over the bank boundary.

Set the index register length flag to "0" before executing this instruction.

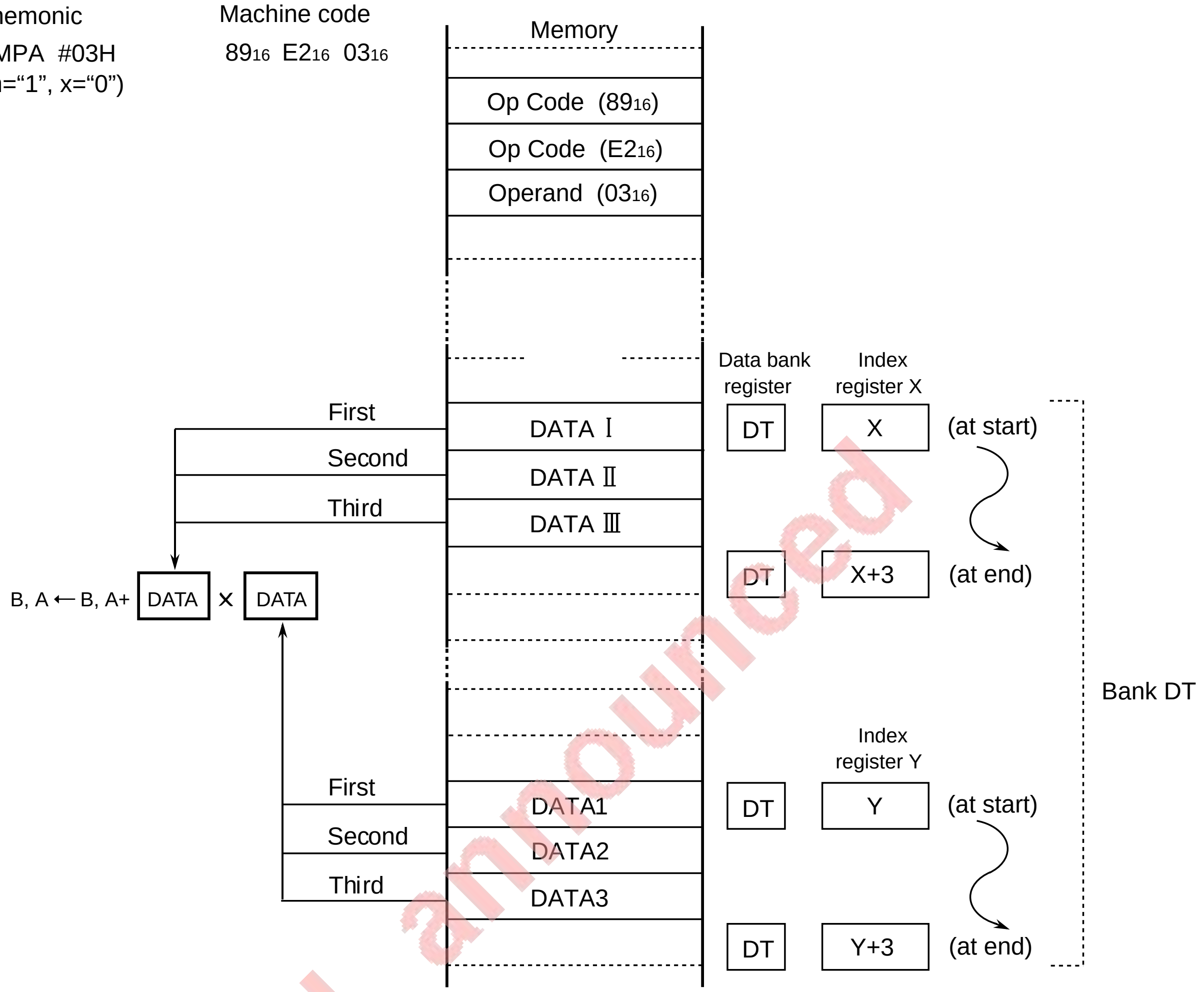
Instruction : RMPA



Multiplied accumulation

ex. : Mnemonic
RMPA #03H
(m="1", x="0")

Machine code
89_{16} $E2_{16}$ 03_{16}



MEMO

CHAPTER 4
INSTRUCTIONS

4.1 Instruction set

The 7751 series CPU uses the instruction set with 109 instructions.

4.1.1 Data transfer instructions

The data transfer instructions move data between data and registers, between a register and a memory, between registers or between memories.
The following table shows the available instructions for data transfer.

Category	Instruction	Description
Load	LDA	Loads the contents of memory into the accumulator.
	LDM	Loads an immediate value into the memory.
	LDT	Loads an immediate value into the data bank register.
	LDX	Loads the contents of memory into the index register X.
	LDY	Loads the contents of memory into the index register Y.
Store	STA	Stores the contents of the accumulator in the memory.
	STX	Stores the contents of the index register X in the memory.
	STY	Stores the contents of the index register Y in the memory.
Transfer	TAX	Transfers the contents of the accumulator A to the index register X.
	TXA	Transfers the contents of the index register X to the accumulator A.
	TAY	Transfers the contents of the accumulator A to the index register Y.
	TYA	Transfers the contents of the index register Y to the accumulator A.
	TSX	Transfers the contents of the stack pointer to the index register X.
	TXS	Transfers the contents of the index register X to the stack pointer.
	TAD	Transfers the contents of the accumulator A to the direct page register.
	TDA	Transfers the contents of the direct page register to the accumulator A.
	TAS	Transfers the contents of the accumulator A to the stack pointer.
	TSA	Transfers the contents of the stack pointer to the accumulator A.
	TBD	Transfers the contents of the accumulator B to the direct page register.
	TDB	Transfers the contents of the direct page register to the accumulator B.
	TBS	Transfers the contents of the accumulator B to the stack pointer.
	TSB	Transfers the contents of the stack pointer to the accumulator B.
	TBX	Transfers the contents of the accumulator B to the index register X.
	TXB	Transfers the contents of the index register X to the accumulator B.
	TBY	Transfers the contents of the accumulator B to the index register Y.
	TYB	Transfers the contents of the index register Y to the accumulator B.
	TXY	Transfers the contents of the index register X to the index register Y.
	TYX	Transfers the contents of the index register Y to the index register X.
	MVN	Transfers a block of data from the lower addresses.
	MVP	Transfers a block of data from the higher addresses.

Category	Instruction	Description
Stack operation	PSH	Pushes the contents of the specified register to the stack.
	PUL	Restores the contents of stack to the specified register.
	PHA	Pushes the contents of the accumulator A to the stack.
	PLA	Restores the contents of stack to the accumulator A.
	PHP	Pushes the contents of the processor status register to the stack.
	PLP	Restores the contents of stack to the processor status register.
	PHB	Pushes the contents of the accumulator B to the stack.
	PLB	Restores the contents of stack to the accumulator B.
	PHD	Pushes the contents of the direct page register to the stack.
	PLD	Restores the contents of stack to the direct page register.
	PHT	Pushes the contents of the data bank register to the stack.
	PLT	Restores the contents of stack to the data bank register.
	PHX	Pushes the contents of the index register X to the stack.
	PLX	Restores the contents of stack to the index register X.
	PHY	Pushes the contents of the index register Y to the stack.
	PLY	Restores the contents of stack to the index register Y.
	PHG	Pushes the contents of the program bank register to the stack.
	PEA	Pushes a numerical value of 2 bytes to the stack.
	PEI	Pushes the contents of a sequence of 2 bytes in the direct page area to the stack.
	PER	Pushes the result of adding a 16-bit numerical value to the program counter's contents to the stack.
Exchange	XAB	Exchanges the contents of the accumulator A for the contents of the accumulator B.

4.1.2 Arithmetic instructions

The arithmetic instructions perform addition, subtraction, multiplication, division, multiplied accumulation, logical operation, comparison, rotation, shift and sign/zero extension of register and memory contents. The following table shows the available instructions for arithmetic operation.

Category	Instruction	Description
Addition, Subtraction, Multiplication, Division	ADC	Adds the contents of the accumulator, the contents of a memory and the contents of the carry flag.
	SBC	Subtracts the contents of memory and the complement of the carry flag from the contents of the accumulator.
	INC	Increments the accumulator or a memory contents by 1.
	DEC	Decrements the accumulator or a memory contents by 1.
	INX	Increments the contents of the index register X by 1.
	DEX	Decrements the contents of the index register X by 1.
	INY	Increments the contents of the index register Y by 1.
	DEY	Decrements the contents of the index register Y by 1.
	MPY	Multiples the contents of the accumulator A and the contents of a memory.
	MPYS	Multiples the contents of the accumulator A and the contents of a memory with sign.
	DIV	Divides the numerical value whose low order is the contents of the accumulator A and high order is the contents of the accumulator B by the contents of a memory.
	DIVS	Divides the numerical value whose low order is the contents of the accumulator A and high order is the contents of the accumulator B by the contents of a memory with sign.
Multiplied accumulation	RMPA	Multiples the contents of a memory and another one, and adds the result to the contents of the accumulator. Repeats these operations by specified times.
Logical operation	AND	Performs logical AND between the contents of the accumulator and a memory.
	ORA	Performs logical OR between the contents of the accumulator and a memory.
	EOR	Performs logical exclusive-OR between the contents of the accumulator and a memory.
Comparison	CMP	Compares the contents of the accumulator with the contents of a memory.
	CPX	Compares the contents of the index register X with the contents of a memory.
	CPY	Compares the contents of the index register Y with the contents of a memory.
Shift, Rotation	ASL	Shifts the contents of the accumulator or a memory to the left by 1 bit.
	ASR	Shifts the contents of the accumulator or a memory holding sign to the right by 1 bit.
	LSR	Shifts the contents of the accumulator or a memory to the right by 1 bit.
	ROL	Links the contents of the accumulator or a memory with the carry flag, and rotates the result to the left by 1 bit.
	ROR	Links the contents of the accumulator or a memory with the carry flag, and rotates the result to the right by 1 bit.
	RLA	Rotates the contents of the accumulator A to the left by the specified number of bits.
Extension with sign / zero	EXTS	Extends the low-order 8 bits of the accumulator to 16 bits by sign extension.
	EXTZ	Extends the low-order 8 bits of the accumulator to 16 bits by zero extension.

4.1.3 Bit manipulation instructions

The bit manipulation instructions set the specified bits of the processor status register or a memory to “1” or “0”.

The following table shows the available instructions for bit manipulation.

Category	Instruction	Description
Bit manipulation	CLB	Clears the specified bit of a memory to “0”.
	SEB	Sets the specified bit of a memory to “1”.
	CLP	Clears the specified bit of the processor status register’s low-order byte (PSL) to “0”.
	SEP	Sets the specified bit of the processor status register’s low-order byte (PSL) to “1”.

4.1.4 Flag manipulation instructions

The flag manipulation instructions set the flag, which is the C, I, m or V flag; to “1” or “0”.

The following table shows the available instructions for flag manipulation.

Category	Instruction	Description
Flag manipulation	CLC	Clears the contents of the carry flag to “0”.
	SEC	Sets the contents of the carry flag to “1”.
	CLM	Clears the contents of the data length select flag to “0”.
	SEM	Sets the contents of the data length select flag to “1”.
	CLI	Clears the contents of the interrupt disable flag to “0”.
	SEI	Sets the contents of the interrupt disable flag to “1”.
	CLV	Clears the contents of the overflow flag to “0”.

4.1.5 Branch and return instructions

The branch and return instructions enable changing program execution sequence.

The following table shows the available instructions for branch and return.

Category	Instruction	Description
Jump	JMP	Sets a new address in the program counter and jumps to the new address.
	BRA	Jumps to the address obtained by adding an offset value to the contents of the program counter.
	JSR	Pushes the contents of the program counter to the stack and then jumps to the new address.

Category	Instruction	Description
Branch	BBC	Branches when the specified bits of a memory are all “0”.
	BBS	Branches when the specified bits of a memory are all “1”.
	BCC	Branches when the carry flag is “0”.
	BCS	Branches when the carry flag is “1”.
	BNE	Branches when the zero flag is “0”.
	BEQ	Branches when the zero flag is “1”.
	BPL	Branches when the negative flag is “0”.
	BMI	Branches when the negative flag is “1”.
	BVC	Branches when the overflow flag is “0”.
	BVS	Branches when the overflow flag is “1”.
Return	RTI	Returns from an interrupt routine to the original routine.
	RTS	Returns from a subroutine to the original routine. The program bank register’s contents are not restored.
	RTL	Returns from a subroutine to the original routine. The program bank register’s contents are also restored.

4.1.6 Interrupt instruction (break instruction)

The interrupt instruction executes a software interrupt.

Category	Instruction	Description
Break	BRK	Executes a software interrupt.

4.1.7 Special instructions

The special instructions showed by the following table control the clock generating circuit.

Category	Instruction	Description
Special	WIT	Stops the internal clock.
	STP	Stops the oscillator’s oscillation.

4.1.8 Other instruction

Category	Instruction	Description
Other	NOP	Only advances the program counter and performs nothing else.

4.2 Description of each instruction

This section describes each instruction of the 7751 series. Each instruction is described using one page per one instruction as a general rule. The description page is headed by the instruction mnemonic, and the pages are arranged in alphabetical order of the mnemonics. For each instruction, its operation and description (Notes 1, 2), status flag change and a list sorted by addressing modes of the assembler coding format (Note 3), the machine code, the byte number and the minimum cycle number (Note 4) are presented.

Note 1: In the description of each instruction operation, the operation regarding the PC (program counter) is described only for instructions affecting the processing.
When an instruction is executed, its instruction bytes are added to the contents of the PC and the PC contains the address of the memory location of the next instruction to be executed. When a carry occurs at this addition, the PG (program bank register) is incremented by 1.

Note 2: [**Operation**] in the description of each instruction shows the contents of each register and memory after executing the instruction. The detailed operation sequence is omitted.

Note 3: The assembler coding formats shown are general examples, and they may differ from the actual formats for the assembler used. Make sure that refer to the mnemonic coding description in the manual for the assembler actually used for programming.

Note 4: The cycle number shown is the minimum possible, and they depend on the following conditions:

•Value of direct page register's low-order byte
The cycle number shown is a number when the direct page register's low-order byte (DPRL) is 00_{16}. When using an addressing mode that uses the direct page register in the condition of DPRL ≠ "00_{16}", the cycle number which is obtained by adding 1 to the shown number is an actual number.

•Number of bytes that have been loaded in the instruction queue buffer

•Whether the address of the memory read/write is even or odd

•Accessing of an external memory are in the condition of BYTE = 1 (using 8-bit external bus)

•Bus cycle.

4.2 Description of each instruction

The following table shows the symbols that are used in instructions' description and the lists of this section, and each instruction is described bellow.

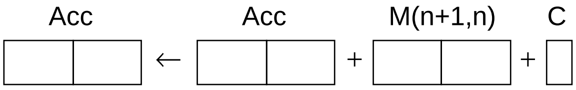
Symbol	Description
C	Carry flag
Z	Zero flag
I	Interrupt disable flag
D	Decimal mode flag
x	Index register length flag
m	Data length flag
V	Overflow flag
N	Negative flag
IPL	Processor interrupt priority level
+	Addition
–	Subtraction
×	Multiplication
/	Division
∧	Logical AND
∨	Logical OR
∀	Exclusive OR
—	Negation
←	Movement toward the arrow direction
→	Movement toward the arrow direction
⇆	Movement toward the arrow direction
Acc	Accumulator
AccH	Accumulator's high-order 8 bits
AccL	Accumulator's low-order 8 bits
A	Accumulator A
AH	Accumulator A's high-order 8 bits
AL	Accumulator A's low-order 8 bits
B	Accumulator B
BH	Accumulator B's high-order 8 bits
BL	Accumulator B's low-order 8 bits
X	Index register X
XH	Index register X's high-order 8 bits
XL	Index register X's low-order 8 bits
Y	Index register Y
YH	Index register Y's high-order 8 bits
YL	Index register Y's low-order 8 bits
S	Stack pointer
PC	Program counter
PCH	Program counter's high-order 8 bits
PCL	Program counter's low-order 8 bits
REL	Relative address
PG	Program bank register
DT	Data bank register

Symbol	Description
DPR	Direct page register
DPR$_H$	Direct page register's high-order 8 bits
DPR$_L$	Direct page register's low-order 8 bits
PS	Processor status register
PS$_H$	Processor status register's high-order 8 bits
PS$_L$	Processor status register's low-order 8 bits
PS$_{Ln}$	Bits of processor status register's low-order 8 bits
M	Memory contents
M(n)	Contents of memory location specified by operand (1-byte data)
M(n+1,n)	Contents of memory location specified by operand (1-word data)
M(m to n)	Contents of memory location specified by operand (plural-byte data)
M(S)	Contents of memory at address indicated by stack pointer
Mb	Bits of memory
ADDR	Low-order 16 bits (A$_{15}$ to A$_0$) of 24-bit address
BANK	High-order 8 bits (A$_{23}$ to A$_{16}$)
IMM	Immediate data
IMM16	16-bit immediate data
IMM$_H$	High-order 8 bits of 16-bit immediate data
IMM$_L$	Low-order 8 bits of 16-bit immediate data
IMM8	8-bit immediate data
bn	n-th bit of data
dd	Displacement for the DPR (8 bits)
i	Number of transfer bytes or rotation
i$_1$, i$_2$	Number of registers pushed or pulled
imm	8-bit immediate value
imm$_H$imm$_L$	16-bit immediate value (imm$_H$ represents the high-order 8 bits, and imm$_L$ represents the low-order 8 bits)
mmll	16-bit address value (mm represents the high-order 8 bits, and ll represents the low-order 8 bits)
hhmmll	24-bit address value (hh represents the high-order 8 bits, mm represents the middle-order 8 bit, and ll represents the low-order 8 bits)
nn	Displacement for the S (8 bits)
rr	Displacement for the PC (signed 8 bits)
rr$_H$rr$_L$	Displacement for the PC (signed 16 bits)
hh$_1$, hh$_2$	Bank specification (2 types of 8-bit data)

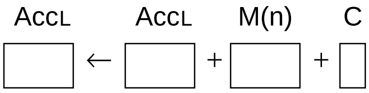
Function : Addition with carry

Operation : Acc ← Acc + M + C

When m = “0”



When m = “1”



Description : Adds the contents of the accumulator, memory and carry flag, and places the result in the accumulator.

Executed as binary addition when the decimal mode flag is “0”.

Executed as decimal addition when the decimal mode flag is “1”.

Status flags

IPL: Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”. Meaningless for decimal addition.

V : Set to “1” when binary addition of signed data results in a value outside the range of –32768 to +32767 (–128 to +127 when the data length flag is “1”). Otherwise, cleared to “0”. Meaningless for decimal addition.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”. Meaningless for decimal addition.

C : Set to “1” when the result of binary addition as unsigned data exceeds +65535 (when the data length flag is “1”, it does +255). Otherwise, cleared to “0”.

Set to “1” when the result of decimal addition as unsigned data exceeds +9999 (when the data length flag is “1”, it does +99). Otherwise, cleared to “0”.

ADC ADd with Carry ADC

Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	ADC A, #imm	69_{16}, imm	2	2
Direct	ADC A, dd	65_{16}, dd	2	4
Direct indexed X	ADC A, dd, X	75_{16}, dd	2	5
Direct indirect	ADC A, (dd)	72_{16}, dd	2	6
Direct indexed X indirect	ADC A, (dd, X)	61_{16}, dd	2	7
Direct indirect indexed Y	ADC A, (dd), Y	71_{16}, dd	2	8
Direct indirect long	ADCL A, (dd)	67_{16}, dd	2	8
Direct indirect long indexed Y	ADCL A, (dd), Y	77_{16}, dd	2	10
Absolute	ADC A, mmll	$6D_{16}$, ll, mm	3	4
Absolute indexed X	ADC A, mmll, X	$7D_{16}$, ll, mm	3	6
Absolute indexed Y	ADC A, mmll, Y	79_{16}, ll, mm	3	6
Absolute long	ADC A, hhmmll	$6F_{16}$, ll, mm, hh	4	6
Absolute long indexed X	ADC A, hhmmll, X	$7F_{16}$, ll, mm, hh	4	7
Stack pointer relative	ADC A, nn,S	63_{16}, nn	2	5
Stack pointer relative indirect indexed Y	ADC A, (nn, S), Y	73_{16}, nn	2	8

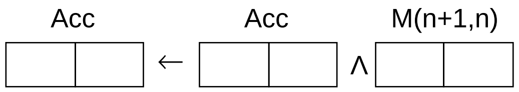
Notes 1: This table applies when using the accumulator A. When using the accumulator B, replace “A” with “B” in the syntax. In this case, “42_{16}” is added at the beginning of the machine code, the byte number increases by 1 and the cycle number increases by 2.

2: When treating a 16-bit data in the immediate addressing mode in the condition of the data length flag = “0”, the byte number increases by 1.

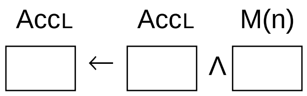
Function : Logical AND

Operation : Acc ← Acc ∧ M

When m = “0”



When m = “1”



Description : Performs logical AND between the contents of the accumulator and the contents of a memory, and places the result in the accumulator.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

AND logical AND AND

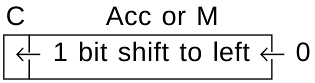
Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	AND A, #imm	29_{16}, imm	2	2
Direct	AND A, dd	25_{16}, dd	2	4
Direct indexed X	AND A, dd, X	35_{16}, dd	2	5
Direct indirect	AND A, (dd)	32_{16}, dd	2	6
Direct indexed X indirect	AND A, (dd, X)	21_{16}, dd	2	7
Direct indirect indexed Y	AND A, (dd), Y	31_{16}, dd	2	8
Direct indirect long	ANDL A, (dd)	27_{16}, dd	2	8
Direct indirect long indexed Y	ANDL A, (dd), Y	37_{16}, dd	2	10
Absolute	AND A, mmll	$2D_{16}$, ll, mm	3	4
Absolute indexed X	AND A, mmll, X	$3D_{16}$, ll, mm	3	6
Absolute indexed Y	AND A, mmll, Y	39_{16}, ll, mm	3	6
Absolute long	AND A, hhmmll	$2F_{16}$, ll, mm, hh	4	6
Absolute long indexed X	AND A, hhmmll, X	$3F_{16}$, ll, mm, hh	4	7
Stack pointer relative	AND A, nn, S	23_{16}, nn	2	5
Stack pointer relative indirect indexed Y	AND A, (nn, S), Y	33_{16}, nn	2	8

Notes 1: This table applies when using the accumulator A. When using the accumulator B, replace "A" with "B" in the syntax. In this case, "42_{16}" is added at the beginning of the machine code, the byte number increases by 1 and the cycle number increases by 2.

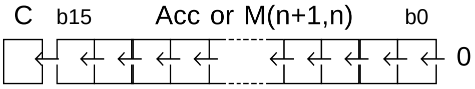
2: When treating a 16-bit data in the immediate addressing mode in the condition of the data length flag = "0", the byte number increases by 1.

Function : Arithmetic shift left

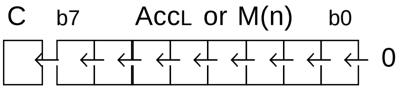
Operation :



When m = “0”



When m = “1”



Description : Shifts all bits of the accumulator or a memory to the one bit left. Its bit 0 is loaded with 0. The carry flag is loaded from bit 15 (or bit 7 when the data length flag is “1”) of the data before the shift.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

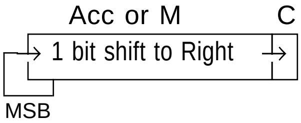
C : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the accumulator or a memory before the operation is “1”. Otherwise, cleared to “0”.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Accumulator	ASL A	$0A_{16}$	1	2
Direct	ASL dd	06_{16}, dd	2	7
Direct indexed X	ASL dd, X	16_{16}, dd	2	7
Absolute	ASL mmll	$0E_{16}$, ll, mm	3	7
Absolute indexed X	ASL mmll, X	$1E_{16}$, ll, mm	3	8

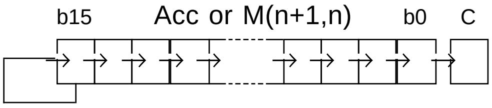
Note: This table applies when using the accumulator A. When using the accumulator B, replace “A” with “B” in the syntax. In this case, “42_{16}” is added at the beginning of the machine code, the byte number increases by 1 and the cycle number increases by 2.

Function : Arithmetic shift right

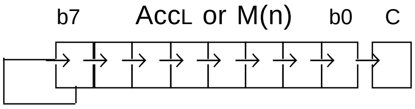
Operation :



When m = “0”



When m = “1”



Description : Shifts all bits of the accumulator or a memory to the one bit right. Its bit 15 (or bit 7 when the data length flag is “1”) is loaded with the value before the shift. The carry flag is loaded from bit 0 of the data before the shift.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Set to “1” when bit 0 before the operation is “1”. Otherwise, cleared to “0”.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Accumulator	ASR A	89_{16}, 08_{16}	2	4
Accumulator	ASR B	42_{16}, 08_{16}	2	4
Direct	ASR dd	89_{16}, 06_{16}, dd	3	9
Direct indexed X	ASR dd, X	89_{16},16_{16}, dd	3	9
Absolute	ASR mmll	89_{16}, $0E_{16}$, ll, mm	4	9
Absolute indexed X	ASR mmll, X	89_{16}, $1E_{16}$, ll, mm	4	10

Function : Branch on condition

Operation : Mb = 0 ? (b is the specified bits)

When $M \wedge IMM = 0$ (True) $PC \leftarrow PC + n \pm REL$

When $M \wedge IMM \neq 0$ (False) $PC \leftarrow PC + n$

❊ PG changes according to the result of the above PC operation

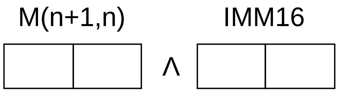
- if carry occurs in PC : $PG \leftarrow PG + 1$
- if borrow occurs in PC : $PG \leftarrow PG - 1$

❊ IMM is an immediate value indicating the bit to be tested with “1”.

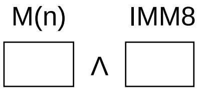
❊ n is the number of instruction bytes in each addressing mode of the BBC instruction

❊ REL is a relative value (–128 to +127) indicated by the last byte of the instruction

When m = “0”



When m = “1”



Description : Tests the specified bits, which may be specified simultaneously, of a memory. The instruction causes a branch to the specified address when the specified bits are all “0”. The branch address is specified by a relative address.

When the specified bits are nothing, that is, IMM is all “0”, a branch is caused.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Direct bit relative	BBC #imm, dd, rr	34_{16}, dd, imm, rr	4	7
Absolute bit relative	BBC #imm, mmll, rr	$3C_{16}$, ll, mm, imm, rr	5	8

Note: The byte number increases by 1 when treating on 16-bit data in the condition of the data length flag = “0”.

Function : Branch on condition

Operation : Mb = 1 ? (b is the specified bits)

When $\overline{M} \wedge IMM = 0$ (True) PC ← PC + n ± REL

When $\overline{M} \wedge IMM \neq 0$ (False) PC← PC + n

❊ PG changes according to the result of the above PC operation

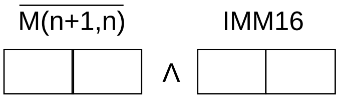
- if carry occurs in PC : PG ← PG + 1
- if borrow occurs in PC : PG ← PG – 1

❊ IMM is an immediate value indicating the bit to be tested with “1”.

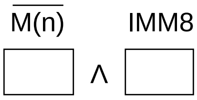
❊ n is the number of instruction bytes in each addressing mode of the BBS instruction

❊ REL is a relative value (–128 to +127) indicated by the last byte of the instruction

When m = “0”



When m = “1”



Description : Tests the specified bits, which may be specified simultaneously, of a memory. The instruction causes a branch to the specified address when the specified bits are all “1”. The branch address is specified by a relative address.
When the specified bits are nothing, that is, IMM is all “0”, a branch is caused.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Direct bit relative	BBS #imm, dd, rr	24_{16}, dd, imm, rr	4	7
Absolute bit relative	BBS #imm, mmll, rr	$2C_{16}$, ll, mm, imm, rr	5	8

Note: The byte number increases by 1 when treating on 16-bit data in the condition of the data length flag = “0”.

Branch on Carry Clear

Function : Branch on condition

Operation : C = 0 ?

When C = 0 (True) PC ← PC + 2 ± REL

When C = 1 (False) PC ← PC + 2

❊ PG changes according to the result of the above PC operation

- if carry occurs in PC : PG ← PG + 1
- if borrow occurs in PC : PG ← PG – 1

❊ 2 is the number of instruction bytes of the BCC instruction

❊ REL is a relative value (–128 to +127) indicated by the 2nd byte of the instruction

Description : When the carry flag is “0”, the BCC instruction causes a branch to the specified address. The branch address is specified by a relative address.
When the carry flag is “1”, the program advances to next step without any action.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Relative	BCC rr	90_{16}, rr	2	4

Branch on Carry Set

Function : Branch on condition

Operation : C = 1 ?

When C = 1 (True) PC ← PC + 2 ± REL

When C = 0 (False) PC ← PC + 2

❊ PG changes according to the result of the above PC operation

- if carry occurs in PC : PG ← PG + 1
- if borrow occurs in PC : PG ← PG – 1

❊ 2 is the number of instruction bytes of the BCS instruction

❊ REL is a relative value (–128 to +127) indicated by the 2nd byte of the instruction

Description : When the carry flag is “1”, the BCS instruction causes a branch to the specified address. The branch address is specified by a relative address.
When the carry flag is “0”, the program advances to next step without any action.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Relative	BCS rr	$B0_{16}$, rr	2	4

BEQ — Branch on EQual — BEQ

Function : Branch on condition

Operation : Z = 1 ?

When Z = 1 (True) PC ← PC + 2 ± REL

When Z = 0 (False) PC ← PC + 2

❊ PG changes according to the result of the above PC operation

- if carry occurs in PC : PG ← PG + 1
- if borrow occurs in PC : PG ← PG – 1

❊ 2 is the number of instruction bytes of the BEQ instruction

❊ REL is a relative value (–128 to +127) indicated by the 2nd byte of the instruction

Description : When the zero flag is “1”, the BEQ instruction causes a branch to the specified address. The branch address is specified by a relative address.
When the zero flag is “0”, the program advances to next step without any action.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Relative	BEQ rr	$F0_{16}$, rr	2	4

Function : Branch on condition

Operation : N = 1 ?

When N = 1 (True) PC ← PC + 2 ± REL

When N = 0 (False) PC ← PC + 2

❊ PG changes according to the result of the above PC operation

- if carry occurs in PC : PG ← PG + 1
- if borrow occurs in PC : PG ← PG – 1

❊ 2 is the number of instruction bytes of the BMI instruction

❊ REL is a relative value (–128 to +127) indicated by the 2nd byte of the instruction

Description : When the negative flag is “1”, the BMI instruction causes a branch to the specified address. The branch address is specified by a relative address.
When the negative flag is “0”, the program advances to next step without any action.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Relative	BMI rr	30_{16}, rr	2	4

BNE Branch on Not Equal BNE

Function : Branch on condition

Operation : Z = 0 ?

When Z = 0 (True) PC ← PC + 2 ± REL

When Z = 1 (False) PC ← PC + 2

❊ PG changes according to the result of the above PC operation

- if carry occurs in PC : PG ← PG + 1
- if borrow occurs in PC : PG ← PG – 1

❊ 2 is the number of instruction bytes of the BNE instruction

❊ REL is a relative value (–128 to +127) indicated by the 2nd byte of the instruction

Description : When the zero flag is “0”, the BNE instruction causes a branch to the specified address. The branch address is specified by a relative address.
When the zero flag is “1”, the program advances to next step without any action.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Relative	BNE rr	$D0_{16}$, rr	2	4

BPL　Branch on result PLus　BPL

Function : Branch on condition

Operation : N = 0 ?

When N = 0 (True)　PC ← PC + 2 ± REL

When N = 1 (False)　PC ← PC + 2

❊ PG changes according to the result of the above PC operation

- if carry occurs in PC : PG ← PG + 1
- if borrow occurs in PC : PG ← PG – 1

❊ 2 is the number of instruction bytes of the BPL instruction

❊ REL is a relative value (–128 to +127) indicated by the 2nd byte of the instruction

Description : When the negative flag is “0”, the BPL instruction causes a branch to the specified address. The branch address is specified by a relative address.
When the negative flag is “1”, the program advances to next step without any action.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Relative	BPL rr	10_{16}, rr	2	4

BRA BRanch Always BRA

Function : Branch always

Operation : PC ← branch address (relative)

PC ← PC + n ± REL

❊ PG changes according to the result of the above PC operation
❊ n is the number of instruction bytes in each addressing mode of the BRA instruction
❊ REL is a relative value (–128 to +127) indicated by the last 1-byte or last 2-byte of the instruction

Branch area :	For short relative	–128 to +127
	For long relative	–32768 to +32767

Description : The BRA instruction causes a branch to the specified address. The branch address is specified by a relative address.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Relative	BRA rr BRAL $rr_H rr_L$	80_{16}, rr 82_{16}, rr_L, rr_H	2 3	3 3

Function : Software interrupt

Operation : Stack ← PG, PC, PS

I ← 1

PG, PC ← 00, Contents of BRK interrupt vector

PC ← PC + 2

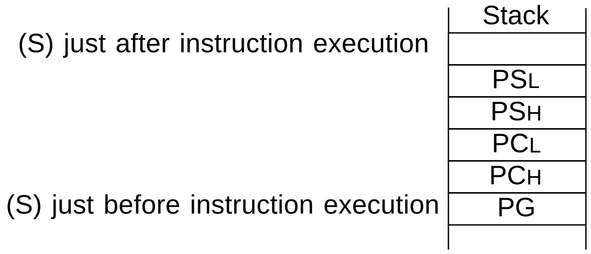
M(S to S–4) ← PG, PC, PS

S ← S – 5

I ← 1

PG ← 00_{16}

PC ← M($FFFB_{16}$,$FFFA_{16}$)


Stack
(S) just after instruction execution
PSL
PSH
PCL
PCH
(S) just before instruction execution
PG

❊ 2 is the number of instruction bytes of the BRK instruction, and PC+2 is the address where the next instruction is stored

Description : When the BRK instruction is executed, the CPU first saves the address where the next instruction is stored, and then saves the contents of the processor status register in the stack. The CPU causes a branch to the address in bank 0_{16} of which low-order address is the contents of $FFFA_{16}$ in bank 0 and high-order address is the contents of $FFFB_{16}$ in bank 0.

Status flags

IPL : Not affected.

N : Not affected.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Set to “1”.

Z : Not affected.

C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	BRK #imn	00_{16}, EA_{16}	2	15

Note: The instruction’s second byte is ignored, so that any value is available.

Branch on oVerflow Clear

Function : Branch on condition

Operation : V = 0 ?

When V = 0 (True) PC ← PC + 2 ± REL

When V = 1 (False) PC ← PC + 2

❊ PG changes according to the result of the above PC operation

- if carry occurs in PC : PG ← PG + 1
- if borrow occurs in PC : PG ← PG – 1

❊ 2 is the number of instruction bytes of the BVC instruction

❊ REL is a relative value (–128 to +127) indicated by the 2nd byte of the instruction

Description : When the overflow flag is "0", the BVC instruction causes a branch to the specified address. The branch address is specified by a relative address.
When the overflow flag is "1", the program advances to next step without any action.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Relative	BVC rr	50_{16}, rr	2	4

Branch on oVerflow Set

Function : Branch on condition

Operation : V = 1 ?

When V = 1 (True) PC ← PC + 2 ± REL

When V = 0 (False) PC ← PC + 2

❊ PG changes according to the result of the above PC operation

- if carry occurs in PC : PG ← PG + 1
- if borrow occurs in PC : PG ← PG – 1

❊ 2 is the number of instruction bytes of the BVS instruction

❊ REL is a relative value (–128 to +127) indicated by the 2nd byte of the instruction

Description : When the overflow flag is “1”, the BVS instruction causes a branch to the specified address.
The branch address is specified by a relative address.
When the overflow flag is “0”, the program advances to next step without any action.

Status flags : Not affected.

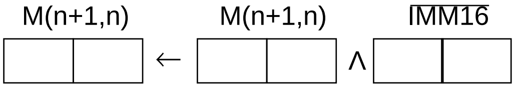
Addressing mode	Syntax	Machine code	Bytes	Cycles
Relative	BVS rr	70_{16}, rr	2	4

CLear Bit

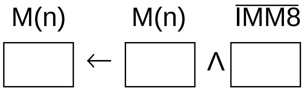
Function : Bit manipulation

Operation : Mb ← 0 (b is the specified bits)

When m = "0"



When m = "1"



❊ IMM is an immediate value indicating the bit to be cleared with a "1". It is specified by the last 1 or 2 bytes of the instruction.

Description : The CLB instruction clears the specified memory bits to "0". Multiple bits to be cleared can be specified at the same time.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Direct bit	CLB #imm, dd	14_{16}, dd, imm	3	8
Absolute bit	CLB #imm, mmll	$1C_{16}$, ll, mm, imm	4	9

Note: The byte number increases by 1 when treating on 16-bit data in the condition of the data length flag = "0".

CLear Carry flag

Function : Flag manipulation

Operation : C ← 0

Description : Clears the contents of carry flag to “0”.

Status flags

IPL :	Not affected.
N :	Not affected.
V :	Not affected.
m :	Not affected.
x :	Not affected.
D :	Not affected.
I :	Not affected.
Z :	Not affected.
C :	Cleared to “0”.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	CLC	18_{16}	1	2

Function : Flag manipulation

Operation : I ← 0

Description : Clears the interrupt disable flag to “0”.

Status flags

IPL	:	Not affected.
N	:	Not affected.
V	:	Not affected.
m	:	Not affected.
x	:	Not affected.
D	:	Not affected.
I	:	Cleared to “0”.
Z	:	Not affected.
C	:	Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	CLI	58_{16}	1	2

CLear M flag

Function : Flag manipulation

Operation : m ← 0

Description : Clears the data length flag to “0”.

Status flags

IPL :	Not affected.
N :	Not affected.
V :	Not affected.
m :	Cleared to “0”.
x :	Not affected.
D :	Not affected.
I :	Not affected.
Z :	Not affected.
C :	Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	CLM	$D8_{16}$	1	2

CLP CLear Processor status CLP

Function : Flag manipulation

Operation : $PS_{L}b \leftarrow 0$ (b is the specified flags)

$PS_{L} \leftarrow PS_{L} \land \overline{IMM8}$

❊ IMM is a 1-byte immediate value indicating the flag to be cleared with a “1”. It is specified by the second byte of the instruction.

b7	b6	b5	b4	b3	b2	b1	b0	
N	V	m	x	D	I	Z	C	PS_{L}

Description : Clears the processor status flags specified by the bit pattern in the second byte of the instruction to “0”.

Status flags : The specified status flags are cleared to “0”. IPL is not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	CLP #imm	$C2_{16}$, imm	2	4

CLear oVerflow flag

Function : Flag manipulation

Operation : V ← 0

Description : Clears the overflow flag to “0”.

Status flags

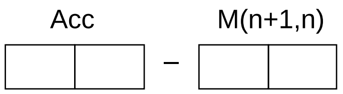
IPL	:	Not affected.
N	:	Not affected.
V	:	Cleared to “0”.
m	:	Not affected.
x	:	Not affected.
D	:	Not affected.
I	:	Not affected.
Z	:	Not affected.
C	:	Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	CLV	$B8_{16}$	1	2

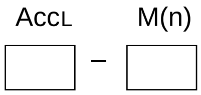
Function : Compare

Operation : Acc – M

When m = “0”



When m = “1”



Description : Subtracts the contents of a memory from the contents of the accumulator. The result is not stored anywhere. The contents of the accumulator and a memory are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Set to “1” when the result of the operation as unsigned data is “0” or larger. Otherwise, cleared to “0”.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	CMP A, #imm	C9$_{16}$, imm	2	2
Direct	CMP A, dd	C5$_{16}$, dd	2	4
Direct indexed X	CMP A, dd, X	D5$_{16}$, dd	2	5
Direct indirect	CMP A, (dd)	D2$_{16}$, dd	2	6
Direct indexed X indirect	CMP A, (dd, X)	C1$_{16}$, dd	2	7
Direct indirect indexed Y	CMP A, (dd), Y	D1$_{16}$, dd	2	8
Direct indirect long	CMPL A, (dd)	C7$_{16}$, dd	2	8
Direct indirect long indexed Y	CMPL A, (dd), Y	D7$_{16}$, dd	2	10
Absolute	CMP A, mmll	CD$_{16}$, ll, mm	3	4
Absolute indexed X	CMP A, mmll, X	DD$_{16}$, ll, mm	3	6
Absolute indexed Y	CMP A, mmll, Y	D9$_{16}$, ll, mm	3	6
Absolute long	CMP A, hhmmll	CF$_{16}$, ll, mm, hh	4	6
Absolute long indexed X	CMP A, hhmmll, X	DF$_{16}$, ll, mm, hh	4	7
Stack pointer relative	CMP A, nn, S	C3$_{16}$, nn	2	5
Stack pointer relative indirect indexed Y	CMP A, (nn, S), Y	D3$_{16}$, nn	2	8

Notes 1: This table applies when using the accumulator A. When using the accumulator B, replace “A” with “B” in the syntax. In this case, “42$_{16}$” is added at the beginning of the machine code, the byte number increases by 1 and the cycle number increases by 2.

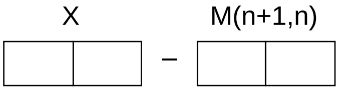
2: When treating a 16-bit data in the immediate addressing mode in the condition of the data length flag = “0”, the byte number increases by 1.

CPX ComPare memory and index register X CPX

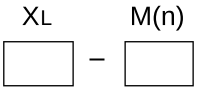
Function : Compare

Operation : X – M

When x = “0”



When x = “1”



Description : Subtracts the contents of a memory from the contents of the index register X. The result is not stored anywhere. The contents of the index register X and a memory are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the index register length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Set to “1” when the result of the operation as unsigned data is “0” or larger. Otherwise, cleared to “0”.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	CPX #imm	$E0_{16}$, imm	2	2
Direct	CPX dd	$E4_{16}$, dd	2	4
Absolute	CPX mmll	EC_{16}, ll, mm	3	4

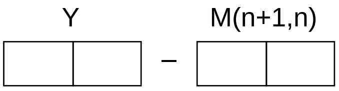
Note: The byte number increases by 1 when treating on 16-bit data in the condition of the index register length flag = “0”.

CPY ComPare memory and index register Y CPY

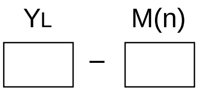
Function : Compare

Operation : Y – M

When x = “0”



When x = “1”



Description : Subtracts the contents of a memory from the contents of the index register Y. The result is not stored anywhere. The contents of the index register Y and a memory contents are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the index register length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Set to “1” when the result of the operation as unsigned data is “0” or larger. Otherwise, cleared to “0”.

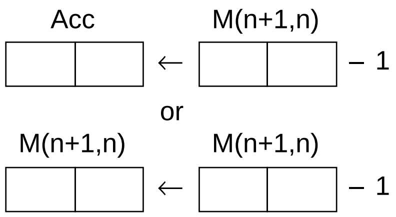
Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	CPY #imm	$C0_{16}$, imm	2	2
Direct	CPY dd	$C4_{16}$, dd	2	4
Absolute	CPY mmll	CC_{16}, ll, mm	3	4

Note: The byte number increases by 1 when treating on 16-bit data in the condition of the index register length flag = “0”.

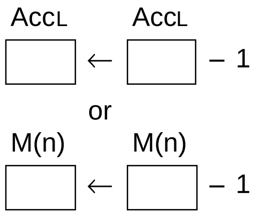
Function : Decrement

Operation : Acc ← Acc – 1 or M ← M – 1

When m = “0”



When m = “1”



Description : Subtracts 1 from the contents of the accumulator or a memory.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Accumulator	DEC A	$1A_{16}$	1	2
Direct	DEC dd	$C6_{16}$, dd	2	7
Direct indexed X	DEC dd, X	$D6_{16}$, dd	2	7
Absolute	DEC mmll	CE_{16}, ll, mm	3	7
Absolute indexed X	DEC mmll, X	DE_{16}, ll, mm	3	8

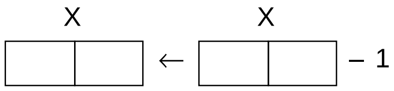
Note: This table applies when using the accumulator A. When using the accumulator B, replace “A” with “B” in the syntax. In this case, “42_{16}” is added at the beginning of the machine code, the byte number increases by 1 and the cycle number increases by 2.

DEX — DEcrement index register X by one — DEX

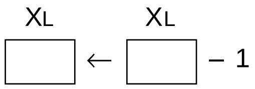
Function : Decrement

Operation : X ← X – 1

When x = “0”



When x = “1”



Description : Subtracts 1 from the contents of the index register X.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the index register length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

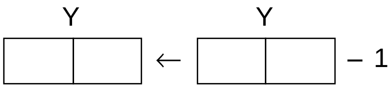
C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	DEX	CA_{16}	1	2

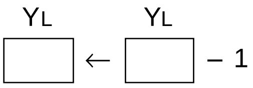
Function : Decrement

Operation : Y ← Y – 1

When x = “0”



When x = “1”



Description : Subtracts 1 from the contents of the index register Y.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the index register length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

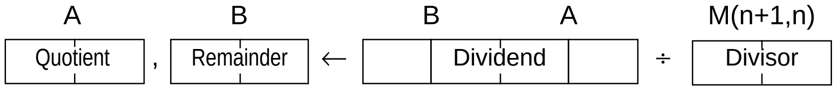
C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	DEY	88_{16}	1	2

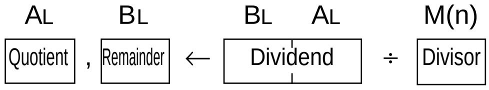
Function : Division (Unsigned)

Operation : A(quotient), B(remainder) ← (B, A) / M

When m = “0”



When m = “1”



Description : When the data length flag is “0”, a 32-bit data stored in the accumulator B, which indicates its higher 16 bits, and A , which indicates its lower 16 bits, is divided by a 16-bit data in a memory. The quotient is placed in the accumulator A, and the remainder is placed in the accumulator B.

When the data length flag is “1”, a 16-bit data is are divided by a 8-bit data in a memory. The lower 8 bits of the accumulator B indicate the higher 8 bits of the 16-bit data, and the lower 8 bits of the accumulator A indicate the lower 8 bits of the16-bit data. The quotient is placed in the lower 8 bits of the accumulator A, and the remainder is placed in the lower 8 bits of the accumulator B.

If an overflow occurs as a result of the operation, the overflow flag is set to “1” and the contents of the accumulators A and B become undefined.

When the divisor is “0”, the zero divide interrupt is generated. In that case, the contents of the program bank register, program counter, and processor status register are saved on the stack and a branch is generated to the address in bank 0 which is specified by the zero divide interrupt vector. The contents of the accumulators A and B are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation quotient is “1”. Otherwise, cleared to “0”.

❊ When an overflow occurs as a result of the operation or the divisor is “0”, N flag is not affected.

V : Cleared to “0”.

❊ Set to “1” when an overflow occurs

❊ Not affected when the divisor is “0”

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

❊ Set to “1” when the divisor is “0”

Z : Set to “1” when the quotient of the operation is “0”. Otherwise, cleared to “0”.

❊ When an overflow occurs as a result of the operation or the divisor is “0”, Z flag is not affected.

C : Cleared to “0”.

❊ Set to “1” when an overflow occurs

❊ Not affected when the divisor is “0”

Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	DIV #imm	89_{16}, 29_{16}, imm	3	21
Direct	DIV dd	89_{16}, 25_{16}, dd	3	23
Direct indexed X	DIV dd, X	89_{16}, 35_{16}, dd	3	24
Direct indirect	DIV (dd)	89_{16}, 32_{16}, dd	3	25
Direct indexed X indirect	DIV (dd, X)	89_{16}, 21_{16}, dd	3	26
Direct indirect indexed Y	DIV (dd), Y	89_{16}, 31_{16}, dd	3	27
Direct indirect long	DIVL (dd)	89_{16}, 27_{16}, dd	3	27
Direct indirect long indexed Y	DIVL (dd), Y	89_{16}, 37_{16}, dd	3	29
Absolute	DIV mmll	89_{16}, $2D_{16}$, ll, mm	4	23
Absolute indexed X	DIV mmll, X	89_{16}, $3D_{16}$, ll ,mm	4	25
Absolute indexed Y	DIV mmll, Y	89_{16}, 39_{16}, ll ,mm	4	25
Absolute long	DIV hhmmll	89_{16}, $2F_{16}$, ll, mm, hh	5	25
Absolute long indexed X	DIV hhmmll, X	89_{16}, $3F_{16}$, ll, mm, hh	5	26
Stack pointer relative	DIV nn, S	89_{16}, 23_{16}, nn	3	24
Stack pointer relative indirect indexed Y	DIV (nn, S), Y	89_{16}, 33_{16}, nn	3	27

Notes 1: When treating a 16-bit data in the immediate addressing mode in the condition of the data length flag = “0”, the byte number increases by 1.

2: The cycle number in this table applies in the case of 16-bit ÷ 8-bit operations. In the case of 32-bit ÷ 16-bit operations, the cycle number increases by 8.

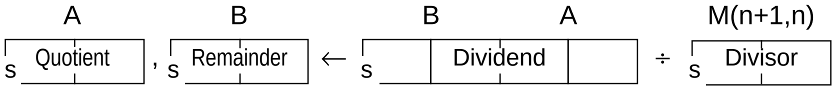
3: The cycle number in this table and Note 2 is the number when the operation is completed normally (no interrupt has been generated). If a zero divide interrupt is generated, its cycle number is the number which is decremented by 3 from in the above table regardless of the operation’s data length.

DIVS DIVide with Sign DIVS

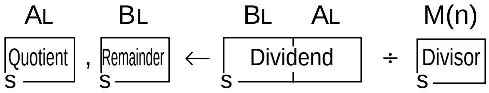
Function : Division (Signed)

Operation : A(quotient), B(remainder) ← (B, A) / M

When m = "0"



When m = "1"



❊ s means a sign bit that is the most significant bit of the data

Description : When the data length flag is "0", a signed 32-bit data stored in the accumulator B, which indicates its higher 16 bits, and A, which indicates its lower 16 bits, is divided by a signed 16-bit data in a memory. The quotient is placed in the accumulator A, and the remainder is placed in the accumulator B. Each of them is a signed 16-bit data.

When the data length flag is "1", a signed 16-bit data is are divided by a signed 8-bit data in a memory. The lower 8 bits of the accumulator B indicate the higher 8 bits of the 16-bit data, and the lower 8 bits of the accumulator A indicate the lower 8 bits of the16-bit data. The quotient is placed in the lower 8 bits of the accumulator A, and the remainder is placed in the lower 8 bits of the accumulator B. Each of them is a signed 8-bit data.

The sign of remainder becomes same as that of dividend.

If an overflow occurs as a result of the operation, in other words, the quotient exceeds the range –32767 to +32767 when m = "0", or –127 to +127 when m = "1"; the operation finishes halfway and the overflow flag is set to "1". Additionally, the contents of the accumulators A and B become undefined.

When the divisor is "0", the zero divide interrupt is generated. In that case, the contents of the program bank register, program counter, and processor status register are saved on the stack and a branch is generated to the address in bank 0 which is specified by the zero divide interrupt vector. The contents of the accumulators A and B are not changed.

Status flags

IPL : Not affected.

N : Set to "1" when bit 15 (or bit 7 when the data length flag is "1") of the operation quotient is "1". Otherwise, cleared to "0".

❊ When an overflow occurs as a result of the operation or the divisor is "0", N flag is not affected.

V : Cleared to "0".

❊ Set to "1" when an overflow occurs

❊ Not affected when the divisor is "0"

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

❊ Set to "1" when the divisor is "0"

Z : Set to “1” when the quotient of the operation is “0”. Otherwise, cleared to “0”.

❊ When an overflow occurs as a result of the operation or the divisor is “0”, Z flag is not affected.

C : Cleared to “0”.

❊ Set to “1” when an overflow occurs

❊ Not affected when the divisor is “0”

Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	DIVS #imm	89_{16}, $A9_{16}$, imm	3	23
Direct	DIVS dd	89_{16}, $A5_{16}$, dd	3	25
Direct indexed X	DIVS dd, X	89_{16}, $B5_{16}$, dd	3	26
Direct indirect	DIVS (dd)	89_{16}, $B2_{16}$, dd	3	27
Direct indexed X indirect	DIVS (dd, X)	89_{16}, $A1_{16}$, dd	3	28
Direct indirect indexed Y	DIVS (dd), Y	89_{16}, $B1_{16}$, dd	3	29
Direct indirect long	DIVSL (dd)	89_{16}, $A7_{16}$, dd	3	29
Direct indirect long indexed Y	DIVSL (dd), Y	89_{16}, $B7_{16}$, dd	3	31
Absolute	DIVS mmll	89_{16}, AD_{16}, ll, mm	4	25
Absolute indexed X	DIVS mmll, X	89_{16}, BD_{16}, ll ,mm	4	27
Absolute indexed Y	DIVS mmll, Y	89_{16}, $B9_{16}$, ll ,mm	4	27
Absolute long	DIVS hhmmll	89_{16}, AF_{16}, ll, mm, hh	5	27
Absolute long indexed X	DIVS hhmmll, X	89_{16}, BF_{16}, ll, mm, hh	5	28
Stack pointer relative	DIVS nn, S	89_{16}, $A3_{16}$, nn	3	26
Stack pointer relative indirect indexed Y	DIVS (nn, S), Y	89_{16}, $B3_{16}$, nn	3	29

Notes 1: When treating a 16-bit data in the immediate addressing mode in the condition of the data length flag = “0”, the byte number increases by 1.

2: The cycle number in this table applies in the case of 16-bit ÷ 8-bit operations. In the case of 32-bit ÷ 16-bit operations, the cycles number increases by 8.

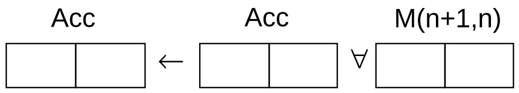
3: The cycle number in this table and Note 2 is the number when the operation completes normally (no interrupt has been generated). If a zero divide interrupt is generated, its cycle number is the number which is decremented by 5 from in the above table regardless of the operation’s data length.

EOR Exclusive OR memory with accumulator EOR

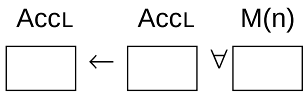
Function : Logical EXCLUSIVE OR

Operation : Acc ← Acc ∀ M

When m = “0”



When m = “1”



Description : Performs the logical EXCLUSIVE OR between the contents of the accumulator and the contents of a memory by each bit, and places the result in the accumulator.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	EOR A, #imm	49_{16}, imm	2	2
Direct	EOR A, dd	45_{16}, dd	2	4
Direct indexed X	EOR A, dd, X	55_{16}, dd	2	5
Direct indirect	EOR A, (dd)	52_{16}, dd	2	6
Direct indexed X indirect	EOR A, (dd, X)	41_{16}, dd	2	7
Direct indirect indexed Y	EOR A, (dd), Y	51_{16}, dd	2	8
Direct indirect long	EORL A, (dd)	47_{16}, dd	2	8
Direct indirect long indexed Y	EORL A, (dd), Y	57_{16}, dd	2	10
Absolute	EOR A, mmll	$4D_{16}$, ll, mm	3	4
Absolute indexed X	EOR A, mmll, X	$5D_{16}$, ll, mm	3	6
Absolute indexed Y	EOR A, mmll, Y	59_{16}, ll, mm	3	6
Absolute long	EOR A, hhmmll	$4F_{16}$, ll, mm, hh	4	6
Absolute long indexed X	EOR A, hhmmll, X	$5F_{16}$, ll, mm, hh	4	7
Stack pointer relative	EOR A, nn, S	43_{16}, nn	2	5
Stack pointer relative indirect indexed Y	EOR A, (nn, S), Y	53_{16}, nn	2	8

Notes 1: This table applies when using the accumulator A. When using the accumulator B, replace “A” with “B” in the syntax. In this case, “42_{16}” is added at the beginning of the machine code, the byte number increases by 1 and the cycle number increases by 2.

2: When treating a 16-bit data in the immediate addressing mode in the condition of the data length flag = “0”, the byte number increases by 1.

Function : Extension sign

Operation : Acc$_H$ ← 00_{16} or FF_{16}

When bit 7 of Acc$_L$ = “0”

Acc$_H$ ← 00_{16}

Acc$_H$	Acc$_L$		Acc$_H$	Acc$_L$
00000000	0XXXXXXX	←	?	0XXXXXXX

When bit 7 of Acc$_L$ = “1”

Acc$_H$ ← FF_{16}

Acc$_H$	Acc$_L$		Acc$_H$	Acc$_L$
11111111	1XXXXXXX	←	?	1XXXXXXX

❊ The high-order byte of Acc changes regardless of the data length flag

Description : This instruction is used to extend a signed 8-bit data stored in the low-order byte of the accumulator to a 16-bit data.

When bit 7 of the accumulator is “0”, bits 8 to 15 become “0”. When bit 7 of the accumulator is “1”, bits 8 to 15 become “1”.

With this instruction, the high-order byte of accumulator changes regardless of the data length flag. However, the content of the data length flag is unchanged.

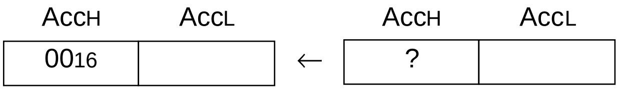
Status flags

IPL : Not affected.
N : Set to “1”, when bit 15 of the operation result is “1”. Otherwise, cleared to “0”.
V : Not affected.
m : Not affected.
x : Not affected.
D : Not affected.
I : Not affected.
Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.
C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Accumulator	EXTS A	89_{16}, $8B_{16}$	2	4
Accumulator	EXTS B	42_{16}, $8B_{16}$	2	4

Function : Extension zero

Operation : $\text{Acc}_H \leftarrow 00_{16}$



❊ The high-order byte of Acc changes regardless of the m flag

Description : This instruction is used to extend a 8-bit data stored in the low-order byte of the accumulator to a 16-bit data.

Bits 8 to 15 of the accumulator become “0”.

With this instruction, the high-order byte of accumulator changes regardless of the data length flag. However, the content of the data length flag is unchanged.

Status flags

IPL	:	Not affected.
N	:	Set to “0”.
V	:	Not affected.
m	:	Not affected.
x	:	Not affected.
D	:	Not affected.
I	:	Not affected.
Z	:	Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.
C	:	Not affected.

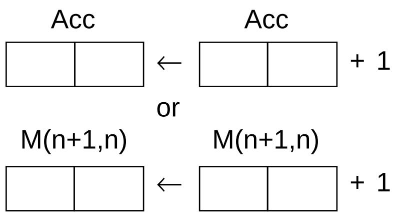
Addressing mode	Syntax	Machine code	Bytes	Cycles
Accumulator	EXTZ A	89_{16}, AB_{16}	2	4
Accumulator	EXTZ B	42_{16}, AB_{16}	2	4

INCrement by one

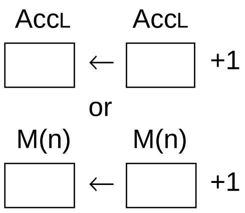
Function : Increment

Operation : Acc ← Acc + 1 or M ← M + 1

When m = “0”



When m = “1”



Description : Adds 1 to the contents of the accumulator or a memory.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Accumulator	INC A	$3A_{16}$	1	2
Direct	INC dd	$E6_{16}$, dd	2	7
Direct indexed X	INC dd, X	$F6_{16}$, dd	2	7
Absolute	INC mmll	EE_{16}, ll, mm	3	7
Absolute indexed X	INC mmll, X	FE_{16}, ll, mm	3	8

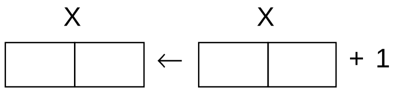
Note: This table applies when using the accumulator A. When using the accumulator B, replace “A” with “B” in the syntax. In this case, “42_{16}” is added at the beginning of the machine code, the byte number increases by 1 and the cycle number increases by 2.

INX INcrement index register X by one INX

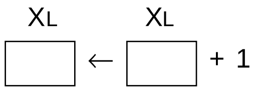
Function : Increment

Operation : X ← X + 1

When x = “0”



When x = “1”



Description : Adds 1 to the contents of the index register X.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the index register length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

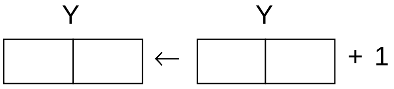
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	INX	$E8_{16}$	1	2

INY INcrement index register Y by one INY

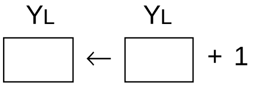
Function : Increment

Operation : Y ← Y + 1

When x = “0”



When x = “1”



Description : Adds 1 to the contents of the index register Y.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the index register length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	INY	$C8_{16}$	1	2

JMP JuMP JMP

Function : Jump always

Operation : [PG], PC ← specified address (absolute or indirect)

When the addressing mode is ...

absolute addressing mode,

PC ← ADDR

absolute long addressing mode,

PC ← ADDR
PG ← BANK

absolute indirect addressing mode,

PC ← M(ADDR+1, ADDR)

absolute indirect long addressing mode,

PC ← M(ADDR+1, ADDR)
PG ← M(ADDR+2)

absolute indexed X indirect addressing mode,

PC ← M(ADDR+X+1, ADDR+X)

❊ ADDR indicates the low-order 16 bits of a 24-bit address which is specified by the 2nd and 3rd bytes of the instruction.

❊ BANK indicates the high-order 8 bits of a 24-bit address which is specified by the 4th byte of the instruction.

Description : The JMP instruction causes a jump to the address specified by each addressing mode. When this instruction is used in addressing modes other than absolute long, the contents of the program bank register is incremented by 1 and the branch destination becomes the next bank if the last byte of the instruction is at the highest address ($XXFFFF_{16}$) of a bank or if the instruction crosses banks.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Absolute	JMP mmll	$4C_{16}$, ll, mm	3	2
Absolute long	JMPL hhmmll	$5C_{16}$, ll, mm, hh	4	4
Absolute indirect	JMP (mmll)	$6C_{16}$, ll, mm	3	4
Absolute indirect long	JMPL (mmll)	DC_{16}, ll, mm	3	6
Absolute indexed X indirect	JMP (mmll, X)	$7C_{16}$, ll, mm	3	6

Function : Jump to subroutine

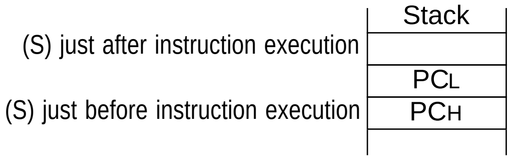
Operation : Stack ← [PG], PC

[PG], PC ← specified address (absolute or indirect)

When the addressing mode is ...

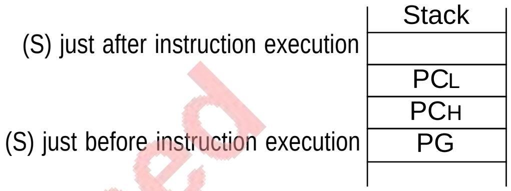
absolute addressing mode,

PC ← PC + 3
M(S,S–1) ← PC
S ← S – 2
PC ← ADDR


Stack
(S) just after instruction execution
PCL
(S) just before instruction execution
PCH

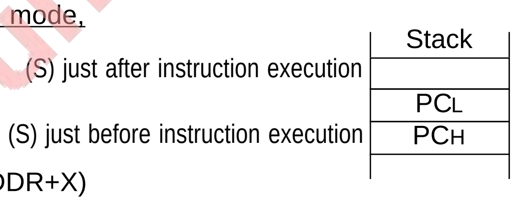
absolute long addressing mode,

PC ← PC + 4
M(S to S–2) ← PG, PC
S ← S – 3
PC ← ADDR
PG ← BANK


Stack
(S) just after instruction execution
PCL
PCH
(S) just before instruction execution
PG

absolute indexed X indirect addressing mode,

PC ← PC + 3
M(S,S–1) ← PC
S ← S – 2
PC ← M(ADDR+X+1,ADDR+X)


Stack
(S) just after instruction execution
PCL
(S) just before instruction execution
PCH

❊ ADDR indicates the low-order 16 bits of a 24-bit address which is specified by the 2nd and 3rd bytes of the instruction.

❊ BANK indicates the high-order 8 bits of a 24-bit address which is specified by the 4th byte of the instruction.

Description : The contents of the program counter (or the program bank register and the program counter in absolute long addressing mode) are first saved on the stack. After that, a jump is caused to the address specified by each addressing mode.

When this instruction is used in addressing modes other than absolute long, the content of the program bank register is incremented by 1 and the branch destination becomes the next bank if the last byte of the instruction is at the highest address ($XXFFFF_{16}$) of a bank or if the instruction crosses banks.

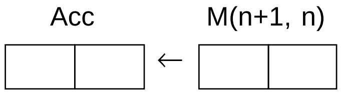
Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Absolute	JSR mmll	20_{16}, ll, mm	3	6
Absolute long	JSRL hhmmll	22_{16}, ll, mm, hh	4	8
Absolute indexed X indirect	JSR (mmll, X)	FC_{16}, ll, mm	3	8

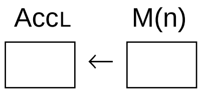
Function : Load

Operation : Acc ← M

When m = “0”



When m = “1”



Description : Loads the contents of a memory into the accumulator.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	LDA A, #imm	A9$_{16}$, imm	2	2
Direct	LDA A, dd	A5$_{16}$, dd	2	4
Direct indexed X	LDA A, dd, X	B5$_{16}$, dd	2	5
Direct indirect	LDA A, (dd)	B2$_{16}$, dd	2	6
Direct indexed X indirect	LDA A, (dd, X)	A1$_{16}$, dd	2	7
Direct indirect indexed Y	LDA A, (dd), Y	B1$_{16}$, dd	2	8
Direct indirect long	LDAL A, (dd)	A7$_{16}$, dd	2	8
Direct indirect long indexed Y	LDAL A, (dd), Y	B7$_{16}$, dd	2	10
Absolute	LDA A, mmll	AD$_{16}$, ll, mm	3	4
Absolute indexed X	LDA A, mmll, X	BD$_{16}$, ll, mm	3	6
Absolute indexed Y	LDA A, mmll, Y	B9$_{16}$, ll, mm	3	6
Absolute long	LDA A, hhmmll	AF$_{16}$, ll, mm, hh	4	6
Absolute long indexed X	LDA A, hhmmll, X	BF$_{16}$, ll, mm, hh	4	7
Stack pointer relative	LDA A, nn, S	A3$_{16}$, nn	2	5
Stack pointer relative indirect indexed Y	LDA A, (nn, S), Y	B3$_{16}$, nn	2	8

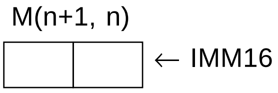
Notes 1: This table applies when using the accumulator A. When using the accumulator B, replace “A” with “B” in the syntax. In this case, “42$_{16}$” is added at the beginning of the machine code, the byte number increases by 1 and the cycle number increases by 2.

2: When treating a 16-bit data in the immediate addressing mode in the condition of the data length flag = “0”, the byte number increases by 1.

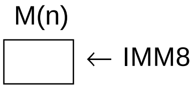
Function : Load

Operation : M ← IMM

When m = “0”



When m = “1”



Description : Loads an immediate value into a memory.

Status flags : Not affected.

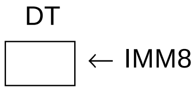
Addressing mode	Syntax	Machine code	Bytes	Cycles
Direct	LDM #imm, dd	64_{16}, dd, imm	3	4
Direct indexed X	LDM #imm, dd, X	74_{16}, dd, imm	3	5
Absolute	LDM #imm, mmll	$9C_{16}$, ll, mm, imm	4	5
Absolute indexed X	LDM #imm, mmll, X	$9E_{16}$, ll, mm, imm	4	6

Note: When treating a 16-bit data in the condition of the data length flag = “0”, the byte number increases by 1.

LDT LoaD immediate to daTa bank register LDT

Function : Load

Operation : DT ← IMM8



Description : Loads an immediate value into the data bank register.

Status flags : Not affected.

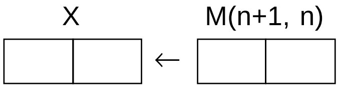
Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	LDT #imm	89_{16}, $C2_{16}$, imm	3	5

LoaD index register X from memory

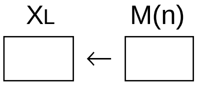
Function : Load

Operation : X ← M

When x = “0”



When x = “1”



Description : Loads the contents of a memory into the index register X.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the index register length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	LDX #imm	$A2_{16}$, imm	2	2
Direct	LDX dd	$A6_{16}$, dd	2	4
Direct indexed Y	LDX dd, Y	$B6_{16}$, dd	2	5
Absolute	LDX mmll	AE_{16}, ll, mm	3	4
Absolute indexed Y	LDX mmll, Y	BE_{16}, ll, mm	3	6

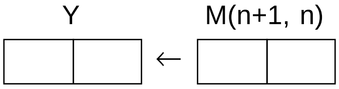
Note: When treating a 16-bit data in the immediate addressing mode in the condition of the index register length flag = “0”, the byte number increases by 1.

LoaD index register Y from memory

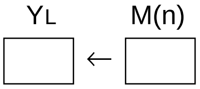
Function : Load

Operation : Y ← M

When x = “0”



When x = “1”



Description : Loads the contents of a memory into the index register Y.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the index register length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	LDY #imm	$A0_{16}$, imm	2	2
Direct	LDY dd	$A4_{16}$, dd	2	4
Direct indexed X	LDY dd, X	$B4_{16}$, dd	2	5
Absolute	LDY mmll	AC_{16}, ll, mm	3	4
Absolute indexed X	LDY mmll, X	BC_{16}, ll, mm	3	6

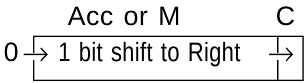
Note: When treating a 16-bit data in the immediate addressing mode in the condition of the index register length flag = “0”, the byte number increases by 1.

LSR

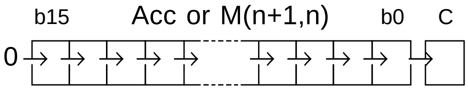
Logical Shift Right

Function : Logical shift right

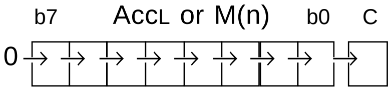
Operation :



When m = “0”



When m = “1”



Description : Shifts all bits of the accumulator or a memory to the one bit right. Its bit 15 (or bit 7 when the data length flag is “1”) of the accumulator or a memory is loaded with “0”.
The carry flag is loaded from bit 0 of the data before the shift.

Status flags

IPL : Not affected.

N : Cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Set to “1” when bit 0 of the accumlator or a memory before the operation is “1”. Otherwise, cleared to “0”.

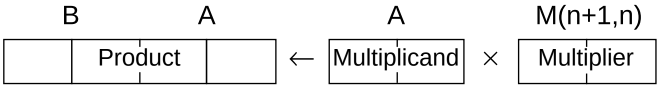
Addressing mode	Syntax	Machine code	Bytes	Cycles
Accumulator	LSR A	4A$_{16}$	1	2
Direct	LSR dd	46$_{16}$, dd	2	7
Direct indexed X	LSR dd, X	56$_{16}$, dd	2	7
Absolute	LSR mmll	4E$_{16}$, ll, mm	3	7
Absolute indexed X	LSR mmll, X	5E$_{16}$, ll, mm	3	8

Note: This table applies when using the accumulator A. When using the accumulator B, replace “A” with “B” in the syntax. In this case, “42$_{16}$” is added at the beginning of the machine code, the byte number increases by 1 and the cycle number increases by 2.

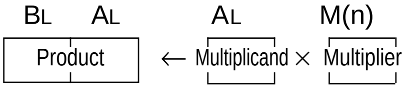
Function : Multiplication (Unsigned)

Operation : B, A ← A × M

When m = "0"



When m = "1"



Description : When the data length flag is "0", the contents of the accumulator A are multiplied by the contents of a memory. Multiplication is performed as 16-bit × 16-bit, and the result becomes a 32-bit data of which higher is placed in the accumulator B and lower is placed in the accumulator A.

When the data length flag is "1", the low-order 8 bits of the accumulator A are multiplied by the contents of a memory. Multiplication is performed as 8-bit × 8-bit, and the result becomes a 16-bit data of which high-order 8 bits are placed in the accumulator B's low-order 8 bits and lower 8 bits are placed in the accumulator A's low-order 8 bits.

Status flags

IPL : Not affected.

N : Set to "1" when bit 31 of the operation result, the accumulator B's bit 15, (or bit 15 of the operation result, the accumulator B's bit 7, when the data length flag is "1") of the operation result is "1". Otherwise, cleared to "0".

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to "1" when the result of the operation is "0". Otherwise, cleared to "0".

C : Cleared to "0".

Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	MPY #imm	89_{16}, 09_{16}, imm	3	8
Direct	MPY dd	89_{16}, 05_{16}, dd	3	10
Direct indexed X	MPY dd, X	89_{16}, 15_{16}, dd	3	11
Direct indirect	MPY (dd)	89_{16}, 12_{16}, dd	3	12
Direct indexed X indirect	MPY (dd, X)	89_{16}, 01_{16}, dd	3	13
Direct indirect indexed Y	MPY (dd), Y	89_{16}, 11_{16}, dd	3	14
Direct indirect long	MPYL (dd)	89_{16}, 07_{16}, dd	3	14
Direct indirect long indexed Y	MPYL (dd), Y	89_{16}, 17_{16}, dd	3	16
Absolute	MPY mmll	89_{16}, $0D_{16}$, ll, mm	4	10
Absolute indexed X	MPY mmll, X	89_{16}, $1D_{16}$, ll, mm	4	12
Absolute indexed Y	MPY mmll, Y	89_{16}, 19_{16}, ll, mm	4	12
Absolute long	MPY hhmmll	89_{16}, $0F_{16}$, ll, mm, hh	5	12
Absolute long indexed X	MPY hhmmll, X	89_{16}, $1F_{16}$, ll, mm, hh	5	13
Stack pointer relative	MPY nn, S	89_{16}, 03_{16}, nn	3	11
Stack pointer relative indirect indexed Y	MPY (nn, S), Y	89_{16}, 13_{16}, nn	3	14

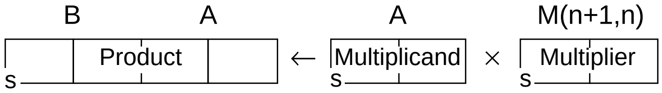
Notes 1: When treating a 16-bit data in the immediate addressing mode in the condition of the data length flag = “0”, the byte number increases by 1.

2: The cycle number in this table applies in the case of 8-bit × 8-bit operations. In the case of 16-bit × 16-bit operations, the cycle number increases by 4.

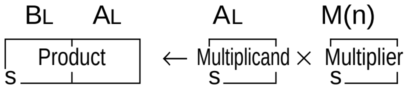
Function : Multiplication (Signed)

Operation : B, A ← A × M

When m = “0”



When m = “1”



❊ s means a sign bit that is the most significant bit of the data.

Description : When the data length flag is “0”, the contents of the accumulator A are multiplied by the contents of a memory as a signed data. Multiplication is performed as 16-bit × 16-bit, and the result becomes a 32-bit data of which higher is placed in the accumulator B and lower is placed in the accumulator A. Then, the accumulator B’s bit 15 is a sign bit.

When the data length flag is “1”, the low-order contents of the accumulator A are multiplied by the contents of a memory as a signed data. Multiplication is performed as 8-bit × 8-bit, and the result becomes a 16-bit data of which high-order 8 bits are placed in the accumulator B’s low-order 8 bits and lower 8 bits are placed in the accumulator A’s low-order 8 bits. Then, the accumulator B’s bit 7 is a sign bit.

Status flags

IPL : Not affected.

N : Set to “1” when bit 31 of the operation result, the accumulator B’s bit 15, (or bit 15 of the operation result, the accumulator B’s bit 7, when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Cleared to “0”.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	MPYS #imm	89_{16}, 89_{16}, imm	3	8
Direct	MPYS dd	89_{16}, 85_{16}, dd	3	10
Direct indexed X	MPYS dd, X	89_{16}, 95_{16}, dd	3	11
Direct indirect	MPYS (dd)	89_{16}, 92_{16}, dd	3	12
Direct indexed X indirect	MPYS (dd, X)	89_{16}, 81_{16}, dd	3	13
Direct indirect indexed Y	MPYS (dd), Y	89_{16}, 91_{16}, dd	3	14
Direct indirect long	MPYSL (dd)	89_{16}, 87_{16}, dd	3	14
Direct indirect long indexed Y	MPYSL (dd), Y	89_{16}, 97_{16}, dd	3	16
Absolute	MPYS mmll	89_{16}, $8D_{16}$, ll, mm	4	10
Absolute indexed X	MPYS mmll, X	89_{16}, $9D_{16}$, ll, mm	4	12
Absolute indexed Y	MPYS mmll, Y	89_{16}, 99_{16}, ll, mm	4	12
Absolute long	MPYS hhmmll	89_{16}, $8F_{16}$, ll, mm, hh	5	12
Absolute long indexed X	MPYS hhmmll, X	89_{16}, $9F_{16}$, ll, mm, hh	5	13
Stack pointer relative	MPYS nn, S	89_{16}, 83_{16}, nn	3	11
Stack pointer relative indirect indexed Y	MPYS (nn, S), Y	89_{16}, 93_{16}, nn	3	14

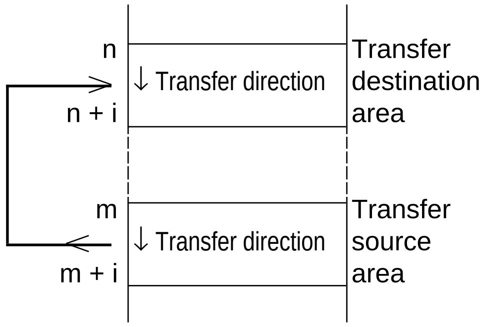
Notes 1: When treating a 16-bit data in the immediate addressing mode in the condition of the data length flag = "0", the byte number increases by 1.

2: The cycle number in this table applies in the case of 8-bit × 8-bit operations. In the case of 16-bit × 16-bit operations, the cycle number increases by 4.

MVN MoVe Negative MVN

Function : Move

Operation : M(n to n + i) ← M(m to m + i)



A = 0 ?

When A = “0”

Instruction execution complete

When A ≠ “0”

Repeat operation

M(DTd: Y) ← M(DTs: X)

X ← X + 2

Y ← Y + 2

A ← A – 2

❊ DTd indicates the transfer destination bank which is specified by the 2nd byte of the instruction.

❊ DTs indicates the transfer source bank which is specified by the 3rd byte of the instruction.

❊ Values set in register before transfer

A: Transfer byte number

When m = “0” The value 0 to 65535 can be set

When m = “1” The value 0 to 255 can be set

X: Transfer source area beginning (lowermost) address

When x = “0” The value 0 to 65535 can be set

When x = “1” The value 0 to 255 can be set (Note)

Y: Transfer destination area beginning (lowermost) address

When x = “0” The value 0 to 65535 can be set

When x = “1” The value 0 to 255 can be set (Note)

❊ Contents of register after transfer

A: $FFFF_{16}$

X: Transfer source area end (highermost) address + 1

Y: Transfer destination area end (highermost) address + 1

DT: Bank number of transfer destination

❊ Transfer is normally performed by two bytes. Accordingly, in the case of a 16-bit bus, the transfer time is shortened when transfer start addresses are even than when they are odd.

Note: This instruction is recommended to use in the condition of the x=“0”. In the condition of the x = “1”, data in the area between $XX00_{16}$ and $XXFF_{16}$ can be only used because the higher bytes of index registers X and Y are not changed.

x: Index register length flag

Description : Normally, a block of data is transferred from higher addresses to lower addresses. The transfer is performed in the ascending address order of the block being transferred. The destination bank is specified by the instruction's second byte, and the address within its bank is specified by the contents of the index register Y. The source bank is specified by the instruction's third byte, and the address within its bank is specified by the contents of the index register X. The accumulator A is loaded with the bytes number of the data to be transferred. As each 1 byte of data is transferred, the index registers X and Y are incremented, so that the index register X will become a value equal to 1 larger than the source address of the last byte transferred and the index register Y will become a value equal to 1 larger than the destination address of the last byte received. The data bank register will become the destination bank number, and the accumulator A will become $FFFF_{16}$.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Block transfer	MVN hh_1, hh_2	54_{16}, hh_1, hh_2	3	5+(i/2)×7

Note: The cycle number in this table applies when the number of bytes transferred, i, is an even number. When i is an odd number, the cycle number is obtained as follows:

$5 + (i \div 2) \times 7 + 6$.

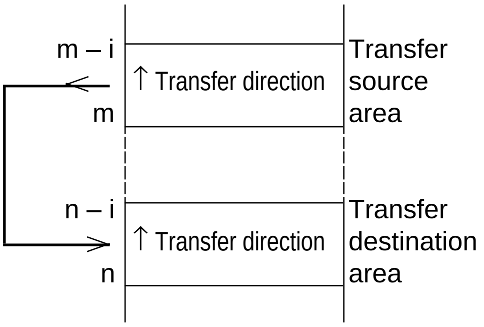
Note that (i ÷ 2) expresses the integer part of the result of dividing i by 2.

MVP MoVe Positive MVP

Function : Move

Operation : M(n – i to n) ← M(m – i to m)

```
A = 0 ?
When A = "0"
    Instruction execution complete
When A ≠ "0"
    Repeat operation
X ← X – 1
Y ← Y – 1
M(DTd: Y) ← M(DTs: X)
X ← X – 1
Y ← Y – 1
A ← A – 2
```



❊ DTd indicates the transfer destination bank which is specified by the 2nd byte of the instruction.

❊ DTs indicates the transfer source bank which is specified by the 3rd byte of the instruction.

❊ Values set in register before transfer

A: Transfer byte number
When m = "0" The value 0 to 65535 can be set
When m = "1" The value 0 to 255 can be set

X: Transfer source area beginning (highermost) address
When x = "0" The value 0 to 65535 can be set
When x = "1" The value 0 to 255 can be set

Y: Transfer destination area beginning (highermost) address
When x = "0" The value 0 to 65535 can be set
When x = "1" The value 0 to 255 can be set

❊ Contents of register after transfer

A: $FFFF_{16}$
X: Transfer source area end (lowermost) address – 1
Y: Transfer destination area end (lowermost) address – 1
DT: Bank number of transfer destination

❊ Transfer is normally performed by two bytes. Accordingly, in the case of a 16-bit bus, the transfer time is shortened when transfer start addresses are odd than when they are even.

Note: This instruction is recommended to use in the condition of the x="0". In the condition of the x = "1", data in the area between $XX00_{16}$ and $XXFF_{16}$ can be only used because the higher bytes of index registers X and Y are not changed.

x: Index register length flag

MVP MoVe Positive MVP

Description : Normally, a block of data is transferred from lower addresses to higher addresses. The transfer is performed in the descending address order of the block being transferred. The destination bank is specified by the instruction's second byte, and the address within its bank is specified by the contents of the index register Y. The source bank is specified by the instruction's third byte, and the address within its bank is specified by the contents of the index register X. The accumulator A is loaded with the byte number of the data to be transferred. As each 1 byte of data is transferred, the index registers X and Y are decremented, so that the index register X will become a value equal to 1 smaller than the source address of the last byte transferred and the index register Y will become a value equal to 1 smaller than the destination address of the last byte received. The data bank register will become the destination bank number, and the accumulator A will become $FFFF_{16}$.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Block transfer	MVP hh_1, hh_2	44_{16}, hh_1, hh_2	3	9+(i/2)×7

Note: The cycle number in this table applies when the number of bytes transferred, i, is an even number. When i is an odd number, the cycle number is obtained as follows:

$9 + (i \div 2) \times 7 + 8$.

Note that $(i \div 2)$ expresses the integer part of the result of dividing i by 2.

No OPeration

Function : No operation

Operation : PC ← PC + 1

❊ PG also changes depending on the result of the above operation on PC.

If a carry occurs in PC: PG ← PG + 1

Description : This instruction only makes the program counter increment by 1 and nothing else.

Status flags : Not affected.

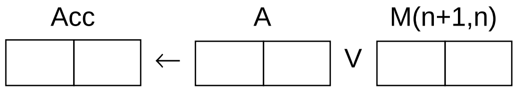
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	NOP	EA_{16}	1	2

ORA OR memory with Accumulator ORA

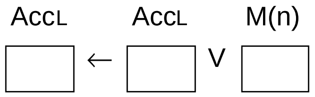
Function : Logical OR

Operation : Acc ← Acc V M

When m = “0”



When m = “1”



Description : Performs the logical OR between the contents of the accumulator and the contents of a memory, and places the result in the accumulator.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	ORA A, #imm	09_{16}, imm	2	2
Direct	ORA A, dd	05_{16}, dd	2	4
Direct indexed X	ORA A, dd, X	15_{16}, dd	2	5
Direct indirect	ORA A, (dd)	12_{16}, dd	2	6
Direct indexed X indirect	ORA A, (dd, X)	01_{16}, dd	2	7
Direct indirect indexed Y	ORA A, (dd), Y	11_{16}, dd	2	8
Direct indirect long	ORAL A, (dd)	07_{16}, dd	2	8
Direct indirect long indexed Y	ORAL A, (dd), Y	17_{16}, dd	2	10
Absolute	ORA A, mmll	$0D_{16}$, ll, mm	3	4
Absolute indexed X	ORA A, mmll, X	$1D_{16}$, ll, mm	3	6
Absolute indexed Y	ORA A, mmll, Y	19_{16}, ll, mm	3	6
Absolute long	ORA A, hhmmll	$0F_{16}$, ll, mm, hh	4	6
Absolute long indexed X	ORA A, hhmmll, X	$1F_{16}$, ll, mm, hh	4	7
Stack pointer relative	ORA A, nn, S	03_{16}, nn	2	5
Stack pointer relative indirect indexed Y	ORA A, (nn, S), Y	13_{16}, nn	2	8

Notes 1: This table applies when using the accumulator A. When using the accumulator B, replace “A” with “B” in the syntax. In this case, “42_{16}” is added at the beginning of the machine code, the byte number increases by 1 and the cycle number increases by 2.

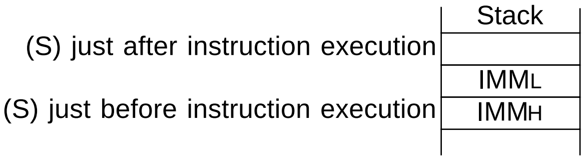
2: When treating a 16-bit data in the immediate addressing mode in the condition of the data length flag = “0”, the byte number increases by 1.

Push Effective Address

Function : Stack manipulation (Push)

Operation : Stack ← IMM16

M(S,S – 1) ← IMM16
S ← S – 2


Stack
(S) just after instruction execution
IMM$_L$
(S) just before instruction execution
IMM$_H$

❊ IMM16 is an immediate value. IMM$_H$ indicates its high-order byte and IMM$_L$ indicates its low-order byte.

Description : The instruction's third and second bytes are saved on the stack in this order.

Status flags : Not affected.

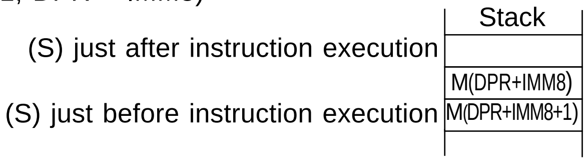
Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PEA #imm$_H$ imm$_L$	F4$_{16}$, imm$_L$, imm$_H$	3	5

PEI Push Effective Indirect address PEI

Function : Stack manipulation (Push)

Operation : Stack ← M(DPR + IMM8 + 1, DPR + IMM8)

M(S, S – 1) ← M(DPR + IMM8 + 1, DPR + IMM8)
S ← S – 2


Stack
(S) just after instruction execution
M(DPR+IMM8)
(S) just before instruction execution
M(DPR+IMM8+1)

❊ IMM8 is an 8-bit immediate value and is used as an displacement from DPR.

Description : Saves the contents of the consecutive 2 bytes in the direct page specified by the sum of the contents of the direct page register and the instruction's second byte on the stack in the order of higher address first and lower address second.

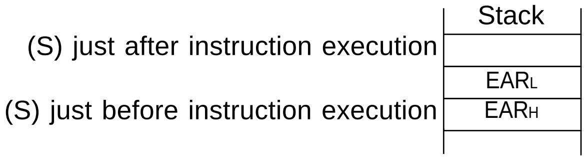
Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PEI #imm	$D4_{16}$, imm	2	6

Push Effective program counter Relative address

Function : Stack manipulation (Push)

Operation : Stack ← PC + IMM16


Stack
(S) just after instruction execution
EAR_L
(S) just before instruction execution
EAR_H

EAR ← PC + IMM16
M(S, S – 1) ← EAR
S ← S – 2

❊ IMM16 is a 16-bit immediate value.

❊ EAR is an execution address which is the result of adding PC to IMM16. EAR_H indicates its high-order byte and EAR_L indicates its low-order byte.

Description : Saves the result of adding a 16-bit data consisting of the instruction's third byte as the higher byte and the instruction's second byte as the lower byte to the contents of the program counter on the stack in the order of the result's higher byte first and lower byte second.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PER #imm_H imm_L	62_{16}, imm_L, imm_H	3	5

PHA PusH accumulator A on stack PHA

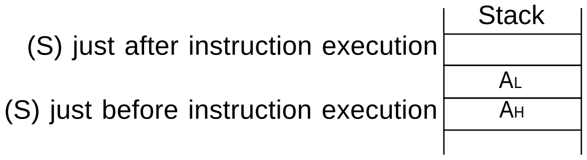
Function : Stack manipulation (Push)

Operation : Stack ← A

When m = “0”

M(S, S – 1) ← A

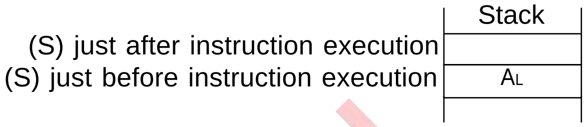
S ← S – 2


Stack
(S) just after instruction execution
A_L
(S) just before instruction execution
A_H

When m = “1”

M(S) ← A_L

S ← S – 1


Stack
(S) just after instruction execution
(S) just before instruction execution
A_L

Description : Saves the contents of the accumulator A to the address specified by the stack pointer. When the data length flag is “0”, the accumulator A’s higher byte is saved on the stack first and then the lower byte. When the data length flag is “1”, only the accumulator A’s lower byte is saved on the stack.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PHA	48_{16}	1	4

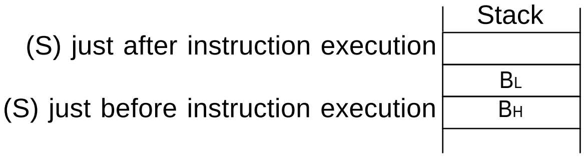
Function : Stack manipulation (Push)

Operation : Stack ← B

When m = “0”

M(S, S – 1) ← B

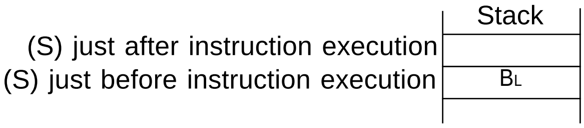
S ← S – 2


Stack
(S) just after instruction execution
B_L
(S) just before instruction execution
B_H

When m = “1”

M(S) ← B_L

S ← S – 1


Stack
(S) just after instruction execution
(S) just before instruction execution
B_L

Description : Saves the contents of the accumulator B to the address specified by the stack pointer. When the data length flag is “0”, the accumulator B’s higher byte is saved on the stack first and then the lower byte. When the data length flag is “1”, only the accumulator B’s lower byte is saved on the stack.

Status flags : Not affected.

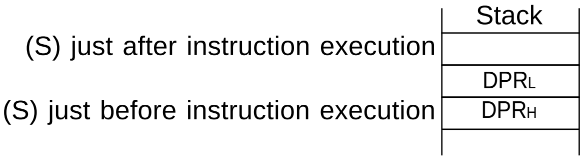
Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PHB	42_{16}, 48_{16}	2	6

PHD — PusH Direct page register on stack — PHD

Function : Stack manipulation (Push)

Operation : Stack ← DPR

M(S, S – 1) ← DPR
S ← S – 2


Stack
(S) just after instruction execution
DPRL
(S) just before instruction execution
DPRH

Description : Saves the contents of the direct page register to the address specified by the stack pointer in the order of higher byte first and then lower byte.

Status flags : Not affected.

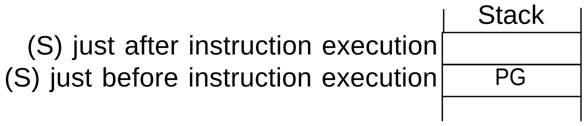
Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PHD	$0B_{16}$	1	4

PHG — PusH proGram bank register on stack — PHG

Function : Stack manipulation (Push)

Operation : Stack ← PG

M(S) ← PG
S ← S – 1


Stack
(S) just after instruction execution
(S) just before instruction execution
PG

Description : Saves the contents of the program bank register to the address specified by the stack pointer.

Status flags : Not affected.

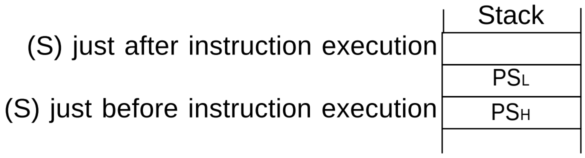
Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PHG	$4B_{16}$	1	3

PHP   PusH Processor status on stack   PHP

Function : Stack manipulation (Push)

Operation : Stack ← PS

M(S, S – 1) ← PS
S ← S – 2


Stack
(S) just after instruction execution
PSL
(S) just before instruction execution
PSH

Description : Saves the contents of the processor status register to the address specified by the stack pointer in the order of higher byte and then lower byte.

Status flags : Not affected.

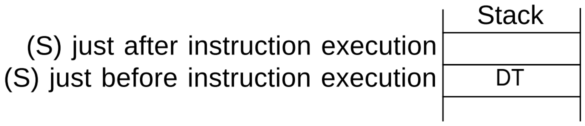
Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PHP	08_{16}	1	4

PHT — PusH daTa bank register on stack — PHT

Function : Stack manipulation (Push)

Operation : Stack ← DT

M(S) ← DT
S ← S – 1


Stack
(S) just after instruction execution
(S) just before instruction execution
DT

Description : Saves the contents of the data bank register to the address specified by the stack pointer.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PHT	$8B_{16}$	1	3

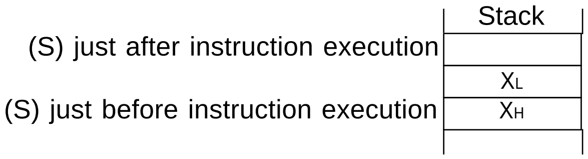
Function : Stack manipulation (Push)

Operation : Stack ← X

When x = “0”

M(S, S − 1) ← X

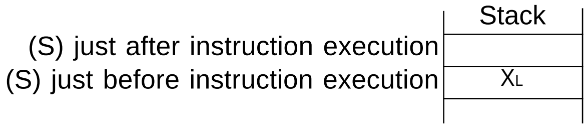
S ← S − 2


Stack
(S) just after instruction execution
XL
(S) just before instruction execution
XH

When x = “1”

M(S) ← X_L

S ← S − 1


Stack
(S) just after instruction execution
(S) just before instruction execution
XL

Description : Saves the contents of the index register X to the address specified by the stack pointer. When the index register length flag is “0”, the contents are saved in the order of higher byte and then lower byte. When the index register length flag is “1”, only the lower byte is saved on the stack.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PHX	DA_{16}	1	4

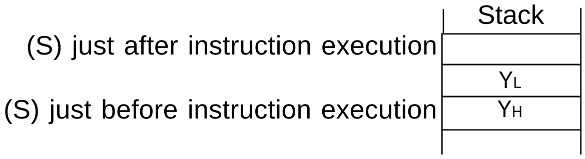
Function : Stack manipulation (Push)

Operation : Stack ← Y

When x = “0”

M(S, S – 1) ← Y

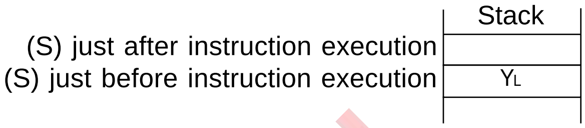
S ← S – 2


Stack
(S) just after instruction execution
Y_L
(S) just before instruction execution
Y_H

When x = “1”

M(S) ← Y_L

S ← S – 1


Stack
(S) just after instruction execution
(S) just before instruction execution
Y_L

Description : Saves the contents of the index register Y to the address specified by the stack pointer. When the index register length flag is “0”, the contents are saved in the order of higher byte and then lower byte. When the index register length flag is “1”, only the lower byte is saved on the stack.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PHY	$5A_{16}$	1	4

PLA — PuLl accumulator A from stack — PLA

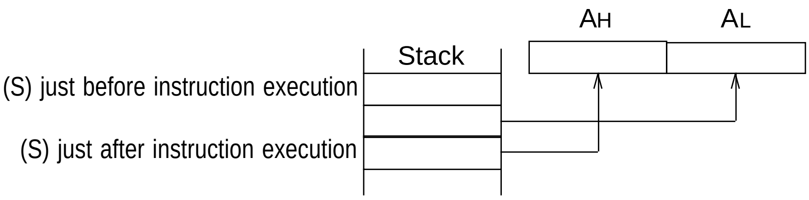
Function : Stack manipulation (Pull)

Operation : A ← Stack

<u>When m = “0”</u>

A ← M(S+2, S+1)

S ← S + 2

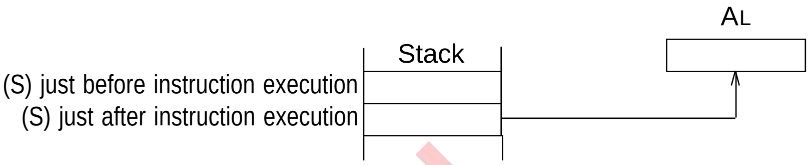


Stack
AH
AL
(S) just before instruction execution
(S) just after instruction execution

<u>When m = “1”</u>

A_L ← M(S+1)

S ← S + 1



Stack
AL
(S) just before instruction execution
(S) just after instruction execution

Description : Increments the stack pointer, and then restores the data at the address specified by the stack pointer to accumulator A. When the data length flag is “0”, 2 bytes are restored. When the data length flag is “1”, only 1 byte is restored to the lower byte of the accumulator A.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PLA	68_{16}	1	5

PLB PuLl accumulator B from stack PLB

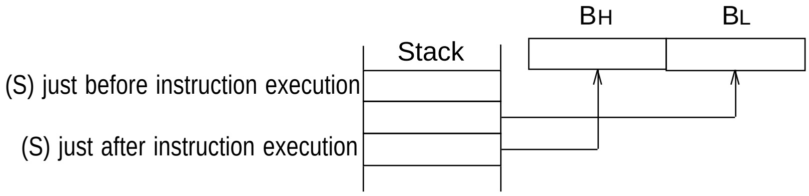
Function : Stack manipulation (Pull)

Operation : B ← Stack

When m = "0"

B ← M(S+2, S+1)

S ← S + 2

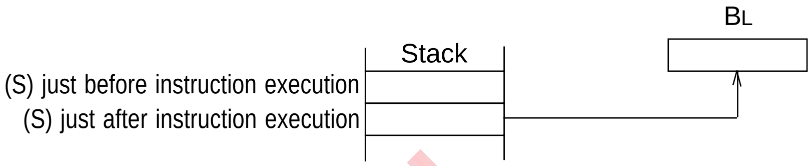


B_H
B_L
Stack
(S) just before instruction execution
(S) just after instruction execution

When m = "1"

B_L ← M(S+1)

S ← S + 1



B_L
Stack
(S) just before instruction execution
(S) just after instruction execution

Description : Increments the stack pointer, and then restores the data at the address specified by the stack pointer to the accumulator B. When the data length flag is "0", 2 bytes are restored. When the data length flag is "1", only 1 byte is restored to the lower byte of the accumulator B.

Status flags

IPL : Not affected.

N : Set to "1" when bit 15 (or bit 7 when the data length flag is "1") of the operation result is "1". Otherwise, cleared to "0".

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to "1" when the result of the operation is "0". Otherwise, cleared to "0".

C : Not affected.

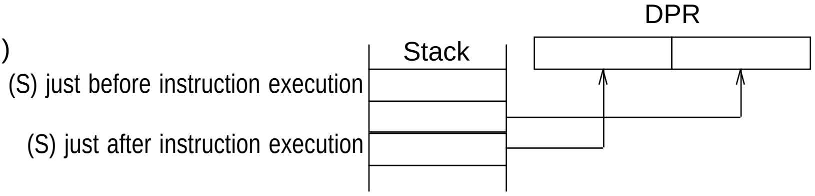
Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PLB	42_{16}, 68_{16}	2	7

PLD PuLl Direct page register from stack PLD

Function : Stack manipulation (Pull)

Operation : DPR ← Stack

DPR ← M(S+2, S+1)
S ← S + 2


DPR
Stack
(S) just before instruction execution
(S) just after instruction execution

Description : Increments the stack pointer, and then restores the data at the address specified by the stack pointer to the direct page register.

Status flags : Not affected.

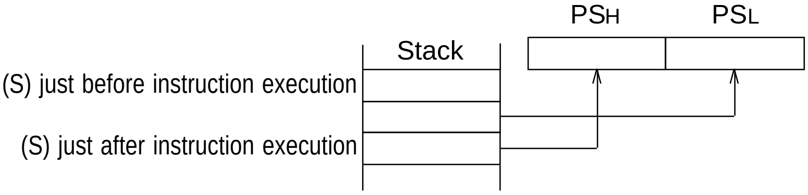
Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PLD	$2B_{16}$	1	5

PuLl Processor status from stack

Function : Stack manipulation (Pull)

Operation : PS ← Stack

PS ← M(S+2, S+1)
S ← S + 2



PSH
PSL
Stack
(S) just before instruction execution
(S) just after instruction execution

Description : Increments the stack pointer, and then restores the data at the address specified by the stack pointer to the processor status register.

Status flags : Changes to the values restored from the stack.

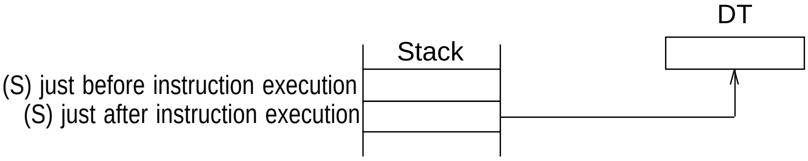
Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PLP	28_{16}	1	6

PLT PuLI daTa bank register from stack PLT

Function : Stack manipulation (Pull)

Operation : DT ← Stack

DT ← M(S+1)
S ← S + 1


DT
Stack
(S) just before instruction execution
(S) just after instruction execution

Description : Increments the stack pointer, and then restores the data at the address specified by the stack pointer to the data bank register.

Status flags

IPL : Not affected.
N : Set to “1” when bit 7 of the operation result is “1”. Otherwise, cleared to “0”.
V : Not affected.
m : Not affected.
x : Not affected.
D : Not affected.
I : Not affected.
Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.
C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PLT	AB_{16}	1	6

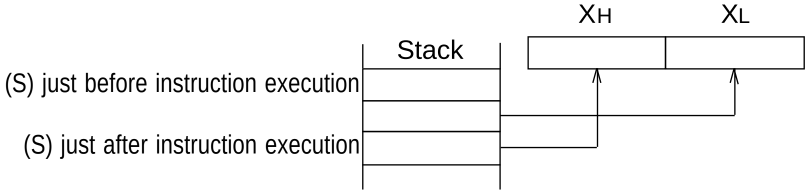
Function : Stack manipulation (Pull)

Operation : X ← Stack

When x = "0"

X ← M(S+2, S+1)

S ← S + 2

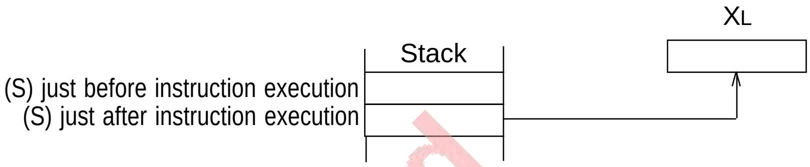


X_H
X_L
Stack
(S) just before instruction execution
(S) just after instruction execution

When x = "1"

X_L ← M(S+1)

S ← S + 1



X_L
Stack
(S) just before instruction execution
(S) just after instruction execution

Description : Increments the stack pointer, and then restores the data at the address specified by the stack pointer to the index register X. When the index register length flag is "0", 2 bytes are restored. When the index register length flag is "1", only 1 byte is restored to the lower byte of the index register X.

Status flags

IPL : Not affected.

N : Set to "1" when bit 15 (or bit 7 when the index register length flag is "1") of the operation result is "1". Otherwise, cleared to "0".

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to "1" when the result of the operation is "0". Otherwise, cleared to "0".

C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PLX	FA_{16}	1	5

PLY PuLl index register Y from stack PLY

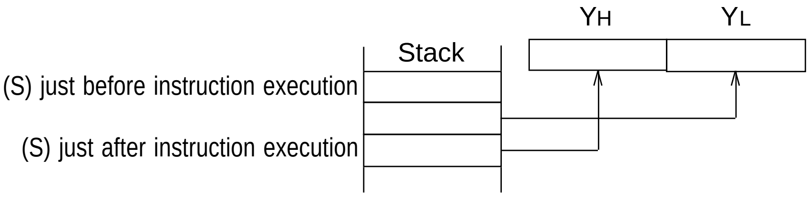
Function : Stack manipulation (Pull)

Operation : Y ← Stack

When x = “0”

Y ← M(S+2, S+1)

S ← S + 2

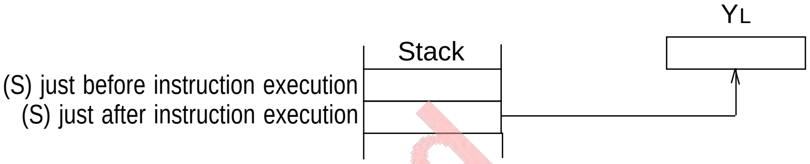


Y_H
Y_L
Stack
(S) just before instruction execution
(S) just after instruction execution

When x = “1”

Y_L ← M(S+1)

S ← S + 1



Y_L
Stack
(S) just before instruction execution
(S) just after instruction execution

Description : Increments the stack pointer, and then restores the data at the address specified by the stack pointer to the index register Y. When the index register length flag is “0”, 2 bytes are restored. When the index register length flag is “1”, only 1 byte is restored to the lower byte of the index register Y.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the index register length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PLY	$7A_{16}$	1	5

PSH PuSH PSH

Function : Stack manipulation (Push)

Operation : Stack ← Specified register of A, B, X, Y, DPR, DT, PG, PS

M(S to S – i) ← A, B, X, Y, DPR, DT, PG, PS
order to save ① ② ③ ④ ⑤ ⑥ ⑦ ⑧
S ← S – i – 1

❊ The immediate value in the second byte of the instruction is used to specify the registers to be saved.

❊ Among the registers being saved, the following registers are affected by the flags just before the instruction is executed.

- A, B registers

  When m = “0” : The high-order and low-order bytes of the register are saved.

  When m = “1” : The low-order byte of the register is saved.

- X, Y registers

  When x= “0” : The high-order and low-order bytes of the register is saved.

  When x= “1” : The low-order byte of the register is saved.

❊ i indicates the number of data bytes to be saved.

Description : This instruction’s second byte specifies the registers to be saved. The registers corresponding to the bits in the second byte that are 1 are saved on the stack. The bit and register correspondence is as follows:

b7							b0
PS	PG	DT	DPR	Y	X	B	A

← Direction to save on the stack

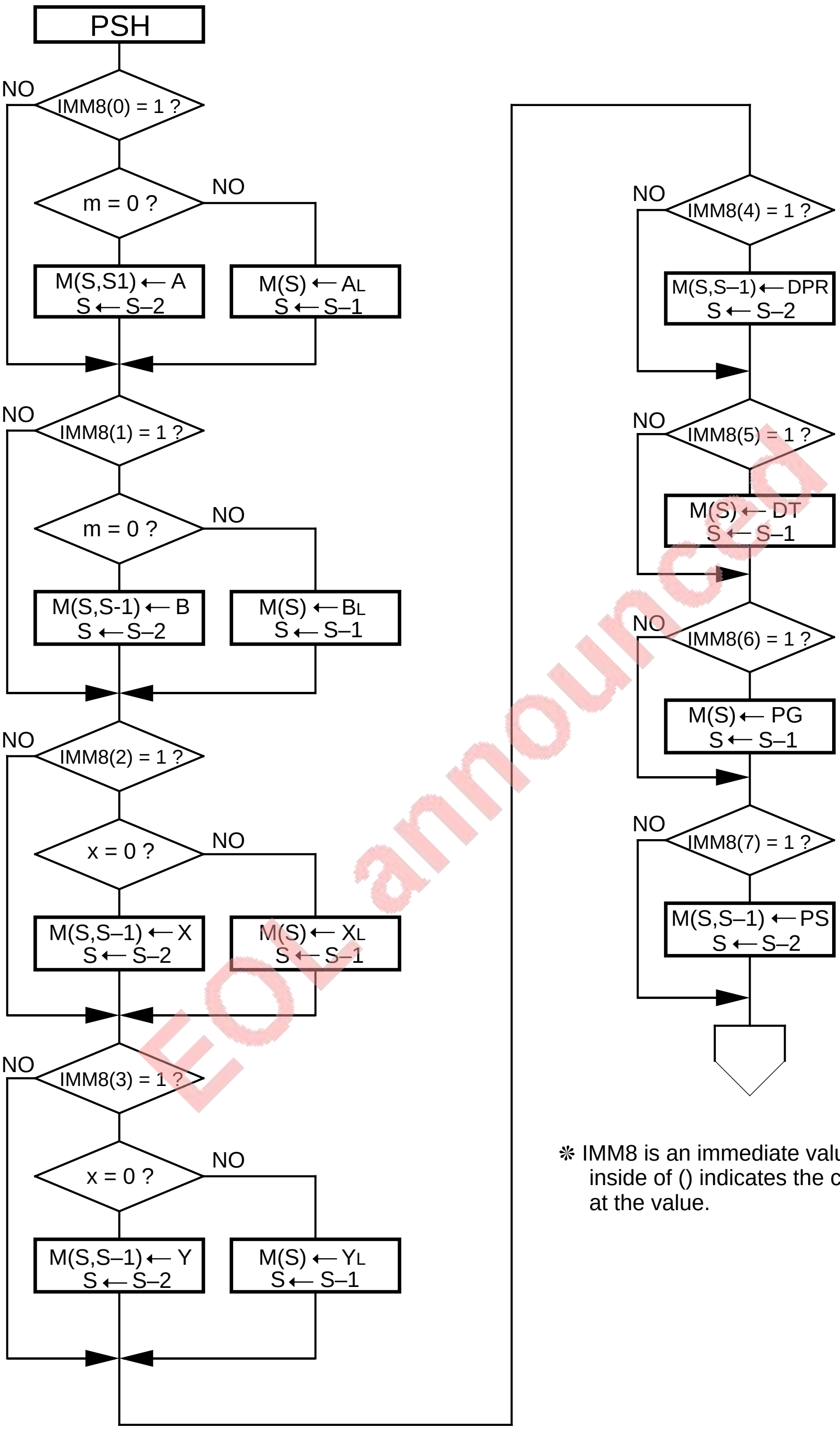
When saving the registers to stack, registers A and B are affected by the m flag, and registers X and Y are affected by the x flag.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PSH #imm	EB_{16}, imm	2	$11+2 \times i_1+i_2$

Note: To the cycle number shown above, the number shown below are added depending on the registers being saved. The number is 11 cycles when no registers are saved. i_1 in above table expresses the number of registers to be saved of A, B, X, Y, DPR and PS; and i_2 expesses it of DT and PG.

Register type	PS	PG	DT	DPR	Y	X	B	A
Cycle number	2	1	1	2	2	2	2	2



❊ IMM8 is an immediate value in 1-byte and inside of () indicates the content of the bit at the value.

Function : Stack manipulation (Pull)

Operation : Specified register of A, B, X, Y, DPR, DT, PG, PS ← Stack

A, B, X, Y, DPR, DT, PS ← M(S + 1 to S + i)
⑦ ⑥ ⑤ ④ ③ ② ① order to restore
S ← S + i

❊ The immediate value in the second byte of the instruction is used to specify the registers to be restored.

❊ Among the registers being restored, the following registers are affected by the flags in the restored PS or the flags just before the instruction is executed.

- A, B registers

  When m = "0" : The high-order and low-order bytes of the register are restored.

  When m = "1" : The low-order byte of the register is restored.

- X, Y registers

  When x = "0" : The high-order and low-order bytes of the register are restored.

  When x = "1" : The low-order byte of the register is restored.

❊ i indicates the number of data bytes to be restored.

Description : This instruction's second byte specifies the registers to be restored. The contents of the stack are restored to the registers corresponding to the bits in the second byte that are 1. The bit and register correspondence is as follows:

b7							b0
PS		DT	DPR	Y	X	B	A

Direction to restore from the stack →

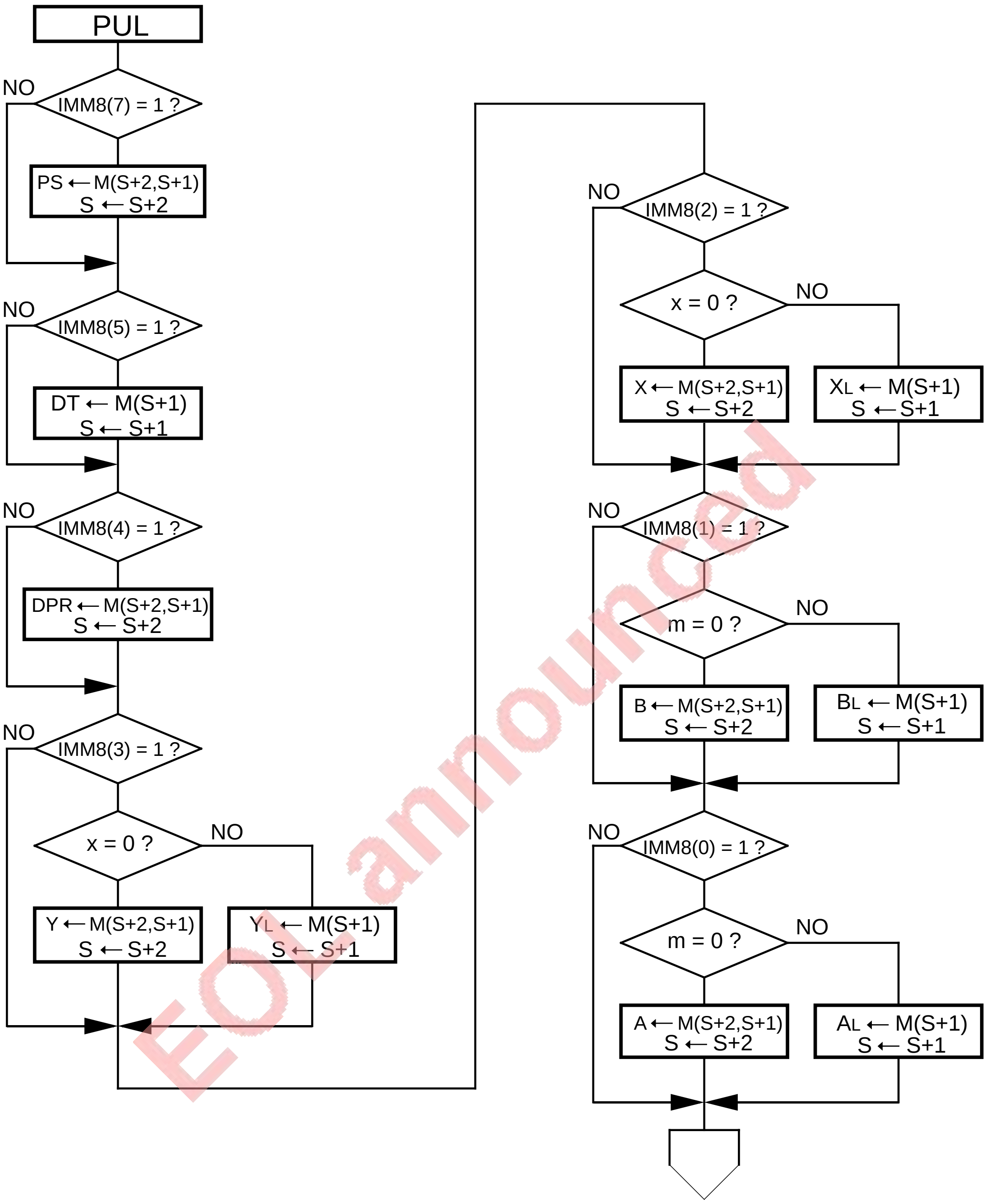
When restoring from stack, registers A and B are affected by the m flag in restored PS, and registers X and Y are affected by the x flag in restored PS. When PS is not restored, the registers are affected by the value of these flags just before instruction execution.

Status flags : When bit 7 of the instruction's second byte is "1", and PS is to be restored, the status flags become restored values. Otherwise, the status flags are not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Stack	PUL #imm	FB_{16}, imm	2	$12+3\times i_1+4\times i_2$

Note: To the cycle number shown above, the number shown below are added depending on the registers being saved. The number is 12 cycles when no registers are saved. i_1 in above table expresses the number of registers to be saved of A, B, X, Y, DT and PS. When restoring DPR, i_2 becomes 1 and When not restoring it, i_2 becomes 0.

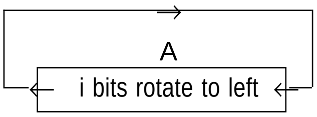
Register type	PS	DT	DPR	Y	X	B	A
Cycle number	3	3	4	3	3	3	3



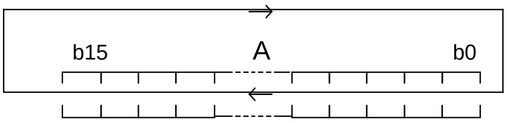
❊ IMM8 is an immediate value in 1-byte and inside of () indicates the content of the bit at the value.

Function : i bits rotate to left

Operation :



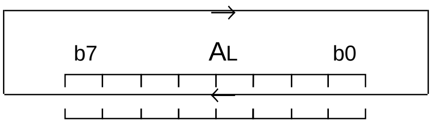
When m = “0”



i bits rotate to left

❊ i = 0 to 65535

When m = “1”



i bits rotate to left

❊ i = 0 to 255

Rotate Left accumulator A

Description : The contents of the accumulator A are rotated to the left by i bits. The value of i is specified by the instruction's third byte (or the third and fourth bytes when the data length flag is "0").

Status flags : Not affected.

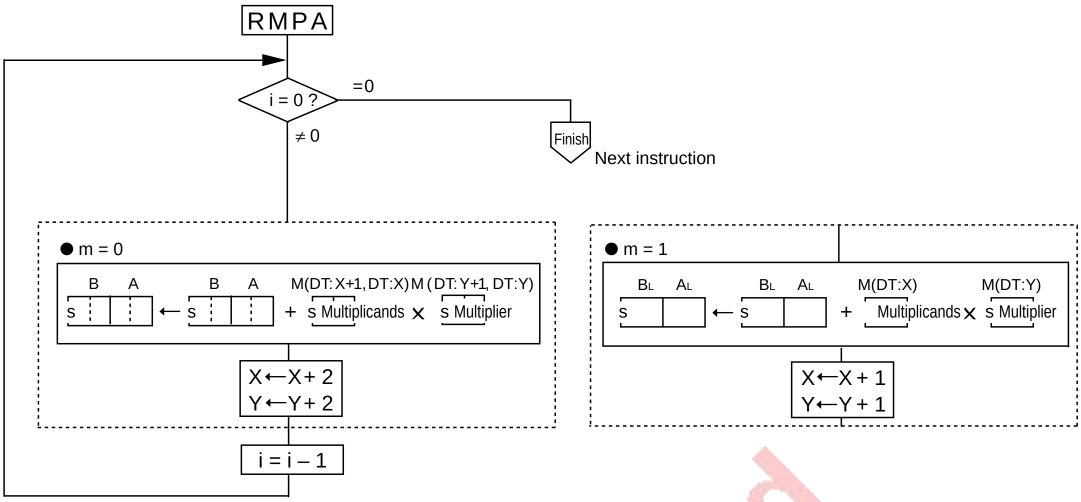
Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	RLA #imm	89_{16}, 49_{16}, imm	3	6+i

i: Number of rotation

Note: When the data length flag is "0", the byte number increases by 1.

Function : Repeat multiply and accumulation

Operation :



- ❊ **s** indicates the sign bit and is the topmost bit of the data to be operated on.
- ❊ **i**, which is a positive number from 0 to 255, indicates the number of repeated operations. It is specified by the third byte of the instruction.
- ❊ Execute this instruction in the condition of index register length flag = “0”.
- ❊ Allocate multipliers and multiplicands in the same bank. Do not allocate each of them across other banks.

Description : When the data length flag is “0”, performs signed multiplication of the 16-bit data in the memory specified by index register X and data bank register, and the 16-bit data in the memory specified by index register Y and data bank register. The multiplication result is added as binary addition to the 32-bit data of which high-order is the contents of accumulator B, and of which low-order is the contents of accumulator A. The high-order 16 bits of the addition result are stored in accumulator B, and the low-order 16 bits are stored in accumulator A again. After the addition, each of the contents of index register X and index register Y is incremented by 2. Additionally, the number of repeated operations is decremented by 1, and when the result is “0”, this instruction execution is finished. When the result is not “0”, the above multiplication and addition are repeated.

When the data length flag (m) is “1”, performs signed multiplication of the 8-bit data in the memory specified by index register X and data bank register, and the 8-bit data in the memory specified by index register Y and data bank register. The multiplication result is added as binary addition to the 16-bit data of which high-order is the contents of accumulator BL, and of which low-order is the contents of accumulator AL. The high-order 8 bits of the addition result are stored in accumulator BL, and the low-order 8 bits are stored in accumulator AL again. After the addition, each of the contents of index register X and index register Y is incremented by 1. Additionally, the number of repeated operations is decremented by 1, and when the result is “0”, this instruction execution is finished. When the result is not “0”, the above multiplication and addition are repeated.

- ● After finishing the instruction execution, index registers X, Y specify the next address of the address where a multiplier and a multiplicand are read last.
- ● When an overflow occurs at addition, the overflow flag becomes “1” and the instruction execution is finished then. When an overflow occurs, accumulators A and B become undefined. Index registers X, Y specify the next address of the address where a multiplier and a multiplicand are read last.
- ● When specifying “0” as the number of repeated operations, the instruction execution is finished without multiplication and addition. The contents of accumulators A, B, and index registers X, Y are not affected.

Status flags :

IPL : Not affected.

N : Checked each time performing addition of the contents of accumulators B, A, and the multiplication result. When bit 31 of the addition result, in other words, bit 15 in accumulator B (when the data length flag (m) = “1”, bit 15 of the addition result, in other words, bit 7 in accumulator B) is “1”, set to 1. Otherwise, cleared to 0. Accordingly, the N flag after finishing the instruction execution is specified by the result of the last addition.

V : Checked each time performing addition of the contents of accumulators B, A, and the multiplication result. When the result is outside range of –2147483648 to +2147483647 (when the data length flag (m) = “1”, outside range of –32768 to +32767), set to “1”. Otherwise, cleared to 0.

● When an overflow occurs at addition, the instruction execution is finished then. Accordingly, the V flag after finishing the instruction execution is cleared to “0” with normal finish, and set to “1” with an overflow.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Checked each time performing addition of the contents of accumulators B, A, and the multiplication result. When the addition result is “0”, set to 1. Otherwise, cleared to 0. Accordingly, Z flag after finishing the instruction execution is specified by the result of the last addition.

C : Checked each time performing addition of the contents of accumulators B, A, and the multiplication result. When the addition result exceeds +4294967295 as not signed data (when the data length flag (m) = “1”, exceeds +65536), set to “1”. Otherwise, cleared to 0.

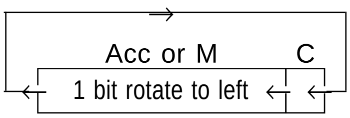
Addressing mode	Syntax	Machine code	Bytes	Cycles
Multiplied accumulation	RMPA #imm	89_{16}, $E2_{16}$, imm	3	6 + 16 × n

Note: Cycles show the numbers of cycle in the case of data length flag = “1”. It becomes 6 + 20 × n cycles in the case of data length flag = “0”.

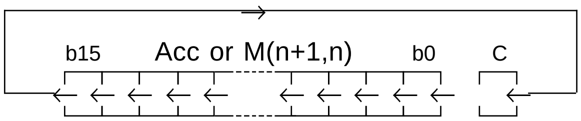
ROL ROtate one bit Left ROL

Function : Rotate to left

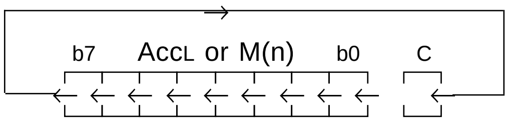
Operation :



When m = "0"



When m = "1"



Description : The carry flag is linked to the accumulator or a memory, and the combined contents are rotated to the 1 bit left.
Bit 0 of the accumulator or a memory is loaded with the content of the carry flag before execution of this instruction. The carry flag is loaded with the content of bit 15 (or bit 7 when the data length flag is "1") of the accumulator or a memory before execution of this instruction.

Status flags

IPL : Not affected.

N : Set to "1" when bit 15 (or bit 7 when the data length flag is "1") of the operation result is "1". Otherwise, cleared to "0".

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to "1" when the result of the operation is "0". Otherwise, cleared to "0".

C : Set to "1" when bit 15 (or bit 7 when the data length flag is "1") before execution of the instruction is "1". Otherwise, cleared to "0".

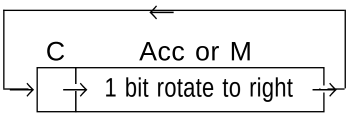
Addressing mode	Syntax	Machine code	Bytes	Cycles
Accumulator	ROL A	$2A_{16}$	1	2
Direct	ROL dd	26_{16}, dd	2	7
Direct indexed X	ROL dd, X	36_{16}, dd	2	7
Absolute	ROL mmll	$2E_{16}$, ll, mm	3	7
Absolute indexed x	ROL mmll, X	$3E_{16}$, ll, mm	3	8

Note: This table applies when using the accumulator A. When using the accumulator B, replace "A" with "B" in the syntax. In this case, "42_{16}" is added at the beginning of the machine code, the byte number increases by 1 and the cycle number increases by 2.

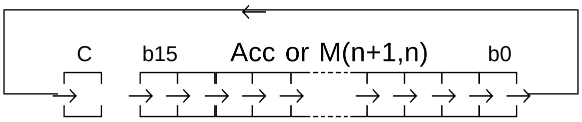
ROR ROtate one bit Right ROR

Function : Rotate to right

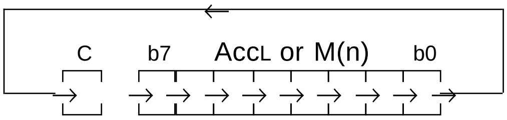
Operation :



When m = “0”



When m = “1”



Description : The carry flag is linked to the accumulator or a memory, and the combined contents are rotated to the 1 bit right.
Bit 15 (or bit 7 when the data length flag is “1”) of the accumulator or a memory is loaded with the content of the carry flag. The carry flag is loaded with the content of bit 0 of the accumulator or a memory before execution of this instruction.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Set to “1” when bit 0 before execution of the instruction is “1”. Otherwise, cleared to “0”.

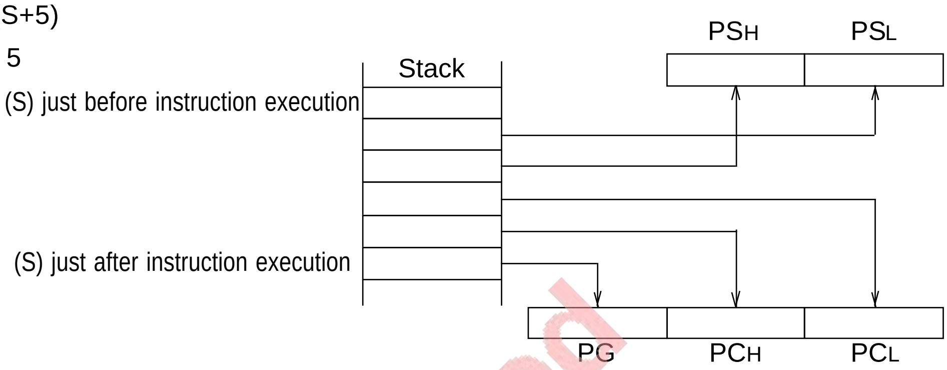
Addressing mode	Syntax	Machine code	Bytes	Cycles
Accumulator	ROR A	$6A_{16}$	1	2
Direct	ROR dd	66_{16}, dd	2	7
Direct indexed X	ROR dd, X	76_{16}, dd	2	7
Absolute	ROR mmll	$6E_{16}$, ll, mm	3	7
Absolute indexed X	ROR mmll, X	$7E_{16}$, ll, mm	3	8

Note: This table applies when using the accumulator A. When using the accumulator B, replace “A” with “B” in the syntax. In this case, “42_{16}” is added at the beginning of the machine code, the byte number increases by 1 and the cycle number increases by 2.

Function : Return from interrupt

Operation : PG, PC, PS ← Stack (Saved content when interrupt is caused)

PS ← M(S+2, S+1)
PC ← M(S+4, S+3)
PG ← M(S+5)
S ← S + 5


Stack
(S) just before instruction execution
(S) just after instruction execution
PSH
PSL
PG
PCH
PCL

Description : The contents of the processor status register, program counter, and program bank register, which were saved on the stack when the interrupt request was accepted, are restored to these registers.
The state becomes the same as that before the acceptance of the interrupt request.

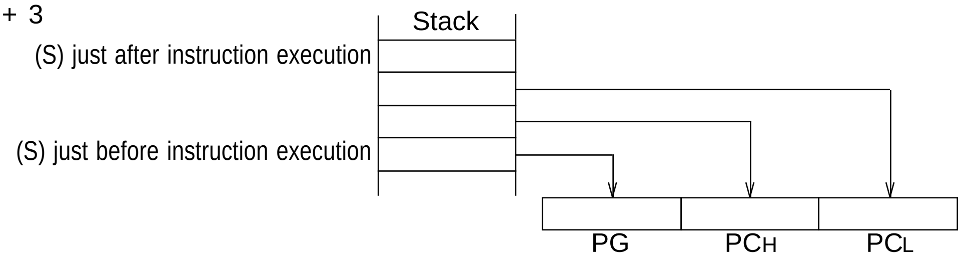
Status flags : Changes to the values that had been on the stack.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	RTI	40_{16}	1	9

Function : Return from subroutine long

Operation : PG, PC ← Stack (Subroutine long return address)

PC ← M(S+2, S+1)
PG ← M(S+3)
S ← S + 3


Stack
(S) just after instruction execution
(S) just before instruction execution
PG
PCH
PCL

Description : The contents of the stack are restored to the program counter and program bank register.

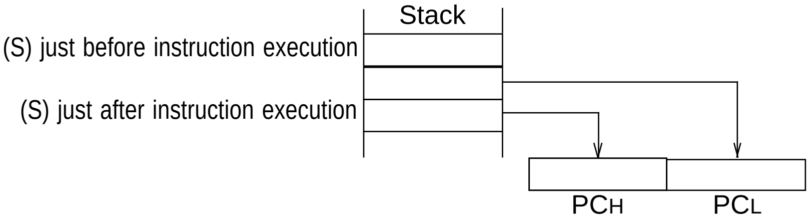
Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	RTL	$6B_{16}$	1	7

Function : Return from subroutine

Operation : PC ← Stack (Subroutine return address)

PC ← M(S+2, S+1)
S ← S + 2


Stack
(S) just before instruction execution
(S) just after instruction execution
PCH
PCL

Description : The contents of the stack are restored to the program counter.
The contents of program bank register is incremented by 1 when this instruction is allocated at the highest address (XXFFFF$_{16}$) of a bank.

Status flags : Not affected.

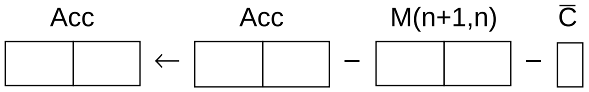
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	RTS	60$_{16}$	1	5

SBC SuBtract with Carry SBC

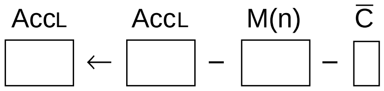
Function : Subtract with carry

Operation : Acc ← Acc – M – $\overline{C}$

When m = "0"



When m = "1"



Description : Subtracts the contents of a memory and the complement of the carry flag from the contents of the accumulator, and places the result in the accumulator. Performed as a binary subtraction when the decimal mode flag is "0". Performed as a decimal subtraction when the decimal mode flag is "1".

Status flags

IPL : Not affected.

N : Set to "1" when bit 15 (or bit 7 when the data length flag is "1") of the operation result is "1". Otherwise, cleared to "0". Meaningless for a decimal subtraction.

V : Set to "1" when a binary subtraction of signed data results in a value exceeding the range of –32768 to +32767 (–128 to +127 when the data length flag is "1"). Otherwise, cleared to "0". Meaningless for a decimal subtraction.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to "1" when the operation result is "0". Otherwise, cleared to "0".

C : Set to "1" when the operation result is equal to or larger than "0". Otherwise, cleared to "0", and a borrow is indicated.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	SBC A, #imm	$E9_{16}$, imm	2	2
Direct	SBC A, dd	$E5_{16}$, dd	2	4
Direct indexed X	SBC A,dd, X	$F5_{16}$, dd	2	5
Direct indirect	SBC A, (dd)	$F2_{16}$, dd	2	6
Direct indexed X indirect	SBC A,(dd, X)	$E1_{16}$, dd	2	7
Direct indirect indexed Y	SBC A,(dd), Y	$F1_{16}$, dd	2	8
Direct indirect long	SBCL A, (dd)	$E7_{16}$, dd	2	8
Direct indirect long indexed Y	SBCL A, (dd), Y	$F7_{16}$, dd	2	10
Absolute	SBC A,mmll	ED_{16},ll,mm	3	4
Absolute indexed X	SBC A, mmll, X	FD_{16}, ll, mm	3	6
Absolute indexed Y	SBC A, mmll, Y	$F9_{16}$, ll, mm	3	6
Absolute long	SBC A, hhmmll	EF_{16}, ll, mm, hh	4	6
Absolute long indexed X	SBC A, hhmmll, X	FF_{16}, ll, mm, hh	4	7
Stack pointer relative	SBC A, nn, S	$E3_{16}$, nn	2	5
Stack pointer relative indirect indexed Y	SBC A, (nn, S), Y	$F3_{16}$, nn	2	8

Notes 1: This table applies when using the accumulator A. When using the accumulator B, replace “A” with “B” in the syntax. In this case, “42_{16}” is added at the beginning of the machine code, the byte number increases by 1 and the cycle number increases by 2.

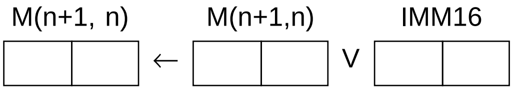
2: When treating a 16-bit data in the immediate addressing mode in the condition of the data length flag = “0”, the byte number increases by 1.

SEB SEt Bit SEB

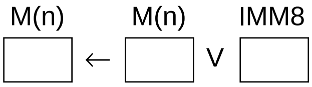
Function : Bit manipulation

Operation : Mb ← 1 (b is the specified bits)

When m = “0”



When m = “1”



❊ IMM is an immediate value indicating the bits to be set with a “1”. The bits are specified by the last 1 or 2 bytes of the instruction.

Description : The SEB instruction sets the specified memory bits to “1”. Multiple bits to be set can be specified at the same time.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Direct bit	SEB #imm, dd	04_{16}, dd, imm	3	8
Absolute bit	SEB #imm, mmll	$0C_{16}$, ll, mm, imm	4	9

Note: When treating a 16-bit data in the condition of the data length flag = “0”, the byte number increases by 1.

SEt Carry flag

Function : Flag manipulation

Operation : C ← 1

Description : Sets the carry flag to “1”.

Status flags

IPL	:	Not affected.
N	:	Not affected.
V	:	Not affected.
m	:	Not affected.
x	:	Not affected.
D	:	Not affected.
I	:	Not affected.
Z	:	Not affected.
C	:	Set to “1”.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	SEC	38_{16}	1	2

SEt Interrupt disable status

Function : Flag manipulation

Operation : I ← 1

Description : Sets the interrupt disable flag to “1”.

Status flags

IPL	:	Not affected.
N	:	Not affected.
V	:	Not affected.
m	:	Not affected.
x	:	Not affected.
D	:	Not affected.
I	:	Set to “1”.
Z	:	Not affected.
C	:	Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	SEI	78_{16}	1	2

SEM SEt M flag SEM

Function : Flag manipulation

Operation : m ← 1

Description : Sets the data length flag to “1”.

Status flags

IPL :	Not affected.
N :	Not affected.
V :	Not affected.
m :	Set to “1”.
x :	Not affected.
D :	Not affected.
I :	Not affected.
Z :	Not affected.
C :	Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	SEM	$F8_{16}$	1	2

SEP — SEt Processor status — SEP

Function : Flag manipulation

Operation : PSLb ← 1 (b is the specified flags)

PSL ← PSL V IMM8

❊ IMM8 is an 1-byte, immediate value indicating the bits to be set with a “1”. The bits are specified by the last 1 or 2 bytes of the instruction.

b7	b6	b5	b4	b3	b2	b1	b0	
N	V	m	x	D	I	Z	C	PSL

Description : Sets the processor status flags specified by the bit pattern in the second byte of the instruction to “1”.

Status flags : The specified status flags are set to “1”. IPL is not affected.

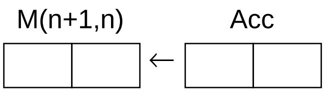
Addressing mode	Syntax	Machine code	Bytes	Cycles
Immediate	SEP #imm	$E2_{16}$, imm	2	3

STA STore Accumulator in memory STA

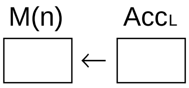
Function : Store

Operation : M ← Acc

When m = “0”



When m = “1”



Description : Stores the contents of the accumulator into a memory.

The contents of the accumulator are not changed.

Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Direct	STA A, dd	85_{16}, dd	2	4
Direct indexed X	STA A, dd, X	95_{16}, dd	2	5
Direct indirect	STA A, (dd)	92_{16}, dd	2	7
Direct indexed X indirect	STA A, (dd, X)	81_{16}, dd	2	7
Direct indirect indexed Y	STA A, (dd), Y	91_{16}, dd	2	7
Direct indirect long	STAL A, (dd)	87_{16}, dd	2	9
Direct indirect long indexed Y	STAL A, (dd), Y	97_{16}, dd	2	9
Absolute	STA A, mmll	$8D_{16}$, ll, mm	3	5
Absolute indexed X	STA A, mmll, X	$9D_{16}$, ll, mm	3	5
Absolute indexed Y	STA A, mmll, Y	99_{16}, ll, mm	3	5
Absolute long	STA A, hhmmll	$8F_{16}$, ll, mm, hh	4	6
Absolute long indexed X	STA A, hhmmll, X	$9F_{16}$, ll, mm, hh	4	7
Stack pointer relative	STA A, nn, S	83_{16}, nn	2	5
Stack pointer relative indirect indexed Y	STA A, (nn, S), Y	93_{16}, nn	2	8

Note: This table applies when using the accumulator A. When using the accumulator B, replace “A” with “B” in the syntax. In this case, “42_{16}” is added at the beginning of the machine code, the byte number increases by 1 and the cycle number increases by 2.

Function : Oscillation control

Operation : Stop the oscillation

Description : Resets the oscillator controlling flip-flop to inhibit the oscillation of the oscillation circuit. To restart the oscillator, either an interrupt or reset must be performed.

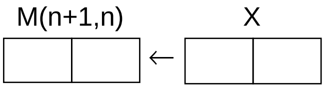
Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	STP	DB_{16}	1	3

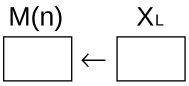
Function : Store

Operation : M ← X

When x = “0”



When x = “1”



Description : Stores the contents of the index register X into a memory. The contents of the index register X are not changed.

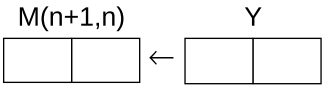
Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Direct	STX dd	86_{16}, dd	2	4
Direct indexed Y	STX dd, Y	96_{16}, dd	2	5
Absolute	STX mmll	$8E_{16}$, ll, mm	3	5

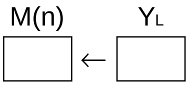
Function : Store

Operation : M ← Y

When x = “0”



When x = “1”



Description : Stores the contents of the index register Y into a memory. The contents of the index register Y are not changed.

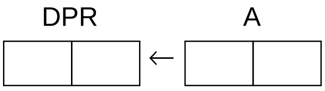
Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Direct	STY dd	84_{16}, dd	2	4
Direct indexed X	STY dd, X	94_{16}, dd	2	5
Absolute	STY mmll	$8C_{16}$, ll, mm	3	5

TAD Transfer accumulator A to Direct page register TAD

Function : Transfer

Operation : DPR ← A



Description : Loads the direct page register with the contents of the accumulator A. Data is transferred as a 16-bit data regardless of the data length flag. The contents of the accumulator A are not changed.

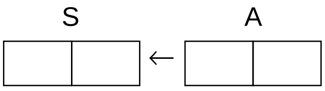
Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TAD	5B$_{16}$	1	2

TAS Transfer accumulator A to Stack pointer TAS

Function : Transfer

Operation : S ← A



Description : Loads the stack pointer with the contents of the accumulator A. Data is transferred as a 16-bit data regardless of the data length flag. The contents of the accumulator A are not changed.

Status flags : Not affected.

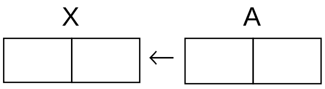
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TAS	$1B_{16}$	1	2

TAX Transfer accumulator A to index register X TAX

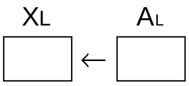
Function : Transfer

Operation : X ← A

When x = “0”



When x = “1”



Description : Loads the index register X with the contents of the accumulator A. The contents of the accumulator A are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the index register length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

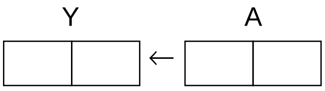
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TAX	AA_{16}	1	2

Transfer accumulator A to index register Y

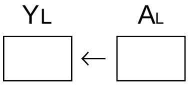
Function : Transfer

Operation : Y ← A

When x = “0”



When x = “1”



Description : Loads the index register Y with the contents of the accumulator A. The contents of the accumulator A are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the index register length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

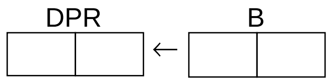
C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TAY	$A8_{16}$	1	2

TBD Transfer accumulator B to Direct page register TBD

Function : Transfer

Operation : DPR ← B



Description : Loads the direct page register with the contents of the accumulator B. Data is transferred as a 16-bit data regardless of the data length flag. The contents of the accumulator B are not changed.

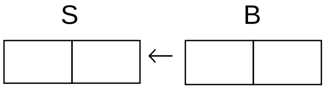
Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TBD	42_{16}, $5B_{16}$	2	4

TBS Transfer accumulator B to Stack pointer TBS

Function : Transfer

Operation : S ← B



Description : Loads the stack pointer with the contents of the accumulator B. Data is transferred as a 16-bit data regardless of the data length flag. The contents of the accumulator B are not changed.

Status flags : Not affected.

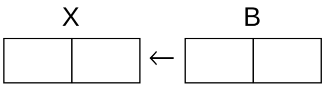
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TBS	42_{16}, $1B_{16}$	2	4

TBX Transfer accumulator B to index register X TBX

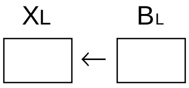
Function : Transfer

Operation : X ← B

When x = “0”



When x = “1”



Description : Loads the index register X with the contents of the accumulator B. The contents of the accumulator B are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the index register length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

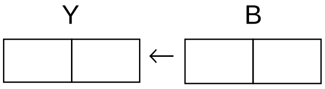
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TBX	42_{16}, AA_{16}	2	4

Transfer accumulator B to index register Y

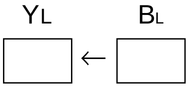
Function : Transfer

Operation : Y ← B

When x = “0”



When x = “1”



Description : Loads the index register Y with the contents of the accumulator B. The contents of the accumulator B are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the index register length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

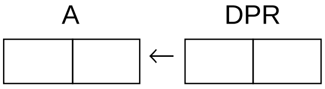
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TBY	42_{16}, $A8_{16}$	2	4

TDA Transfer Direct page register to accumulator A TDA

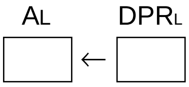
Function : Transfer

Operation : A ← DPR

When m = “0”



When m = “1”



Description : Loads the accumulator A with the contents of the direct page register. The contents of the direct page register are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

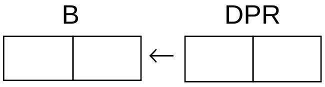
C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TDA	$7B_{16}$	1	2

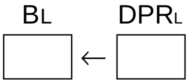
Function : Transfer

Operation : B ← DPR

When m = “0”



When m = “1”



Description : Loads the accumulator B with the contents of the direct page register. The contents of the direct page register are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

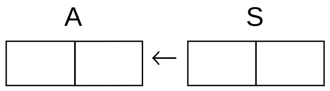
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TDB	42_{16}, $7B_{16}$	2	4

Transfer Stack pointer to accumulator A

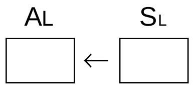
Function : Transfer

Operation : A ← S

When m = “0”



When m = “1”



Description : Loads the accumulator A with the contents of the stack pointer. The contents of the stack pointer are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

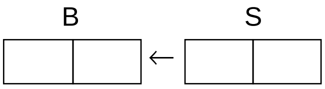
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TSA	$3B_{16}$	1	2

Transfer Stack pointer to accumulator B

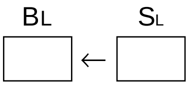
Function : Transfer

Operation : B ← S

When m = “0”



When m = “1”



Description : Loads the accumulator B with the contents of the stack pointer. The contents of the stack pointer are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

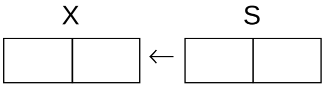
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TSB	42_{16}, $3B_{16}$	2	4

TSX Transfer Stack pointer to index register X TSX

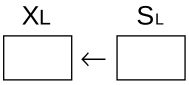
Function : Transfer

Operation : X ← S

When x = “0”



When x = “1”



Description : Loads the index register X with the contents of the stack pointer. The contents of the stack pointer are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the index register length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

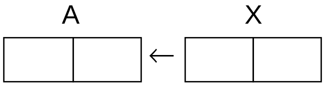
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TSX	BA_{16}	1	2

Transfer index register X to accumulator A

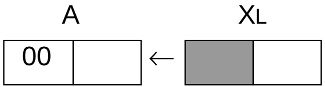
Function : Transfer

Operation : A ← X

When m = “0” and x = “0”

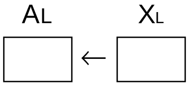


When m = “0” and x = “1”



❊ Under this condition, 00_{16} is transferred to A_H regardless of X_H.

When m = “1”



Description : Loads the accumulator A with the contents of the index register X. The contents of the index register X are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

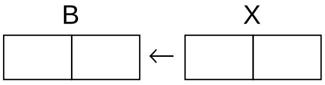
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TXA	$8A_{16}$	1	2

TXB Transfer index register X to accumulator B TXB

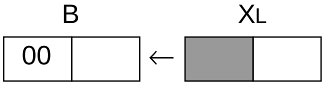
Function : Transfer

Operation : B ← X

When m = “0” and x = “0”

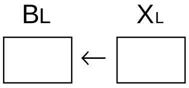


When m = “0” and x = “1”



❇ Under this condition, 00_{16} is transferred to A_H regardless of X_H.

When m = “1”



Description : Loads the accumulator B with the contents of the index register X. The contents of the index register X are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

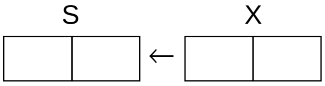
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TXB	42_{16}, $8A_{16}$	2	4

TXS Transfer index register X to Stack pointer TXS

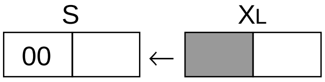
Function : Transfer

Operation : S ← X

When x = “0”



When x = “1”



❇ Under this condition, 00_{16} is transferred to S_H regardless of X_H.

Description : Loads the stack pointer with the contents of the index register X. The contents of the index register X are not changed.

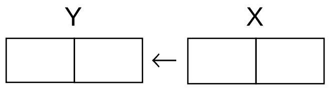
Status flags : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TXS	$9A_{16}$	1	2

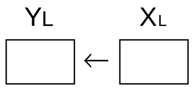
Function : Transfer

Operation : Y ← X

When x = “0”



When x = “1”



Description : Loads the index register Y with the contents of the index register X. The contents of the index register X are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the index register length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

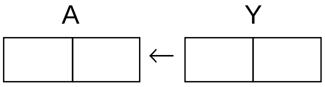
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TXY	$9B_{16}$	1	2

TYA Transfer index register Y to accumulator A TYA

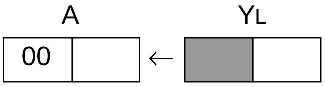
Function : Transfer

Operation : A ← Y

When m = “0” and x = “0”

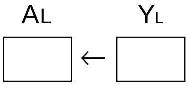


When m = “0” and x = “1”



❊ Under this condition, 00_{16} is transferred to A_H regardless of Y_H.

When m = “1”



Description : Loads the accumulator A with the contents of the index register Y. The contents of the index register Y are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

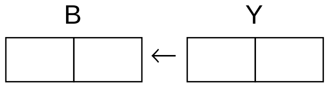
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TYA	98_{16}	1	2

TYB Transfer index register Y to accumulator B TYB

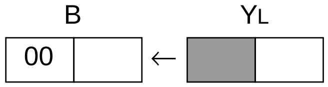
Function : Transfer

Operation : B ← Y

When m = “0” and x = “0”

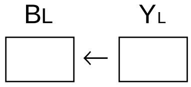


When m = “0” and x = “1”



❊ Under this condition, 00_{16} is transferred to B_H regardless of Y_H.

When m = “1”



Description : Loads the accumulator B with the contents of the index register Y. The contents of the index register Y are not changed.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the operation result is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the result of the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

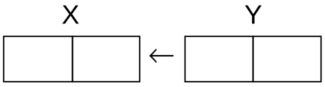
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TYB	42_{16}, 98_{16}	2	4

TYX Transfer index register Y to X TYX

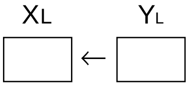
Function : Transfer

Operation : X ← Y

When x = "0"



When x = "1"



Description : Loads the index register X with the contents of the index register Y. The contents of the index register Y are not changed.

Status flags

IPL : Not affected.

N : Set to "1" when bit 15 (or bit 7 when the index register length flag is "1") of the operation result is "1". Otherwise, cleared to "0".

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to "1" when the result of the operation is "0". Otherwise, cleared to "0".

C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	TYX	BB_{16}	1	2

WaIT

Function : Clock control

Operation : Stop the CPU clock

Description : The WIT instruction stops the internal clock. However, the oscillation of the oscillation circuit is not stopped. To restart the internal clock, either an interrupt or reset must be performed.

Status flags : Not affected.

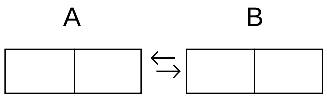
Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	WIT	CB_{16}	1	3

XAB eXchange accumulator A and B XAB

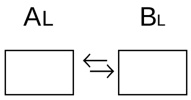
Function : Exchange

Operation : A ⇆ B

When m = “0”



When m = “1”



Description : Swaps the contents of the accumulators A and B.

Status flags

IPL : Not affected.

N : Set to “1” when bit 15 (or bit 7 when the data length flag is “1”) of the accumulator A after the operation is “1”. Otherwise, cleared to “0”.

V : Not affected.

m : Not affected.

x : Not affected.

D : Not affected.

I : Not affected.

Z : Set to “1” when the contents of the accumulator A after the operation is “0”. Otherwise, cleared to “0”.

C : Not affected.

Addressing mode	Syntax	Machine code	Bytes	Cycles
Implied	XAB	89_{16}, 28_{16}	2	5

4.3 Notes for programming

Make sure of the following when programming.

(1) Set an initial value to the stack pointer because it is undefined immediately after removing reset.

Example)
```
LDX    #27FH
TXS
```

(2) Do not set a value other than “00_{16}” to the program bank register and the data bank register. It is because they are invalid in the single-chip mode.

(3) When performing a decimal operation, in the condition of the decimal mode flag = “1” :

With ADC instruction

Only the carry flag is valid.
The zero, negative, and overflow flags are invalid.

With SBC instruction

Only the carry and zero flags are valid.
The negative and overflow flags are invalid.

Note: Decimal operation can be done only with the ADC and the SBC instructions.

(4) Using a 16-bit immediate data in the condition of the data length flag = “1” (data length : 8 bits), or using an 8-bit immediate data in the condition of the data length flag = “0” (data length : 16 bits) will cause the program runaway. The same rule is applied to the index register length flag. Accordingly, take care of the condition of these flags when programming.

(5) The 7751 series can prefetch the instructions using the 3-byte instruction queue buffer. Make sure that when creating a timer with the software, the cycle number shown in the list of machine instructions is the minimum. (Refer to Chapter 5.)

(6) When setting a value other than “00_{16}” into the low-order 8 bits of the direct page register (DPR_L), the processing time needs 1 machine cycle longer than when setting “00_{16}”.

(7) The processing speed will deteriorate when a 16-bit data is accessed from an odd address. Allocate a 16-bit data from an even address when the processing speed is important.

(8) By execution of the PLA instruction, the zero and negative flags will change. When the only accumulator A is restored by execution of the PUL instruction, the contents of the processor status register will not change.

(9) The PSH instruction can save the program bank register on the stack by setting “1” in bit 6 of the operand. However, the PUL instruction cannot restore the program bank register.

(10) Any code in the second byte of the BRK instruction will not affect to the CPU.

MEMO

CHAPTER 5
NUMBER OF INSTRUCTION CYCLES

5.1 Description

The number of instruction cycles shows instruction execution time with the cycle number of ϕ.
The number of instruction cycles for execution time of one instruction can be shown as the following:

- Execution time of one instruction (s) = $\frac{1}{\phi}\times$ Instruction cycle number of one instruction

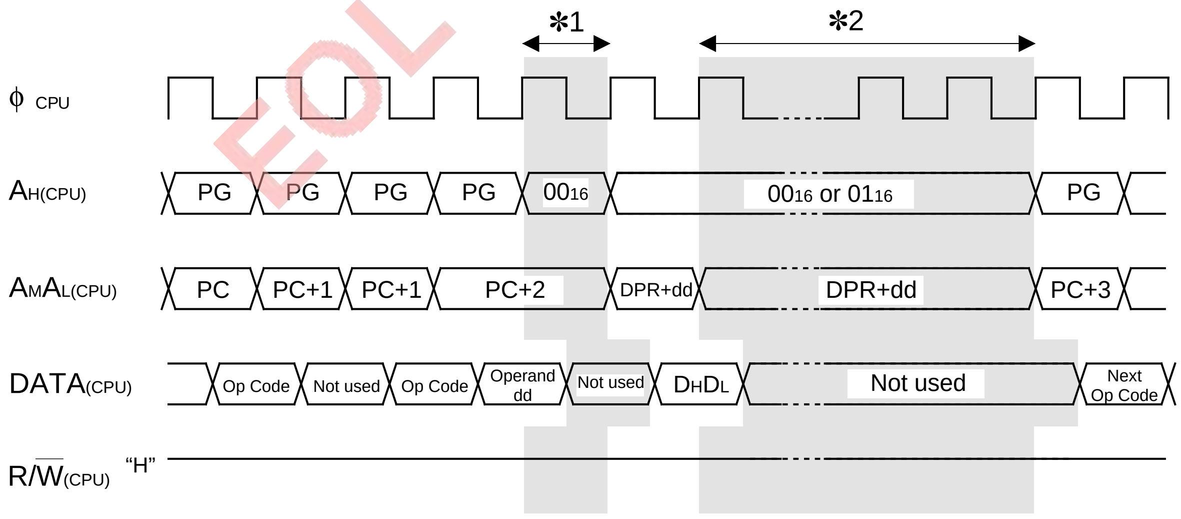
The number of instruction cycles changes depending on the instruction execution condition even when the same instruction is executed in the same addressing mode.
This paragraph explains change factors of the number of instruction cycles.

5.1.1 CPU instruction execution sequence

The number of cycles which is necessary so that the central processing unit (CPU) can execute an instruction is shown in Chapter 6 CPU instruction execution sequence for each addressing mode. It is the number of cycles of ϕ_{CPU}.
Those numbers of cycles are the ideal values; it is assumed to be possible to supply the bus interface unit (BIU) with the instructions and the data of a necessary number of bytes which the CPU requires then.
Actually, the CPU standby cycle, which is explained in the next paragraph, is generated because the supply capability of BIU is limited.
With part of instructions or addressing modes, the cycle number of ϕ_{CPU} which is necessary so that the CPU can execute the instruction changes owing to the factors shown in Table 5.1.1.
Figure 5.1.1 shows an example of the change of the CPU instruction execution sequence.

Table 5.1.1 Change factors of CPU instruction execution sequence

Factor	Instruction/Addressing mode
Value of direct page register's low-order byte (DPR_L)	Addressing mode using direct page register
Value of data length flag (m)	Instructions DIV, DIVS, MPY, MPYS



✻1 This cycle is shortened when DPR_L = "00_{16}".
This cycle is valid when $DPR_L \neq$ "00_{16}".
✻2 This term is 25 cycles when m = "0".
This term is 17 cycles when m = "1".

Note : This is not observed externally because this is the internal operation of the CPU.

Fig. 5.1.1 Example of change of the CPU instruction execution sequence (in DIV instruction and direct addressing mode)

5.1.2 CPU standby cycle

In the case of Table 5.1.2, the BIU makes the CPU stand by owing to extending the duration at the “L” level of ϕ CPU (Refer to 2.2 Bus interface unit”).

In this case, the number of instruction cycles becomes the following:

•Cycle number of ϕCPU which is necessary so that the CPU can execute the instruction + Cycle number of CPU standby.

Timing that the BIU makes the CPU stand by and the cycle number of the CPU standby change by the change of the memory access frequency and the memory access time shown in Table 5.1.3.

As a rule, the more memory access time and the more memory access frequency, the more the CPU standby cycle increases.

Table 5.1.2 When BIU makes CPU stand by

Item	Standby duration
Being short of instructions in the instruction queue buffer when the CPU requires them.	Standby until BIU places required or more instructions into the instruction queue buffer.
BIU reading data.	Standby until the reading is completed.
BIU using bus to prefetch instructions or write data when the CPU requires data read or data write.	Standby until to prefetch instructions or write data is completed.

Table 5.1.3 Change factor of cycle number of CPU standby

Factor	Change of memory access time and memory access frequency
Bus cycle	The longer bus cycle, the more time of memory access
Input level of BYTE pin (External data bus width)	●Access frequency of 16-bit data from an even address at the external area (Note 1) "L": Once "H": Twice ●Prefetch unit (frequency) of instructions at the external area "L": 2-byte unit (a few of prefetch frequency) "H": 1-byte unit (a lot of prefetch frequency)
Access address (Beginning address)	●Access frequency of 16-bit data in the case of data bus width = 16 bits (Note 1) Even: Once Odd: Twice ●Prefetch unit (frequency) of instructions in the case of data bus width = 16 bits "L": 2-byte unit "H": 1-byte unit (Note 2)
Number of instructions in the instruction queue buffer when beginning to execute an instruction	The fewer instructions in the instruction queue buffer, the more prefetch frequency.

Notes 1: In the instruction affected by the data length flag (m) and index register length flag (x), in the condition of m(x) flag = "0".

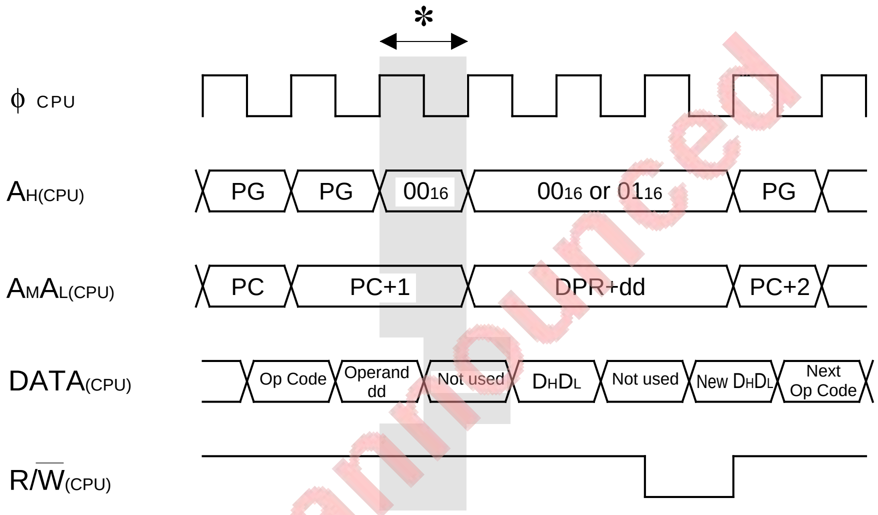
2: An instruction is prefetched by one byte when the address accessed first at such as branches of the program is only an odd address. After that, the instruction is prefetched by two bytes from an even address during executing instructions continuously.

3: 0 bytes when beginning to execute the program and a branch.
Otherwise, the user cannot select it because it is changed depending on the state of the previous instruction completion.
3 bytes or less in the case of when the input level of BYTE pin is "L", 2 bytes or less in the case of when it is "H".

5.2 Points of view

When thinking about the number of instruction cycles as calculation, it is required to concurrently think about the operation of the CPU and BIU taking the factors of above-mentioned Tables 5.1.1 to 5.1.3 in consideration. The idea examples of the number of instruction cycles are shown using Figures 5.2.1 – 5.2.3.
Figure 5.2.1 shows a CPU instruction execution sequence (in ASL instruction and direct addressing mode). Figures 5.2.2 and 5.2.3 show the operation of the CPU and BIU of Figure 5.2.1 with based on ϕ.

When instructions are executed continuously, the number of the following instruction cycles is affected by the state of the instruction queue buffer and the bus at the previous instruction's completion.
Accordingly, examine instructions of that routine continuously in order of execution when you calculate execution time of one routine.



✻ This cycle is shortened when DPR_L = "00_{16}".
This cycle is valid when $DPR_L \neq$ "00_{16}".

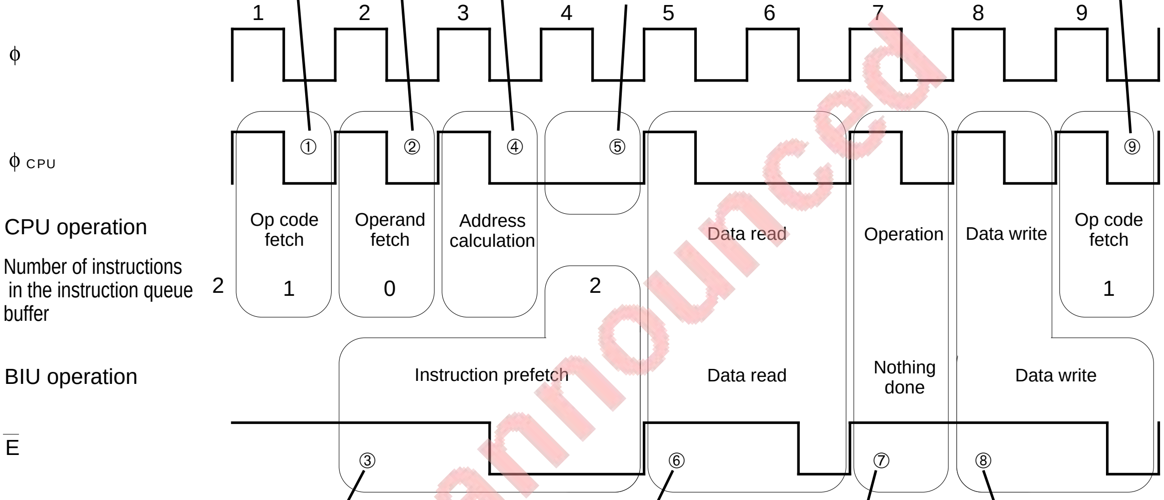
Fig. 5.2.1 CPU instruction execution sequence (in ASL instruction and direct addressing mode)

5.2 Points of view

Instruction execution condition

•High-speed running, using internal area (The bus cycle is 3 ϕ cycles when accessing ROM and 2 ϕ cycles when accessing RAM.)
•Data read/write to RAM
•Start address of access: Even address to ROM or RAM
•Number of instructions in the instruction queue buffer at beginning to execute an instruction: 2 bytes

Thinking number of instruction cycles



① The CPU fetches an op code in the instruction queue buffer. The number of instructions in the instruction queue buffer becomes 1 byte.

② The CPU fetches an operand in the instruction queue buffer. The number of instructions in the instruction queue buffer becomes 0 byte.

④ The CPU does not use bus.

⑤ Wait until instruction prefetch is completed, that is, bus can be used, because next operation is data read.

⑨ The CPU fetches an op code of next instruction prefetched in the instruction queue buffer while the BIU writes the data. After that, next instructions are executed. (Writing is performed until the 9th cycle. The ASL instruction is completed with 8 cycles.

③ The BIU prefetches instructions because the number of instructions in the instruction queue buffer become 1 byte at the first cycle.
2 bytes instructions are prefetched at one access because the start address of access is an even.
The instructions are read from the ROM, so that 1 bus cycle becomes 3 ϕ.

⑦ The CPU does not use bus. However, The BIU does not prefetch instructions because the number of instructions in the instruction queue buffer is 2 bytes.

⑧ Write data. The CPU transfers the data to the BIU and specifies the write address.
The BIU writes it once with 1 bus cycle = 2 ϕ as well as reading.

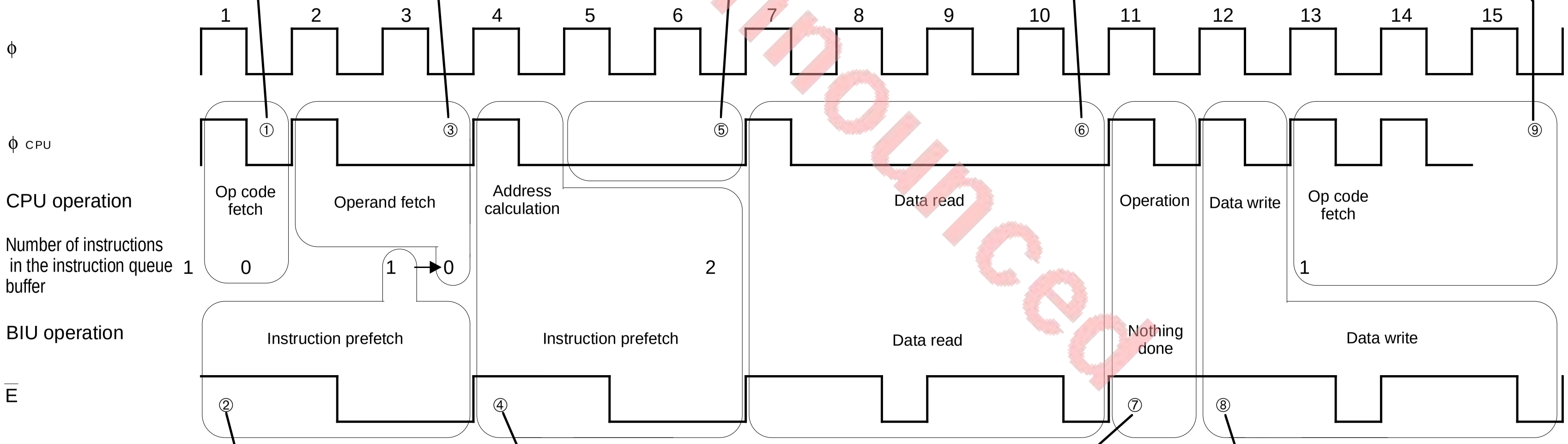
⑥ Read the data.
16-bit data is read once because the start address of access is an even. The data is read from the RAM, so that 1 bus cycle becomes 2 ϕ.

Fig. 5.2.2 Idea example of number of instruction cycles (1)

Instruction execution condition

- High-speed running, using internal area (The bus cycle is 3 ϕ cycles when accessing ROM and 2 ϕ cycles when accessing RAM.)
- Data read/write to RAM
- Start address of access: Odd address to ROM or RAM
- Number of instructions in the instruction queue buffer at beginning to execute an instruction: 1 byte

Thinking number of instruction cycles



① The CPU fetches an op code in the instruction queue buffer. The number of instructions in the instruction queue buffer becomes 0 byte.

② The BIU prefetches instructions because the number of instructions in the instruction queue buffer is 1 byte at the cycle before execution of ASL instruction.
1 byte instruction is prefetched at one access because the start address of access is an odd. The instructions are read from the ROM, so that 1 bus cycle becomes 3ϕ.

③ The CPU stands by because the number of instructions in the instruction queue buffer is 0 byte. When instructions are placed in the instruction queue buffer, the CPU fetches an operand in the instruction queue buffer. The number of instructions in the instruction queue buffer becomes 0 byte.

④ The CPU does not use bus.
The BIU prefetches instructions because the number of instructions in the instruction queue buffer is 0 byte.
2 bytes instructions are prefetched at one access because the start address of access is an even.

⑤ Wait until instruction prefetch is completed, that is, bus can be used, because next operation is read.

⑥ Read the data.
16-bit data is read with twice because the start address of access is an odd. The data is read from the RAM, so that 1 bus cycle becomes 2 ϕ.

⑦ The CPU does not use bus. However, The BIU does not prefetch instructions because the number of instructions in the instruction queue buffer is 2 bytes.

⑧ Write data. The CPU transfers the data to the BIU and specifies the write address.
The BIU writes it with twice and 1 bus cycle = 2 ϕ as well as reading.

⑨ The CPU fetches an op code of next instruction prefetched in the instruction queue buffer while the BIU writes the data. After that, next instructions are executed. (Writing is performed until the 15th cycle.
The ASL instruction is completed with 12 cycles.

Fig. 5.2.3 Idea example of number of instruction cycles (2)

5.2 Points of view

5.2.1 Using internal area

Tables 5.2.1 – 5.2.8 show the number of instruction cycles for each instruction when using the internal area.

That value of the table is calculated by almost matching the number of instructions in the instruction queue buffer at beginning to execute the instruction to that at its completion. Accordingly, execution cycles of the routine can be estimated by adding the number of instruction cycles of each instruction.

However, evaluate it with an actual system when an accurate value is necessary because it is a rough estimate.

The instruction execution condition of Tables 5.2.1 – 5.2.8 is described as the following.

(1) Instruction execution condition of Tables 5.2.1 – 5.2.4

●Low-order byte of the direct page register (DPRL) = "00_{16}"

●Low-speed running (The bus cycle is 2ϕ when the ROM is accessed or when the RAM is done)

●Data read/write to the RAM

●Start address of access: Even address to the ROM or the RAM

●The number of instructions in the instruction queue buffer at beginning to execute an instruction: 1 byte
(However, add "time required to place the first byte of next instruction – 1 cycle" to it when the instruction execution is completed at the state where there are no instructions in the instruction queue buffer.

(2) Instruction execution condition of Tables 5.2.5 – 5.2.8

●Low-order byte of the direct page register (DPRL) = "00_{16}"

●High-speed running (The bus cycle is 3ϕ when the ROM is accessed, and 2ϕ when the RAM is done)

●Data read/write to the RAM

●Start address of access: Even address to the ROM or the RAM

●The number of instructions in the instruction queue buffer at beginning to execute an instruction: 1 byte
(However, add "time required to place the first byte of next instruction – 1 cycle" to it when the instruction execution is completed at the state where there are no instructions in the instruction queue buffer.

Table 5.2.1 Number of instruction cycles in low-speed running (1)

Instruction symbol \ Addressing mode		IMP	IMM	A	DIR	DIR, b	DIR, X	DIR, Y	(DIR)	(DIR,X)	(DIR),Y	L(DIR)	L(DIR), Y	ABS	ABS, b	ABS, X	ABS, Y	ABL	ABL, X	(ABS)	L(ABS)	(ABS, X)	STK	REL	DIR, b, R	ABS,b, R	SR	(SR), Y	BLK	Multiplied accumulation
ADC	Accumulator A		3		4		6		6	8	8	8	10	5		6	6	6	8								6	9		
	Accumulator B		5		7		7		9	9	10	11	12	7		9	9	9	9								7	10		
AND	Accumulator A		3		4		6		6	8	8	8	10	5		6	6	6	8								6	9		
	Accumulator B		5		7		7		9	9	10	11	12	7		9	9	9	9								7	10		
ASL	Accumulator A			2	7		8							7		8														
	Accumulator B			4																										
ASR	Accumulator A			4	9		9							10		11														
	Accumulator B			4																										
BBC	Branch																								9	11				
	No branch																								8	9				
BBS	Branch																								9	11				
	No branch																								8	9				
BCC	Branch																							5						
	No branch																							4						
BCS	Branch																							5						
	No branch																							4						
BEQ	Branch																							5						
	No branch																							4						
BMI	Branch																							5						
	No branch																							4						
BNE	Branch																							5						
	No branch																							4						
BPL	Branch																							5						
	No branch																							4						
BRA																								4						
BRK		16																												
BVC	Branch																							5						
	No branch																							4						
BVS	Branch																							5						
	No branch																							4						
CLB						9									9															
CLC		2																												
CLI		2																												
CLM		2																												
CLP			4																											
CLV		2																												
CMP	Accumulator A		3		4		6		6	8	8	8	10	5		6	6	6	8								6	9		
	Accumulator B		5		7		7		9	9	10	11	12	7		9	9	9	9								7	9		
CPX			3		4									5																
CPY			3		4									5																
DEC	Accumulator A			2	7		8							7		8														
	Accumulator B			4																										
DEX		2																												
DEY		2																												
DIV	m = “0”		30		32		32		34	34	35	36	37	32		34	34	34	34								32	35		
	m = “1”		22		24		24		26	26	27	28	29	24		26	26	26	26								24	27		

5.2 Points of view

Table 5.2.2 Number of instruction cycles in low-speed running (2)

Instruction symbol \ Addressing mode		IMP	IMM	A	DIR	DIR, b	DIR, X	DIR, Y	(DIR)	(DIR,X)	(DIR),Y	L(DIR)	L(DIR), Y	ABS	ABS, b	ABS, X	ABS, Y	ABL	ABL, X	(ABS)	L(ABS)	(ABS, X)	STK	REL	DIR, b, R	ABS, b, R	SR	(SR), Y	BLK	Multiplied accumulation
DIVS	m = "0"		32		34		34		36	36	37	38	39	34		36	36	36	36								34	37		
	m = "1"		24		26		26		28	28	29	30	31	26		28	28	28	28								26	29		
EOR	Accumulator A		3		4		6		6	8	8	8	10	5		6	6	6	8								6	9		
	Accumulator B		5		7		7		9	9	10	11	12	7		9	9	9	9								7	10		
EXTS	Accumulator A			4																										
	Accumulator B			4																										
EXTZ	Accumulator A			4																										
	Accumulator B			4																										
INC	Accumulator A			2	7		8							7		8														
	Accumulator B			4																										
INX		2																												
INY		2																												
JMP														3				5		5	7	7								
JSR														8				9				9								
LDA	Accumulator A		3		4		6		6	8	8	8	10	5		6	6	6	8								6	9		
	Accumulator B		5		7		7		9	9	10	11	12	7		9	9	9	9								7	10		
LDM					5		7							7		7														
LDT			5																											
LDX			3		4			6						5			6													
LDY			3		4		6							5		6														
LSR	Accumulator A			2	7		8							7		8														
	Accumulator B			4																										
MPY	m = "0"		13		15		15		17	17	18	19	20	15	17		17	17	17	17							15	18		
	m = "1"		8		11		11		13	13	14	15	16	11	13		13	13	13	13							11	14		
MPYS	m = "0"		13		15		15		17	17	18	19	20	15	17		17	17	17	17							15	18		
	m = "1"		8		11		11		13	13	14	15	16	11	13		13	13	13	13							11	14		
MVN	0 byte transfer																												5	
	1 byte transfer																												11	
	2 or more even bytes transfer																												✻1	
	3 or more odd bytes transfer																												✻2	
MVP	0 byte transfer																												5	
	1 byte transfer																												11	
	2 or more even bytes transfer																												✻3	
	3 or more odd bytes transfer																												✻4	
NOP		2																												
ORA	Accumulator A		3		4		6		6	8	8	8	10	5		6	6	6	8								6	9		
	Accumulator B		5		7		7		9	9	10	11	12	7		9	9	9	9								7	10		
PEA																							5							
PEI																							7							

✻1: $5 + \frac{i}{2} \times 7$ (i = number of transfer bytes)

✻2: $11 + \frac{i-1}{2} \times 7$ (i = number of transfer bytes)

✻3: $9 + \frac{i}{2} \times 7$ (i = number of transfer bytes)

✻4: $17 + \frac{i-1}{2} \times 7$ (i = number of transfer bytes)

Table 5.2.3 Number of instruction cycles in low-speed running (3)

Instruction symbol \ Addressing mode		IMP	IMM	A	DIR	DIR, b	DIR, X	DIR, Y	(DIR)	(DIR,X)	(DIR),Y	L(DIR)	L(DIR), Y	ABS	ABS, b	ABS, X	ABS, Y	ABL	ABL, X	(ABS)	L(ABS)	(ABS, X)	STK	REL	DIR, b, R	ABS, b, R	SR	(SR), Y	BLK	Multiplied accumulation
PER																							5							
PHA																							4							
PHB																							6							
PHD																							4							
PHG																							3							
PHP																							4							
PHT																							3							
PHX																							4							
PHY																							4							
PLA																							5							
PLB																							7							
PLD																							5							
PLP																							6							
PLT																							6							
PLX																							5							
PLY																							5							
PSH																							✻1							
PUL																							✻2							
RMPA	m = “0”																													✻3
	m = “1”																													✻4
RLA			✻5																											
ROL	Accumulator A			2	7		8							7		8														
	Accumulator B			4																										
ROR	Accumulator A			2	7		8							7		8														
	Accumulator B			4																										
RTI		10																												
RTL		8																												
RTS		6																												
SBC	Accumulator A		3		4		6		6	8	8	8	10	5		6	6	6	8								6	9		
	Accumulator B		5		7		7		9	9	10	11	12	7		9	9	9	9								7	10		
SEB						9									9															
SEC		2																												
SEI		2																												
SEM		2																												
SEP			3																											
STA	Accumulator A				5		5		7	8	7	9	9	5		5	5	7	7								5	9		
	Accumulator B				6		7		9	9	10	11	12	8		8	8	8	9								7	10		

✻ 1: 11 + 2 i_1 + i_2
(i_1 = number of registers saved among A, B, X, Y, DPR and PS; i_2 = number of registers saved of DT and PG)

✻ 2: 12 + 3 i_1 + 4 i_2
(i_1 = number of registers restored among A, B, X, Y, DT and PS; i_2 = 1 when restoring DPR and 0 when not doing)

✻ 3: 6 + 20 i (i = number of repeated operations)

✻ 4: 6 + 16 i (i = number of repeated operations)

✻ 5: 7 + i (i = number of rotations)

Table 5.2.4 Number of instruction cycles in low-speed running (4)

Instruction symbol \ Addressing mode	IMP	IMM	A	DIR	DIR, b	DIR, X	DIR, Y	(DIR)	(DIR,X)	(DIR),Y	L(DIR)	L(DIR), Y	ABS	ABS, b	ABS, X	ABS, Y	ABL	ABL, X	(ABS)	L(ABS)	(ABS, X)	STK	REL	DIR, b, R	ABS, b, R	SR	(SR), Y	BLK	Multiplied accumulation
STP	–																												
STX				5			5						5																
STY				5		5							5																
TAD	2																												
TAS	2																												
TAX	2																												
TAY	2																												
TBD	4																												
TBS	4																												
TBX	4																												
TBY	4																												
TDA	2																												
TDB	4																												
TSA	2																												
TSB	4																												
TSX	2																												
TXA	2																												
TXB	4																												
TXS	2																												
TXY	2																												
TYA	2																												
TYB	4																												
TYX	2																												
WIT	–																												
XAB	5																												

Table 5.2.5 Number of instruction cycles in high-speed running (1)

Instruction symbol	Addressing mode	IMP	IMM	A	DIR	DIR, b	DIR, X	DIR, Y	(DIR)	(DIR,X)	(DIR),Y	L(DIR)	L(DIR), Y	ABS	ABS, b	ABS, X	ABS, Y	ABL	ABL, X	(ABS)	L(ABS)	(ABS, X)	STK	REL	DIR, b, R	ABS, b, R	SR	(SR), Y	BLK	Multiplied accumulation
ADC	Accumulator A		5		5		8		7	10	10	9	12	7		8	8	8	11								8	11		
	Accumulator B		6		8		8		10	10	11	12	13	8		11	11	11	11								8	11		
AND	Accumulator A		5		5		8		7	10	10	9	12	7		8	8	8	11								8	11		
	Accumulator B		6		8		8		10	10	11	12	13	8		11	11	11	11								8	11		
ASL	Accumulator A			2	9		10							9		10														
	Accumulator B			4																										
ASR	Accumulator A			4	10		10							12		13														
	Accumulator B			4																										
BBC	Branch																								13	16				
	No branch																								10	12				
BBS	Branch																								13	16				
	No branch																								10	12				
BCC	Branch																							8						
	No branch																							5						
BCS	Branch																							8						
	No branch																							5						
BEQ	Branch																							8						
	No branch																							5						
BMI	Branch																							8						
	No branch																							5						
BNE	Branch																							8						
	No branch																							5						
BPL	Branch																							8						
	No branch																							5						
BRA																								8						
BRK		18																												
BVC	Branch																							8						
	No branch																							5						
BVS	Branch																							8						
	No branch																							5						
CLB						12									12															
CLC		2																												
CLI		2																												
CLM		2																												
CLP			5																											
CLV		2																												
CMP	Accumulator A		5		5		8		7	10	10	9	12	7		8	8	8	11								8	11		
	Accumulator B		6		8		8		10	10	11	12	13	8		11	11	11	11								8	11		
CPX			5		5									7																
CPY			5		5									7																
DEC	Accumulator A			2	9		10							9		10														
	Accumulator B			4																										
DEX		2																												
DEY		2																												
DIV	m = “0”		31		33		33		35	35	36	37	38	33		36	36	36	36								33	36		
	m = “1”		23		25		25		27	27	28	29	30	25		28	28	28	28								25	28		

Table 5.2.6 Number of instruction cycles in high-speed running (2)

Addressing mode Instruction symbol		IMP	IMM	A	DIR	DIR, b	DIR, X	DIR, Y	(DIR)	(DIR,X)	(DIR),Y	L(DIR)	L(DIR), Y	ABS	ABS, b	ABS, X	ABS, Y	ABL	ABL, X	(ABS)	L(ABS)	(ABS, X)	STK	REL	DIR, b, R	ABS, b, R	SR	(SR), Y	BLK	Multiplied accumulation
DIVS	m = “0”		33		35		35		37	37	38	39	40	35		38	38	38	38								35	38		
	m = “1”		25		27		27		29	29	30	31	32	27		30	30	30	30								27	30		
EOR	Accumulator A		5		5		8		7	10	10	9	12	7		8	8	8	11								8	11		
	Accumulator B		6		8		8		10	10	11	12	13	8		11	11	11	11								8	11		
EXTS	Accumulator A			4																										
	Accumulator B			4																										
EXTZ	Accumulator A			4																										
	Accumulator B			4																										
INC	Accumulator A			2	9		10							9		10														
	Accumulator B			4																										
INX		2																												
INY		2																												
JMP														5				8		7	11	10								
JSR														10				12				12								
LDA	Accumulator A		5		5		8		7	10	10	9	12	7		8	8	8	11								8	11		
	Accumulator B		6		8		8		10	10	11	12	13	8		11	11	11	11								8	11		
LDM					7		10							10		10														
LDT			5																											
LDX			5		5			8						7			8													
LDY			5		5		8							7		8														
LSR	Accumulator A			2	9		10							9		10														
	Accumulator B			4																										
MPY	m = “0”		14		16		16		18	18	19	20	21	16		19	19	19	19								16	19		
	m = “1”		10		12		12		14	14	15	16	17	12		15	15	15	15								12	15		
MPYS	m = “0”		14		16		16		18	18	19	20	21	16		19	19	19	19								16	19		
	m = “1”		10		12		12		14	14	15	16	17	12		15	15	15	15								12	15		
MVN	0 byte transfer																												6	
	1 byte transfer																												12	
	2 or more even bytes transfer																												✻1	
	3 or more odd bytes transfer																												✻2	
MVP	0 byte transfer																												6	
	1 byte transfer																												12	
	2 or more even bytes transfer																												✻3	
	3 or more odd bytes transfer																												✻4	
NOP		2																												
ORA	Accumulator A		5		5		8		7	10	10	9	12	7		8	8	8	11								8	11		
	Accumulator B		6		8		8		10	10	11	12	13	8		11	11	11	11								8	11		
PEA																							7							
PEI																							9							

✻ 1: $6 + \frac{i}{2} \times 7$ (i = number of transfer bytes)

✻ 2: $12 + \frac{i-1}{2} \times 7$ (i = number of transfer bytes)

✻ 3: $10 + \frac{i}{2} \times 7$ (i = number of transfer bytes)

✻ 4: $18 + \frac{i-1}{2} \times 7$ (i = number of transfer bytes)

Table 5.2.7 Number of instruction cycles in high-speed running (3)

Instruction symbol \ Addressing mode		IMP	IMM	A	DIR	DIR, b	DIR, X	DIR, Y	(DIR)	(DIR,X)	(DIR),Y	L(DIR)	L(DIR), Y	ABS	ABS, b	ABS, X	ABS, Y	ABL	ABL, X	(ABS)	L(ABS)	(ABS, X)	STK	REL	DIR, b, R	ABS, b, R	SR	(SR), Y	BLK	Multiplied accumulation
PER																							7							
PHA																							4							
PHB																							7							
PHD																							4							
PHG																							4							
PHP																							4							
PHT																							4							
PHX																							4							
PHY																							4							
PLA																							5							
PLB																							8							
PLD																							5							
PLP																							6							
PLT																							6							
PLX																							5							
PLY																							5							
PSH																							✻1							
PUL																							✻2							
RMPA	m = “0”																													✻3
	m = “1”																													✻4
RLA			✻5																											
ROL	Accumulator A			2	9		10							9		10														
	Accumulator B			4																										
ROR	Accumulator A			2	9		10							9		10														
	Accumulator B			4																										
RTI		11																												
RTL		9																												
RTS		7																												
SBC	Accumulator A		5		5		8		7	10	10	9	12	7		8	8	8	11								8	11		
	Accumulator B		6		8		8		10	10	11	12	13	8		11	11	11	11								8	11		
SEB						12									12															
SEC		2																												
SEI		2																												
SEM		2																												
SEP			4																											
STA	Accumulator A				7		7		9	10	9	11	11	7		7	7	10	10								7	11		
	Accumulator B				7		7		10	10	11	12	13	10		10	10	10	10								7	11		

✻ 1: 12 + 2 i_1 + i_2
(i_1 = number of registers saved among A, B, X, Y, DPR and PS; i_2 = number of registers saved of DT and PG)

✻ 2: 14 + 3 i_1 + 4 i_2
(i_1 = number of registers restored among A, B, X, Y, DT and PS; i_2 = 1 when restoring DPR and 0 when not doing)

✻ 3: 6 + 20 i (i = number of repeated operations)

✻ 4: 6 + 16 i (i = number of repeated operations)

✻ 5: 8 + i (i = number of rotations)

5.2 Points of view

Table 5.2.8 Number of instruction cycles in high-speed running (4)

Addressing mode / Instruction symbol	IMP	IMM	A	DIR	DIR, b	DIR, X	DIR, Y	(DIR)	(DIR,X)	(DIR),Y	L(DIR)	L(DIR), Y	ABS	ABS, b	ABS, X	ABS, Y	ABL	ABL, X	(ABS)	L(ABS)	(ABS, X)	STK	REL	DIR, b, R	ABS, b, R	SR	(SR), Y	BLK	Multiplied accumulation
STP	–																												
STX				7			7						7																
STY				7		7							7																
TAD	2																												
TAS	2																												
TAX	2																												
TAY	2																												
TBD	4																												
TBS	4																												
TBX	4																												
TBY	4																												
TDA	2																												
TDB	4																												
TSA	2																												
TSB	4																												
TSX	2																												
TXA	2																												
TXB	4																												
TXS	2																												
TXY	2																												
TYA	2																												
TYB	4																												
TYX	2																												
WIT	–																												
XAB	5																												

CHAPTER 6

CPU INSTRUCTION EXECUTION SEQUENCE FOR EACH ADDRESSING MODE

CPU INSTRUCTION EXECUTION SEQUENCE FOR EACH ADDRESSING MODE

The following are the CPU instruction execution sequences for each addressing mode. The execution sequences shown here describe the internal operation of the CPU. Accordingly, the signals are all CPU internal signals, and cannot be observed from outside. The CPU internal operation, the actual execution time, and the relation between signals that can be externally checked are described in “Chapter 5 NUMBER OF INSTRUCTION CYCLES”.

The accumulator used in the instructions in the CPU instruction execution sequence is accumulator A. When accumulator B is used, the execution cycle has the two ϕ_{CPU} more of a “42_{16}” that indicates accumulator B, and an internal processing cycle added at the front. (Refer to the figure on page 6–4.)

In the cases of instructions ASR, EXTS and EXTZ, however, the number of cycles using accumulator B is the same.

The number of ϕ_{CPU} cycles differs in the addressing mode that uses the direct page register, according to whether the low-order 8 bits (DPR_L) are “00_{16}” or others. The number of cycles when DPR_L = “00_{16}” is 1 ϕ_{CPU} cycle (address calculation cycle) less than that when $DPR_L \neq$ “00_{16}”.

The number of cycles differs in the PSH and PUL instructions according to the number and type of registers stored in (restored from) the stack.

The number of cycles differs in the block transfer instructions (MVN, MVP), according to the number of a transfer data. The following table shows the signals and the symbols indicating the contents.

CPU INSTRUCTION EXECUTION SEQUENCE FOR EACH ADDRESSING MODE

Symbol	Description
ϕ_{CPU}	CPU basic cycle
$A_{H(CPU)}$	High-order 8 bits of the CPU internal address bus.
$A_MA_{L(CPU)}$	Low-order 16 bits of the CPU internal address bus.
$DATA_{(CPU)}$	The CPU internal data bus.
$R/\overline{W}_{(CPU)}$	$R/\overline{W}$ signal to the bus interface unit.
PG	Contents of program bank register just before the instruction is executed.
PC	Contents of program counter just before the instruction is executed.
DT	Contents of data bank register just before the instruction is executed.
DPR	Contents of direct page register just before the instruction is executed.
DPR_L	Contents of the low-order 8 bits of direct page register just before the instruction is executed.
S	Contents of stack pointer just before the instruction is executed.
A	Contents of accumulator A just before the instruction is executed. Its data length is determined by the m flag.
B	Contents of accumulator B just before the instruction is executed. Its data length is determined by the m flag.
X	Contents of index register X just before the instruction is executed. Its data length is determined by the x flag.
Y	Contents of index register Y just before the instruction is executed. Its data length is determined by the x flag.
PS	Contents of processer status register just before the instruction is executed.
m	Contents of the data length flag just before the instruction is executed.
x	Contents of the index register length flag just before the instruction is executed.
AD_H	The valid address high-order 8 bits at indirect addressing modes.
AD_MAD_L	The valid address low-order 16 bits at indirect addressing modes.
DATA	8 bits or 16 bits data. The data length is determined by the m flag or x flag.
New DATA	The data, which is read, modified.
D_HD_L	16 bits data
D_L	8 bits data
imm	8 bits or 16 bits immediate value
dd	Displacement to DPR (8 bits)
rr	Displacement to PC (signed 8 bits)
rr_Hrr_L	Displacement to PC (signed 16 bits)
nn	Displacement to S (8 bits)
hh	The high-order 8 bits address indicated by operand direcly.
mmll	The low-order 16 bits address indicated by operand direcly.

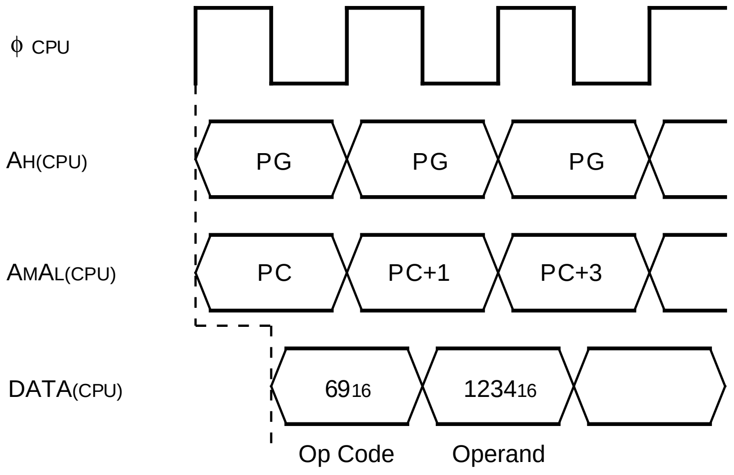
CPU INSTRUCTION EXECUTION SEQUENCE FOR EACH ADDRESSING MODE

Variation of execution cycles according to used accumulator

ADC instruction ; Immediate addressing mode

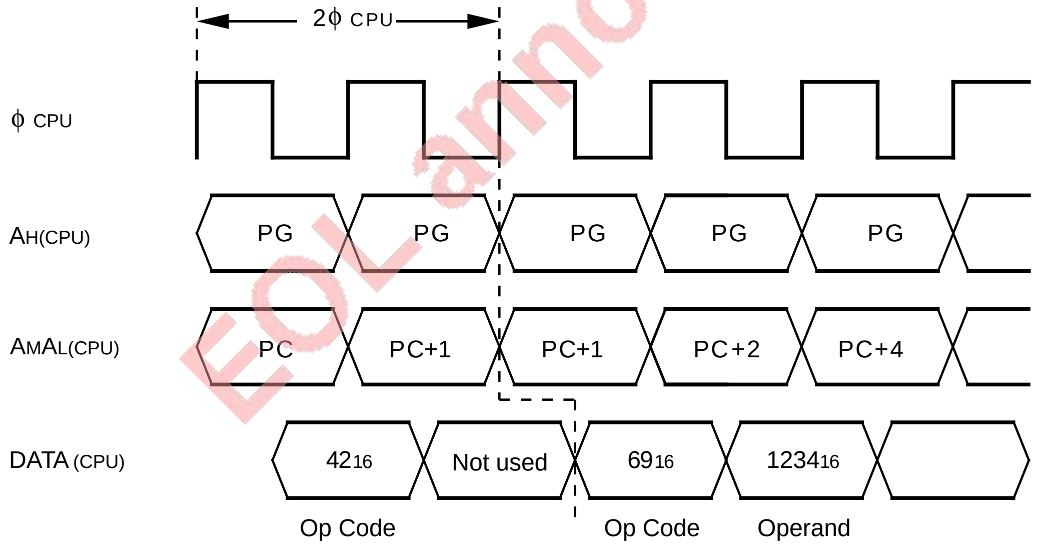
<<Using accumulator A>>

Mnemonic : ADC A,#1234H Machine code : 69_{16} 34_{16} 12_{16}



<<Using accumulator B>>

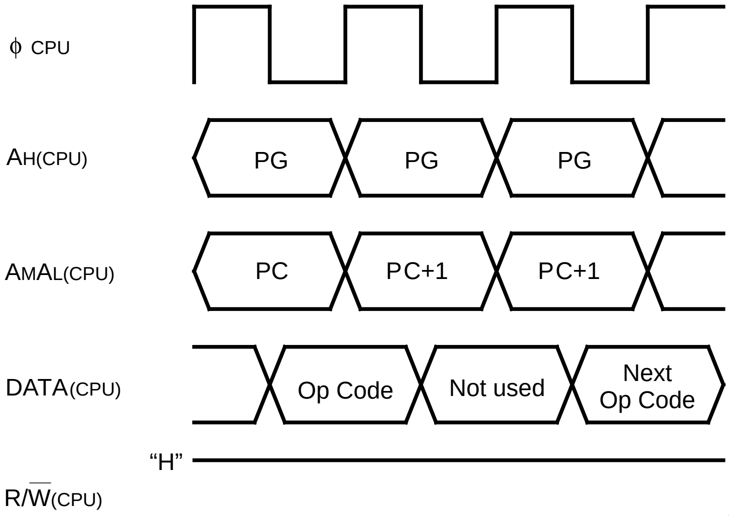
Mnemonic : ADC B,#1234H Machine code : 42_{16} 69_{16} 34_{16} 12_{16}



Implied

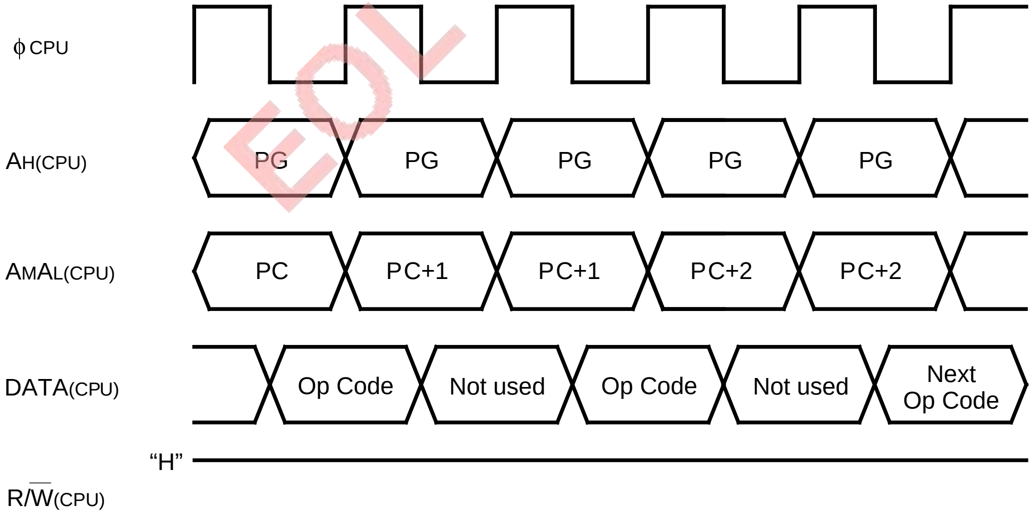
Instructions : CLC, CLI, CLM, CLV, DEX, DEY, INX, INY, NOP,
SEC, SEI, SEM, TAD, TAS, TAX, TAY, TDA, TSA,
TSX, TXA, TXS, TXY, TYA, TYX

Timing :



Instructions : TBD, TBS, TBX, TBY, TDB, TSB, TXB, TYB

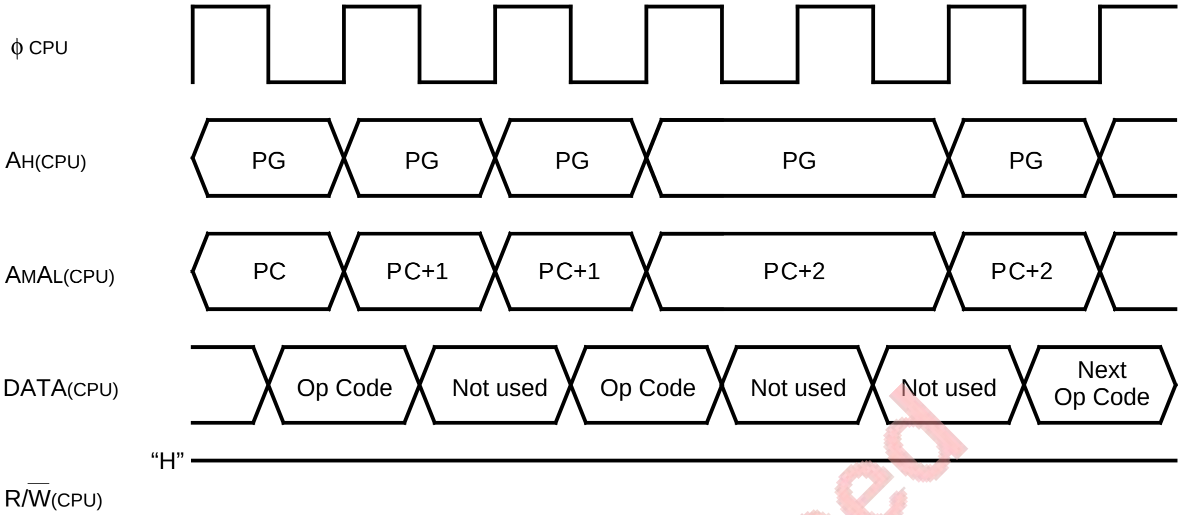
Timing :



Implied

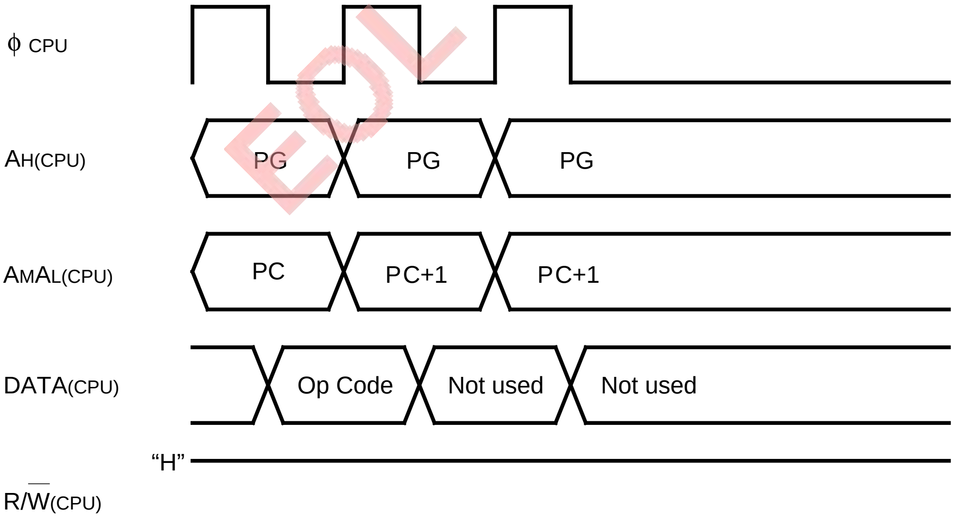
Instructions : XAB

Timing :



Instructions : STP, WIT

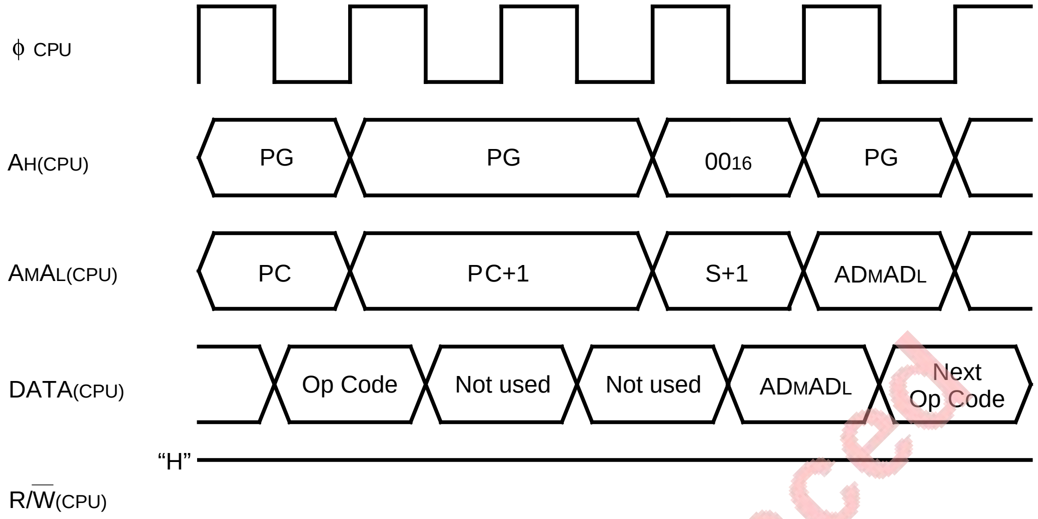
Timing :



Implied

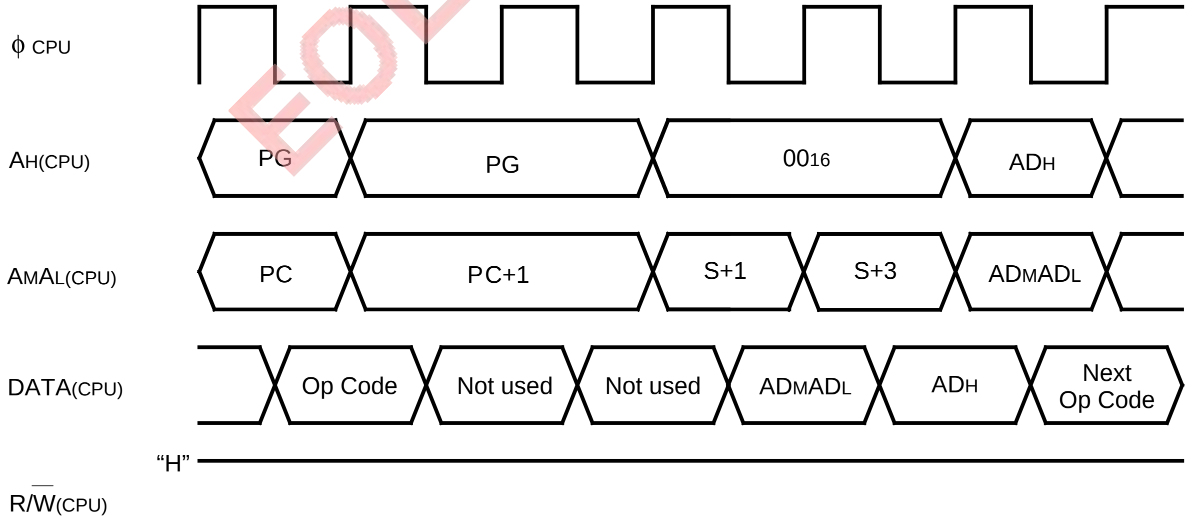
Instructions : RTS

Timing :



Instructions : RTL

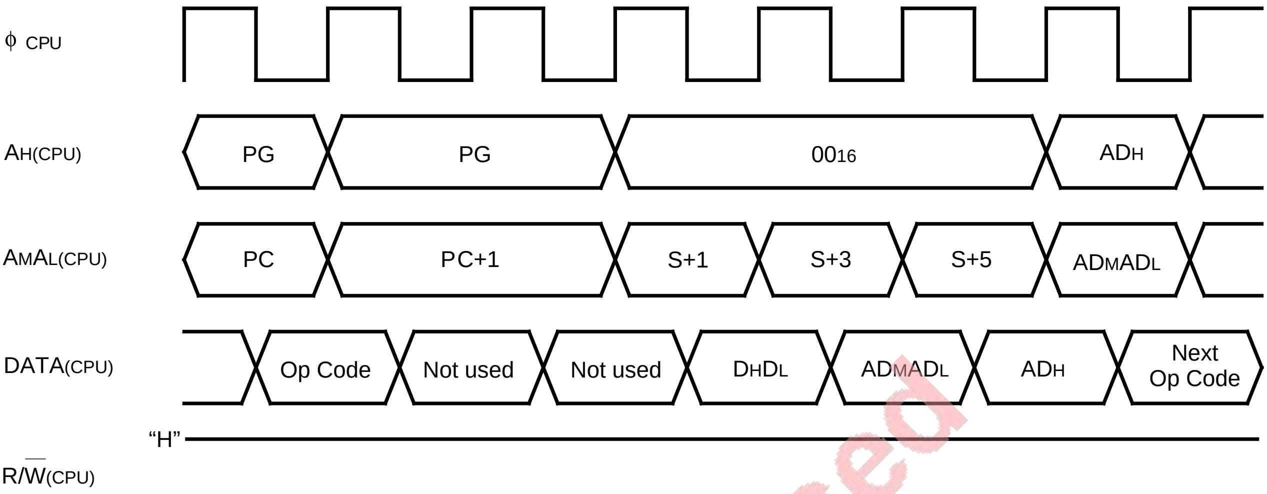
Timing :



Implied

Instructions : RTI

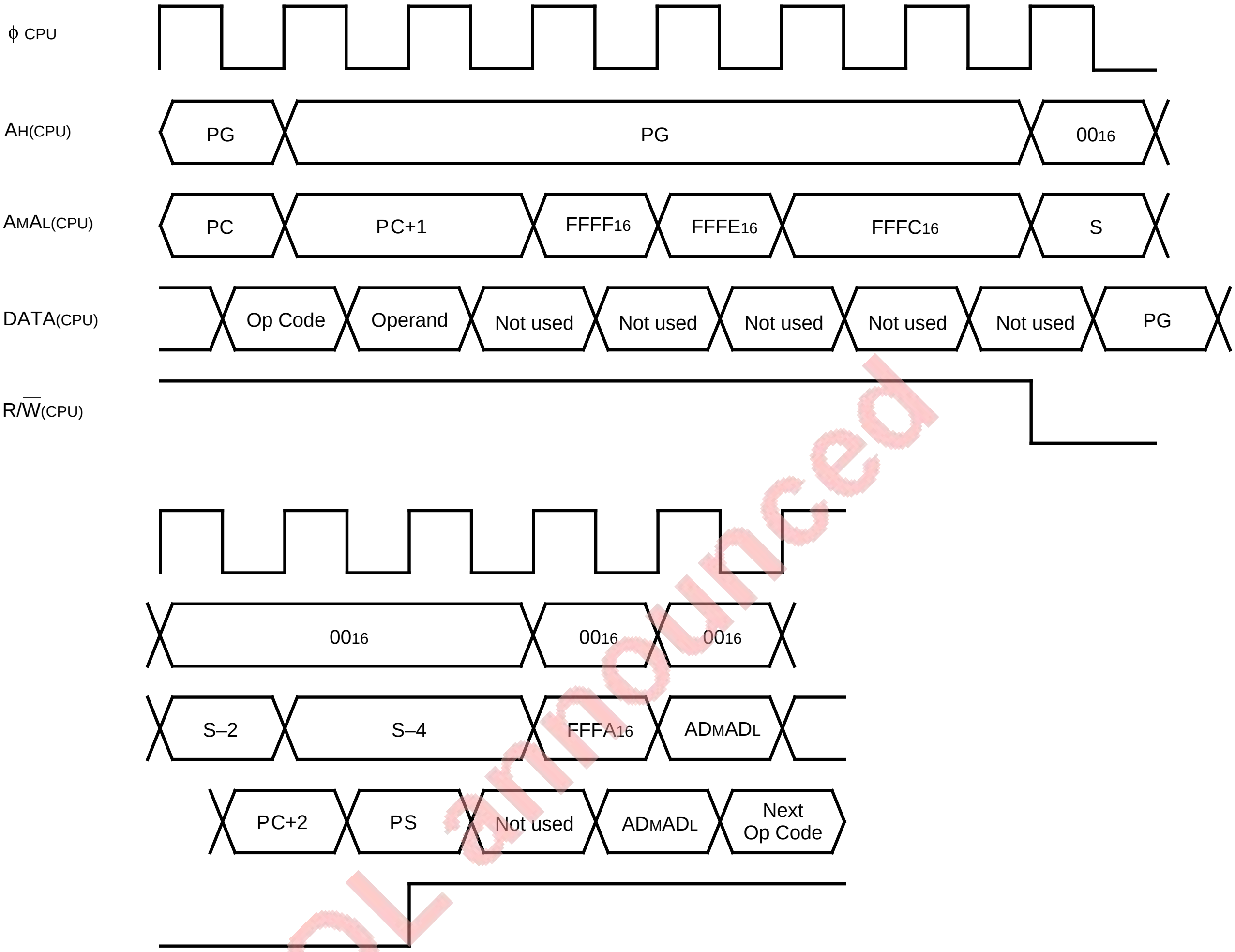
Timing :



Implied

Instructions : BRK

Timing :

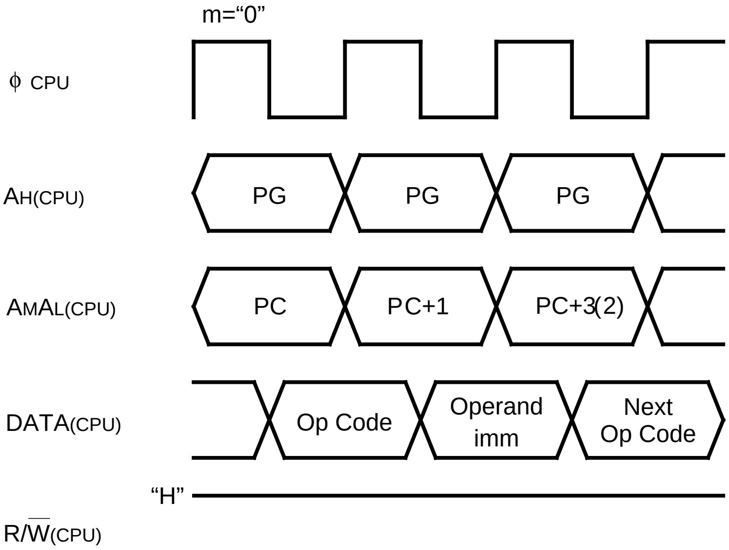


Note: The operand which is fetched at the 2nd cycle is 1 byte. Whatever it contains is available.

Immediate

Instructions : ADC, AND, CMP, EOR, LDA, ORA, SBC

Timing :

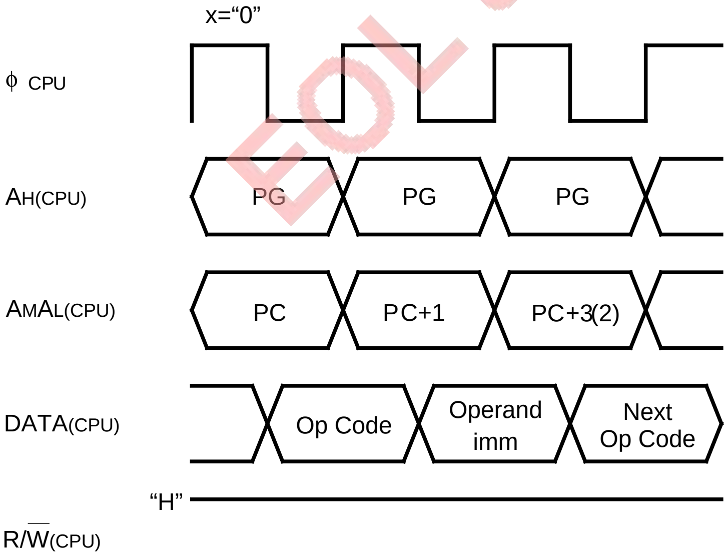


Notes 1: The operand which is fetched at the 2nd cycle is as follows:
When m="0", 2 bytes
When m="1", 1 byte

2: "( )" shows the case of m="1" .

Instructions : LDX, LDY, CPX, CPY

Timing :



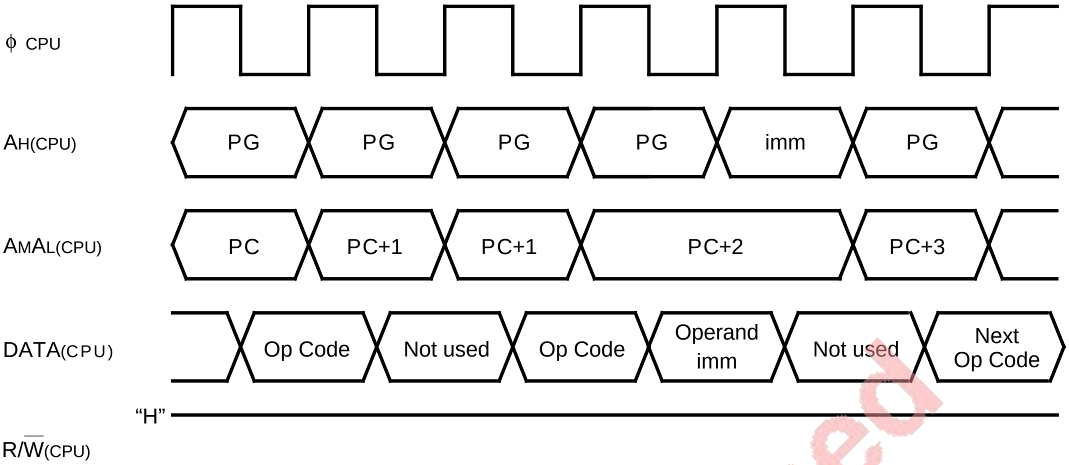
Notes 1: The operand which is fetched at the 2nd cycle is as follows:
When x="0", 2 bytes
When x="1", 1 byte

2: "( )" shows the case of x="1".

Immediate

Instructions : LDT

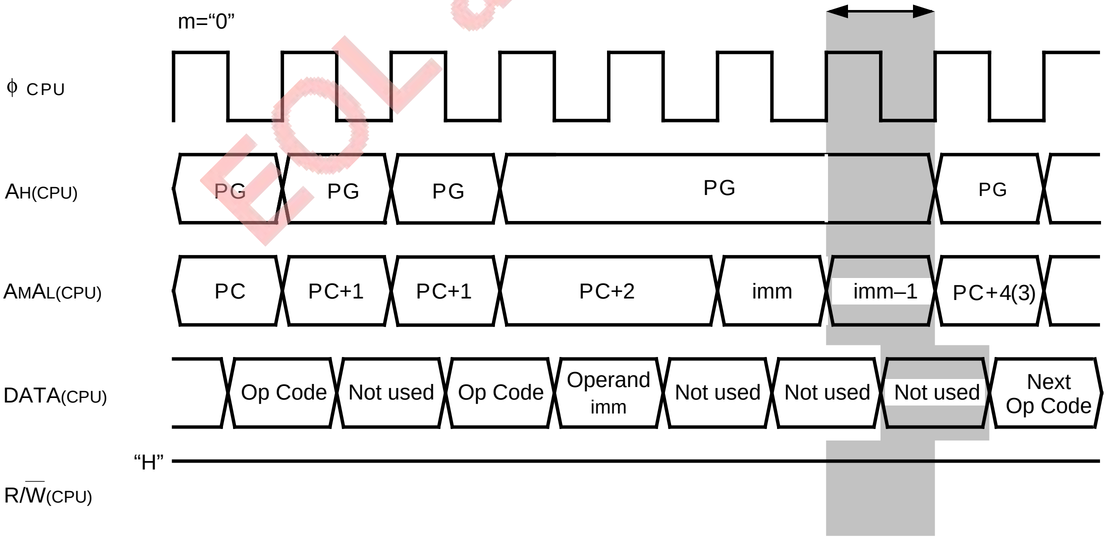
Timing :



Note: The operand at the 4th cycle is 1 byte.

Instructions : RLA

Timing :

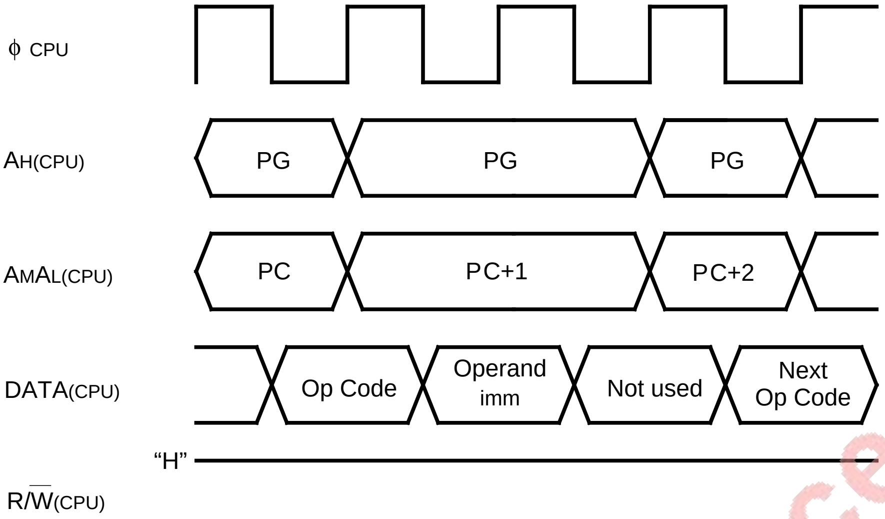


Notes 1: This figure is an example shifted by 1 bit.
When 2 or more bits are shifted, the cycle "◄►" is repeated by each shift times.
When 0 bit is shifted (not shifted), the cycle "◄►" is shortened.
2: The operand which is fetched at the 4th cycle is as follows:
When m="0", 2 bytes
When m="1", 1 byte
3: "( )" shows the case of m="1".

Immediate

Instructions : SEP

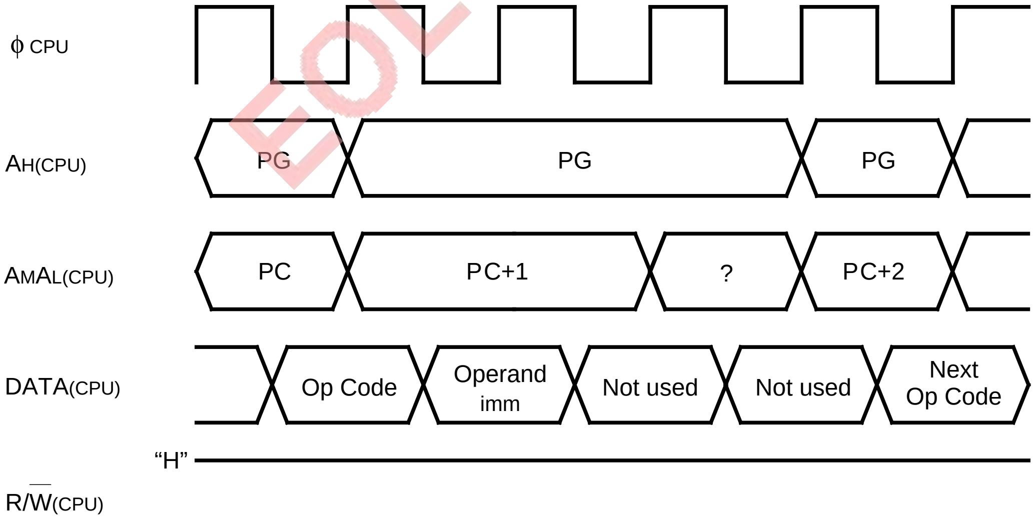
Timing :



Note: The operand at the 2nd cycle is 1 byte.

Instructions : CLP

Timing :

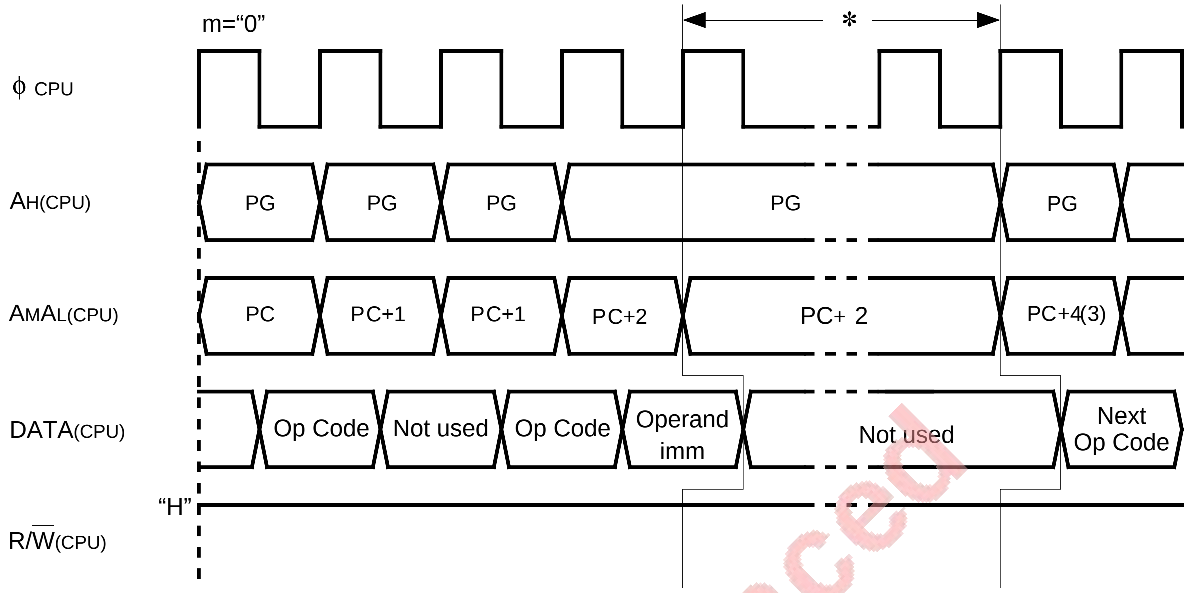


Note: The operand at the 2nd cycle is 1 byte.

Immediate

Instructions : DIV, DIVS, MPY, MPYS

Timing :



Notes 1: The operand which is fetched at the 4th cycle is as follows:
When m="0", 2 bytes
When m="1", 1 byte

2: "(   )" shows the case of m="1".

3: The cycle number during "✻" is shown as the following table:

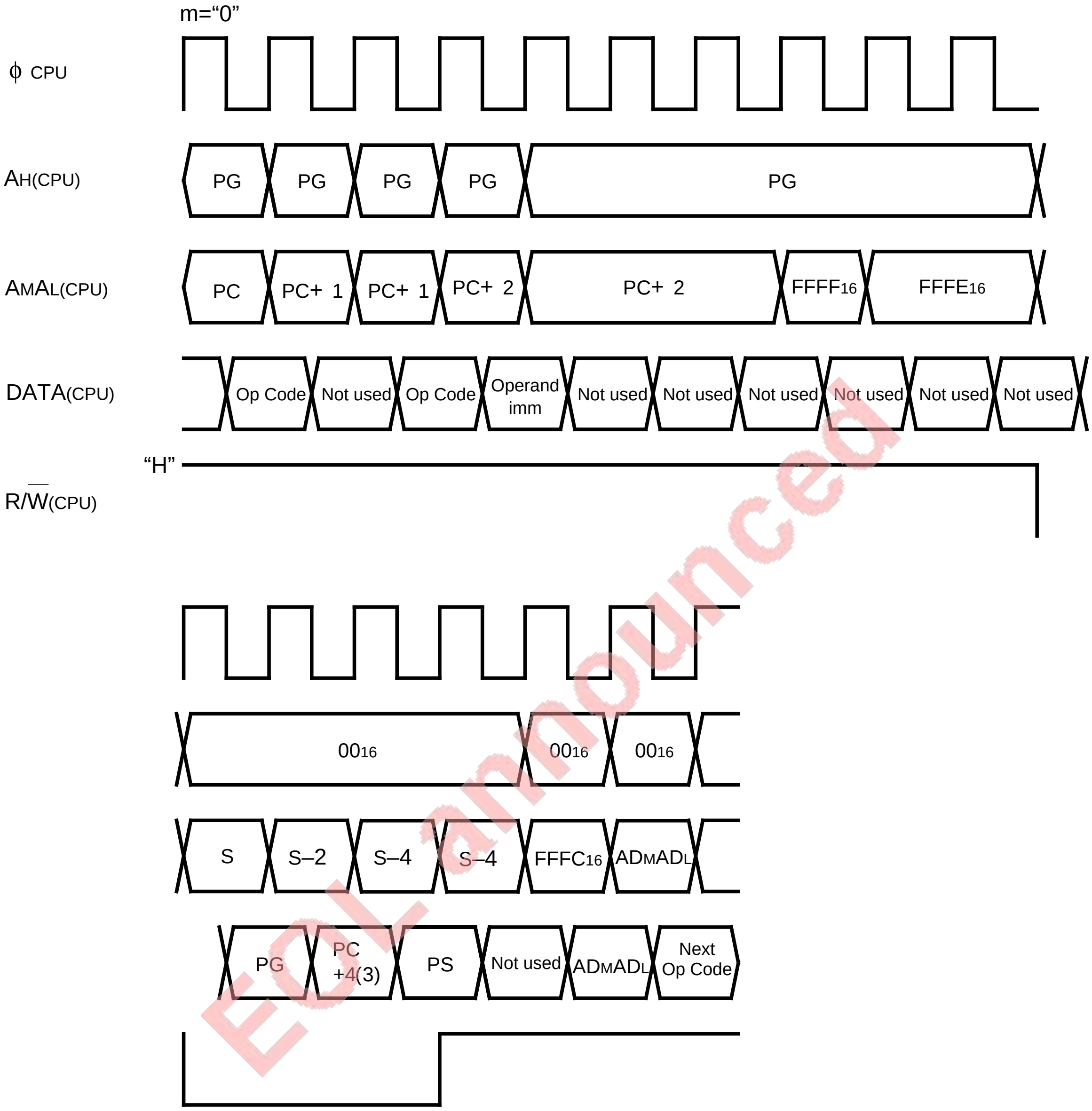
Instruction	Cycle number (ϕ CPU)	
	m= "0"	m = "1"
DIV	25	17
DIVS	27	19
MPY	8	4
MPYS	8	4

- The contents of AMAL(CPU) during "✻" with DIVS are undefined.
- When each sign of a multiplier and a multiplicand with MPYS is different, the contents of AMAL(CPU) at the last 3 of ϕ CPU during "✻" are undefined.

Immediate

Instructions : DIV, DIVS (case of zero division)

Timing :



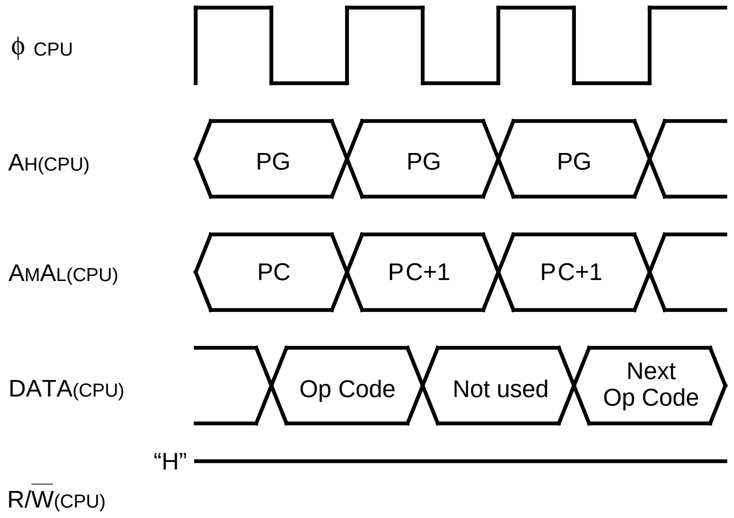
Notes 1: The operand which is fetched at the 4th cycle is as follows:
When m="0", 2 bytes
When m="1", 1 byte

2: "( )" shows the case of m="1".

Accumulator

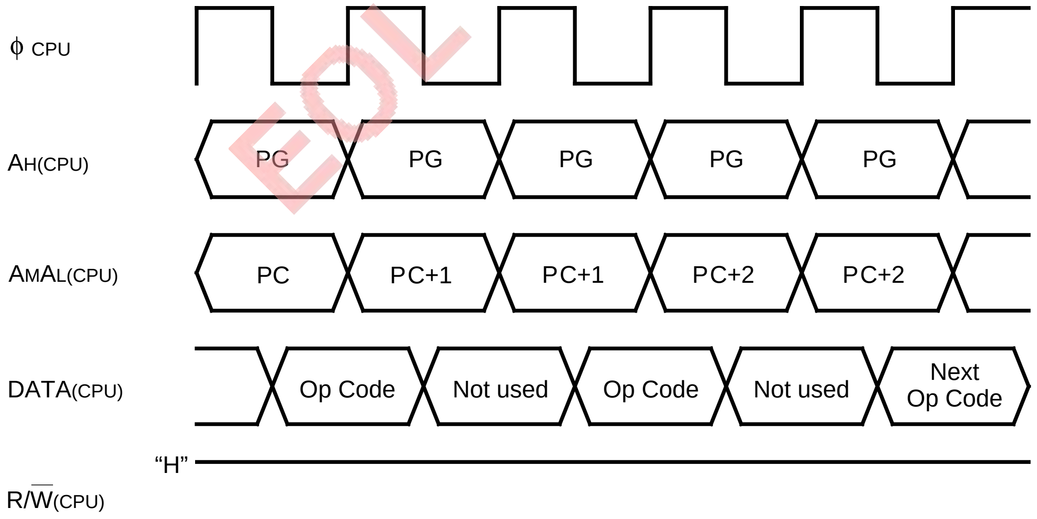
Instructions : ASL, DEC, INC, LSR, ROL, ROR

Timing :



Instructions : ASR, EXTS, EXTZ

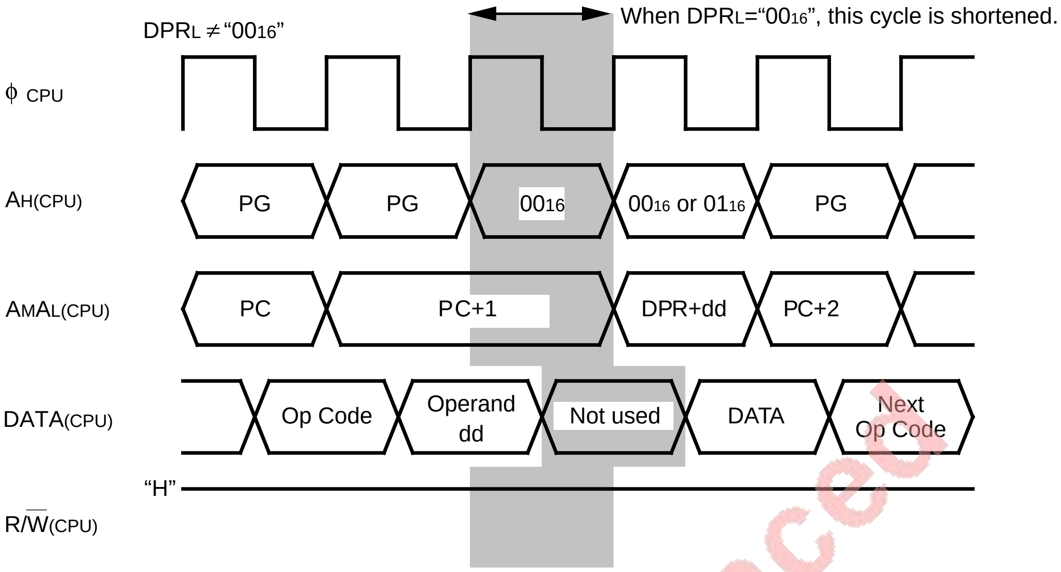
Timing :



Direct

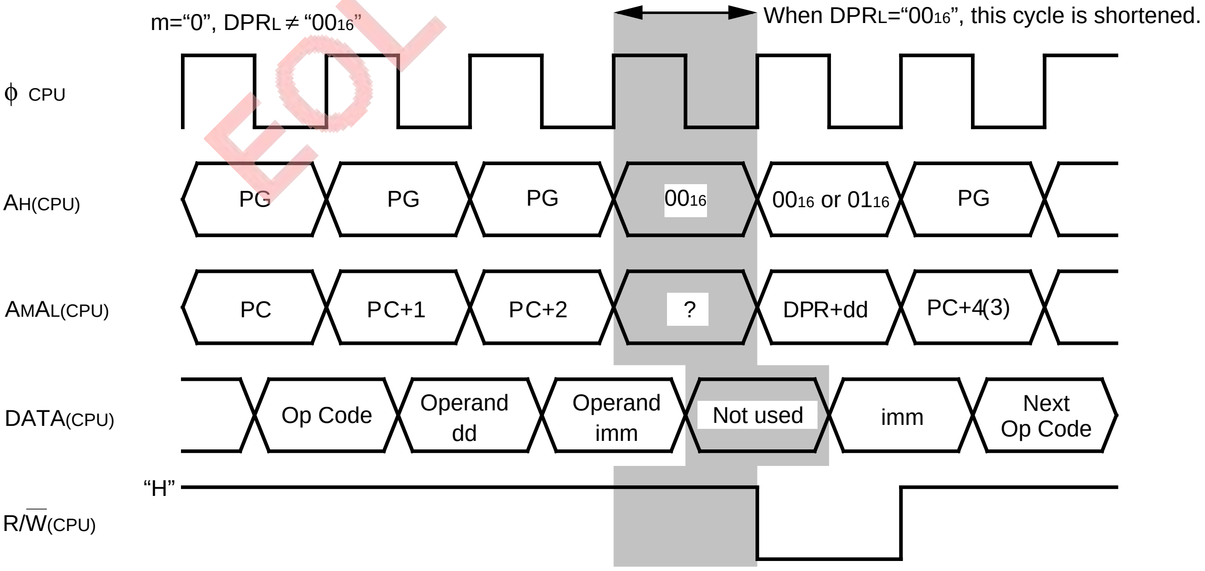
Instructions : ADC, AND, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA, SBC

Timing :



Instructions : LDM

Timing :



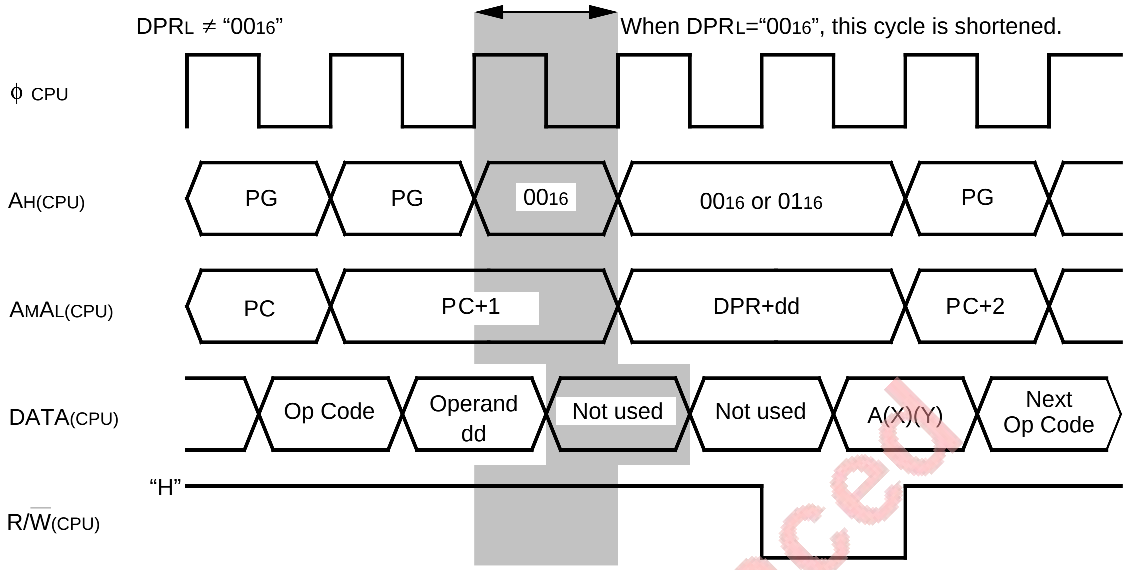
Notes 1: Each of the operand which is fetched at the 3rd cycle and the data which is at the 5th cycle is as follows:
When m="0", 2 bytes
When m="1", 1 byte

2: "(   )" shows the case of m="1".

Direct

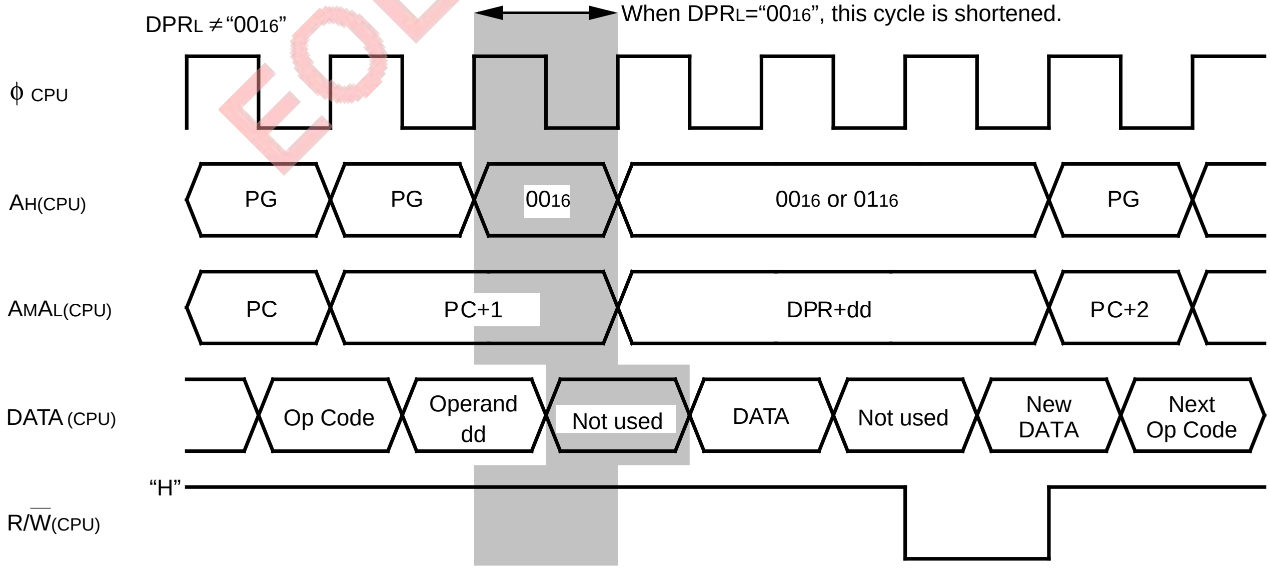
Instructions : STA, STX, STY

Timing :



Instructions : ASL, DEC, INC, LSR, ROL, ROR

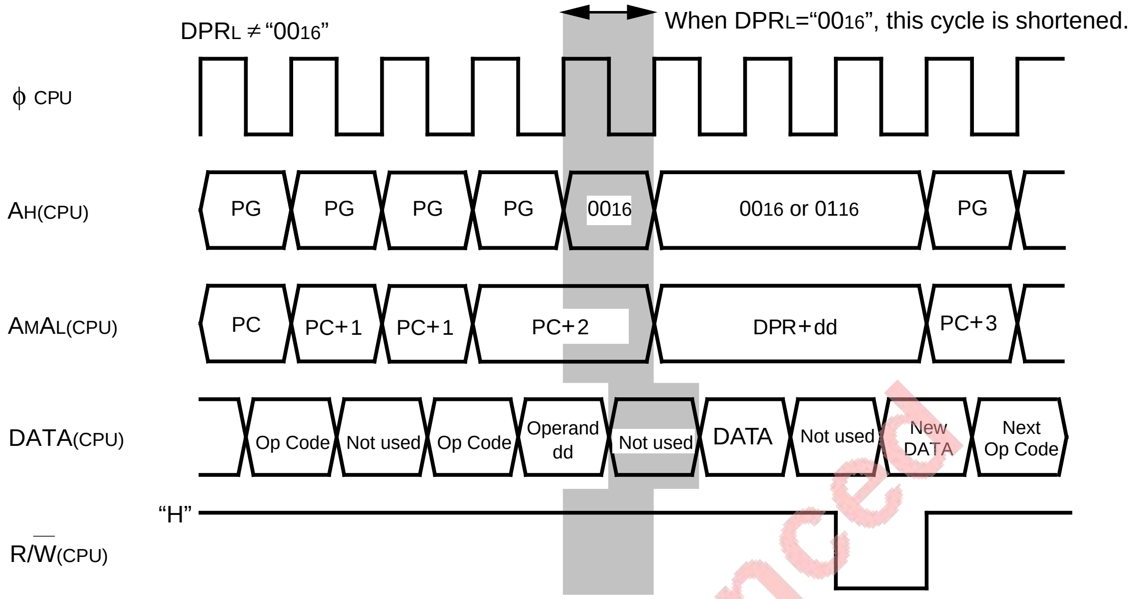
Timing :



Direct

Instructions : ASR

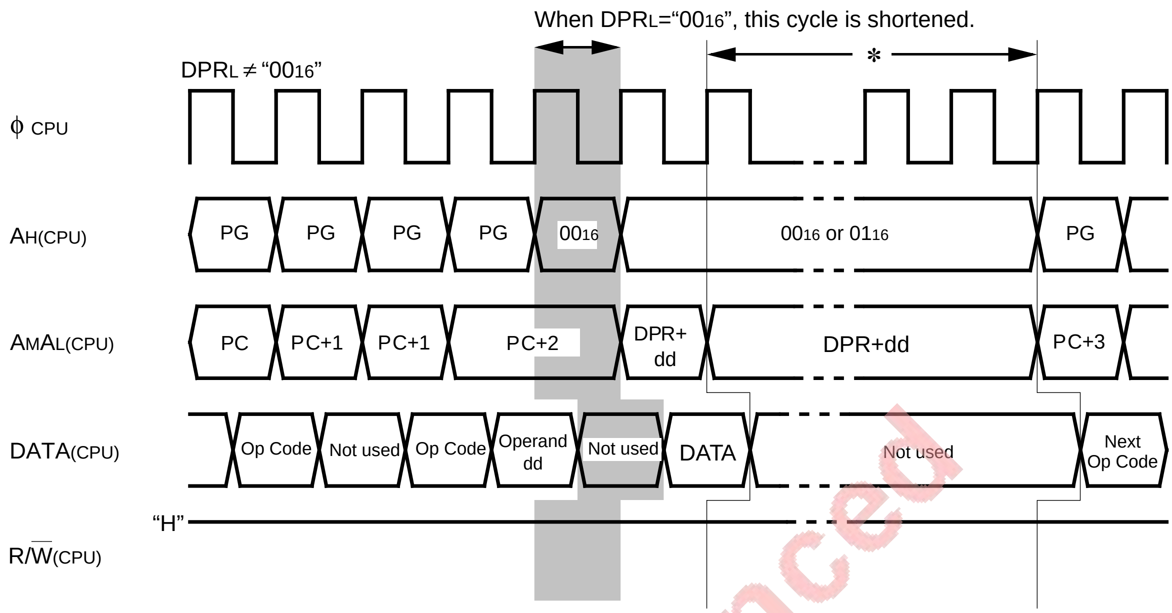
Timing :



Direct

Instructions : DIV, DIVS, MPY, MPYS

Timing :



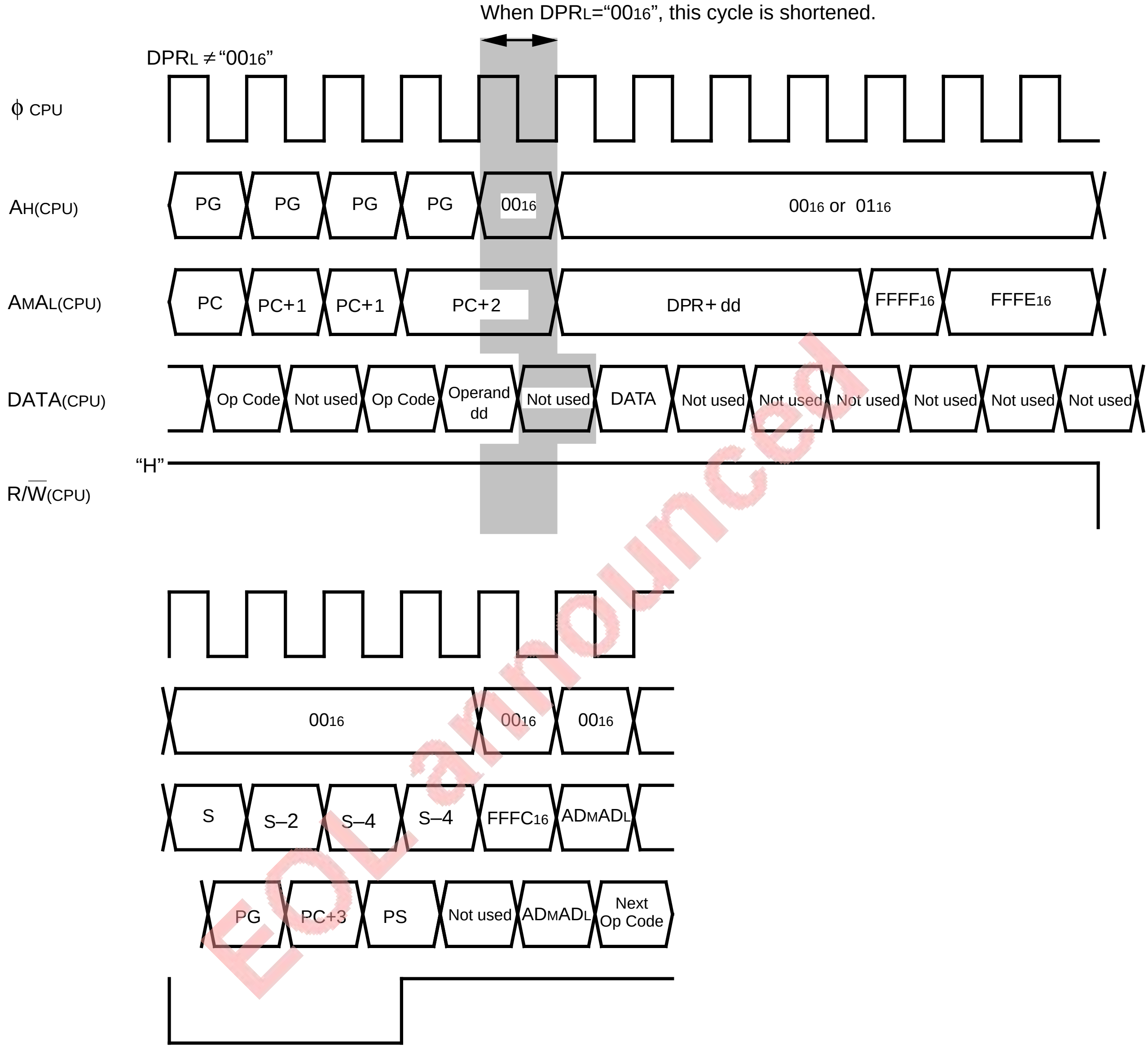
Note: The cycle number during "✻" is shown as the following table:

Instruction	Cycle number (ϕ CPU)	
	m ="0"	m ="1"
DIV	25	17
DIVS	27	19
MPY	8	4
MPYS	8	4

- The contents of AMAL(CPU) during "✻" with DIVS are undefined.
- When each sign of a multiplier and a multiplicand with MPYS is different, the contens of AMAL(CPU) at the last 3 of ϕ CPU during "✻" are undefined.

Instructions : DIV, DIVS (case of 0 division)

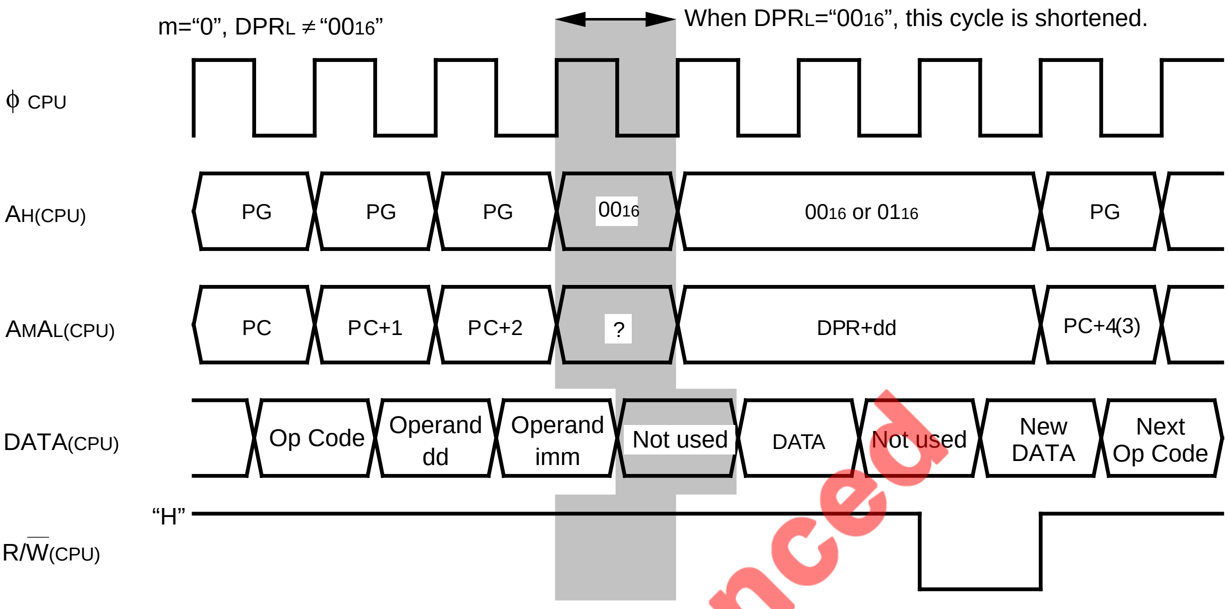
Timing :



Direct Bit

Instructions : CLB, SEB

Timing :



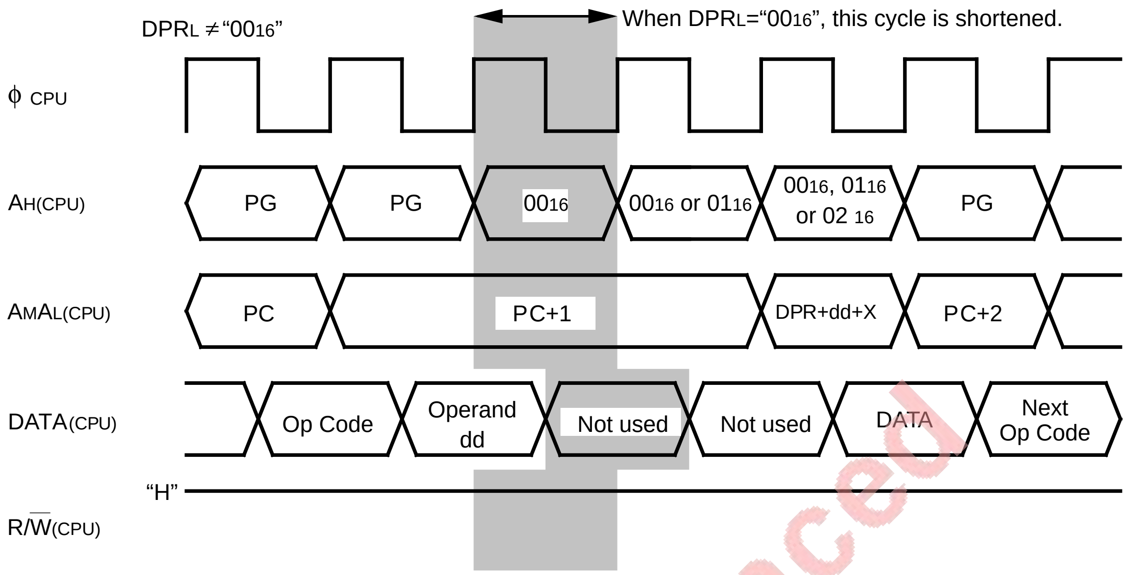
Notes 1: The operand which is fetched at the 3rd cycle is as follows:
When m="0", 2 bytes
When m="1", 1 byte

2: "(   )" shows the case of m="1".

Direct Indexed X

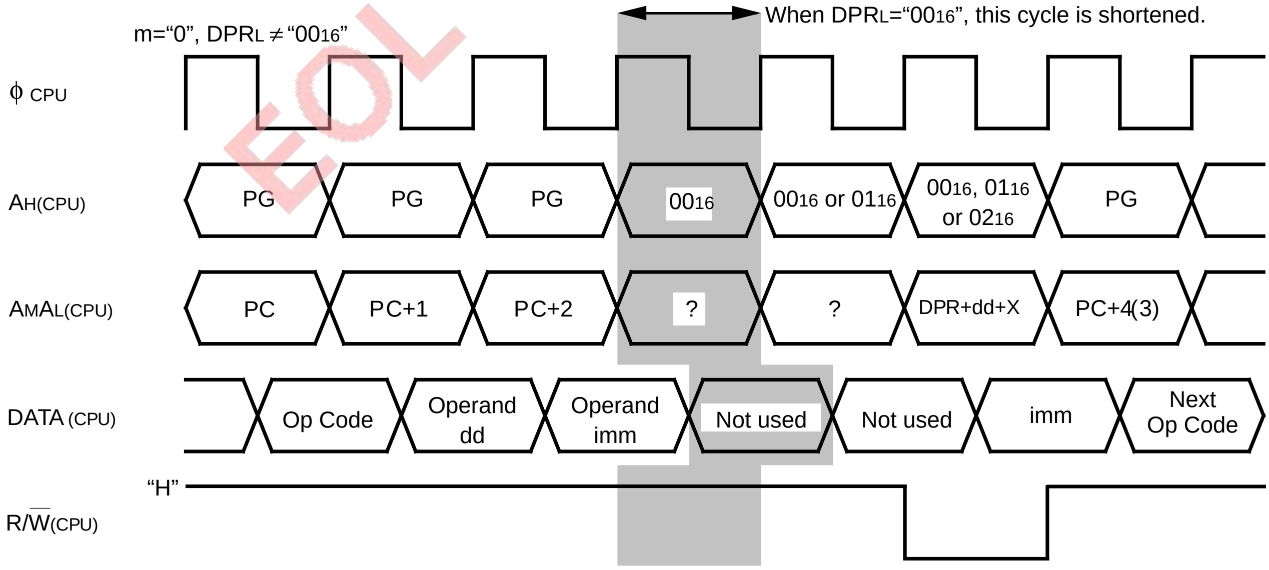
Instructions : ADC, AND, CMP, EOR, LDA, LDY, ORA, SBC

Timing :



Instructions : LDM

Timing :



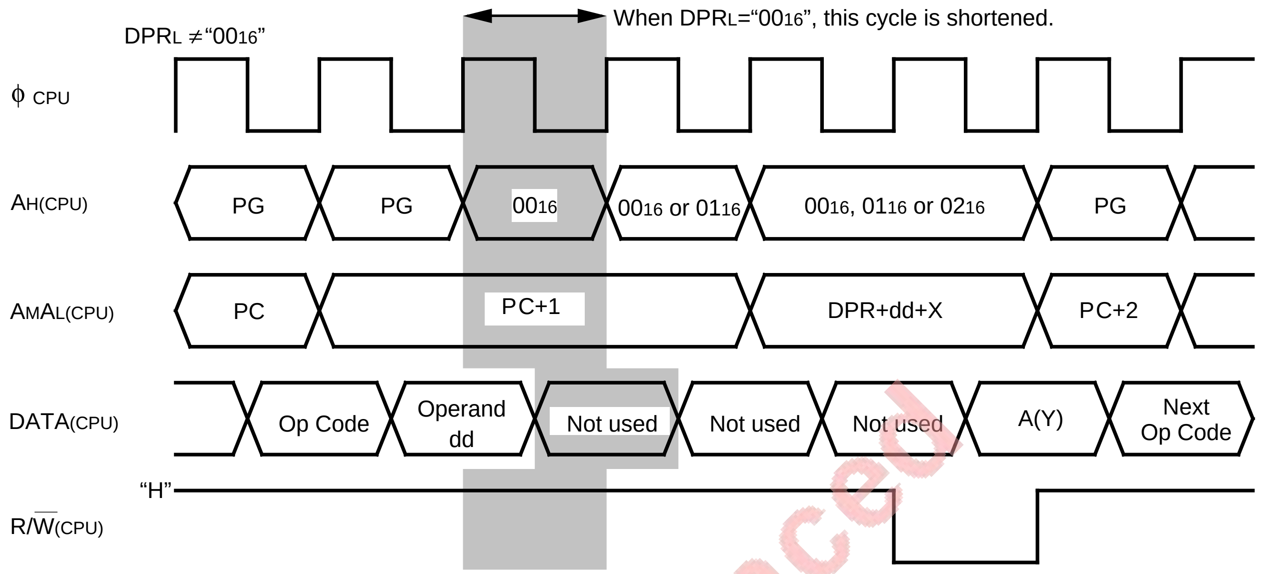
Notes 1: Each of the operand which is fetched at the 3rd cycle and the data which is at the 6th cycle is as follows:
When m="0", 2 bytes
When m="1", 1 byte

2: "( )" shows the case of m="1".

Direct Indexed X

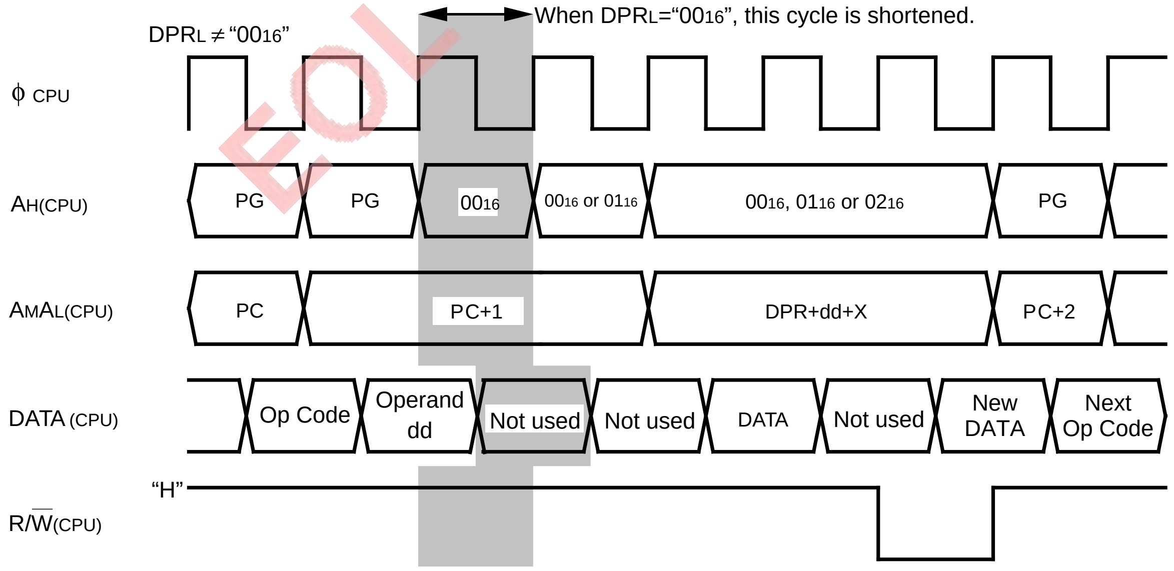
Instructions : STA, STY

Timing :



Instructions : ASL, DEC, INC, LSR, ROL, ROR

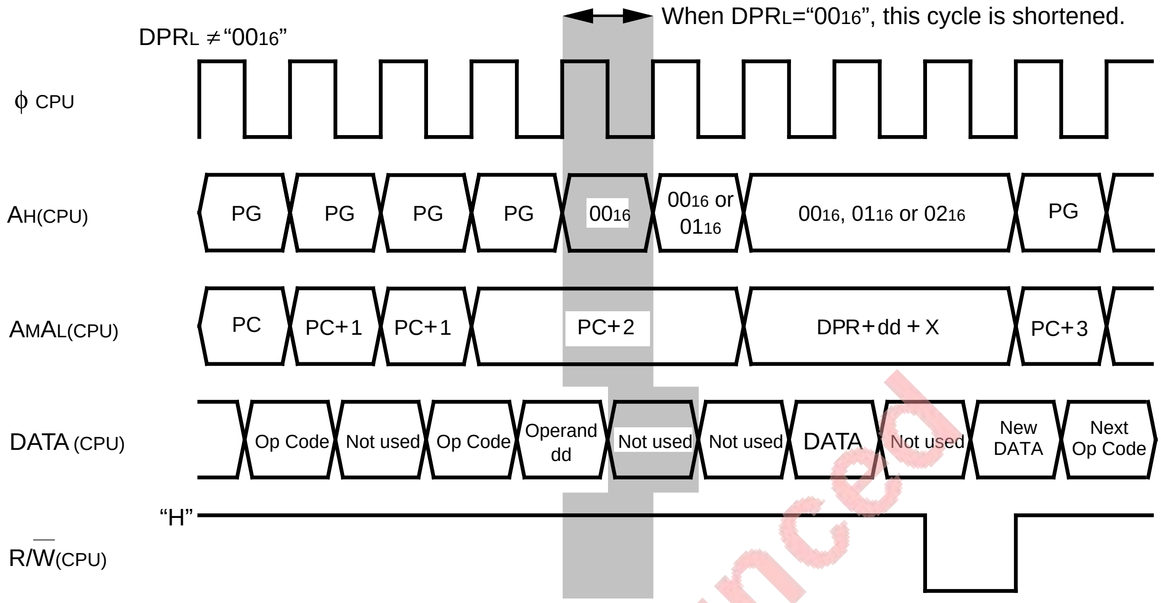
Timing :



Direct Indexed X

Instructions : ASR

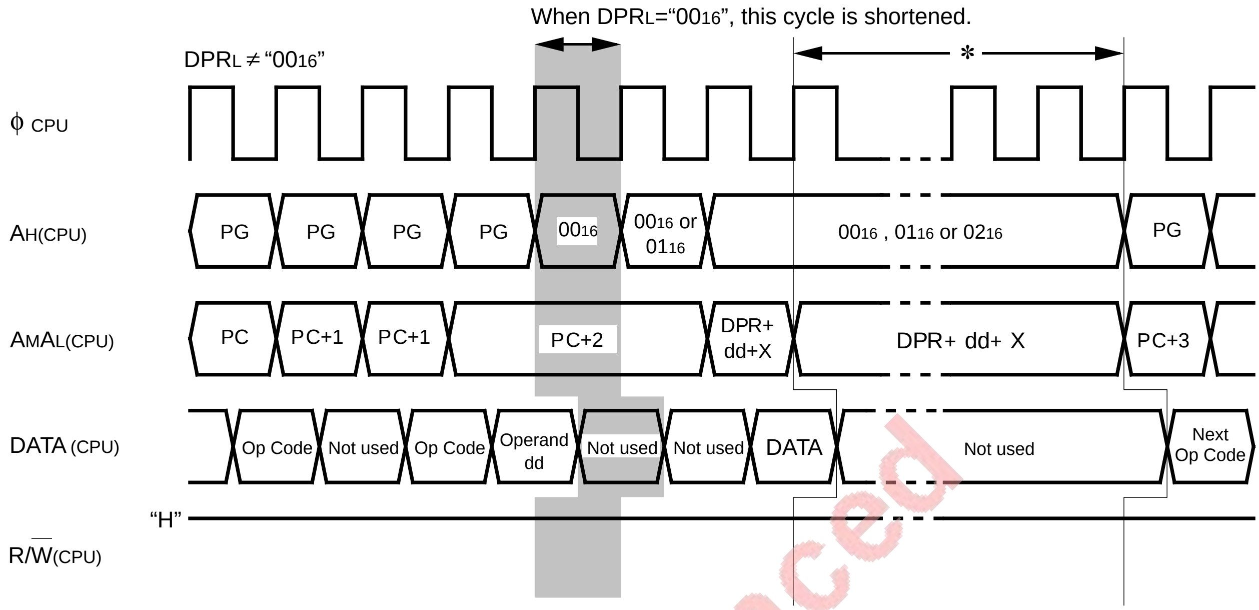
Timing :



Direct Indexed X

Instructions : DIV, DIVS, MPY, MPYS

Timing :



Note: The cycle number during "✻" is shown as the following table:

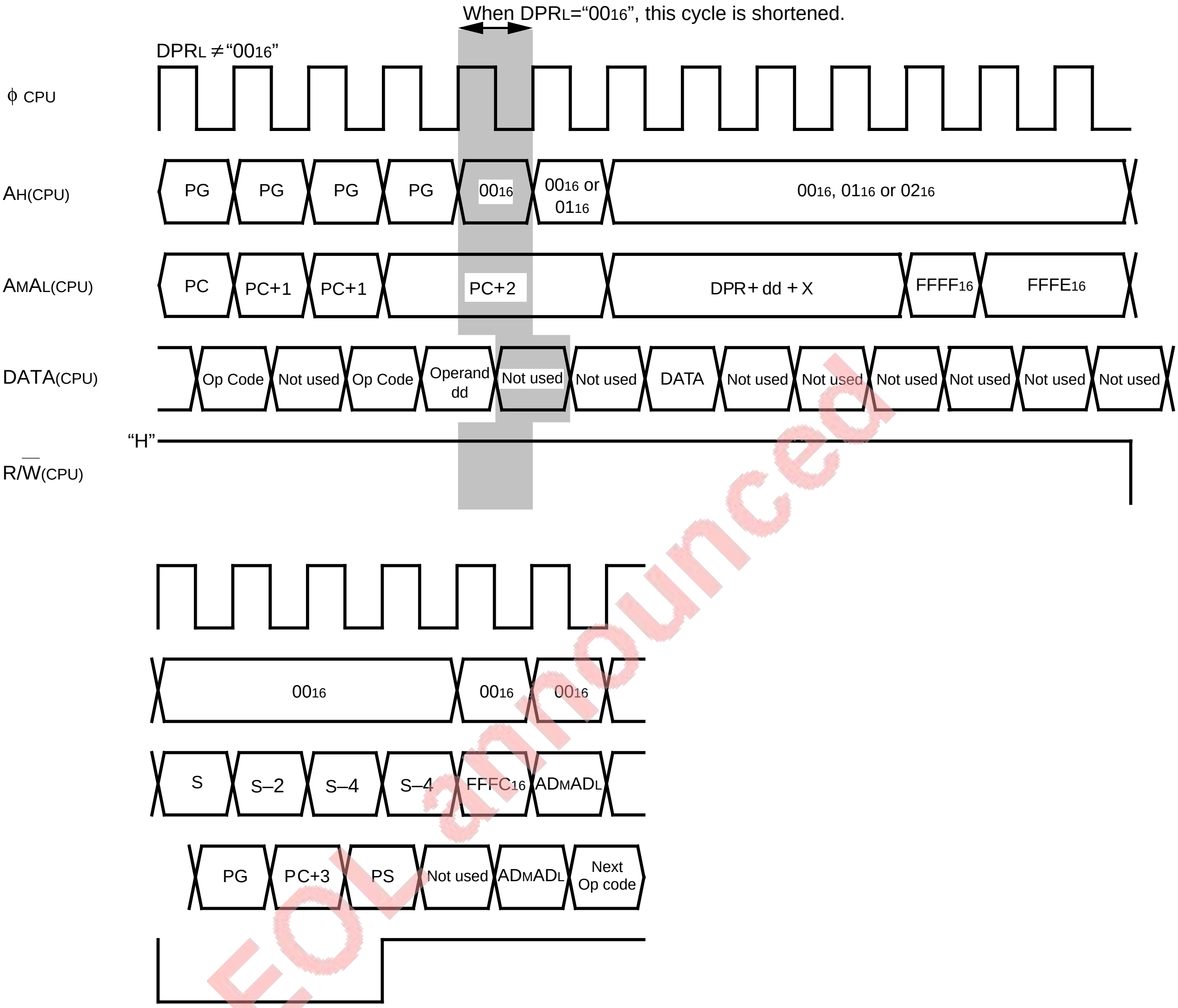
Instruction	Cycle number (φ CPU)	
	m = "0"	m = "1"
DIV	25	17
DIVS	27	19
MPY	8	4
MPYS	8	4

- The contents of AMAL(CPU) during "✻" with DIVS are undefined.
- When each sign of a multiplier and a multiplicand with MPYS is different, the contens of AMAL(CPU) at the last 3 of φ CPU during "✻" are undefined.

Direct Indexed X

Instructions : DIV, DIVS (case of 0 division)

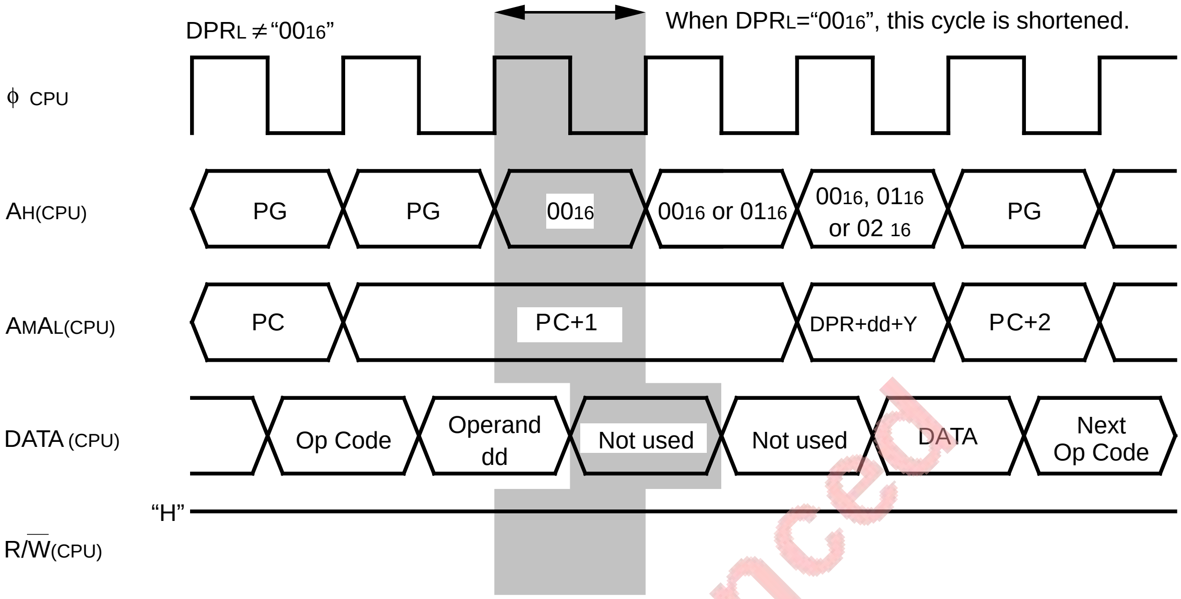
Timing :



Direct Indexed Y

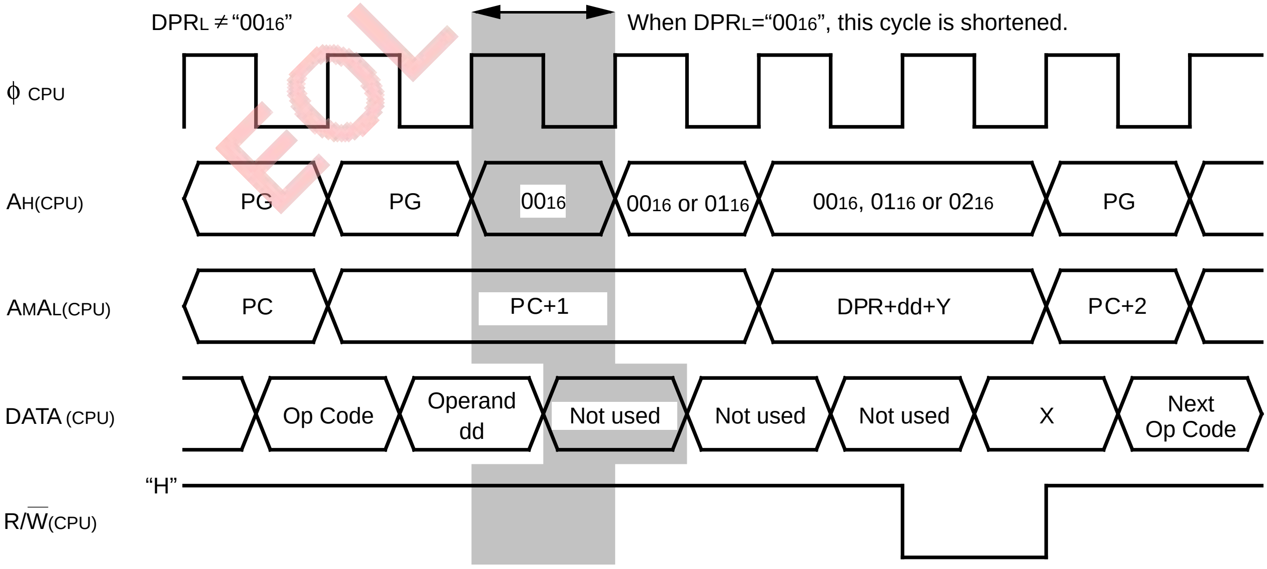
Instructions : LDX

Timing :



Instructions : STX

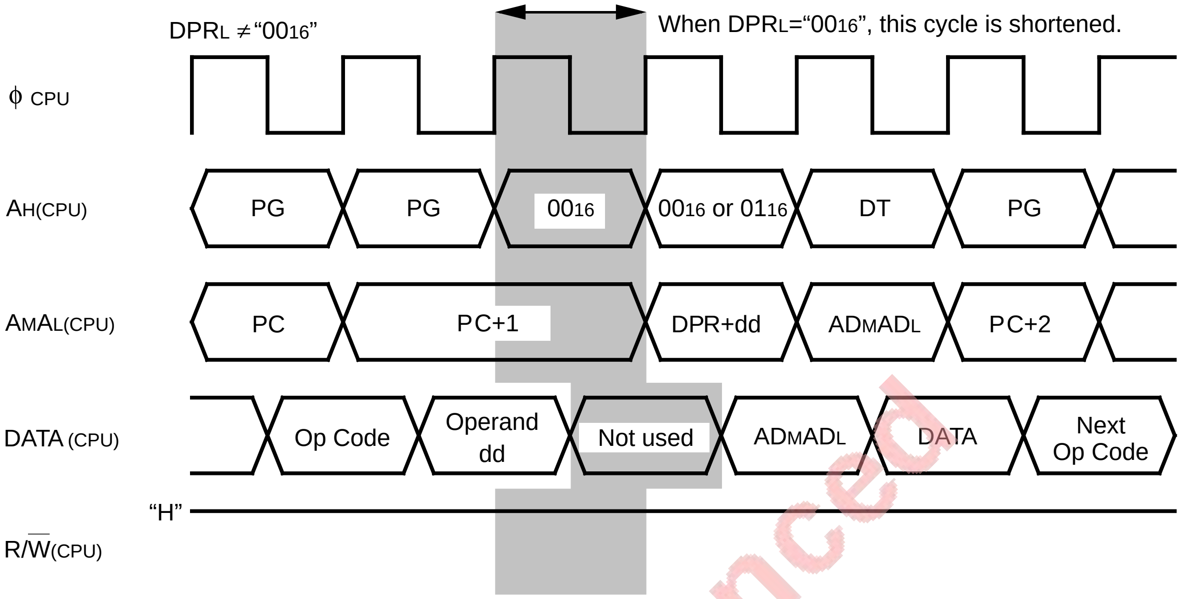
Timing :



Direct Indirect

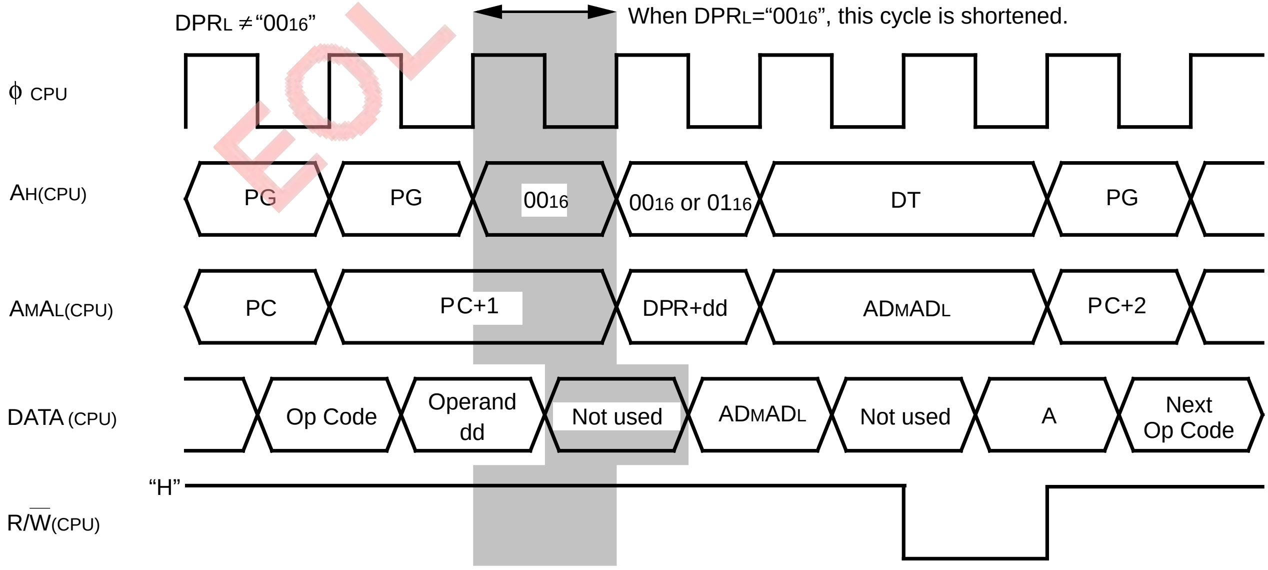
Instructions : ADC, AND, CMP, EOR, LDA, ORA, SBC

Timing :



Instructions : STA

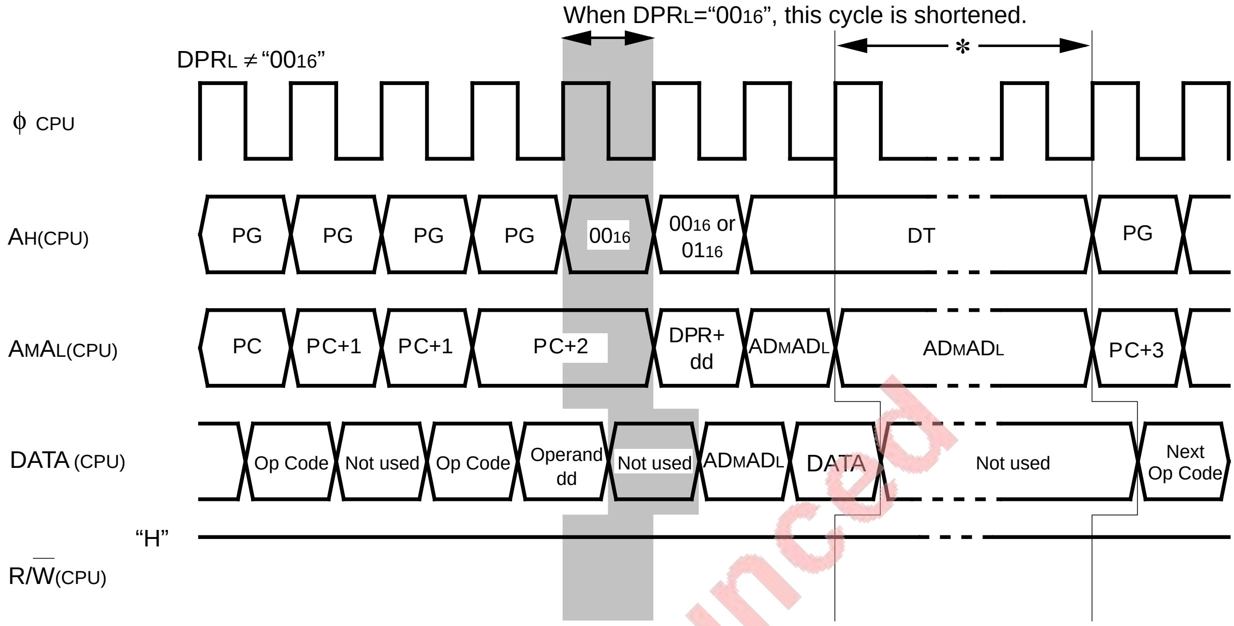
Timing :



Direct Indirect

Instructions : DIV, DIVS, MPY, MPYS

Timing :



Note: The cycle number during "∗" is shown as the following table:

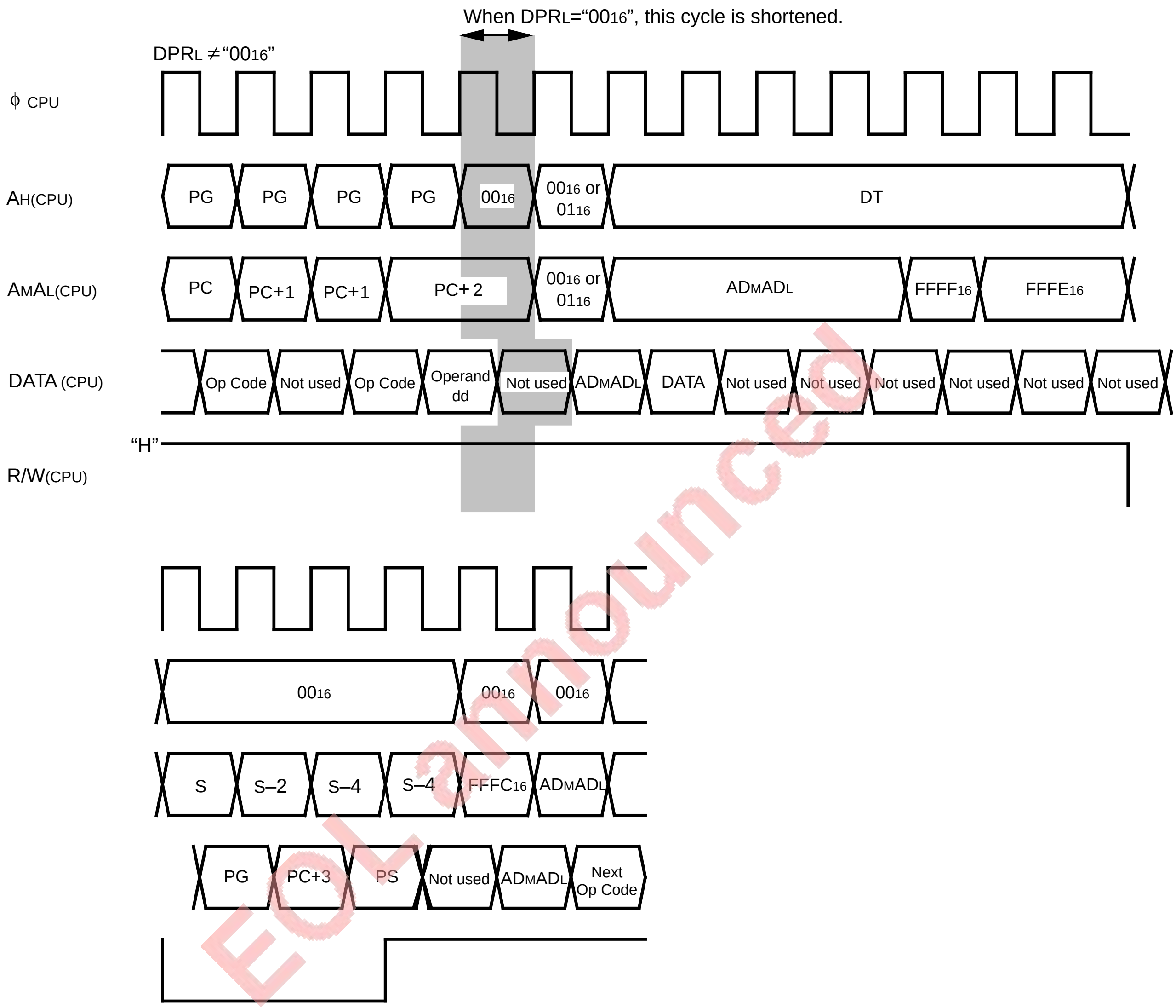
Instruction	Cycle number (φ CPU)	
	m = "0"	m = "1"
DIV	25	17
DIVS	27	19
MPY	8	4
MPYS	8	4

- The contents of AMAL(CPU) during "∗" with DIVS are undefined.
- When each sign of a multiplier and a multiplicand with MPYS is different, the contens of AMAL(CPU) at the last 3 of φ CPU during "∗" are undefined.

Direct Indirect

Instructions : DIV, DIVS (case of 0 division)

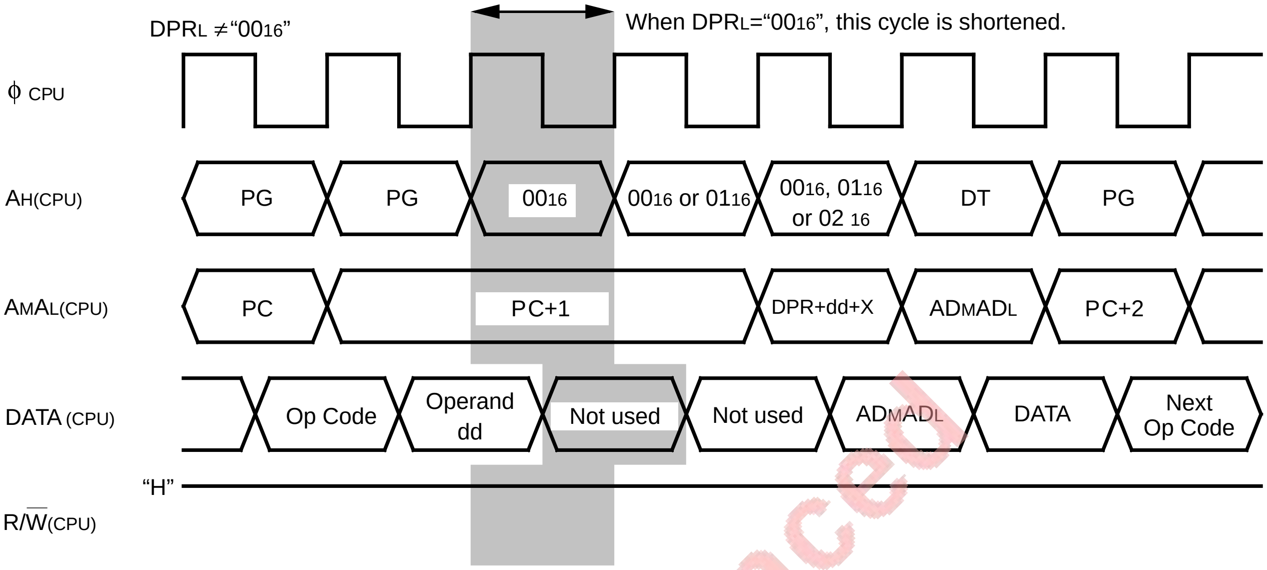
Timing :



Direct Indexed X Indirect

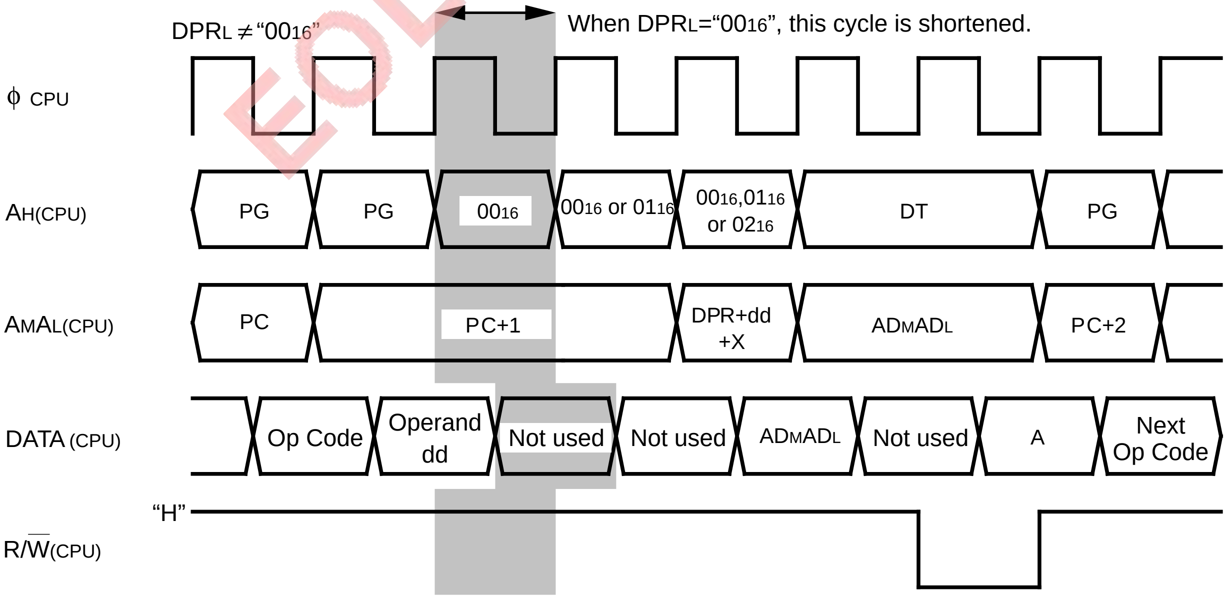
Instructions : ADC, AND, CMP, EOR, LDA, ORA, SBC

Timing :



Instructions : STA

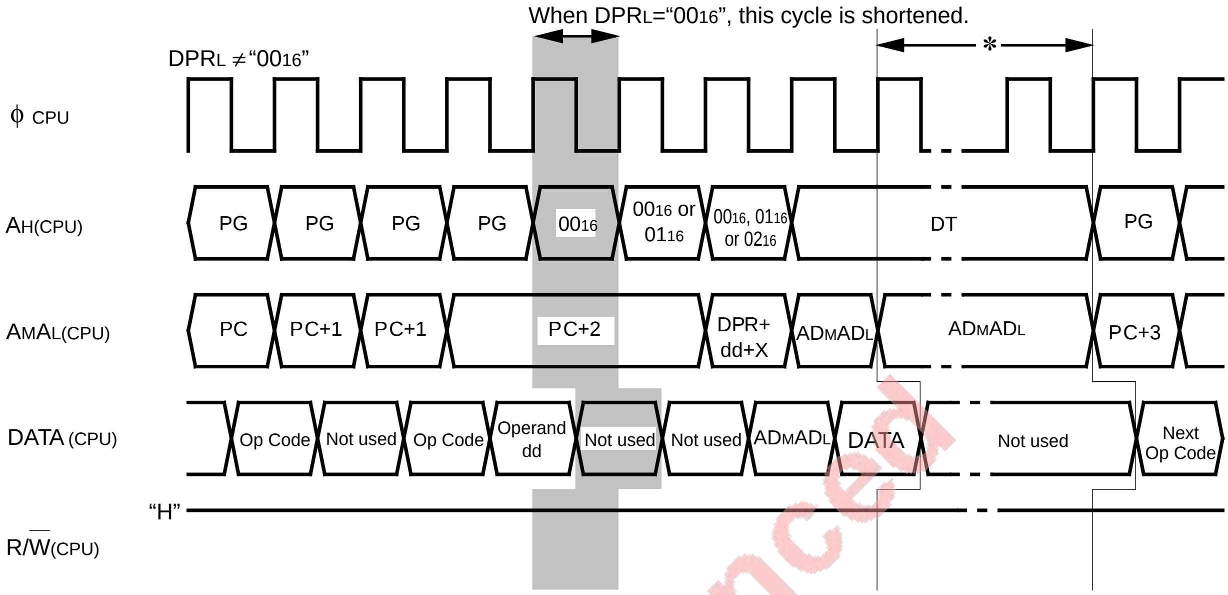
Timing :



Direct Indexed X Indirect

Instructions : DIV, DIVS, MPY, MPYS

Timing :



Note: The cycle number during "✻" is shown as the following table:

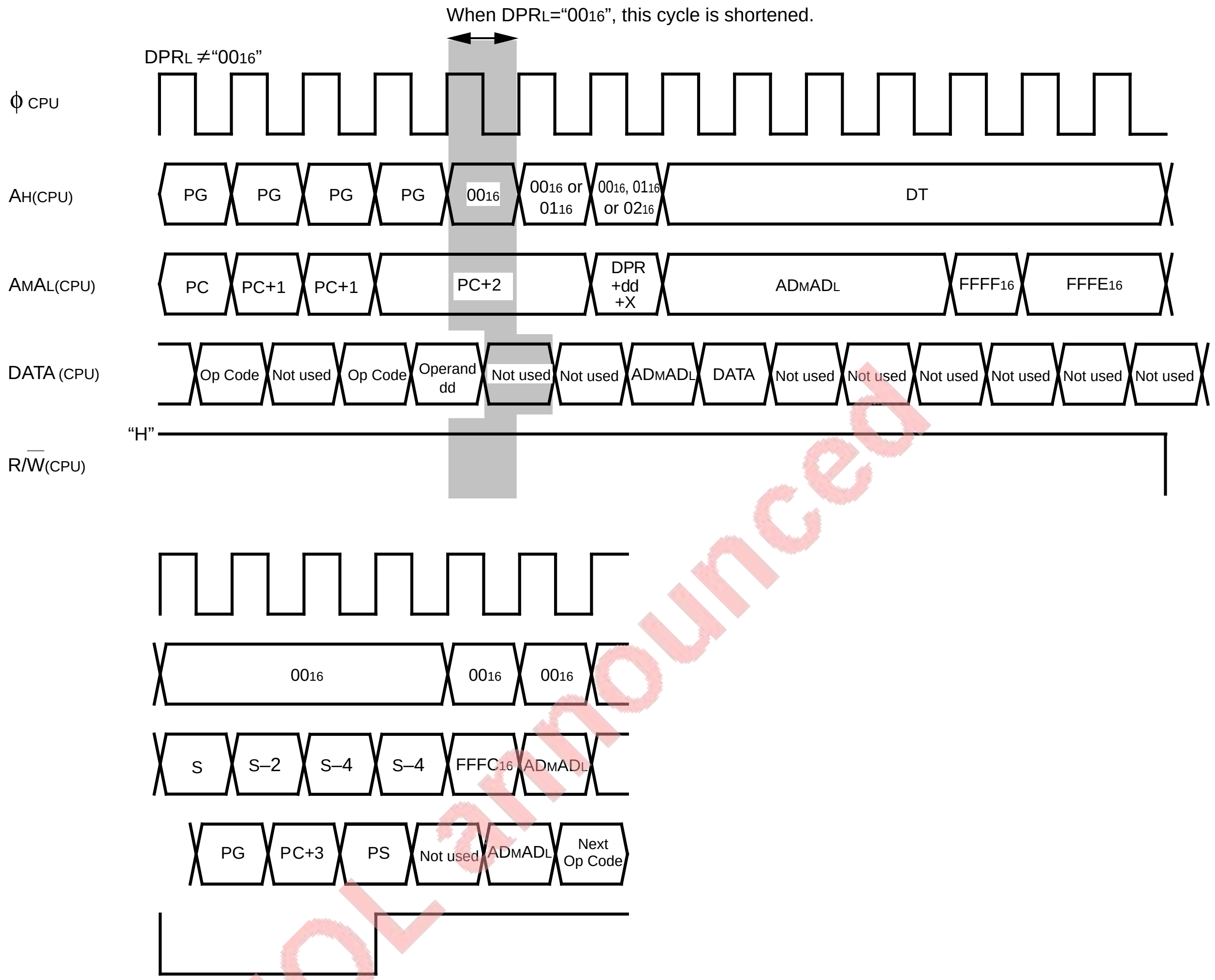
Instruction	Cycle number (φ CPU)	
	m = "0"	m = "1"
DIV	25	17
DIVS	27	19
MPY	8	4
MPYS	8	4

- The contents of AMAL(CPU) during "✻" with DIVS are undefined.
- When each sign of a multiplier and a multiplicand with MPYS is different, the contents of AMAL(CPU) at the last 3 of φ CPU during "✻" are undefined.

Direct Indexed X Indirect

Instructions : DIV, DIVS (case of 0 division)

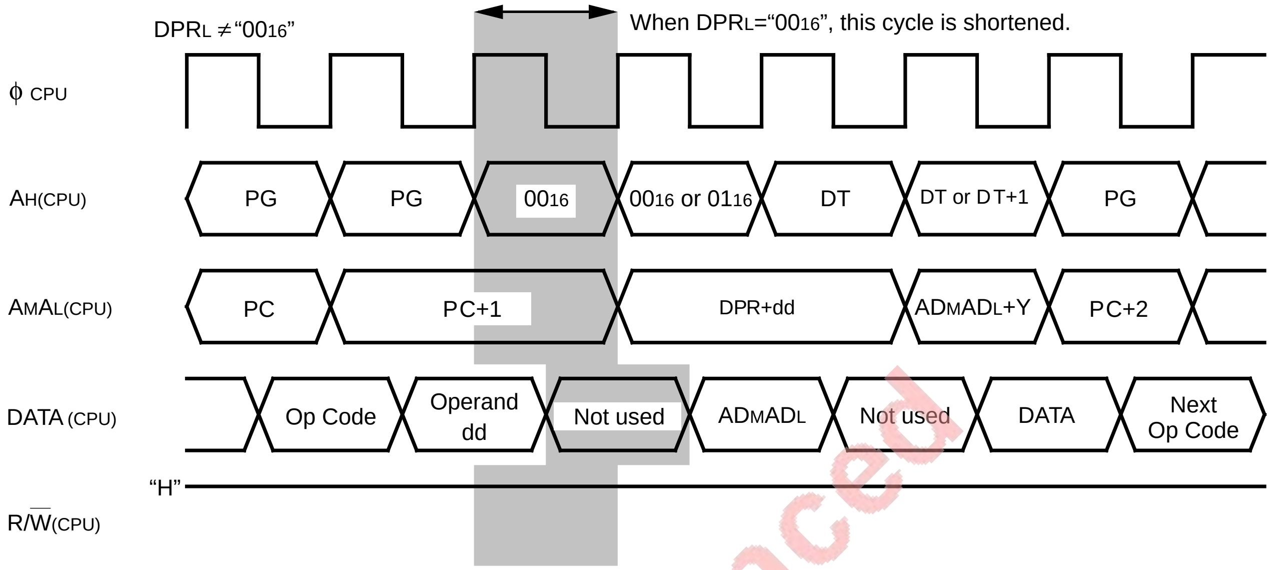
Timing :



Direct Indirect Indexed Y

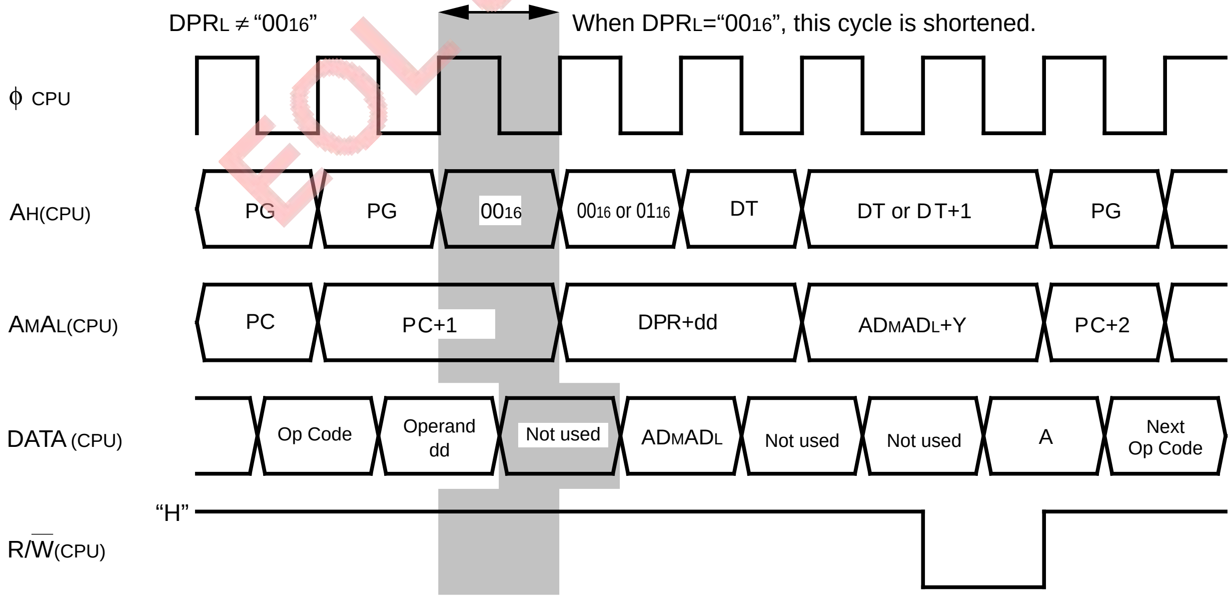
Instructions : ADC, AND, CMP, EOR, LDA, ORA, SBC

Timing :



Instructions : STA

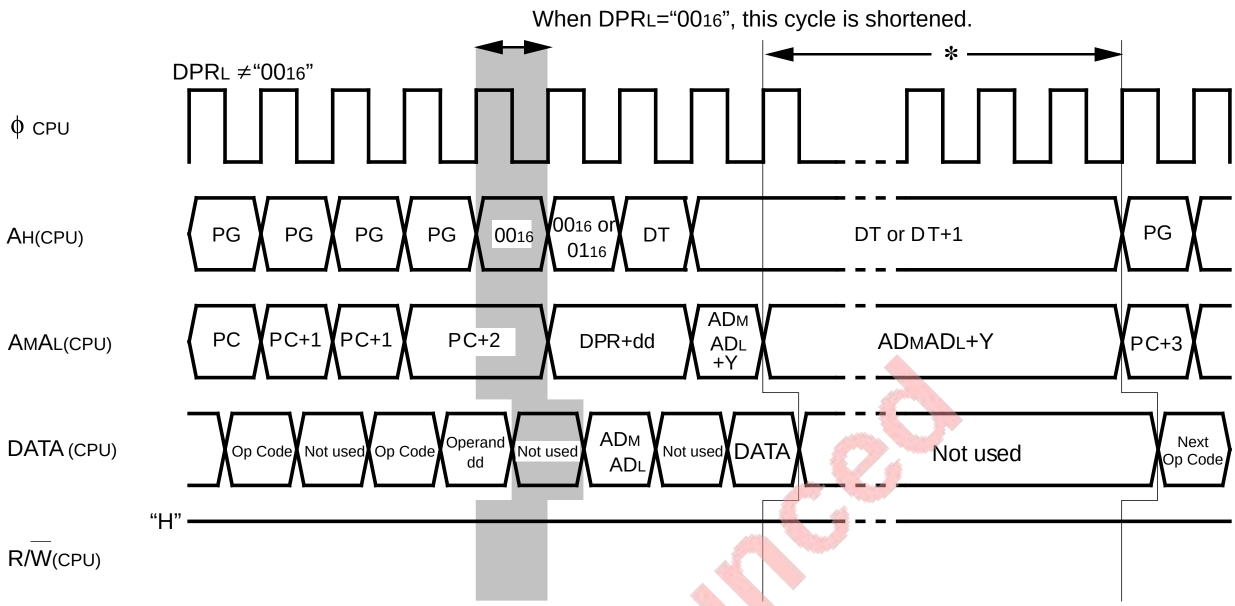
Timing :



Direct Indirect Indexed Y

Instructions : DIV, DIVS, MPY, MPYS

Timing :



Note: The cycle number during "✻" is shown as the following table:

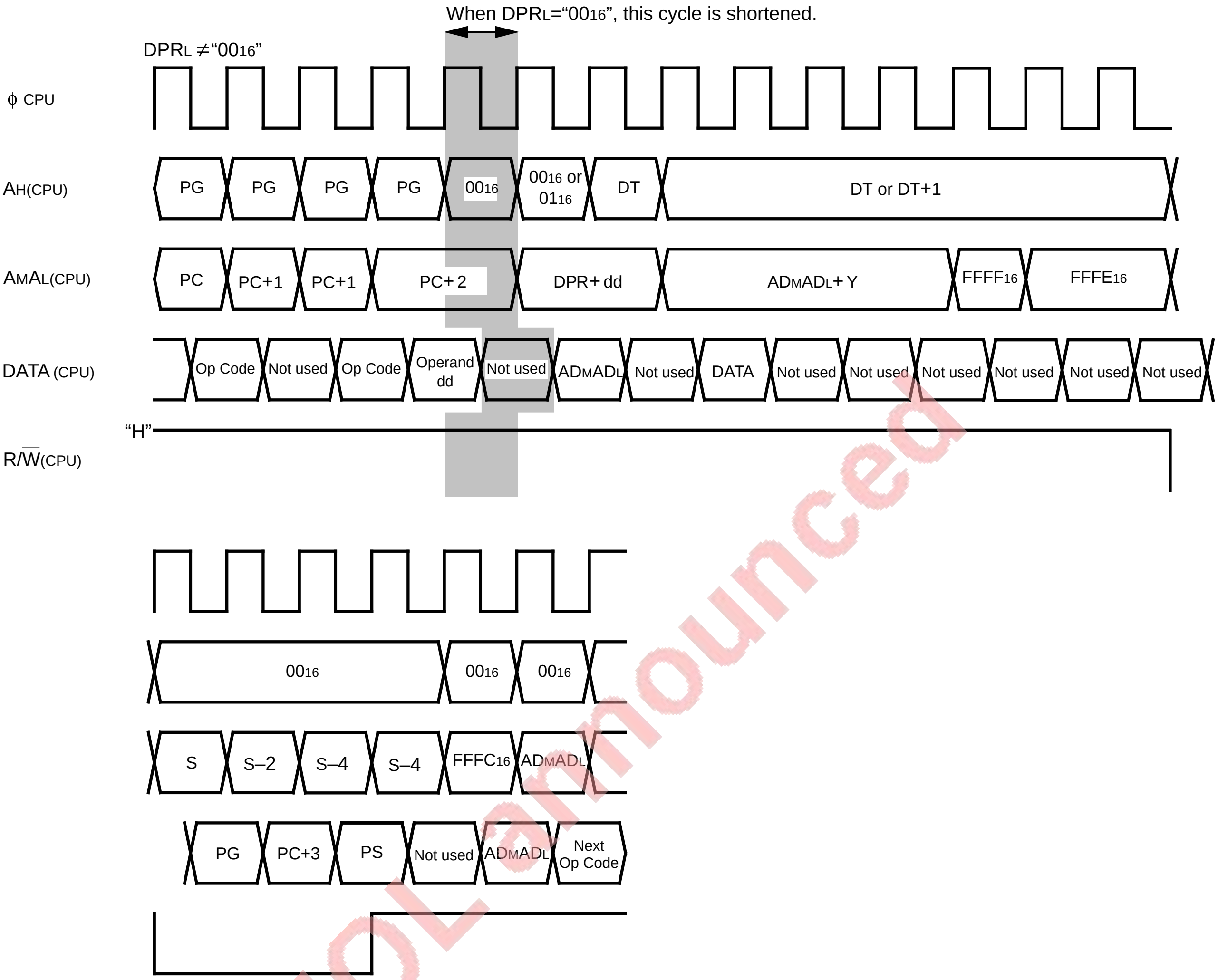
Instruction	Cycle number (φ CPU)	
	m = "0"	m = "1"
DIV	25	17
DIVS	27	19
MPY	8	4
MPYS	8	4

- The contents of AMAL(CPU) during "✻" with DIVS are undefined.
- When each sign of a multiplier and a multiplicand with MPYS is different, the contents of AMAL(CPU) at the last 3 of φ CPU during "✻" are undefined.

Direct Indirect Indexed Y

Instructions : DIV, DIVS (case of 0 division)

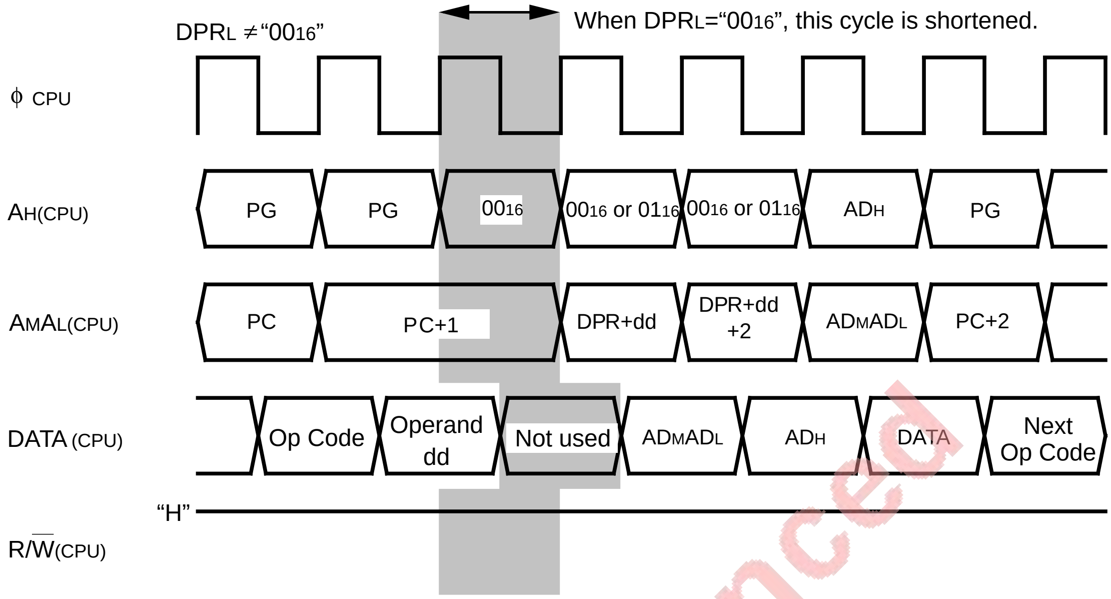
Timing :



Direct Indirect Long

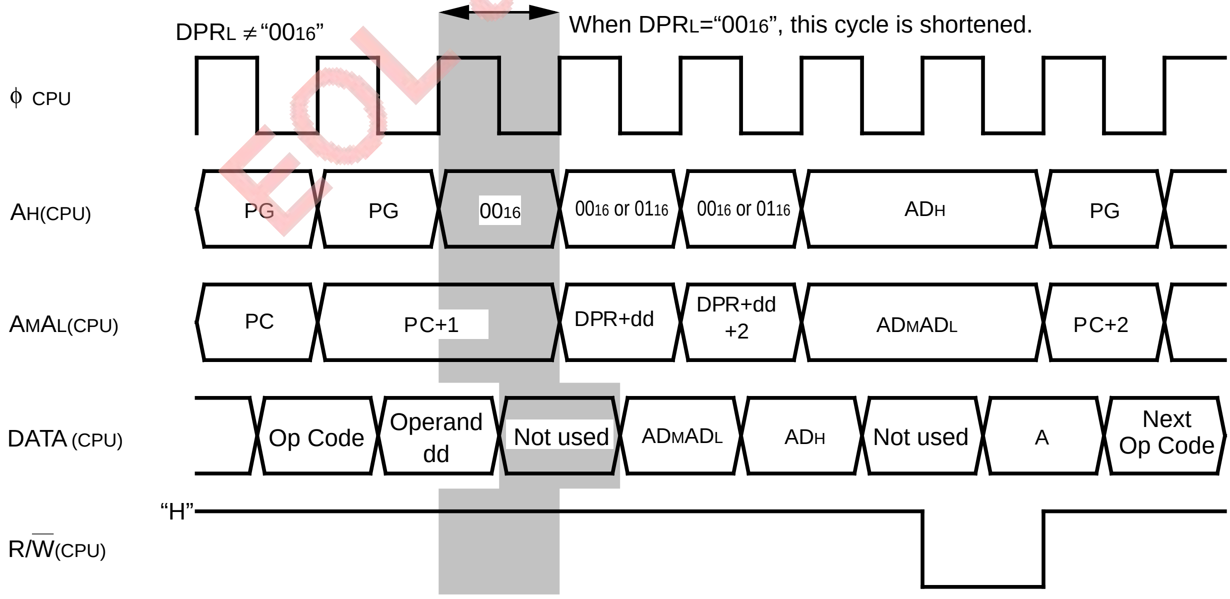
Instructions : ADC, AND, CMP, EOR, LDA, ORA, SBC

Timing :



Instructions : STA

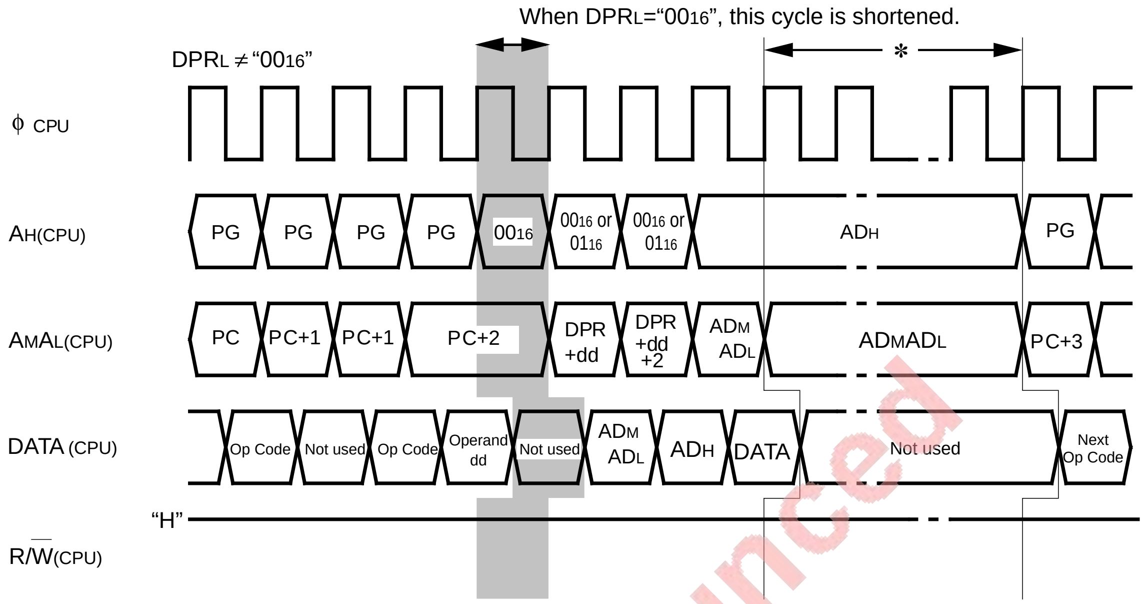
Timing :



Direct Indirect Long

Instructions : DIV, DIVS, MPY, MPYS

Timing :



Note: The cycle number during "✻" is shown as the following table:

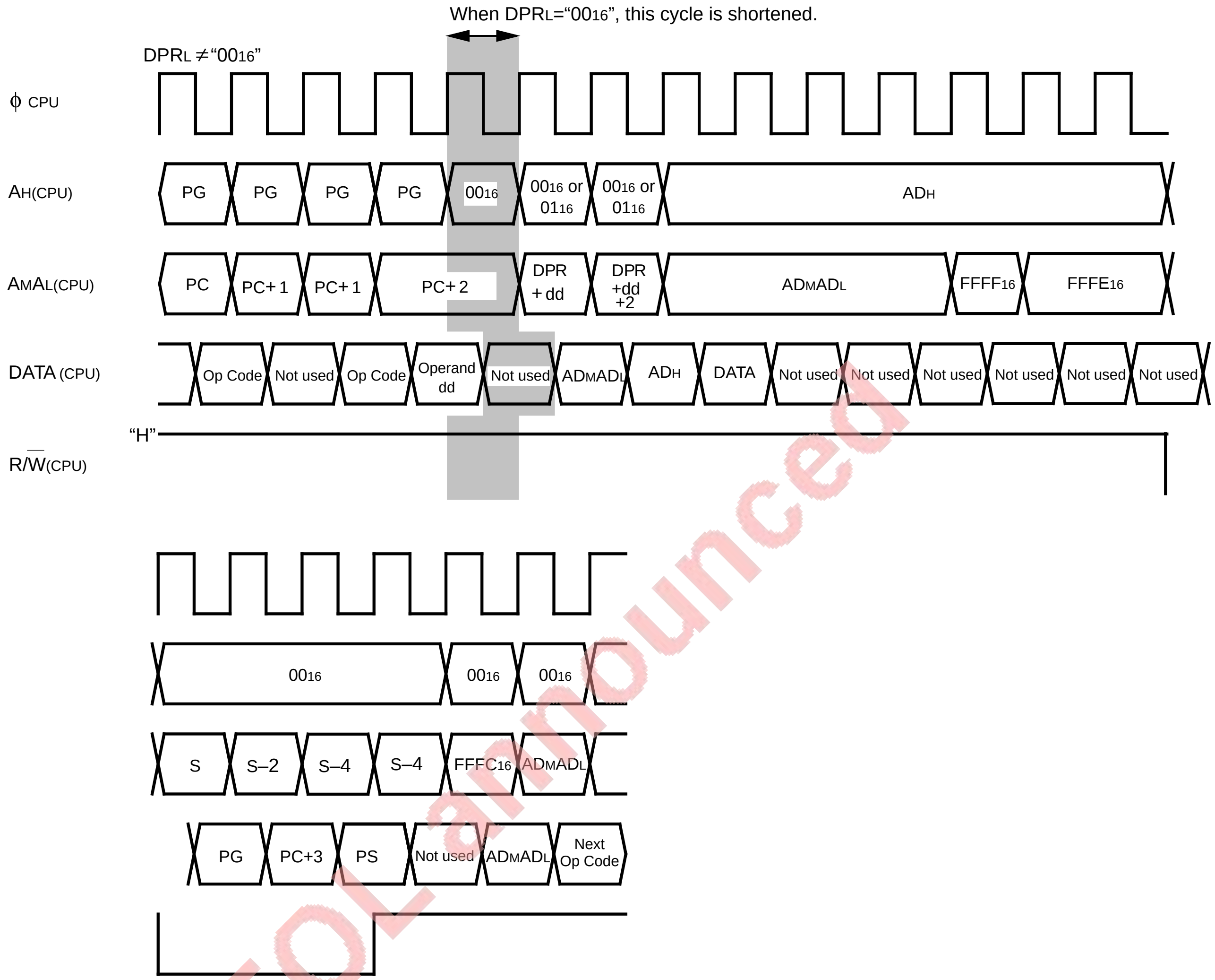
Instruction	Cycle number (φCPU)	
	m = "0"	m = "1"
DIV	25	17
DIVS	27	19
MPY	8	4
M PYS	8	4

- The contents of AMAL(CPU) during "✻" with DIVS are undefined.
- When each sign of a multiplier and a multiplicand with MPYS is different, the contents of AMAL(CPU) at the last 3 of φ CPU during "✻" are undefined.

Direct Indirect Long

Instructions : DIV, DIVS (case of 0 division)

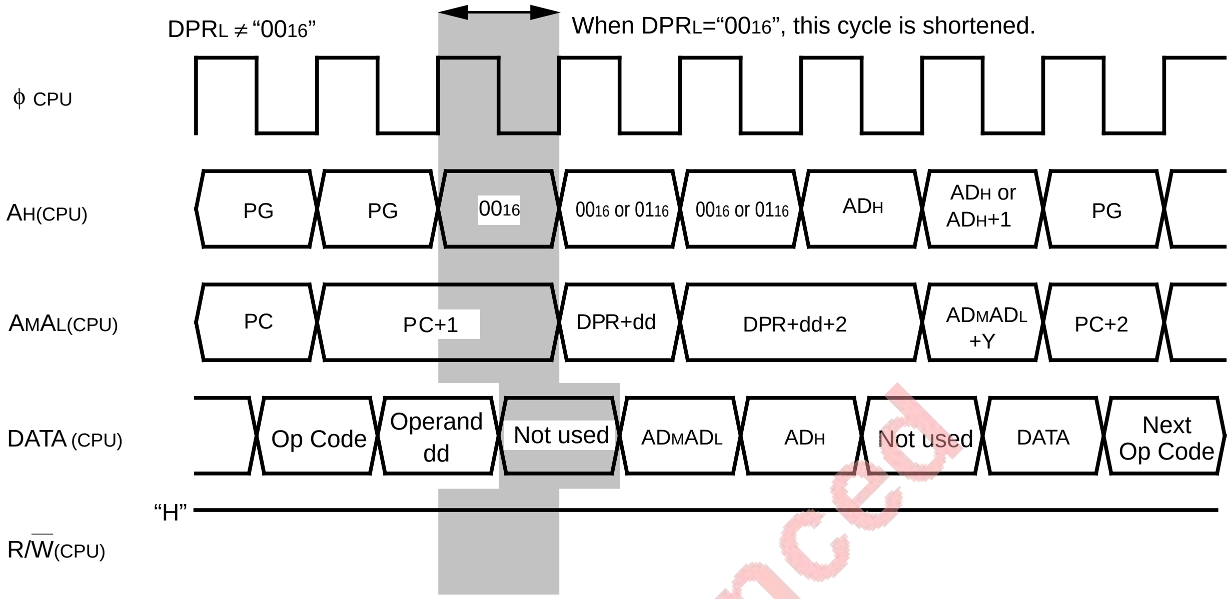
Timing :



Direct Indirect Long Indexed Y

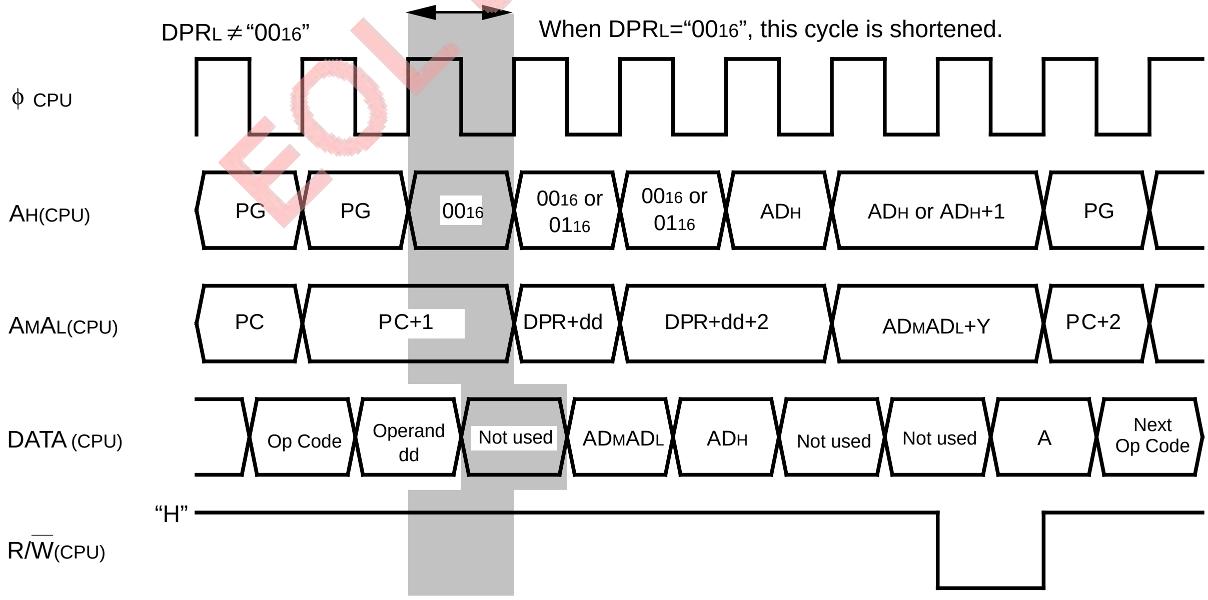
Instructions : ADC, AND, CMP, EOR, LDA, ORA, SBC

Timing :



Instructions : STA

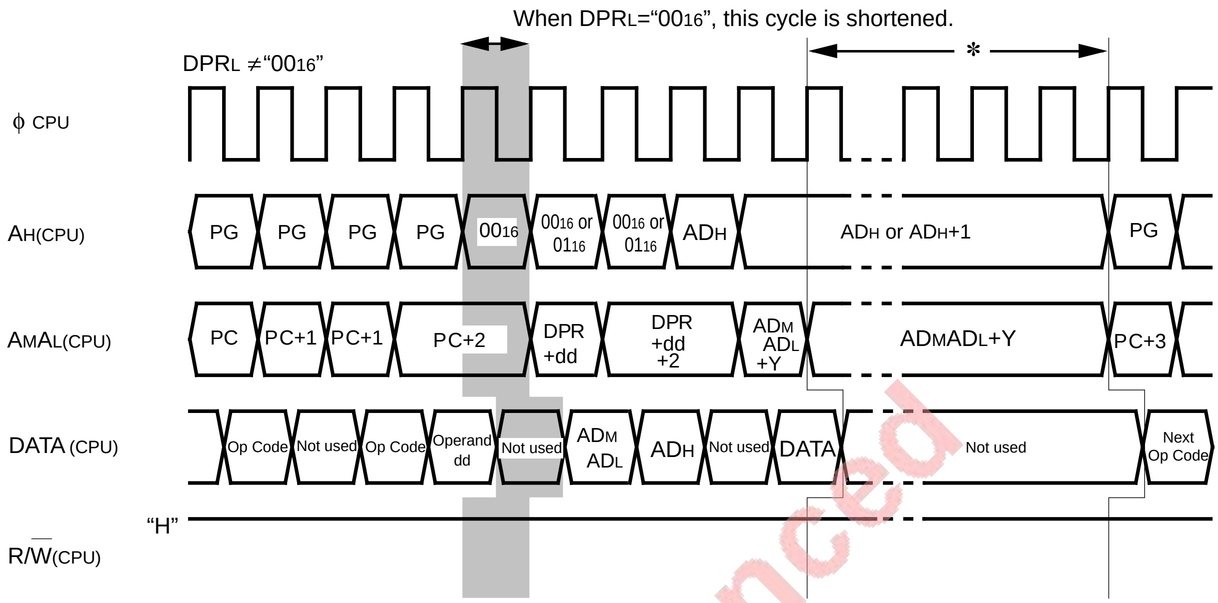
Timing :



Direct Indirect Long Indexed Y

Instructions : DIV, DIVS, MPY, MPYS

Timing :



Note: The cycle number during "∗" is shown as the following table:

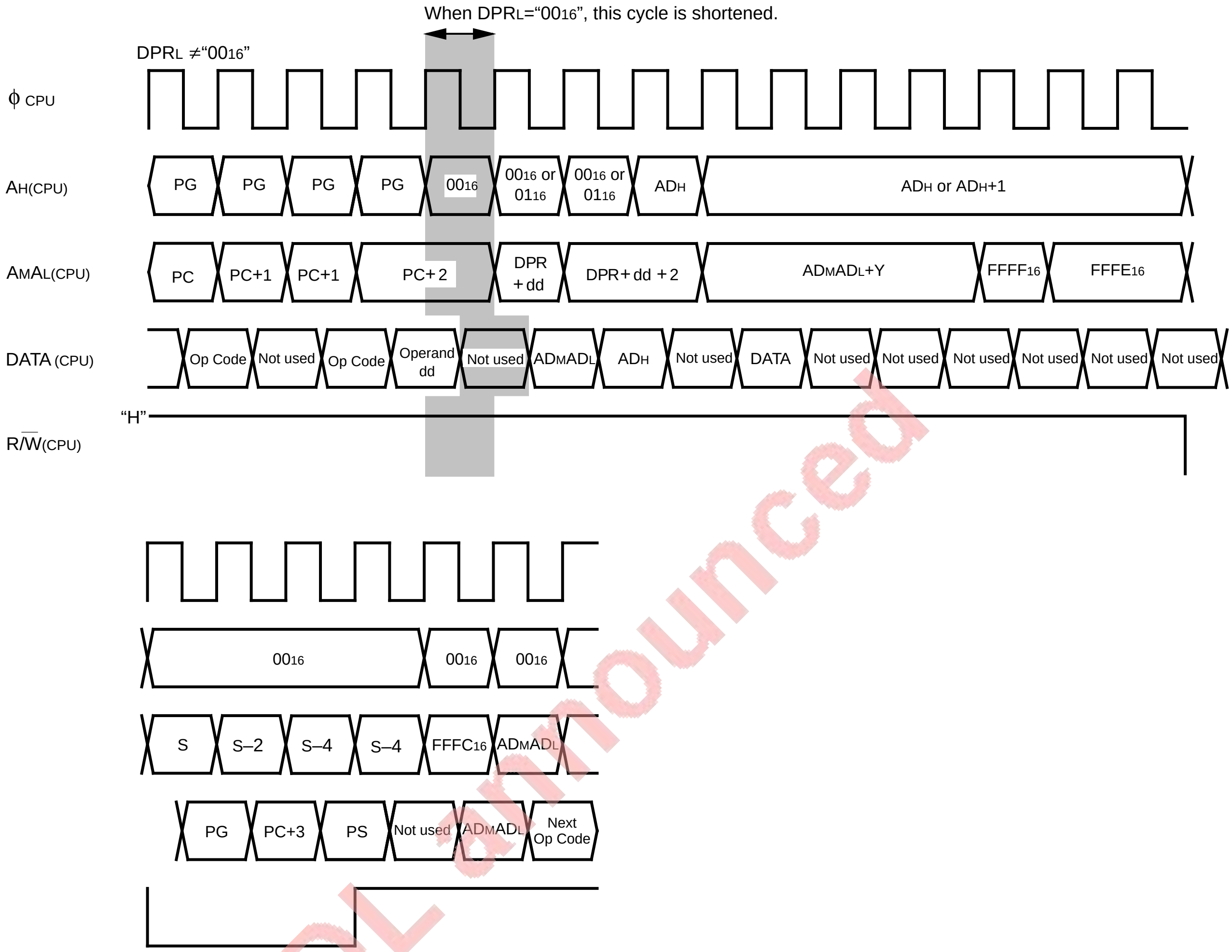
Instruction	Cycle number (φ CPU)	
	m = "0"	m = "1"
DIV	25	17
DIVS	27	19
MPY	8	4
MPYS	8	4

- The contents of AMAL(CPU) during "∗" with DIVS are undefined.
- When each sign of a multiplier and a multiplicand with MPYS is different, the contents of AMAL(CPU) at the last 3 of φ CPU during "∗" are undefined.

Direct Indirect Long Indexed Y

Instructions : DIV, DIVS (case of 0 division)

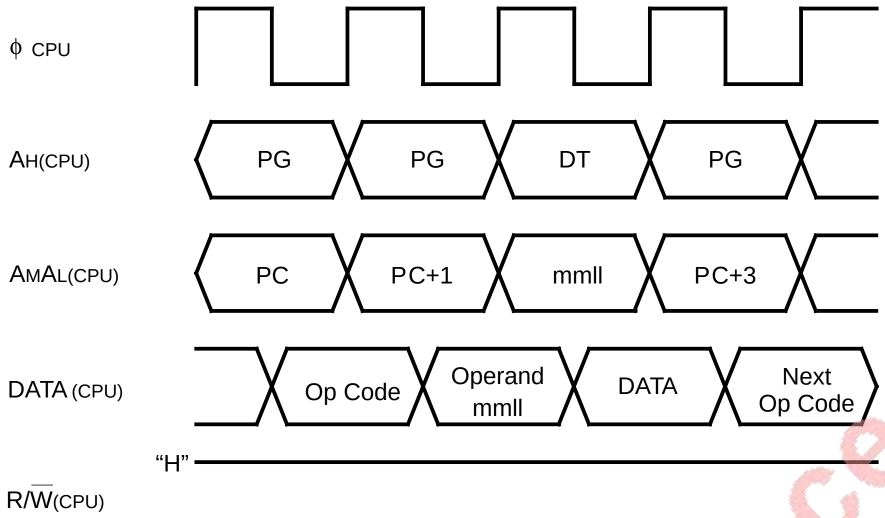
Timing :



Absolute

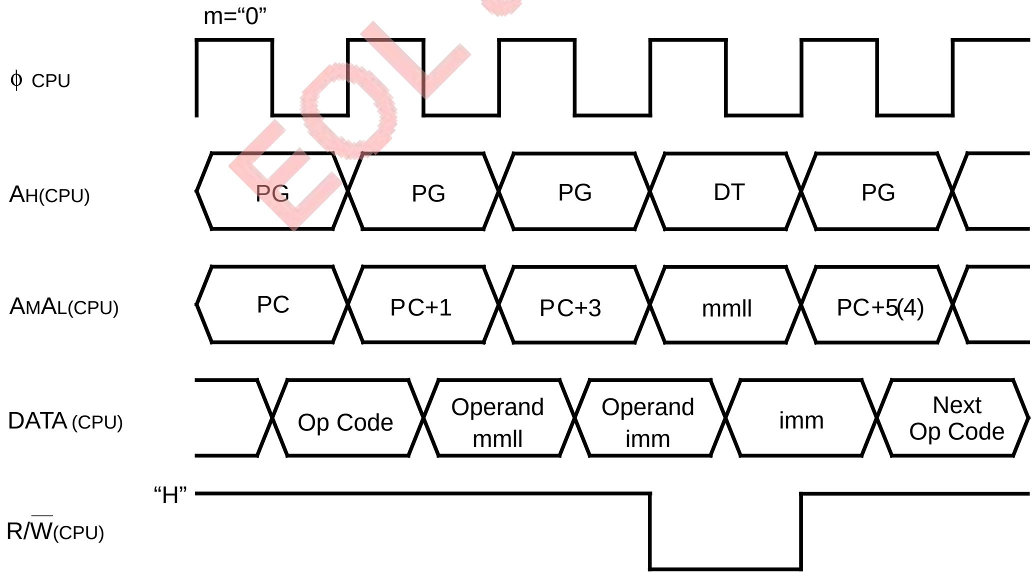
Instructions : ADC, AND, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA, SBC

Timing :



Instructions : LDM

Timing :



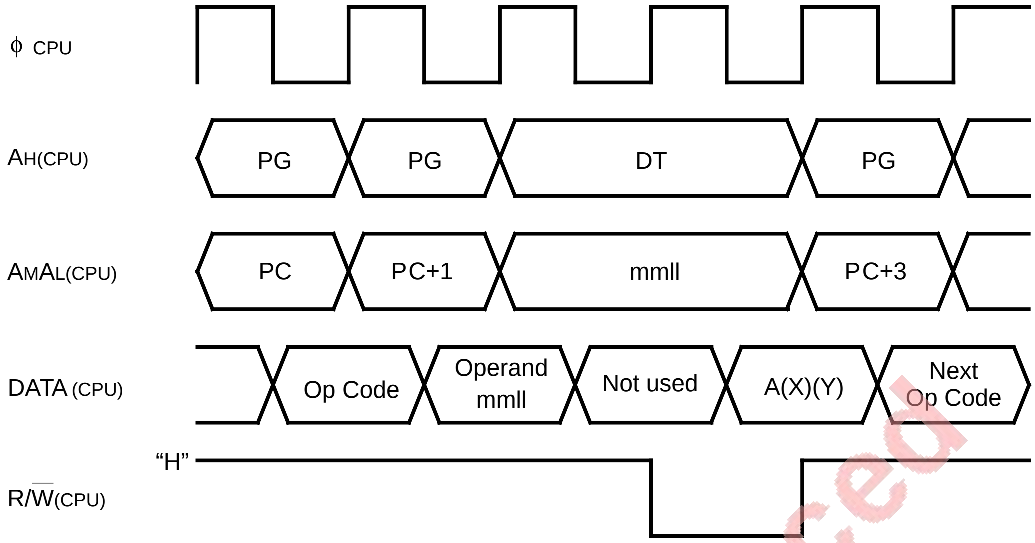
Notes 1: Each of the operand which is fetched at the 3rd cycle and the data which is at the 4th cycle is as follows:
When m="0", 2 bytes
When m="1", 1 byte

2: "( )" shows the case of m="1".

Absolute

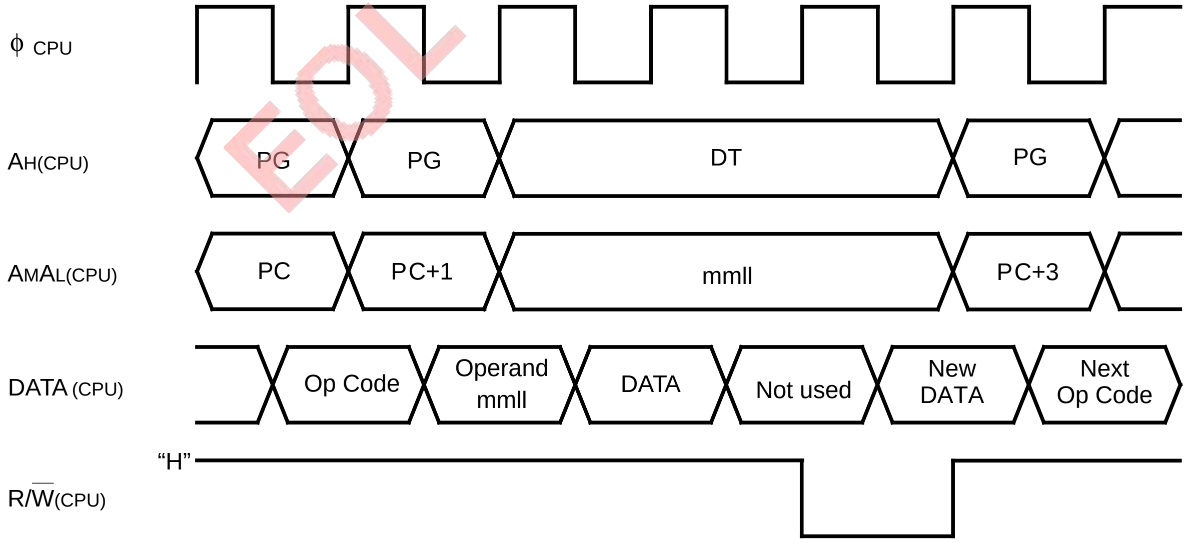
Instructions : STA, STX, STY

Timing :



Instructions : ASL, DEC, INC, LSR, ROL, ROR

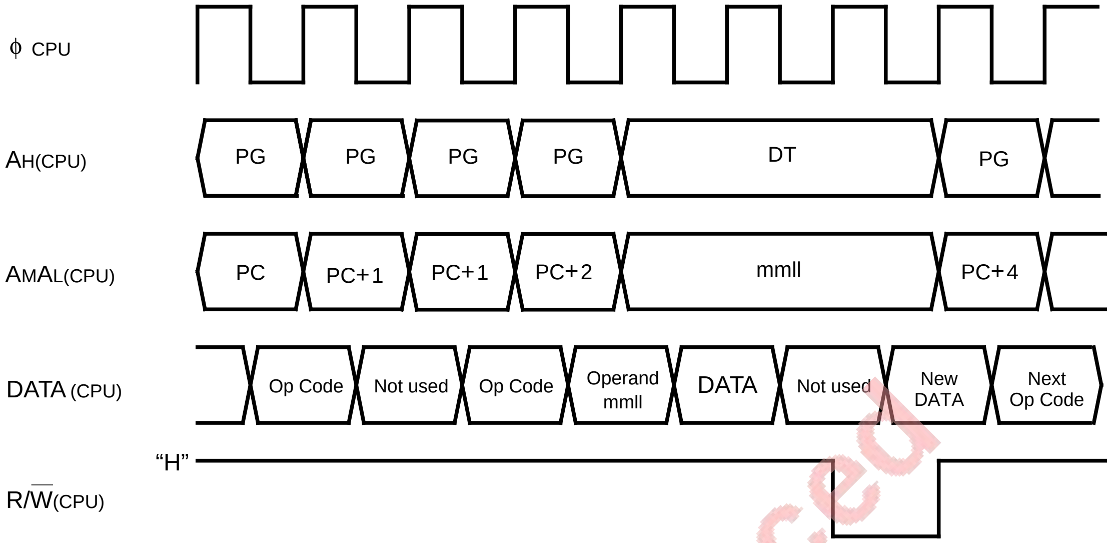
Timing :



Absolute

Instructions : ASR

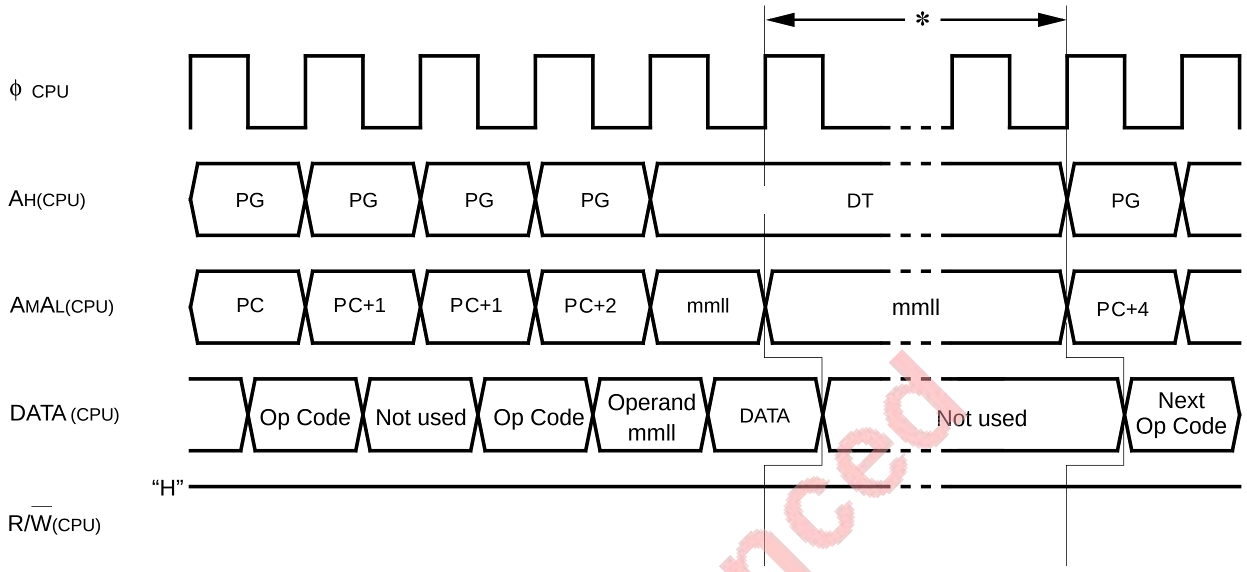
Timing :



Absolute

Instructions : DIV, DIVS, MPY, MPYS

Timing :



Note: The cycle number during "✻" is shown as the following table:

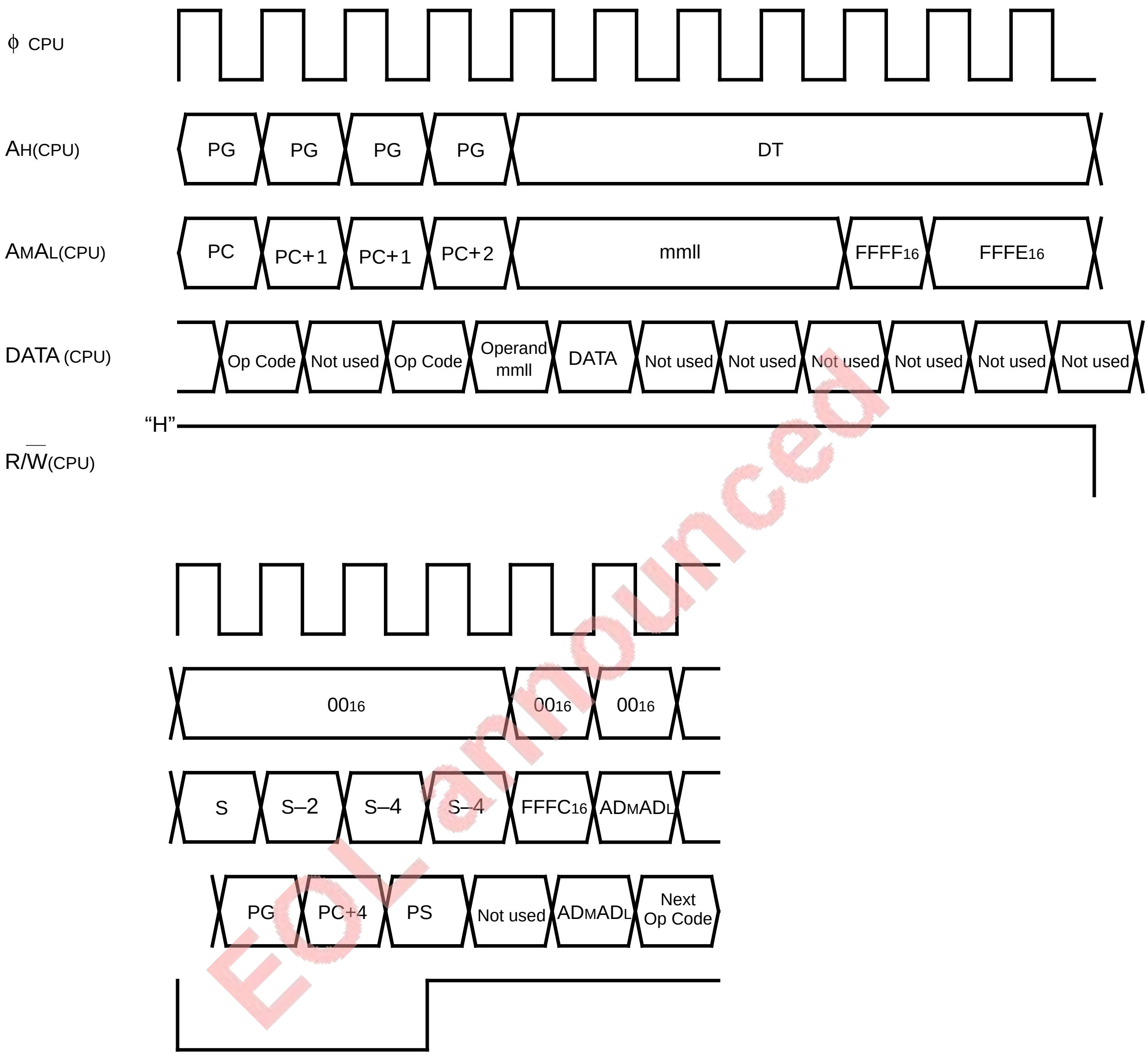
Instruction	Cycle number (φ CPU)	
	m = "0"	m = "1"
DIV	25	17
DIVS	27	19
MPY	8	4
MPYS	8	4

- The contents of AMAL(CPU) during "✻" with DIVS are undefined.
- When each sign of a multiplier and a multiplicand with MPYS is different, the contents of AMAL(CPU) at the last 3 of φ CPU during "✻" are undefined.

Absolute

Instructions : DIV, DIVS (case of 0 division)

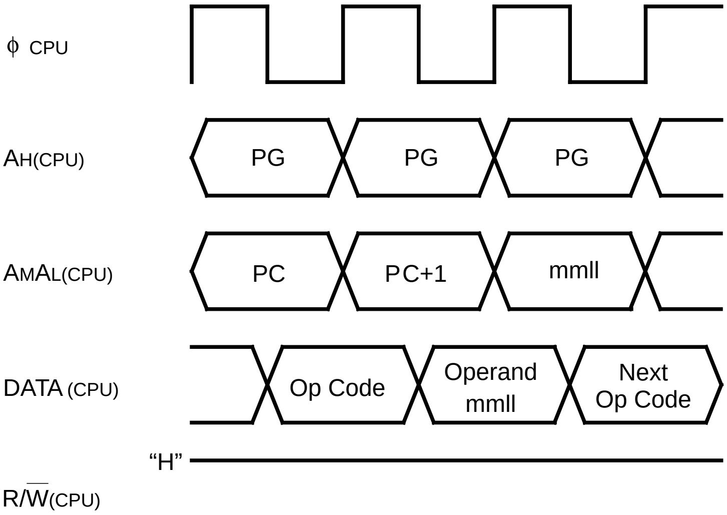
Timing :



Absolute

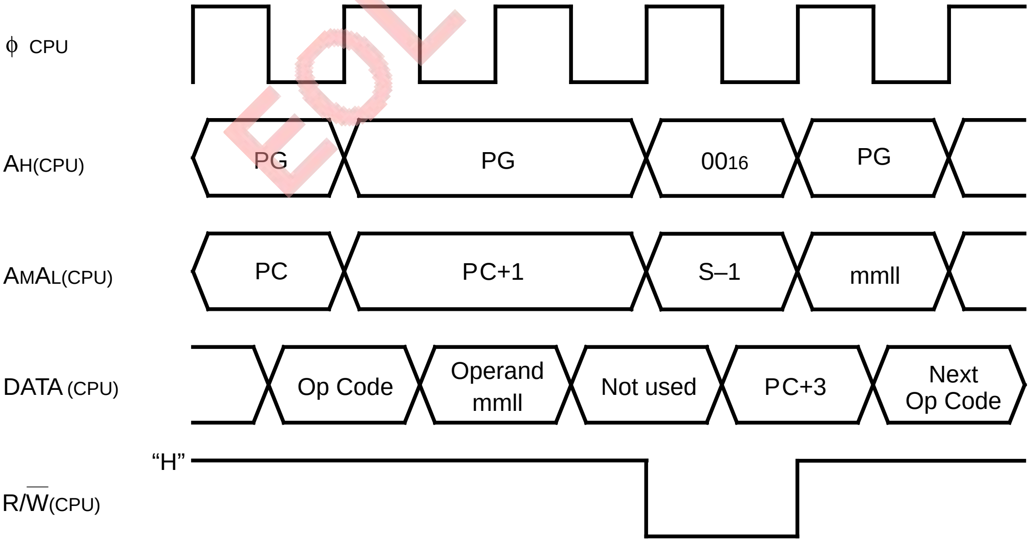
Instructions : JMP

Timing :



Instructions : JSR

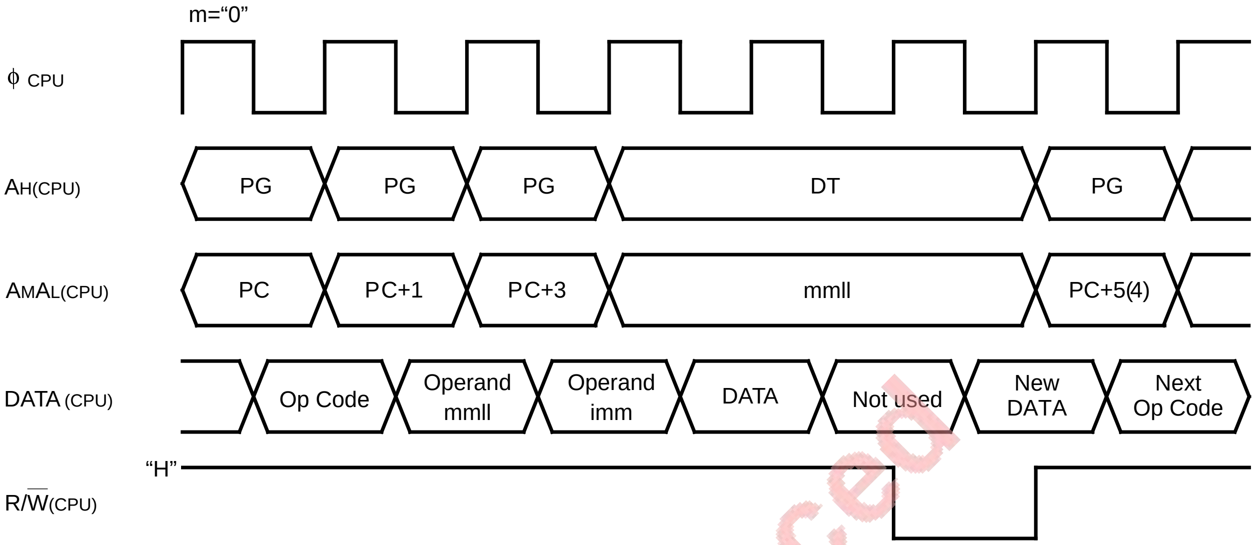
Timing :



Absolute Bit

Instructions : CLB, SEB

Timing :



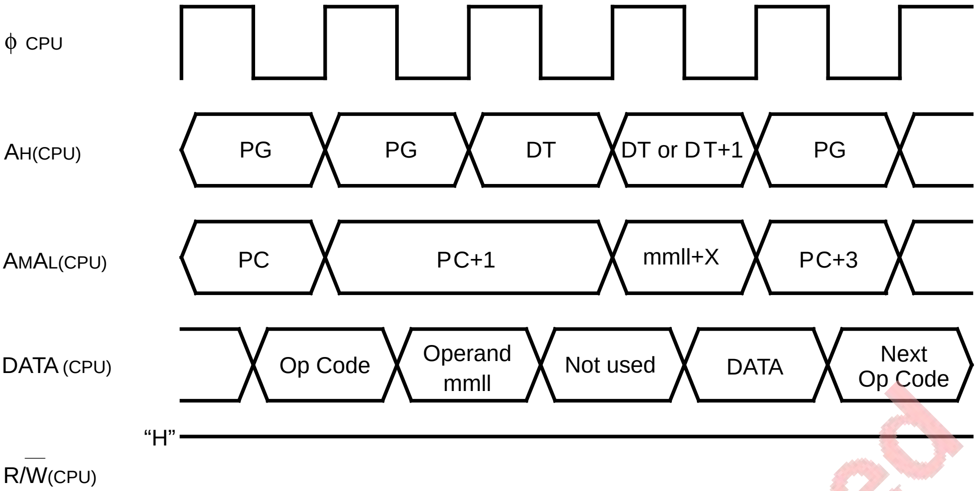
Notes 1: The operand which is fetched at the 3rd cycle is as follows:
When m="0", 2 bytes
When m="1", 1 byte

2: "( )" shows the case of m="1".

Absolute Indexed X

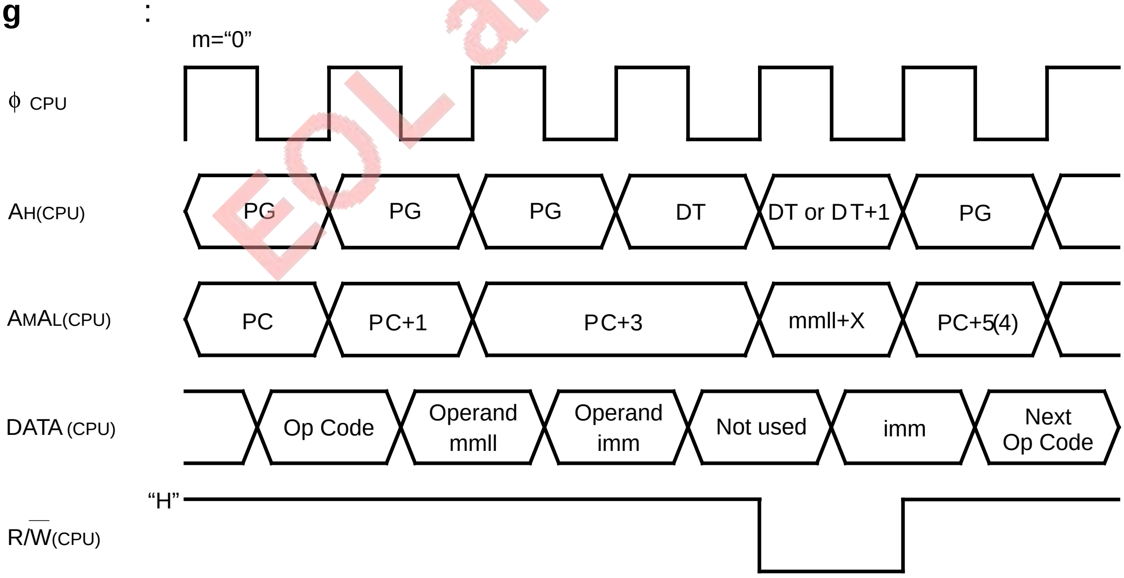
Instructions : ADC, AND, CMP, EOR, LDA, LDY, ORA, SBC

Timing :



Instructions : LDM

Timing :



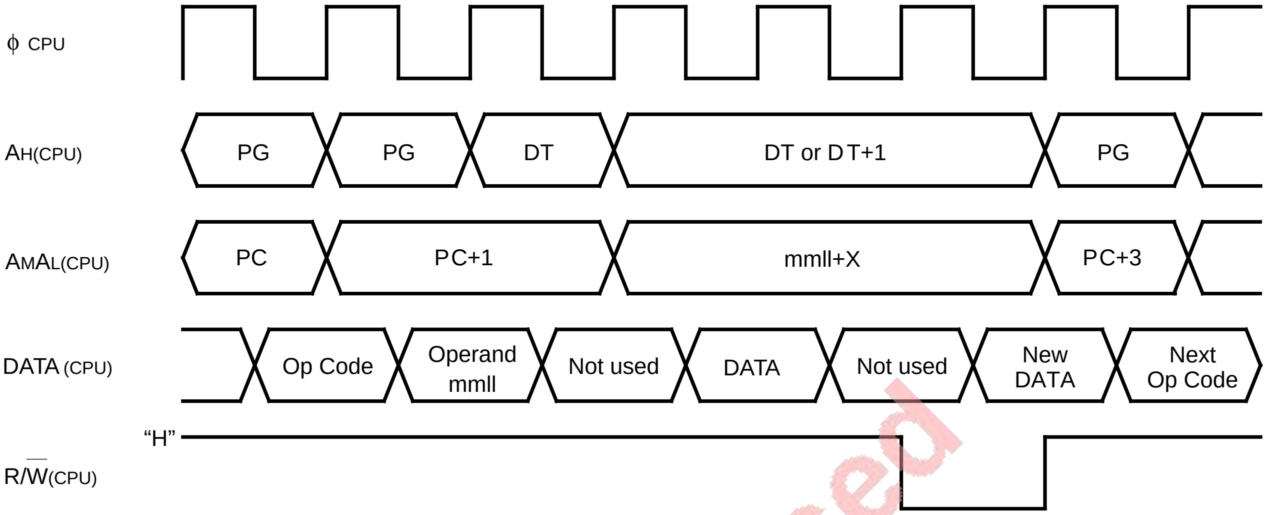
Notes 1: Each of the operand which is fetched at the 3rd cycle and the data which is at the 5th cycle is as follows:
When m="0", 2 bytes
When m="1", 1 byte

2: "( )" shows the case of m="1".

Absolute Indexed X

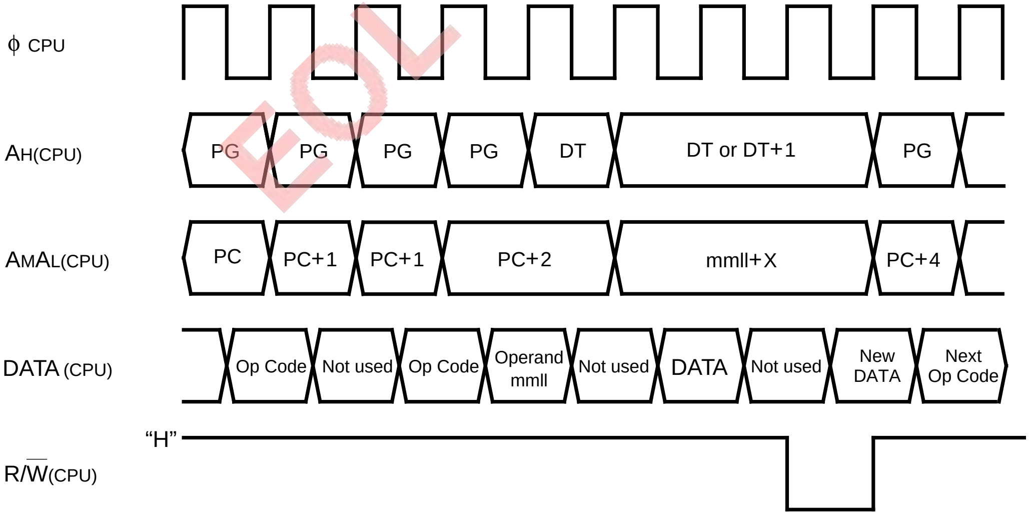
Instructions : ASL, DEC, INC, LSR, ROL, ROR

Timing :



Instructions : ASR

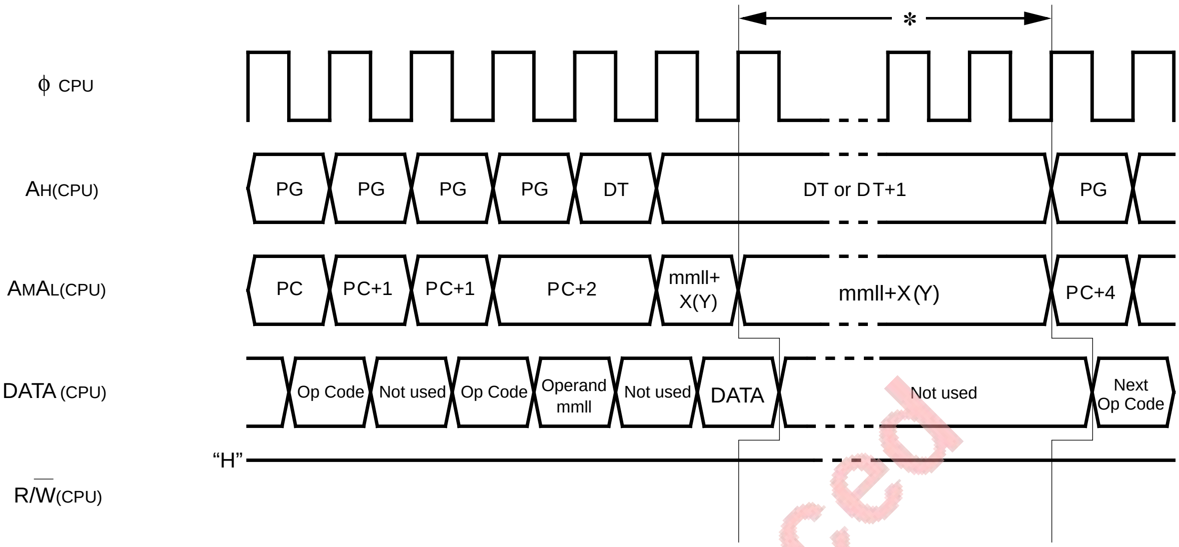
Timing :



Absolute Indexed X
Absolute Indexed Y

Instructions : DIV, DIVS, MPY, MPYS

Timing :



Note: The cycle number during "✻" is shown as the following table:

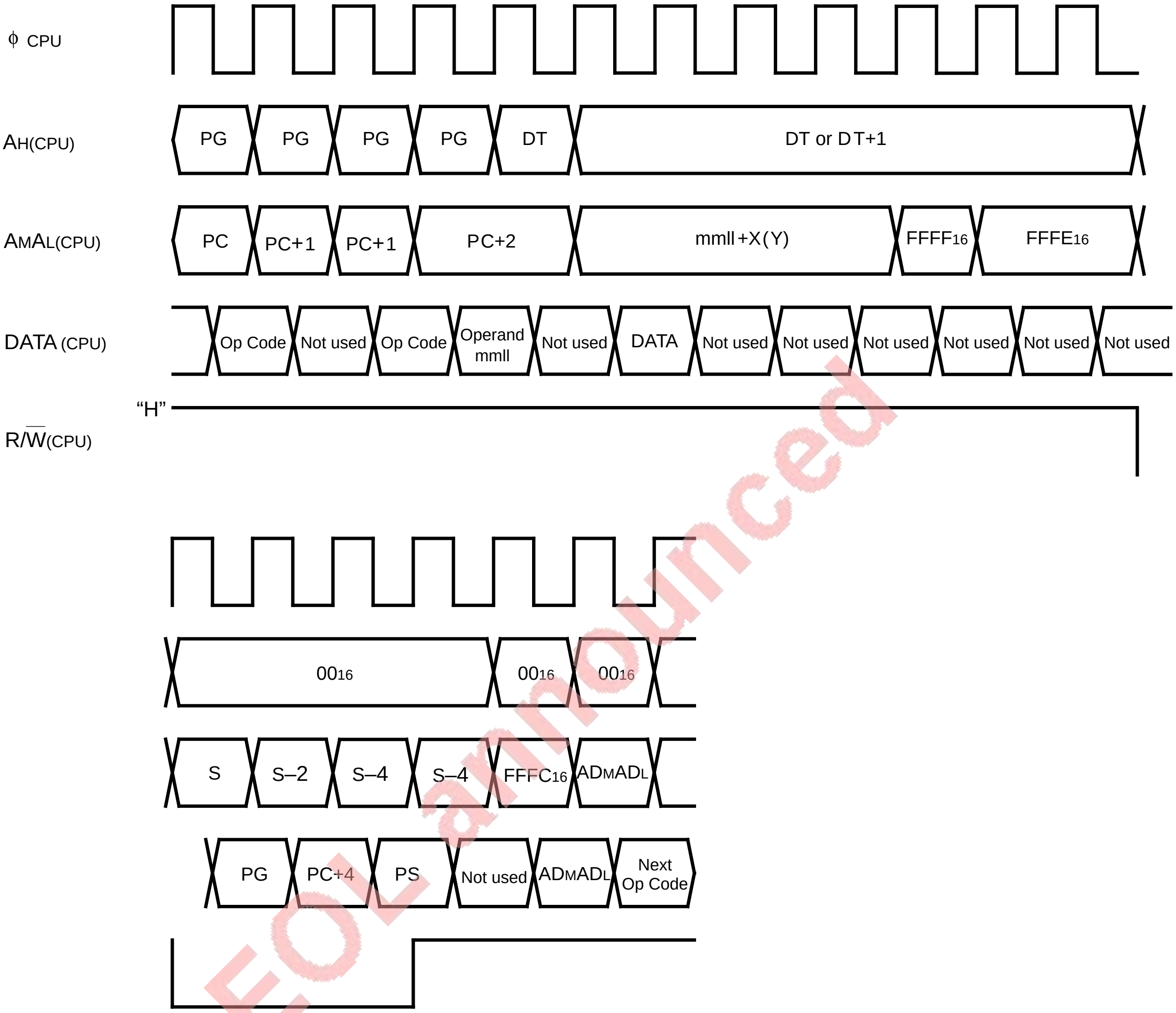
Instruction	Cycle number (φ CPU)	
	m = "0"	m = "1"
DIV	25	17
DIVS	27	19
MPY	8	4
MPYS	8	4

- The contents of AMAL(CPU) during "✻" with DIVS are undefined.
- When each sign of a multiplier and a multiplicand with MPYS is different, the contents of AMAL(CPU) at the last 3 of φ CPU during "✻" are undefined.

Absolute Indexed X
Absolute Indexed Y

Instructions : DIV, DIVS (case of 0 division)

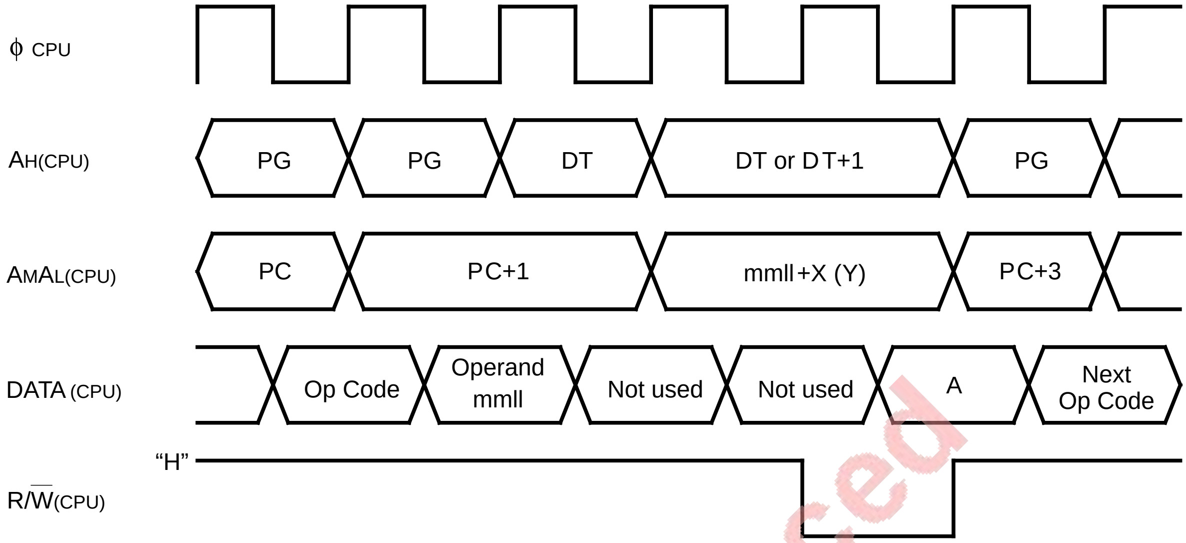
Timing :



Absolute Indexed X
Absolute Indexed Y

Instructions : STA

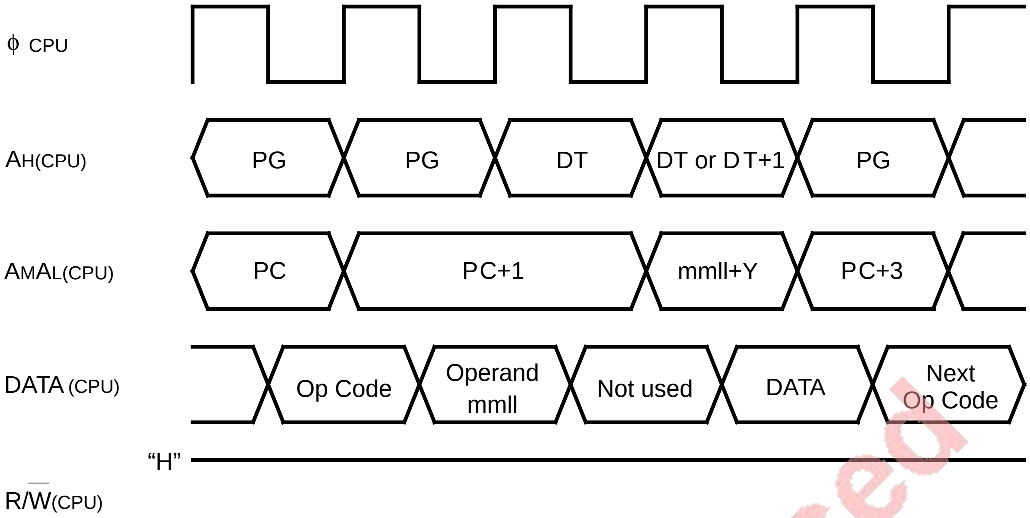
Timing :



Absolute Indexed Y

Instructions : ADC, AND, CMP, EOR, LDA, LDX, ORA, SBC

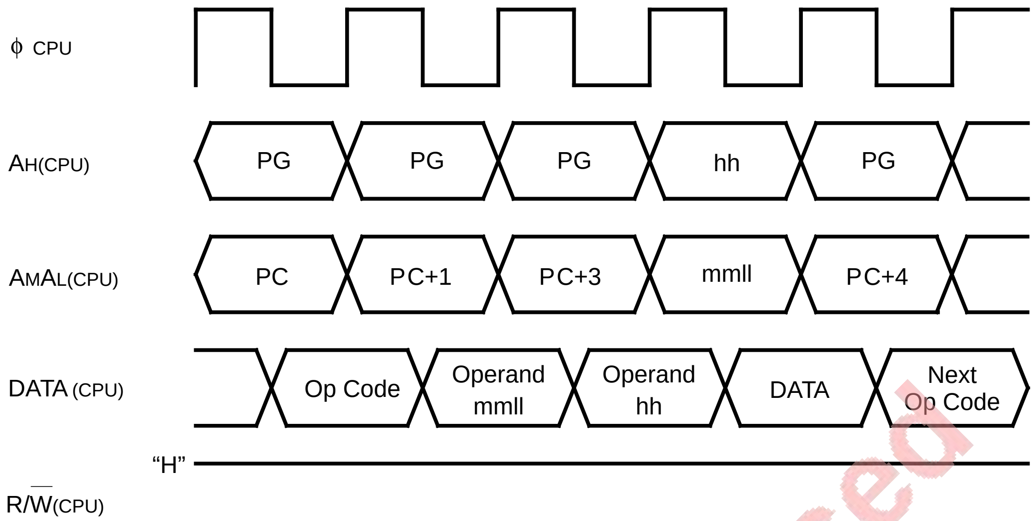
Timing :



Absolute Long

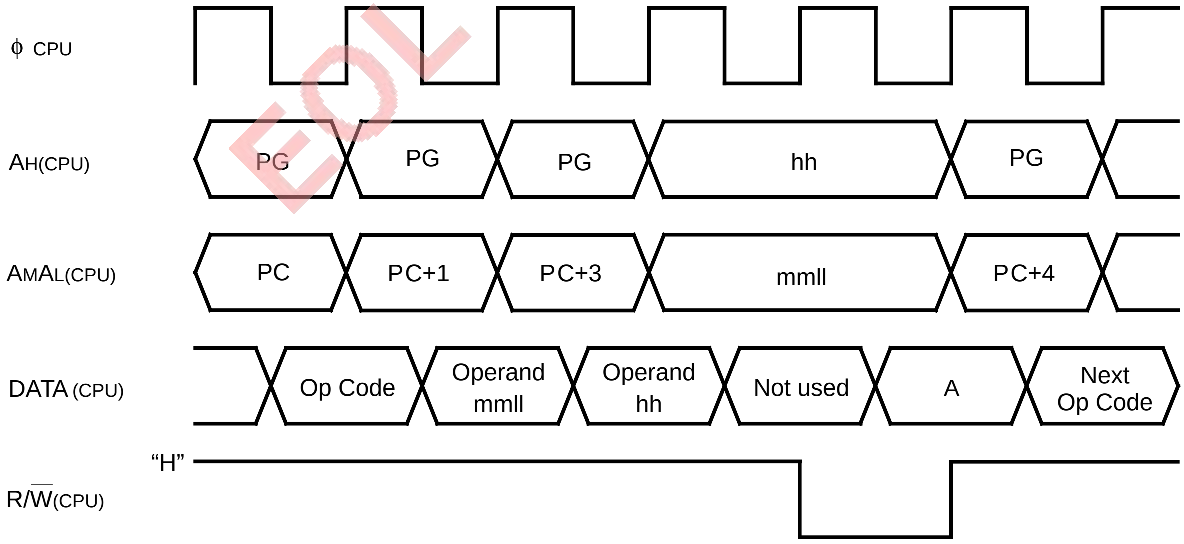
Instructions : ADC, AND, CMP, EOR, LDA, ORA, SBC

Timing :



Instructions : STA

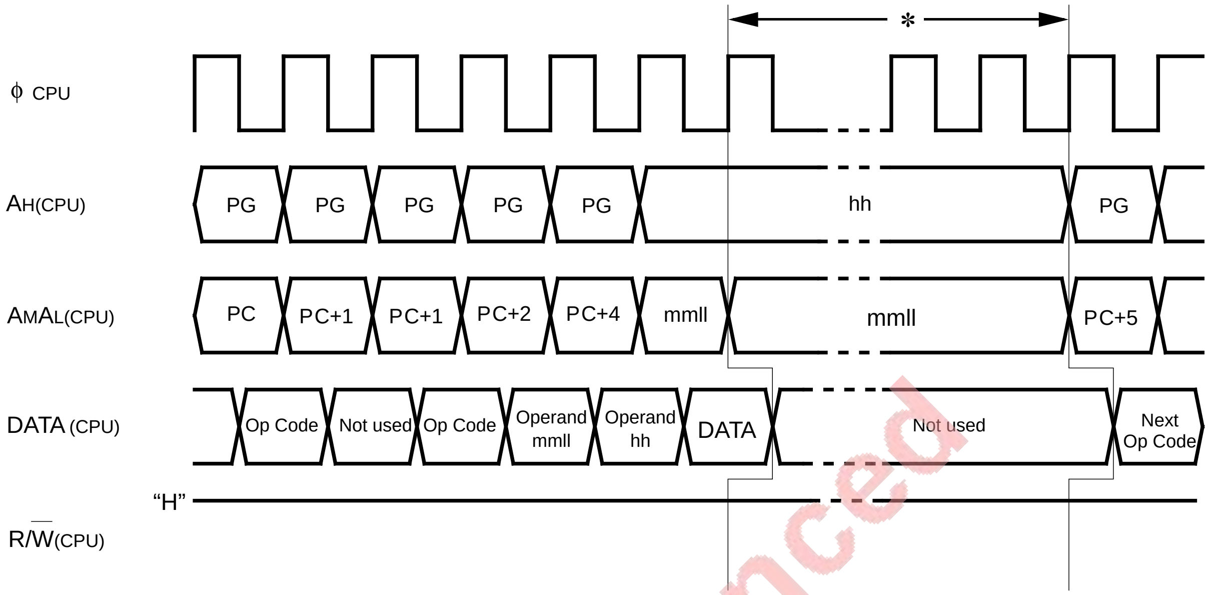
Timing :



Absolute Long

Instructions : DIV, DIVS, MPY, MPYS

Timing :



Note: The cycle number during "✻" is shown as the following table:

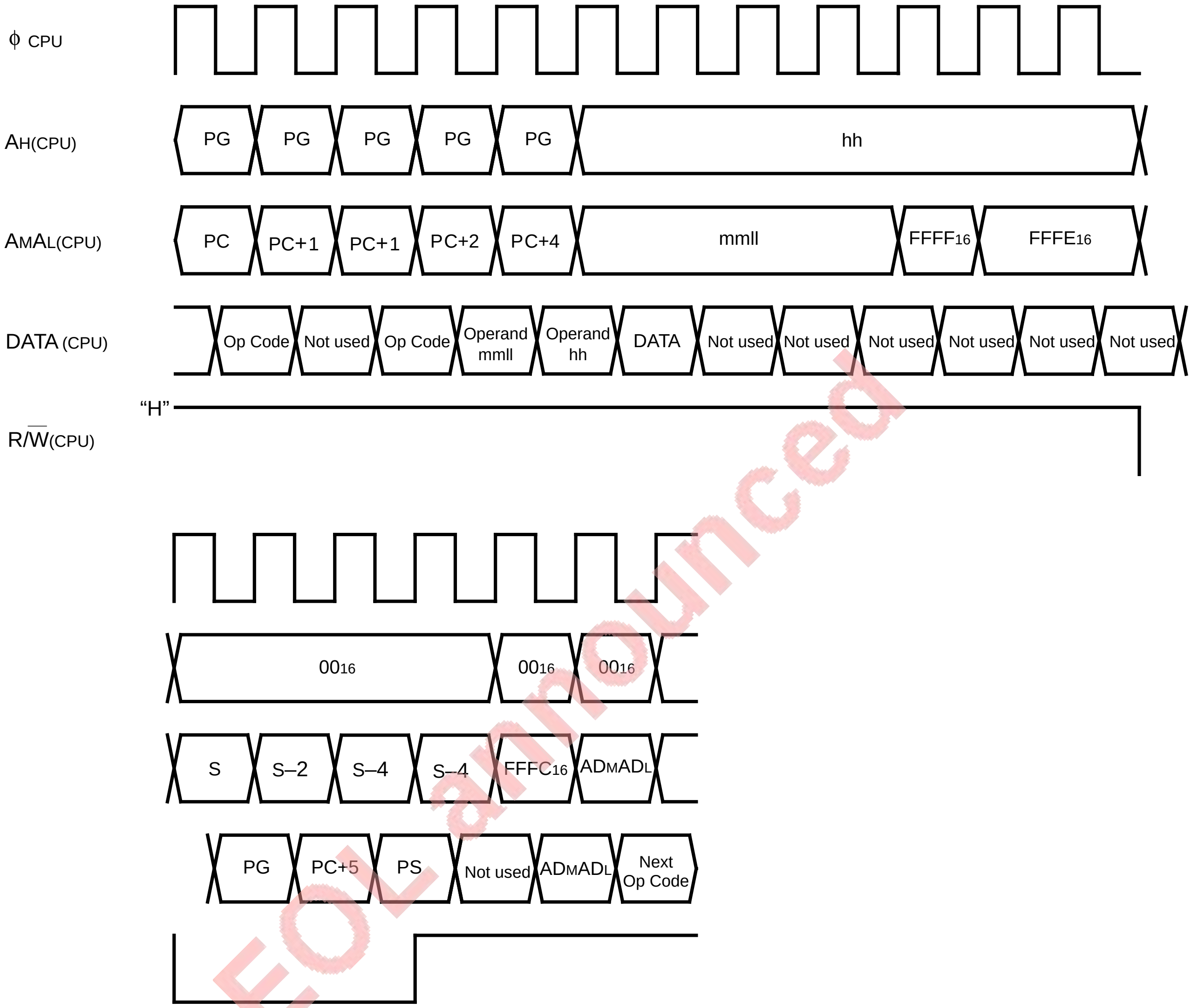
Instruction	Cycle number (φ CPU)	
	m = "0"	m = "1"
DIV	25	17
DIVS	27	19
MPY	8	4
M PYS	8	4

- The contents of AMAL(CPU) during "✻" with DIVS are undefined.
- When each sign of a multiplier and a multiplicand with MPYS is different, the contents of AMAL(CPU) at the last 3 of φ CPU during "✻" are undefined.

Absolute Long

Instructions : DIV , DIVS (case of 0 division)

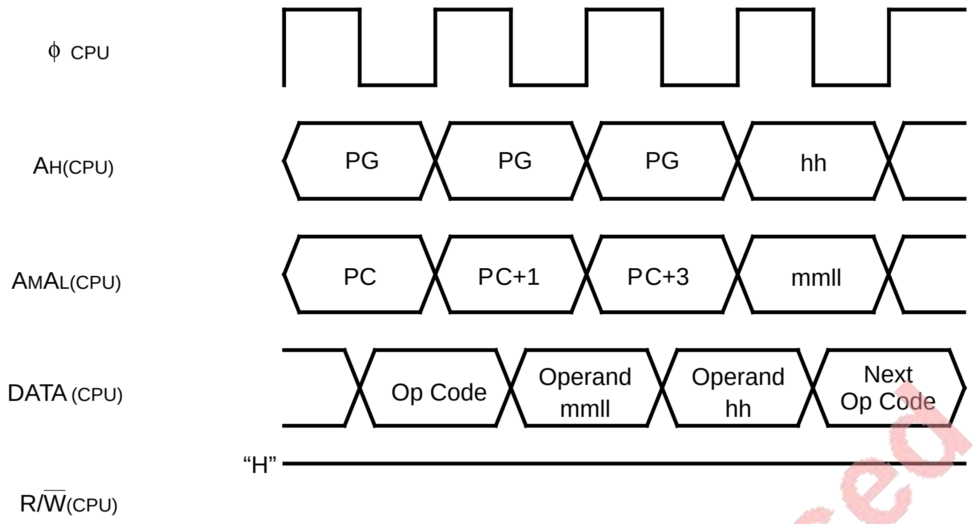
Timing :



Absolute Long

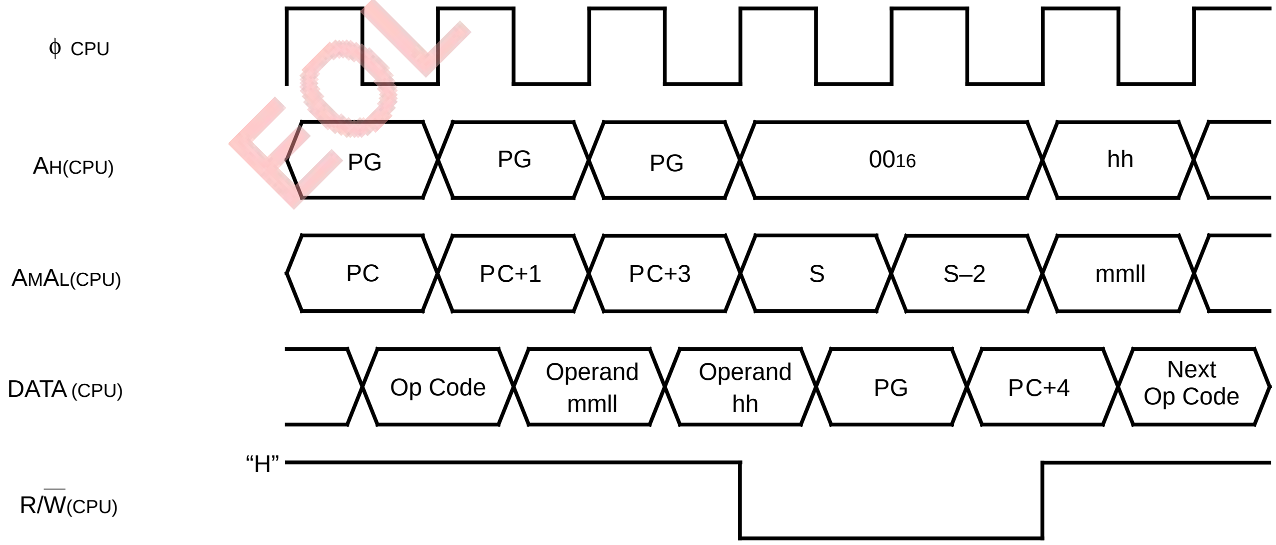
Instructions : JMP

Timing :



Instructions : JSR

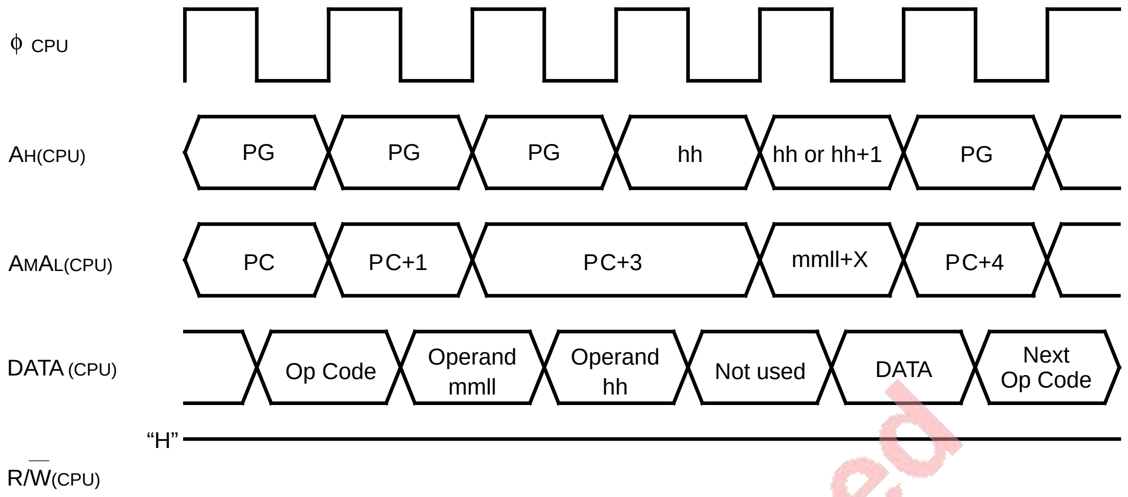
Timing :



Absolute Long Indexed X

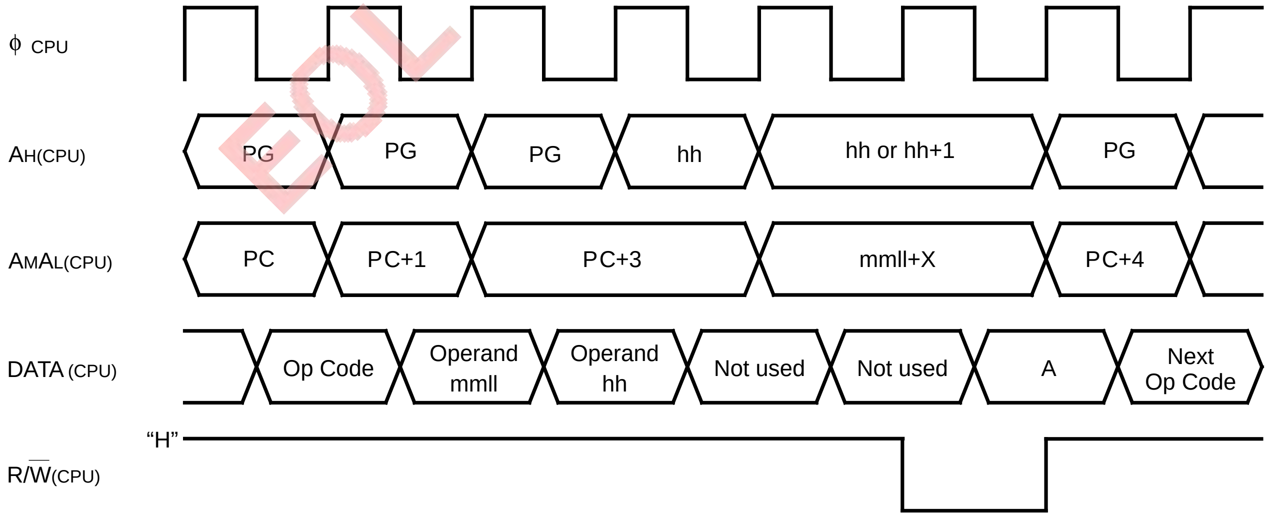
Instructions : ADC, AND, CMP, EOR, LDA, ORA, SBC

Timing :



Instructions : STA

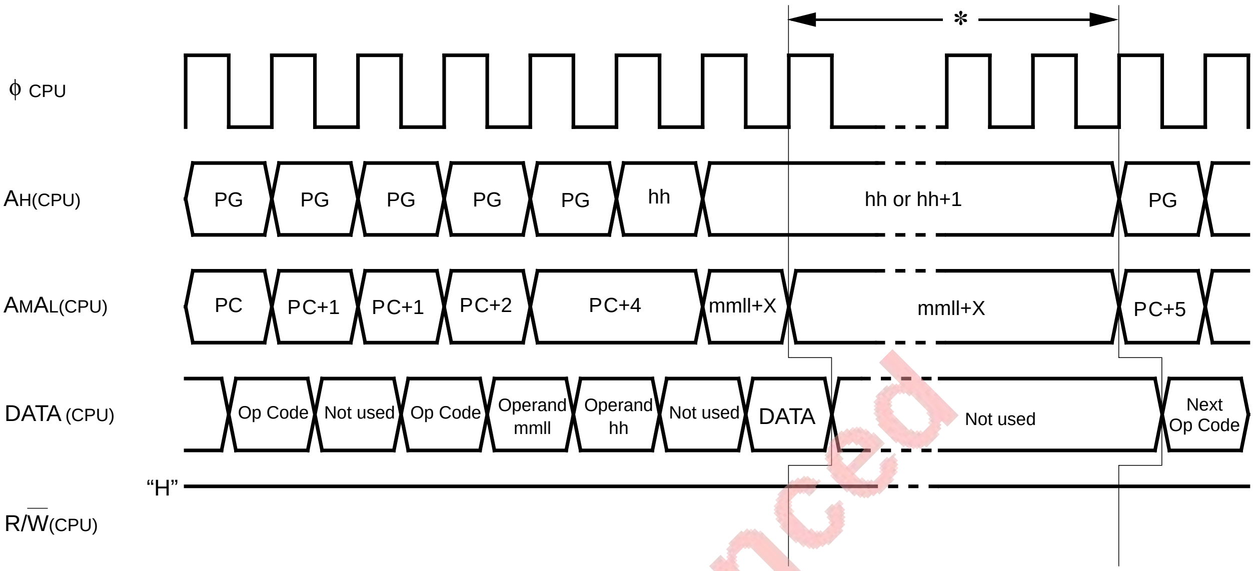
Timing :



Absolute Long Indexed X

Instructions : DIV, DIVS, MPY, MPYS

Timing :



Note: The cycle number during "✻" is shown as the following table:

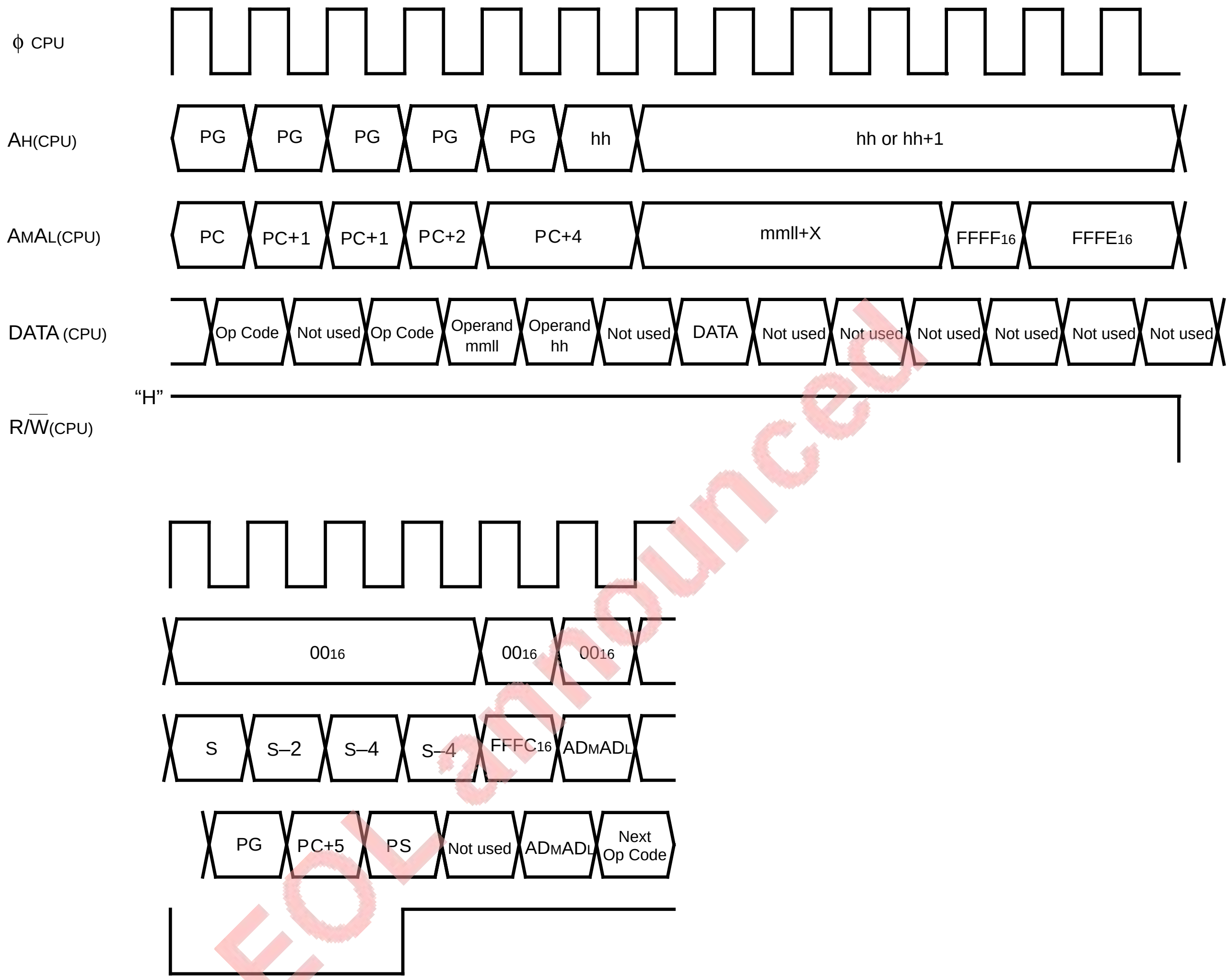
Instruction	Cycle number (φ CPU)	
	m = "0"	m = "1"
DIV	25	17
DIVS	27	19
MPY	8	4
MPYS	8	4

- The contents of AMAL(CPU) during "✻" with DIVS are undefined.
- When each sign of a multiplier and a multiplicand with MPYS is different, the contents of AMAL(CPU) at the last 3 of φ CPU during "✻" are undefined.

Absolute Long Indexed X

Instructions : DIV, DIVS (case of 0 division)

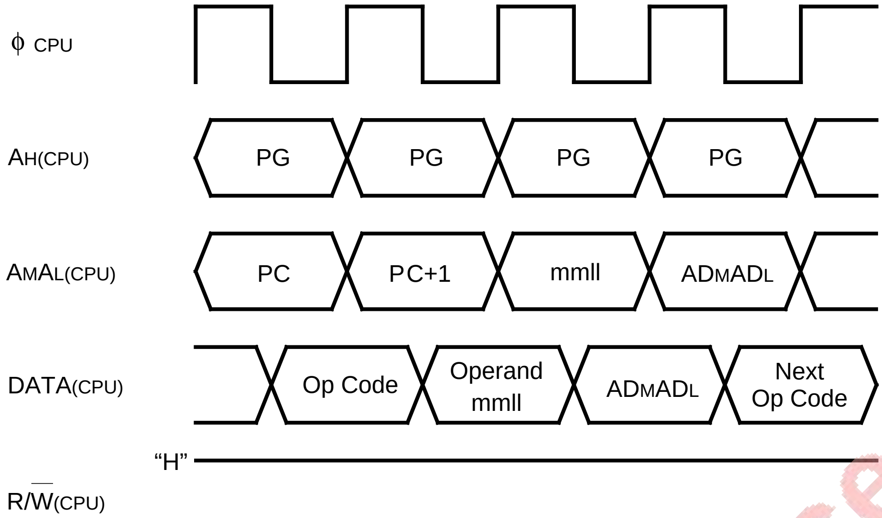
Timing :



Absolute Indirect

Instructions : JMP

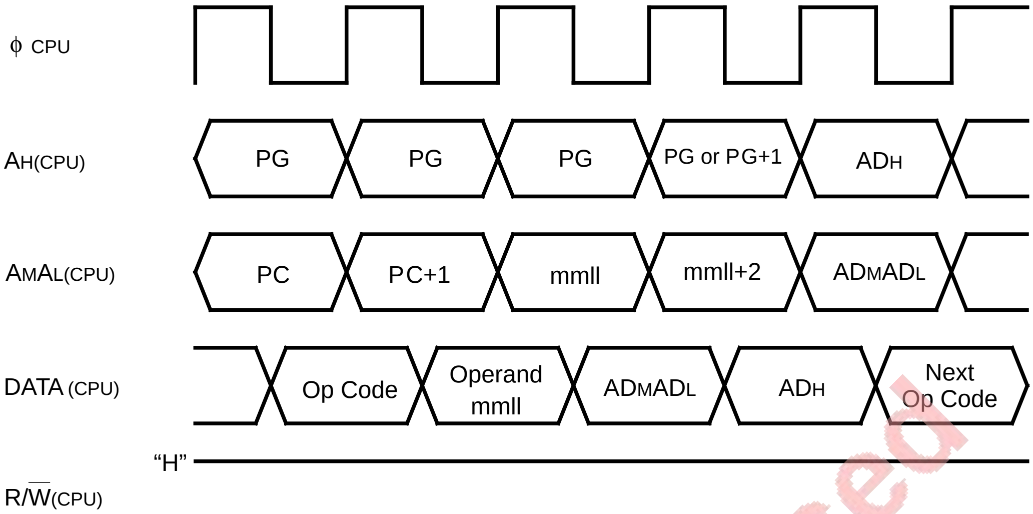
Timing :



Absolute Indirect Long

Instructions : JMP

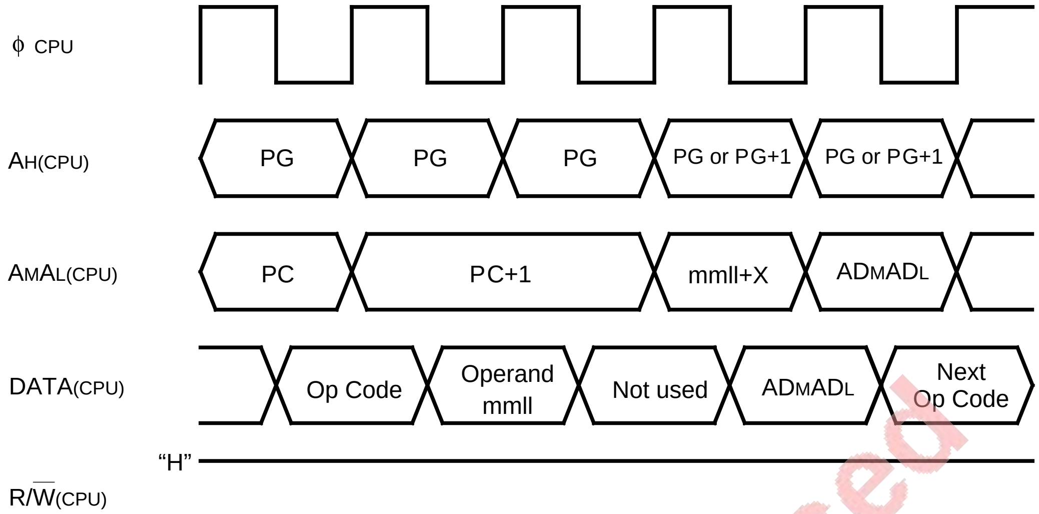
Timing :



Absolute Indexed X Indirect

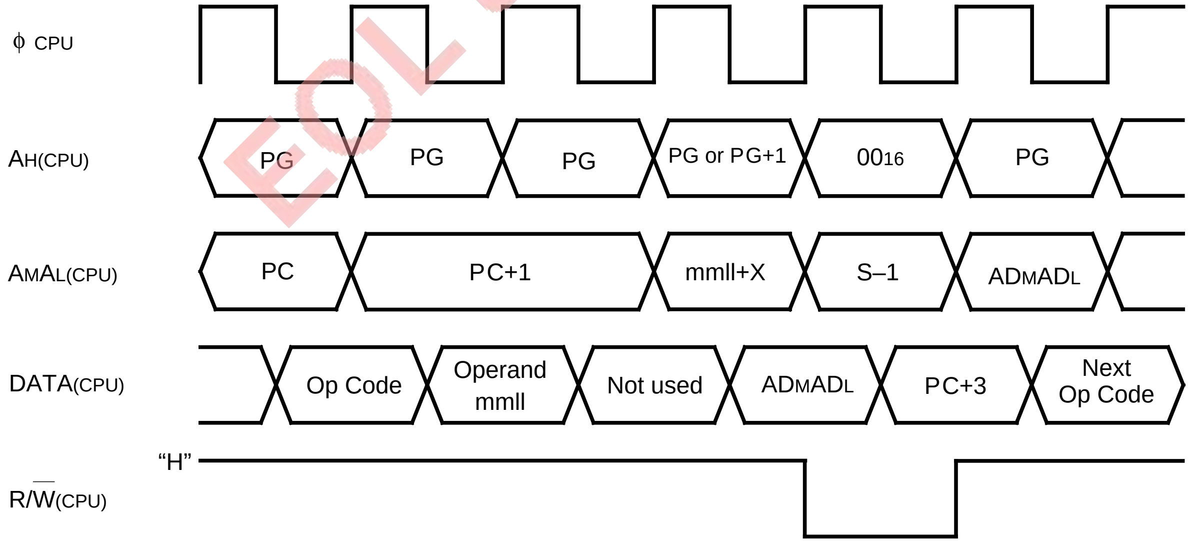
Instructions : JMP

Timing :



Instructions : JSR

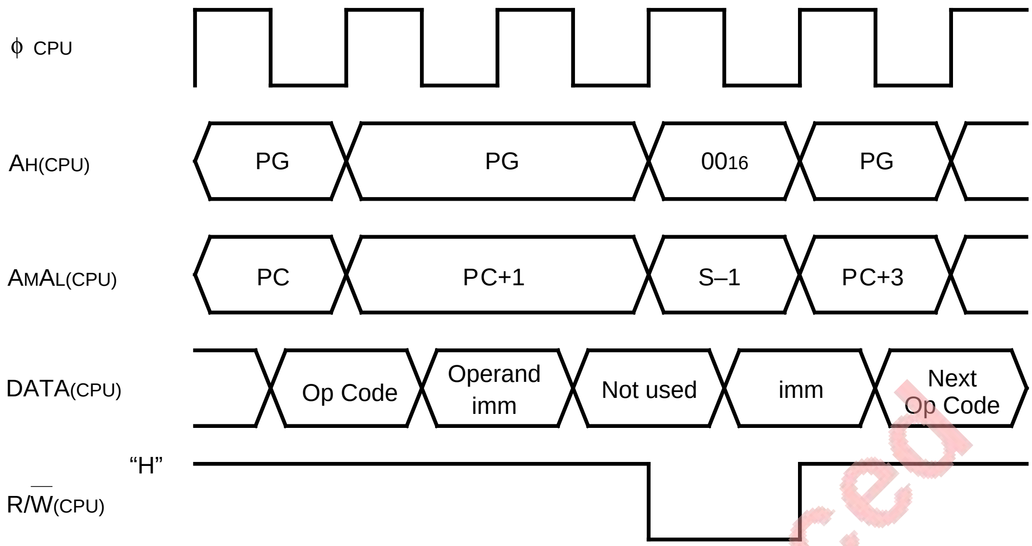
Timing :



Stack

Instructions : PEA

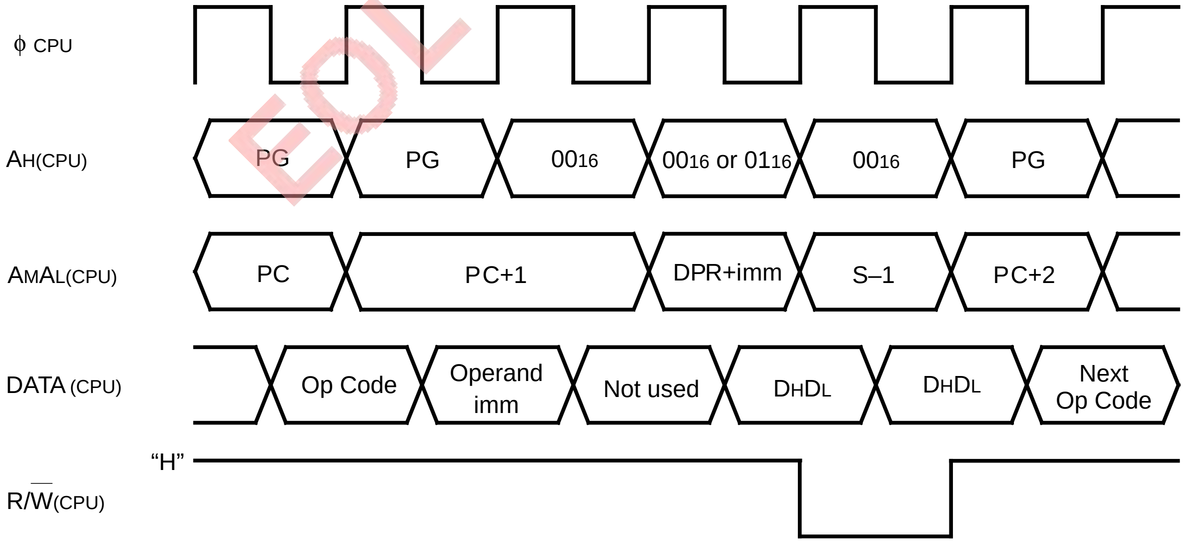
Timing :



Note: Each of the operand at the 2nd cycle and the data at the 4th cycle is 2 bytes.

Instructions : PEI

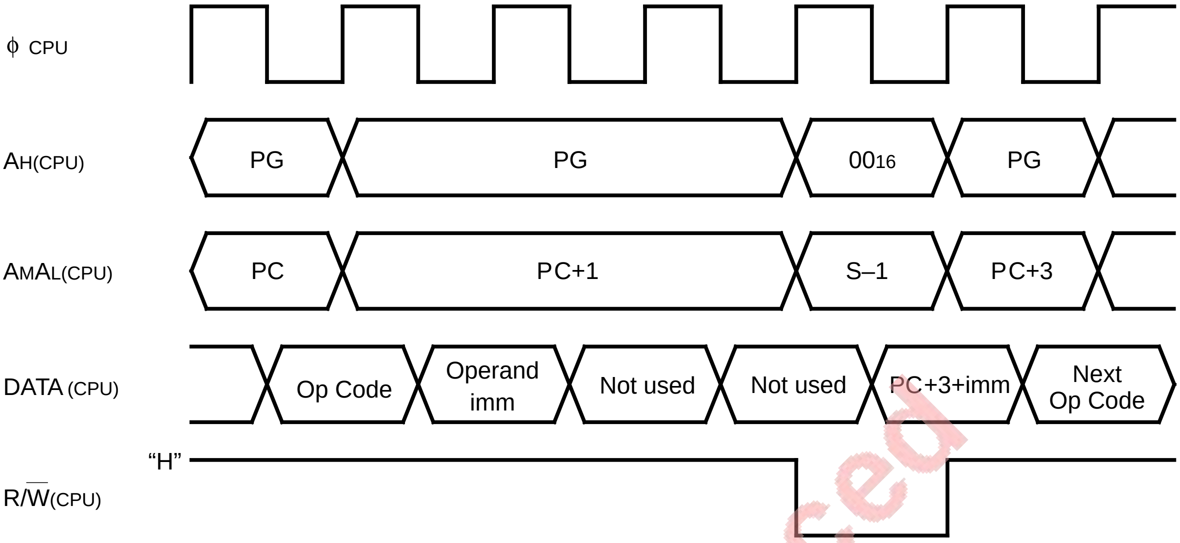
Timing :



Note: The operand at the 2nd cycle is 1 byte.

Instructions : PER

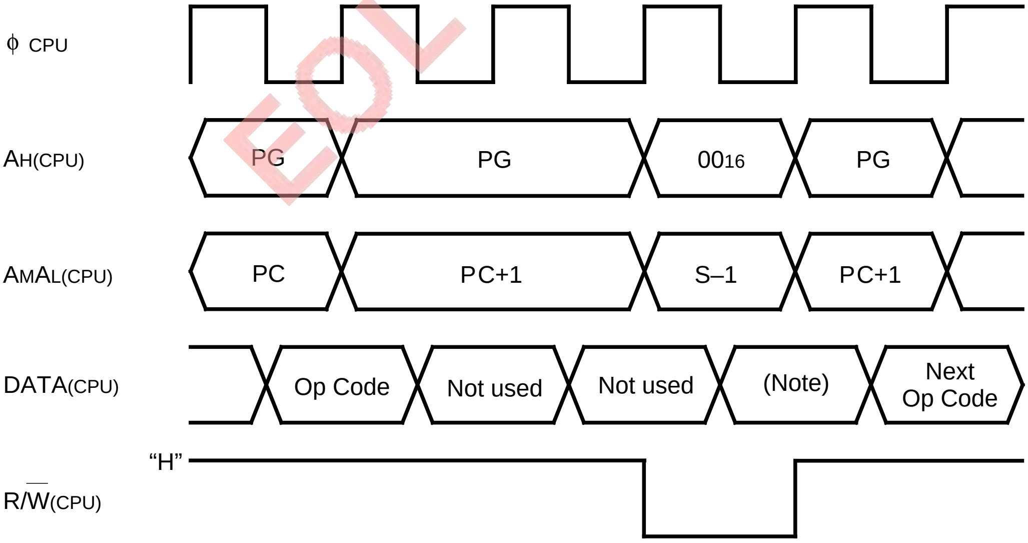
Timing :



Note: Each of the operand at the 2nd cycle and the data at the 5th cycle is 2 bytes.

Instructions : PHA, PHD, PHP, PHX, PHY

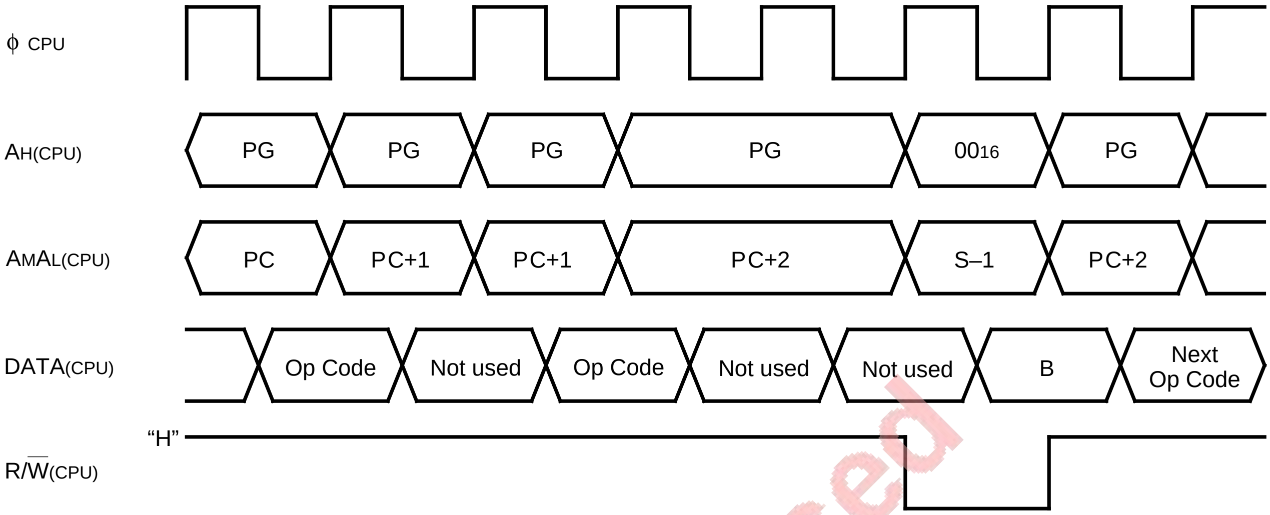
Timing :



Note: A(DPR)(PS)(X)(Y)

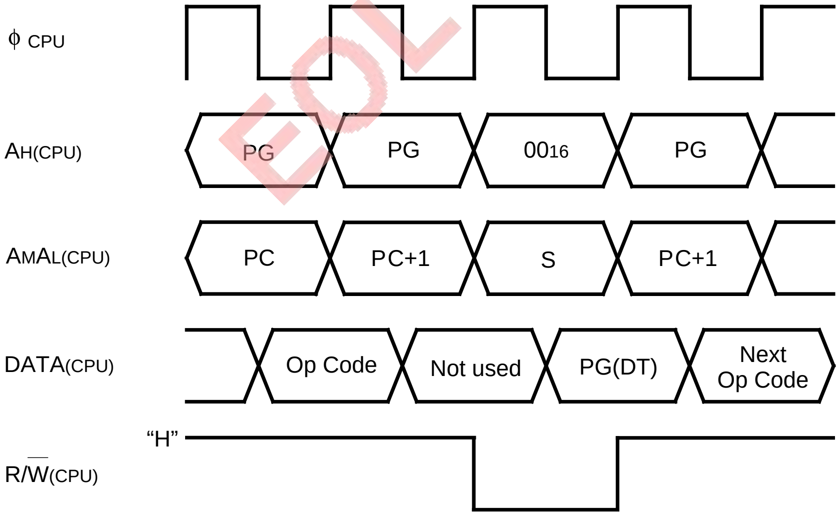
Instructions : PHB

Timing :



Instructions : PHG, PHT

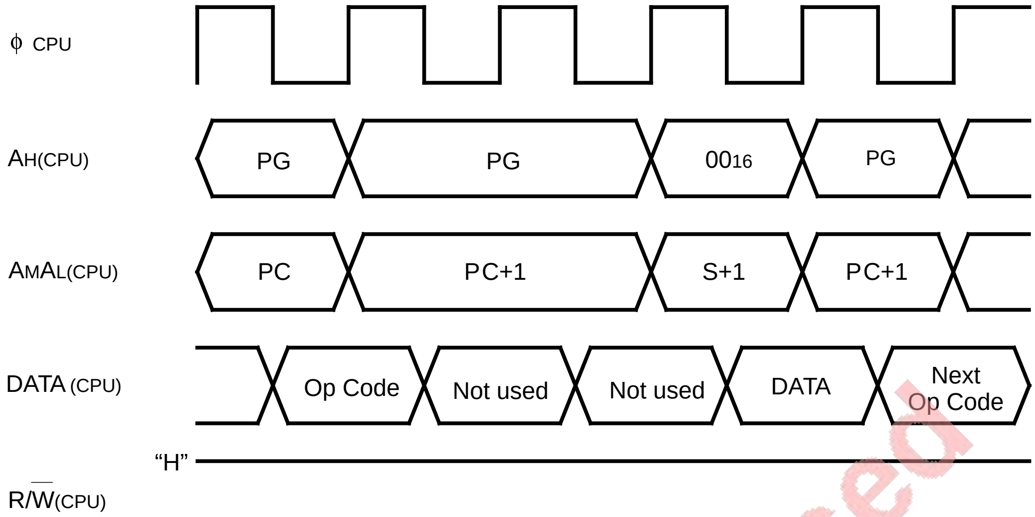
Timing :



Stack

Instructions : PLA, PLD, PLX, PLY

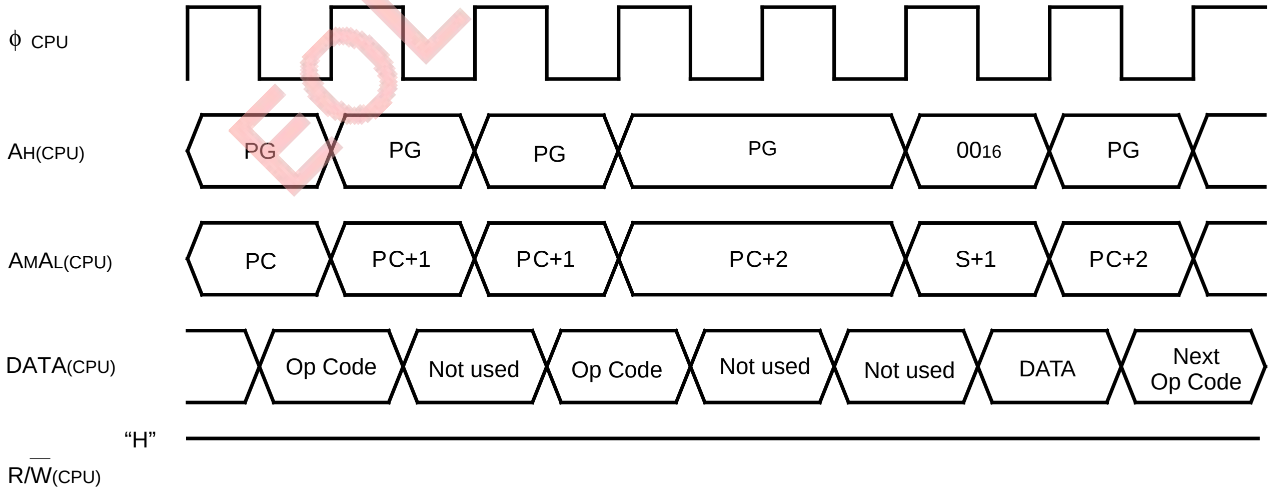
Timing :



Note: The data at the 4th cycle is 2 bytes with PLD.

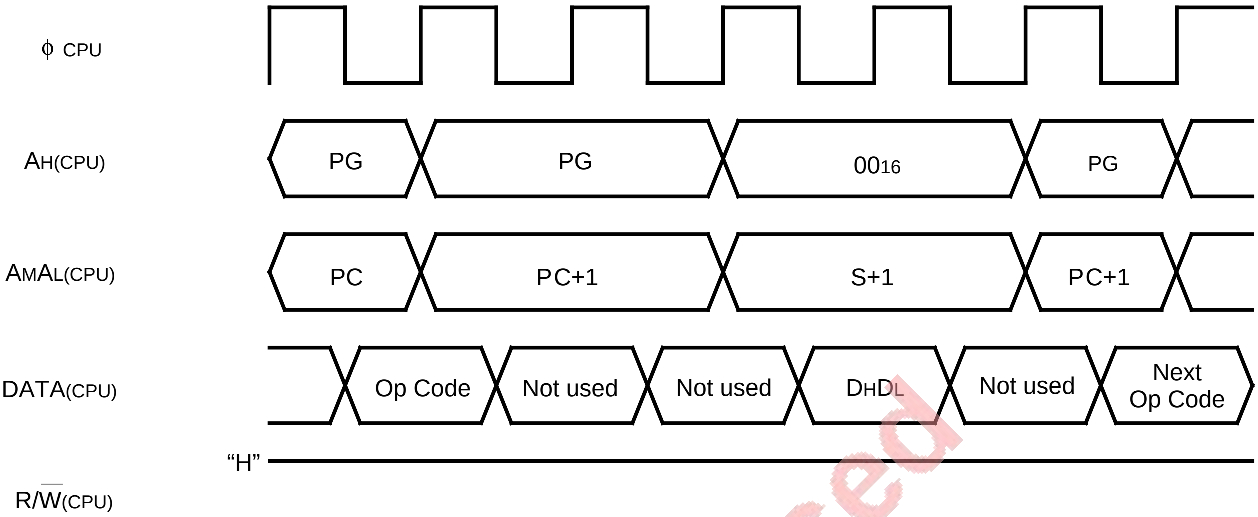
Instructions : PLB

Timing :



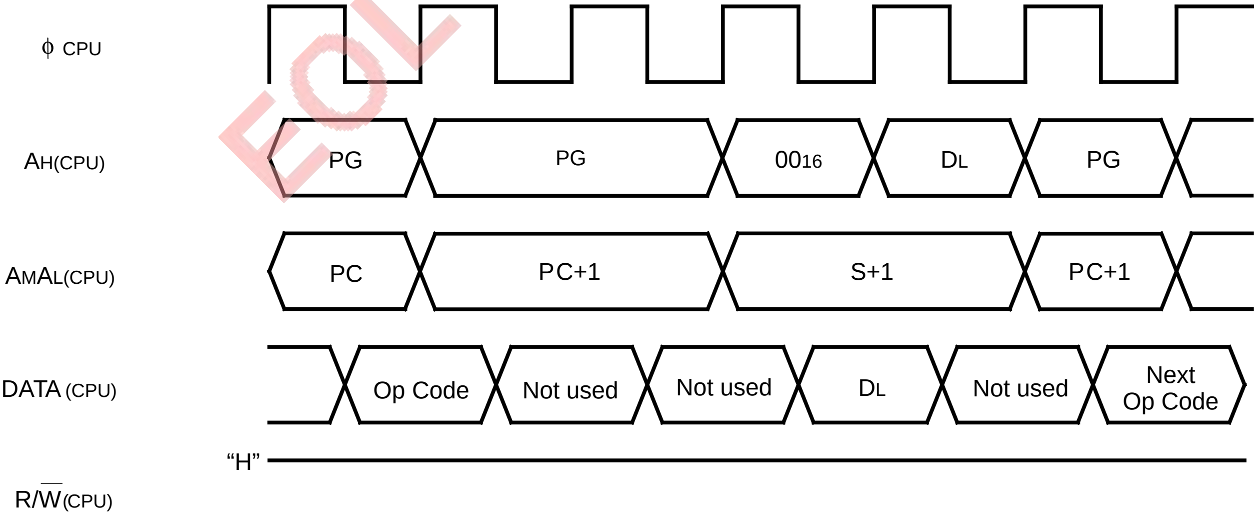
Instructions : PLP

Timing :



Instructions : PLT

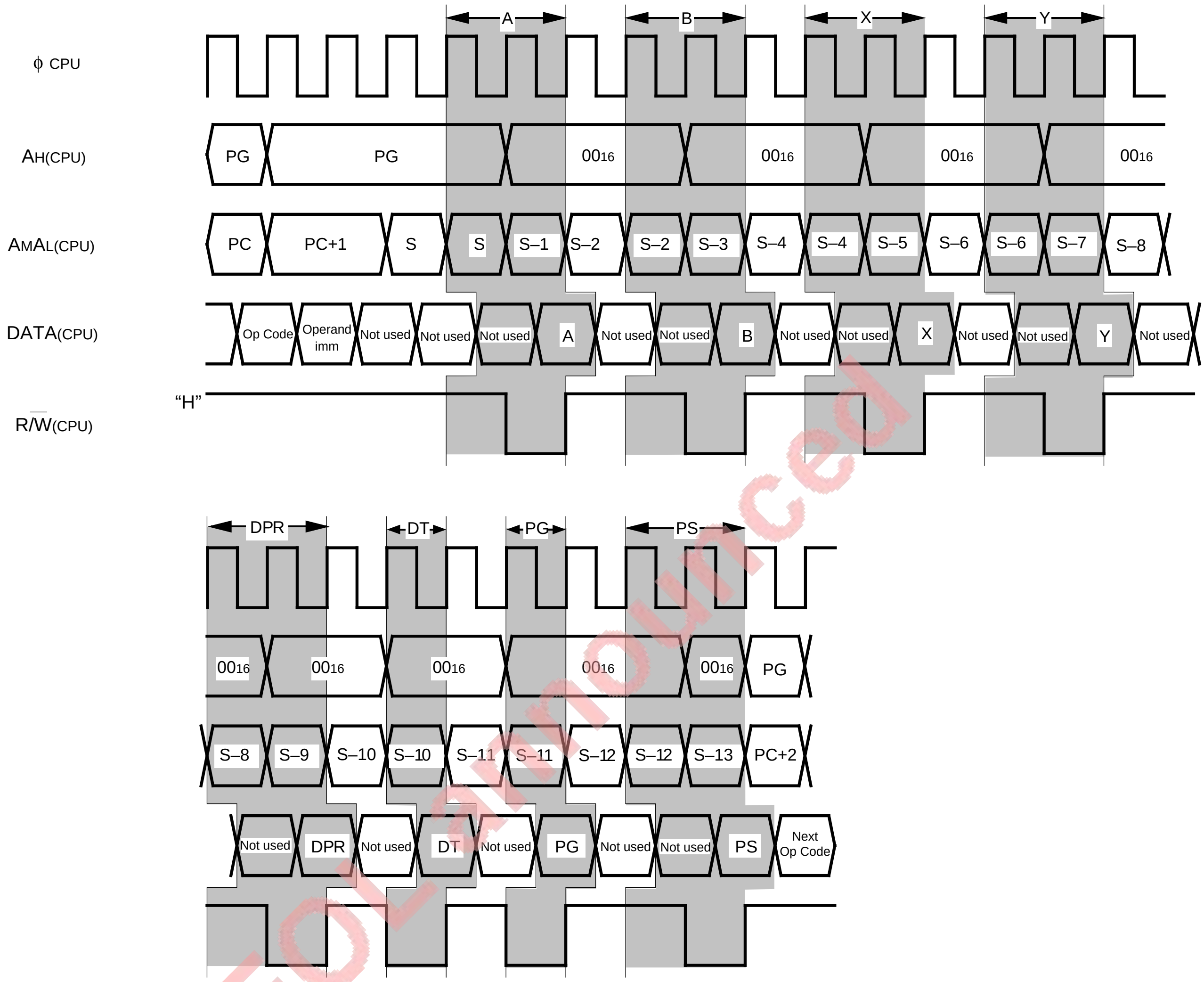
Timing :



Stack

Instructions : PSH

Timing :

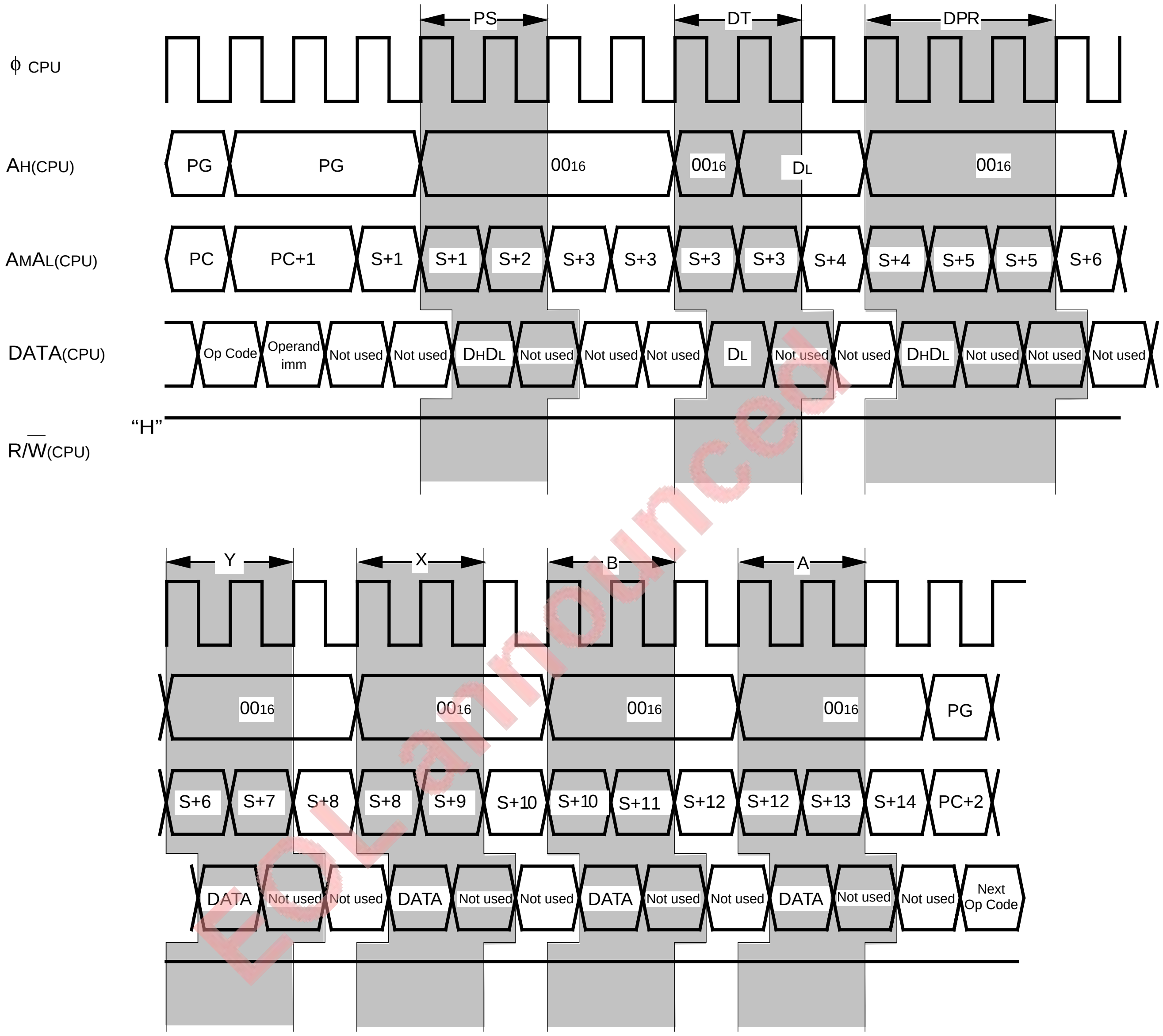


Notes 1: This figure is an example when executing PSH to all registers.
When some of them are not to be done, the cycle "◄—►" corresponding to its register is shortened.

2: The operand at the 2nd cycle is 1 byte.

Instructions : PUL

Timing :



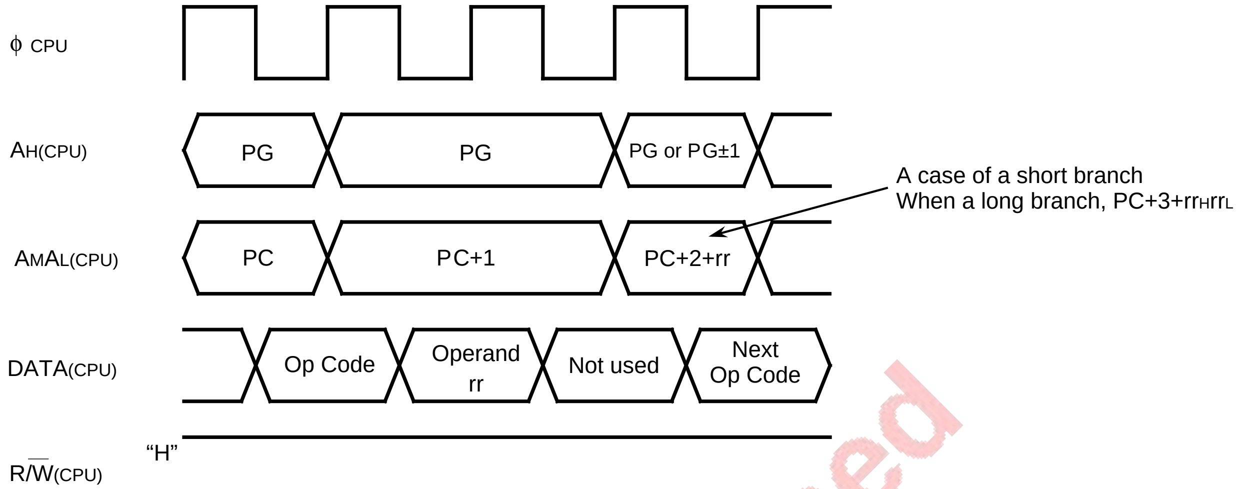
Notes 1: This figure is an example when executing PUL to all registers.
When some of them are not to be done, the cycle "◀━▶" corresponding to its register is shortened.

2: The operand at the 2nd cycle is 1 byte.

Relative

Instructions : BRA

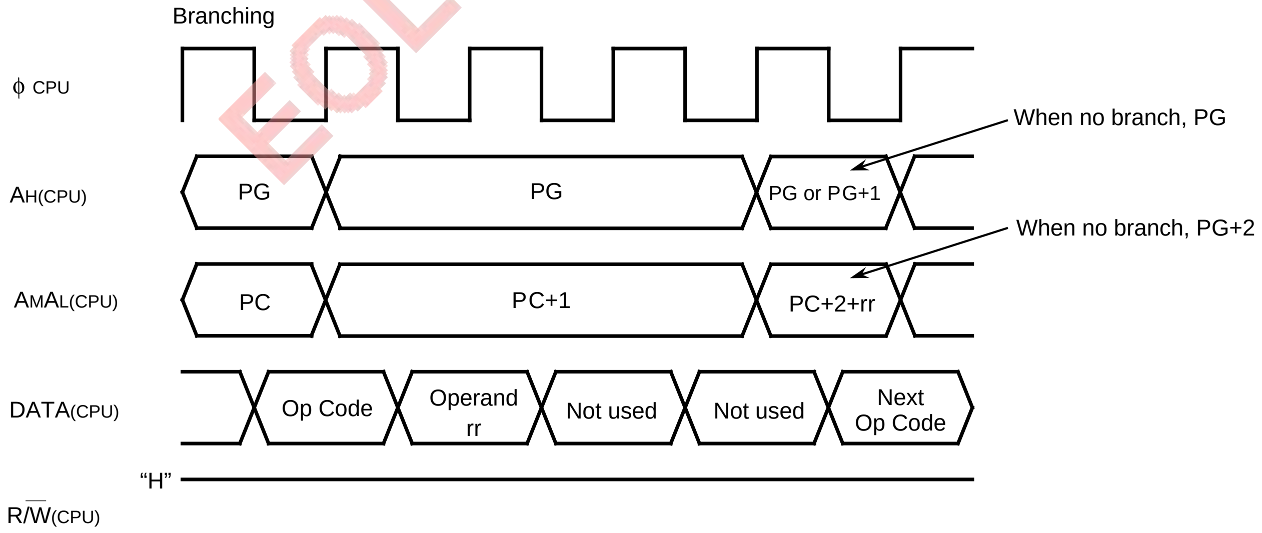
Timing :



Note: The operand at the 2nd cycle is 1 byte in the case of a short relative, and 2 bytes (rrHrrL) in the case of a long relative.

Instructions : BCC, BCS, BEQ, BMI, BNE, BPL, BVC, BVS

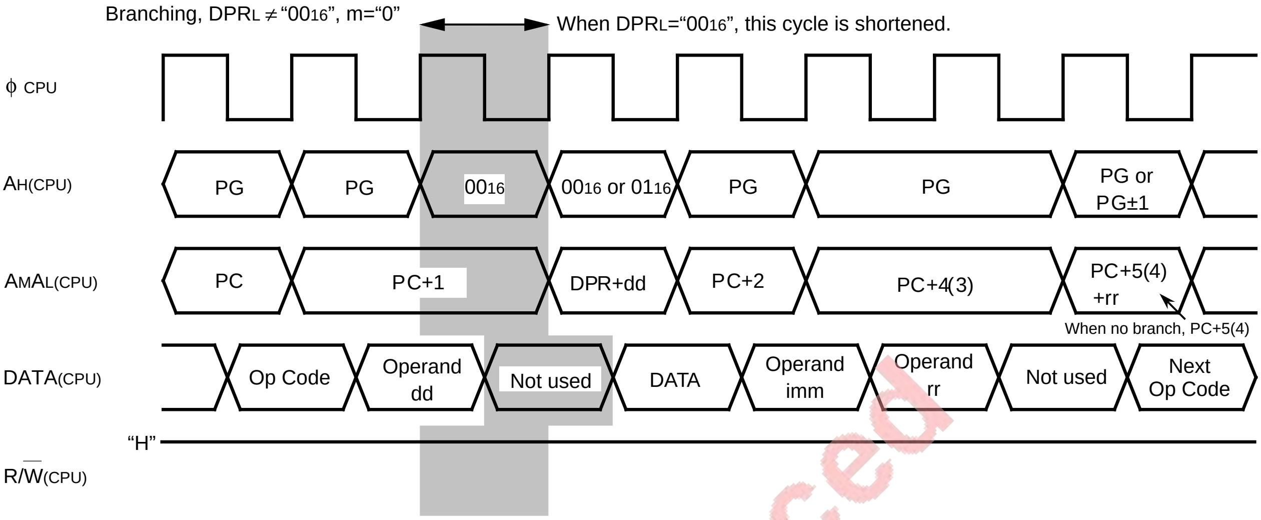
Timing :



Direct Bit Relative

Instructions : BBC, BBS

Timing :



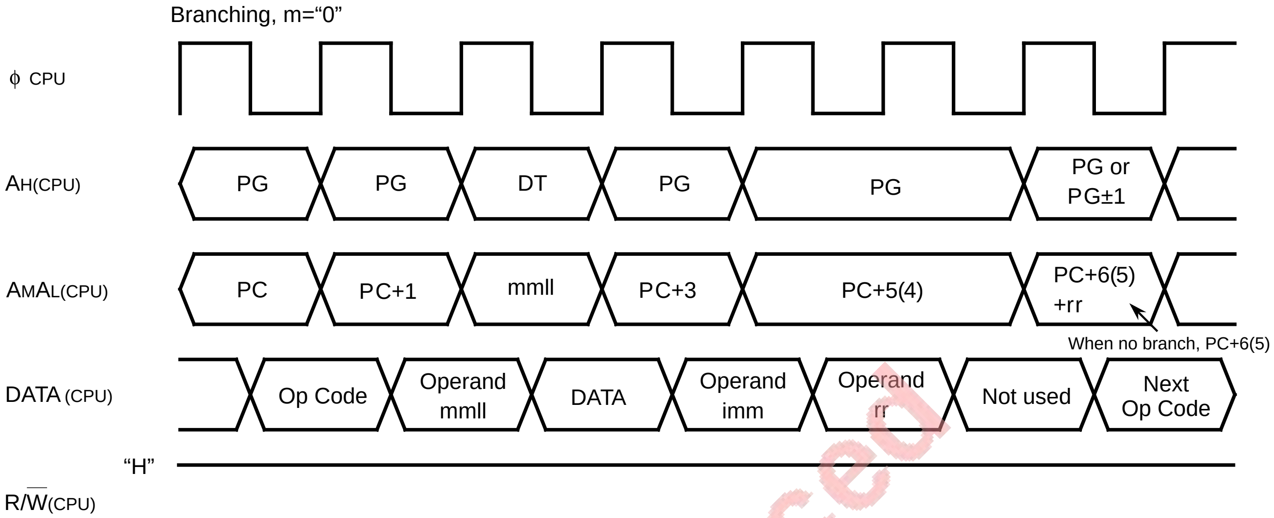
Notes 1: The operand which is fetched at the 5th cycle is as follows:
When m="0", 2 bytes
When m="1", 1 byte

2: "( )" shows the case of m="1".

Absolute Bit Relative

Instructions : BBC, BBS

Timing :



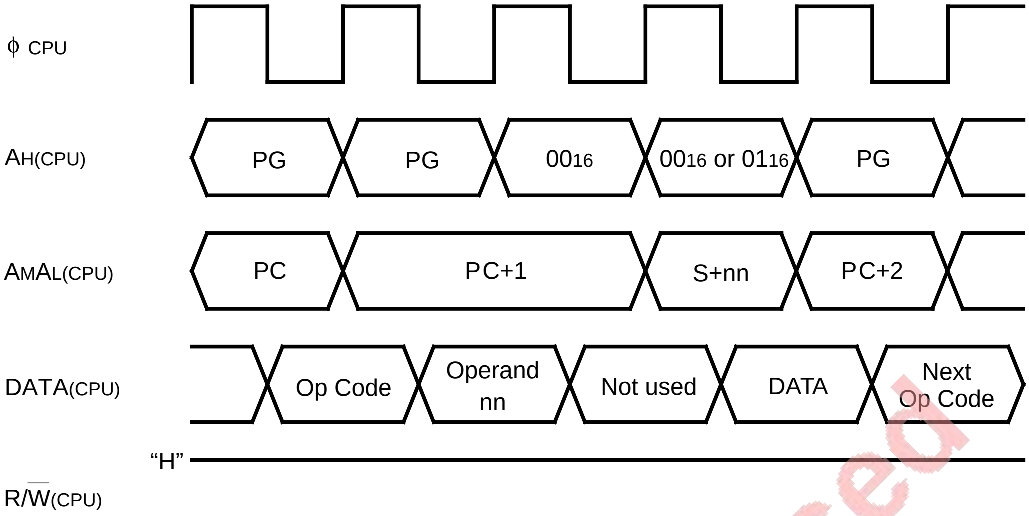
Notes 1: The operand which is fetched at the 4th cycle is as follows:
When m="0", 2 bytes
When m="1", 1 byte

2: "( )" shows the case of m="1".

Stack Pointer Relative

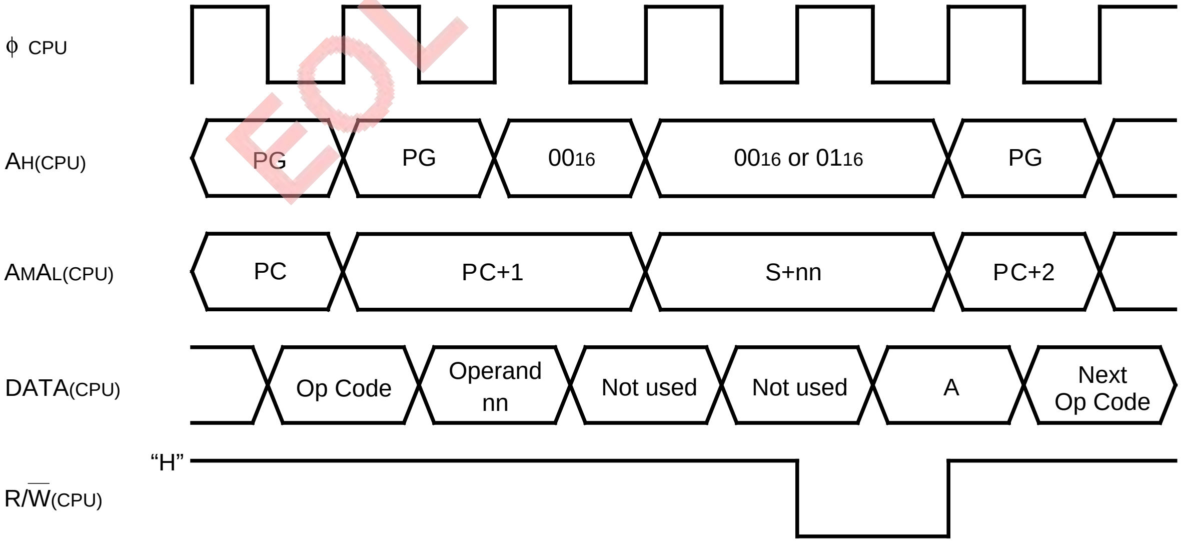
Instructions : ADC, AND, CMP, EOR, LDA, ORA, SBC

Timing :



Instructions : STA

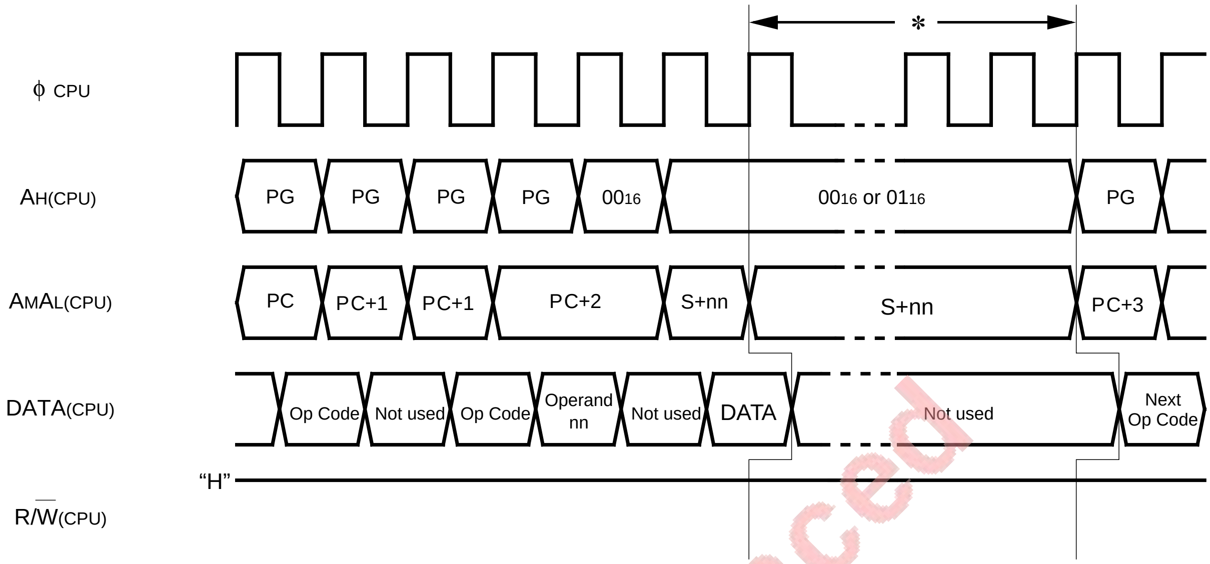
Timing :



Stack Pointer Relative

Instructions : DIV, DIVS, MPY, MPYS

Timing :



Note: The cycle number during "✻" is shown as the following table:

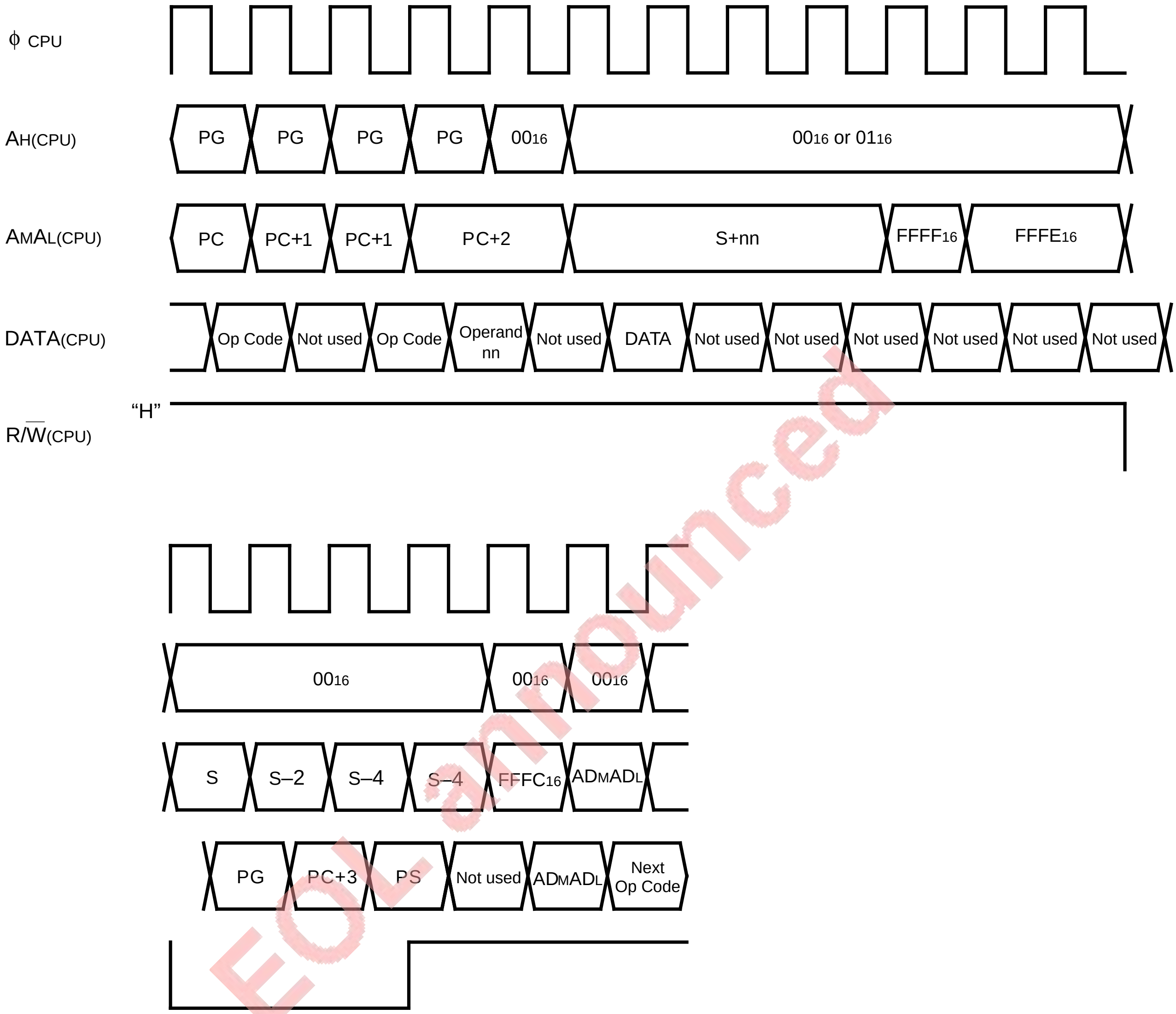
Instruction	Cycle number (φ CPU)	
	m = "0"	m = "1"
DIV	25	17
DIVS	27	19
MPY	8	4
MPYS	8	4

- The contents of AMAL(CPU) during "✻" with DIVS are undefined.
- When each sign of a multiplier and a multiplicand with MPYS is different, the contents of AMAL(CPU) at the last 3 of φ CPU during "✻" are undefined.

Stack Pointer Relative

Instructions : DIV, DIVS (case of 0 division)

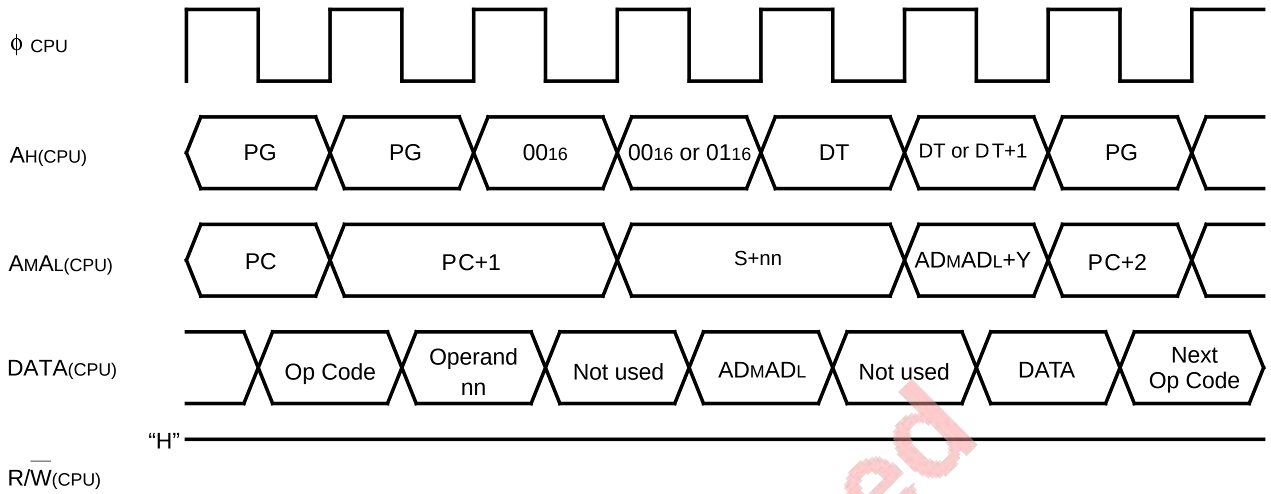
Timing :



Stack Pointer Relative Indirect Indexed Y

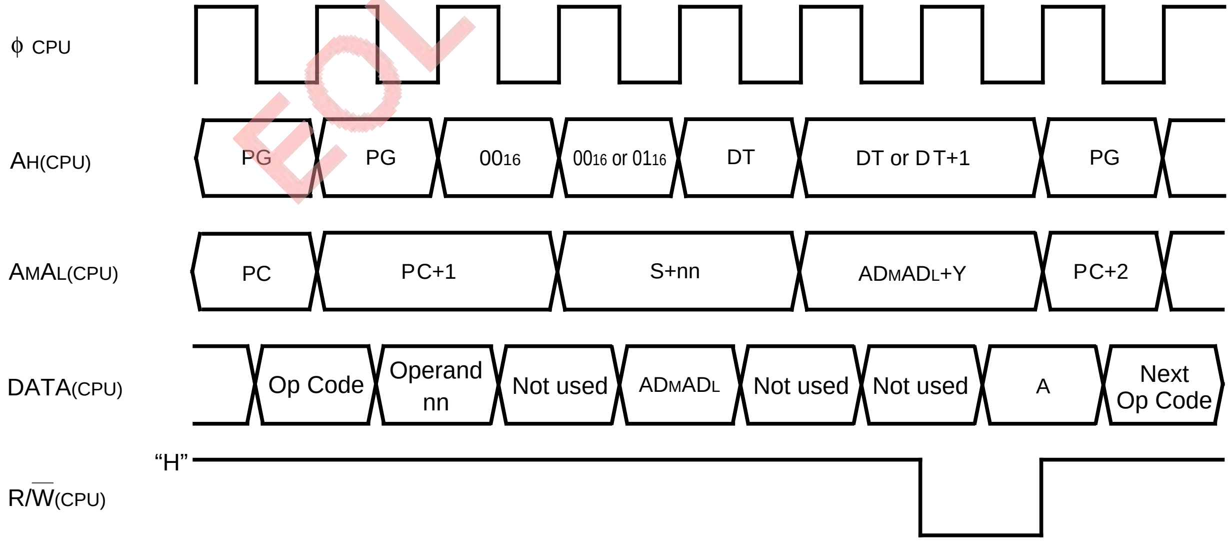
Instructions : ADC, AND, CMP, EOR, LDA, ORA, SBC

Timing :



Instructions : STA

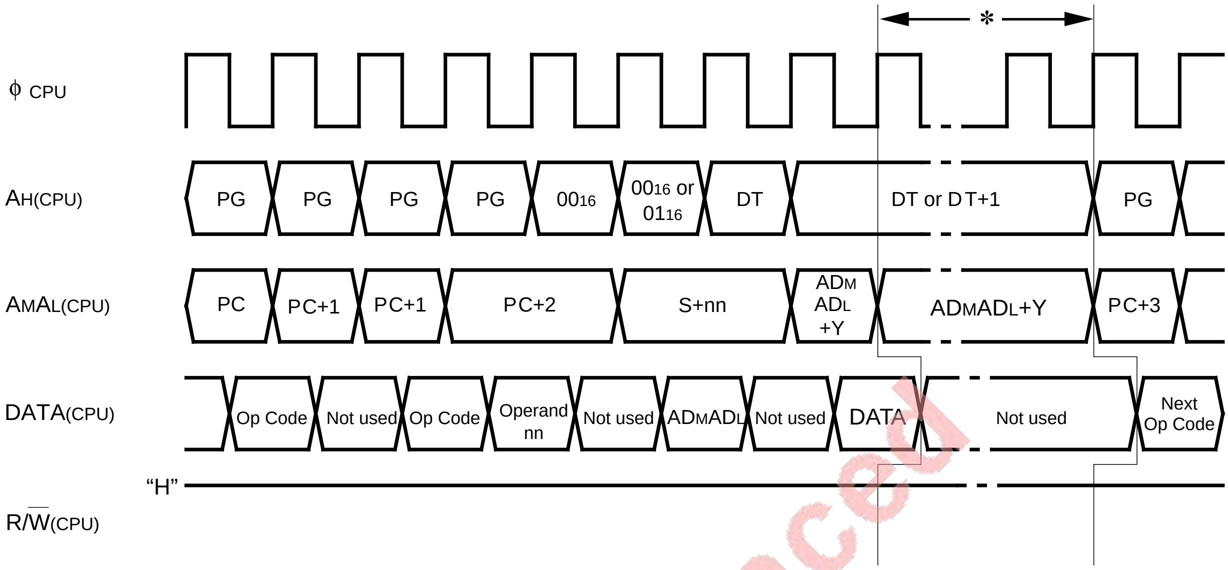
Timing :



Stack Pointer Relative Indirect Indexed Y

Instructions : DIV, DIVS, MPY, MPYS

Timing :



Note: The cycle number during "✻" is shown as the following table:

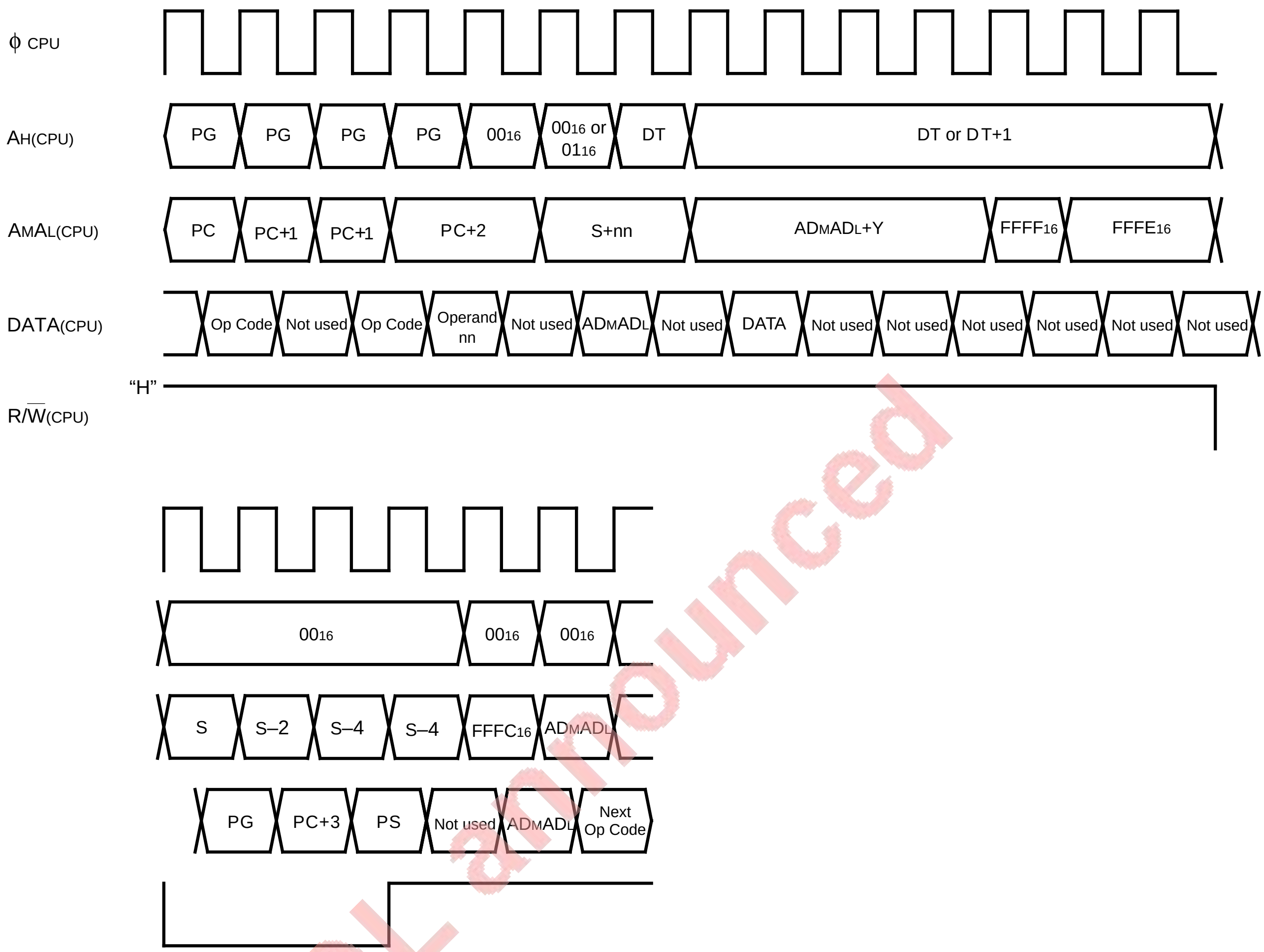
Instruction	Cycle number (φ CPU)	
	m = "0"	m = "1"
DIV	25	17
DIVS	27	19
MPY	8	4
MPYS	8	4

- The contents of AMAL(CPU) during "✻" with DIVS are undefined.
- When each sign of a multiplier and a multiplicand with MPYS is different, the contents of AMAL(CPU) at the last 3 of φ CPU during "✻" are undefined.

Stack Pointer Relative Indirect Indexed Y

Instructions : DIV , DIVS (case of 0 division)

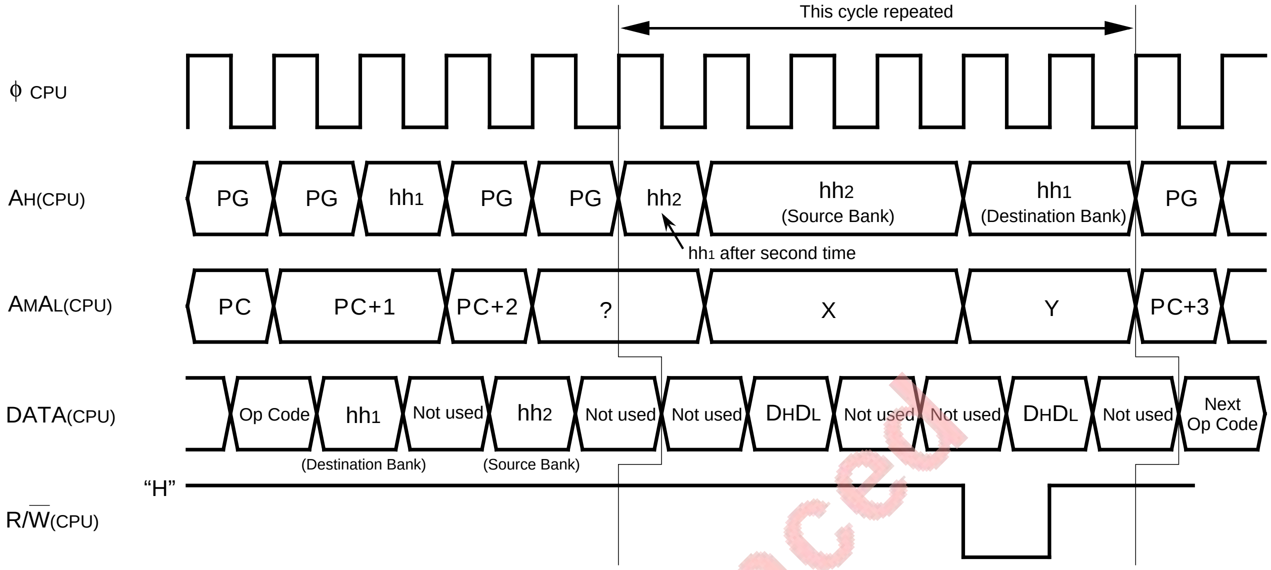
Timing :



Block Transfer

Instructions : MVN (transfer of even bytes)

Timing :

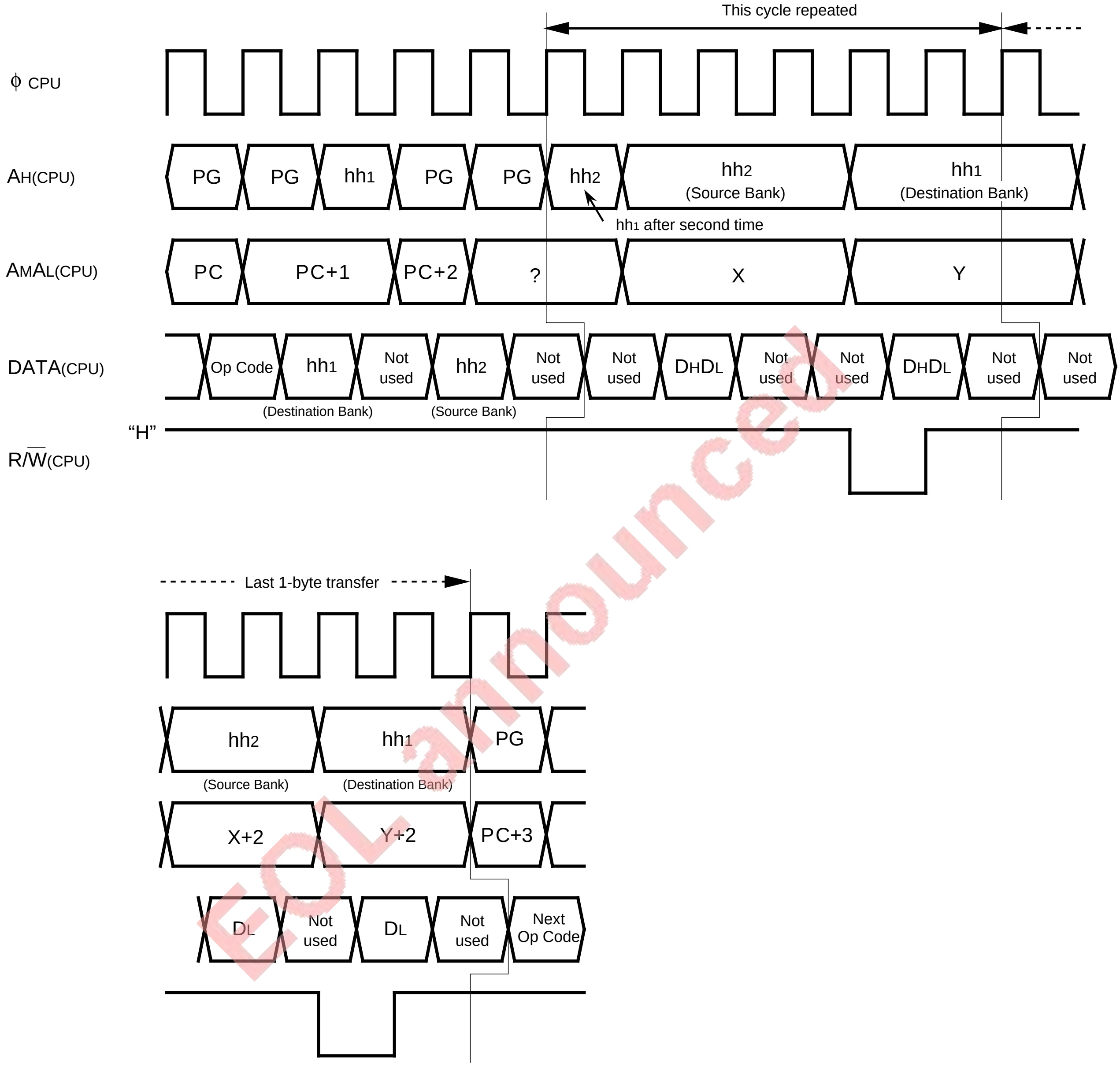


Note: This figure is an example when transferring 2-byte data.
When transferring 3 or more bytes data, the cycle "←→" is repeated at each 2-byte data.
The CPU instruction execution sequence is identical regardless of even address or odd address which the transferring start address is.

Block Transfer

Instructions : MVN (transfer of odd bytes)

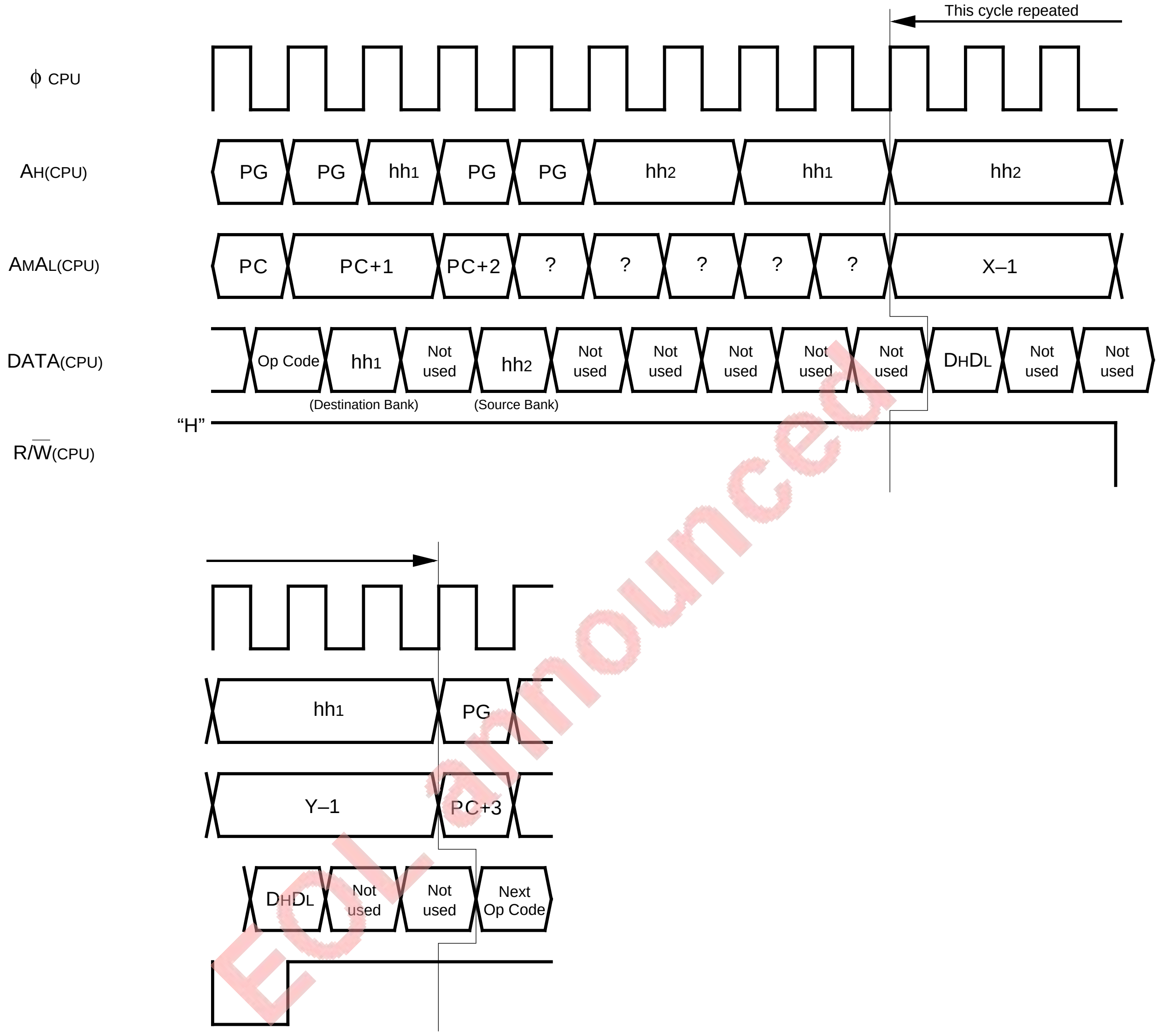
Timing :



Note: This figure is an example when transferring 3-byte data.
When transferring 4 or more bytes data, the cycle "◀━▶ " is repeated at each 2-byte data.
The CPU instruction execution sequence is identical regardless of even address or odd address which the transferring start address is.
The transfer of the last 1 byte is performed by reading 2 bytes and writing 1 byte.

Instructions : MVP (transfer of even bytes)

Timing :

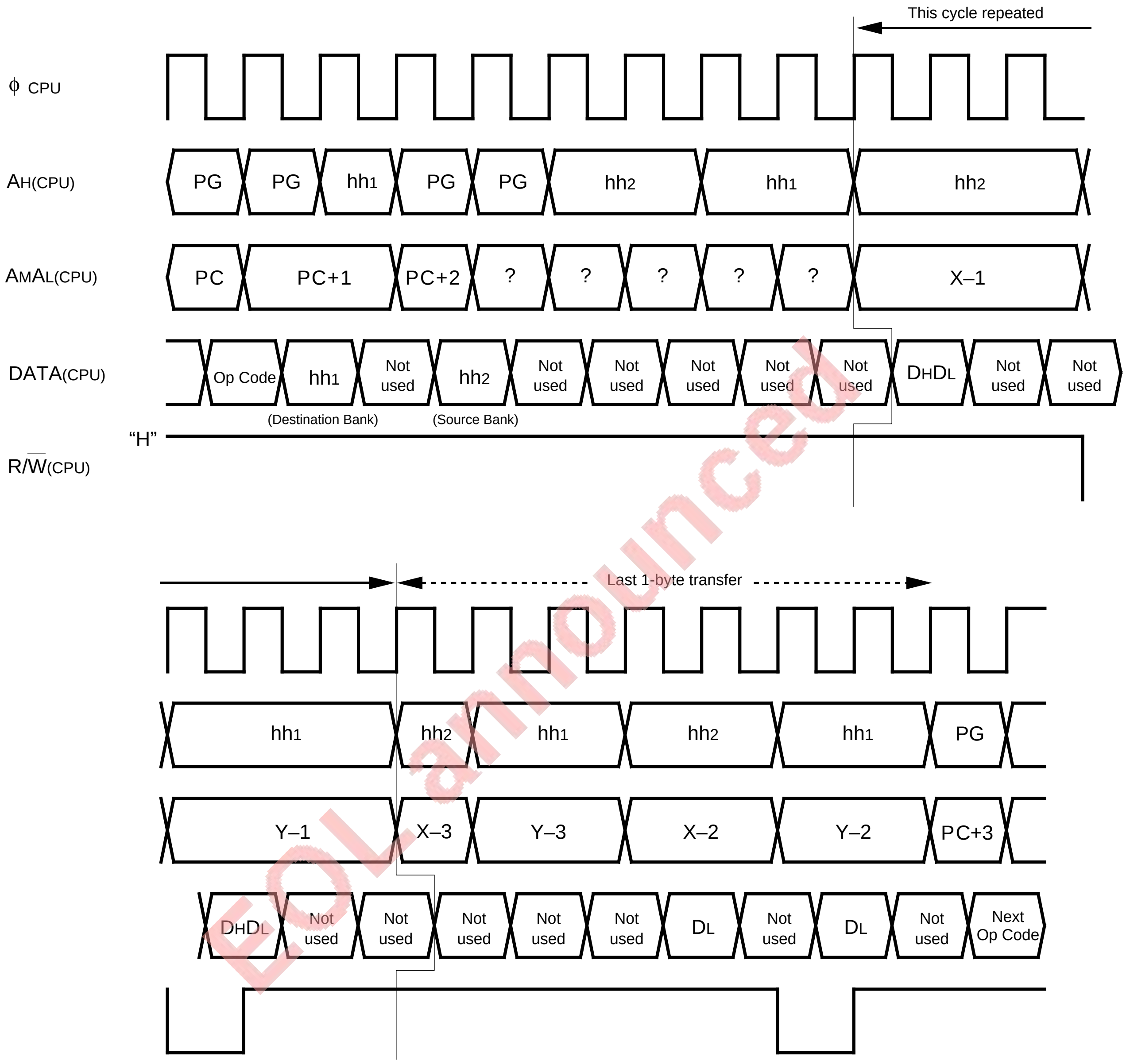


Note: This figure is an example when transferring 2-byte data.
When transferring 3 or more bytes data, the cycle "←→" is repeated at each 2-byte data.
The CPU instruction execution sequence is identical regardless of even address or odd address which the transferring start address is.

Block Transfer

Instructions : MVP (transfer of odd bytes)

Timing :

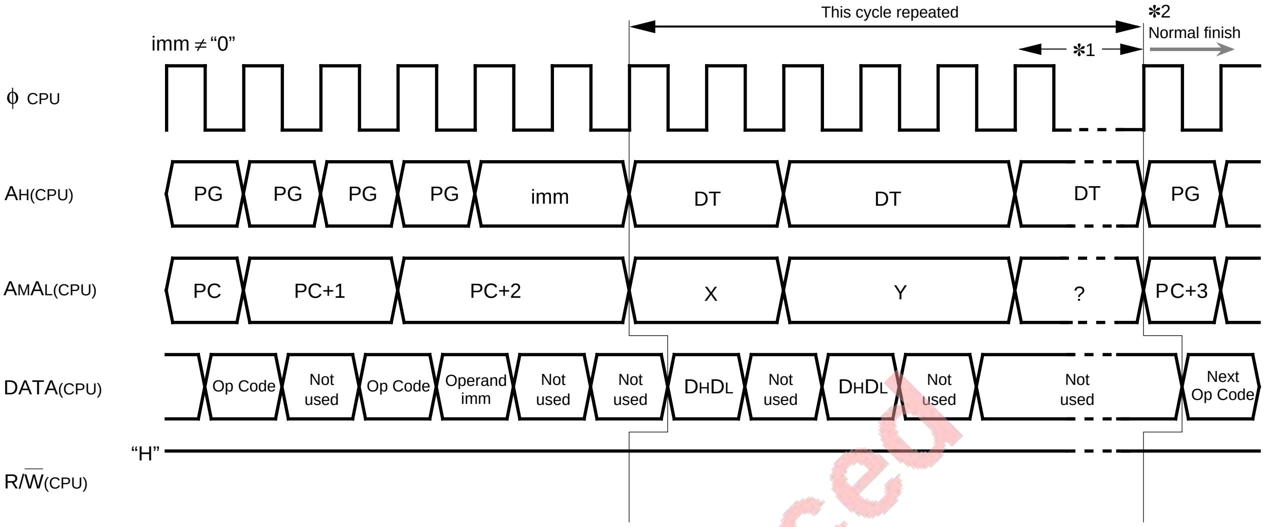


Note: This figure is an example when transferring 3-byte data.
When transferring 4 or more bytes data, the cycle "◄──►" is repeated at each 2-byte data.
The CPU instruction execution sequence is identical regardless of even address or odd address which the transferring start address is.

Multiplied Accumulation

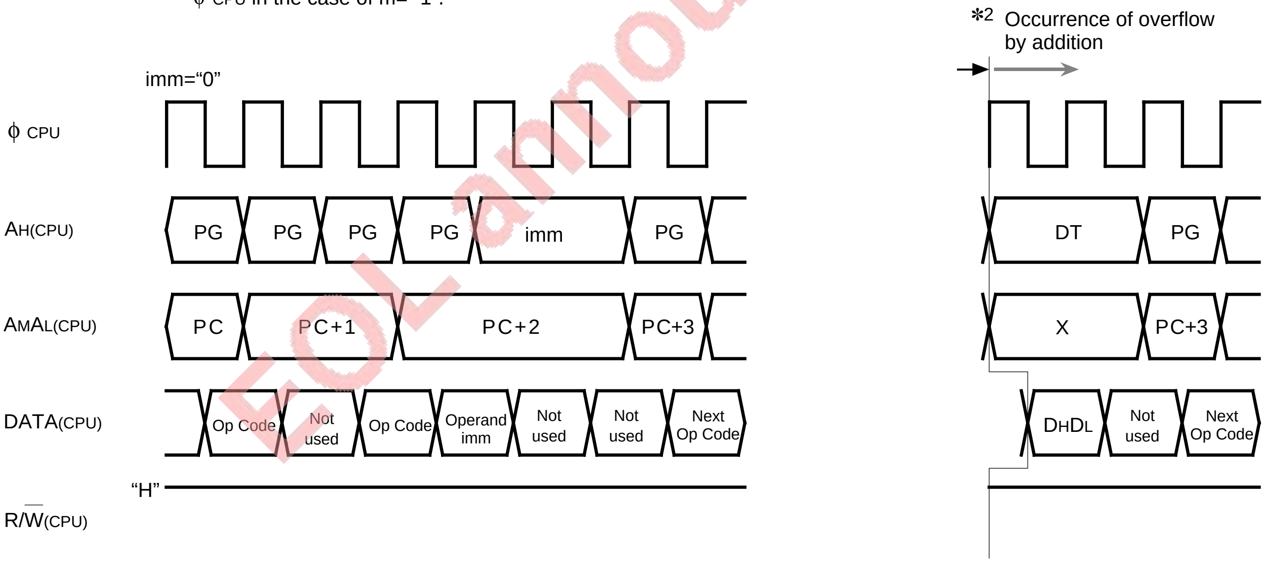
Instructions : RMPA

Timing :



Notes 1: This figure is an example when performing multiply and accumulate operations once. When doing them 1 or more times, the cycle "◀—▶" is repeated by the times.

2: The cycle number during "✻1" is 13 cycles of φ CPU in the case of m= "0", and 9 cycles of φ CPU in the case of m= "1".



APPENDIX

Appendix 1. 7751 series machine instructions

7751 SERIES MACHINE INSTRUCTIONS

Symbol	Function	Details	Addressing mode																																
			IMP			IMM			A			DIR			DIR,b			DIR,X			DIR,Y			(DIR)			(DIR,X)			(DIR),Y			L(DIR)		
			op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#
ADC (Note 1,2)	Acc, C←Acc+M+C	Adds the carry, the accumulator and the memory contents. The result is entered into the accumulator. When the D flag is “0”, binary additions is done, and when the D flag is “1”, decimal addition is done.				69	2 2	2				65	4 4	2				75	5 6	2				72	6 6	2	61	7 8	2	71	8 9	2	67	8 8	2
						42 69	4 4	3				42 65	6 6	3				42 75	7 8	3				42 72	8 8	3	42 61	9 10	3	42 71	10 11	3	42 67	10 10	3
AND (Note 1,2)	Acc←Acc∧M	Obtains the logical product of the contents of the accumulator and the contents of the memory. The result is entered into the accumulator.				29	2 2	2				25	4 4	2				35	5 6	2				32	6 6	2	21	7 8	2	31	8 9	2	27	8 8	2
						42 29	4 4	3				42 25	6 6	3				42 35	7 8	3				42 32	8 8	3	42 21	9 10	3	42 31	10 11	3	42 27	10 10	3
ASL (Note 1)	m=0 C ← b15···b0 ← 0 m=1 C ← b7···b0 ← 0	Shifts the accumulator or the memory contents one bit to the left. “0” is entered into bit 0 of the accumulator or the memory. The contents of bit 15 (bit 7 when the m flag is “1”) of the accumulator or memory before shift is entered into the C flag.							0A	2 2	1	06	7 8	2				16	7 8	2															
									42 0A	4 4	2																								
ASR (Note 1)	m=0 →b15···b0 →C m=1 →b7···b0 →C	Shifts the accumulator or the memory contents one bit to the right. The bit 0 of the accumulator or memory is entered into the C flag. The contents of bit 15 (bit 7 when the m flag is “1”) of the accumulator or memory before shift is entered into bit 15 (bit 7).							89 08	4 4	2	89 06	9 10	3				89 16	9 10	3															
									42 08	4 4	2																								
BBC (Note 4)	Mb=0?	Tests the specified bit of the memory. Branches when all the contents of the specified bit is “0”.																																	
BBS (Note 4)	Mb=1?	Tests the specified bit of the memory. Branches when all the contents of the specified bit is “1”.																																	
BCC	C=0?	Branches when the contents of the C flag is “0”.																																	
BCS	C=1?	Branches when the contents of the C flag is “1”.																																	
BEQ	Z=1?	Branches when the contents of the Z flag is “1”.																																	
BMI	N=1?	Branches when the contents of the N flag is “1”.																																	
BNE	Z=0?	Branches when the contents of the Z flag is “0”.																																	
BPL	N=0?	Branches when the contents of the N flag is “0”.																																	

Addressing mode																																																						Processor status register										
L(DIR),Y			ABS			ABS,b			ABS,X			ABS,Y			ABL			ABL,X			(ABS)			L(ABS)			(ABS,X)			STK			REL			DIR,b,R			ABS,b,R			SR			(SR),Y			BLK			Multiplied accumulation			10	9	8	7	6	5	4	3	2	1	0
op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	IPL			N	V	m	x	D	I	Z	C
77	10 11	2	6D	4 4	3				7D	6 7	3	79	6 7	3	6F	6 7	4	7F	7 9	4																						63	5 6	2	73	8 9	2							•	•	•	N	V	•	•	•	•	Z	C
42 77	12 13	3	42 6D	6 6	4				42 7D	8 9	4	42 79	8 9	4	42 6F	8 9	5	42 7F	9 11	5																						42 63	7 8	3	42 73	10 11	3																	
37	10 11	2	2D	4 4	3				3D	6 7	3	39	6 7	3	2F	6 7	4	3F	7 9	4																						23	5 6	2	33	8 9	2							•	•	•	N	•	•	•	•	•	Z	•
42 37	12 13	3	42 2D	6 6	4				42 3D	8 9	4	42 39	8 9	4	42 2F	8 9	5	42 3F	9 11	5																						42 23	7 8	3	42 33	10 11	3																	
			0E	7 8	3				1E	8 9	3																																											•	•	•	N	•	•	•	•	•	Z	C
			89 0E	9 10	4				89 1E	10 11	4																																											•	•	•	N	•	•	•	•	•	Z	C
																																		34	7 8	4	3C	8 9	5													•	•	•	•	•	•	•	•	•	•	•
																																		24	7 8	4	2C	8 9	5													•	•	•	•	•	•	•	•	•	•	•
																															90	4 4	2																					•	•	•	•	•	•	•	•	•	•	•
																															B0	4 4	2																					•	•	•	•	•	•	•	•	•	•	•
																															F0	4 4	2																					•	•	•	•	•	•	•	•	•	•	•
																															30	4 4	2																					•	•	•	•	•	•	•	•	•	•	•
																															D0	4 4	2																					•	•	•	•	•	•	•	•	•	•	•
																															10	4 4	2																					•	•	•	•	•	•	•	•	•	•	•

APPENDIX

Appendix 1. 7751 series machine instructions

Symbol	Function	Details	Addressing mode																																
			IMP			IMM			A			DIR			DIR,b			DIR,X			DIR,Y			(DIR)			(DIR,X)			(DIR),Y			L(DIR)		
			op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#
BRA (Note 3)	PC←PC±offset PG←PG+1 (carry occurred) PG←PG−1 (borrow occurred)	Jumps to the address indicated by the program counter plus the offset value.																																	
BRK	PC←PC+2 M(S)←PG S←S−1 M(S)←PCH S←S−1 M(S)←PCL S←S−1 M(S)←PSH S←S−1 M(S)←PSL S←S−1 I←1 PCL←ADL PCH←ADM PG←00_{16}	Executes software interruption.	00	15 15	2																														
BVC	V=0?	Branches when the contents of the V flag is "0".																																	
BVS	V=1?	Branches when the contents of the V flag is "1".																																	
CLB (Note 4)	Mb←0	Makes the contents of the specified bit in the memory "0".													14	8 9	3																		
CLC	C←0	Makes the contents of the C flag "0".	18	2 2	1																														
CLI	I←0	Makes the contents of the I flag "0".	58	2 2	1																														
CLM	m ←0	Makes the contents of the m flag "0".	D8	2 2	1																														
CLP	PSb←0	Specifies the bit position in the processor status register by the bit pattern of the second byte in the instruction, and sets "0" in that bit.				C2	4 4	2																											
CLV	V←0	Makes the contents of the V flag "0".	B8	2 2	1																														
CMP (Note 1,2)	Acc−M	Compares the contents of the accumulator with the contents of the memory.				C9	2 2	2				C5	4 4	2				D5	5 6	2				D2	6 6	2	C1	7 8	2	D1	8 9	2	C7	8 8	2
						42 C9	4 4	3				42 C5	6 6	3				42 D5	7 8	3				42 D2	8 8	3	42 C1	9 10	3	42 D1	10 11	3	42 C7	10 10	3
CPX (Note 11)	X−M	Compares the contents of the index register X with the contents of the memory.				E0	2 2	2				E4	4 4	2																					
CPY (Note 11)	Y−M	Compares the contents of the index register Y with the contents of the memory.				C0	2 2	2				C4	4 4	2																					
DEC (Note 11)	Acc←Acc−1 or M←M−1	Decrements the contents of the accumulator or memory by 1.							1A	2 2	1	C6	7 8	2				D6	7 8	2															
									42 1A	4 4	2																								
DEX	X←X−1	Decrements the contents of the index register X by 1.	CA	2 2	1																														

Addressing mode																																																						Processor status register										
L(DIR),Y			ABS			ABS,b			ABS,X			ABS,Y			ABL			ABL,X			(ABS)			L(ABS)			(ABS,X)			STK			REL			DIR,b,R			ABS,b,R			SR			(SR),Y			BLK			Multiplied accumulation			10	9	8	7	6	5	4	3	2	1	0
op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	IPL			N	V	m	x	D	I	Z	C
																																	80	3 3	2																			•	•	•	•	•	•	•	•	•	•	•
																																	82	3 3	3																													
																																																					•	•	•	•	•	•	•	•	•	•	•
																																	50	4 4	2																			•	•	•	•	•	•	•	•	•	•	•
																																	70	4 4	2																			•	•	•	•	•	•	•	•	•	•	•
						1C	9 11	4																																																•	•	•	•	•	•	•	•	•	•	•
																																																					•	•	•	•	•	•	•	•	•	•	0
																																																					•	•	•	•	•	•	•	•	0	•	•
																																																					•	•	•	•	•	0	•	•	•	•	•
																																																					•	•	•	Specified flag becomes “0”.							
																																																					•	•	•	•	0	•	•	•	•	•	•
D7	10 11	2	CD	4 4	3				DD	6 7	3	D9	6 7	3	CF	6 7	4	DF	7 9	4																						C3	5 6	2	D3	8 9	2							•	•	•	N	•	•	•	•	•	Z	C
42 D7	12 13	3	42 CD	6 6	4				42 DD	8 9	4	42 D9	8 9	4	42 CF	8 9	5	42 DF	9 11	5																						42 C3	7 8	3	42 D3	10 11	3																	
			EC	4 4	3																																																		•	•	•	N	•	•	•	•	•	Z	C
			CC	4 4	3																																																		•	•	•	N	•	•	•	•	•	Z	C
			CE	7 8	3				DE	8 9	3																																												•	•	•	N	•	•	•	•	•	Z	•
																																																					•	•	•	N	•	•	•	•	•	Z	•

APPENDIX

Appendix 1. 7751 series machine instructions

Symbol	Function	Details	Addressing mode																																
			IMP			IMM			A			DIR			DIR,b			DIR,X			DIR,Y			(DIR)			(DIR,X)			(DIR),Y			L(DIR)		
			op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#
DEY	Y←Y−1	Decrements the contents of the index register Y by 1.	88	2 2	1																														
DIV (Note 2,9,13)	A(quotient)←B, A+M B(remainder)	The numeral that places the contents of accumulator B to the higher order and the contents of accumulator A to the lower order is divided by the contents of the memory. The quotient is entered into accumulator A and the remainder into accumulator B.				89 29	21 21	3				89 25	23 23	3				89 35	24 25	3				89 32	25 25	3	89 21	26 27	3	89 31	27 28	3	89 27	27 27	3
DIVS (Note 2,9,14)	A(quotient)←B, A+M (with sign) B(remainder)	The numeral with sign that places the contents of accumulator B to the higher order and the contents of accumulator A to the lower order is divided by the contents of the memory. The quotient is entered into accumulator A and the remainder into accumulator B.				89 A9	23 23	3				89 A5	25 25	3				89 B5	26 27	3				89 B2	27 27	3	89 A1	28 29	3	89 B1	29 30	3	89 A7	29 29	3
EOR (Note 1,2)	Acc←Acc∀M	Logical exclusive sum is obtained of the contents of the accumulator and the contents of the memory. The result is placed into the accumulator.				49	2 2	2				45	4 4	2				55	5 6	2				52	6 6	2	41	7 8	2	51	8 9	2	47	8 8	2
						42 49	4 4	3				42 45	6 6	3				42 55	7 8	3				42 52	8 8	3	42 41	9 10	3	42 51	10 11	3	42 47	10 10	3
EXTS (Note 1)	Bit 7 of Acc=1 b15 b7 b0 11111111 1 Bit 7 of Acc=0 b15 b7 b0 00000000 0	The signed 8-bit data stored in the low-order byte of the accumulator is extended to a 16-bit data.							89 8B	4 4	2																								
									42 8B	4 4	2																								
EXTZ (Note 1)	Acc b15 b8 b7 b0 00000000	The 8-bit data stored in the low-order byte of the accumulator is extended to a 16-bit data. Bits 8 to 15 of the accumulator are set to "0".							89 AB	4 4	2																								
									42 AB	4 4	2																								
INC (Note 1)	Acc←Acc+1 or M←M+1	Increments the contents of the accumulator or memory by 1.							3A	2 2	1	E6	7 8	2				F6	7 8	2															
									42 3A	4 4	2																								
INX	X←X+1	Increments the contents of the index register X by 1.	E8	2 2	1																														
INY	Y←Y+1	Increments the contents of the index register Y by 1.	C8	2 2	1																														
JMP	ABS PCL←ADL PCH←ADM ABL PCL←ADL PCH←ADM PG←ADH (ABS) PCL←(ADM, ADL) PCH←(ADM, ADL+1) L(ABS) PCL←(ADM, ADL) PCH←(ADM, ADL+1) PG←(ADM, ADL+2) (ABS, X) PCL←(ADM, ADL+X) PCH←(ADM, ADL+X+1)	Places a new address into the program counter and jumps to that new address.																																	

Addressing mode																																																						Processor status register										
L(DIR),Y			ABS			ABS,b			ABS,X			ABS,Y			ABL			ABL,X			(ABS)			L(ABS)			(ABS,X)			STK			REL			DIR,b,R			ABS,b,R			SR			(SR),Y			BLK			Multiplied accumulation			10	9	8	7	6	5	4	3	2	1	0
op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	IPL			N	V	m	x	D	I	Z	C
																																																							•	•	•	N	•	•	•	•	•	Z	•
89 37	29 30	3	89 2D	23 23	4				89 3D	25 26	4	89 39	25 26	4	89 2F	25 26	5	89 3F	26 28	5																						89 23	24 25	3	89 33	27 28	3							•	•	•	N	V	•	•	•	•	Z	C
89 B7	31 32	3	89 AD	25 25	4				89 BD	27 28	4	89 B9	27 28	4	89 AF	27 28	5	89 BF	28 30	5																						89 A3	26 27	3	89 B3	29 30	5							•	•	•	N	V	•	•	•	•	Z	C
57 •	10 11	2	4D	4 4	3				5D	6 7	3	59	6 7	3	4F	6 7	4	5F	7 9	4																						43	5 6	2	53	8 9	2							•	•	•	N	•	•	•	•	•	Z	•
42 57	12 13	3	42 4D	6 6	4				42 5D	8 9	4	42 59	8 9	4	42 4F	8 9	5	42 5F	9 11	5																						42 43	7 8	3	42 53	10 11	3																	
																																																						•	•	•	N	•	•	•	•	•	Z	•
																																																						•	•	•	0	•	•	•	•	•	Z	•
			EE	7 8	3				FE	8 9	3																																											•	•	•	N	•	•	•	•	•	Z	•
																																																						•	•	•	N	•	•	•	•	•	Z	•
																																																						•	•	•	N	•	•	•	•	•	Z	•
			4C	2 2	3										5C	4 5	4				6C	4 4	3	DC	6 6	3	7C	6 7	3																									•	•	•	•	•	•	•	•	•	•	•

APPENDIX

Appendix 1. 7751 series machine instructions

Symbol	Function	Details	Addressing mode																																
			IMP			IMM			A			DIR			DIR,b			DIR,X			DIR,Y			(DIR)			(DIR,X)			(DIR),Y			L(DIR)		
			op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#
JSR	ABS M(S)←PCH S←S−1 M(S)←PCL S←S−1 PCL←ADL PCH←ADM ABL M(S)←PG S←S−1 M(S)←PCH S←S−1 M(S)←PCL S←S−1 PCL←ADL PCH←ADM PG←ADH (ABS, X) M(S)←PCH S←S−1 M(S)←PCL S←S−1 PCL←(ADM, ADL+X) PCH←(ADM, ADL+X+1)	Saves the contents of the program counter (also the contents of the program bank register for ABL) into the stack, and jumps to the new address.																																
LDA (Note 1,2)	ACC←M	Enters the contents of the memory into the accumulator.				A9	2 2	2				A5	4 4	2				B5	5 6	2				B2	6 6	2	A1	7 8	2	B1	8 9	2	A7	8 8	2
						42 A9	4 4	3				42 A5	6 6	3				42 B5	7 8	3				42 B2	8 8	3	42 A1	9 10	3	42 B1	10 11	3	42 A7	10 10	3
LDM (Note 4)	M←IMM	Enters the immediate value into the memory.										64	4 5	3				74	5 6	3															
LDT	DT←IMM	Enters the immediate value into the data bank register.				89 C2	5 5	3																											
LDX (Note 11)	X←M	Enters the contents of the memory into index register X.				A2	2 2	2				A5	4 4	2							B6	5 6	2												
LDY (Note 11)	Y←M	Enters the contents of the memory into index register Y.				A0	2 2	2				A4	4 4	2				B4	5 6	2															
LSR (Note 1)	m=0 0 → b15 ··· b0 → C m=1 0 → b7 ··· b0 → C	Shifts the contents of the accumulator or the contents of the memory one bit to the right. The bit 0 of the accumulator or the memory is entered into the C flag. "0" is entered into bit 15 (bit 7 when the m flag is "1".)							4A	2 2	1	46	7 8	2				56	7 8	2															
									42 4A	4 4	2																								
MPY (Note 2,10)	B, A←A×M	Multiplies the contents of accumulator A and the contents of the memory. The higher order of the result of operation are entered into accumulator B, and the lower order into accumulator A.				89 09	8 8	3				89 05	10 10	3				89 15	11 12	3				89 12	12 12	3	89 01	13 14	3	89 11	14 15	3	89 07	14 14	3
MPYS (Note 2,10)	B, A←A×M (with sign)	The content of the accumulator A is multiplied by the content of memory as signed data. The result is a 32-bit data which is placed in the accumulators B (upper 16 bits of the result) and A (lower 16 bits of the result).				89 89	8 8	3				89 85	10 10	3				89 95	11 12	3				89 92	12 12	3	89 81	13 14	3	89 91	14 15	3	89 87	14 14	3
MVN (Note 7)	M(Y+k)←M(X+k) k=0 to i−1	Transmits the data block. The transmission is done from the lower order address of the block.																																	

Addressing mode																													
L(DIR),Y			ABS			ABS,b			ABS,X			ABS,Y			ABL			ABL,X			(ABS)			L(ABS)			(ABS,X)		
op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#
			20	6 7	3										22	8 9	4										FC	8 9	3
B7	10 11	2	AD	4 4	3				BD	6 7	3	B9	6 7	3	AF	6 7	4	BF	7 9	4									
42 B7	12 13	3	42 AD	6 6	4				42 BD	8 9	4	42 B9	8 9	4	42 AF	8 9	5	42 BF	9 11	5									
			9C	5 6	4				9E	6 8	4																		
			AE	4 4	3							BE	6 7	3															
			AC	4 4	3				BC	6 7	3																		
			4E	7 8	3				5E	8 9	3																		
89 17	16 17	3	89 0D	10 10	4				89 1D	12 13	4	89 19	12 13	4	89 0F	12 13	5	89 1F	13 15	5									
89 97	16 17	3	89 8D	10 10	4				89 9D	12 13	4	89 99	12 13	4	89 8F	12 13	5	89 9F	13 15	5									

Addressing mode																								Processor status register										
STK			REL			DIR,b,R			ABS,b,R			SR			(SR),Y			BLK			Multiplied accumulation			10	9	8	7	6	5	4	3	2	1	0
op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	IPL			N	V	m	x	D	I	Z	C
																								•	•	•	•	•	•	•	•	•	•	•
												A3	5 6	2	B3	8 9	2							•	•	•	N	•	•	•	•	•	Z	•
												42 A3	7 8	3	42 B3	10 11	3																	
																								•	•	•	•	•	•	•	•	•	•	•
																								•	•	•	•	•	•	•	•	•	•	•
																								•	•	•	N	•	•	•	•	•	Z	•
																								•	•	•	N	•	•	•	•	•	Z	•
																								•	•	•	0	•	•	•	•	•	Z	C
												89 03	11 12	3	89 13	14 15	3							•	•	•	N	•	•	•	•	•	Z	0
												89 83	11 12	3	89 93	14 15	3							•	•	•	N	•	•	•	•	•	Z	0
																		54	5 + $\frac{i}{2}$ × 7	3				•	•	•	•	•	•	•	•	•	•	•
																			5 + $\frac{i}{2}$ × 7															

APPENDIX

Appendix 1. 7751 series machine instructions

Symbol	Function	Details	Addressing mode																																
			IMP			IMM			A			DIR			DIR,b			DIR,X			DIR,Y			(DIR)			(DIR,X)			(DIR),Y			L(DIR)		
			op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#
MVP (Note 8)	M(Y–K)←M(X–k) k=0~i–1	Transmits the data block. Transmission is done form the higher order address of the data block.																																	
NOP	PC←PC+1	Advances the program counter, but performs nothing else.	EA	2 2	1																														
ORA (Note 1,2)	Acc←Acc∨M	Logical sum per bit of the contents of the accumulator and the contents of the memory is obtained. The result is entered into the accumulator.				09	2 2	2				05	4 4	2				15	5 6	2				12	6 6	2	01	7 8	2	11	8 9	2	07	8 8	2
						42 09	4 4	3				42 05	6 6	3				42 15	7 8	3				42 12	8 8	3	42 01	9 10	3	42 11	10 11	3	42 07	10 10	3
PEA	M(S)←IMM$_2$ S←S–1 M(S)←IMM$_1$ S←S–1	The 3rd and the 2nd bytes of the instruction are saved into the stack, in this order.																																	
PEI	M(S)←M((DPR)+IMM+1) S←S–1 M(S)←M((DPR)+IMM) S←S–1	Specifies 2 sequential bytes in the direct page in the 2nd byte of the instruction, and saves the contents into the stack.																																	
PER	EAR←PC+IMM$_2$, IMM$_1$ M(S)←EAR$_H$ S←S–1 M(S)←EAR$_L$ S←S–1	Regards the 2nd and 3rd bytes of the instruction as 16-bit numerals, adds them to the program counter, and saves the result into the stack.																																	
PHA	m=0 M(S)←A$_H$ S←S–1 M(S)←A$_L$ S←S–1 m=1 M(S)←A$_L$ S←S–1	Saves the contents of accumulator A into the stack.																																	
PHB	m=0 M(S)←B$_H$ S←S–1 M(S)←B$_L$ S←S–1 m=1 M(S)←B$_L$ S←S–1	Saves the contents of accumulator B into the stack.																																	
PHD	M(S)←DPR$_H$ S←S–1 M(S)←DPR$_L$ S←S–1	Saves the contents of the direct page register into the stack.																																	
PHG	M(S)←PG S←S–1	Saves the contents of the program bank register into the stack.																																	
PHP	M(S)←PS$_H$ S←S–1 M(S)←PS$_L$ S←S–1	Saves the contents of the program status register into the stack.																																	
PHT	M(S)←DT S←S–1	Saves the contents of the data bank register into the stack.																																	

Addressing mode																																																						Processor status register										
L(DIR),Y			ABS			ABS,b			ABS,X			ABS,Y			ABL			ABL,X			(ABS)			L(ABS)			(ABS,X)			STK			REL			DIR,b,R			ABS,b,R			SR			(SR),Y			BLK			Multiplied accumulation			10	9	8	7	6	5	4	3	2	1	0
op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	IPL			N	V	m	x	D	I	Z	C
																																																44 $\frac{i}{2}$ $\frac{i}{2}$	9 + × 9 + ×	3 7 7				•	•	•	•	•	•	•	•	•	•	•
																																																						•	•	•	•	•	•	•	•	•	•	•
17	10 11	2	0D	4 4	3				1D	6 7	3	19	6 7	3	0F	6 7	4	1F	7 9	4																						03	5 6	2	13	8 9	2							•	•	•	N	•	•	•	•	•	Z	•
42 17	12 13	3	42 0D	6 6	4				42 1D	8 9	4	42 19	8 9	4	42 0F	8 9	5	42 1F	9 11	5																						42 03	7 8	3	42 13	10 11	3																	
																														F4	5 7	3																						•	•	•	•	•	•	•	•	•	•	•
																														D4	6 7	2																						•	•	•	•	•	•	•	•	•	•	•
																														62	5 6	3																						•	•	•	•	•	•	•	•	•	•	•
																														48	4 4	1																						•	•	•	•	•	•	•	•	•	•	•
																														42 48	6 6	2																						•	•	•	•	•	•	•	•	•	•	•
																														0B	4 4	1																						•	•	•	•	•	•	•	•	•	•	•
																														4B	3 3	1																						•	•	•	•	•	•	•	•	•	•	•
																														08	4 4	1																						•	•	•	•	•	•	•	•	•	•	•
																														8B	3 3	1																						•	•	•	•	•	•	•	•	•	•	•

APPENDIX

Appendix 1. 7751 series machine instructions

Symbol	Function	Details	Addressing mode																																
			IMP			IMM			A			DIR			DIR,b			DIR,X			DIR,Y			(DIR)			(DIR,X)			(DIR),Y			L(DIR)		
			op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#
PHX	x=0 M(S)←X_H S←S−1 M(S)←X_L S←S−1 x=1 M(S)←X_L S←S−1	Saves the contents of the index register X into the stack.																																	
PHY	x=0 M(S)←Y_H S←S−1 M(S)←Y_L S←S−1 x=1 M(S)←Y_L S←S−1	Saves the contents of the index register Y into the stack.																																	
PLA	m=0 S←S+1 A_L←M(S) S←S+1 A_H←M(S) m=1 S←S+1 A_L←M(S)	Restores the contents of the stack on the accumulator A.																																	
PLB	m=0 S←S+1 B_L←M(S) S←S+1 B_H←M(S) m=1 S←S+1 B_L←M(S)	Restores the contents of the stack on the accumulator B.																																	
PLD	S←S+1 DPR_L←M(S) S←S+1 DPR_H←M(S)	Restores the contents of the stack on the direct page register.																																	
PLP	S←S+1 PS_L←M(S) S←S+1 PS_H←M(S)	Restores the contents of the stack on the processor status register.																																	
PLT	S←S+1 DT←M(S)	Restores the contents of the stack on the data bank register.																																	
PLX	x=0 S←S+1 X_L←M(S) S←S+1 X_H←M(S) x=1 S←S+1 X_L←M(S)	Restores the contents of the stack on the index register X.																																	
PLY	x=0 S←S+1 Y_L←M(S) S←S+1 Y_H←M(S) x=1 S←S+1 Y_L←M(S)	Restores the contents of the stack on the index register Y.																																	

L(DIR),Y			ABS			ABS,b			ABS,X			ABS,Y			ABL			ABL,X			(ABS)			L(ABS)			(ABS,X)			STK			REL			DIR,b,R			ABS,b,R			SR			(SR),Y			BLK			Multiplied accumulation			Processor status register										
																																																						10	9	8	7	6	5	4	3	2	1	0
op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	IPL			N	V	m	x	D	I	Z	C
																														DA	4 4	1																						•	•	•	•	•	•	•	•	•	•	•
																														5A	4 4	1																						•	•	•	•	•	•	•	•	•	•	•
																														68	5 5	1																						•	•	•	N	•	•	•	•	•	Z	•
																														42 68	7 7	2																						•	•	•	N	•	•	•	•	•	Z	•
																														2B	5 5	1																						•	•	•	•	•	•	•	•	•	•	•
																														2B	6 6	1																						Value saved in stack.										
																														AB	6 6	1																						•	•	•	N	•	•	•	•	•	Z	•
																														FA	5 5	1																						•	•	•	N	•	•	•	•	•	Z	•
																														7A	5 5	1																						•	•	•	N	•	•	•	•	•	Z	•

APPENDIX

Appendix 1. 7751 series machine instructions

Symbol	Function	Details	Addressing mode																																						
			IMP			IMM			A			DIR			DIR,b			DIR,X			DIR,Y			(DIR)			(DIR,X)			(DIR),Y			L(DIR)		
			op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#
PSH (Note 5)	M(S)←A, B, X···	Saves the registers among accumulator, index register, direct page register, data bank register, program bank register, or processor status register, specified by the bit pattern of the second byte of the instruction into the stack.																																						
PUL (Note 6)	A, B, X...←M(S)	Restores the contents of the stack to the registers among accumulator, index register, direct page register, data bank register, or processor status register, specified by the bit pattern of the second byte of the instruction.																																						
RLA (Note 12)	m=0 i bit rotate left ← b15 ··· b0 ← m=1 i bit rotate left ← b7 ··· b0 ←	Rotates the contents of the accumulator A, i bits to the left.				89 49	6+i 6+i	3																														
RMPA (Note 15)	m=0 Repeat B, A←B, A+M(DT, X)× M(DT, Y)(with sign) X←X+2 Y←Y+2 i←i−1 Until i=0 m=1 Repeat B_L,A_L←B_L,A_L+M(DT,X)× M(DT, Y)(with sign) X←X+1 Y←Y+1 i←i−1 Until i=0	Performs signed multiplication of the data in the memory specified by index register X and data bank register, and the data in the memory specified by index register Y and data bank register. The multiplication result is added as binary addition to the data of which high-order is the contents of accumulator B, and of which low-order is the contents of accumulator A. The high-order result is stored in accumulator B, and the low-order result is stored in accumulator A again. After the addition, when the data length flag (m) is "0", each of the contents of index register X and index register Y is incremented by 2. Additionally, the number of multiplication and addition is decremented by 1. When the data length flag (m) is "1", each of the contents of index register X and index register Y is incremented by 1. Additionally, the number of multiplication and addition is decremented by 1. The above multiplication and addition are repeated until the number of multiplication and addition is "0".																																						
ROL (Note 1)	m=0 ← b15 ··· b0 ← C ←	Links the accumulator or the memory to C flag, and rotates result to the left by 1 bit.							2A	2 2	1	26	7 8	2				36	7 8	2															
	m=1 ← b7 ··· b0 ← C ←								42 2A	4 4	2																								
ROR (Note 2)	m=0 → C → b15 ··· b0 →	Links the accumulator or the memory to C flag, and rotates result to the right by 1 bit.							6A	2 2	1	66	7 8	2				76	7 8	2															
	m=1 → C → b7 ··· b0 →								42 6A	4 4	2																								

Addressing mode																																																						Processor status register										
L(DIR),Y			ABS			ABS,b			ABS,X			ABS,Y			ABL			ABL,X			(ABS)			L(ABS)			(ABS,X)			STK			REL			DIR,b,R			ABS,b,R			SR			(SR),Y			BLK			Multiplied accumulation			10	9	8	7	6	5	4	3	2	1	0
op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	IPL			N	V	m	x	D	I	Z	C
																														EB	$11 + 2i_1 + i_2$ $11 + 2i_1 + i_2$	2																						•	•	•	•	•	•	•	•	•	•	•
																														FB	$12 + 3i_1 + 4i_2$ $12 + 3i_1 + 4i_2$	2																						If restored the contents of PS, it becomes its value. And the other case is no change.										
																																																				•	•	•	•	•	•	•	•	•	•	•
																																																		89 E2	$6 + i \times 16$ $6 + i \times 16$	3	•	•	•	N	V	•	•	•	•	Z	C
			2E	7 8	3				3E	8 9	3																																													•	•	•	N	•	•	•	•	•	Z	C
			6E	7 8	3				7E	8 9	3																																													•	•	•	N	•	•	•	•	•	Z	C

Appendix 1. 7751 series machine instructions

Symbol	Function	Details	Addressing mode																															
			IMP			IMM			A			DIR			DIR,b			DIR,X			DIR,Y			(DIR)			(DIR,X)			(DIR),Y			L(DIR)		
			op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#
RTI	S←S+1 PSL←M(S) S←S+1 PSH←M(S) S←S+1 PCL←M(S) S←S+1 PCH←M(S) S←S+1 PG←M(S)	Returns from the interruption routine.	40	9 9	1																														
RTL	S←S+1 PCL←M(S) S←S+1 PCH←M(S) S←S+1 PG←M(S)	Returns from the subroutine. The contents of the program bank register are also restored.	6B	7 7	1																														
RTS	S←S+1 PCL←M(S) S←S+1 PCH←M(S)	Returns from the subroutine. The contents of the program bank register are not restored.	60	5 5	1																														
SBC (Note 1, 2)	Acc, C←Acc–M–$\overline{C}$	Subtracts the contents of the memory and the borrow from the contents the accumulator.				E9	2 2	2				E5	4 4	2				F5	5 6	2				F2	6 6	2	E1	7 8	2	F1	8 9	2	E7	8 8	2
						42 E9	4 4	3				42 E5	6 6	3				42 F5	7 8	3				42 F2	8 8	3	42 E1	9 10	3	42 F1	10 11	3	42 E7	10 10	3
SEB (Note 4)	Mb ←1	Makes the contents of the specified bit in the memory "1".													04	8 9	3																		
SEC	C←1	Makes the contents of the C flag "1".	38	2 2	1																														
SEI	I←1	Makes the contents of the I flag "1".	78	2 2	1																														
SEM	m←1	Makes the contents of the m flag "1".	F8	2 2	1																														
SEP	PSb←1	Set the specified bit of the processor status register's lower byte (PSL) to "1".				E2	3 3	2																											
STA (Note 1)	M←Acc	Stores the contents of the accumulator into the memory.										85	4 5	2				95	5 5	2				92	7 8	2	81	7 8	2	91	7 8	2	87	9 10	2
												42 85	6 7	3				42 95	7 7	3				42 92	9 10	3	42 81	9 10	3	42 91	9 10	3	42 87	11 12	3
STP		Stops the oscillation of the oscillator.	DB	– –	1																														
STX	M←X	Stores the contents of the index register X into the memory.										86	4 5	2							96	5 5	2												
STY	M←Y	Stores the contents of the index register Y into the memory.										84	4 5	2				94	5 5	2															
TAD	DPR←A	Transmits the contents of the accumulator A to the direct page register.	5B	2 2	1																														
TAS	S←A	Transmits the contents of the accumulator A to the stack pointer.	1B	2 2	1																														

Addressing mode																																																						Processor status register										
L(DIR),Y			ABS			ABS,b			ABS,X			ABS,Y			ABL			ABL,X			(ABS)			L(ABS)			(ABS,X)			STK			REL			DIR,b,R			ABS,b,R			SR			(SR),Y			BLK			Multiplied accumulation			10	9	8	7	6	5	4	3	2	1	0
op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	IPL			N	V	m	x	D	I	Z	C
																																																						Value saved in stack.										
																																																						•	•	•	•	•	•	•	•	•	•	•
																																																						•	•	•	•	•	•	•	•	•	•	•
F7	10 11	2	ED	4 4	3				FD	6 7	3	F9	6 7	3	EF	6 7	4	FF	7 9	4																						E3	5 6	2	F3	8 9	2							•	•	•	N	V	•	•	•	•	Z	C
42 F7	12 13	3	42 ED	6 6	4				42 FD	8 9	4	42 F9	8 9	4	42 EF	8 9	5	42 FF	9 11	5																						42 E3	7 8	3	42 F3	10 11	3																	
						0C	9 11	4																																																•	•	•	•	•	•	•	•	•	•	•
																																																						•	•	•	•	•	•	•	•	•	•	1
																																																						•	•	•	•	•	•	•	•	1	•	•
																																																						•	•	•	•	•	1	•	•	•	•	•
																																																						•	•	•	Specified flag becomes “1”.							
97	9 10	2	8D	5 6	3				9D	5 5	3	99	5 5	3	8F	6 8	4	9F	7 9	4																						83	5 5	2	93	8 9	2							•	•	•	•	•	•	•	•	•	•	•
42 97	11 12	3	42 8D	7 8	4				42 9D	7 8	4	42 99	7 8	4	42 8F	8 10	5	42 9F	9 10	5																						42 83	7 7	3	42 93	10 11	3																	
																																																						•	•	•	•	•	•	•	•	•	•	•
			8E	5 6	3																																																	•	•	•	•	•	•	•	•	•	•	•
			8C	5 6	3																																																	•	•	•	•	•	•	•	•	•	•	•
																																																						•	•	•	•	•	•	•	•	•	•	•
																																																							•	•	•	•	•	•	•	•	•	•	•

Appendix 1. 7751 series machine instructions

Symbol	Function	Details	Addressing mode																																
			IMP			IMM			A			DIR			DIR,b			DIR,X			DIR,Y			(DIR)			(DIR,X)			(DIR),Y			L(DIR)		
			op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#
TAX	X←A	Transmits the contents of the accumulator A to the index register X.	AA	2 2	1																														
TAY	Y←A	Transmits the contents of the accumulator A to the index register Y.	A8	2 2	1																														
TBD	DPR←B	Transmits the contents of the accumulator B to the direct page register.	42 5B	4 4	2																														
TBS	S←B	Transmits the contents of the accumulator B to the stack pointer.	42 1B	4 4	2																														
TBX	X←B	Transmits the contents of the accumulator B to the index register X.	42 AA	4 4	2																														
TBY	Y←B	Transmits the contents of the accumulator B to the index register Y.	42 A8	4 4	2																														
TDA	A←DPR	Transmits the contents of the direct page register to the accumulator A.	7B	2 2	1																														
TDB	B←DPR	Transmits the contents of the direct page register to the accumulator B.	42 7B	4 4	2																														
TSA	A←S	Transmits the contents of the stack pointer to the accumulator A.	3B	2 2	1																														
TSB	B←S	Transmits the contents of the stack pointer to the accumulator B.	42 3B	4 4	2																														
TSX	X←S	Transmits the contents of the stack pointer to the index register X.	BA	2 2	1																														
TXA	A←X	Transmits the contents of the index register X to the accumulator A.	8A	2 2	1																														
TXB	B←X	Transmits the contents of the index register X to the accumulator B.	42 8A	4 4	2																														
TXS	S←X	Transmits the contents of the index register X to the stack pointer.	9A	2 2	1																														
TXY	Y←X	Transmits the contents of the index register X to the index register Y.	9B	2 2	1																														
TYA	A←Y	Transmits the contents of the index register Y to the accumulator A.	98	2 2	1																														
TYB	B←Y	Transmits the contents of the index register Y to the accumulator B.	42 98	4 4	2																														
TYX	X←Y	Transmits the contents of the index register Y to the index register X.	BB	2 2	1																														
WIT		Stops the ϕCPU, ϕBIU.	CB	– –	1																														
XAB	A⇔B	Exchanges the contents of the accumulator A and the contents of the accumulator B.	89 28	5 5	2																														

Addressing mode																																																						Processor status register										
L(DIR),Y			ABS			ABS,b			ABS,X			ABS,Y			ABL			ABL,X			(ABS)			L(ABS)			(ABS,X)			STK			REL			DIR,b,R			ABS,b,R			SR			(SR),Y			BLK			Multiplied accumulation			10	9	8	7	6	5	4	3	2	1	0
op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	op	n	#	IPL			N	V	m	x	D	I	Z	C
																																																				•	•	•	N	•	•	•	•	•	Z	•
																																																				•	•	•	N	•	•	•	•	•	Z	•
																																																				•	•	•	•	•	•	•	•	•	•	•
																																																					•	•	•	•	•	•	•	•	•	•	•
																																																					•	•	•	N	•	•	•	•	•	Z	•
																																																					•	•	•	N	•	•	•	•	•	Z	•
																																																					•	•	•	N	•	•	•	•	•	Z	•
																																																					•	•	•	N	•	•	•	•	•	Z	•
																																																					•	•	•	N	•	•	•	•	•	Z	•
																																																					•	•	•	N	•	•	•	•	•	Z	•
																																																					•	•	•	N	•	•	•	•	•	Z	•
																																																					•	•	•	N	•	•	•	•	•	Z	•
																																																					•	•	•	N	•	•	•	•	•	Z	•
																																																					•	•	•	•	•	•	•	•	•	•	•
																																																					•	•	•	N	•	•	•	•	•	Z	•
																																																					•	•	•	N	•	•	•	•	•	Z	•
																																																					•	•	•	N	•	•	•	•	•	Z	•
																																																					•	•	•	N	•	•	•	•	•	Z	•
																																																					•	•	•	•	•	•	•	•	•	•	•
																																																					•	•	•	N	•	•	•	•	•	Z	•

Notes for machine instructions table

A number of cycles on the upper row is the number when fetching instructions at 2 ϕ access in low-speed running under the condition of $f(X_{IN}) \leqq 25$ MHz. A number of cycles on the lower row is the number when fetching instructions at 3 ϕ access in high-speed running under the condition of 25 MHz $< f(X_{IN}) \leqq 40$ MHz.

The cycles' number of addressing modes concerning DPR is the number of the case of DPR="0". When DPR ≠ "0", the number of cycles is incremented by 1.

The number of cycles shown in the table differs according to the bytes fetched into the instruction queue buffer, or according to whether the memory accessed is odd address or even address. It also differs when the external area is accessed by BYTE="H". This table shows the fastest number of cycles for each instruction.

Note 1. The operation code at the upper row is used for accumulator A, and the operation at the lower row is used for accumulator B.

Note 2. When setting flag m=0 to handle the data as 16-bit data in the immediate addressing mode, the number of bytes increments by 1.

Note 3. The operation code on the upper row is used for branching in the range of –128 to +127, and the operation code on the lower row is used for branching in the range of –32768 to +32767.

Note 4. When handling 16-bit data with flag m=0, the byte in the table is incremented by 1.

Note 5.

Type of register	A	B	X	Y	DPR	DT	PG	PS
Number of cycles	2	2	2	2	2	1	1	2

The number of cycles corresponding to the register to be pushed are added. The number of cycles when no pushing is done is 11. i_1 indicates the number of registers among A, B, X, Y, DPR, and PS to be saved. i_2 indicates the number of registers among DT and PG to be saved.

Note 6.

Type of register	A	B	X	Y	DPR	DT	PS
Number of cycles	3	3	3	3	4	3	3

The number of cycles corresponding to the register to be pulled are added. The number of cycles when no pulling is done is 12. i_1 indicates the number of registers among A, B, X, Y, DT, and PS to be restored. i_2=1 when DPR is to be restored, and i_2=0 when DPR is not to be restored.

Note 7. The number of cycles is the case when the number of bytes to be transferred is even.
When the number of bytes to be transferred is odd, the number is calculated as;
$5 + (i/2) \times 7 + 6$
Note that, (i/2) shows the integer part when i is divided by 2.

Note 8. The number of cycles is the case when the number of bytes to be transfered is even.
When the number of bytes to be transfered is odd, the number is calculated as;
$9 + (i/2) \times 7 + 8$
Note that, (i/2) shows the integer part when i is divided by 2.

Note 9. The number of cycles is the case in the 16-bit ÷ 8-bit operation. The number of cycles is incremented by 8 for 32-bit ÷ 16-bit operation.

Note 10. The number of cycles is the case in the 8-bit × 8-bit operation. The number of cycles is incremented by 4 for 16-bit × 16-bit operation.

Note 11. When setting flag x=0 to handle the data as 16-bit data in the immediate addressing mode, the number of bytes increments by 1.

Note 12. When flag m is 0, the byte in the table is incremented by 1.

Note 13. When a zero division interrupt occurs, the number of cycles is the number when it does not occur decremented by 3. It is regardless of the data length.

Note 14. When a zero division interrupt occurs, the number of cycles is the number when it does not occur decremented by 5. It is regardless of the data length.

Note 15. The number of cycles is the case when flag *m* is 1.
When flag m=0, the number is calculated as;
$6 + 20 \times i$

Symbols in machine instructions table

Symbol	Description	Symbol	Description
IMP	Implied addressing mode	∀	Exclusive OR
IMM	Immediate addressing mode	—	Negation
A	Accumulator addressing mode	←	Movement to the arrow direction
DIR	Direct addressing mode	→	
DIR, b	Direct bit addressing mode	⇔	
DIR, X	Direct indexed X addressing mode	ACC	Accumulator
DIR, Y	Direct indexed Y addressing mode	A	Accumulator A
(DIR)	Direct indirect addressing mode	AH	Accumulator A's upper 8 bits
(DIR, X)	Direct indexed X indirect addressing mode	AL	Accumulator A's lower 8 bits
(DIR), Y	Direct indirect indexed Y addressing mode	B	Accumulator B
L (DIR)	Direct indirect long addressing mode	BH	Accumulator B's upper 8 bits
L (DIR), Y	Direct indirect long indexed Y addressing mode	BL	Accumulator B's lower 8 bits
ABS	Absolute addressing mode	X	Index register X
ABS, b	Absolute bit addressing mode	XH	Index register X's upper 8 bits
ABS, X	Absolute indexed X addressing mode	XL	Index register X's lower 8 bits
ABS, Y	Absolute indexed Y addressing mode	Y	Index register Y
ABL	Absolute long addressing mode	YH	Index register Y's upper 8 bits
ABL, X	Absolute long indexed X addressing mode	YL	Index register Y's lower 8 bits
(ABS)	Absolute indirect addressing mode	S	Stack pointer
L (ABS)	Absolute indirect long addressing mode	PC	Program counter
(ABS, X)	Absolute indexed X indirect addressing mode	PCH	Program counter's upper 8 bits
STK	Stack addressing mode	PCL	Program counter's lower 8 bits
REL	Relative addressing mode	PG	Program bank register
DIR, b R	Direct bit relative addressing mode	DT	Data bank register
ABS, b, R	Absolute bit relative addressing mode	DPR	Direct page register
SR	Stack pointer relative addressing mode	DPRH	Direct page register's upper 8 bits
(SR), Y	Stack pointer relative indirect indexed Y	DPRL	Direct page register's lower 8 bits
	addressing mode	PS	Processor status register
BLK	Block transfer addressing mode	PSH	Processor status register's upper 8 bits
Multiplied	Multiply and accumulate addressing mode	PSL	Processor status register's lower 8 bits
accumulation		PSb	Bit in processor status register
op	Operation code	M	Memory
n	Number of cycle	M(S)	Contents of memory at address indicated by
#	Number of byte		stack pointer
C	Carry flag	Mb	Bit in memory location
Z	Zero flag	ADH	Value of 24-bit address's upper 8-bit (A_{23}–A_{16})
I	Interrupt disable flag	ADM	Value of 24-bit address's middle 8-bit (A_{15}–A_8)
D	Decimal operation mode flag	ADL	Value of 24-bit address's lower 8-bit (A_7–A_0)
x	Index register length selection flag	IMM	Immediate value
m	Data length selection flag	EAR	Executed address (16 bits)
V	Overflow flag	EARH	Upper 8-bit address executed
N	Negative flag	EARL	Lower 8-bit address executed
IPL	Processor interrupt priority level	bn	Bit position of accumulator or memory indicated
+	Addition		by n
−	Subtraction	i	Number of transfer byte, rotation or repeated
×	Multiplication		operation
÷	Division	i_1, i_2	Number of registers pushed or pulled
∧	Logical AND		
∨	Logical OR		

Appendix 2. 7751 series instruction code table

7751 SERIES INSTRUCTION CODE TABLE-1

	D_3–D_0	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
D_7–D_4	Hexadecimal notation	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0	BRK	ORA A,(DIR,X)		ORA A,SR	SEB DIR,b	ORA A,DIR	ASL DIR	ORA A,L(DIR)	PHP	ORA A,IMM	ASL A	PHD	SEB ABS,b	ORA A,ABS	ASL ABS	ORA A,ABL
0001	1	BPL	ORA A,(DIR),Y	ORA A,(DIR)	ORA A,(SR),Y	CLB DIR,b	ORA A,DIR,X	ASL DIR,X	ORA A,L(DIR),Y	CLC	ORA A,ABS,Y	DEC A	TAS	CLB ABS,b	ORA A,ABS,X	ASL ABS,X	ORA A,ABL,X
0010	2	JSR ABS	AND A,(DIR,X)	JSR ABL	AND A,SR	BBS DIR,b,R	AND A,DIR	ROL DIR	AND A,L(DIR)	PLP	AND A,IMM	ROL A	PLD	BBS ABS,b,R	AND A,ABS	ROL ABS	AND A,ABL
0011	3	BMI	AND A,(DIR),Y	AND A,(DIR)	AND A,(SR),Y	BBC DIR,b,R	AND A,DIR,X	ROL DIR,X	AND A,L(DIR),Y	SEC	AND A,ABS,Y	INC A	TSA	BBC ABS,b,R	AND A,ABS,X	ROL ABS,X	AND A,ABL,X
0100	4	RTI	EOR A,(DIR,X)	Note 1	EOR A,SR	MVP	EOR A,DIR	LSR DIR	EOR A,L(DIR)	PHA	EOR A,IMM	LSR A	PHG	JMP ABS	EOR A,ABS	LSR ABS	EOR A,ABL
0101	5	BVC	EOR A,(DIR),Y	EOR A,(DIR)	EOR A,(SR),Y	MVN	EOR A,DIR,X	LSR DIR,X	EOR A,L(DIR),Y	CLI	EOR A,ABS,Y	PHY	TAD	JMP ABL	EOR A,ABS,X	LSR ABS,X	EOR A,ABL,X
0110	6	RTS	ADC A,(DIR,X)	PER	ADC A,SR	LDM DIR	ADC A,DIR	ROR DIR	ADC A,L(DIR)	PLA	ADC A,IMM	ROR A	RTL	JMP (ABS)	ADC A,ABS	ROR ABS	ADC A,ABL
0111	7	BVS	ADC A,(DIR),Y	ADC A,(DIR)	ADC A,(SR),Y	LDM DIR,X	ADC A,DIR,X	ROR DIR,X	ADC A,L(DIR),Y	SEI	ADC A,ABS,Y	PLY	TDA	JMP (ABS,X)	ADC A,ABS,X	ROR ABS,X	ADC A,ABL,X
1000	8	BRA REL	STA A,(DIR,X)	BRA REL	STA A,SR	STY DIR	STA A,DIR	STX DIR	STA A,L(DIR)	DEY	Note 2	TXA	PHT	STY ABS	STA A,ABS	STX ABS	STA A,ABL
1001	9	BCC	STA A,(DIR),Y	STA A,(DIR)	STA A,(SR),Y	STY DIR,X	STA A,DIR,X	STX DIR,Y	STA A,L(DIR),Y	TYA	STA A,ABS,Y	TXS	TXY	LDM ABS	STA A,ABS,X	LDM ABS,X	STA A,ABL,X
1010	A	LDY IMM	LDA A,(DIR,X)	LDX IMM	LDA A,SR	LDY DIR	LDA A,DIR	LDX DIR	LDA A,L(DIR)	TAY	LDA A,IMM	TAX	PLT	LDY ABS	LDA A,ABS	LDX ABS	LDA A,ABL
1011	B	BCS	LDA A,(DIR),Y	LDA A,(DIR)	LDA A,(SR),Y	LDY DIR,X	LDA A,DIR,X	LDX DIR,Y	LDA A,L(DIR),Y	CLV	LDA A,ABS,Y	TSX	TYX	LDY ABS,X	LDA A,ABS,X	LDX ABS,Y	LDA A,ABL,X
1100	C	CPY IMM	CMP A,(DIR,X)	CLP IMM	CMP A,SR	CPY DIR	CMP A,DIR	DEC DIR	CMP A,L(DIR)	INY	CMP A,IMM	DEX	WIT	CPY ABS	CMP A,ABS	DEC ABS	CMP A,ABL
1101	D	BNE	CMP A,(DIR),Y	CMP A,(DIR)	CMP A,(SR),Y	PEI	CMP A,DIR,X	DEC DIR,X	CMP A,L(DIR),Y	CLM	CMP A,ABS,Y	PHX	STP	JMP L(ABS)	CMP A,ABS,X	DEC ABS,X	CMP A,ABL,X
1110	E	CPX IMM	SBC A,(DIR,X)	SEP IMM	SBC A,SR	CPX DIR	SBC A,DIR	INC DIR	SBC A,L(DIR)	INX	SBC A,IMM	NOP	PSH	CPX ABS	SBC A,ABS	INC ABS	SBC A,ABL
1111	F	BEQ	SBC A,(DIR),Y	SBC A,(DIR)	SBC A,(SR),Y	PEA	SBC A,DIR,X	INC DIR,X	SBC A,L(DIR),Y	SEM	SBC A,ABS,Y	PLX	PUL	JSR (ABS,X)	SBC A,ABS,X	INC ABS,X	SBC A,ABL,X

Notes 1. 42_{16} specifies the contents of the INSTRUCTION CODE TABLE-2. About the second word's codes, refer to the INSTRUCTION CODE TABLE-2.

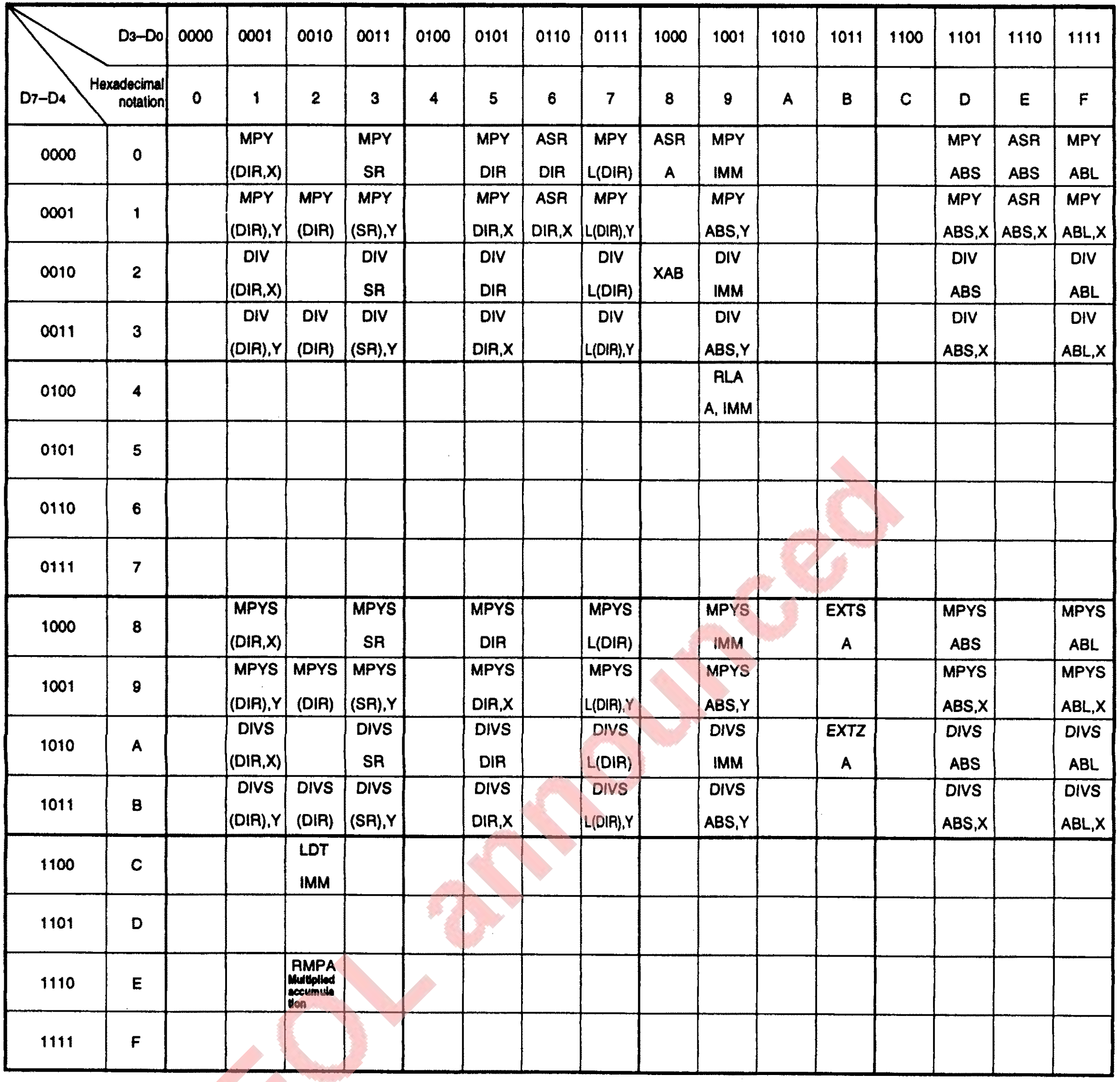
2. 89_{16} specifies the contents of the INSTRUCTION CODE TABLE-3. About the second word's codes, refer to the INSTRUCTION CODE TABLE-3.

7751 SERIES INSTRUCTION CODE TABLE-2(The first word's code of each instruction is 42_{16}.)

	D_3–D_0	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
D_7–D_4	Hexadecimal notation	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0		ORA B,(DIR,X)		ORA B,SR		ORA B,DIR		ORA B,L(DIR)	ASR B	ORA B,IMM	ASL B			ORA B,ABS		ORA B,ABL
0001	1		ORA B,(DIR),Y	ORA B,(DIR)	ORA B,(SR),Y		ORA B,DIR,X		ORA B,L(DIR),Y		ORA B,ABS,Y	DEC B	TBS		ORA B,ABS,X		ORA B,ABL,X
0010	2		AND B,(DIR,X)		AND B,SR		AND B,DIR		AND B,L(DIR)		AND B,IMM	ROL B			AND B,ABS		AND B,ABL
0011	3		AND B,(DIR),Y	AND B,(DIR)	AND B,(SR),Y		AND B,DIR,X		AND B,L(DIR),Y		AND B,ABS,Y	INC B	TSB		AND B,ABS,X		AND B,ABL,X
0100	4		EOR B,(DIR,X)		EOR B,SR		EOR B,DIR		EOR B,L(DIR)	PHB	EOR B,IMM	LSR B			EOR B,ABS		EOR B,ABL
0101	5		EOR B,(DIR),Y	EOR B,(DIR)	EOR B,(SR),Y		EOR B,DIR,X		EOR B,L(DIR),Y		EOR B,ABS,Y		TBD		EOR B,ABS,X		EOR B,ABL,X
0110	6		ADC B,(DIR,X)		ADC B,SR		ADC B,DIR		ADC B,L(DIR)	PLB	ADC B,IMM	ROR B			ADC B,ABS		ADC B,ABL
0111	7		ADC B,(DIR),Y	ADC B,(DIR)	ADC B,(SR),Y		ADC B,DIR,X		ADC B,L(DIR),Y		ADC B,ABS,Y		TDB		ADC B,ABS,X		ADC B,ABL,X
1000	8		STA B,(DIR,X)		STA B,SR		STA B,DIR		STA B,L(DIR)			TXB	EXTS B		STA B,ABS		STA B,ABL
1001	9		STA B,(DIR),Y	STA B,(DIR)	STA B,(SR),Y		STA B,DIR,X		STA B,L(DIR),Y	TYB	STA B,ABS,Y				STA B,ABS,X		STA B,ABL,X
1010	A		LDA B,(DIR,X)		LDA B,SR		LDA B,DIR		LDA B,L(DIR)	TBY	LDA B,IMM	TBX	EXTZ B		LDA B,ABS		LDA B,ABL
1011	B		LDA B,(DIR),Y	LDA B,(DIR)	LDA B,(SR),Y		LDA B,DIR,X		LDA B,L(DIR),Y		LDA B,ABS,Y				LDA B,ABS,X		LDA B,ABL,X
1100	C		CMP B,(DIR,X)		CMP B,SR		CMP B,DIR		CMP B,L(DIR)		CMP B,IMM				CMP B,ABS		CMP B,ABL
1101	D		CMP B,(DIR),Y	CMP B,(DIR)	CMP B,(SR),Y		CMP B,DIR,X		CMP B,L(DIR),Y		CMP B,ABS,Y				CMP B,ABS,X		CMP B,ABL,X
1110	E		SBC B,(DIR,X)		SBC B,SR		SBC B,DIR		SBC B,L(DIR)		SBC B,IMM				SBC B,ABS		SBC B,ABL
1111	F		SBC B,(DIR),Y	SBC B,(DIR)	SBC B,(SR),Y		SBC B,DIR,X		SBC B,L(DIR),Y		SBC B,ABS,Y				SBC B,ABS,X		SBC B,ABL,X

Appendix 2. 7751 series instruction code table

7751 SERIES INSTRUCTION CODE TABLE-3(The first word’s code of each instruction is 89_{16}.)



	D_3–D_0	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
D_7–D_4	Hexadecimal notation	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0		MPY (DIR,X)		MPY SR		MPY DIR	ASR DIR	MPY L(DIR)	ASR A	MPY IMM				MPY ABS	ASR ABS	MPY ABL
0001	1		MPY (DIR),Y	MPY (DIR)	MPY (SR),Y		MPY DIR,X	ASR DIR,X	MPY L(DIR),Y		MPY ABS,Y				MPY ABS,X	ASR ABS,X	MPY ABL,X
0010	2		DIV (DIR,X)		DIV SR		DIV DIR		DIV L(DIR)	XAB	DIV IMM				DIV ABS		DIV ABL
0011	3		DIV (DIR),Y	DIV (DIR)	DIV (SR),Y		DIV DIR,X		DIV L(DIR),Y		DIV ABS,Y				DIV ABS,X		DIV ABL,X
0100	4										RLA A, IMM						
0101	5																
0110	6																
0111	7																
1000	8		MPYS (DIR,X)		MPYS SR		MPYS DIR		MPYS L(DIR)		MPYS IMM		EXTS A		MPYS ABS		MPYS ABL
1001	9		MPYS (DIR),Y	MPYS (DIR)	MPYS (SR),Y		MPYS DIR,X		MPYS L(DIR),Y		MPYS ABS,Y				MPYS ABS,X		MPYS ABL,X
1010	A		DIVS (DIR,X)		DIVS SR		DIVS DIR		DIVS L(DIR)		DIVS IMM		EXTZ A		DIVS ABS		DIVS ABL
1011	B		DIVS (DIR),Y	DIVS (DIR)	DIVS (SR),Y		DIVS DIR,X		DIVS L(DIR),Y		DIVS ABS,Y				DIVS ABS,X		DIVS ABL,X
1100	C			LDT IMM													
1101	D																
1110	E			RMPA Multiplied accumulation													
1111	F																

MITSUBISHI SEMICONDUCTORS
SOFTWARE MANUAL
7751 Series

Jul. First Edition 1996

Editioned by
Committee of editing of Mitsubishi Semiconductor USER'S MANUAL

Published by
Mitsubishi Electric Corp., Semiconductor Marketing Division

7751 Group
User’s Manual

RENESAS

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan