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.

RENESAS

User's Manual

8

740 Family

Software Manual

RENESAS MCU

Renesas Electronics
www.renesas.com

Rev.2.00 2006.11

REVISION HISTORY	740 Family Software Manual

Rev.	Date	Description	
		Page	Summary
1.00	Aug 29, 1997	–	First edition issued
2.00	Nov 14, 2006	–	Changed to the RENESAS style.
			"Preface" is changed to "Using This Manual".
		4	2.5 Processor Status Register: Description added.
		26	3.2 Instruction Set : Description revised.
		31	ADC : Note 2 is revised.
		53	CMP : Function revised.
		60	DIV : Note 3 is added.
		65	JMP : Note is added.
		72, 133, 134	XX instruction cannot be used for any products → XX instruction cannot be used for some products.
		72	MUL : Note 3 is added.
		74	ORA : N is when bit 7..... → N is "1" when bit 7.....
		78	PLP : Note is added.
		82	RTI : Status flag is revised.
		83	RTS : Operation is revised.
		84	SBC : Note 2 is revised.
		101	WIT : Function is revised.
		102 to 104	3.4 Instructions Related to Interrupt Processing and Subroutine Processing added.
		105	NOTES ON USE : "4.1 Notes on input and output ports" is added.
		107	Fig. 4.3.1 is revised.
			4.3.2 : Description revised.
		108	4.3.3 Distinction of interrupt request bit : Description revised.
			Fig. 4.3.2 is revised.
		110	Fig. 4.4.4 is revised.
		111	"4.4.5 Multiplication and division instruction", "4.4.6 Ports" and "4.4.7 Instruction execution time" are added.
		112	Valid signal for each product : Table is revised and note is added.
		178	Part of instruction table is revised.
		184	Part of instruction code is revised.
			Table of products which unuse these instructions is eliminated.

Using This Manual

This software manual is written for the 740 Family. It applies to all microcomputers integrating the 740 Family CPU core.
The reader of this manual is assumed to have a basic knowledge of electrical circuits, logic circuits, and microcomputers.

740 Family Documents

The following documents were prepared for the 740 family.

Document	Contents
Data Sheet	Hardware overview and electrical characteristics Hardware specifications (pin assignments, memory maps, peripheral specifications, electrical characteristics, timing charts).
Software Manual	Detailed description of assembly instructions and microcomputer performance of each instruction
Application Note	• Usage and application examples of peripheral functions • Sample programs

Table of contents

<Addressing Mode>

<Instructions>

1. OVERVIEW

The distinctive features of the CMOS 8-bit microcomputers 740 Family's software are described below:

1) An efficient instruction set and many addressing modes allow the effective use of ROM.
2) The same bit management, test, and branch instructions can be performed on the Accumulator, memory, or I/O area.
3) Multiple interrupts with separate interrupt vectors allow servicing of different non-periodic events.
4) Byte processing and table referencing can be easily performed using the index addressing mode.
5) Decimal mode needs no software correction for proper decimal operation.
6) The Accumulator does not need to be used in operations using memory and/or I/O.

2. CENTRAL PROCESSING UNIT (CPU)

Six main registers are built into the CPU of the 740 Family.
The Program Counter (PC) is a sixteen-bit register; however, the Accumulator (A), Index Register X (X), Index Register Y (Y), Stack Pointer (S) and Processor Status Register (PS) are eight-bit registers.

☞ Except for the I flag, the contents of these registers are indeterminate after a hardware reset; therefore, initialization is required with some programs (immediately after reset the I flag is set to “1”).

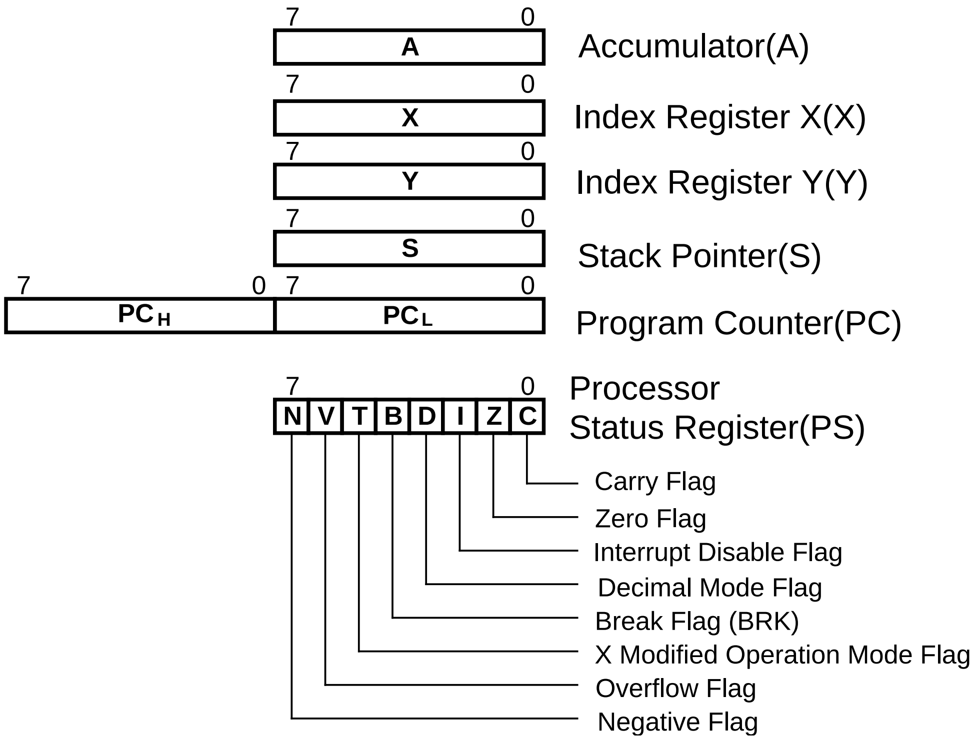


Fig.2.1.1 Register Configuration

2.1 Accumulator (A)

The Accumulator, an eight-bit register, is the main register of the microcomputer.
This general-purpose register is used most frequently for arithmetic operations, data transfer, temporary memory, conditional judgments, etc.

2.2 Index Register X (X), Index Register Y (Y)

The 740 Family has an Index Register X and an Index Register Y, both of which are eight-bit registers.
When using addressing modes which use these index registers, the address, which is added the contents of Index Register to the address specified with operand, is accessed. These modes are extremely effective for referencing subroutine and memory tables.
The index registers also have increment, decrement, compare, and data transfer functions; therefore, these registers can be used as simple accumulators.

2.3 Stack Pointer (S)

The Stack Pointer is an eight-bit register used for generating interrupts and calling subroutines. When an interrupt is received, the following procedure is performed automatically in the indicated sequence:

(1) The contents of the high-order eight bits of the Program Counter (PC_H) are saved to an address using the Stack Pointer contents for the low-order eight bits of the address.
(2) The Stack Pointer contents are decremented by 1.
(3) The contents of the low-order eight bits of the Program Counter (PC_L) are saved to an address using the Stack Pointer Contents for the low-order eight bits of the address.
(4) The Stack Pointer contents are decremented by 1.
(5) The contents of the Processor Status Register (PS) are saved to an address using the Stack Pointer contents for the low-order eight bits of the address.
(6) The Stack Pointer contents are decremented by 1.

The Processor Status Register is not saved when calling subroutines (items (5) and (6) above are not executed). The Processor Status Register is saved by executing the PHP instruction in software.

To prevent data loss when generating interrupts and calling subroutines, it is necessary to save other registers as well. This is done by executing the proper instruction in software while in the interrupt service routine or subroutine.

The high-order eight bits of the address are determined by the Stack Page Selection Bit.

For example, the PHA instruction is executed to save the contents of the Accumulator. Executing the PHA instruction saves the Accumulator contents to an address using the Stack Pointer contents as the low-order eight bits of the address.

The RTI instruction is executed to return from an interrupt routine.

When the RTI instruction is executed, the following procedure is performed automatically in sequence.

(1) The Stack Pointer contents are incremented by 1.
(2) The contents of an address using the Stack Pointer contents as the low-order eight bits of the address is returned to the Processor Status Register (PS).
(3) The Stack Pointer contents are incremented by 1.
(4) The contents of an address using the Stack Pointer as the low-order eight bits of the address is returned to the low-order eight bits of the Program Counter (PC_L).
(5) The Stack Pointer contents are incremented by 1.
(6) The contents of an address using the Stack Pointer as the low-order eight bits of the address is returned to the high-order eight bits of the Program Counter (PC_H).

Steps (1) and (2) are not performed when returning from a subroutine using the RTS instruction. The Processor Status Register should be restored before returning from a subroutine by using the PLP instruction. The Accumulator should be restored before returning from a subroutine or an interrupt servicing routine by using the PLA instruction.

The PLA and PLP instructions increment the Stack Pointer by 1 and return the contents of an address stored in the Stack Pointer to the Accumulator or Processor Status Register, respectively.

☞ Saving data in the stack area gradually fills the RAM area with saved data; therefore, caution must exercised concerning the depth of interrupt levels and subroutine nesting.

2.4 Program Counter (PC)

The Program Counter is a sixteen-bit counter consisting of PCH and PCL, which are each eight-bit registers. The contetnts of the Program Counter indicates the address which an instruction to be executed next is stored.
The 740 Family uses a stored program system; to start a new operation it is necessary to transfer the instruction and relevant data from memory to the CPU.
Normally the Program Counter is used to indicate the next memory address. After each instruction is executed, the next instruction required is read. This cycle is repeated until the program is finished.
☞ The control of the Program Counter of the 740 Family is almost fully automatic. However, caution must be exercised to avoid differences between program flow and Program Counter contents when using the Stack Pointer or directly altering the contents of the Program Counter.

2.5 Processor Status Register (PS)

The Processor Status Register is an eight-bit register consisting of 5 flags which indicate the status of arithmetic operations and 3 flags which determine operation. Immediately after a reset, only the interrupt disable flag is set to "1," and the other flags are undefined. Therefore, initialize the flags that effect program execution. Especially, initialize the T and D flags because of their effect on operation.
Each of these flags is described below. Table 2.5.1 lists the instructions to set/clear each flag. Refer to the section "Appendix 2 MACHINE LANGUAGE INSTRUCTION TABLE" or "3.3 INSTRUCTIONS" for details on when these flags are altered.

[Carry flag C]-- Bit 0
This flag stores any carry or borrow from the Arithmetic Logic Unit (ALU) after an arithmetic operation and is also changed by the Shift or Rotate instruction.
This flag is set by the SEC instruction and is cleared by the CLC instruction.

[Zero flag Z]--- Bit 1
This flag is set when the result of an arithmetic operation or data transfer is "0" and is cleared by any other result.

[Interrupt disable flag I]-------------------------------------- Bit 2
This flag disables interrupts when it is set to "1." This flag immediately becomes "1" when an interrupt is received.
This flag is set by the SEI instruction and is cleared by the CLI instruction.

[Decimal mode flag D]-- Bit 3
This flag determines whether addition and subtraction are performed in binary or decimal notation. Addition and subtraction are performed in binary notation when this flag is set to "0" and as a 2-digit, 1-word decimal numeral when set to "1." Decimal notation correction is performed automatically at this time.
This flag is set by the SED instruction and is cleared by the CLD instruction.
Only the ADC and SBC instructions are used for decimal arithmetic operations.
Note that the flags N, V and Z are invalid when decimal arithmetic operations are performed by these instructions.

[Break flag B] -- Bit 4
This flag determines whether an interrupt was generated with the BRK instruction. When a BRK instruction interrupt occurs, the flag B is set to "1" and saved to the stack; for all other interrupts the flag is set to "0" and saved to the stack.

[X modified operation mode flag T] ---------------------- Bit 5

This flag determines whether arithmetic operations are performed via the Accumulator or directly on a memory location. When the flag is set to "0", arithmetic operations are performed between the Accumulator and memory. When "1", arithmetic operations are performed directly on a memory location.

This flag is set by the SET instruction and is cleared by the CLT instruction.

(1) When the T flag = 0

A ← A * M2

* : indicates an arithmetic operation

A: accumulator contents

M2: contents of a memory location specified by the addressing mode of the arithmetic operation

(2) When the T flag = 1

M1 ← M1 * M2

* : indicates arithmetic operation

M1: contents of a memory location, designated by the contents of Index Register X.

M2: contents of a memory location specified by the addressing mode of arithmetic operation.

[Overflow flag V] -- Bit 6

This flag is set to "1" when an overflow occurs as a result of a signed arithmetic operation. An overflow occurs when the result of an addition or subtraction exceeds +127 ($7F_{16}$) or –128 (80_{16}) respectively.

The CLV instruction clears the Overflow Flag. There is no set instruction.

The overflow flag is also set during the BIT instruction when bit 6 of the value being tested is "1."

☞ Overflows do not occur when the result of an addition or subtraction is equal to or smaller than the above numerical values, or for additions involving values with different signs.

[Negative flag N] -- Bit 7

This flag is set to match the sign bit (bit 7) of the result of a data or arithmetic operation. This flag can be used to determine whether the results of arithmetic operations are positive or negative, and also to perform a simple bit test.

Table 2.5.1 Instructions to set/clear each flag of processor status register

	Flag C	Flag Z	Flag I	Flag D	Flag B	Flag T	Flag V	Flag N
Set instruction	SEC	——	SEI	SED	——	SET	——	——
Clear instruction	CLC	——	CLI	CLD	——	CLT	CLV	——

3. INSTRUCTIONS

3.1 Addressing Mode

The 740 Family has 19 addressing modes and a powerful memory access capability. When extracting data required for arithmetic and logic operations from memory or when storing the results of such operations in memory, a memory address must be specified. The specification of the memory address is called addressing. The data required for addressing and the registers involved are described below. The 740 Family instructions can be classified into three kinds, by the number of bytes required in program memory for the instruction: 1-byte, 2-byte and 3-byte instructions. In each case, the first byte is known as the “Op-Code (operation code)” which forms the basis of the instruction. The second or third byte is called the “operand” which affects the addressing. The contents of index registers X and Y can also effect the addressing.

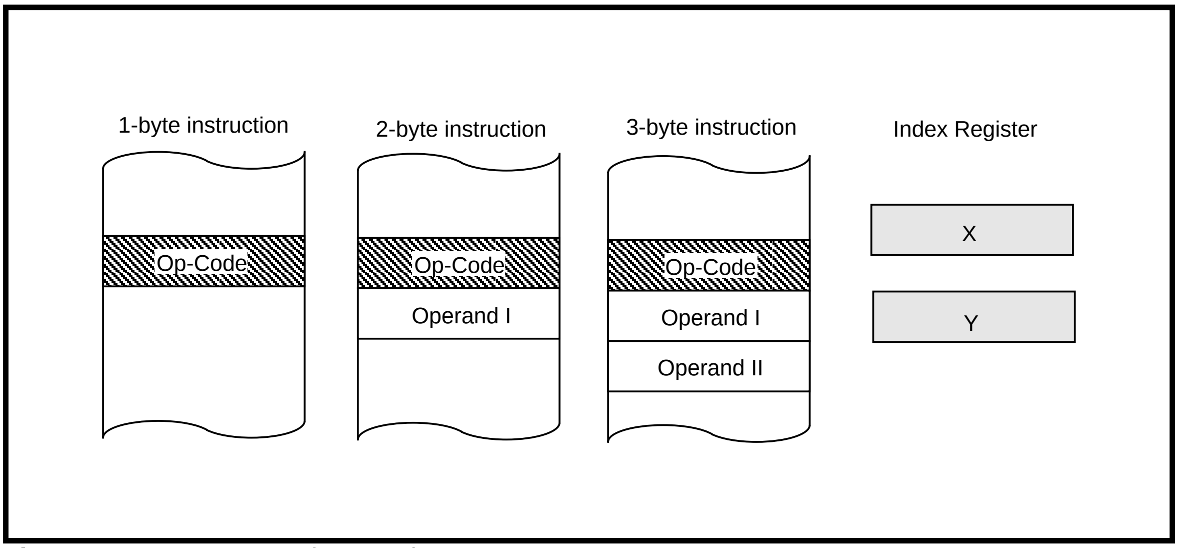


Fig.3.1.1 Byte Structure of Instructions

Although there are many addressing modes, there is always a particular memory location specified. What differs is whether the operand, or the index register contents, or a combination of both should be used to specify the memory or jump destination. Based on these 3 types of instructions, the range of variation is increased and operation is enhanced by combinations of the bit operation instructions, jump instruction, and arithmetic instructions.

As for 1-byte instruction, an accumulator or a register is specified, so that the instruction does not have “operand,” which specify memory.

Immediate

Addressing mode : **Immediate**

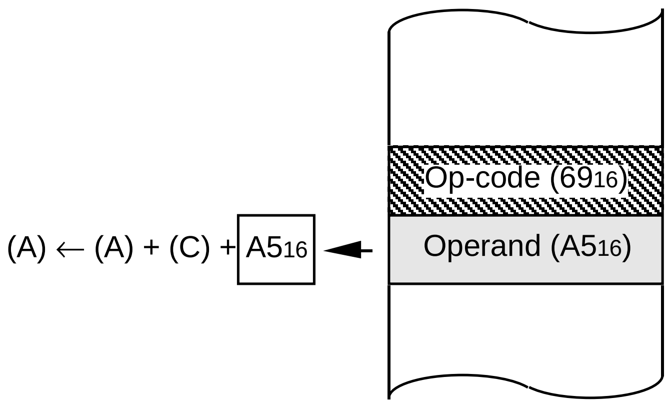
Function : **Specifies the Operand as the data for the instruction.**

Instructions : **ADC, AND, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA, SBC**

Example : Mnemonic — Machine code

ΔADCΔ#$A5 — **$69_{16}$ $A5_{16}$**

This symbol(#) indicates the Immediate addressing mode.



Accumulator

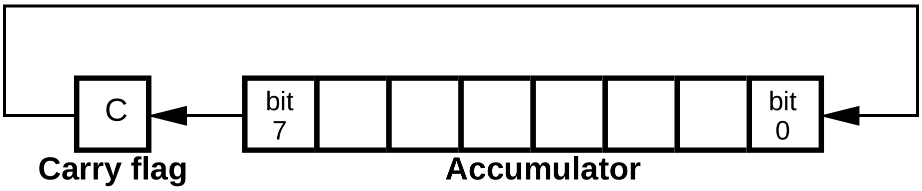
Addressing mode : **Accumulator**

Function : **Specifies the contents of the Accumulator as the data for the instruction.**

Instructions : **ASL, DEC, INC, LSR, ROL, ROR**

Example : Mnemonic Machine code

ΔROLΔA **2A16**



Zero Page

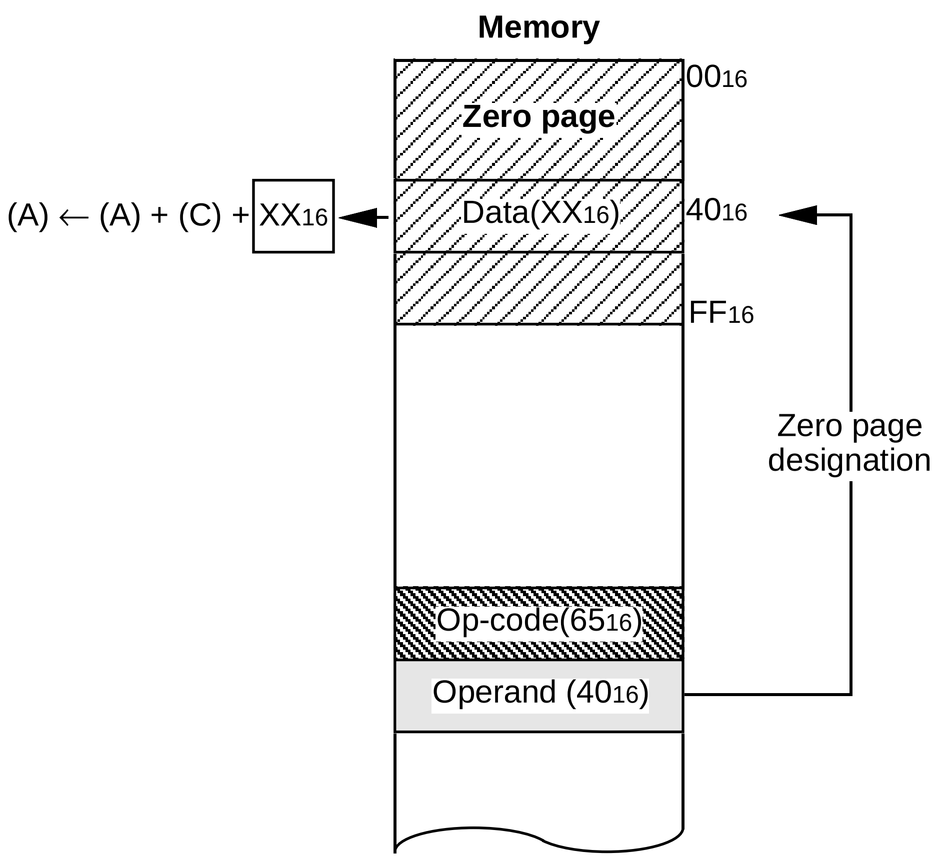
Addressing mode : **Zero Page**

Function : **Specifies the contents in a Zero Page memory location as the data for the instruction. The address in the Zero Page memory location is determined by using Operand as the low-order byte of the address and 00_{16} as the high-order byte.**

Instructions : **ADC, AND, ASL, BIT, CMP, COM, CPX, CPY, DEC, EOR, INC, LDA, LDM, LDX, LDY, LSR, ORA, ROL, ROR, RRF, SBC, STA, STX, STY, TST**

Example : Mnemonic / Machine code

Mnemonic	Machine code
ΔADCΔ$40	**65_{16} 40_{16}**



Zero Page X

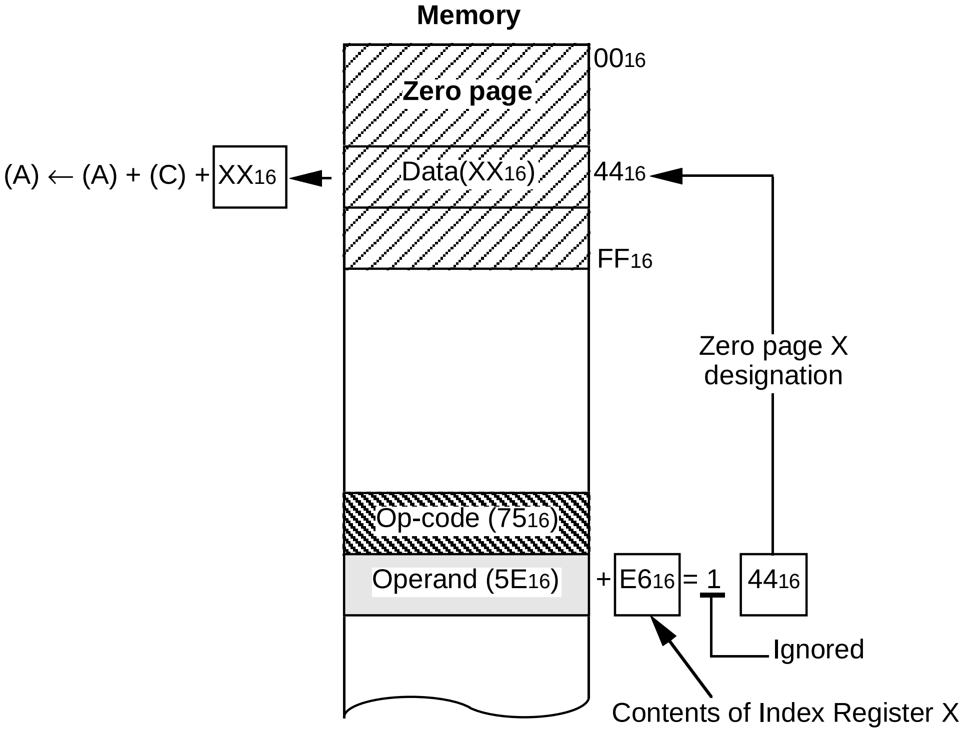
Addressing mode : **Zero Page X**

Function : **Specified the contents in a Zero Page memory location as the data for the instruction. The address in the Zero Page memory location is determined by the following:**
(a) Operand and the Index Register X are added. (If as a result of this addition a carry occurs, it is ignored.)
(b) The result of the addition is used as the low-order byte of the address and 00_{16} as the high-order byte.

Instructions : **ADC, AND, ASL, CMP, DEC, DIV, EOR, INC, LDA, LDY, LSR, MUL, ORA, ROL, ROR, SBC, STA, STY**

Example : Mnemonic — Machine code

ΔADCΔ$5E,X — **$75_{16}$ $5E_{16}$**



Zero Page Y

Addressing mode : **Zero Page Y**

Function : **Specifies the contents in a Zero Page memory location as the data for the instruction. The address in the Zero Page memory location is determined by the following:**

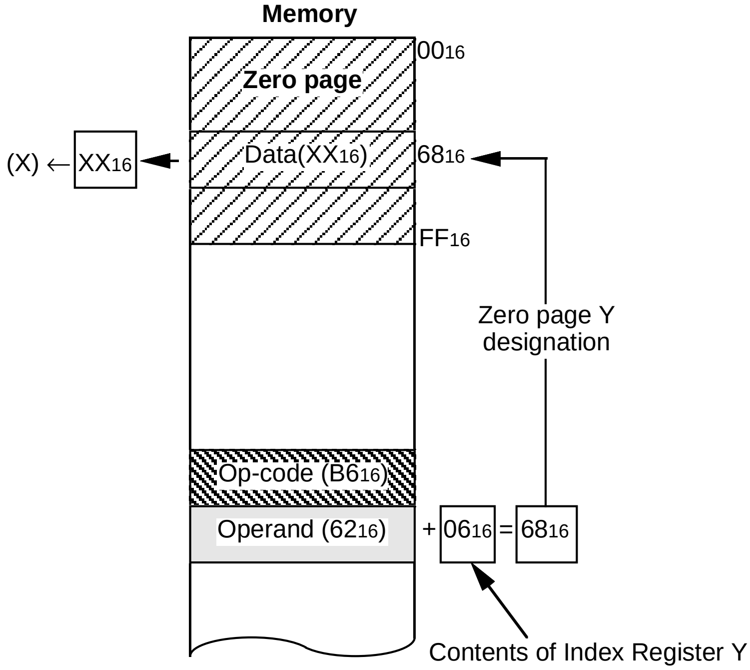
(a) Operand and the Index Register Y are added (if as a result of this addition a carry occurs, it is ignored).

(b) The result of the addition is used as the low-order byte of the address and 00_{16} as the high-order byte.

Instructions : **LDX, STX**

Example :

Mnemonic	Machine code
ΔLDXΔ$62,Y	**$B6_{16}$ 62_{16}**



Absolute

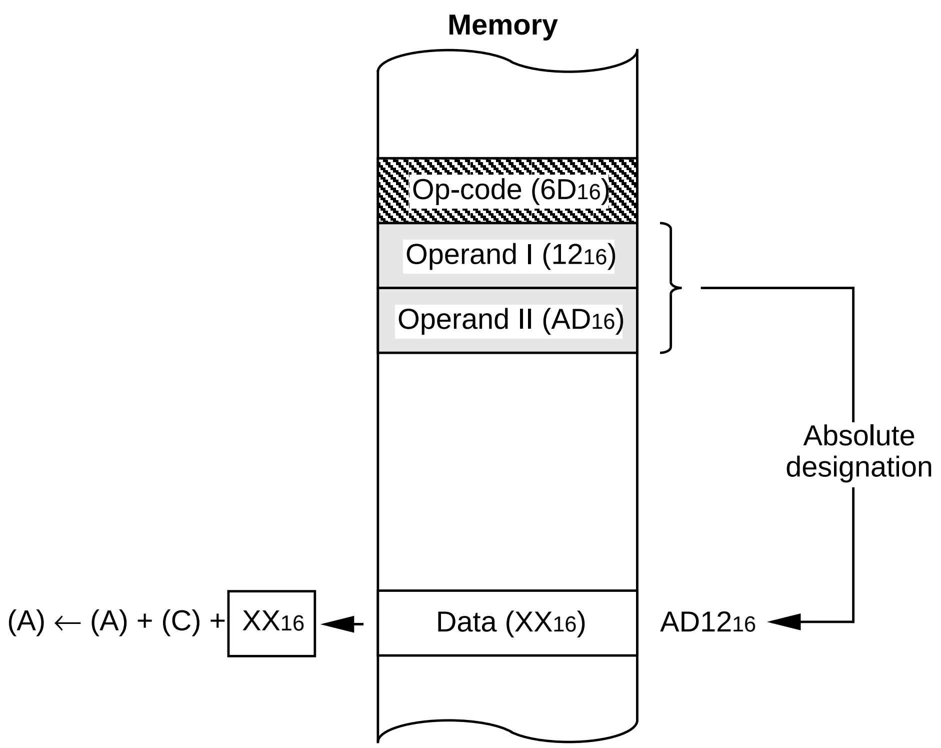
Addressing mode : **Absolute**

Function : **Specifies the contents in a memory location as the data for the instruction. The address in the memory location is determined by using Operand I as the low-order byte of the address and Operand II as the high-order byte.**

Instructions : **ADC, AND, ASL, BIT, CMP, CPX, CPY, DEC, EOR, INC, JMP, JSR, LDA, LDX, LDY, LSR, ORA, ROL, ROR, SBC, STA, STX, STY**

Example : Mnemonic — Machine code

ΔADCΔ$AD12 — **$6D_{16}$ 12_{16} AD_{16}**



Absolute X

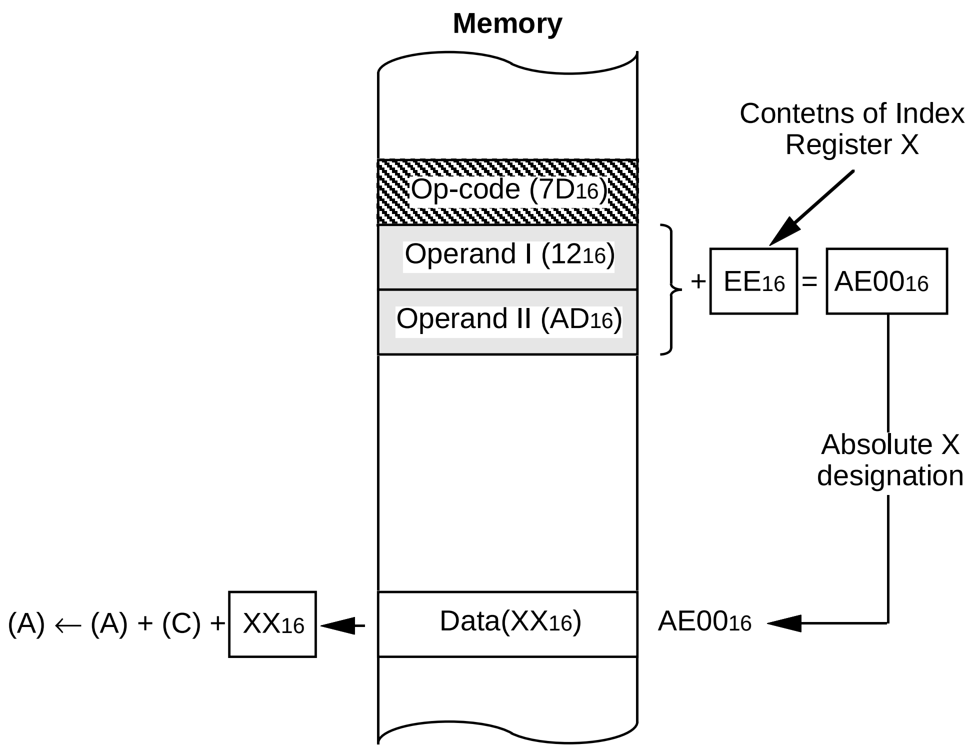
Addressing mode : **Absolute X**

Function : **Specifies the contents in a memory location as the data for the instruction. The address in the memory location is determined by the following:**
(a) Operand I is used as the low-order byte of an address, Operand II as the high-order byte.
(b) Index Register X is added to the address above. The result is the address in the memory location.

Instructions : **ADC, AND, ASL, CMP, DEC, EOR, INC, LDA, LDY, LSR, ORA, ROL, ROR, SBC, STA**

Example : Mnemonic — Machine code

ΔADCΔ$AD12, X — **$7D_{16}$ 12_{16} AD_{16}**



Absolute Y

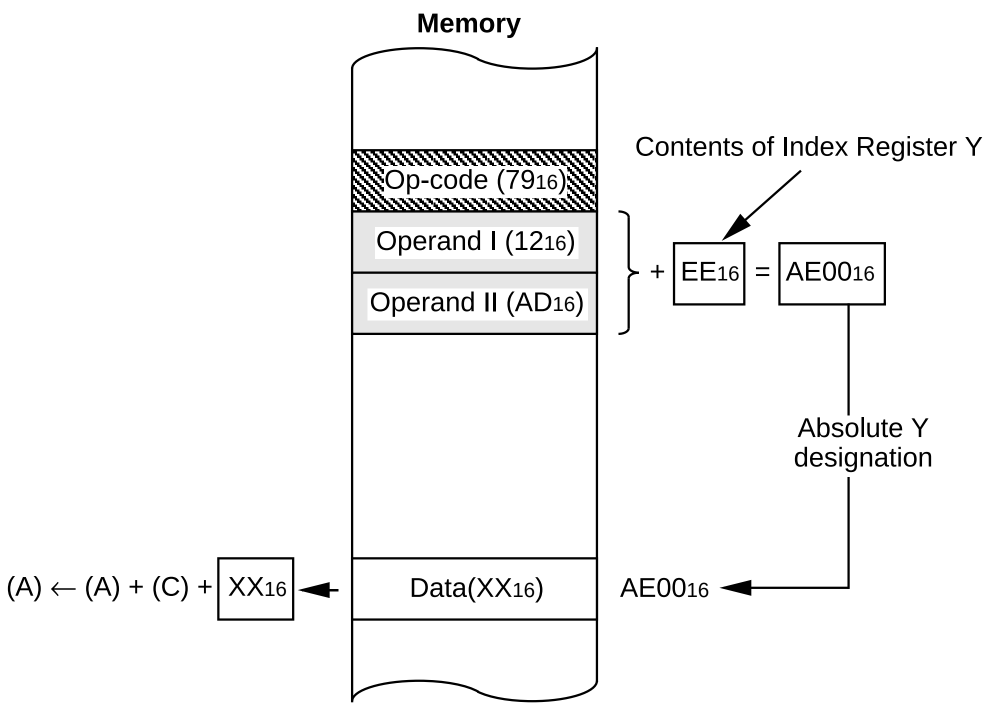
Addressing mode : **Absolute Y**

Function : **Specifies the contents in a memory location as the data for the instruction. The address in the memory location is determined by the following:**
(a) Operand I is used as the low-order byte of an address, Operand II as the high-order byte.
(b) Index Register Y is added to the address above. The result is the address in the memory location.

Instructions : **ADC, AND, CMP, EOR, LDA, LDX, ORA, SBC, STA**

Example : Mnemonics — Machine code

ΔADCΔ$AD12, Y — **$79_{16}$ 12_{16} AD_{16}**



Implied

Addressing mode : **Implied**

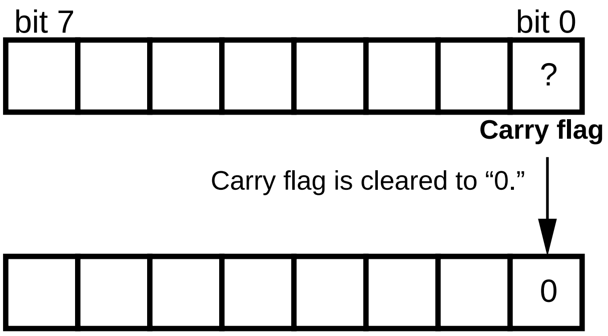
Function : **Operates on a given register or the Accumulator, but the address is always inherent in the instruction.**

Instructions : **BRK, CLC, CLD, CLI, CLT, CLV, DEX, DEY, INX, INY, NOP, PHA, PHP, PLA, PLP, RTI, RTS, SEC, SED, SEI, SET, STP, TAX, TAY, TSX, TXA, TXS, TYA, WIT**

Example : Mnemonic Machine code

ΔCLC **18_{16}**

Processor status register



Relative

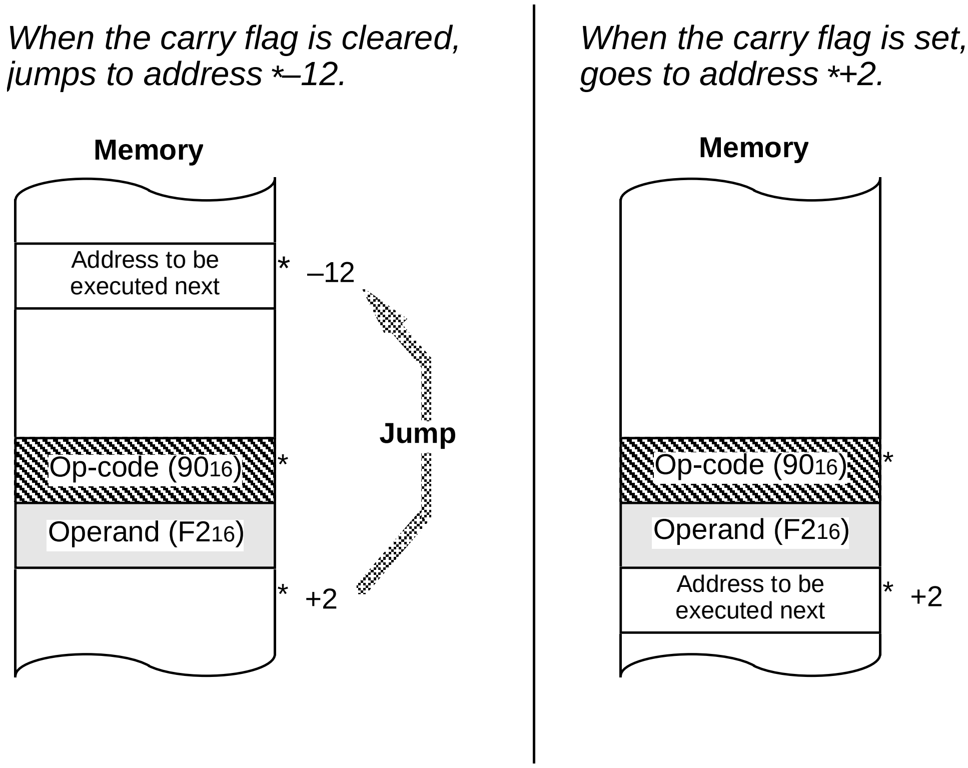
Addressing mode

Addressing mode : **Relative**

Function : **Specifies the address in a memory location where the next Op-Code is located. When the branch condition is satisfied, Operand and the Program Counter are added. The result of this addition is the address in the memory location. When the branch condition is not satisfied, the next instruction is executed.**

Instructions : **BCC, BCS, BEQ, BMI, BNE, BPL, BRA, BVC, BVS**

Example : Mnemonic — **ΔBCCΔ*–12** (12: Decimal) — Machine code — **90_{16} $F2_{16}$**



Indirect X

Addressing mode : **Indirect X**

Function : **Specifies the contents in a memory location as the data for the instruction. The address in the memory location is determined by the following:**

(a) A Zero Page memory location is determined by the adding the Operand and Index Register X (if as a result of this addition a carry occurs, it is ignored).

(b) The result of the addition is used as the low-order byte of an address in the Zero Page memory location and 00_{16} as the high-order byte.

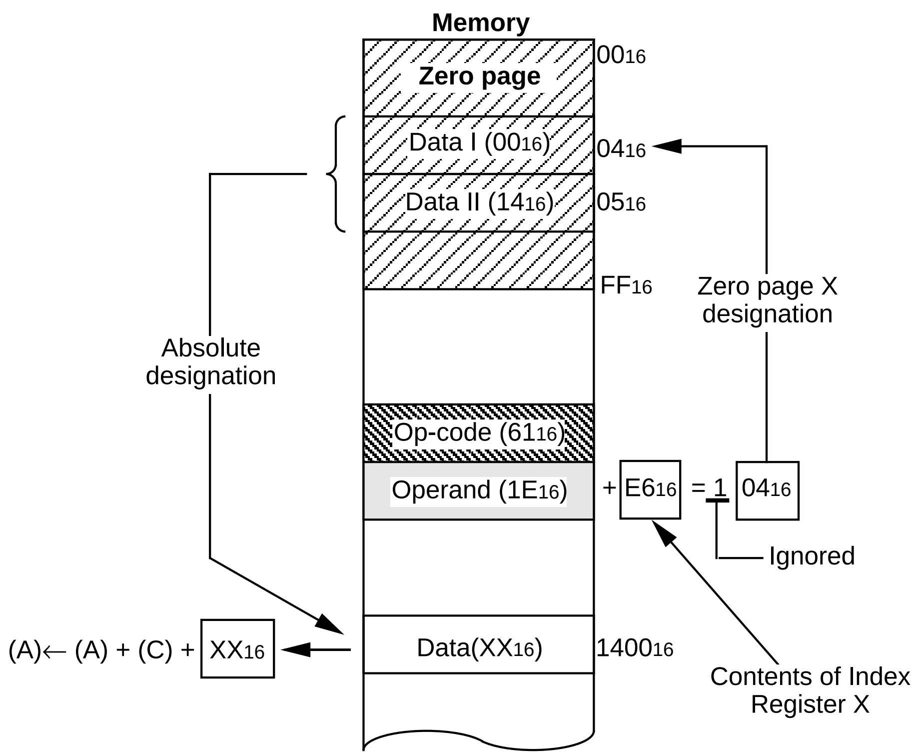
(c) The contents of the address in the Zero Page memory location is used as the low-order byte of the address in the memory location.

(d) The next Zero Page memory location is used as the high-order byte of the address in the memory location.

Instructions : **ADC, AND, CMP, EOR, LDA, ORA, SBC, STA**

Example : Mnemonic / Machine code

Mnemonic	Machine code
ΔADCΔ($1E,X)	**61_{16} $1E_{16}$**



Assuming that "00_{16}" for Data I, and "14_{16}" for Data II are stored in advance.

Indirect Y

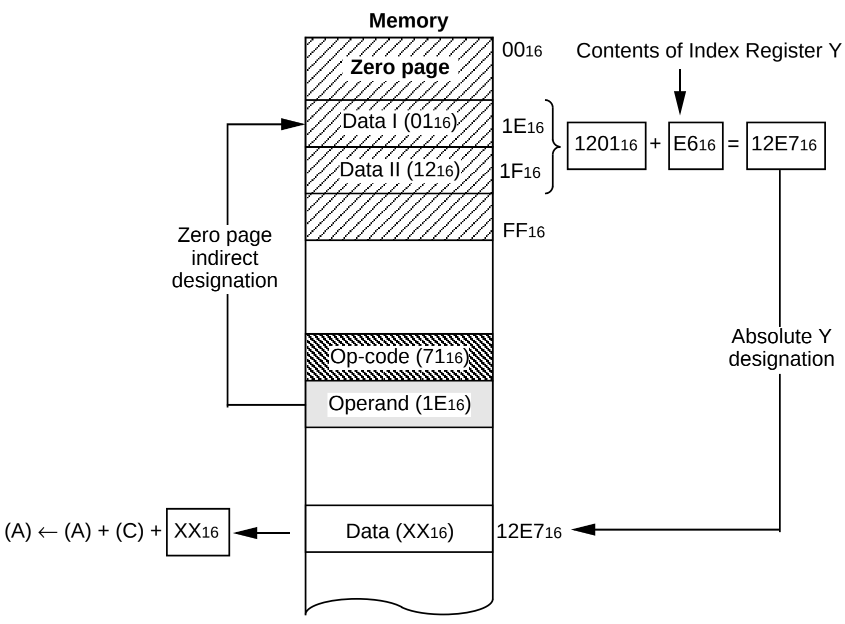
Addressing mode : **Indirect Y**

Function : **Specifies the contents in a memory location as the data for the instruction. The address in the memory location is determined by the following:**
(a) The Operand is used the low-order byte of an address in the Zero Page memory location and 00_{16} of the high-order byte.
(b) The contents of the address in the Zero Page memory location is used as the low-order byte of an address. The next Zero Page memory location is used as the high-order byte.
(c) The Index Register Y is added to the address in Step b. The result of this addition is the address in the memory location.

Instructions : **ADC, AND, CMP, EOR, LDA, ORA, SBC, STA**

Example : Mnemonic — Machine code

ΔADCΔ($1E),Y — **$71_{16}$ $1E_{16}$**



Assuming that "01_{16}" for Data I, and "12_{16}" for Data II are stored in advance.

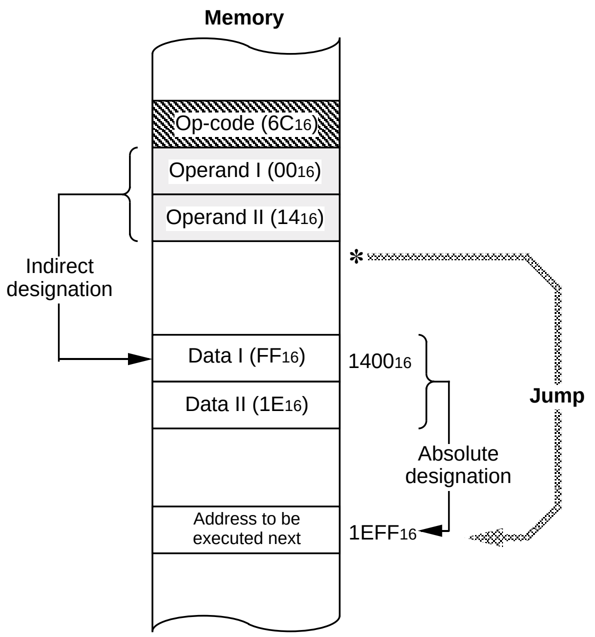
Indirect Absolute Addressing mode

Addressing mode : **Indirect Absolute**

Function : **Specifies the address in a memory location as the jump destination address.**
The address in the memory location is determined by the following:
(a) Operand I is used as the low-order byte of an address and Operand II as the high-order byte.
(b) The contents of the address above is used as the low-order byte and the contents of the next address as the high-order byte.
(c) The high-order and low-order bytes in step b together form the address in the memory location.

Instructions : **JMP**

Example : Mnemonic Machine code
ΔJMPΔ($1400) **$6C_{16}$ 00_{16} 14_{16}**



Assuming that "FF_{16}" for Data I, and "$1E_{16}$" for Data II are stored in advance.

Note: The page's last address (address $XXFF_{16}$) cannot be specified for the indirect designation address; in other words, JMP ($XXFF) cannot be executed.

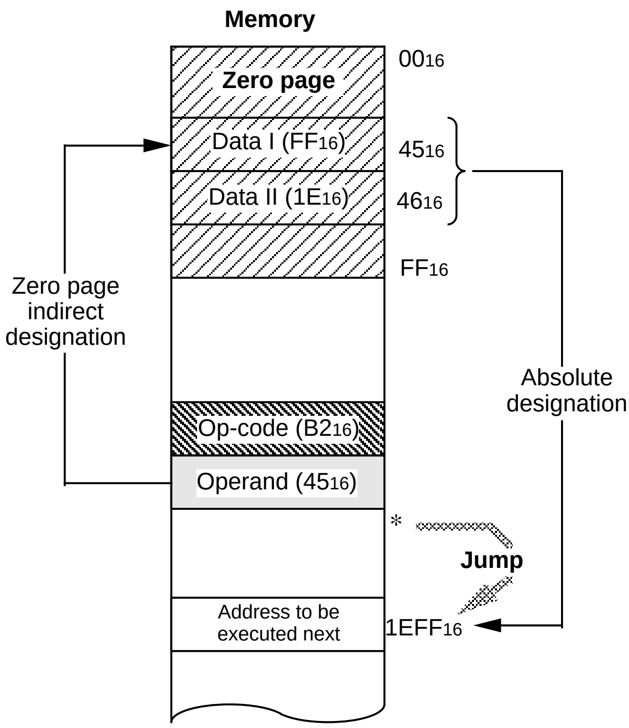
Zero Page Indirect Addressing mode

Addressing mode : **Zero Page Indirect Absolute**

Function : **Specifies the address in a memory location as the jump destination address. The address in the memory location is determined by the following:**
(a) Operand is used as the low-order byte of an address in the Zero Page memory location and 00_{16} as the high-order byte.
(b) The contents of the address in the Zero Page memory location is used as the low-order byte and the contents of the next Zero Page memory location as high-order byte.
(c) The high-order and low-order bytes in step b together form the address of the memory location.

Instructions : **JMP, JSR**

Example : Mnemonic Machine code
ΔJMPΔ($45) **$B2_{16}$ 45_{16}**



Assuming that "FF_{16}" for Data I, and "$1E_{16}$" for Data II are stored in advance.

Special Page

Addressing mode : **Special Page**

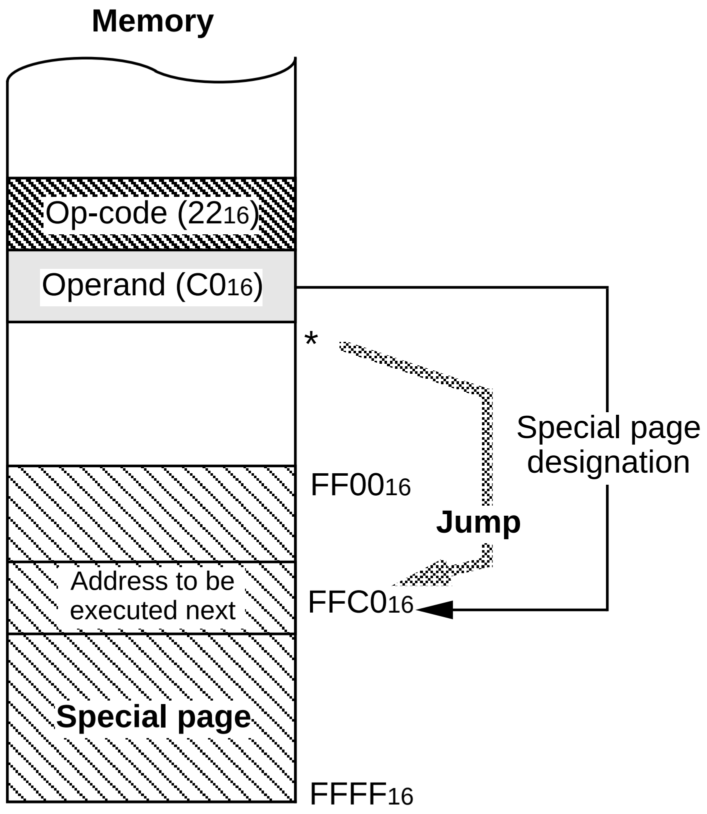
Function : **Specifies the address in a Special Page memory location as the jump destination address. The address in the Special Page memory location is determined by using Operand as the low-order byte of the address and FF_{16} as the high-order byte.**

Instructions : **JSR**

Example : Mnemonic Machine code

ΔJSRΔ\$FFC0 **22_{16} $C0_{16}$**

This symbol indicates the Special page mode.



Zero Page Bit

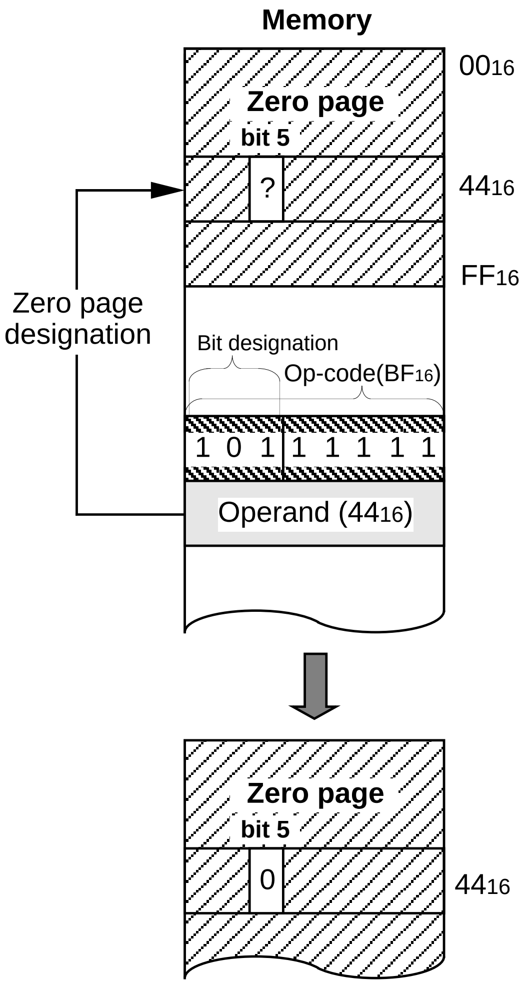
Addressing mode : **Zero Page Bit**

Function : **Specifies one bit of the contents in a Zero Page memory location as the data for the instruction. Operand is used as the low-order byte of the address in the Zero Page memory location and 00_{16} as the high-order byte. The bit position is designated by the high-order three bits of the Op-code.**

Instructions : **CLB, SEB**

Example : Mnemonic Machine code

ΔCLBΔ5,$44 **$BF_{16}$ 44_{16}**



Accumulator Bit

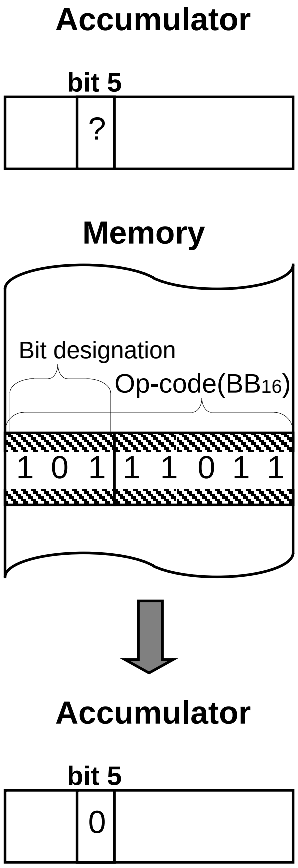
Addressing mode : **Accumulator Bit**

Function : **Specifies one bit of the Accumulator as the data for the instruction. The bit position is designated by the high-order three bits of the Op-Code.**

Instruction: **CLB, SEB**

Example : Mnemonic — Machine code

ΔCLBΔ5,A — **BB_{16}**



Accumulator Bit Relative

Addressing mode :**Accumulator Bit Relative**

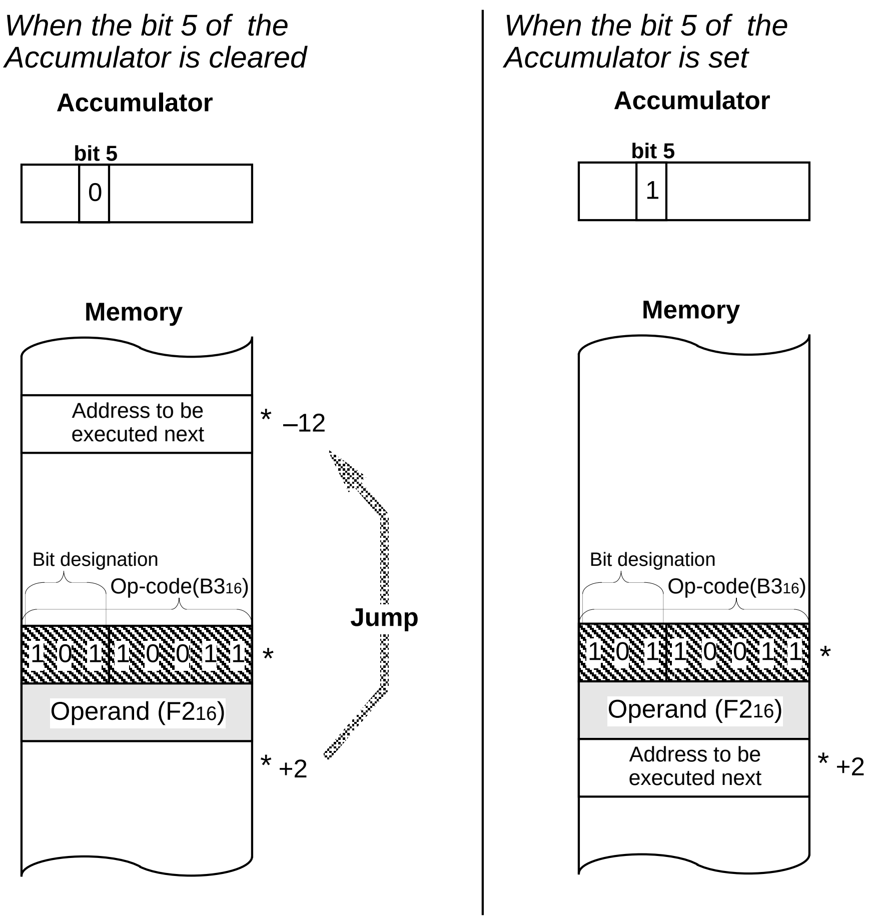
Function :**Specifies the address in a memory location where the next Op-Code is located. The bit position is designated by the high-order three bits of the Op-Code. If the branch condition is satisfied, Operand and the Program Counter are added. The result of this addition is the address in the memory location. When the branch condition is not satisfied, the next instruction is executed.**

Instructions :**BBC, BBS**

Example :Mnemonic Machine code

ΔBBCΔ5,A,*–12 **B3$_{16}$ F2$_{16}$**

Decimal



Zero Page Bit Relative
Addressing mode

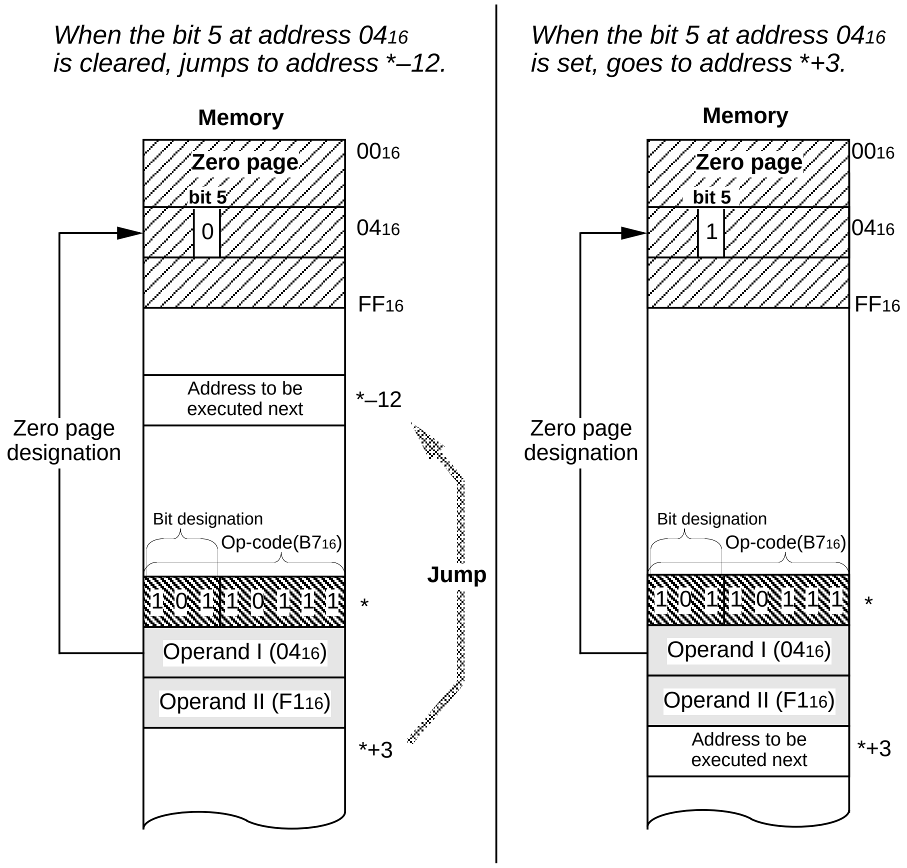
Addressing mode : **Zero Page Bit Relative**

Function : **Specifies the address of a memory location where the next Op-Code is located. The bit position is designated by the high-order three bits of the Op-Code. The address in the Zero Page memory location is determined by using Operand I as low-order byte of the address and 00_{16} as the high-order byte. If the branch condition is satisfied, Operand II and the Program Counter are added. The result of this addition is the address in the memory location. When the branch condition is not satisfied, the next instruction is executed.**

Instructions : **BBC, BBS**

Example : Mnemonic — **ΔBBCΔ5,$04,*–12** (12: Decimal)

Machine language — **$B7_{16}$ 04_{16} $F1_{16}$**



3.2 Instruction Set

The 740 Family has 71 types of instructions. The detailed explanation of the instructions is presented in §3.3. Note that some instructions cannot be used for some products.

3.2.1 Data transfer instructions

These instructions transfer the data between registers, register and memory, and memories. The following are data transfer instructions.

	Instruction	Function
Load	LDA	Load memory value into Accumulator, or memory where is indicated by Index Register X
	LDM	Load immediate value into memory
	LDX	Load memory contents into Index Register X
	LDY	Load memory contents into Index Register Y
Store	STA	Store Accumulator into memory
	STX	Store Index Register X into memory
	STY	Store Index Register Y into memory
Transfer	TAX	Transfer Accumulator to the Index Register X
	TXA	Transfer Index Register X into the Accumulator
	TAY	Transfer Accumulator into the Index Register Y
	TYA	Transfer Index Register Y into the Accumulator
	TSX	Transfer Stack Pointer into the Index Register X
	TXS	Transfer Index Register X into the Stack Pointer
Stack Operation	PHA	Push Accumulator onto the Stack
	PHP	Push Processor Status onto the Stack
	PLA	Pull Accumulator from the Stack
	PLP	Pull Processor Status from the Stack

3.2.2 Operating instruction

The operating instructions include the operations of addition and subtraction, logic, comparison, rotation, and shift.

The operating instructions are as follows:

	Instructions	Contents
Addition & Subtraction	ADC	Add memory contents and C flag to Accumulator or memory where is indicated by Index Register X
	SBC	Subtracts memory contents and C flag's complement from Accumulator or memory where is indicated by Index Register X
	INC	Increment Accumulator or memory contents by 1
	DEC	Decrement Accumulator or memory contents by 1
	INX	Increment Index Register X by 1
	DEX	Decrement Index Register X by 1
	INY	Increment Index Register Y by 1
	DEY	Decrement Index Register Y by 1
Multiplication & Division	MUL(Note)	Multiply Accumulator with memory specified by Zero Page X addressing mode and store high-order byte of result on Stack and low-order byte in Accumulator
	DIV(Note)	Quotient is stored in Accumulator and one's complement of remainder is pushed onto stack
Logical Operation	AND	"AND" memory with Accumulator or memory where is indicated by Index Register X
	ORA	"OR" memory with Accumulator or memory where is indicated by Index Register X
	EOR	"Exclusive-OR" memory with Accumulator or memory where is indicated by Index Register X
	COM	Store one's complement of memory contents to memory
	BIT	"AND" memory with Accumulator (The result is not stored into anywhere.)
	TST	Test whether memory content is "0" or not
Comparison	CMP	Compare memory contents and Accumulator or memory where is indicated by Index Register X
	CPX	Compare memory contents and Index Register X
	CPY	Compare memory contents and Index Register Y
Shift & Rotate	ASL	Shift left one bit (memory contents or Accumulator)
	LSR	Shift right one bit (memory contents or Accumulator)
	ROL	Rotate one bit left with carry (memory contents or Accumulator)
	ROR	Rotate one bit right with carry (memory contents or Accumulator)
	RRF	Rotate four bits right witout carry (memory)

Note: For some products, multiplication and division instructions cannot be used.

3.2.3 Bit managing instructions

The bit managing instructions clear "0" or set "1" designated bits of the Accumulator or memory.

	Instructions	Contents
Bit Managing	CLB	Clear designated bit in the Accumulator or memory
	SEB	Set designated bit in the Accumulator or memory

3.2.4 Flag setting instructions

The flag setting instructions clear "0" or set "1" C, D, I, T and V flags.

	Instructions	Contents	
Flag Setting	CLC	Clear C flag	C flag : Carry Flag
	SEC	Set C flag	
	CLD	Clear D flag	D flag : Decimal Mode Flag
	SED	Set D flag	
	CLI	Clear I flag	I flag : Interrupt Disable Flag
	SEI	Set I flag	
	CLT	Clear T flag	T flag : X Modified Operation Mode Flag
	SET	Set T flag	
	CLV	Clear V flag	V flag : Overflow Flag

3.2.5 Jump, Branch and Return instructions

The jump, branch and return instructions as following are used to change program flow.

	Instructions	Contents	
Jump	JMP	Jump to new location	
	BRA	Jump to new location	
	JSR	Jump to new location saving the current address	
Branch	BBC	Branch when the designated bit in the Accumulator or memory is "0"	
	BBS	Branch when the designated bit in the Accumulator or memory is "1"	
	BCC	Branch when the C Flag is "0"	C flag : Carry Flag
	BCS	Branch when the C Flag is "1"	
	BNE	Branch when the Z Flag is "0"	Z flag : Zero Flag
	BEQ	Branch when the Z Flag is "1"	
	BPL	Branch when the N Flag is "0"	N flag : Negative Flag
	BMI	Branch when the N Flag is "1"	
	BVC	Branch when the V Flag is "0"	V flag : Overflow Flag
	BVS	Branch when the V Flag is "1"	
Return	RTI	Return from interrupt	
	RTS	Return from subroutine	

3.2.6 Interrupt instruction (Break instruction)

This instruction causes a software interrupt.

	Instruction	Contents
Interrupt	BRK	Executes a software interrupt.

3.2.7 Special instructions

These special instructions control the oscillation and the internal clock.

	Instructions	Contents
Special	WIT	Stops the internal clock.
	STP	Stops the oscillation of oscillator.

3.2.8 Other instruction

	Instruction	Contents
Other	NOP	Only advances the program counter.

3.3 Description of instructions

This section presents in detail the 740 Family instructions by arranging mnemonics of instructions alphabetically and dividing each instruction essentially into one page.

The heading of each page is a mnemonic. Operation, explanation and changes of status flags are indicated for each instruction. In addition, assembler coding format, machine code, byte number, and list of cycle numbers for each addressing mode are indicated.

The following are symbols used in this manual:

Symbol	Description	Symbol	Description
A	Accumulator	hh	Address high-order byte data in 0 to 255
Ai	Bit i of Accumulator	ll	Address low-order byte data in 0 to 255
PC	Program Counter	zz	Zero page address data in 0 to 255
PC_L	Low-order byte of Program Counter	nn	Data in 0 to 255
PC_H	High-order byte of Program Counter	i	Data in 0 to 7
PS	Processor Status Register	✻	Contents of the Program Counter
S	Stack Pointer	Δ	Tab or space
X	Index Register X	#	Immediate mode
Y	Index Register Y	\	Special page mode
M	Memory	$	Hexadecimal symbol
Mi	Bit i of memory	+	Addition
C	Carry Flag	–	Subtraction
Z	Zero Flag	×	Multiplication
I	Interrupt Disable Flag	/	Division
D	Decimal Operation Mode Flag	∧	Logical AND
B	Break Flag	∨	Logical OR
T	X Modified Operations Mode Flag	∀	Logical exclusive OR
V	Overflow Flag	( )	Contents of register, memory, etc.
N	Negative Flag	←	Direction of data transfer
REL	Relative address		
BADRS	Break address		

ADC ADD WITH CARRY ADC

Operation : When (T) = 0, (A) ← (A) + (M) + (C)
(T) = 1, (M(X)) ← (M(X)) + (M) + (C)

Function : When T = 0, this instruction adds the contents M, C, and A; and stores the results in A and C.
When T = 1, this instruction adds the contents of M(X), M and C; and stores the results in M(X) and C. When T=1, the contents of A remain unchanged, but the contents of status flags are changed.
M(X) represents the contents of memory where is indicated by X.

Status flag:
N : N is 1 when bit 7 is 1 after the operation; otherwise it is 0.
V : V is 1 when the operation result exceeds +127 or –128; otherwise it is 0.
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the operation result is 0; otherwise it is 0.
C : C is 1 when the result of a binary addition exceeds 255 or when the result of a decimal addition exceeds 99; otherwise it is 0.

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Immediate	ΔADCΔ#$nn	69_{16}, nn_{16}	2	2
Zero page	ΔADCΔ$zz	65_{16}, zz_{16}	2	3
Zero page X	ΔADCΔ$zz,X	75_{16}, zz_{16}	2	4
Absolute	ΔADCΔ$hhll	$6D_{16}$, ll_{16}, hh_{16}	3	4
Absolute X	ΔADCΔ$hhll,X	$7D_{16}$, ll_{16}, hh_{16}	3	5
Absolute Y	ΔADCΔ$hhll,Y	79_{16}, ll_{16}, hh_{16}	3	5
(Indirect X)	ΔADCΔ($zz,X)	61_{16}, zz_{16}	2	6
(Indirect Y)	ΔADCΔ($zz),Y	71_{16}, zz_{16}	2	6

Notes 1: When T=1, add 3 to the cycle number.
2: When ADC instruction is executed in decimal operation mode (D = 1), execute at least one instruction after the ADC instruction before executing a SEC, CLC, or CLD instruction.
In decimal operation mode, the N, V, Z flags are invalid.

AND

LOGICAL **AND**

Operation : When (T) = 0, (A) ← (A) ∧ (M)
(T) = 1, (M(X)) ← (M(X)) ∧ (M)

Function : When T = 0, this instruction transfers the contents of A and M to the ALU which performs a bit-wise AND operation and stores the result back in A.
When T = 1, this instruction transfers the contents M(X) and M to the ALU which performs a bit-wise AND operation and stores the results back in M(X). When T = 1 the contents of A remain unchanged, but status flags are changed.
M(X) represents the contents of memory where is indicated by X.

Status flag:
- **N :** N is 1 when bit 7 is 1 after the operation; otherwise it is 0.
- **V :** No change
- **T :** No change
- **B :** No change
- **I :** No change
- **D :** No change
- **Z :** Z is 1 when the operation result is 0; otherwise it is 0.
- **C :** No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Immediate	ΔANDΔ#$nn	29_{16}, nn_{16}	2	2
Zero page	ΔANDΔ$zz	25_{16}, zz_{16}	2	3
Zero page X	ΔANDΔ$zz,X	35_{16}, zz_{16}	2	4
Absolute	ΔANDΔ$hhll	$2D_{16}$, ll_{16}, hh_{16}	3	4
Absolute X	ΔANDΔ$hhll,X	$3D_{16}$, ll_{16}, hh_{16}	3	5
Absolute Y	ΔANDΔ$hhll,Y	39_{16}, ll_{16}, hh_{16}	3	5
(Indirect X)	ΔANDΔ($zz,X)	21_{16}, zz_{16}	2	6
(Indirect Y)	ΔANDΔ($zz),Y	31_{16}, zz_{16}	2	6

Note: When T = 1, add 3 to a cycle number.

ASL — ARITHMETIC SHIFT LEFT

Operation : C ← b7 | | | | | | | b0 ← 0

Function : This instruction shifts the content of A or M by one bit to the left, with bit 0 always being set to 0 and bit 7 of A or M always being contained in C.

Status flag:
- **N :** N is 1 when bit 7 of A or M is 1 after the operation; otherwise it is 0.
- **V :** No change
- **T :** No change
- **B :** No change
- **I :** No change
- **D :** No change
- **Z :** Z is 1 when the operation result is 0; otherwise it is 0.
- **C :** C is 1 when bit 7 of A or M is 1, before this operation; otherwise it is 0.

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Accumulator	ΔASLΔA	$0A_{16}$	1	2
Zero page	ΔASLΔ$zz	06_{16}, zz_{16}	2	5
Zero page X	ΔASLΔ$zz,X	16_{16}, zz_{16}	2	6
Absolute	ΔASLΔ$hhll	$0E_{16}$, ll_{16}, hh_{16}	3	6
Absolute X	ΔASLΔ$hhll,X	$1E_{16}$, ll_{16}, hh_{16}	3	7

BBC

BRANCH ON BIT CLEAR

Operation : When (Mi) or (Ai) = 0, (PC) ← (PC) + n + REL
(Mi) or (Ai) = 1, (PC) ← (PC) + n
n: If addressing mode is Zero Page Bit Relative, n=3. And if addressing mode is Accumulator Bit Relative, n=2.

Function : This instruction tests the designated bit i of M or A and takes a branch if the bit is 0. The branch address is specified by a relative address. If the bit is 1, next instruction is executed.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Accumulator bit Relative	ΔBBCΔi,A,$hhll	$(20i+13)_{16}$, rr_{16}	2	4
Zero page bit Relative	ΔBBCΔi,$zz,$hhll	$(20i+17)_{16}$, zz_{16}, rr_{16}	3	5

Notes 1: rr_{16}=$hhll–(✻+n). The rr_{16} is a value in a range of –128 to +127.
2: When a branch is executed, add 2 to the cycle number.
3: When executing the BBC instruction after the contents of the interrupt request bit is changed, one instruction or more must be passed before the BBC instruction is executed.

BBS — BRANCH ON BIT SET

Operation : When (Mi) or (Ai) = 1, (PC) ← (PC) + n + REL
(Mi) or (Ai) = 0, (PC) ← (PC) + n
n : If addressing mode is Zero Page Bit Relative, n=3. And if addressing mode is Accumulator Bit Relative, n=2.

Function : This instruction tests the designated bit i of the M or A and takes a branch if the bit is 1. The branch address is specified by a relative address. If the bit is 0, next instruction is executed.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Accumulator bit Relative	ΔBBSΔi,A,$hhll	$(20i+3)_{16}$, rr_{16}	2	4
Zero page bit Relative	ΔBBSΔi,$zz,$hhll	$(20i+7)_{16}$, zz_{16}, rr_{16}	3	5

Notes 1: rr_{16}=$hhll–(✻+n). The rr_{16} is a value in a range of –128 to +127.
2: When a branch is executed, add 2 to the cycle number.
3: When executing the BBS instruction after the contents of the interrupt request bit is changed, one instruction or more must be passed before the BBS instruction is executed.

BCC BRANCH ON CARRY CLEAR BCC

Operation : When (C) = 0, (PC) ← (PC) + 2 + REL
(C) = 1, (PC) ← (PC) + 2

Function : This instruction takes a branch to the appointed address if C is 0. The branch address is specified by a relative address. If C is 1, the next instruction is executed.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Relative	ΔBCCΔ$hhll	90_{16}, rr_{16}	2	2

Notes 1: rr_{16}=$hhll–(✻+2). The rr_{16} is a value in a range of –128 to +127.
2: When a branch is executed, add 2 to the cycle number.

BCS BRANCH ON CARRY SET BCS

Operation : When (C) = 1, (PC) ← (PC) + 2 + REL
(C) = 0, (PC) ← (PC) + 2

Function : This instruction takes a branch to the appointed address if C is 1. The branch address is specified by a relative address. If C is 0, the next instruction is executed.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Relative	∆BCS∆$hhll	$B0_{16}$, rr_{16}	2	2

Notes 1: rr_{16}=$hhll–(✻+2). The rr_{16} is a value in a range of –128 to +127.
2: When a branch is executed, add 2 to the cycle number.

Operation : When (Z) = 1, (PC) ← (PC) + 2 + REL
(Z) = 0, (PC) ← (PC) + 2

Function : This instruction takes a branch to the appointed address when Z is 1. The branch address is specified by a relative address. If Z is 0, the next instruction is executed.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Relative	ΔBEQΔ$hhll	$F0_{16}$,rr_{16}	2	2

Notes 1: rr_{16}=$hhll–(✻+2). The rr_{16} is a value in a range of –128 to +127.
2: When a branch is executed, add 2 to the cycle number.

BIT TEST BIT IN MEMORY WITH ACCUMULATOR BIT

Operation : (A) $\wedge$ (M)

Function : This instruction takes a bit-wise logical AND of A and M contents; however, the contents of A and M are not modified. The contents of N, V, Z are changed, but the contents of A, M remain unchanged.

Status flag:
N : N is 1 when bit 7 of M is 1; otherwise it is 0.
V : V is 1 when bit 6 of M is 1; otherwise it is 0.
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the result of the operation is 0; otherwise Z is 0.
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Zero page	ΔBITΔ$zz	24_{16}, zz_{16}	2	3
Absolute	ΔBITΔ$hhll	$2C_{16}$, ll_{16}, hh_{16}	3	4

Operation : When (N) = 1, (PC) ← (PC) + 2 + REL
(N) = 0, (PC) ← (PC) + 2

Function : This instruction takes a branch to the appointed address when N is 1. The branch address is specified by a relative address. If N is 0, the next instruction is executed.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Relative	∆BMI∆$hhll	30_{16}, rr_{16}	2	2

Notes 1: rr_{16}=$hhll–(✻+2). The rr_{16} is a value in a range of –128 to +127.
2: When a branch is executed, add 2 to the cycle number.

BNE — BRANCH ON NOT EQUAL

Operation : When (Z) = 0, (PC) ← (PC) + 2 + REL
(Z) = 1, (PC) ← (PC) + 2

Function : This instruction takes a branch to the appointed address if Z is 0. The branch address is specified by a relative address. If Z is 1, the next instruction is executed.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Relative	ΔBNEΔ$hhll	$D0_{16}$, rr_{16}	2	2

Notes 1: rr_{16}=$hhll–(✻+2). The rr_{16} is a value in a range of –128 to +127.
2: When a branch is executed, add 2 to the cycle number.

BPL — BRANCH ON RESULT PLUS

Operation : When (N) = 0, (PC) ← (PC) + 2 + REL
(N) = 1, (PC) ← (PC) + 2

Function : This instruction takes a branch to the appointed address if N is 0. The branch address is specified by a relative address. If N is 1, the next instruction is executed.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Relative	ΔBPLΔ$hhll	10_{16}, rr_{16}	2	2

Notes 1: rr_{16}=$hhll–(✻+2). The rr_{16} is a value in a range of –128 to +127.
2: When a branch is executed, add 2 to the cycle number.

Operation : (PC) ← (PC) + 2 + REL

Function : This instruction branches to the appointed address. The branch address is specified by a relative address.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Relative	ΔBRAΔ$hhll	80_{16}, rr_{16}	2	4

Note: rr_{16}=$hhll–(✻+2). The rr_{16} is a value in a range of –128 to +127.

BRK

FORCE **BR**EA**K**

Operation : (B) ← 1
(PC) ← (PC) + 2
(M(S)) ← (PC_H)
(S) ← (S) – 1
(M(S)) ← (PC_L)
(S) ← (S) – 1
(M(S)) ← (PS)
(S) ← (S) – 1
(I) ← 1
(PC) ← BADRS (Note 1)

Function : When the BRK instruction is executed, the CPU pushes the current PC contents onto the stack. The BADRS designated in the interrupt vector table is stored into the PC.

Status flag:
N : No change
V : No change
T : No change
B : 1
I : 1
D : No change
Z : No change
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔBRKΔ	00_{16}	1	7

Notes 1: “BADRS” means a break address.
2: The value of the PC pushed onto the stack by the execution of the BRK instruction is the BRK instruction address plus two. Therefore, the byte following the BRK will not be executed when the value of the PC is returned from the BRK routine.
3: Both after the BRK instruction is executed and after INT is input, the program is branched to the address where is specified by the interrupt vector table. By testing the value of the B Flag in the PS (pushed on the Stack) in the interrupt service routine, the user can determine if the interrupt was caused by the BRK instruction.

BVC — BRANCH ON OVERFLOW CLEAR

Operation : When (V) = 0, (PC) ← (PC) + 2 + REL
(V) = 1, (PC) ← (PC) + 2

Function : This instruction takes a branch to the appointed address if V is 0. The branch address is specified by a relative address. If V is 1, the next instruction is executed.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Relative	ΔBVCΔ$hhll	50_{16}, rr_{16}	2	2

Notes 1: rr_{16}=$hhll–(✻+2). The rr_{16} is a value in a range of –128 to +127.
2: When a branch is executed, add 2 to the cycle number.

Operation : When (V) = 1, (PC) ← (PC) + 2 + REL
(V) = 0, (PC) ← (PC) + 2

Function : This instruction takes a branch to the appointed address when V is 1. The branch address is specified by a relative address. When V is 0, the next instruction is executed.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Relative	ΔBVSΔ$hhll	70_{16}, rr_{16}	2	2

Notes 1: rr_{16}=$hhll–(✻+2). The rr_{16} is a value in a range of –128 to +127.
2: When a branch is executed, add 2 to the cycle number.

CLB CLEAR BIT CLB

Operation : (Ai) ← 0, or
(Mi) ← 0

Function : This instruction clears the designated bit i of A or M.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Accumulator bit	ΔCLBΔi,A	$(20i+1B)_{16}$	1	2
Zero page bit	ΔCLBΔi,$zz	$(20i+1F)_{16}$, ZZ_{16}	2	5

CLC — CLEAR CARRY FLAG

Operation : (C) ← 0

Function : This instruction clears C.

Status flag:
N : No change
V : No change
T : No change
B : No change
I : No change
D : No change
Z : No change
C : 0

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔCLC	18_{16}	1	2

CLD — CLEAR DECIMAL MODE

CLD

Operation : (D) ← 0

Function : This instruction clears D.

Status flag:
N : No change
V : No change
T : No change
B : No change
I : No change
D : 0
Z : No change
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔCLD	$D8_{16}$	1	2

CLI

CLEAR INTERRUPT DISABLE STATUS

Operation : (I) ← 0

Function : This instruction clears I.

Status flag:
N : No change
V : No change
T : No change
B : No change
I : 0
D : No change
Z : No change
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔCLI	58_{16}	1	2

CLT

CLEAR TRANSFER FLAG

CLT

Operation : (T) ← 0

Function : This instruction clears T.

Status flag:
N : No change
V : No change
T : 0
B : No change
I : No change
D : No change
Z : No change
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔCLT	12_{16}	1	2

CLV CLEAR OVERFLOW FLAG CLV

Operation : (V) ← 0

Function : This instruction clears V.

Status flag
N : No change
V : 0
T : No change
B : No change
I : No change
D : No change
Z : No change
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔCLV	$B8_{16}$	1	2

CMP

COMPARE

Operation : When (T) = 0, (A) – (M)
(T) = 1, (M(X)) – (M)

Function : When T = 0, this instruction subtracts the contents of M from the contents of A. The result is not stored and the contents of A or M are not modified.
When T = 1, the CMP subtracts the contents of M from the contents of M(X). The result is not stored and the contents of M(X), M, and A are not modified.
M(X) represents the contents of memory where is indicated by X.

Status flag:
- **N :** N is 1 when bit 7 of the operation result is 1 after the operation; otherwise N is 0.
- **V :** No change
- **T :** No change
- **B :** No change
- **I :** No change
- **D :** No change
- **Z :** Z is 1 when the operation result is 0; otherwise Z is 0.
- **C :** C is 1 when the subtracted result is equal to or greater than 0; otherwise C is 0.

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Immediate	ΔCMPΔ#$nn	$C9_{16}$, nn_{16}	2	2
Zero page	ΔCMPΔ$zz	$C5_{16}$, zz_{16}	2	3
Zero page X	ΔCMPΔ$zz,X	$D5_{16}$, zz_{16}	2	4
Absolute	ΔCMPΔ$hhll	CD_{16}, ll_{16}, hh_{16}	3	4
Absolute X	ΔCMPΔ$hhll,X	DD_{16}, ll_{16}, hh_{16}	3	5
Absolute Y	ΔCMPΔ$hhll,Y	$D9_{16}$, ll_{16}, hh_{16}	3	5
(Indirect X)	ΔCMPΔ($zz,X)	$C1_{16}$, zz_{16}	2	6
(Indirect Y)	ΔCMPΔ($zz),Y	$D1_{16}$, zz_{16}	2	6

Note: When T=1, add 1 to the cycle number.

COM COMPLEMENT COM

Operation : (M) ← $\overline{(M)}$

Function : This instruction takes the one's complement of the contents of M and stores the result in M.

Status flag:
N : N is 1 when bit 7 of the M is 1 after the operation; otherwise N is 0.
V : No change
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the operation result is 0; otherwise Z is 0.
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Zero page	ΔCOMΔ$zz	44_{16}, zz_{16}	2	5

CPX COMPARE MEMORY AND INDEX REGISTER X CPX

Operation : (X) – (M)

Function : This instruction subtracts the contents of M from the contents of X. The result is not stored and the contents of X and M are not modified.

Status flag:

- **N :** N is 1 when bit 7 of the operation result is 1 after the operation; otherwise N is 0.
- **V :** No change
- **T :** No change
- **B :** No change
- **I :** No change
- **D :** No change
- **Z :** Z is 1 when the operation result is 0; otherwise Z is 0.
- **C :** C is 1 when the subtracted result is equal to or greater than 0; otherwise C is 0.

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Immediate	ΔCPXΔ#$nn	$E0_{16}$, nn_{16}	2	2
Zero page	ΔCPXΔ$zz	$E4_{16}$, zz_{16}	2	3
Absolute	ΔCPXΔ$hhll	EC_{16}, ll_{16}, hh_{16}	3	4

CPY COMPARE MEMORY AND INDEX REGISTER Y

Operation : (Y) – (M)

Function : This instruction subtracts the contents of M from the contents of Y. The result is not stored and the contents of Y and M are not modified.

Status flag:
- **N :** N is 1 when bit 7 of the operation result is 1 after the operation; otherwise N is 0.
- **V :** No change
- **T :** No change
- **B :** No change
- **I :** No change
- **D :** No change
- **Z :** Z is 1 when the operation result is 0; otherwise Z is 0.
- **C :** C is 1 when the subtracted result is equal to or greater than 0; otherwise C is 0.

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Immediate	ΔCPYΔ#$nn	$C0_{16}$, nn_{16}	2	2
Zero page	ΔCPYΔ$zz	$C4_{16}$, zz_{16}	2	3
Absolute	ΔCPYΔ$hhll	CC_{16}, ll_{16}, hh_{16}	3	4

DEC — DECREMENT BY ONE

Operation : (A) ← (A) − 1, or
(M) ← (M) − 1

Function : This instruction subtracts 1 from the contents of A or M.

Status flag:
N : N is 1 when bit 7 is 1 after the addition; otherwise N is 0.
V : No change
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the operation result is 0; otherwise Z is 0.
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Accumulator	ΔDECΔA	$1A_{16}$	1	2
Zero page	ΔDECΔ$zz	$C6_{16}$, zz_{16}	2	5
Zero page X	ΔDECΔ$zz,X	$D6_{16}$, zz_{16}	2	6
Absolute	ΔDECΔ$hhll	CE_{16}, ll_{16}, hh_{16}	3	6
Absolute X	ΔDECΔ$hhll,X	DE_{16}, ll_{16}, hh_{16}	3	7

DEX DECREMENT INDEX REGISTER X BY ONE

DEX

Operation : (X) ← (X) – 1

Function : This instruction subtracts one from the current contents of X.

Status flag:
- **N :** N is 1 when bit 7 is 1 after the operation; otherwise N is 0.
- **V :** No change
- **T :** No change
- **B :** No change
- **I :** No change
- **D :** No change
- **Z :** Z is 1 when the operation result is 0; otherwise Z is 0.
- **C :** No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔDEX	CA_{16}	1	2

DEY DECREMENT INDEX REGISTER Y BY ONE DEY

Operation : (Y) ← (Y) – 1

Function : This instruction subtracts one from the current contents of Y.

Status flag:
N : N is 1 when bit 7 is 1 after the operation; otherwise N is 0.
V : No change
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the operation result is 0; otherwise Z is 0.
C : No change

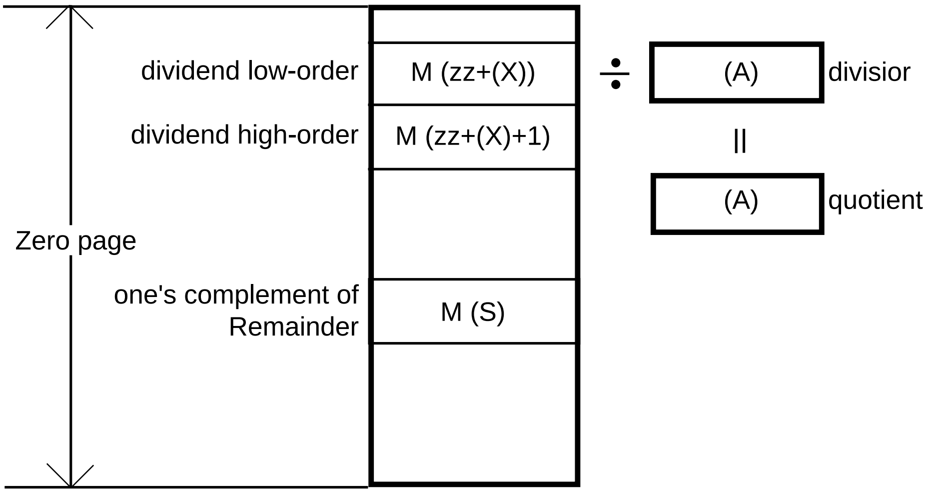
Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	∆DEY	88_{16}	1	2

DIV

DIVIDE MEMORY BY ACCUMULATOR

Operation : (A) ← (M(zz+(X)+1),M(zz+(X)) / (A)
M(S) ← one's complement of Remainder
(S) ← (S) – 1

Function : Divides the 16-bit data in M(zz+(X)) (low-order byte) and M(zz+(X)+1) (high-order byte) by the contents of A. The quotient is stored in A and the one's complement of the remainder is pushed onto the stack.



Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Zero page X	∆DIV∆$zz,X	$E2_{16}$, zz_{16}	2	16

Notes 1: The quotient's overflow and zero division can not be detected. Check the quotient's overflow and zero division by software before DIV instruction is executed. This instruction changes the Stack Pointer and the contents of the Accumulator.

2: The DIV instruction can not be used for some products.

3: The DIV instruction is not affected by T and D flags.

EOR

EXCLUSIVE OR MEMORY WITH ACCUMULATOR

Operation : When (T) = 0, (A) ← (A) ∀ (M)
(T) = 1, (M(X)) ← (M(X)) ∀ (M)

Function : When T = 0, this instruction transfers the contents of the M and A to the ALU which performs a bit-wise Exclusive OR, and stores the result in A.
When T = 1, the contents of M(X) and M are transferred to the ALU, which performs a bit-wise Exclusive OR and stores the results in M(X). The contents of A remain unchanged, but status flags are changed.
M(X) represents the contents of memory where is indicated by X.

Status flag:
N : N is 1 when bit 7 is 1 after the operation; otherwise N is 0.
V : No change
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the operation result is 0; otherwise Z is 0.
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Immediate	ΔEORΔ#$nn	49_{16}, nn_{16}	2	2
Zero page	ΔEORΔ$zz	45_{16}, zz_{16}	2	3
Zero page X	ΔEORΔ$zz,X	55_{16}, zz_{16}	2	4
Absolute	ΔEORΔ$hhll	$4D_{16}$, ll_{16}, hh_{16}	3	4
Absolute X	ΔEORΔ$hhll,X	$5D_{16}$, ll_{16}, hh_{16}	3	5
Absolute Y	ΔEORΔ$hhll,Y	59_{16}, ll_{16}, hh_{16}	3	5
(Indirect X)	ΔEORΔ($zz,X)	41_{16}, zz_{16}	2	6
(Indirect Y)	ΔEORΔ($zz),Y	51_{16}, zz_{16}	2	6

Note: When T=1, add 3 to the cycle number.

INC

INCREMENT BY ONE

Operation : (A) ← (A) + 1, or
(M) ← (M) + 1

Function : This instruction adds one to the contents of A or M.

Status flag:
- **N :** N is 1 when bit 7 is 1 after the operation; otherwise N is 0.
- **V :** No change
- **T :** No change
- **B :** No change
- **I :** No change
- **D :** No change
- **Z :** Z is 1 when the operation result is 0; otherwise Z is 0.
- **C :** No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Accumulator	∆INC∆A	$3A_{16}$	1	2
Zero page	∆INC∆$zz	$E6_{16}$, zz_{16}	2	5
Zero page X	∆INC∆$zz,X	$F6_{16}$, zz_{16}	2	6
Absolute	∆INC∆$hhll	EE_{16}, ll_{16}, hh_{16}	3	6
Absolute X	∆INC∆$hhll,X	FE_{16}, ll_{16}, hh_{16}	3	7

INX — INCREMENT INDEX REGISTER X BY ONE — INX

Operation : (X) ← (X) + 1

Function : This instruction adds one to the contents of X.

Status flag:
- **N :** N is 1 when bit 7 is 1 after the operation; otherwise N is 0.
- **V :** No change
- **T :** No change
- **B :** No change
- **I :** No change
- **D :** No change
- **Z :** Z is 1 when the operation result is 0; otherwise Z is 0.
- **C :** No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	∆INX	$E8_{16}$	1	2

Operation : (Y) ← (Y) + 1

Function : This instruction adds one to the contents of Y.

Status flag:
N : N is 1 when bit 7 is 1 after the operation; otherwise N is 0.
V : No change
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the operation result is 0; otherwise Z is 0.
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔINY	$C8_{16}$	1	2

Operation : When addressing mode is
(a) Absolute, then
$(PC) \leftarrow hhll$
(b) Indirect Absolute, then
$(PC_L) \leftarrow (hhll)$
$(PC_H) \leftarrow (hhll+1)$
(c) Zero page Indirect Absolute, then
$(PC_L) \leftarrow (zz)$
$(PC_H) \leftarrow (zz+1)$

Function : This instruction jumps to the address designated by the following three addressing modes:
Absolute
Indirect Absolute
Zero Page Indirect Absolute

Status flag: No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Absolute	ΔJMPΔ$hhll	$4C_{16}$,ll_{16},hh_{16}	3	3
Indirect Absolute	ΔJMPΔ($hhll)	$6C_{16}$,ll_{16},hh_{16}	3	5
Zero Page Indirect	ΔJMPΔ($zz)	$B2_{16}$,zz_{16}	2	4

Note: The page's last address (address $XXFF_{16}$) cannot be specified for the indirect designation address; in other words, JMP ($XXFF) cannot be executed.

Operation : (M(S)) ← (PC_H)
(S) ← (S) − 1
(M(S)) ← (PC_L)
(S) ← (S) − 1
After the above operations, if the addressing mode is
(a) Absolute, then
(PC) ← hhll
(b) Special page, then
(PC_L) ← ll
(PC_H) ← FF_{16}
(c) Zero page Indirect, then
(PC_L) ← (zz)
(PC_H) ← (zz+1)

Function : This instruction stores the contents of the PC in the stack, then jumps to the address designated by the following addressing modes:
Absolute
Special Page
Zero Page Indirect Absolute

Status flag: No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Absolute	ΔJSRΔ$hhll	20_{16}, ll_{16}, hh_{16}	3	6
Special page	ΔJSRΔ\$hhll (Note)	22_{16}, ll_{16}	2	5
Zero page Indirect	ΔJSRΔ($zz)	02_{16}, zz_{16}	2	7

(Note) "\" ($5C_{16}$ of the ASCII code) denotes special page. hh_{16} must be FF_{16} in the special page addressing mode.

Operation : When (T) = 0, (A) ← (M)
(T) = 1, (M(X)) ← (M)

Function : When T = 0, this instruction transfers the contents of M to A. When T = 1, this instruction transfers the contents of M to (M(X)). The contents of A remain unchanged, but status flags are changed.
M(X) represents the contents of memory where is indicated by X.

Status flag:
N : N is 1 when bit 7 is 1 after the operation; otherwise N is 0.
V : No change
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the operation result is 0; otherwise Z is 0.
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Immediate	ΔLDAΔ#$nn	$A9_{16}$, nn_{16}	2	2
Zero page	ΔLDAΔ$zz	$A5_{16}$, zz_{16}	2	3
Zero page X	ΔLDAΔ$zz,X	$B5_{16}$, zz_{16}	2	4
Absolute	ΔLDAΔ$hhll	AD_{16}, ll_{16}, hh_{16}	3	4
Absolute X	ΔLDAΔ$hhll,X	BD_{16}, ll_{16}, hh_{16}	3	5
Absolute Y	ΔLDAΔ$hhll,Y	$B9_{16}$, ll_{16}, hh_{16}	3	5
(Indirect X)	ΔLDAΔ($zz,X)	$A1_{16}$, zz_{16}	2	6
(Indirect Y)	ΔLDAΔ($zz),Y	$B1_{16}$, zz_{16}	2	6

Note: When T = 1, add 2 to the cycle number.

LDM

LOAD IMMEDIATE DATA TO MEMORY

Operation : (M) ← nn

Function : This instruction loads the immediate value in M.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Zero page	ΔLDMΔ#$nn,$zz	$3C_{16}$, nn_{16}, zz_{16}	3	4

LDX LOAD INDEX REGISTER X FROM MEMORY LDX

Operation : (X) ← (M)

Function : This instruction loads the contents of M in X.

Status flag:

N : N is 1 when bit 7 is 1 after the operation; otherwise N is 0.
V : No change
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the operation result is 0; otherwise Z is 0.
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Immediate	ΔLDXΔ#$nn	$A2_{16}$, nn_{16}	2	2
Zero page	ΔLDXΔ$zz	$A6_{16}$, zz_{16}	2	3
Zero page Y	ΔLDXΔ$zz,Y	$B6_{16}$, zz_{16}	2	4
Absolute	ΔLDXΔ$hhll	AE_{16}, ll_{16}, hh_{16}	3	4
Absolute Y	ΔLDXΔ$hhll,Y	BE_{16}, ll_{16}, hh_{16}	3	5

LDY LOAD INDEX REGISTER Y FROM MEMORY LDY

Operation : (Y) ← (M)

Function : This instruction loads the contents of M in Y.

Status flag:
N : N is 1 when bit 7 is 1 after the operation; otherwise N is 0.
V : No change
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the operation result is 0; otherwise Z is 0.
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Immediate	ΔLDYΔ#$nn	$A0_{16}$, nn_{16}	2	2
Zero page	ΔLDYΔ$zz	$A4_{16}$, zz_{16}	2	3
Zero page X	ΔLDYΔ$zz,X	$B4_{16}$, zz_{16}	2	4
Absolute	ΔLDYΔ$hhll	AC_{16}, ll_{16}, hh_{16}	3	4
Absolute X	ΔLDYΔ$hhll,X	BC_{16}, ll_{16}, hh_{16}	3	5

LSR

LOGICAL SHIFT RIGHT

LSR

Operation : 0 → | b7 | | | | | | | b0 | → | C |

Function : This instruction shifts either A or M one bit to the right such that bit 7 of the result always is set to 0, and the bit 0 is stored in C.

Status flag:
N : 0
V : No change
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the operation result is 0; otherwise Z is 0.
C : C is 1 when the bit 0 of either the A or the M before the operation is 1; otherwise C is 0.

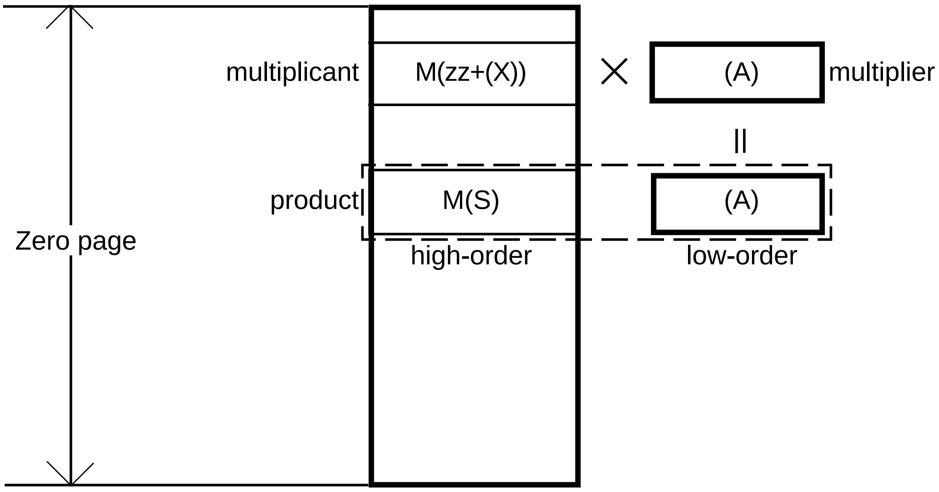
Addressing mode	Statement	Machine codes	Byte number	Cycle number
Accumulator	ΔLSRΔA	$4A_{16}$	1	2
Zero page	ΔLSRΔ$zz	46_{16}, zz_{16}	2	5
Zero page X	ΔLSRΔ$zz,X	56_{16}, zz_{16}	2	6
Absolute	ΔLSRΔ$hhll	$4E_{16}$, ll_{16}, hh_{16}	3	6
Absolute X	ΔLSRΔ$hhll,X	$5E_{16}$, ll_{16}, hh_{16}	3	7

MUL

MULTIPLY ACCUMULATOR AND MEMORY

Operation : M(S) • (A) ← (A) × M(zz+(X))
(S) ← (S) − 1

Function : Multiplies Accumulator with the memory specified by the Zero Page X addressing mode and stores the high-order byte of the result on the Stack and the low-order byte in A.



Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Zero page X	ΔMULΔ$zz,X	62_{16}, zz_{16}	2	15

Notes 1: This instruction changes the contents of S and A.
2: The MUL instruction cannot be used for some products.
3: The MUL instruction is not affected by T and D flags.

NOP

NO OPERATION

Operation : (PC) ← (PC) + 1

Function : This instruction adds one to the PC but does no other operation.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle Number
Implied	ΔNOP	EA_{16}	1	2

ORA OR MEMORY WITH ACCUMULATOR ORA

Operation : When (T) = 0, (A) ← (A) ∨ (M)
(T) = 1, (M(X)) ← (M(X)) ∨ (M)

Function : When T = 0, this instruction transfers the contents of A and M to the ALU which performs a bit-wise "OR", and stores the result in A.
When T = 1, this instruction transfers the contents of M(X) and the M to the ALU which performs a bit-wise OR, and stores the result in M(X). The contents of A remain unchanged, but status flags are changed.
M(X) represents the contents of memory where is indicated by X.

Status flag:
N : N is "1" when bit 7 is 1 after the operation; otherwise N is 0.
V : No change
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the execution result is 0; otherwise Z is 0.
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Immediate	∆ORA∆#$nn	09_{16}, nn_{16}	2	2
Zero page	∆ORA∆$zz	05_{16}, zz_{16}	2	3
Zero page X	∆ORA∆$zz,X	15_{16}, zz_{16}	2	4
Absolute	∆ORA∆$hhll	$0D_{16}$, ll_{16}, hh_{16}	3	4
Absolute X	∆ORA∆$hhll,X	$1D_{16}$, ll_{16}, hh_{16}	3	5
Absolute Y	∆ORA∆$hhll,Y	19_{16}, ll_{16}, hh_{16}	3	5
(Indirect X)	∆ORA∆($zz,X)	01_{16}, zz_{16}	2	6
(Indirect Y)	∆ORA∆($zz),Y	11_{16}, zz_{16}	2	6

Note: When T=1, add 3 to the cycle number.

Operation : (M(S)) ← (A)
(S) ← (S) − 1

Function : This instruction pushes the contents of A to the memory location designated by S, and decrements the contents of S by one.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔPHA	48_{16}	1	3

PHP — PUSH PROCESSOR STATUS ON STACK — PHP

Operation : (M(S)) ← (PS)
(S) ← (S) – 1

Function : This instruction pushes the contents of PS to the memory location designated by S and decrements the contents of S by one.

Status flag: No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔPHP	08_{16}	1	3

Operation : (S) ← (S) + 1
(A) ← (M(S))

Function : This instruction increments S by one and stores the contents of the memory designated by S in A.

Status flag:
N : N is 1 when bit 7 is 1 after the operation ; otherwise N is 0.
V : No change
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the operation result is 0; otherwise Z is 0.
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔPLA	68_{16}	1	4

Note: A NOP instruction should be executed after every PLP instruction.

Operation : (S) ← (S) + 1
(PS) ← (M(S))

Function : This instruction increments S by one and stores the contents of the memory location designated by S in PS.

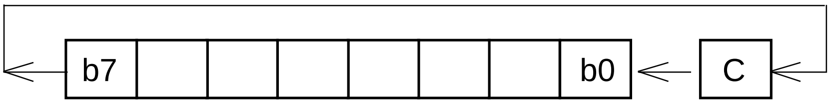
Status flag : Value returns to the original one that was pushed in the stack.

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔPLP	28_{16}	1	4

Note: A NOP instruction should be executed after every PLP instruction.

ROL — ROTATE ONE BIT LEFT

Operation :



Function : This instruction shifts either A or M one bit left through C. C is stored in bit 0 and bit 7 is stored in C.

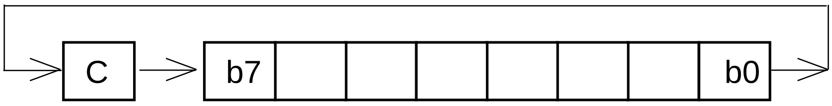
Status flag:

N : N is 1 when bit 6 is 1 before the operation; otherwise N is 0.
V: No change
T: No change
B: No change
I: No change
D: No change
Z: Z is 1 when the operation result is 0; otherwise Z is 0.
C: C is 1 when bit 7 is 1 before the operation; otherwise C is 0.

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Accumulator	ΔROLΔA	$2A_{16}$	1	2
Zero page	ΔROLΔ$zz	26_{16}, zz_{16}	2	5
Zero page X	ΔROLΔ$zz,X	36_{16}, zz_{16}	2	6
Absolute	ΔROLΔ$hhll	$2E_{16}$, ll_{16}, hh_{16}	3	6
Absolute X	ΔROLΔ$hhll,X	$3E_{16}$, ll_{16}, hh_{16}	3	7

ROR — ROTATE ONE BIT RIGHT — ROR

Operation :



Function : This instruction shifts either A or M one bit right through C. C is stored in bit 7 and bit 0 is stored in C.

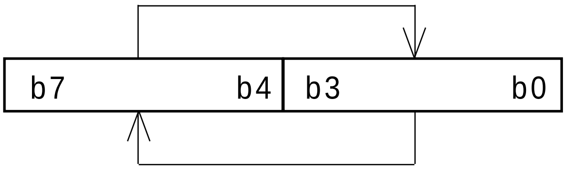
Status flag:

- **N :** N is 1 when C is 1 before the operation; otherwise N is 0.
- **V:** No change
- **T:** No change
- **B:** No change
- **I:** No change
- **D:** No change
- **Z:** Z is 1 when the operation result is 0; otherwise Z is 0.
- **C:** C is 1 when bit 0 is 1 before the operation; otherwise C is 0.

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Accumulator	ΔRORΔA	$6A_{16}$	1	2
Zero page	ΔRORΔ$zz	66_{16}, zz_{16}	2	5
Zero page X	ΔRORΔ$zz,X	76_{16}, zz_{16}	2	6
Absolute	ΔRORΔ$hhll	$6E_{16}$, ll_{16}, hh_{16}	3	6
Absolute X	ΔRORΔ$hhll,X	$7E_{16}$, ll_{16}, hh_{16}	3	7

RRF ROTATE RIGHT OF FOUR BITS RRF

Operation :



Function : This instruction rotates 4 bits of the M content to the right.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Zero page	ΔRRFΔ$zz	82_{16}, zz_{16}	2	8

RETURN FROM INTERRUPT

Operation : (S) ← (S) + 1
(PS) ← (M(S))
(S) ← (S) + 1
(PC_L) ← (M(S))
(S) ← (S) + 1
(PC_H) ← (M(S))

Function : This instruction increments S by one, and stores the contents of the memory location designated by S in PS. S is again incremented by one and stores the contents of the memory location designated by S in PC_L. S is again incremented by one and stores the contents of memory location designated by S in PC_H.

Status flag : Value returns to the original one that was pushed in the stack.

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔRTI	40_{16}	1	6

Operation : (S) ← (S) + 1
(PC$_L$) ← (M(S))
(S) ← (S) + 1
(PC$_H$) ← (M(S))
(PC) ← (PC) + 1

Function : This instruction increments S by one and stores the contents of the memory location designated by S in PC$_L$. S is again incremented by one and the contents of the memory location is stored in PC$_H$. PC is incremented by 1.

Status flag: No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔRTS	60_{16}	1	6

SBC SUBTRACT WITH CARRY SBC

Operation : When (T) = 0, (A) ← (A) – (M) – $\overline{(C)}$
(T) = 1, (M(X)) ← (M(X)) – (M) – $\overline{(C)}$

Function : When T = 0, this instruction subtracts the value of M and the complement of C from A, and stores the results in A and C.
When T = 1, the instruction subtracts the contents of M and the complement of C from the contents of M(X), and stores the results in M(X) and C.
A remain unchanged, but status flag are changed.
M(X) represents the contents of memory where is indicated by X.

Status flag:
- **N :** N is 1 when bit 7 is 1 after the operation; otherwise N is 0.
- **V :** V is 1 when the operation result exceeds +127 or |–128|; otherwise V is 0.
- **T :** No change
- **B :** No change
- **I :** No change
- **D :** No change
- **Z :** Z is 1 when the operation result is 0; otherwise Z is 0.
- **C :** C is 1 when the subtracted result is equal to or greater than 0; otherwise C is 0.

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Immediate	ΔSBCΔ#$nn	$E9_{16}$, nn_{16}	2	2
Zero page	ΔSBCΔ$zz	$E5_{16}$, zz_{16}	2	3
Zero page X	ΔSBCΔ$zz,X	$F5_{16}$, zz_{16}	2	4
Absolute	ΔSBCΔ$hhll	ED_{16}, ll_{16}, hh_{16}	3	4
Absolute X	ΔSBCΔ$hhll,X	FD_{16}, ll_{16}, hh_{16}	3	5
Absolute Y	ΔSBCΔ$hhll,Y	$F9_{16}$, ll_{16}, hh_{16}	3	5
(Indirect X)	ΔSBCΔ($zz,X)	$E1_{16}$, zz_{16}	2	6
(Indirect Y)	ΔSBCΔ($zz),Y	$F1_{16}$, zz_{16}	2	6

Notes 1: When T=1, add 3 to the cycle number.
2: When SBC instruction is executed in decimal operation mode (D = 1), execute at least one instruction after the SBC instruction before executing a SEC, CLC, or CLD instruction.
In decimal operation mode, the N, V, Z flags are invalid.

SEB — SET BIT

Operation : (Ai) ← 1, or
(Mi) ← 1

Function : This instruction sets the designated bit i of A or M.

Status flag: No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Accumulator bit	ΔSEBΔi,A	$(20i+B)_{16}$	1	2
Zero page bit	ΔSEBΔi,$zz	$(20i+F)_{16}$, zz_{16}	2	5

SEC — SET CARRY FLAG

Operation : (C) ← 1

Function : This instruction sets C.

Status flag:
N : No change
V : No change
T : No change
B : No change
I : No change
D : No change
Z : No change
C : 1

Addressing mode	Statement	Machine code	Byte number	Cycle number
Implied	ΔSEC	38_{16}	1	2

Operation : (D) ← 1

Function : This instruction set D.

Status flag:
N : No change
V : No change
T : No change
B : No change
I : No change
D : 1
Z : No change
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔSED	$F8_{16}$	1	2

SEI

SET INTERRUPT DISABLE FLAG

Operation : (I) ← 1

Function : This instruction sets I.

Status flag:
N : No change
V : No change
T : No change
B : No change
I : 1
D : No change
Z : No change
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	∆SEI	78_{16}	1	2

Operation : (T) ← 1

Function : This instruction sets T.

Status flag:
N : No change
V : No change
T : 1
B : No change
I : No change
D : No change
Z : No change
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔSET	32_{16}	1	2

STA — STORE ACCUMULATOR IN MEMORY

Operation : (M) ← (A)

Function : This instruction stores the contents of A in M.
The contents of A does not change.

Status flag: No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Zero page	ΔSTAΔ$zz	85_{16}, zz_{16}	2	4
Zero page X	ΔSTAΔ$zz,X	95_{16}, zz_{16}	2	5
Absolute	ΔSTAΔ$hhll	$8D_{16}$, ll_{16}, hh_{16}	3	5
Absolute X	ΔSTAΔ$hhll,X	$9D_{16}$, ll_{16}, hh_{16}	3	6
Absolute Y	ΔSTAΔ$hhll,Y	99_{16}, ll_{16}, hh_{16}	3	6
(Indirect X)	ΔSTAΔ($zz,X)	81_{16}, zz_{16}	2	7
(Indirect Y)	ΔSTAΔ($zz),Y	91_{16}, zz_{16}	2	7

STP

STOP

Operation : CPU ← Stand-by state (Oscillation stopped)

Function : This instruction resets the oscillation control F/F and the oscillation stops. Reset or interrupt input is needed to wake up from this mode.

Status flag: No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔSTP	42_{16}	1	2

Note: If the STP instruction is disabled the cycle number will be 2 (same in operation as NOP). However, disabling this instruction is an optional feature; therefore, consult the specifications for the particular chip in question.

Operation : (M) ← (X)

Function : This instruction stores the contents of X in M. The contents of X does not change.

Status flag: No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Zero page	ΔSTXΔ$zz	86_{16}, zz_{16}	2	4
Zero page Y	ΔSTXΔ$zz,Y	96_{16}, zz_{16}	2	5
Absolute	ΔSTXΔ$hhll	$8E_{16}$, ll_{16}, hh_{16}	3	5

STY — STORE INDEX REGISTER Y IN MEMORY

STY

Operation : (M) ← (Y)

Function : This instruction stores the contents of Y in M.
The contents of Y does not change.

Status flag: No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Zero page	ΔSTYΔ$zz	84_{16}, zz_{16}	2	4
Zero page X	ΔSTYΔ$zz,X	94_{16}, zz_{16}	2	5
Absolute	ΔSTYΔ$hhll	$8C_{16}$, ll_{16}, hh_{16}	3	5

TAX
TRANSFER ACCUMULATOR TO INDEX REGISTER X

Operation : (X) ← (A)

Function : This instruction stores the contents of A in X. The contents of A does not change.

Status flag:
N : N is 1 when bit 7 is 1 after the operation; otherwise N is 0.
V : No change
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the operation result is 0; otherwise Z is 0.
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	∆TAX	AA_{16}	1	2

TAY

TRANSFER ACCUMULATOR TO INDEX REGISTER Y

Operation : (Y) ← (A)

Function : This instruction stores the contents of A in Y. The contents of A does not change.

Status flag:
N : N is 1 when bit 7 is 1 after the operation; otherwise N is 0.
V : No change
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the operation result is 0; otherwise Z is 0.
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔTAY	$A8_{16}$	1	2

Operation : (M) = 0 ?

Function : This instruction tests whether the contents of M are “0” or not and modifies the N and Z.

Status flag:
N : N is 1 when bit 7 of M is 1; otherwise N is 0.
V : No change
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the M content is 0; otherwise Z is 0.
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Zero page	ΔTSTΔ$zz	64_{16}, zz_{16}	2	3

TSX TSX

TRANSFER **S**TACK POINTER TO INDEX REGISTER **X**

Operation : (X) ← (S)

Function : This instruction transfers the contents of S in X.

Status flag:
N : N is 1 when bit 7 is 1 after the operation; otherwise N is 0.
V : No change
T : No change
B : No change
I : No change
D : No change
Z : Z is 1 when the operation result is 0; otherwise Z is 0.
C : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔTSX	BA_{16}	1	2

TXA TXA

TRANSFER INDEX REGISTER X TO ACCUMULATOR

Operation : (A) ← (X)

Function : This instruction stores the contents of X in A.

Status flag:
- **N :** N is 1 when bit 7 is 1 after the operation; otherwise N is 0.
- **V:** No change
- **T:** No change
- **B:** No change
- **I:** No change
- **D:** No change
- **Z:** Z is 1 when the operation result is 0; otherwise Z is 0.
- **C:** No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔTXA	$8A_{16}$	1	2

TRANSFER INDEX REGISTER X TO STACK POINTER

Operation : (S) ← (X)

Function : This instruction stores the contents of X in S.

Status flag No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔTXS	$9A_{16}$	1	2

TYA TYA

TRANSFER INDEX REGISTER Y TO ACCUMULATOR

Operation : (A) ← (Y)

Function : This instruction stores the contents of Y in A.

Status flag:
N : N is 1 when bit 7 is 1 after the operation; otherwise N is 0.
V: No change
T: No change
B: No change
I: No change
D: No change
Z: Z is 1 when the operation result is 0; otherwise Z is 0.
C: No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔTYA	98_{16}	1	2

WAIT

Operation : CPU ← Wait state

Function : The WIT instruction stops the internal clock but the oscillation of the oscillation circuit is not stopped. Reset or interrupt input is needed to wake up from this mode.

Status flag : No change

Addressing mode	Statement	Machine codes	Byte number	Cycle number
Implied	ΔWIT	$C2_{16}$	1	2

3.4 Instructions Related to Interrupt Handling and Subroutine Processing

3.4.1 Instructions Related to Interrupt Handling

When an interrupt is accepted, the contents of the processor status register are pushed onto the memory location indicated by the stack pointer. There is therefore no need to execute the PHP instruction.

If it is necessary to save the contents of the accumulator, the PHA instruction should be executed within an interrupt routine (before any instruction that manipulates the accumulator).

Whenever a stack operation instruction such as PHA is executed within an interrupt routine, make sure that instructions such as PLA that affect the stack operation instruction are also executed within the same interrupt routine.

Execute the RTI instruction to return from the interrupt routine.

3.4.2 Instructions Related to Interrupt Control

The factors that control an interrupt are the interrupt disable flag (I) as well as the interrupt enable bit and request bit corresponding to the interrupt source. (This does not apply to software interrupts triggered by the BRK instruction.)

(1) Disabling Interrupts

An interrupt may be disabled by setting the interrupt disable flag (I) to “1” using the SEI instruction or by using an instruction such as LDM or CLB (a variety of other instructions can be used as well) to clear the interrupt enable bit to “0”.

(2) Enabling Interrupts

An interrupt may be enabled by setting the interrupt enable bit to “1” using an instruction such as LDM or SEB, and by using the CLI instruction to clear the interrupt disable flag (I) to “0”.

(3) Clearing Interrupt Requests

When an interrupt is generated, the interrupt request bit corresponding to the interrupt source is set to “1” automatically. The interrupt request bit is cleared to “0” when the interrupt is accepted. Therefore, there is no need to clear the interrupt request bit (within an interrupt routine) by means of a user program.

If interrupt generation occurs while an interrupt is disabled, the interrupt request bit is set to “1”. If, under this condition, the interrupt is subsequently enabled (the interrupt disable flag (I) is cleared to “0” and the interrupt enable bit is set to “1”), the interrupt is accepted. To prevent an interrupt from being accepted in such a case, use an instruction such as LDM or CLB to clear the interrupt request bit to “0” before enabling the interrupt.

In such cases, the following point should be considered.

- While the interrupt disable flag (I) is “0”, if the interrupt request bit is cleared to “0” and the interrupt enable bit is cleared to “0” at the same time using an instruction such as LDM, the interrupt will actually be enabled before the request bit is cleared to “0”, causing the interrupt to be accepted.

  To prevent this, use an instruction such as CLB to clear the request bit to “0” first, then enable the interrupt.

(4) Interrupt Control within Interrupt Routines

After an interrupt is accepted and execution of the interrupt routine begins, the interrupt disable flag (I) is set to “1” automatically to prevent multiple interrupts. To enable multiple interrupts, use the CLI instruction within the interrupt routine to clear the interrupt disable flag (I) to “0”.

3.4.3 Instructions Related to Subroutine Processing

Normally, the JSR instruction is used to jump to a subroutine. When this instruction is executed, the current program counter values, first PCH then PCL, are pushed onto the stack automatically and the stack pointer is moved accordingly. However, in contrast to interrupt handling, the contents of the processor status register are not saved automatically when a subroutine is called. If it is necessary to save the contents of the processor status register, execute the PHP instruction. Executing the JSR instruction does not alter the content of the processor status register. Therefore, saving the contents of the processor status register using the PHP instruction may be performed either immediately before the JSR instruction or immediately after it (at the beginning of the subroutine). However, if such a stack operation instruction is executed within a subroutine, do not fail to perform the opposite operation before returning from (that is, within) the subroutine.

Execute the RTS instruction to return from a subroutine. When this instruction is executed, the return address saved by the JSR instruction is returned to the program counter automatically. Likewise in contrast to interrupt handling, the contents of the processor status register are not restored. If the PHP or PHA instruction is used within a subroutine to store the contents of the processor status register or accumulator, do not fail to perform the opposite stack operation, using the PLP or PLA instruction, before returning from (that is, within) the subroutine.

Figure 3.4.1 shows pushing and pulling values onto and from the stack during interrupt handling and subroutine processing. Table 3.4.1 shows instructions for storing and retrieving values in the accumulator and processor status register.

Instructions Related to Interrupt Processing and Subroutine Processing

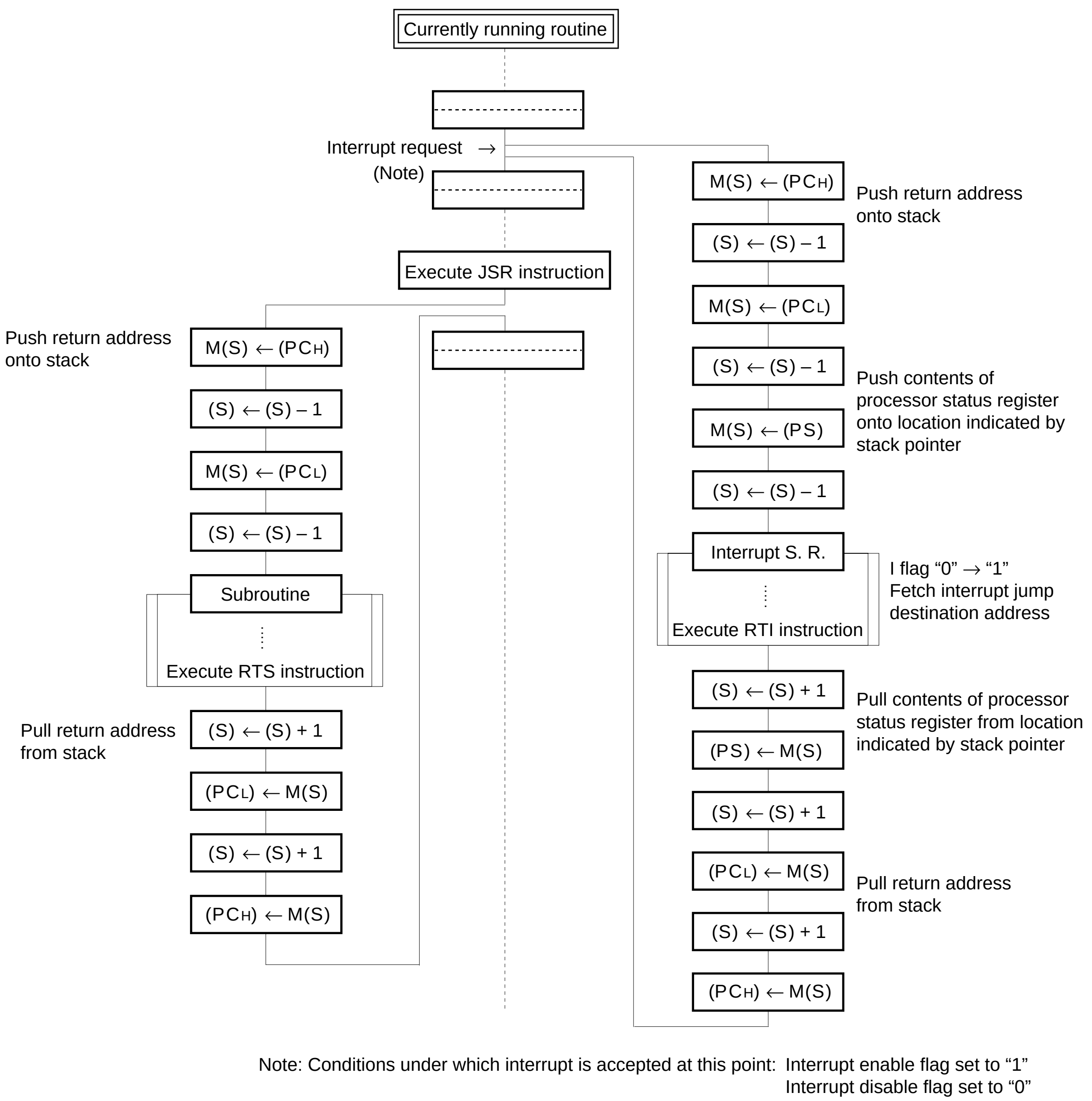


Fig.3.4.1 Pushing and pulling values onto and from the stack

Table 3.4.1 Instructions for storing and retrieving values in the accumulator or processor status register

	Instruction to push onto Stack	Instruction to pull from Stack
Accumulator	PHA	PLA
Processor status register	PHP	PLP

4. NOTES ON USE

The information below applies to the entire 740 Family. Please refer to it in conjunction with the usage notes of each specific product model.

4.1 Notes on input and output ports

4.1.1 Notes in standby state

In standby state*1, do not make pin levels "undefined" when I/O ports are set to input mode. In addition, the same note is necessary even when N-channel open-drain I/O ports are set to output mode.
Pull-up (connect the port to V_{CC}) or pull-down (connect the port to V_{SS}) these ports through a resistor.
When determining a resistance value, note the following points:

- External circuit
- Variation of output levels during the ordinary operation

●Reason

An transistor becomes an OFF state when an I/O port is set as input mode by the direction register, so that the port enter a high-impedance state. At this time, the potential which is input to the input buffer in a microcomputer is unstable in the state that input levels are "undefined". This may cause power source current. Even when an I/O port of N-channel open-drain is set as output mode by the direction register, if the contents of the port latch is "1", the same phenomenon as that of an input port will occur.

✻1 standby state: Stop mode by executing STP instruction
Wait mode by executing WIT instruction

4.1.2 Modifying output data with bit managing instruction

When the port latch of an I/O port is modified with the bit managing instruction*2, the value of the unspecified bit may be changed.

●Reason

I/O ports are set to input or output mode in bit units. Reading from a port register or writing to it involves the following operations.

- Port in input mode
  - Read: Read the pin level.
  - Write: Write to the port latch.
- Port in output mode
  - Read: Read the port latch or read the output from the peripheral function (specifications differ depending on the port).
  - Write: Write to the port latch. (The port latch value is output from the pin.)

Since bit managing instructions*1 are read-modify-write instructions, *2 using such an instruction on a port register causes a read and write to be performed simultaneously on the bits other than the one specified by the instruction.
When an unspecified bit is in input mode, its pin level is read and that value is written to the port latch. If the previous value of the port latch differs from the pin level, the port latch value is changed.
If an unspecified bit is in output mode, the port latch is generally read. However, for some ports the peripheral function output is read, and the value is written to the port latch. In this case, if the previous value of the port latch differs from the peripheral function output, the port latch value is changed.

✻1. Bit managing instructions: SEB and CLB instructions
✻2. Read-modify-write instructions: Instructions that read memory in byte units, modify the value, and then write the result to the same location in memory in byte units

4.2 Termination of unused pins

At the termination of unused pins, perform wiring at the shortest possible distance (20 mm or less) from microcomputer pins. With regard to an effects on the system, thoroughly perform system evaluation on the user side.

4.2.1 Appropriate termination of unused pins

① Output-only pins:
Open.

② Input-only pins:
Connect each pin via a 1 kΩ to 10 kΩ resistor (reference value) to V_{CC} or V_{SS}. If the port allows selection of an on-chip pull-up or pull-down resistor, the on-chip pull-up or pull-down resistor may be used.
In addition, pins (CNV_{SS} and INT pins, etc.) for which the operating mode is affected by the voltage level, select V_{CC} or V_{SS} after checking the mode.

③ I/O ports:
Set the I/O ports for the input mode and connect them to V_{CC} or V_{SS} through each resistor of 1 kΩ to 10 kΩ (reference value).
Ports that permit the selecting of a built-in pull-up/pull-down resistor can also use this resistor. Set the I/O ports for the output mode and open them at “L” or “H”.

- When opening them in the output mode, the input mode of the initial status remains until the mode of the ports is switched over to the output mode by the program after reset. Thus, the potential at these pins is undefined and the power source current may increase in the input mode. With regard to an effects on the system, thoroughly perform system evaluation on the user side.
- Since the direction register setup may be changed because of a program runaway or noise, set direction registers by program periodically to increase the reliability of program.

④ The AVss pin when not using the A/D converter:
When not using the A/D converter, handle a power source pin for the A/D converter, AV_{SS} and AV_{CC} pins as follows:

- AV_{SS}: Connect to the V_{SS} pin.
- AV_{CC}: Connect to the V_{CC} pin.

4.2.2 Termination remarks

① I/O ports:
Do not open in the input mode.

●Reason

- The power source current may increase depending on the first-stage circuit.
- An effect due to noise may be easily produced as compared with proper termination ① and shown on the above.

② I/O ports:
When setting for the input mode, do not connect to V_{CC} or V_{SS} directly.

●Reason
If the direction register setup changes for the output mode because of a program runaway or noise, a short circuit may occur between a port and V_{CC} (or V_{SS}).

③ I/O ports:
When setting for the input mode, do not connect multiple ports in a lump to V_{CC} or V_{SS} through a resistor.

●Reason
If the direction register setup changes for the output mode because of a program runaway or noise, a short circuit may occur between ports.

4.3 Notes on interrupts

4.3.1 Setting for interrupt request bit and interrupt enable bit

To set an interrupt request bit and an interrupt enable bit for interrupts, execute as the following sequence:

① Clear an interrupt request bit to “0” (no interrupt request issued).
② Set an interrupt enable bit to “1” (interrupts enabled).

●Reason

If the above setting are performed simultaneously with one instruction, an unnecessary interrupt processing routine is executed. Because an interrupt enable bit is set to “1” (interrupts enabled) before an interrupt request bit is cleared to “0.”

4.3.2 Switching of detection edge

If it is not necessary to generate interrupts synchronized with certain settings, such as setting the active edge for external interrupts or switching the interrupt source for a vector in cases where multiple interrupt sources are assigned to the same interrupt vector, use the following procedure to make the settings.

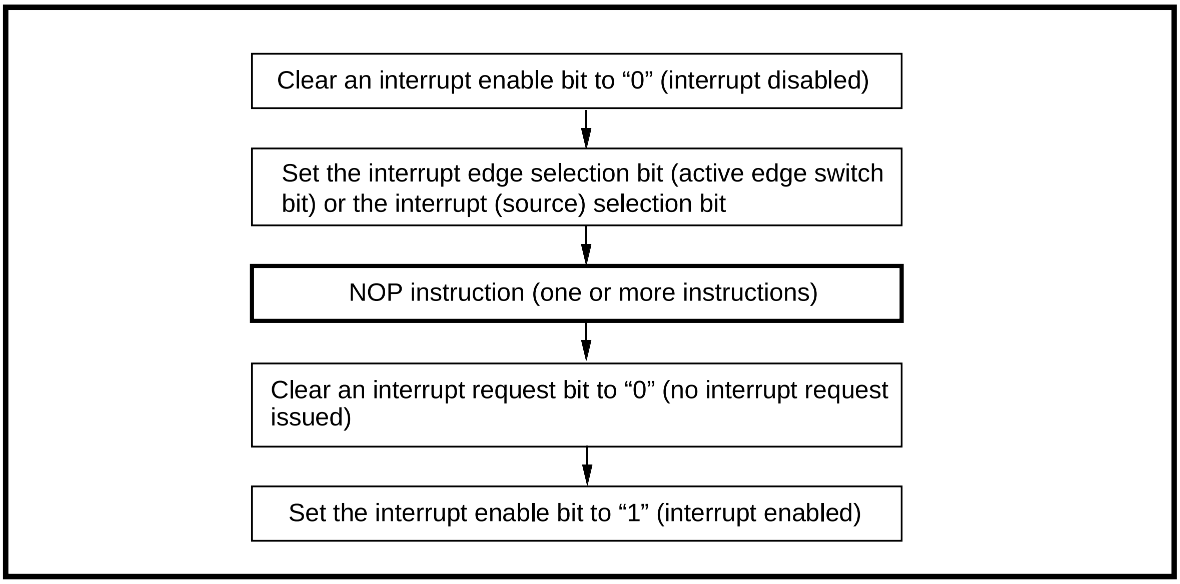


Fig. 4.3.1 Switching sequence of detection edge

●Reason

The interrupt request bit may be set to “1” in the following cases:

• When switching the active edge for external interrupts.
• When switching the interrupt source for a vector in cases where multiple interrupt sources are assigned to the same interrupt vector.

4.3.3 Distinction of interrupt request bit

When executing the BBC or BBS instruction to an interrupt request (request distinguish) bit of an interrupt request register (interrupt request distinguish register) immediately after this bit is set to "0", execute one or more instructions before executing the BBC or BBS instruction.

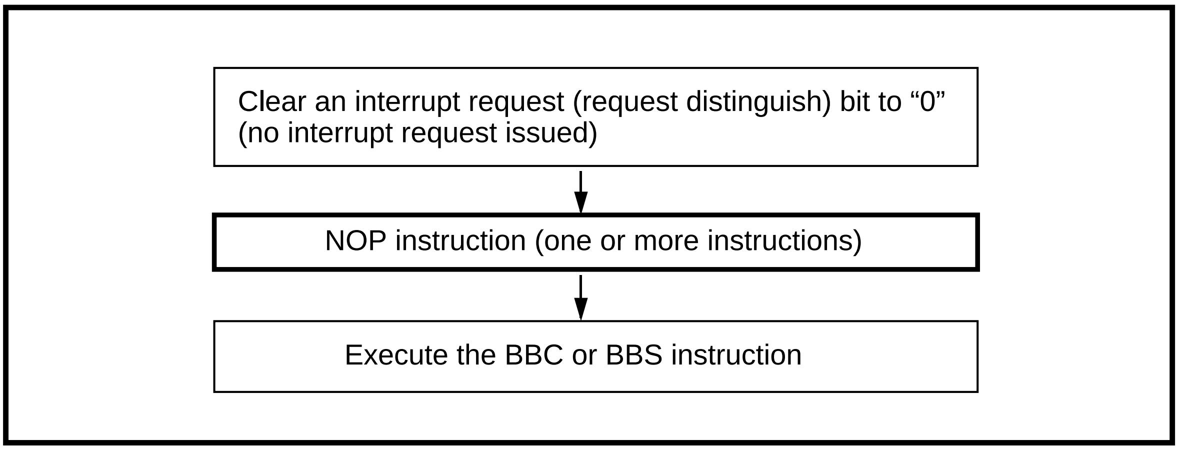


Fig. 4.3.2 Distinction sequence of interrupt request bit

●Reason

If the BBC or BBS instruction is executed immediately after an interrupt request (request distinguish) bit of an interrupt request register (interrupt request distinguish register) is cleared to "0," the value of the interrupt request (request distinguish) bit before being cleared to "0" is read.

4.4 Notes on programming

4.4.1 Processor Status Register

(1) Initialization of Processor Status Register

Flags which affect program execution must be initialized after a reset. In particular, it is essential to initialize the T and D flags because they have an important effect on calculations.

●Reason

After a reset, the contents of processor status register (PS) are undefined except for the I flag which is “1.”

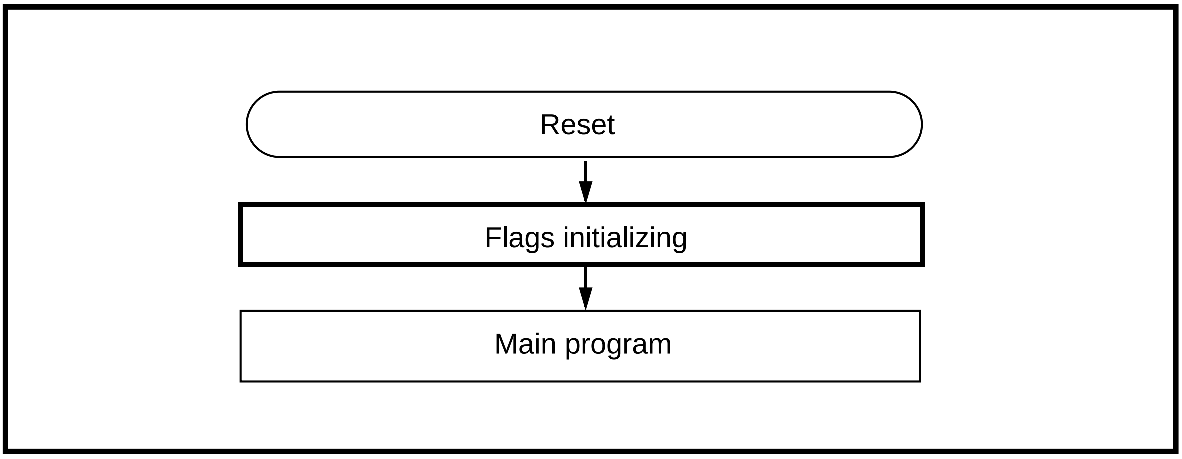


Fig. 4.4.1 Initialization of flags in Processor Status Register

(2) How to reference Processor Status Register

To reference the contents of the processor status register (PS), execute the PHP instruction once then read the contents of (S + 1). If necessary, execute the PLP instruction to return the PS to its original status.

A NOP instruction should be executed after every PLP instruction.

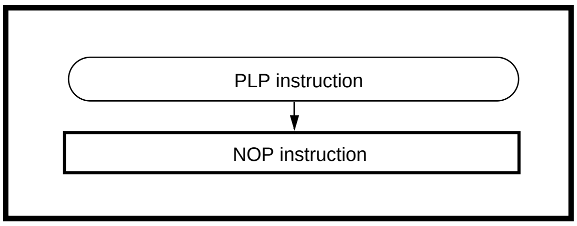


Fig. 4.4.2 PLP instruction execution sequence

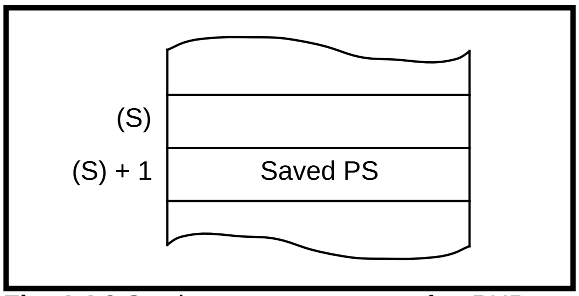


Fig. 4.4.3 Stack memory contents after PHP instruction execution

4.4.2 BRK instruction

(1) Method detecting interrupt source

It can be detected that the BRK instruction interrupt event or the least priority interrupt event by referring the stored B flag state. Refer the stored B flag state in the interrupt routine, in this case.

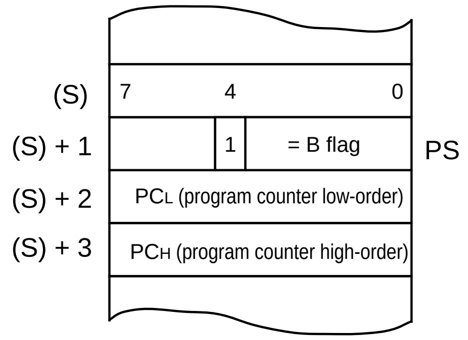


Fig. 4.4.4 Contents of stack memory in interrupt processing routine

(2) Interrupt priority level

At the following status,
① the interrupt request bit has set to "1."
② the interrupt enable bit has set to "1."
③ the interrupt disable flag (I) has set to "1."
If the BRK instruction is executed, the interrupt disable state is cancelled and it becomes in the interrupt enable state. So that the requested interrupts (the interrupts that corresponding to their request bits have set to "1") are accepted.

4.4.3 Decimal calculations

(1) Execution of Decimal calculations

The ADC and SBC are the only instructions which will yield proper decimal results in decimal mode. To calculate in decimal notation, set the decimal mode flag (D) to "1" with the SED instruction. After executing the ADC or SBC instruction, execute another instruction before executing the SEC, CLC, or CLD instruction.

(2) Status flags in decimal mode

When decimal mode is selected (D = 1), the values of three of the flags in the status register (the flags N, V, and Z) are invalid after a ADC or SBC instruction is executed. The carry flag (C) is set to “1” if a carry is generated as a result of the calculation, or is cleared to “0” if a borrow is generated. To determine whether a calculation has generated a carry, the C flag must be initialized to “0” before each calculation. To check for a borrow, the C flag must be initialized to “1” before each calculation.

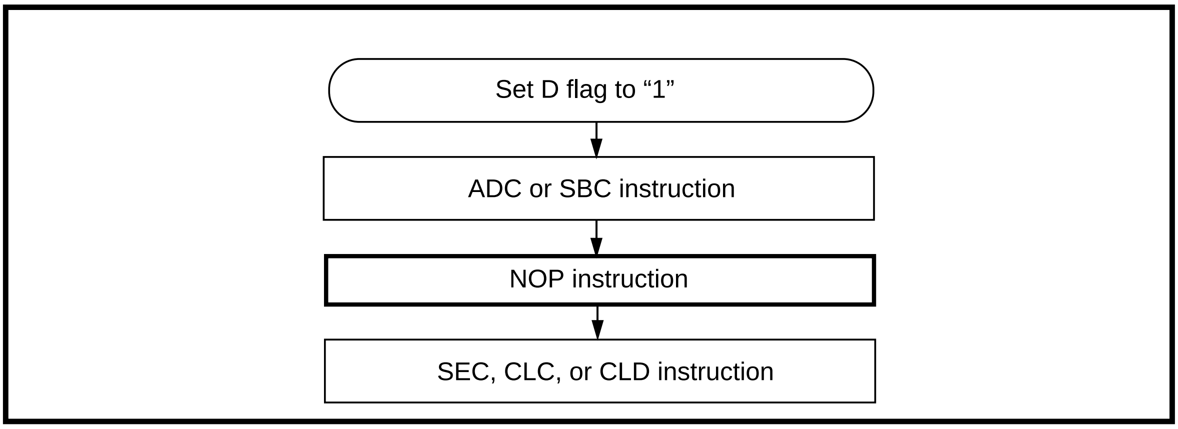


Fig. 4.4.5 Status flags in decimal mode

4.4.4 JMP instruction

When using the JMP instruction in indirect addressing mode, do not specify the last address on a page as an indirect address.

4.4.5 Multiplication and division instructions

The index mode (T) and the decimal mode (D) flags do not affect the MUL and DIV instruction. The execution of these instructions does not change the contents of the processor status register.

4.4.6 Ports

The contents of the port direction registers cannot be read.
The following cannot be used:

- The data transfer instruction (LDA, etc.)
- The operation instruction when the index X mode flag (T) is “1”
- The addressing mode which uses the value of a direction register as an index
- The bit-test instruction (BBC or BBS, etc.) to a direction register
- The read-modify-write instruction (ROR, CLB, or SEB, etc.) to a direction register

Use instructions such as LDM and STA, etc., to set the port direction registers.

4.4.7 Instruction execution time

The instruction execution time is obtained by multiplying the frequency of the internal clock ϕ by the number of cycles needed to execute an instruction.
The number of cycles required to execute an instruction is shown in the list of machine instructions.

APPENDIX 1. Instruction Cycles in each Addressing Mode

Clock ϕ controls the system timing of 740 Family. The SYNC signal and the value of PC (Program Counter) are output in every instruction fetch cycle. The Op-Code is fetched during the next half-period of ϕ. The instruction decoder of CPU decodes this Op-Code and determines the following how to execute the instruction. The instruction timings of all addressing modes are described on the following pages.
The ϕ, SYNC, R/$\overline{\text{W}}$ ($\overline{\text{RD}}$, $\overline{\text{WR}}$), ADDR (ADDR_L, ADDR_H), and DATA signals in these figures indicate the status of the internal bus. These signals cannot be seen directly in single-chip mode, but they can be checked on products that support use of microprocessor mode.
The combination of these signals differs according to the microcomputer's type. The following table lists the valid signal for each product.

Valid signal for each product

Type	ϕ	SYNC	R/$\overline{\text{W}}$	$\overline{\text{RD}}$	$\overline{\text{WR}}$	ADDR	DATA	ADDR_H	ADDR_L/DATA
M507XX M509XX M374XX (Except M37451)	○	○	○			○	○		
M38XXX M375XX M372XX M371XX	○	○		○	○	○	○		
M37451	○	○	○	○ (Note)	○ (Note)	○	○		
M50734	○	○		○	○			○	○

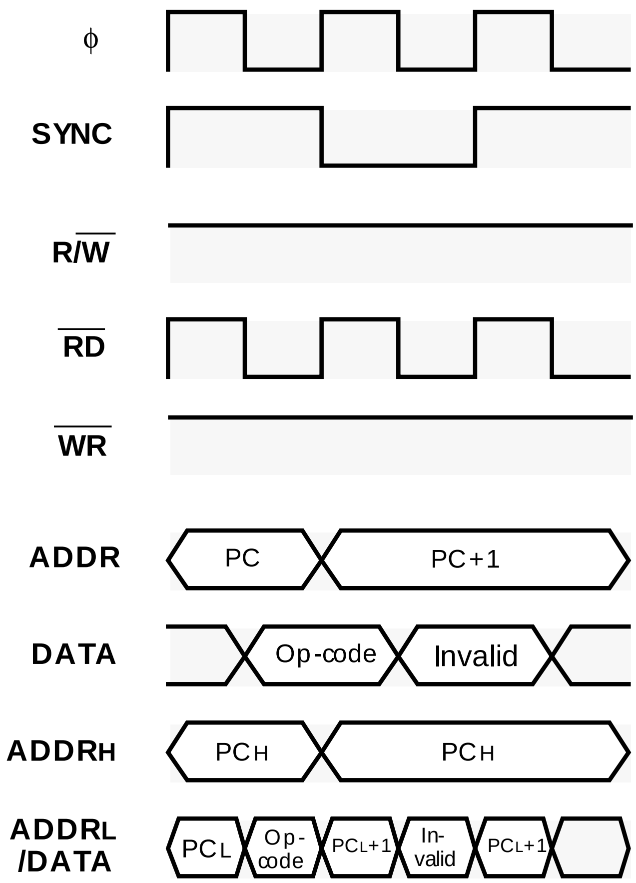
Note: Only 80-pin version.

IMPLIED

Instructions : ∆CLC ∆SEC
∆CLD ∆SED
∆CLI ∆SEI
∆CLT ∆SET
∆CLV ∆TAX
∆DEX ∆TAY
∆DEY ∆TSX
∆INX ∆TXA
∆INY ∆TXS
∆NOP ∆TYA

Byte length : **1**
Cycle number : **2**

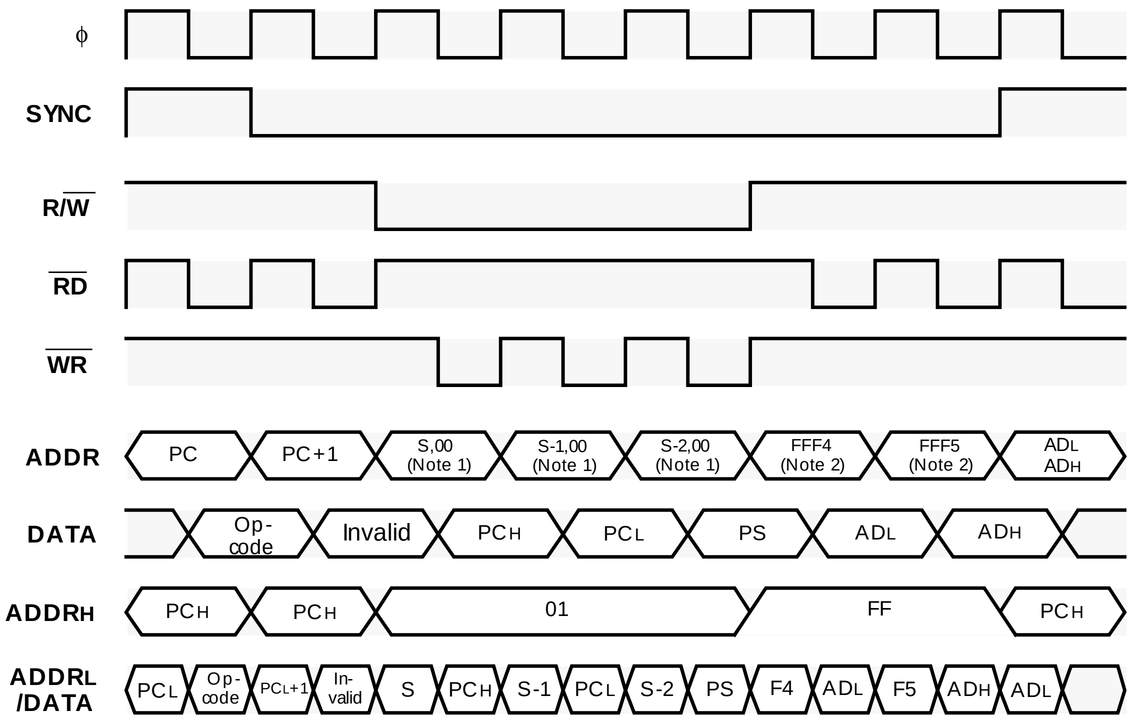
Timing :



IMPLIED

Instruction : ΔBRK
Byte length : **1**
Cycle number : **7**

Timing :



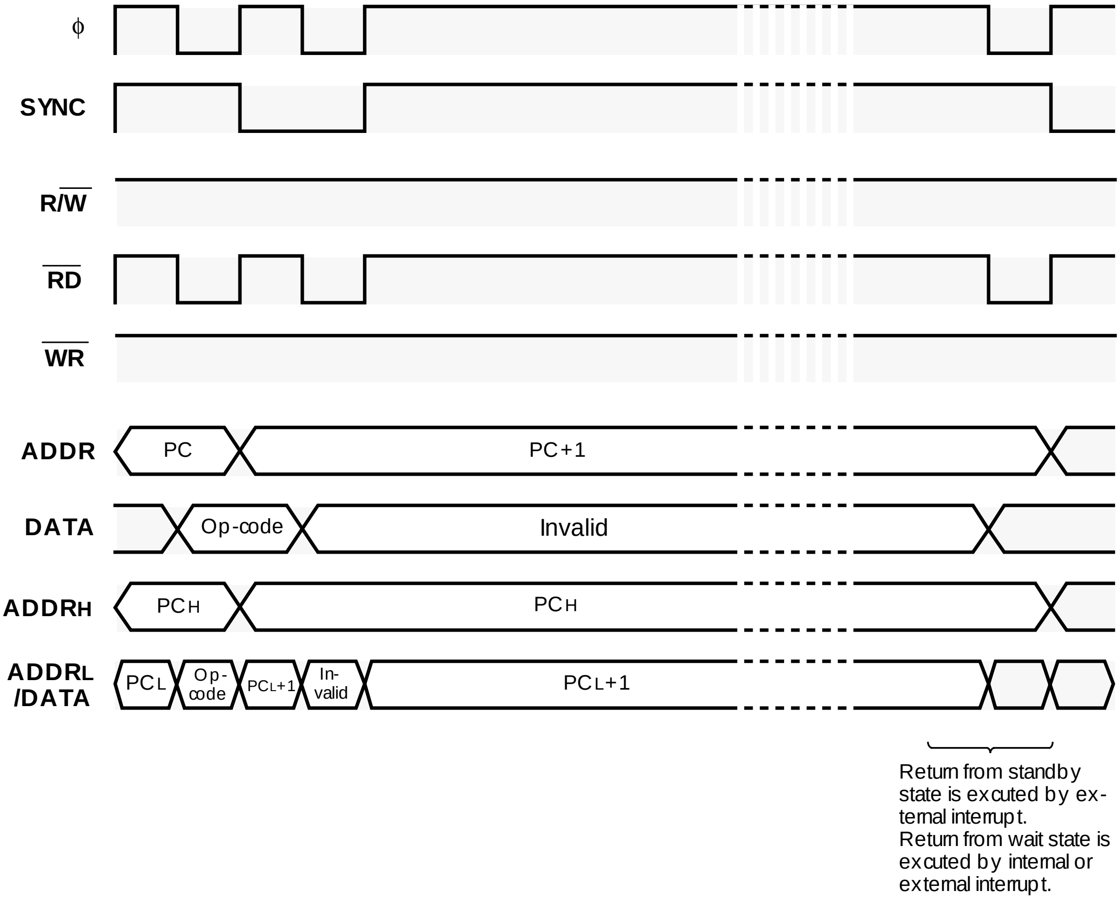
Notes 1 : Some products are "01" or content of SPS flag.
2 : Some products differ the address.

IMPLIED

Instructions : ∆**STP**
∆**WIT**

Byte length : **1**

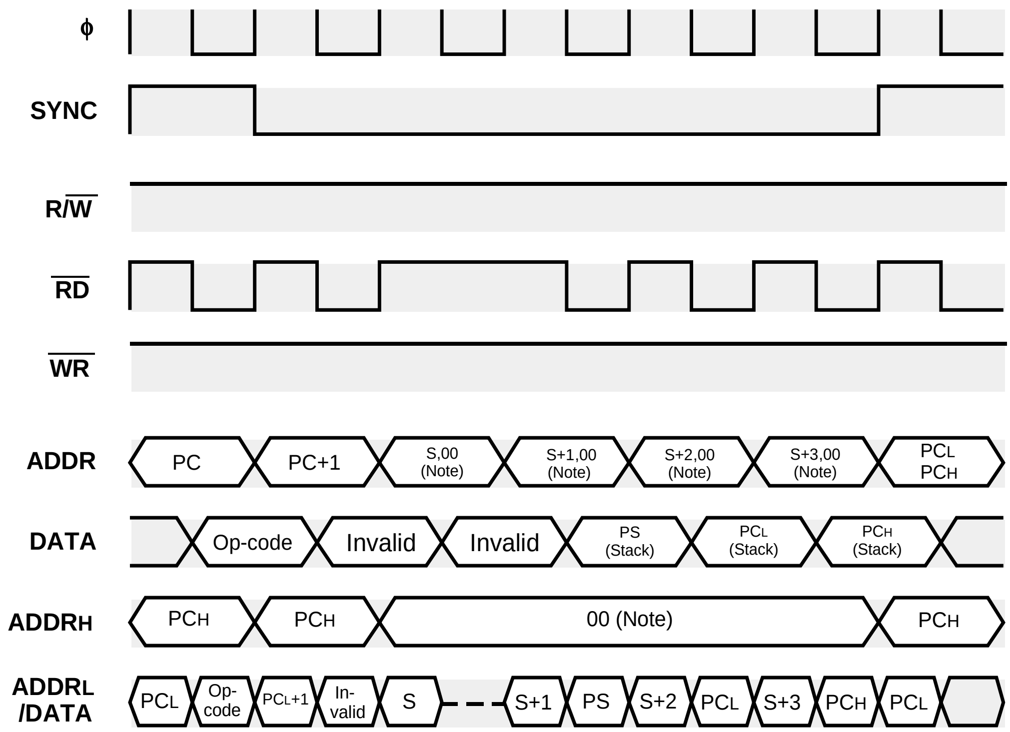
Timing :



IMPLIED

Instruction : ΔRTI
Byte length : **1**
Cycle number : **6**

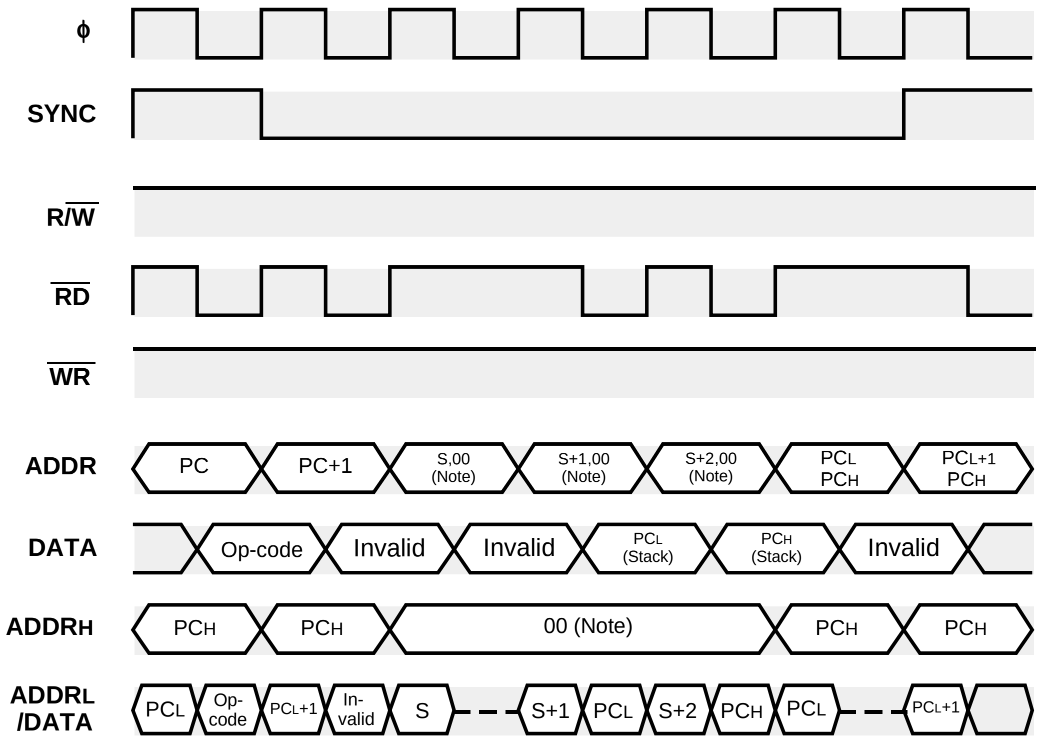
Timing :



Note: Some products are “01” or content of SPS flag.

Instruction : ΔRTS
Byte length : **1**
Cycle number : **6**

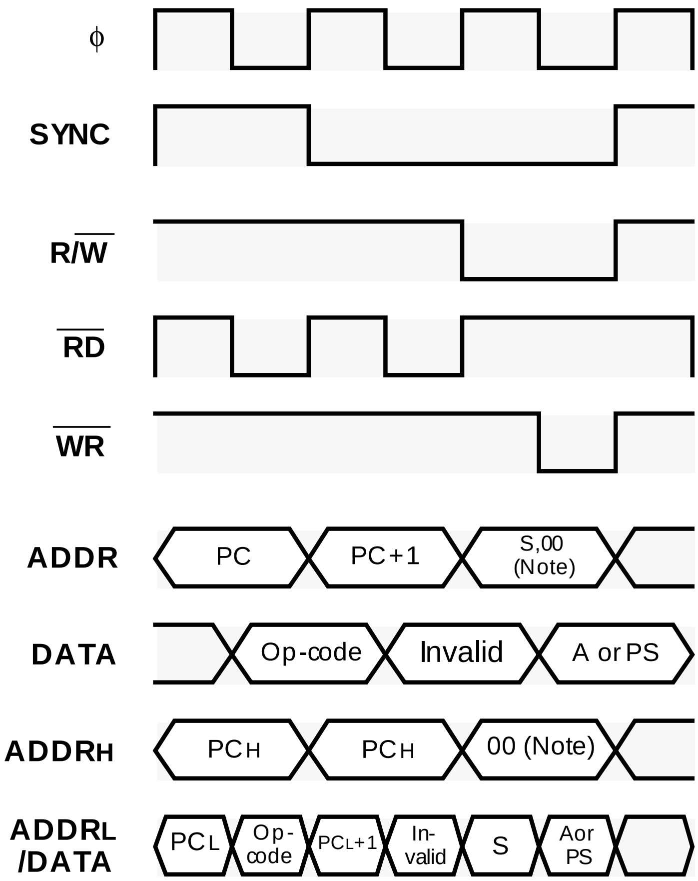
Timing :



Note: Some products are “01” or content of SPS flag.

Instructions : ΔPHA
ΔPHP
Byte length : **1**
Cycle number : **3**

Timing :



Note: Some products are “01” or content of SPS flag.

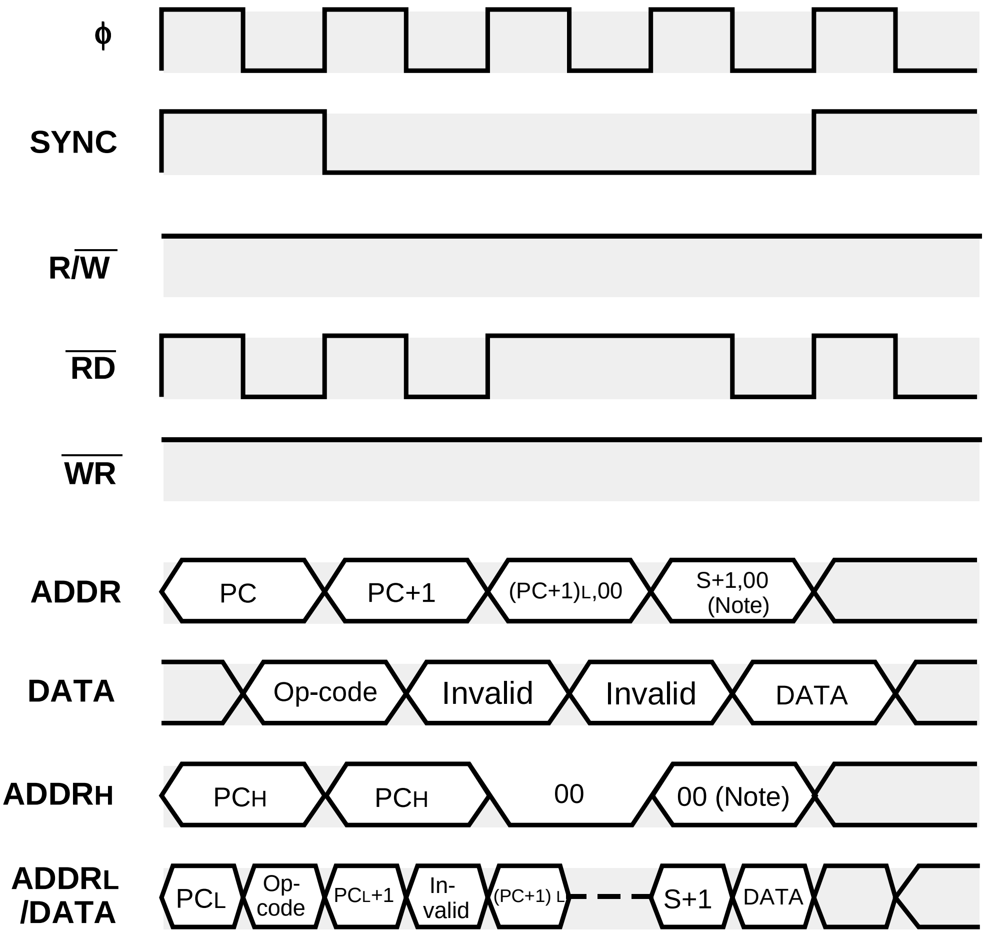
IMPLIED

Instructions : ΔPLA
ΔPLP

Byte length : **1**

Cycle number : **4**

Timing :



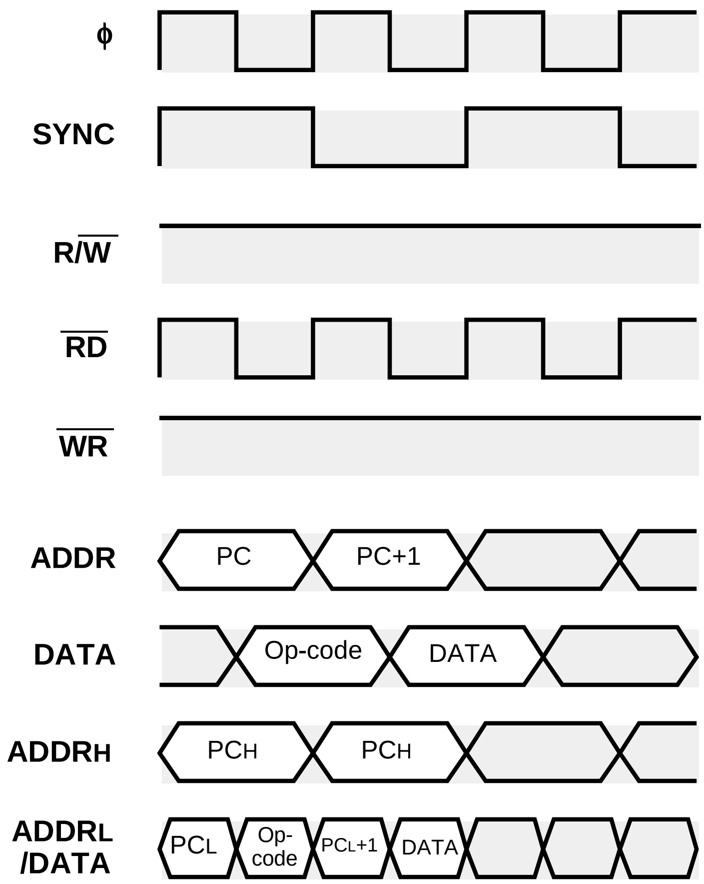
Note: Some products are "01" or content of SPS flag.

IMMEDIATE

Instructions : **ΔADCΔ#$nn (T=0)**
ΔANDΔ#$nn (T=0)
ΔCMPΔ#$nn (T=0)
ΔCPXΔ#$nn
ΔCPYΔ#$nn
ΔEORΔ#$nn (T=0)
ΔLDAΔ#$nn (T=0)
ΔLDXΔ#$nn
ΔLDYΔ#$nn
ΔORAΔ#$nn (T=0)
ΔSBCΔ#$nn (T=0)

Byte length : **2**
Cycle number : **2**

Timing :

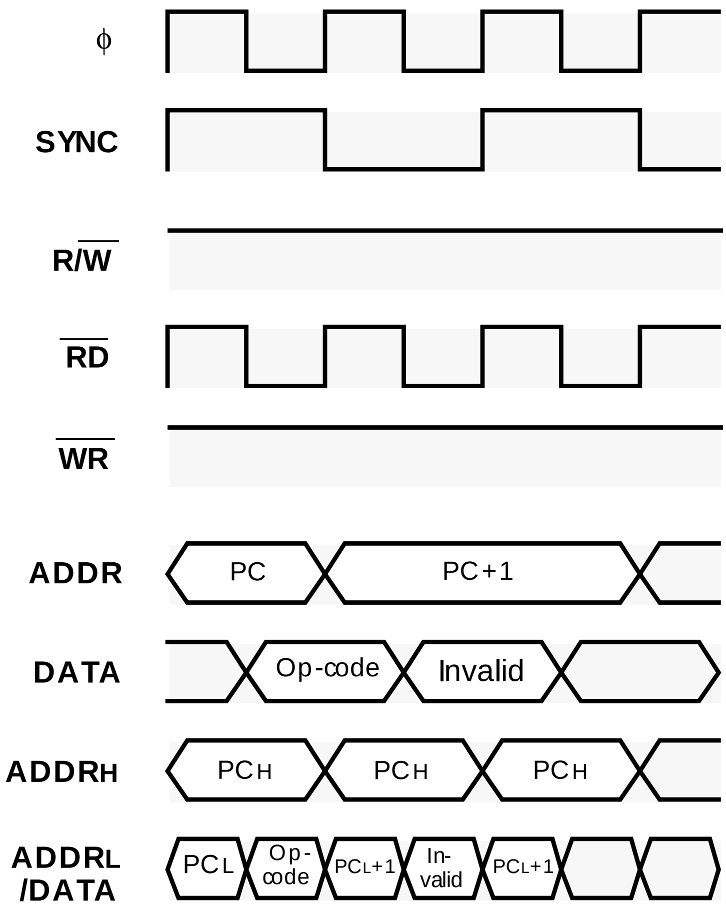


Instructions : **ΔASL ΔA**
ΔDEC ΔA
ΔINC ΔA
ΔLSR ΔA
ΔROL ΔA
ΔROR ΔA

Byte length : **1**

Cycle number : **2**

Timing :



ACCUMULATOR BIT RELATIVE

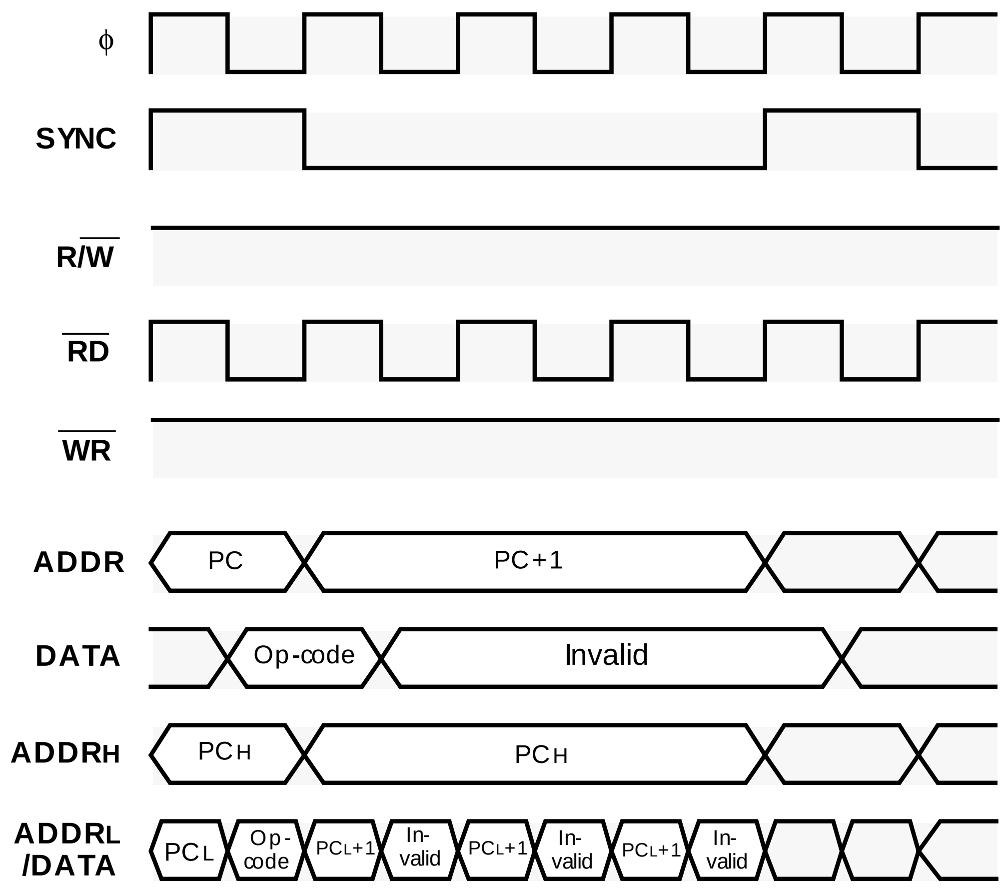
Instructions : ∆BBC∆i,A,$hhll
∆BBS∆i,A,$hhll

Byte length : **2**

(1) With no branch

Cycle number : **4**

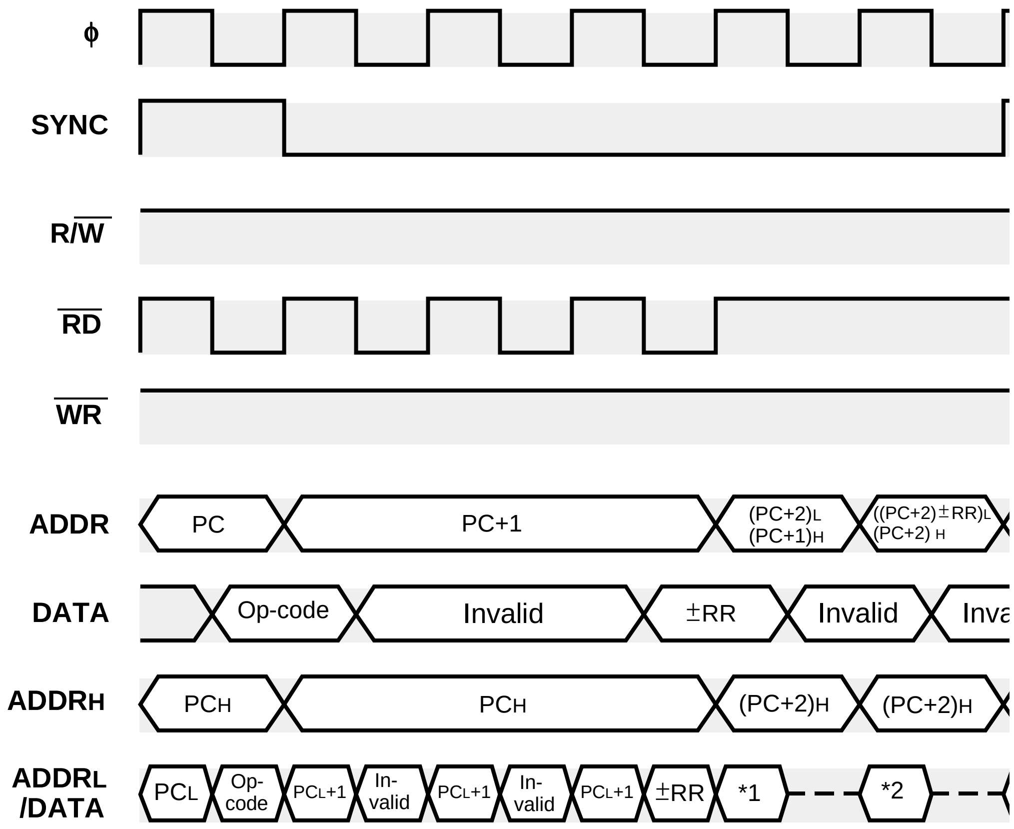
Timing :



Instructions : ΔBBCΔi,A,$hhll
ΔBBSΔi,A,$hhll
Byte length : 2

(2) With branch
Cycle number : 6

Timing :

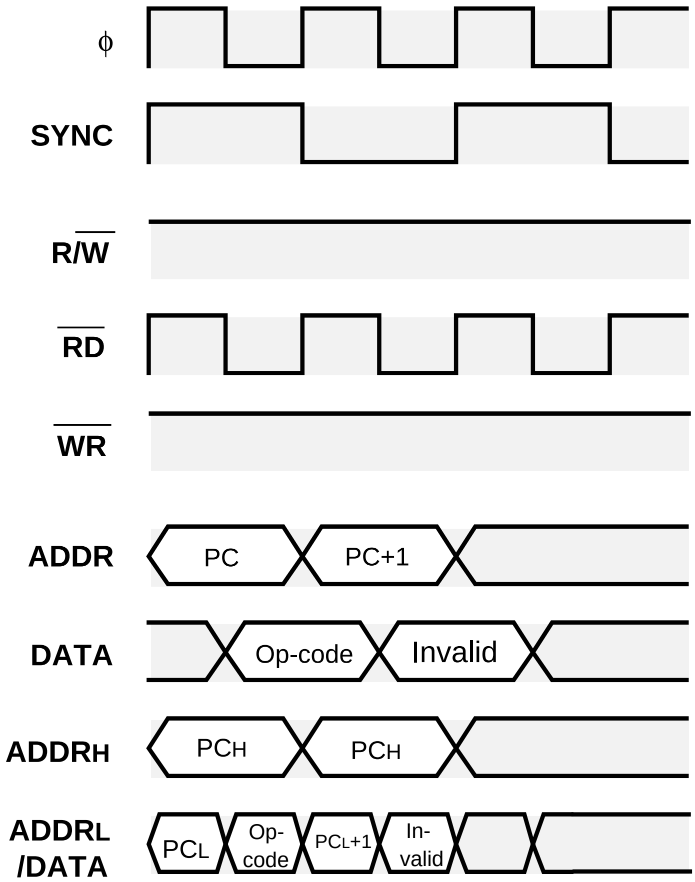


RR : Offset address
*1 : (PC+1)L
*2 : ((PC+2)±RR)L

ACCUMULATOR BIT

Instructions : ΔCLBΔi,A
ΔSEBΔi,A
Byte length : 1
Cycle number : 2

Timing :



BIT RELATIVE

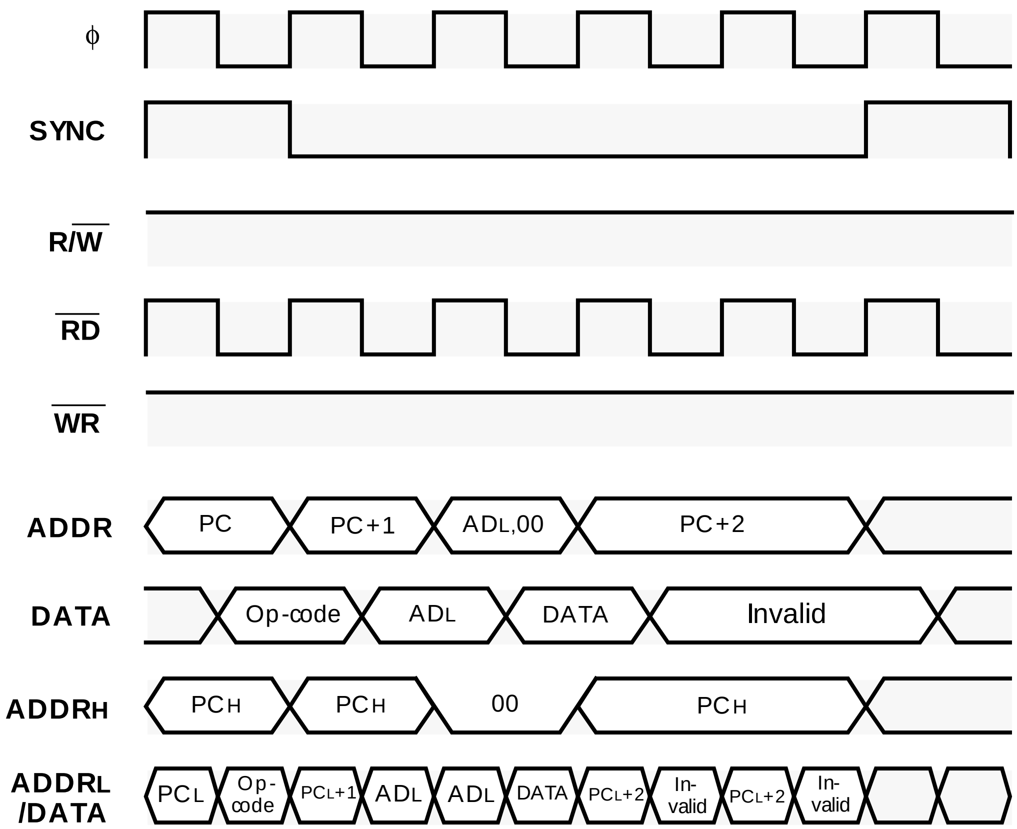
Instructions : ∆**BBC∆i,$zz,$hhll**
∆**BBS∆i,$zz,$hhll**

Byte length : **3**

(1) With no branch

Cycle number : **5**

Timing :



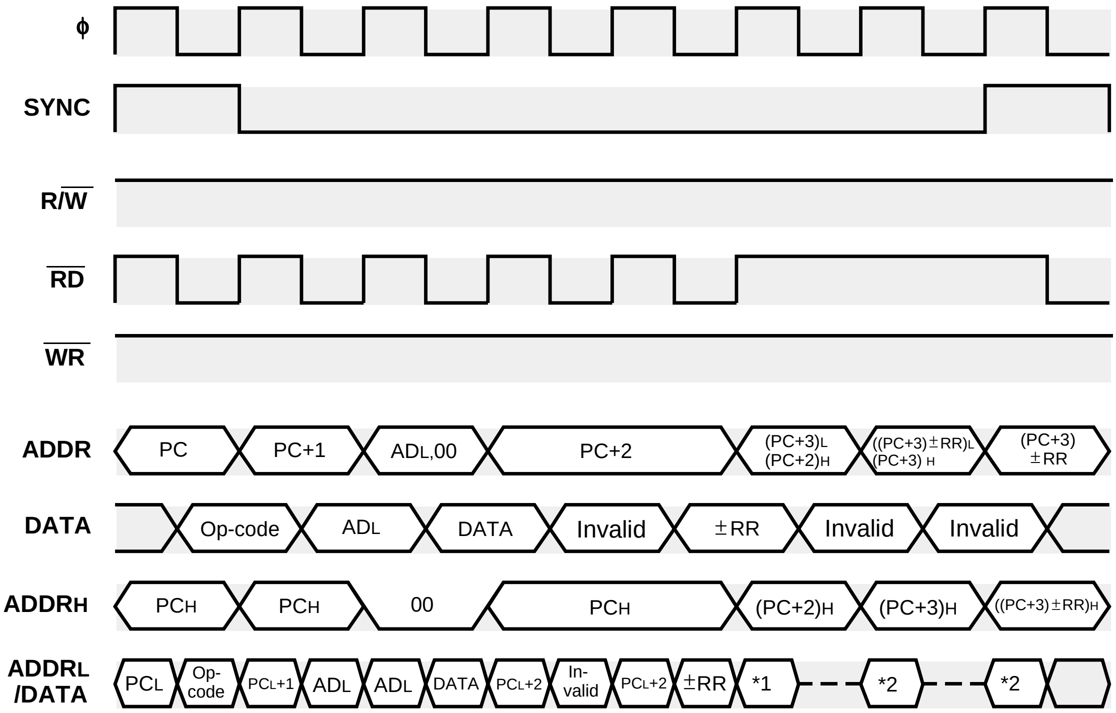
Instructions : ΔBBCΔi,$zz,$hhll
ΔBBSΔi,$zz,$hhll

Byte length : **3**

(2) With branch

Cycle number : **7**

Timing :

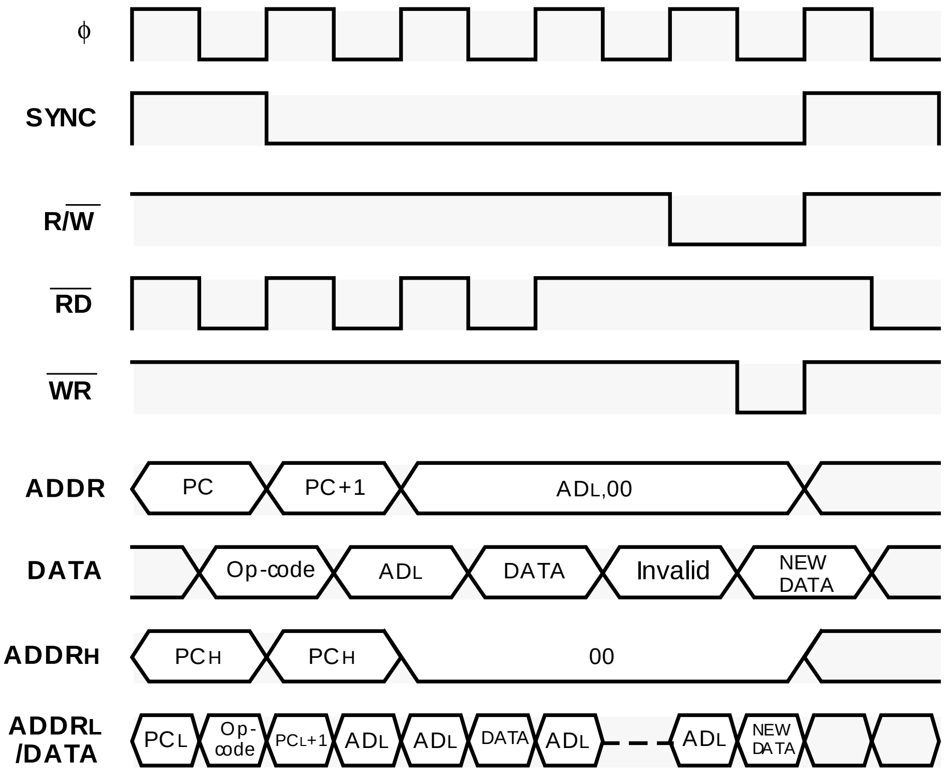


RR : Offset address
*1 : (PC+3)L
*2 : ((PC+3)±RR)L

ZERO PAGE BIT

Instructions : ∆CLB∆i,$zz
∆SEB∆i,$zz
Byte length : **2**
Cycle number : **5**

Timing :



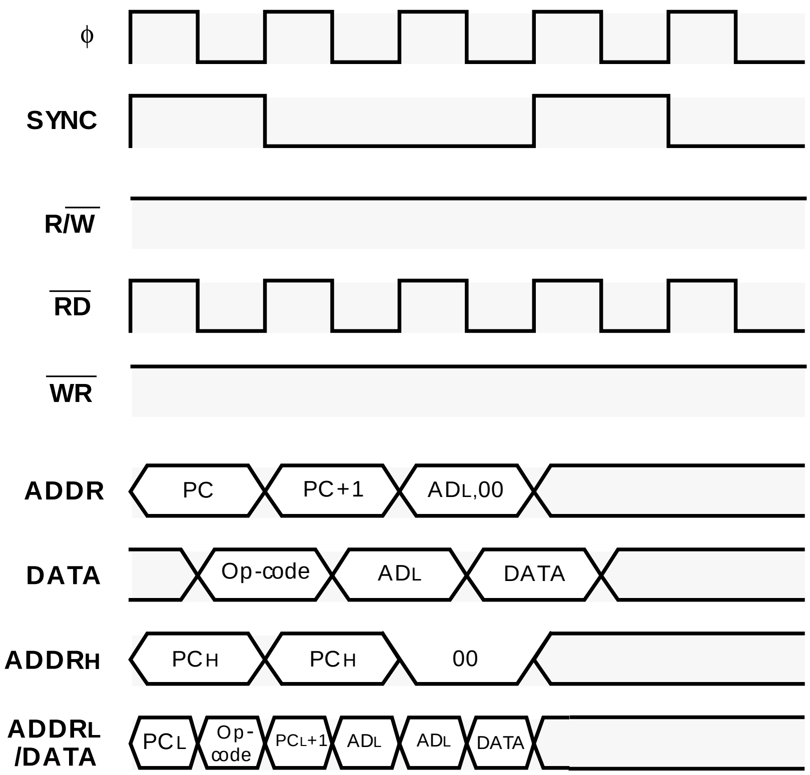
ZERO PAGE

Instructions : ∆ADC ∆$zz (T=0)
∆AND ∆$zz (T=0)
∆BIT ∆$zz
∆CMP ∆$zz (T=0)
∆CPX ∆$zz
∆CPY ∆$zz
∆EOR ∆$zz (T=0)
∆LDA ∆$zz (T=0)
∆LDX ∆$zz
∆LDY ∆$zz
∆ORA ∆$zz (T=0)
∆SBC ∆$zz (T=0)
∆TST ∆$zz

Byte length : **2**

Cycle number : **3**

Timing :



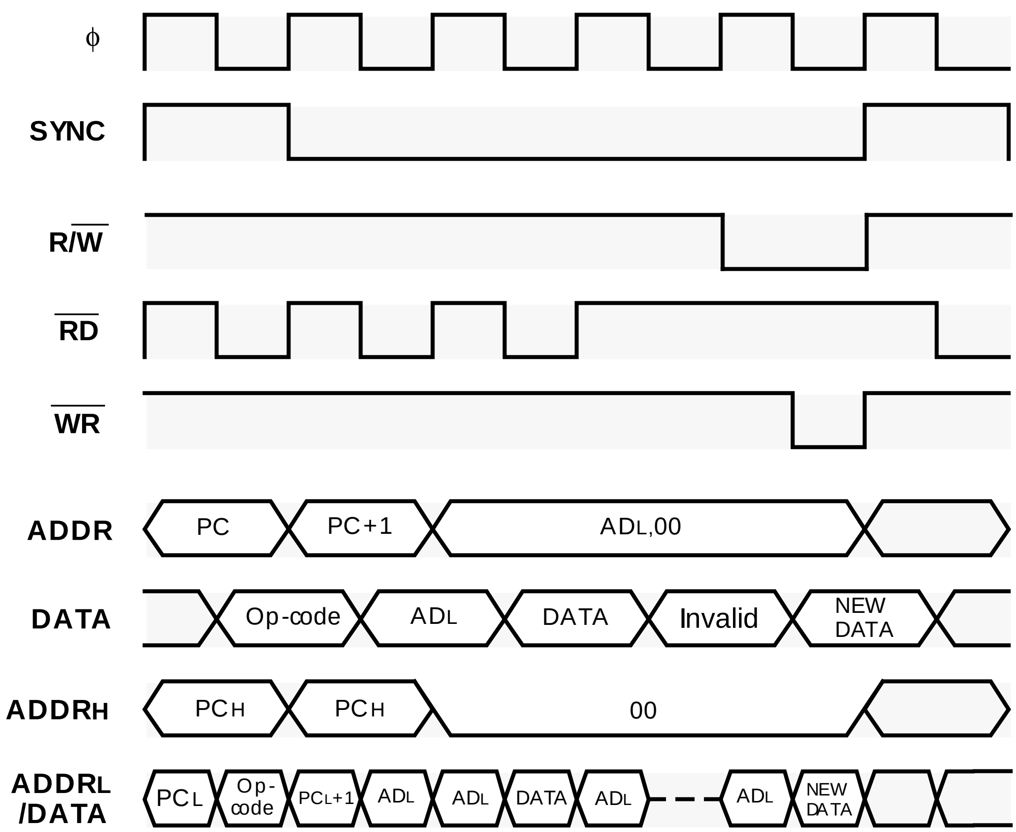
ZERO PAGE

Instructions : ΔASL Δ$zz
ΔCOM Δ$zz
ΔDEC Δ$zz
ΔINC Δ$zz
ΔLSR Δ$zz
ΔROL Δ$zz
ΔROR Δ$zz

Byte length : 2

Cycle number : 5

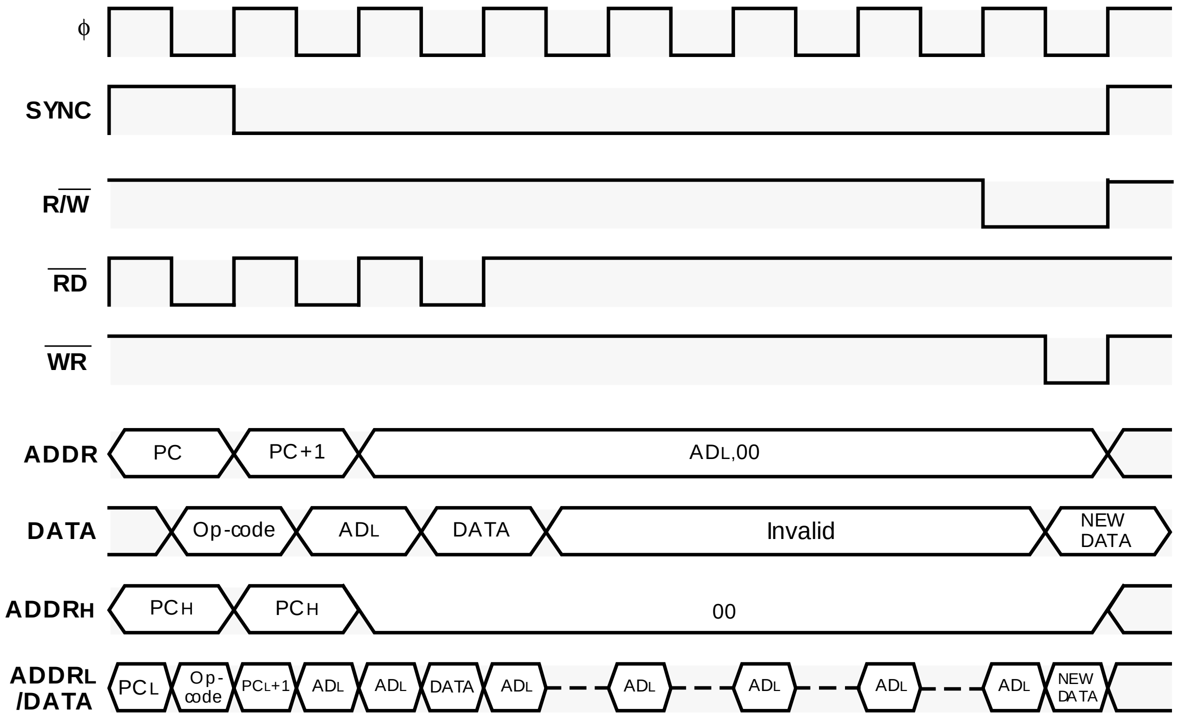
Timing :



ZERO PAGE

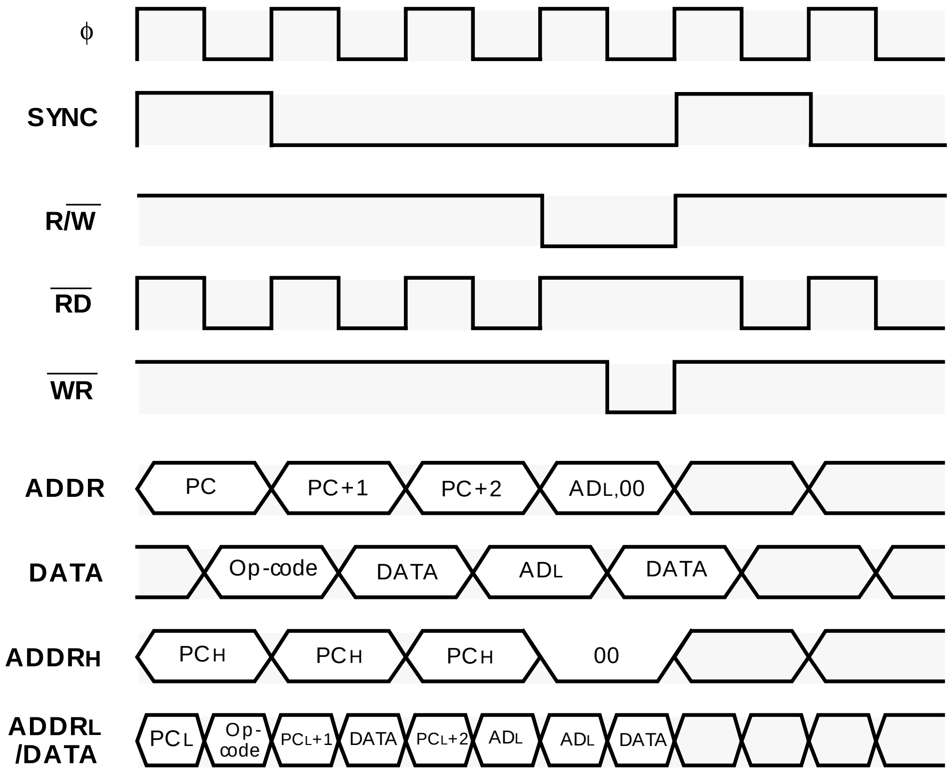
Instruction : ∆RRF∆$zz
Byte length : **2**
Cycle number : **8**

Timing :



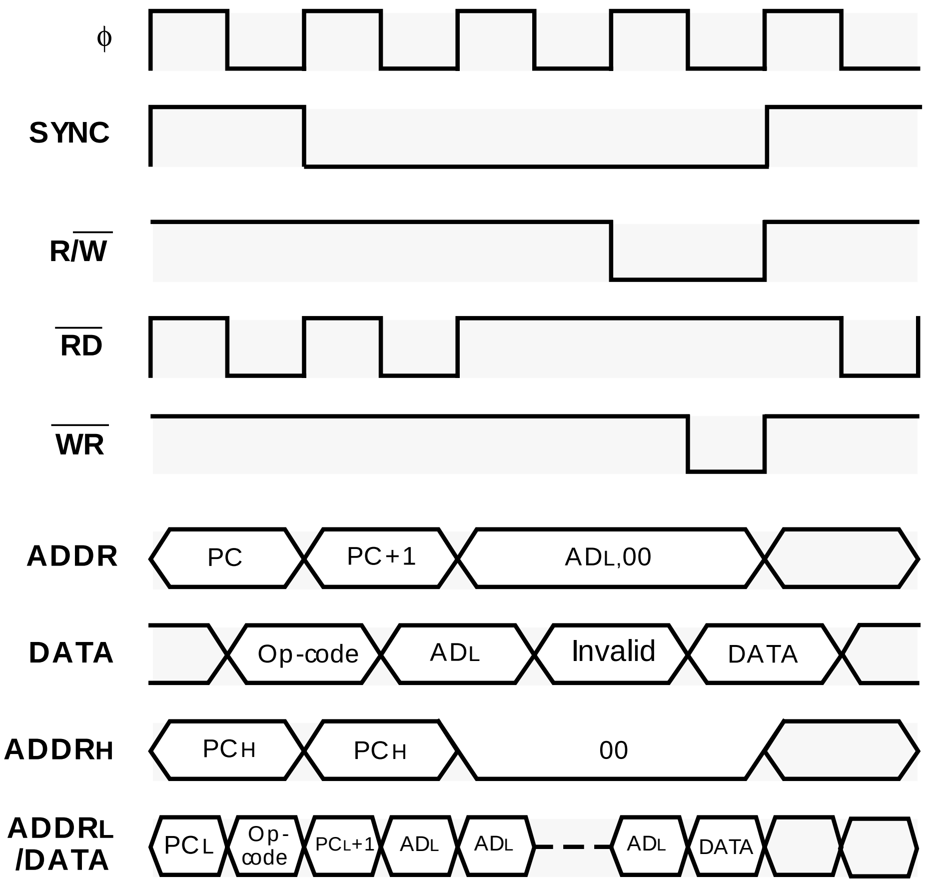
Instruction : ∆LDM∆#$nn,$zz
Byte length : **3**
Cycle number : **4**

Timing :



ZERO PAGE

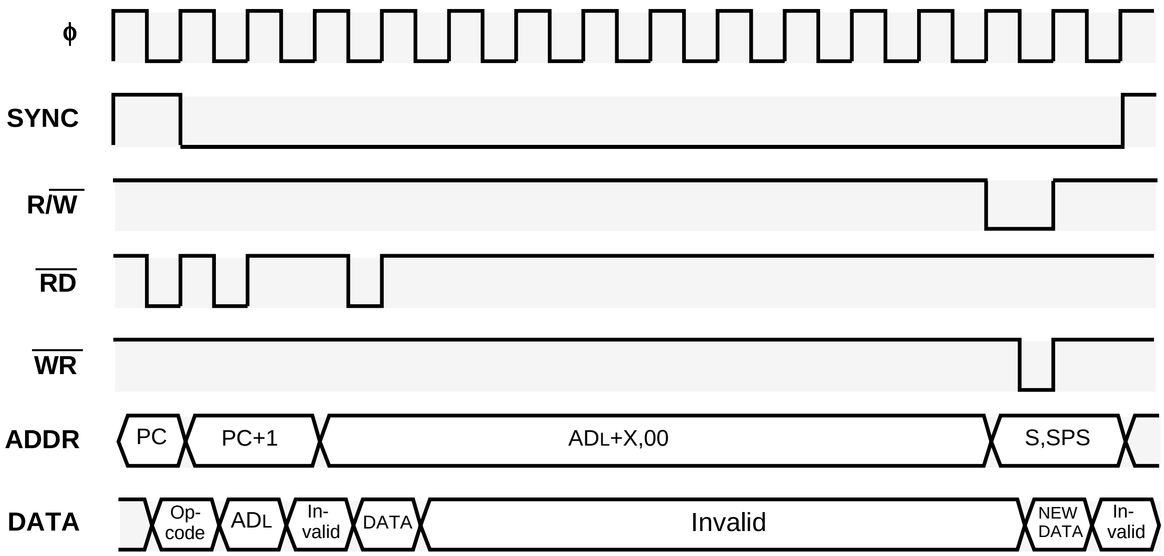
Instructions	: **ΔSTAΔ$zz** **ΔSTXΔ$zz** **ΔSTYΔ$zz**
Byte length	: **2**
Cycle number	: **4**
Timing	:



Zero Page X

Instruction : ΔMULΔ$zz,X **(Note)**
Byte length : **2**
Cycle number : **15**

Timing :



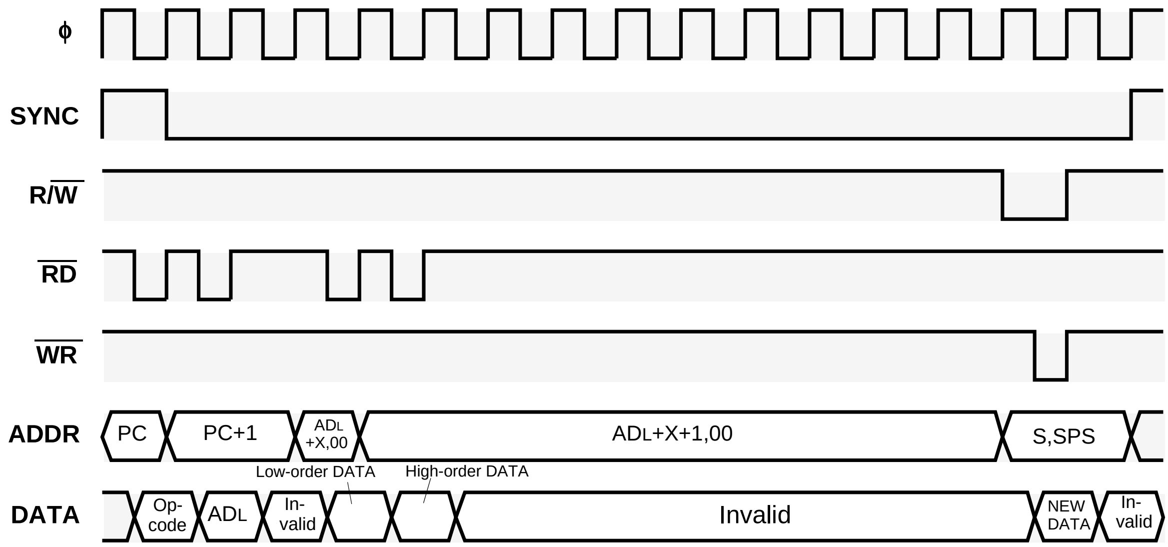
SPS: A selected page by stack page selection bit of the CPU mode register.

Note: This instruction cannot be used for some products.

Zero Page X

Instruction : ΔDIVΔ$zz,X **(Note)**
Byte length : **2**
Cycle number : **16**

Timing :



SPS: A selected page by stack page selection bit of the CPU mode register.

Note: This instruction cannot be used for some products.

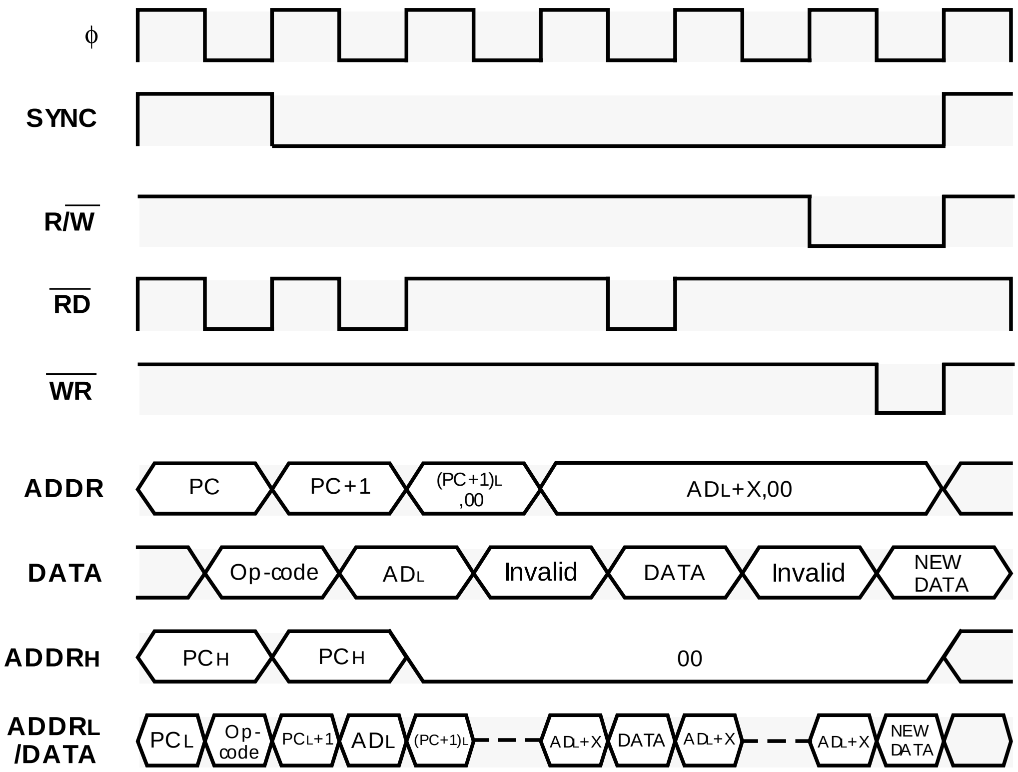
Zero Page X

Instructions : ∆ASL ∆$zz,X
∆DEC ∆$zz,X
∆INC ∆$zz,X
∆LSR ∆$zz,X
∆ROL ∆$zz,X
∆ROR ∆$zz,X

Byte length : 2

Cycle number : 6

Timing :



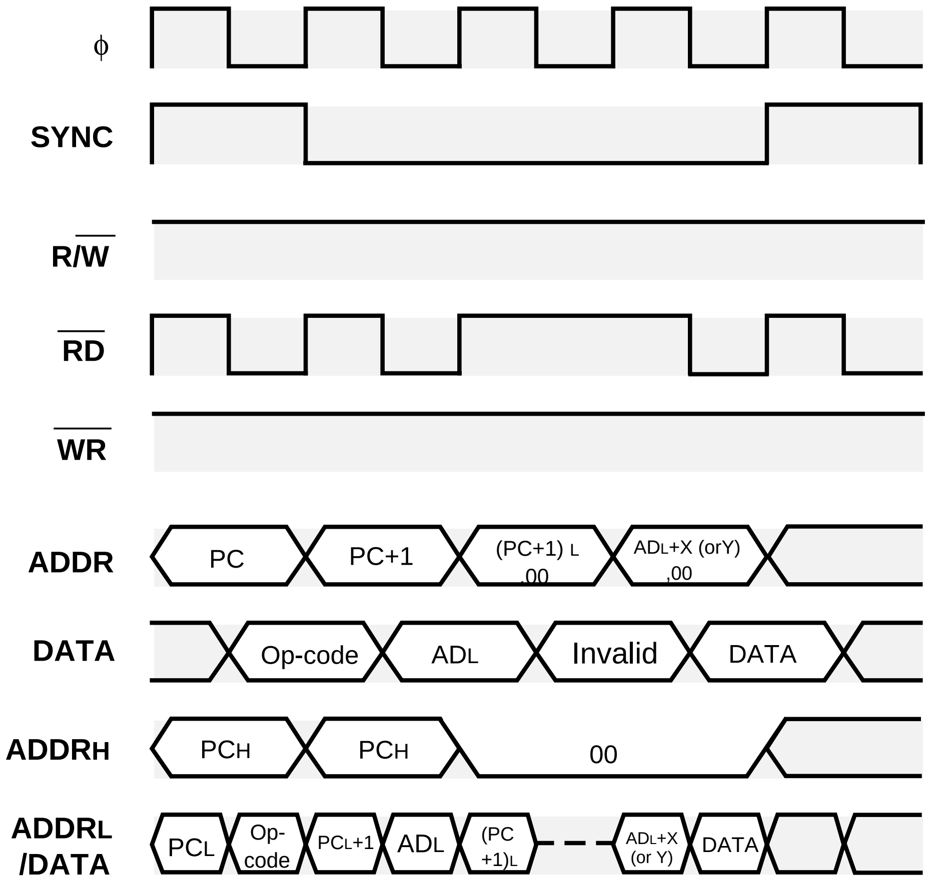
ZERO PAGE X, ZERO PAGE Y

Instructions : ∆ADC ∆$zz,X (T=0)
∆AND ∆$zz,X (T=0)
∆CMP ∆$zz,X (T=0)
∆EOR ∆$zz,X (T=0)
∆LDA ∆$zz,X (T=0)
∆LDX ∆$zz,Y
∆LDY ∆$zz,X
∆ORA ∆$zz,X (T=0)
∆SBC ∆$zz,X (T=0)

Byte length : **2**

Cycle number : **4**

Timing :

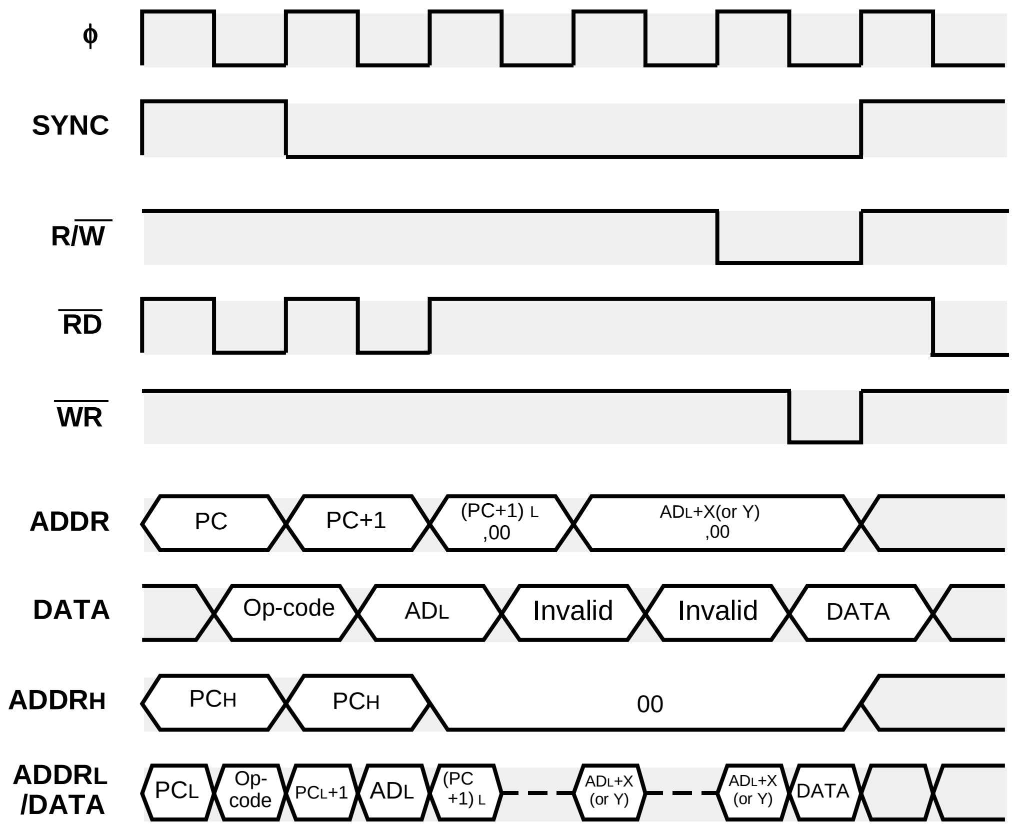


ZERO PAGE X, ZERO PAGE Y

Instructions : ∆STA∆$zz,X
∆STX∆$zz,Y
∆STY∆$zz,X

Byte length : 2
Cycle number : 5

Timing :



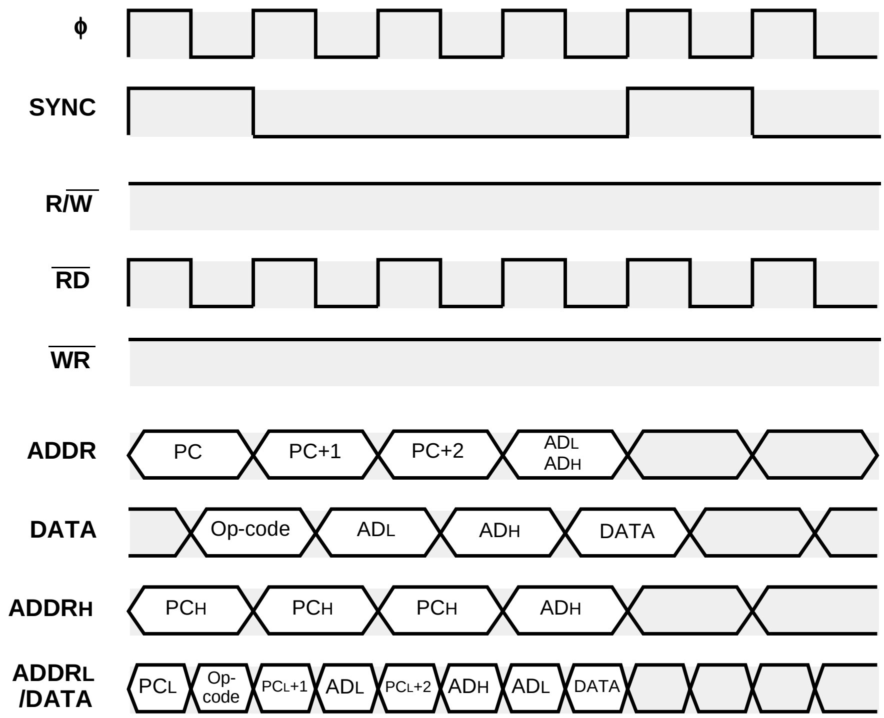
ABSOLUTE

Instructions : ∆ADC ∆$hhll (T=0)
∆AND ∆$hhll (T=0)
∆BIT ∆$hhll
∆CMP ∆$hhll (T=0)
∆CPX ∆$hhll
∆CPY ∆$hhll
∆EOR ∆$hhll (T=0)
∆LDA ∆$hhll (T=0)
∆LDX ∆$hhll
∆LDY ∆$hhll
∆ORA ∆$hhll (T=0)
∆SBC ∆$hhll (T=0)

Byte length : **3**

Cycle number : **4**

Timing :



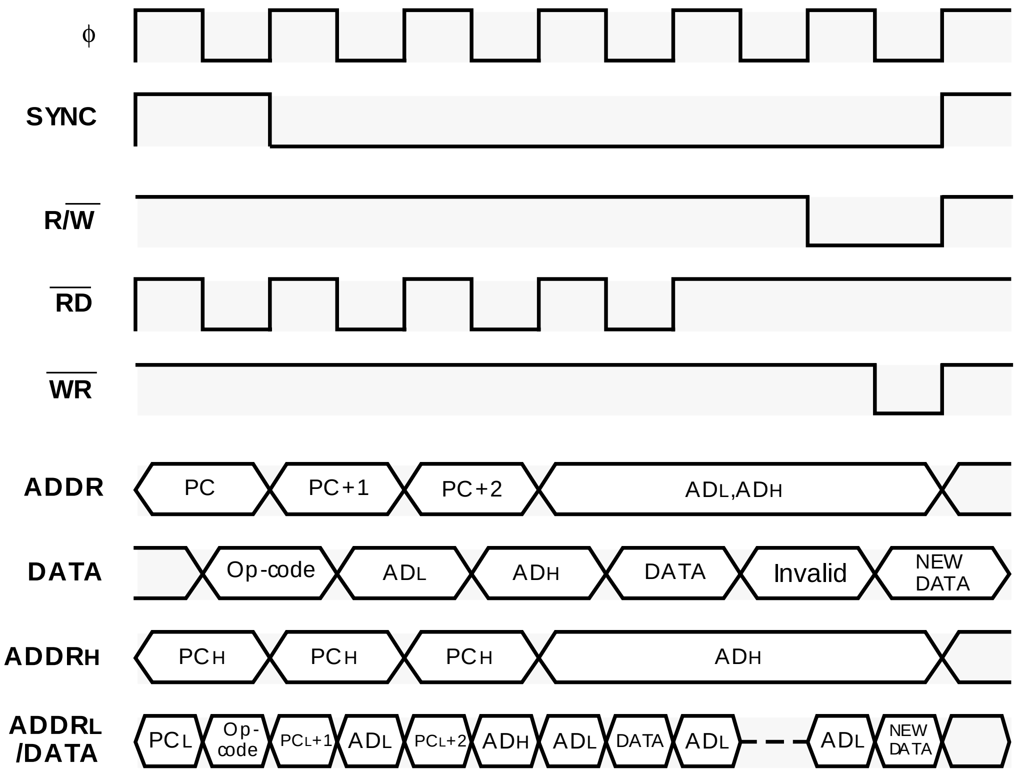
ABSOLUTE

Instructions : ∆**ASL** ∆**$hhll**
∆**DEC** ∆**$hhll**
∆**INC** ∆**$hhll**
∆**LSR** ∆**$hhll**
∆**ROL** ∆**$hhll**
∆**ROR**∆**$hhll**

Byte length : **3**

Cycle number : **6**

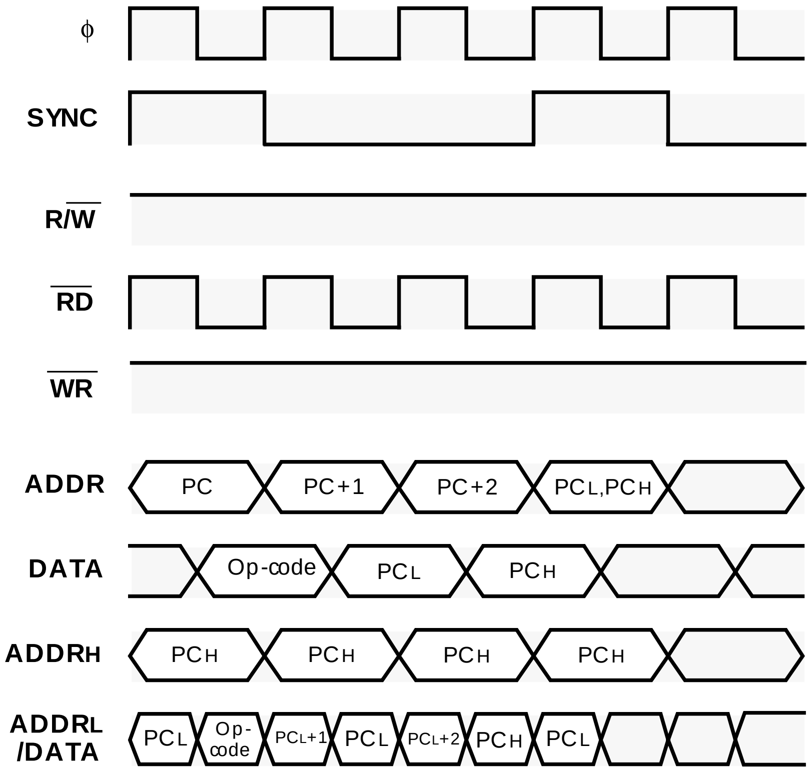
Timing :



ABSOLUTE

Instruction : ∆JMP∆$hhll
Byte length : **3**
Cycle number : **3**

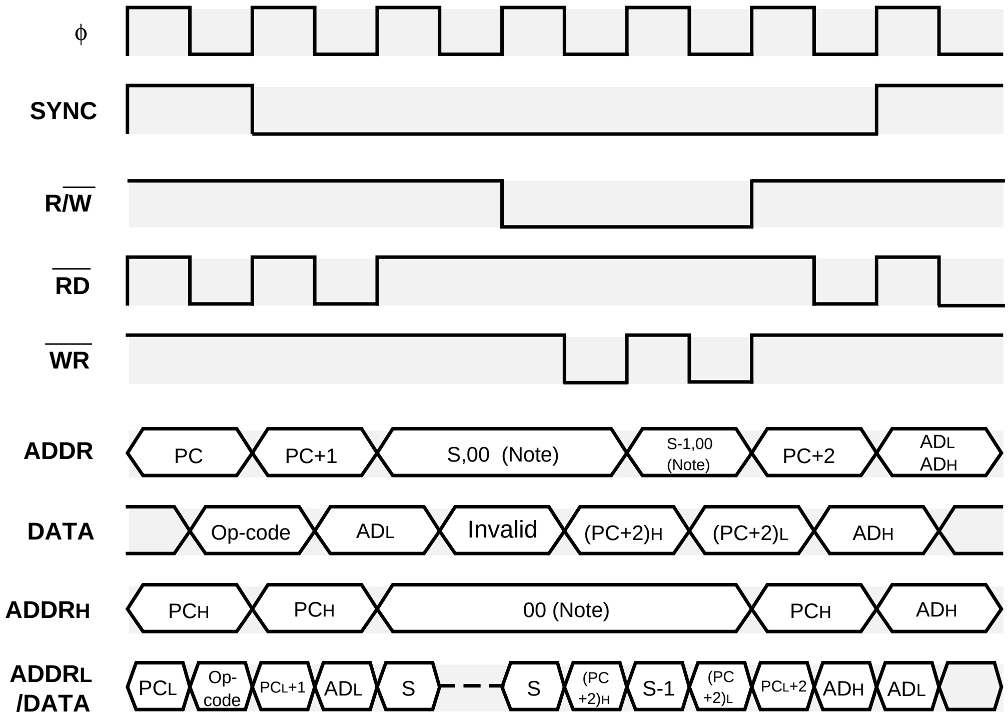
Timing :



ABSOLUTE

Instruction : **ΔJSRΔ$hhll**
Byte length : **3**
Cycle number : **6**

Timing :



Note: Some products are "01" or content of SPS flag.

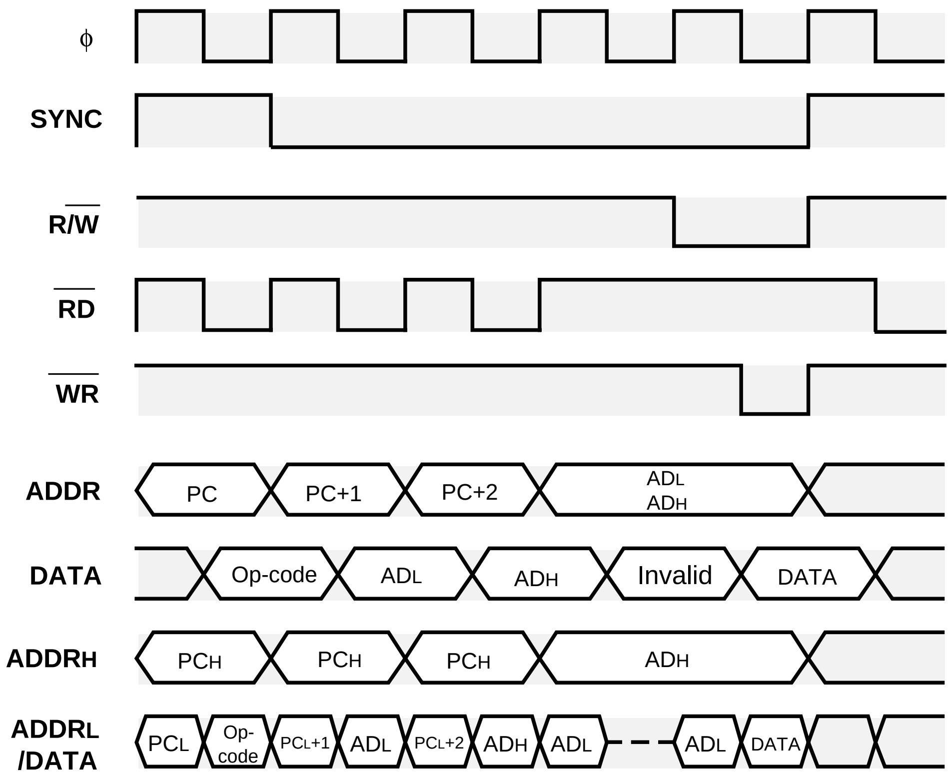
ABSOLUTE

Instructions : **ΔSTAΔ$hhll**
ΔSTXΔ$hhll
ΔSTYΔ$hhll

Byte length : **3**

Cycle number : **5**

Timing :



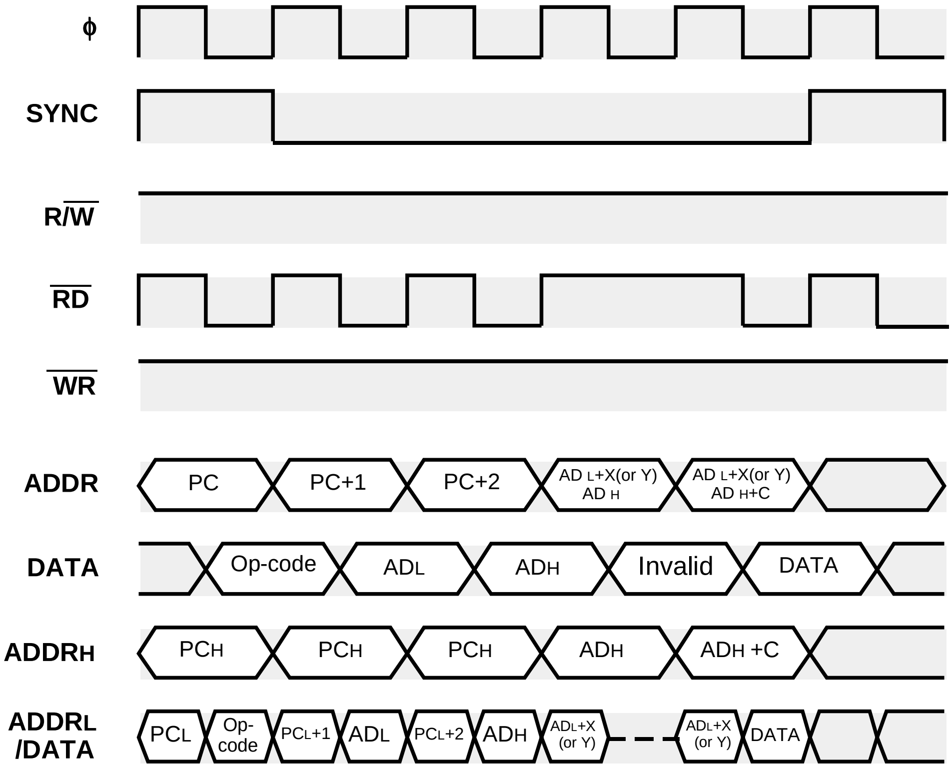
ABSOLUTE X, ABSOLUTE Y

Instructions : ∆ADC ∆$hhll,X or Y (T=0)
∆AND ∆$hhll,X or Y (T=0)
∆CMP ∆$hhll,X or Y (T=0)
∆EOR ∆$hhll,X or Y (T=0)
∆LDA ∆$hhll,X or Y (T=0)
∆LDX ∆$hhll,Y
∆LDY ∆$hhll,X
∆ORA ∆$hhll,X or Y (T=0)
∆SBC ∆$hhll,X or Y (T=0)

Byte length : **3**

Cycle number : **5**

Timing :



C : Carry of ADL+X or Y

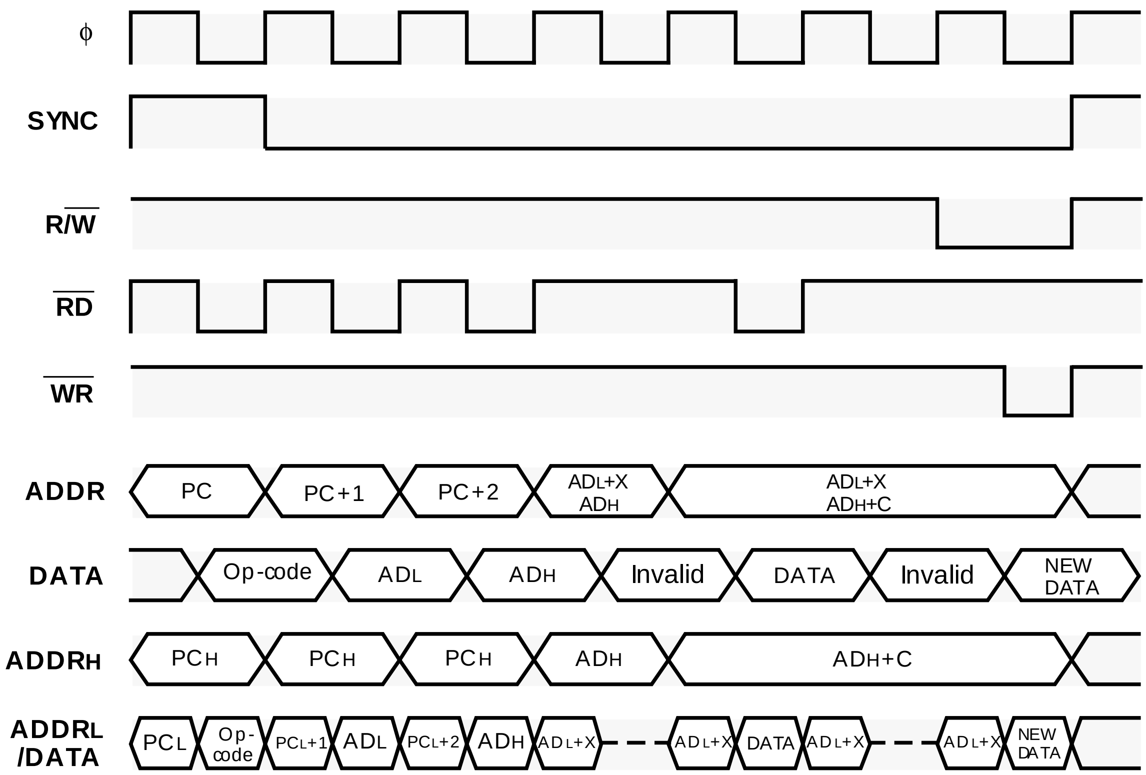
ABSOLUTE X

Instructions : ΔASL Δ$hhll,X
ΔDEC Δ$hhll,X
ΔINC Δ$hhll,X
ΔLSR Δ$hhll,X
ΔROL Δ$hhll,X
ΔROR Δ$hhll,X

Byte length : **3**

Cycle number : **7**

Timing :

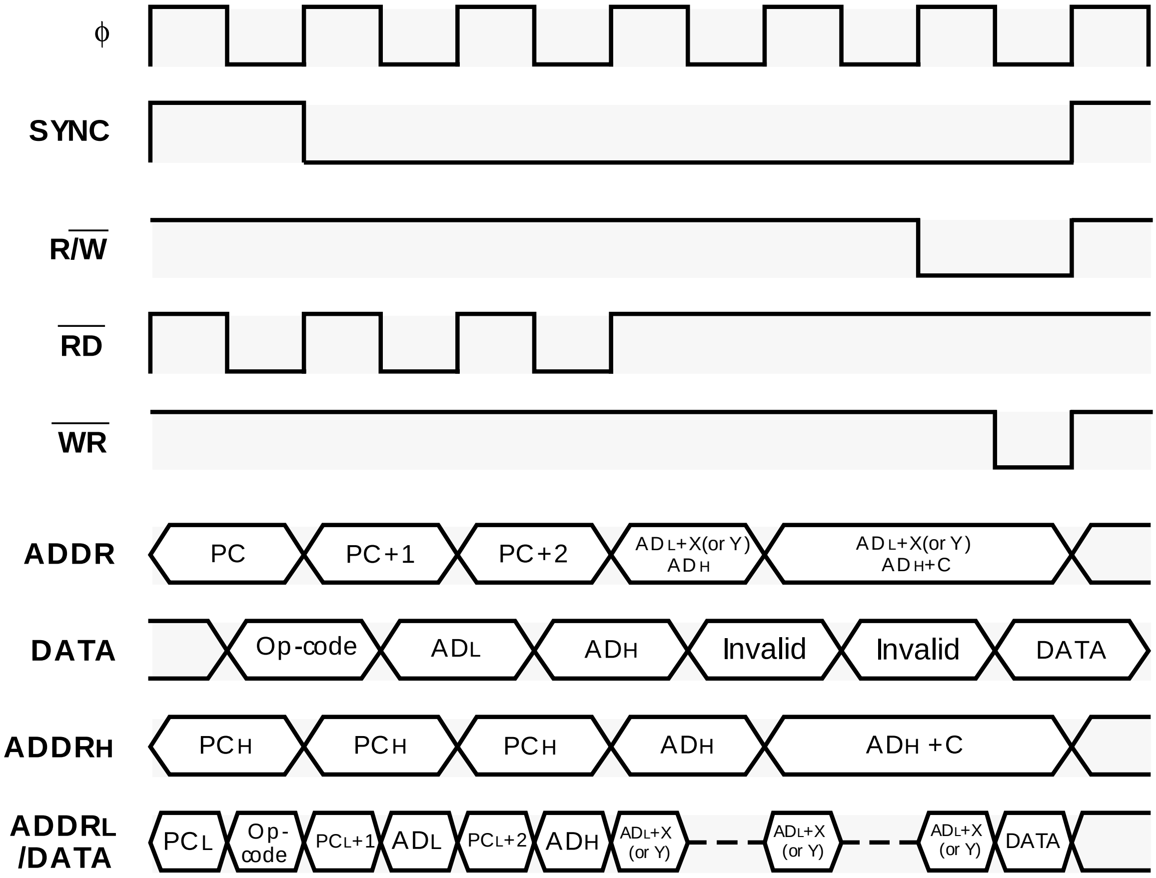


C : Carry of ADL+X

ABSOLUTE X, ABSOLUTE Y

Instruction : **∆STA∆$hhll,X or Y**
Byte length : **3**
Cycle number : **6**

Timing :

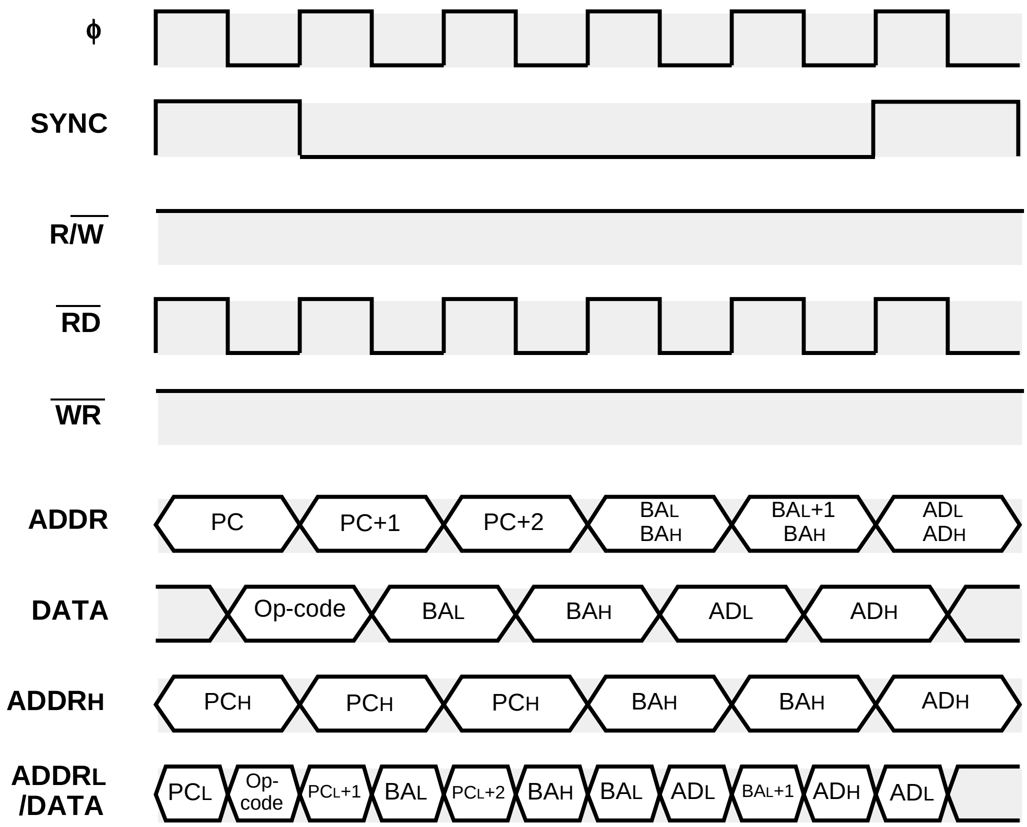


C : Carry of ADL+X or Y

INDIRECT

Instruction : ∆JMP∆($hhll)
Byte length : **3**
Cycle number : **5**

Timing :

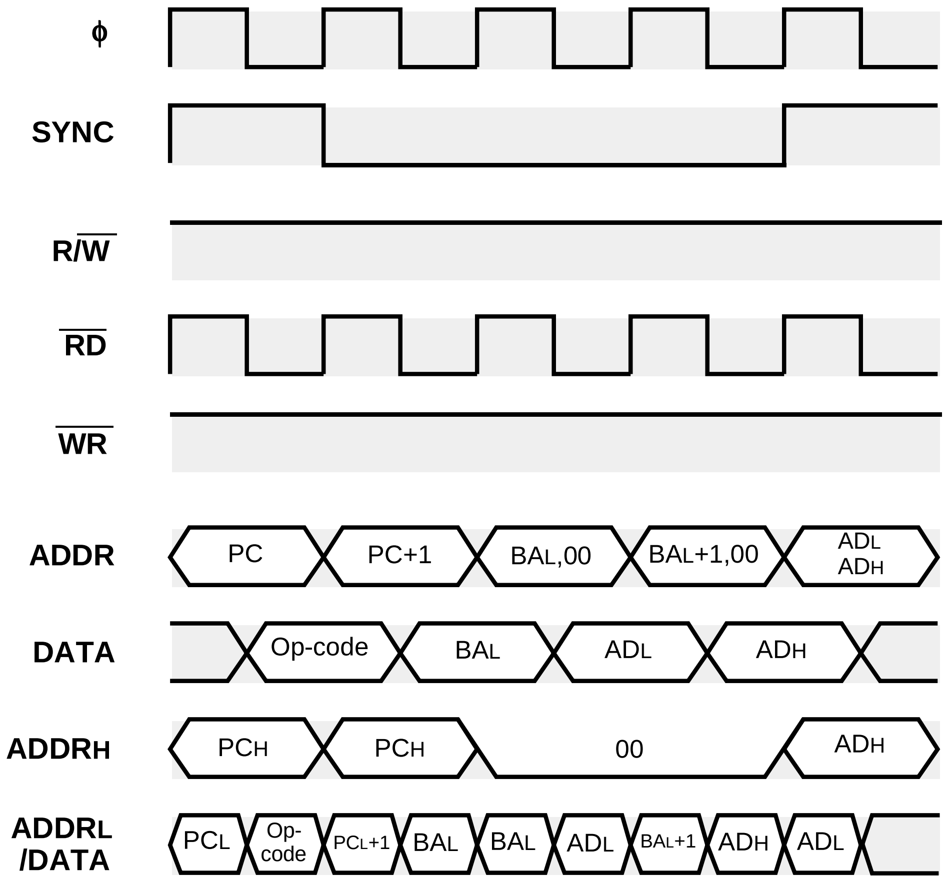


BA : Basic address

ZERO PAGE INDIRECT

Instruction : ΔJMPΔ($zz)
Byte length : 2
Cycle number : 4

Timing :

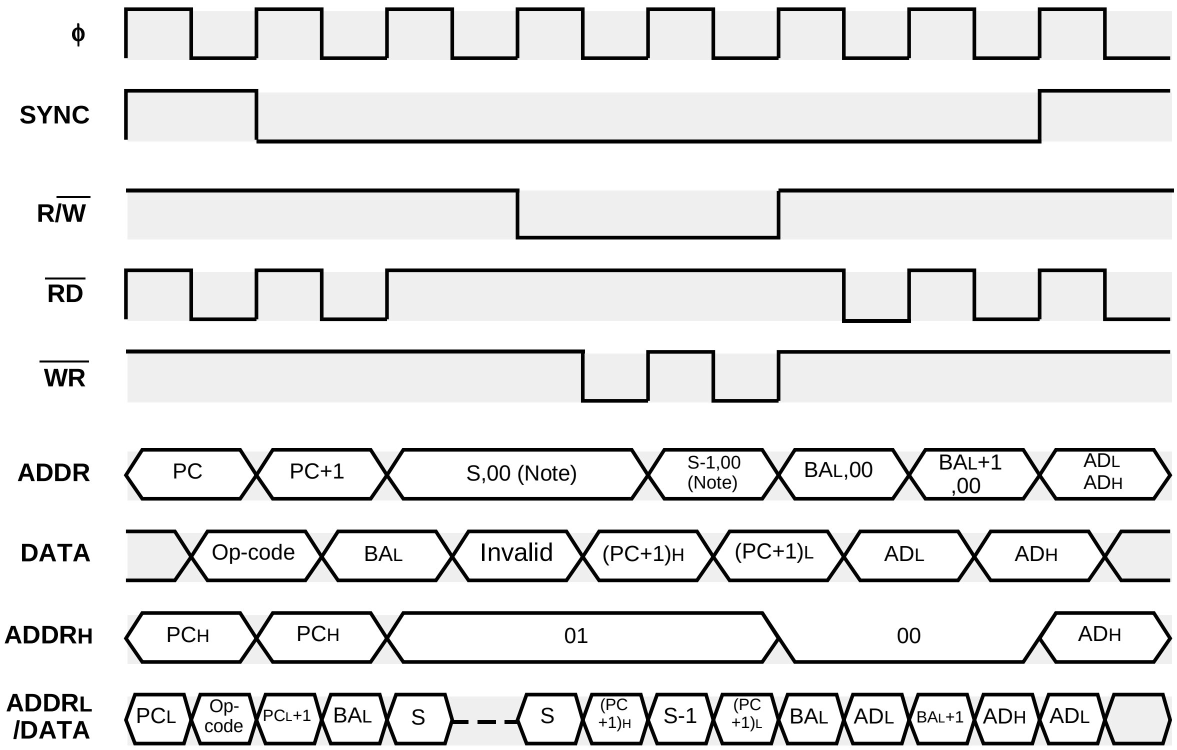


BA : Basic address

ZERO PAGE INDIRECT

Instruction : ∆JSR∆($zz)
Byte length : **2**
Cycle number : **7**

Timing :

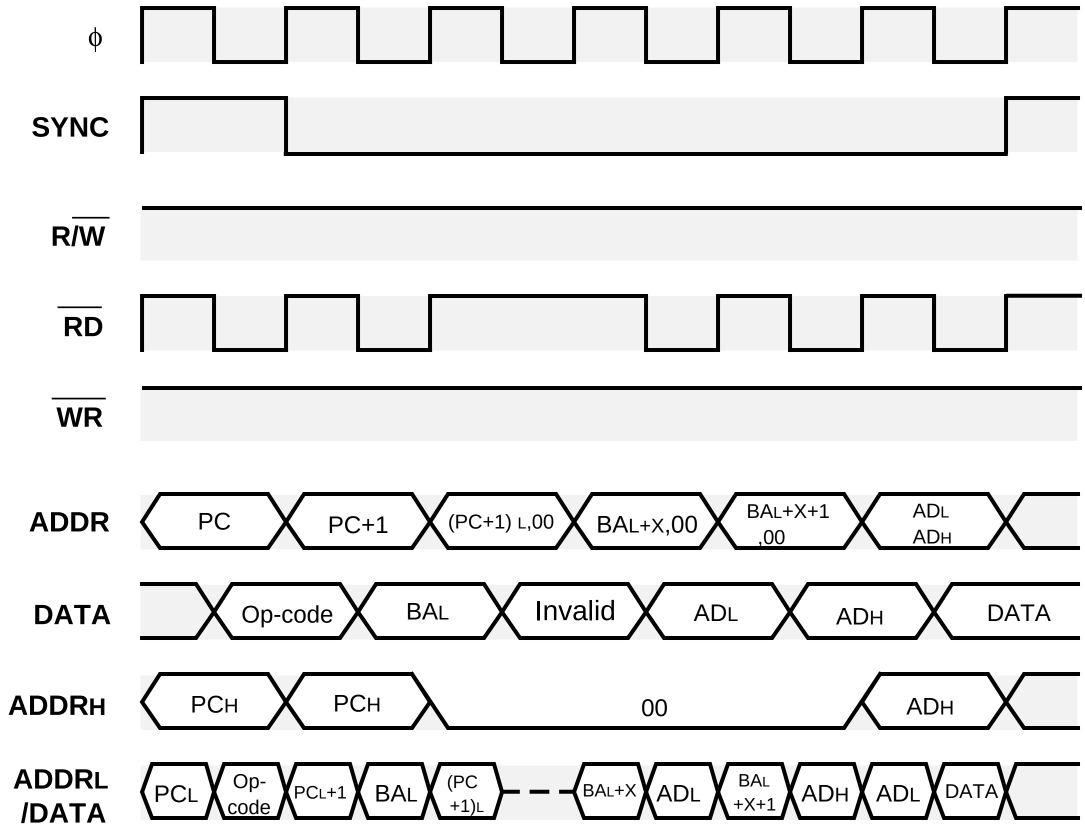


BA : Basic address

Note: Some kind types are "01" or content of SPS flag.

INDIRECT X

Instructions	: ΔADC Δ($zz,X)	(T=0)
	ΔAND Δ($zz,X)	(T=0)
	ΔCMP Δ($zz,X)	(T=0)
	ΔEOR Δ($zz,X)	(T=0)
	ΔLDA Δ($zz,X)	(T=0)
	ΔORA Δ($zz,X)	(T=0)
	ΔSBC Δ($zz,X)	(T=0)
Byte length	: 2	
Cycle number	: 6	
Timing	:	

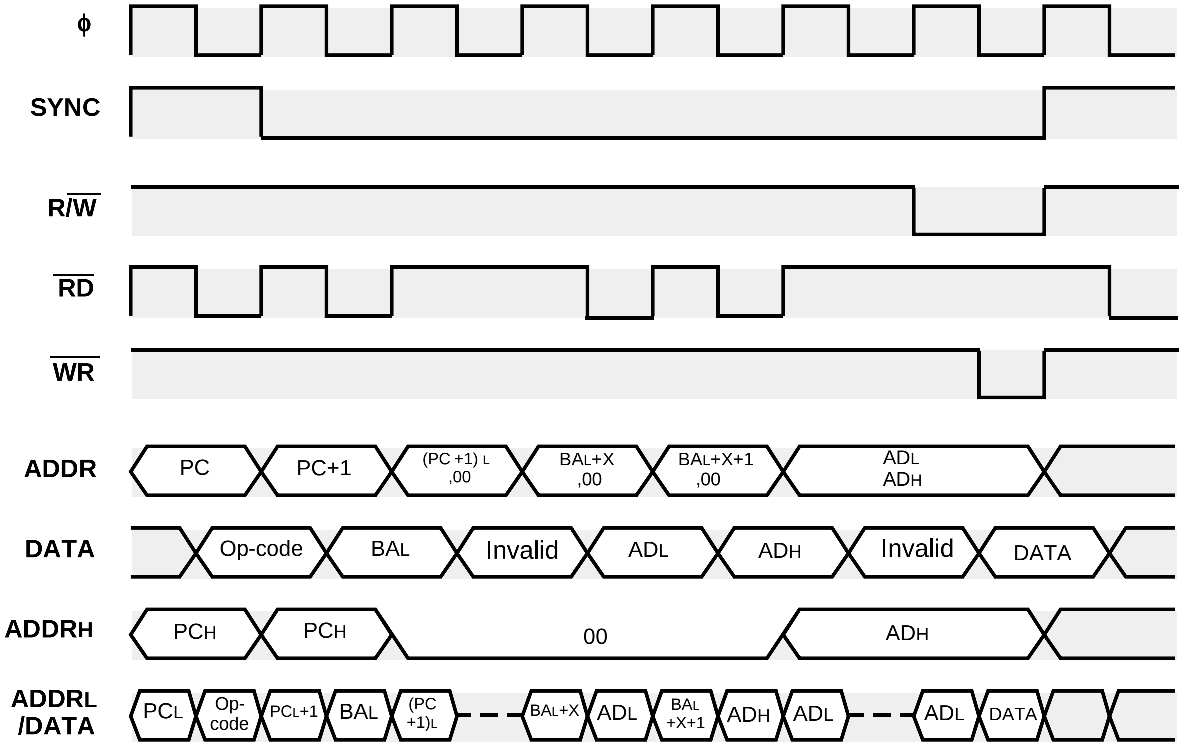


BA : Basic address

INDIRECT X

Instruction : ∆STA∆($zz,X)
Byte length : **2**
Cycle number : **7**

Timing :



BA : Basic address

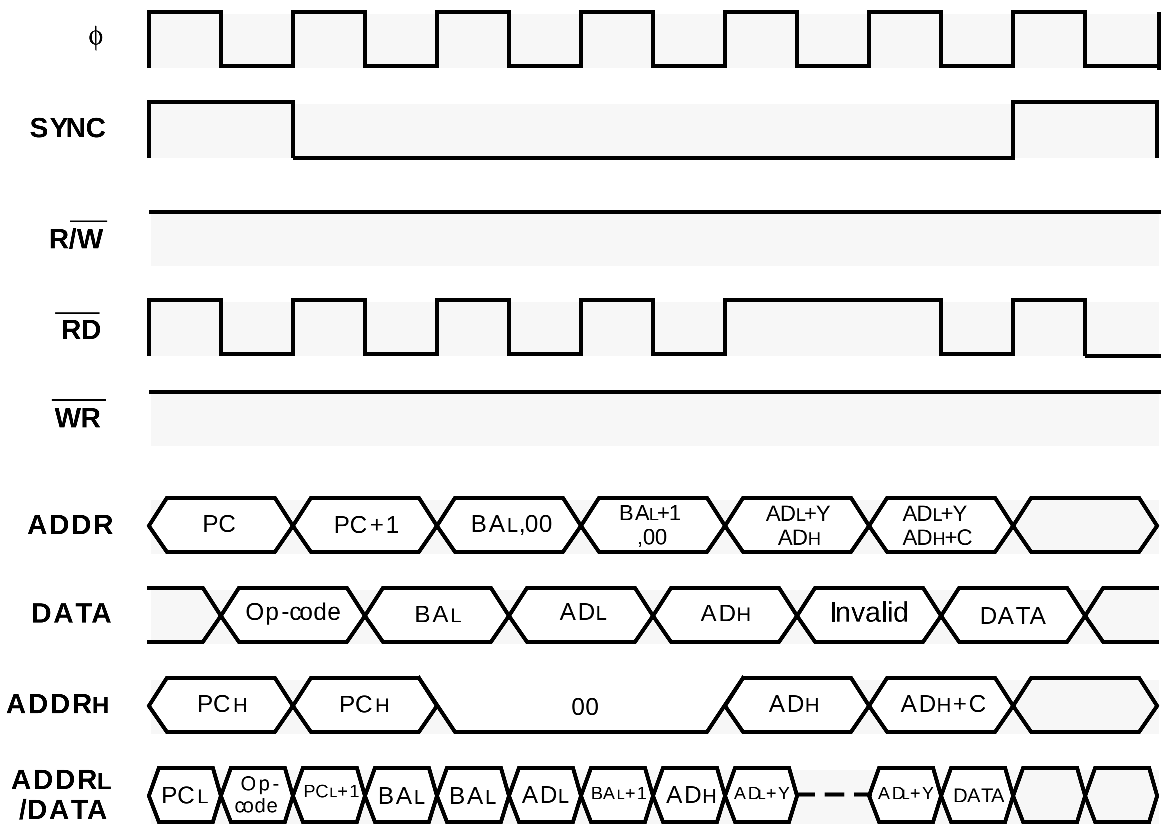
INDIRECT Y

Instructions : ∆**ADC**∆**($zz),Y** **(T=0)**
∆**AND**∆**($zz),Y** **(T=0)**
∆**CMP**∆**($zz),Y** **(T=0)**
∆**EOR**∆**($zz),Y** **(T=0)**
∆**LDA** ∆**($zz),Y** **(T=0)**
∆**ORA**∆**($zz),Y** **(T=0)**
∆**SBC** ∆**($zz),Y** **(T=0)**

Byte length : **2**

Cycle number : **6**

Timing :



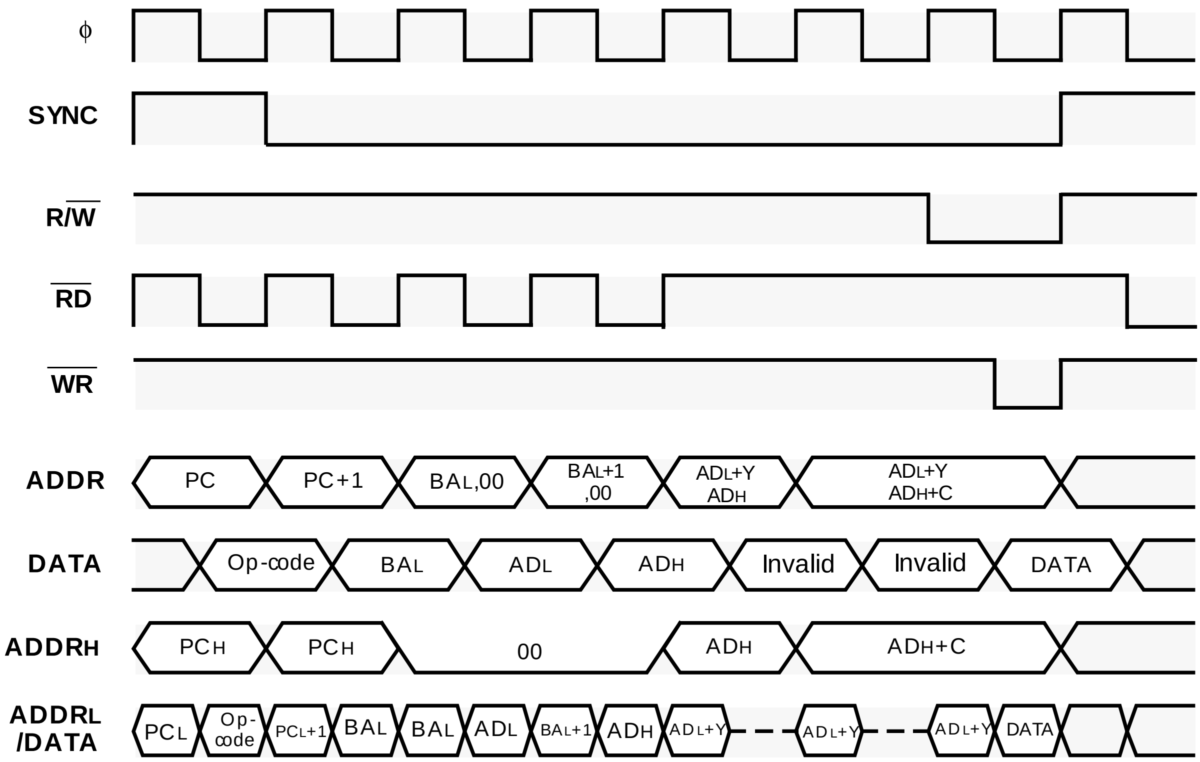
BA : Basic address

C : Carry of ADL+Y

INDIRECT Y

Instruction : ∆STA∆($zz),Y
Byte length : **2**
Cycle number : **7**

Timing :



BA : Basic address

C : Carry of ADL+Y

RELATIVE

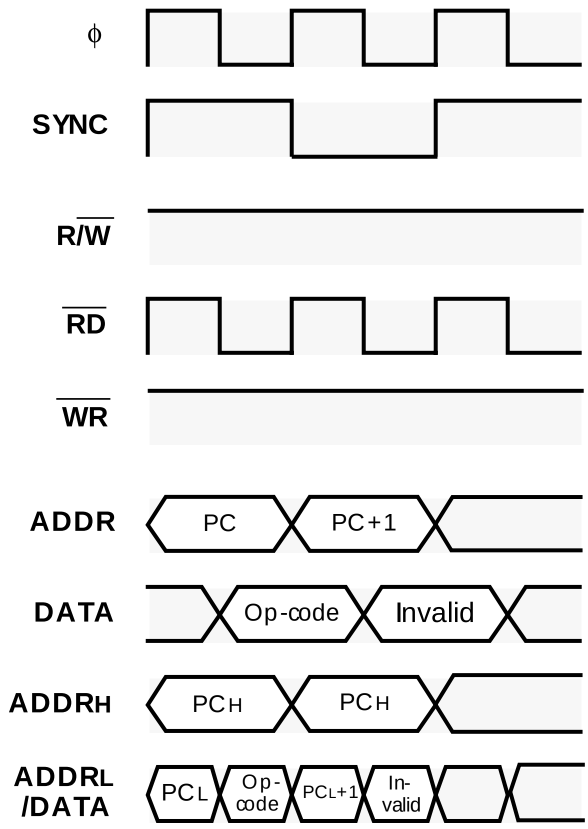
Instructions : ΔBCC Δ$hhll
ΔBCS Δ$hhll
ΔBEQ Δ$hhll
ΔBMI Δ$hhll
ΔBNE Δ$hhll
ΔBPL Δ$hhll
ΔBVC Δ$hhll
ΔBVS Δ$hhll

Byte length : **2**

(1)With no branch

Cycle number : **2**

Timing :

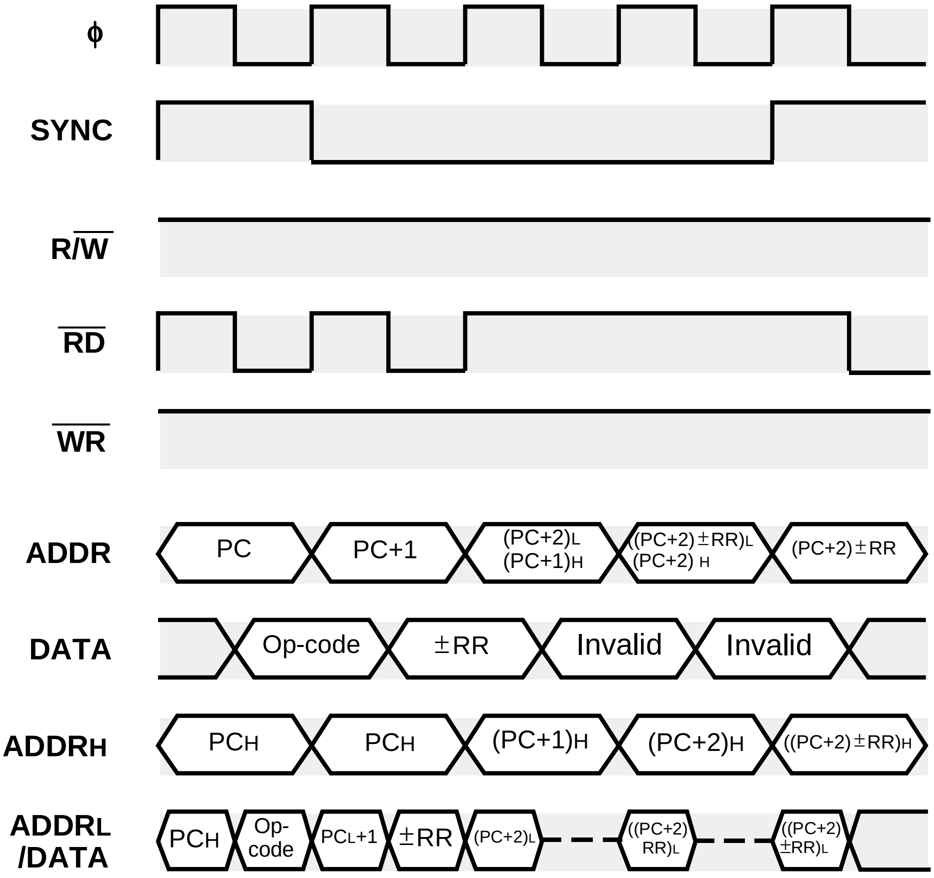


Instructions : ΔBCC Δ$hhll
ΔBCS Δ$hhll
ΔBEQ Δ$hhll
ΔBMI Δ$hhll
ΔBNE Δ$hhll
ΔBPL Δ$hhll
ΔBVC Δ$hhll
ΔBVS Δ$hhll

Byte length : **2**

(2)With branch
Cycle number : **4**

Timing :

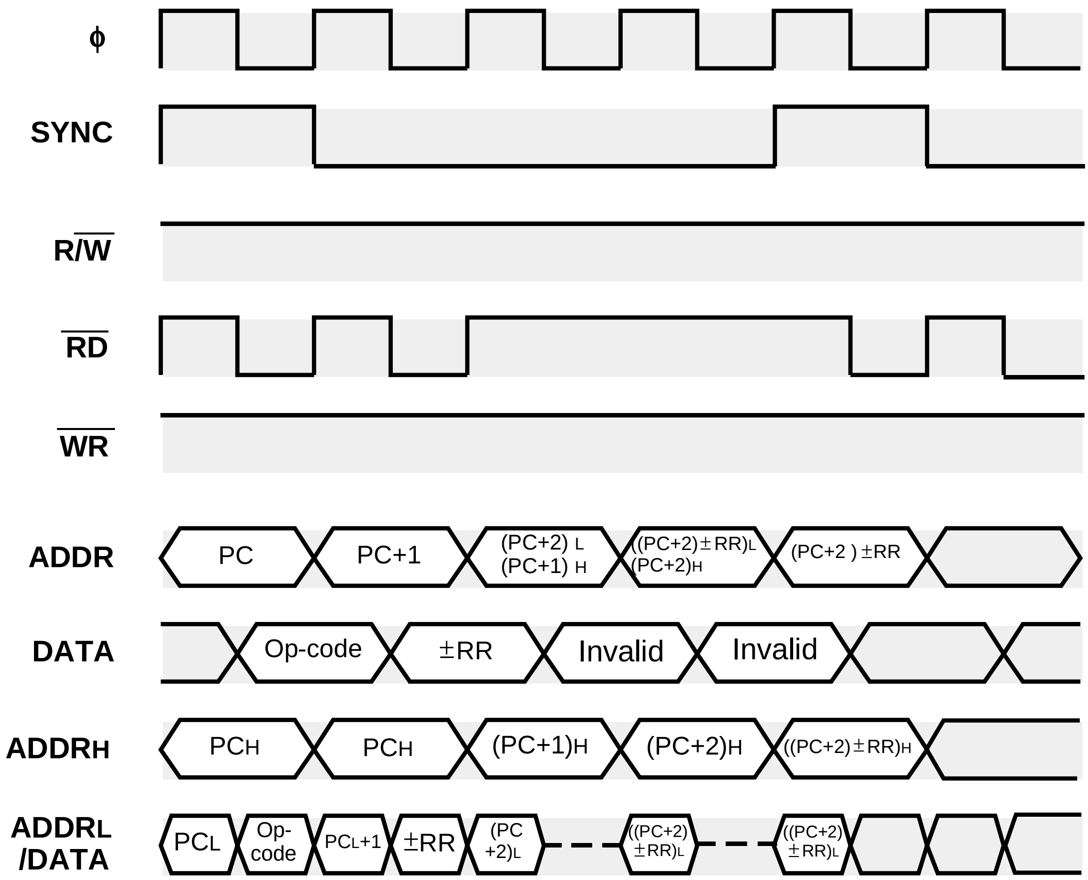


RR : Offset value

RELATIVE

Instruction : ∆**BRA**∆**$hhll**
Byte length : **2**
Cycle number : **4**

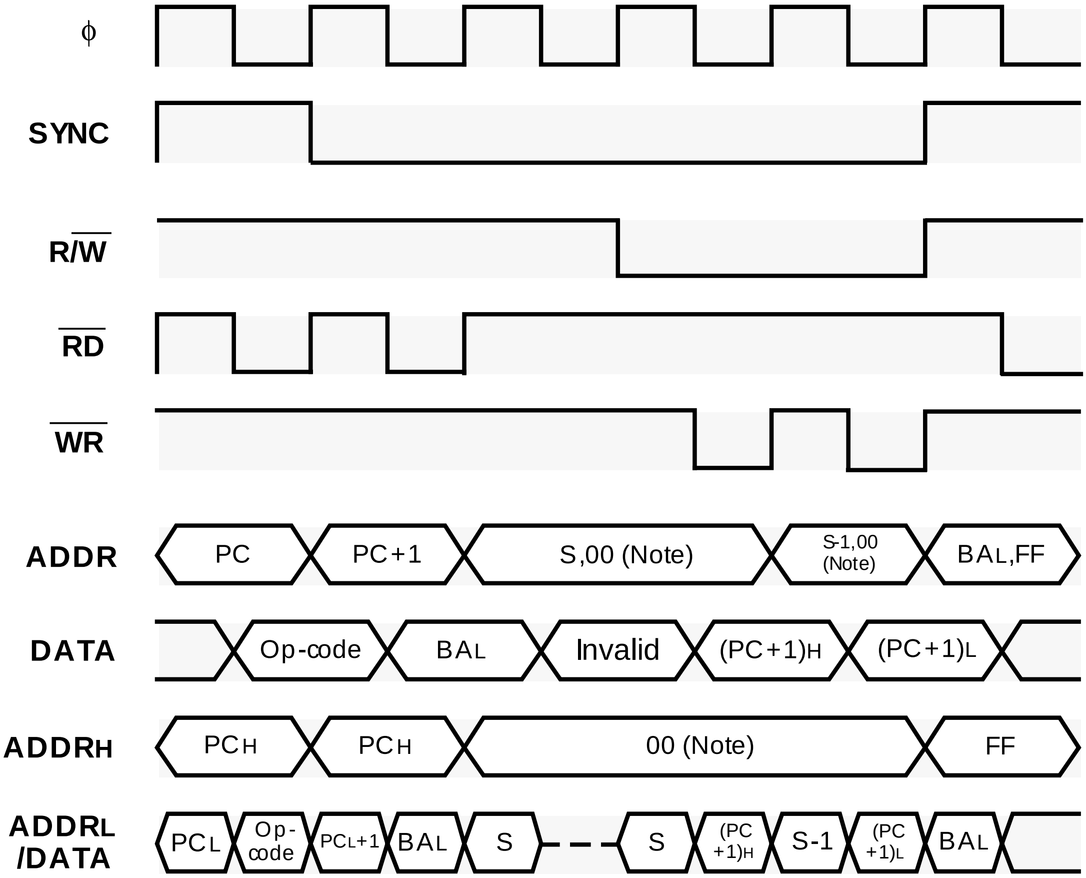
Timing :



RR : Offset value

Instruction : ∆JSR∆\$hhll
Byte length : **2**
Cycle number : **5**

Timing :

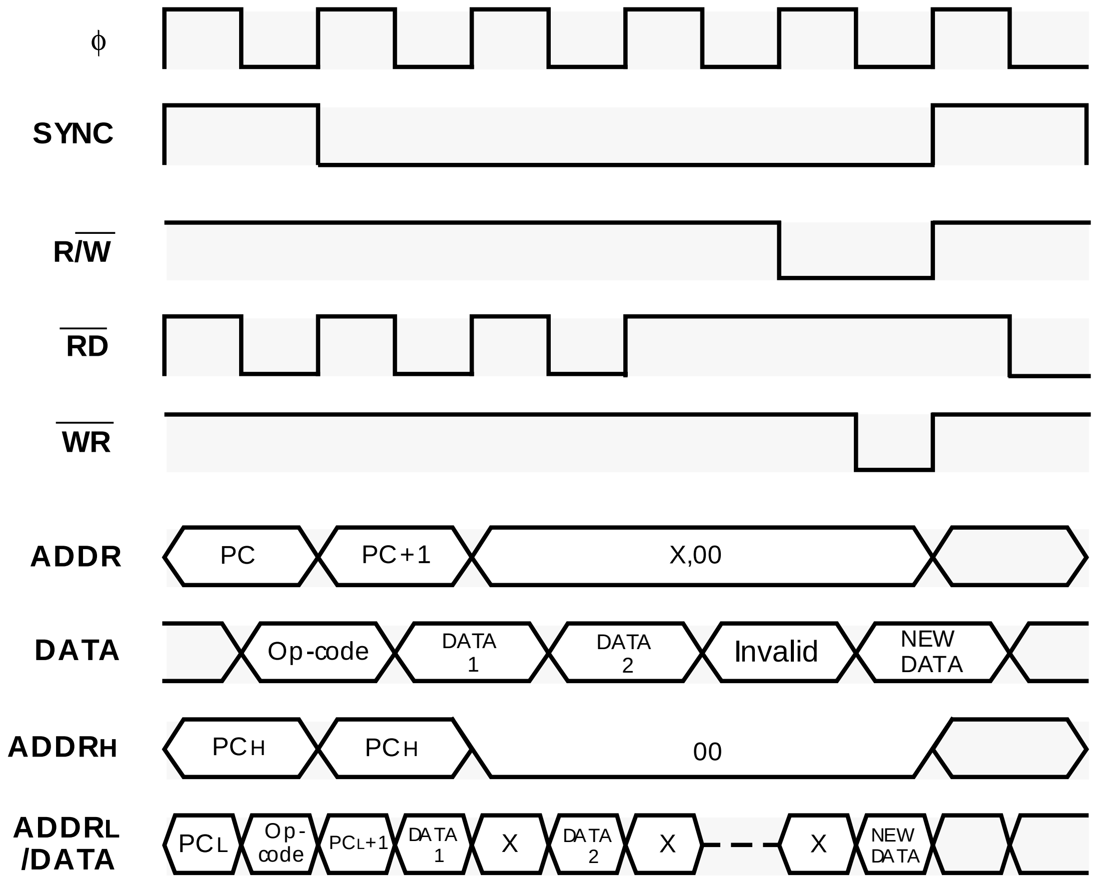


BA : Basic address

Note : Some products are “01” or content of SPS flag.

IMMEDIATE

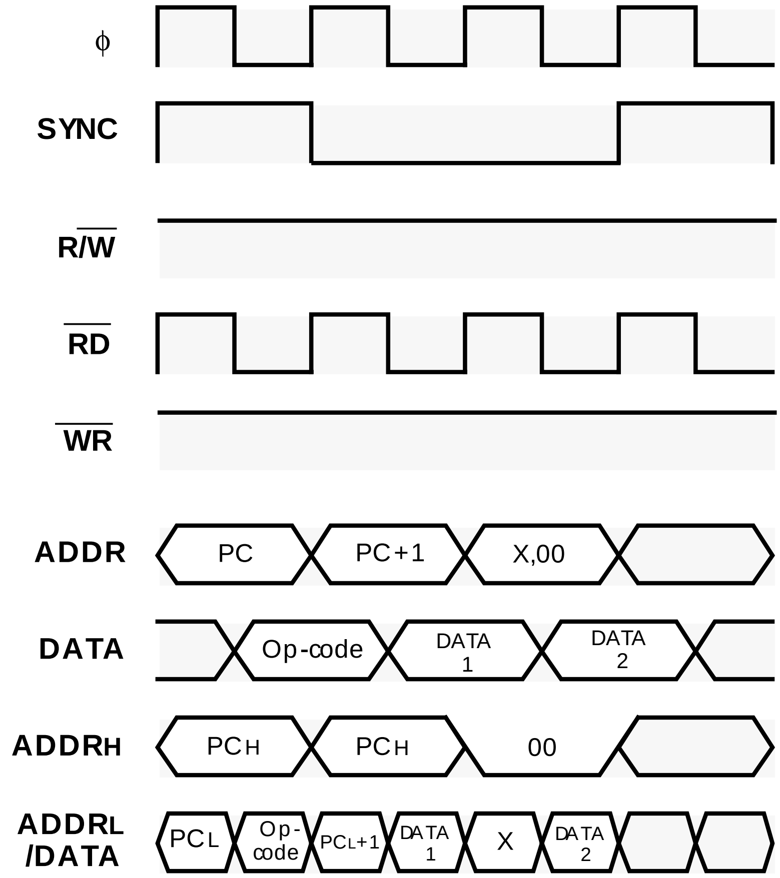
Instructions	: ∆ADC∆#$nn (T=1)
	∆AND∆#$nn (T=1)
	∆EOR∆#$nn (T=1)
	∆ORA∆#$nn (T=1)
	∆SBC∆#$nn (T=1)
Byte length	: 2
Cycle number	: 5
Timing	:



IMMEDIATE

Instruction : ∆CMP∆#$nn (T=1)
Byte length : 2
Cycle number : 3

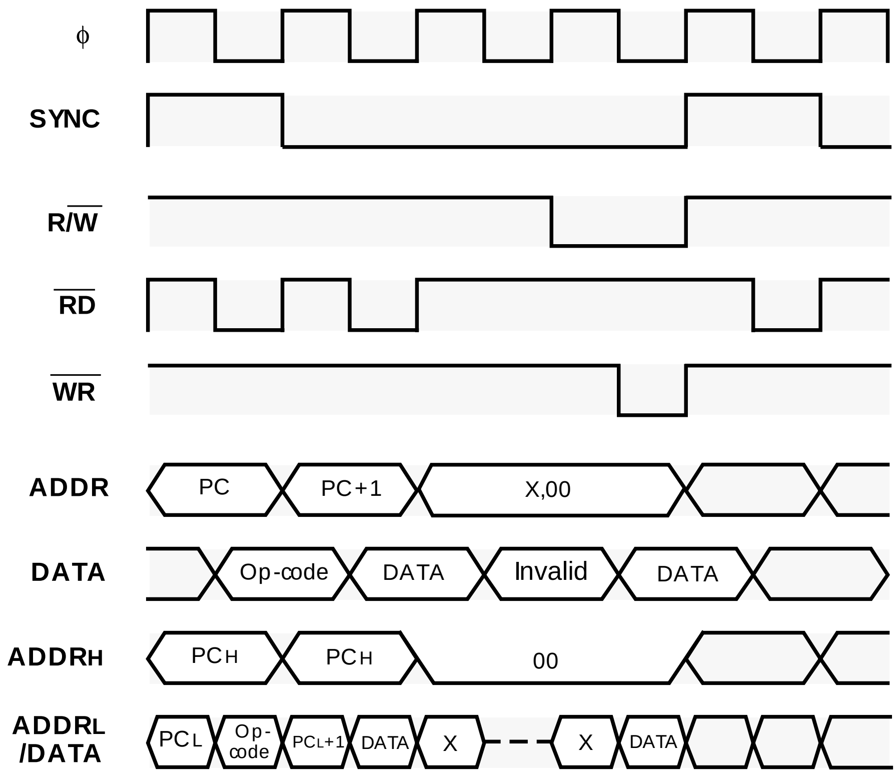
Timing :



IMMEDIATE

Instruction : ΔLDAΔ#$nn (T=1)
Byte length : 2
Cycle number : 4

Timing :



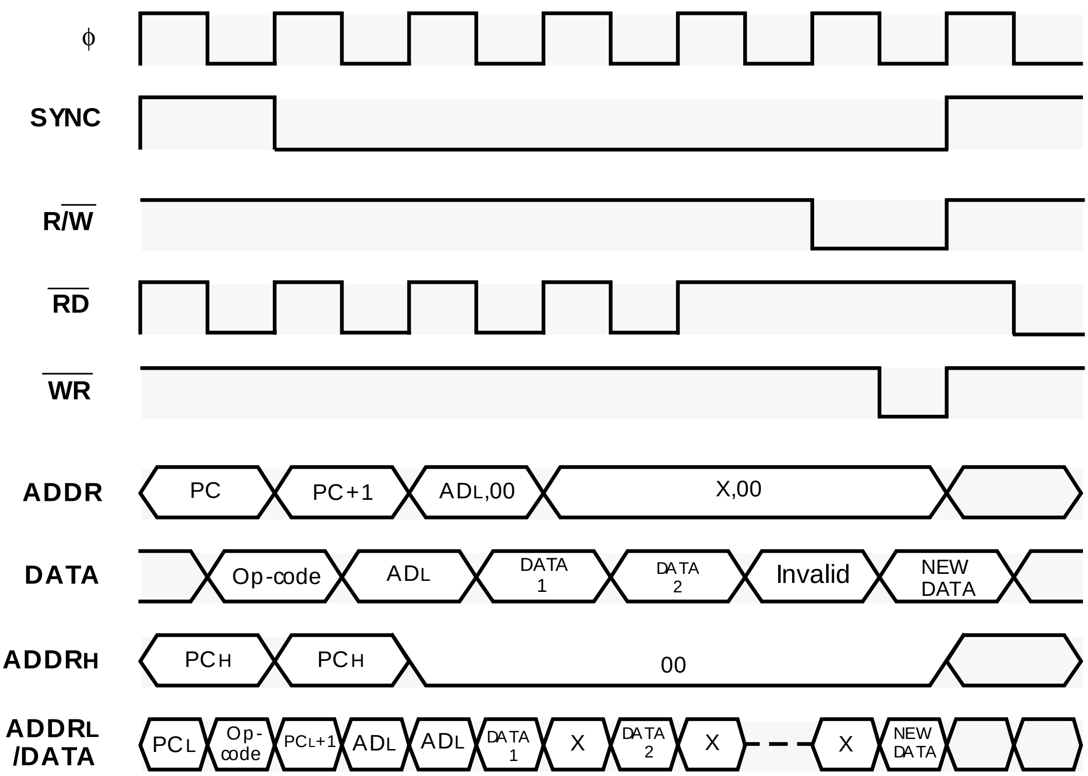
ZERO PAGE

Instructions : ∆ADC∆$zz (T=1)
∆AND∆$zz (T=1)
∆EOR∆$zz (T=1)
∆ORA∆$zz (T=1)
∆SBC∆$zz (T=1)

Byte length : **2**

Cycle number : **6**

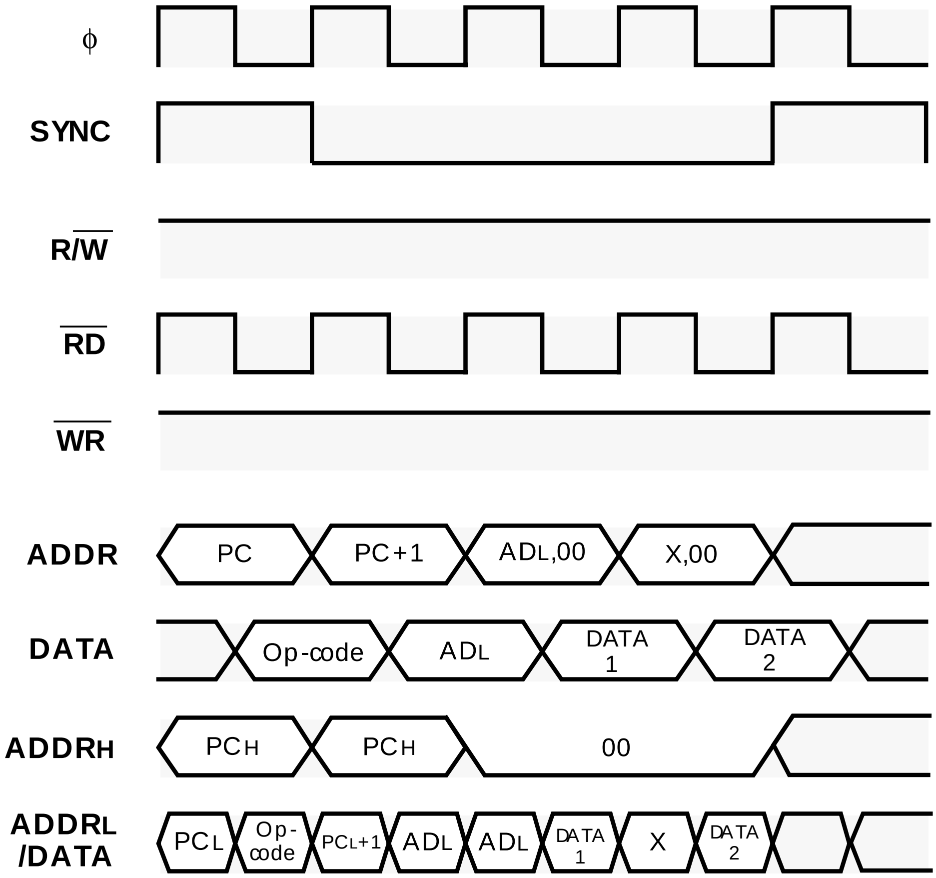
Timing :



ZERO PAGE

Instruction : ∆CMP∆$zz (T=1)
Byte length : 2
Cycle number : 4

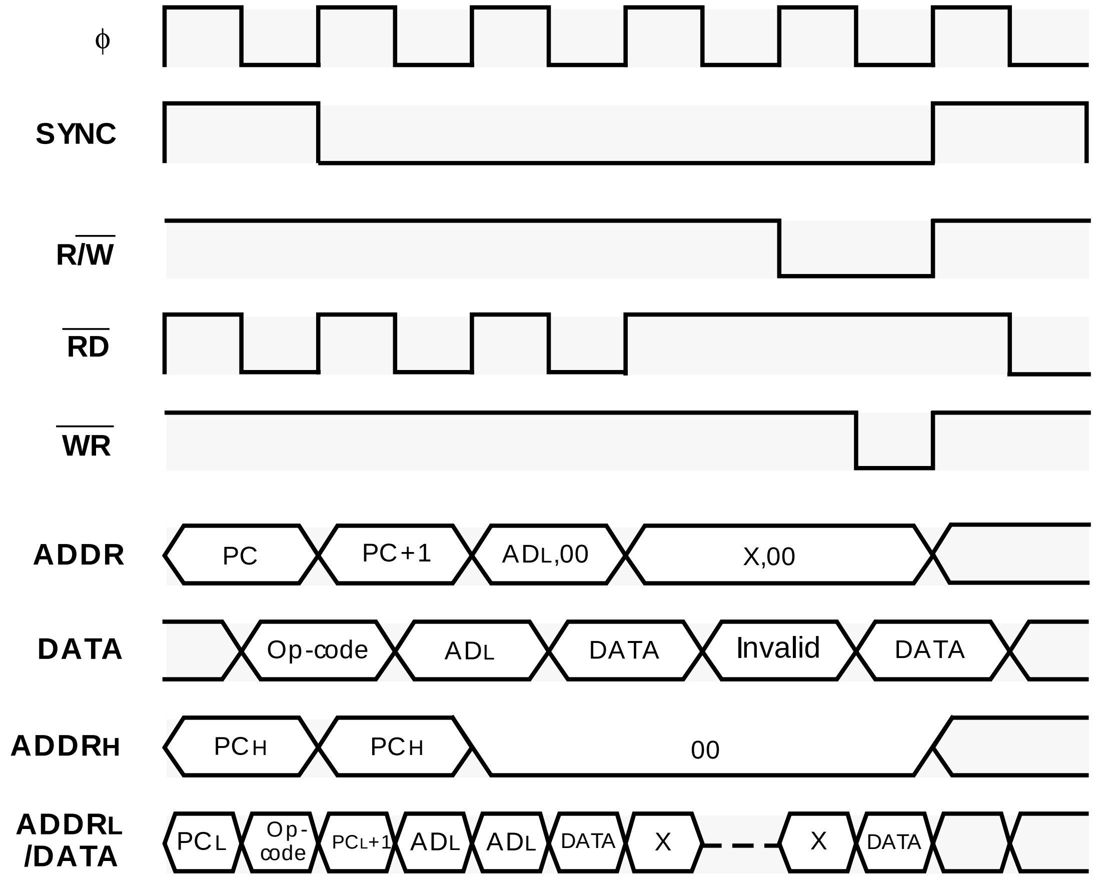
Timing :



ZERO PAGE

Instruction : ΔLDAΔ$zz (T=1)
Byte length : **2**
Cycle number : **5**

Timing :



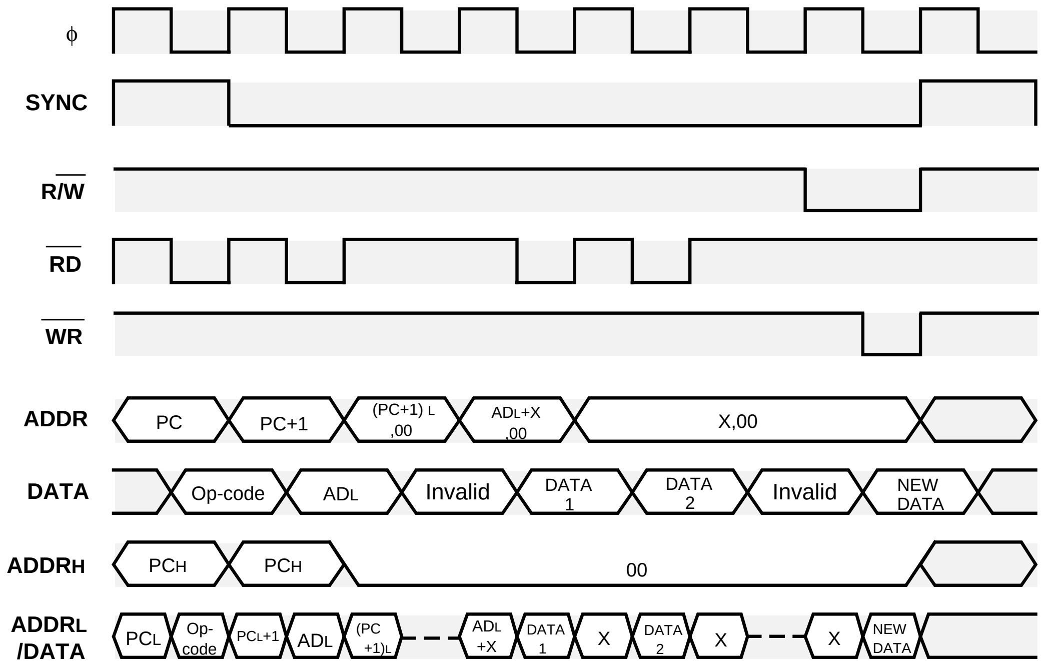
ZERO PAGE X

Instructions : **ΔADCΔ$zz,X (T=1)**
ΔANDΔ$zz,X (T=1)
ΔEORΔ$zz,X (T=1)
ΔORAΔ$zz,X (T=1)
ΔSBCΔ$zz,X (T=1)

Byte length : **2**

Cycle number : **7**

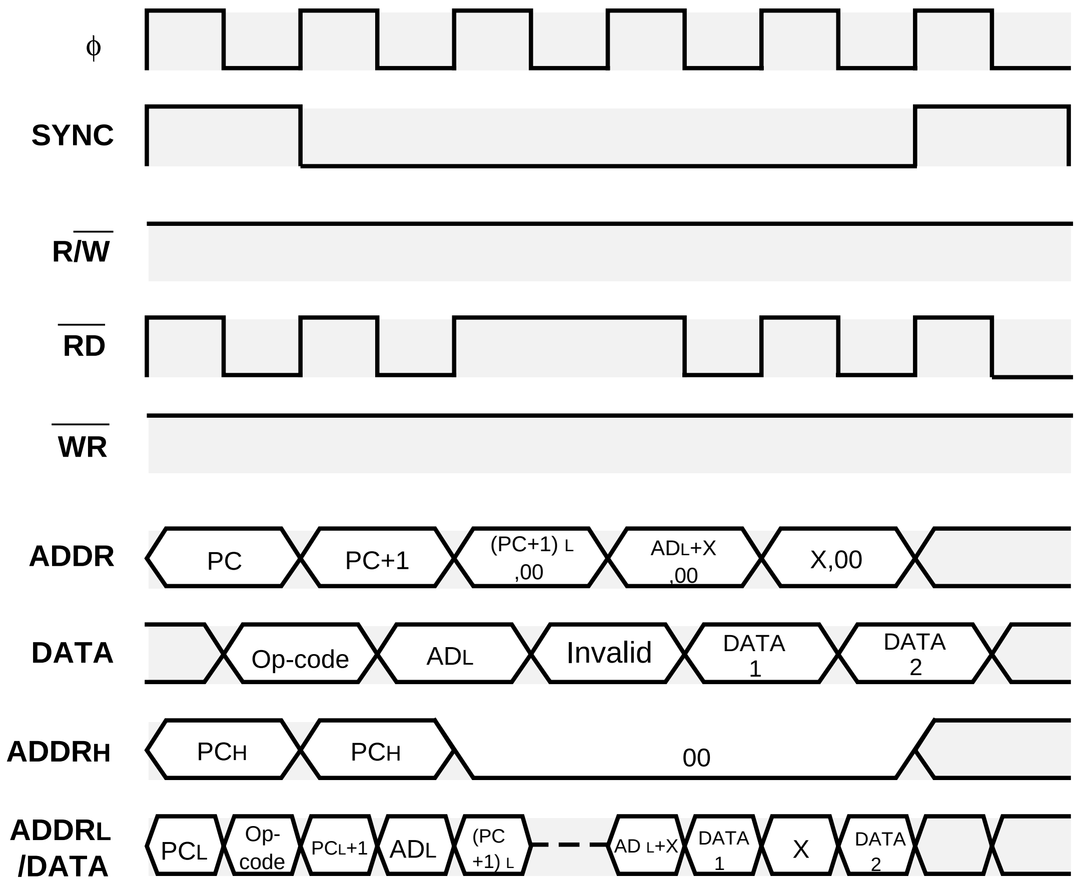
Timing :



ZERO PAGE X

Instruction : ΔCMPΔ$zz,X (T=1)
Byte length : **2**
Cycle number : **5**

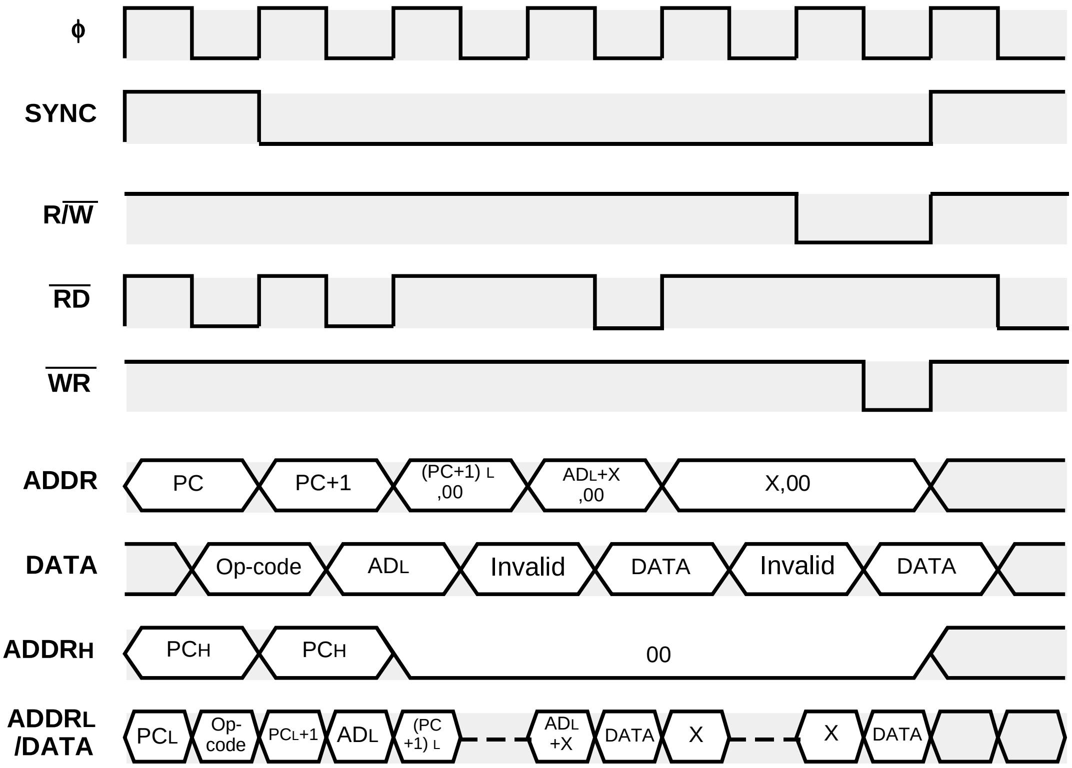
Timing :



ZERO PAGE X

Instruction : ΔLDAΔ$zz,X (T=1)
Byte length : 2
Cycle number : 6

Timing :



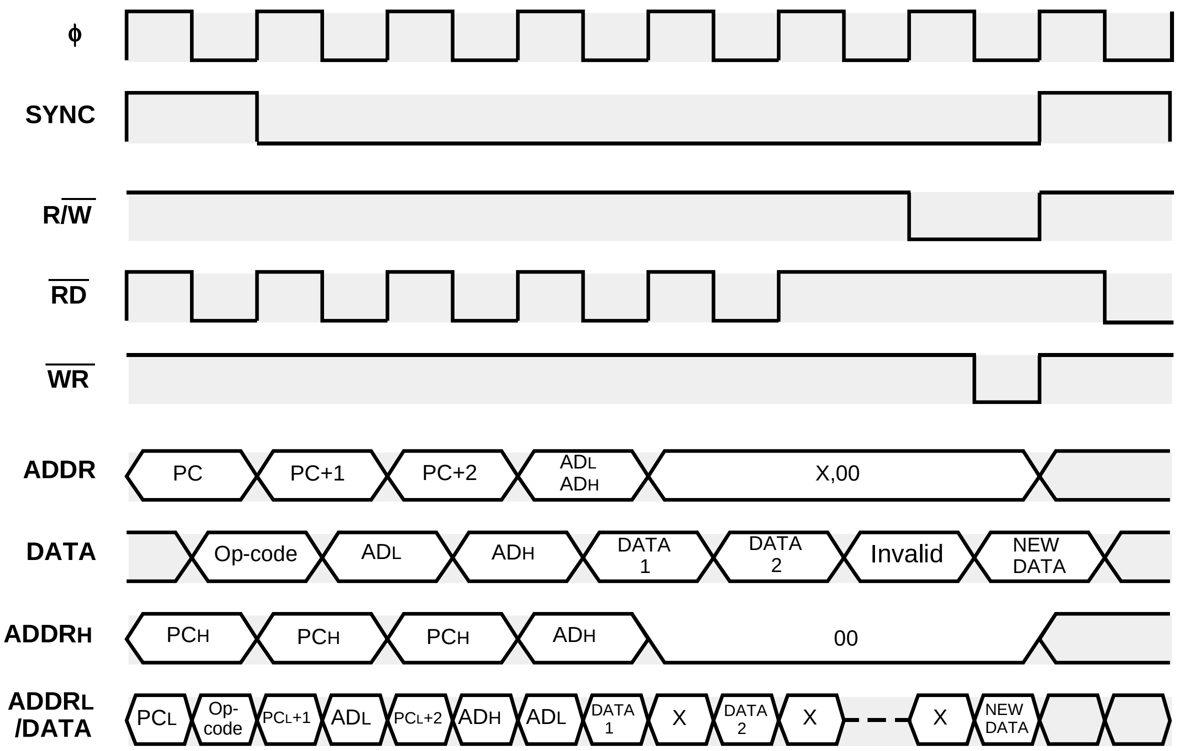
ABSOLUTE

Instructions : ΔADCΔ$hhll (T=1)
ΔANDΔ$hhll (T=1)
ΔEORΔ$hhll (T=1)
ΔORAΔ$hhll (T=1)
ΔSBCΔ$hhll (T=1)

Byte length : **3**

Cycle number : **7**

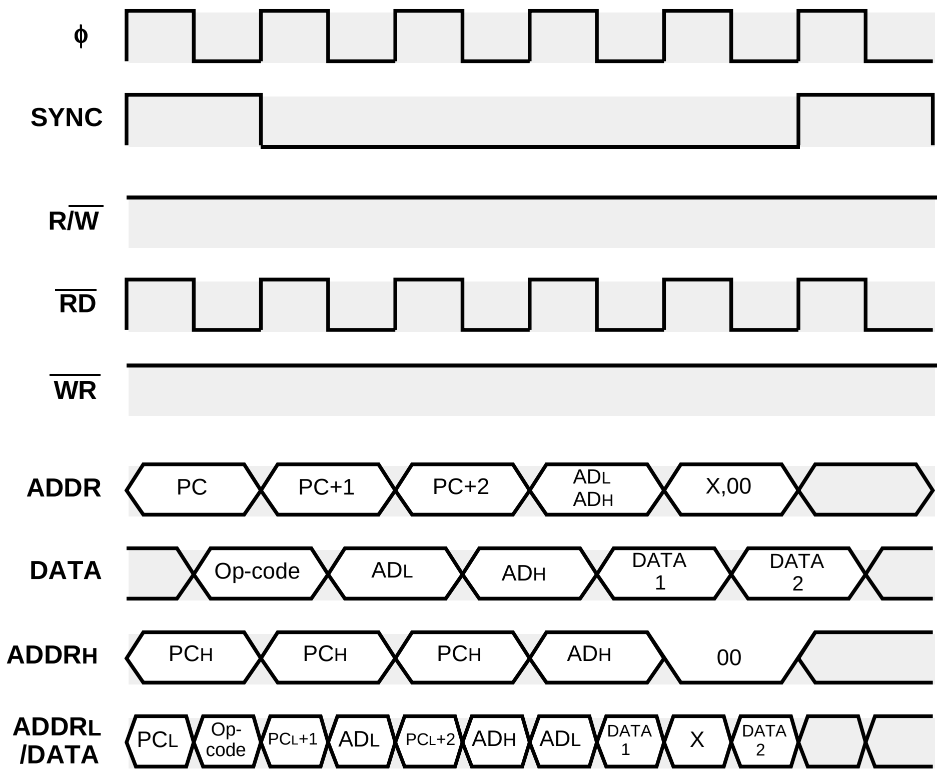
Timing :



ABSOLUTE

Instruction : ∆**CMP**∆**$hhll** **(T=1)**
Byte length : **3**
Cycle number : **5**

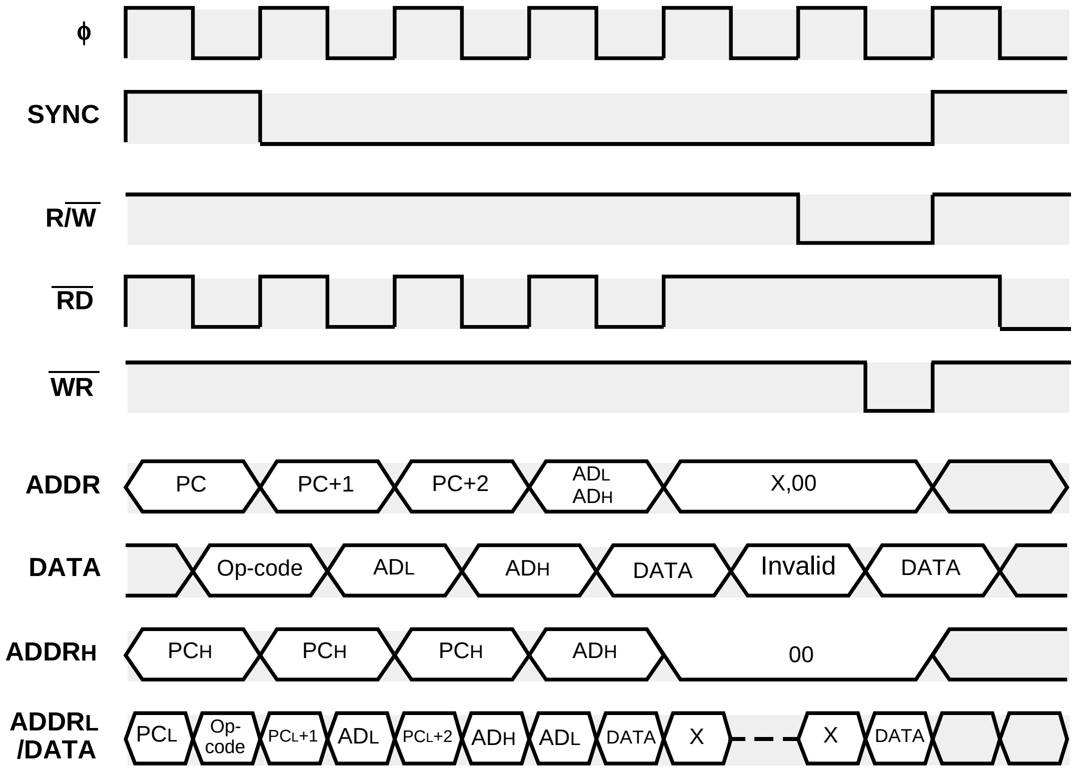
Timing :



ABSOLUTE

Instruction : ΔLDAΔ$hhll (T=1)
Byte length : 3
Cycle number : 6

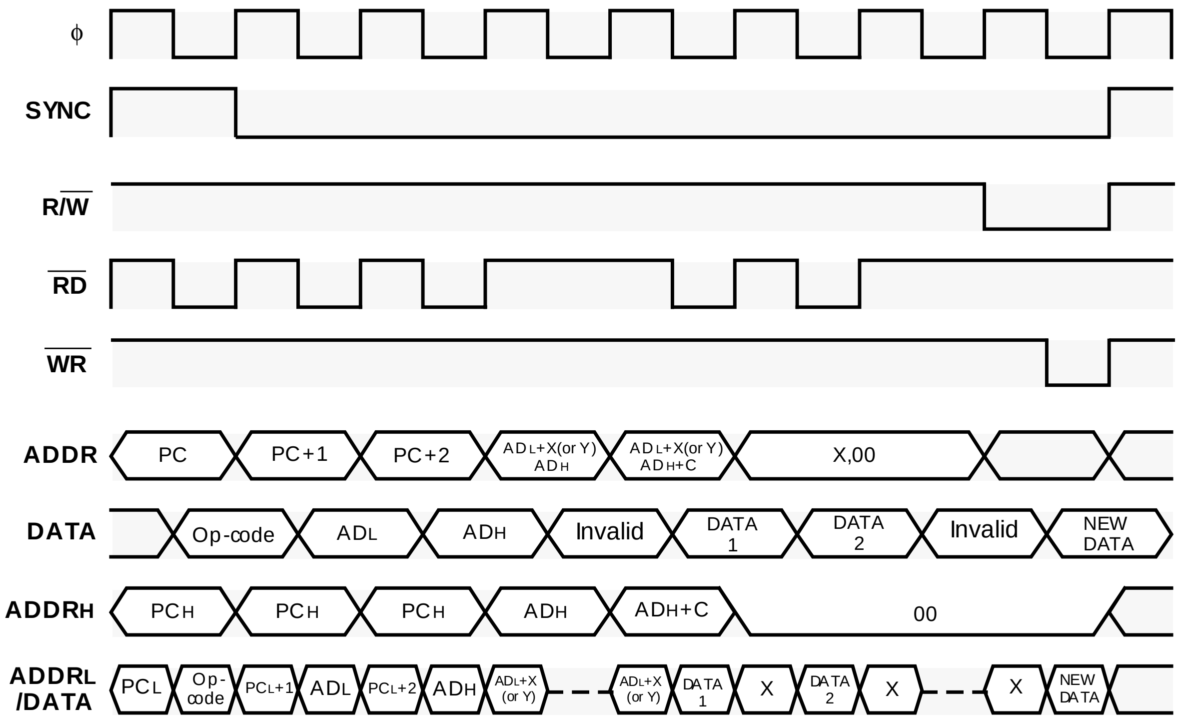
Timing :



ABSOLUTE X, ABSOLUTE Y

Instructions : **∆ADC∆$hhll,X or Y (T=1)**
∆AND∆$hhll,X or Y (T=1)
∆EOR∆$hhll,X or Y (T=1)
∆ORA∆$hhll,X or Y (T=1)
∆SBC∆$hhll,X or Y (T=1)
Byte length : **3**
Cycle number : **8**

Timing :

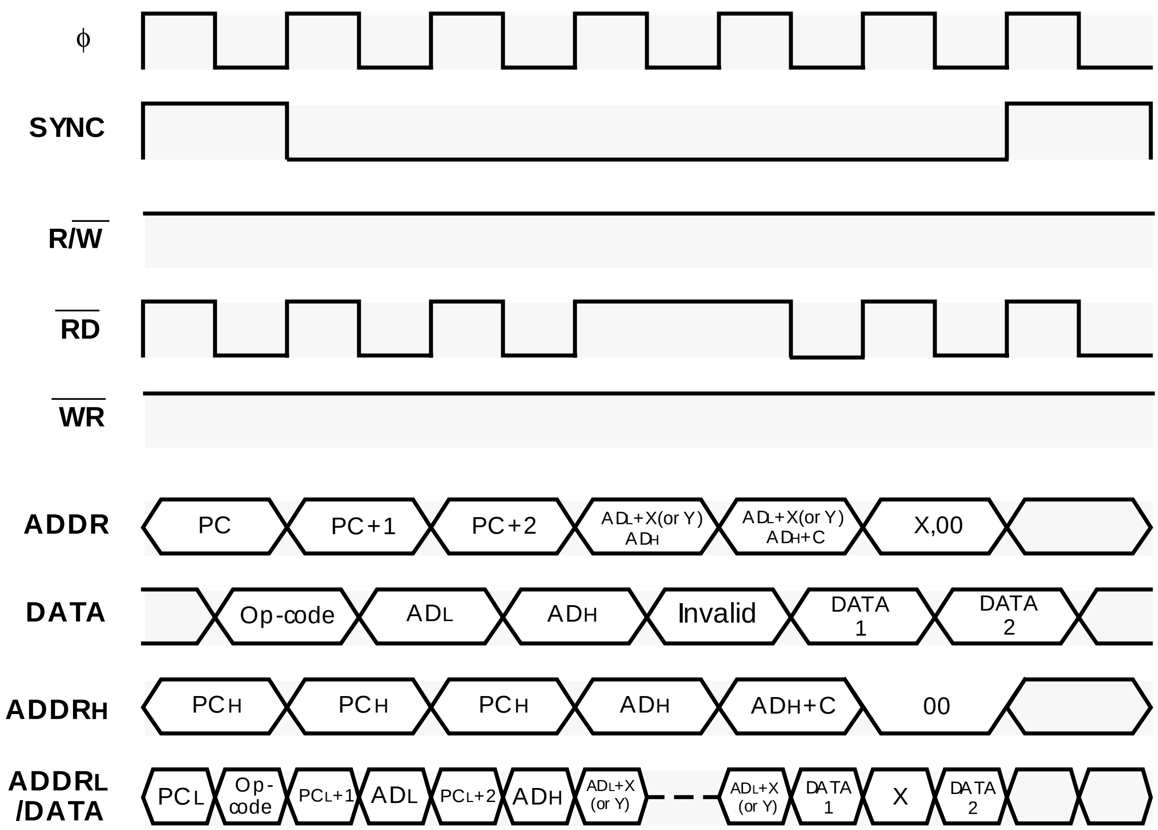


C : Carry of ADL+X or Y

[T=1] **ABSOLUTE X, ABSOLUTE Y**

Instruction : ΔCMPΔ$hhll,X or Y (T=1)
Byte length : **3**
Cycle number : **6**

Timing :

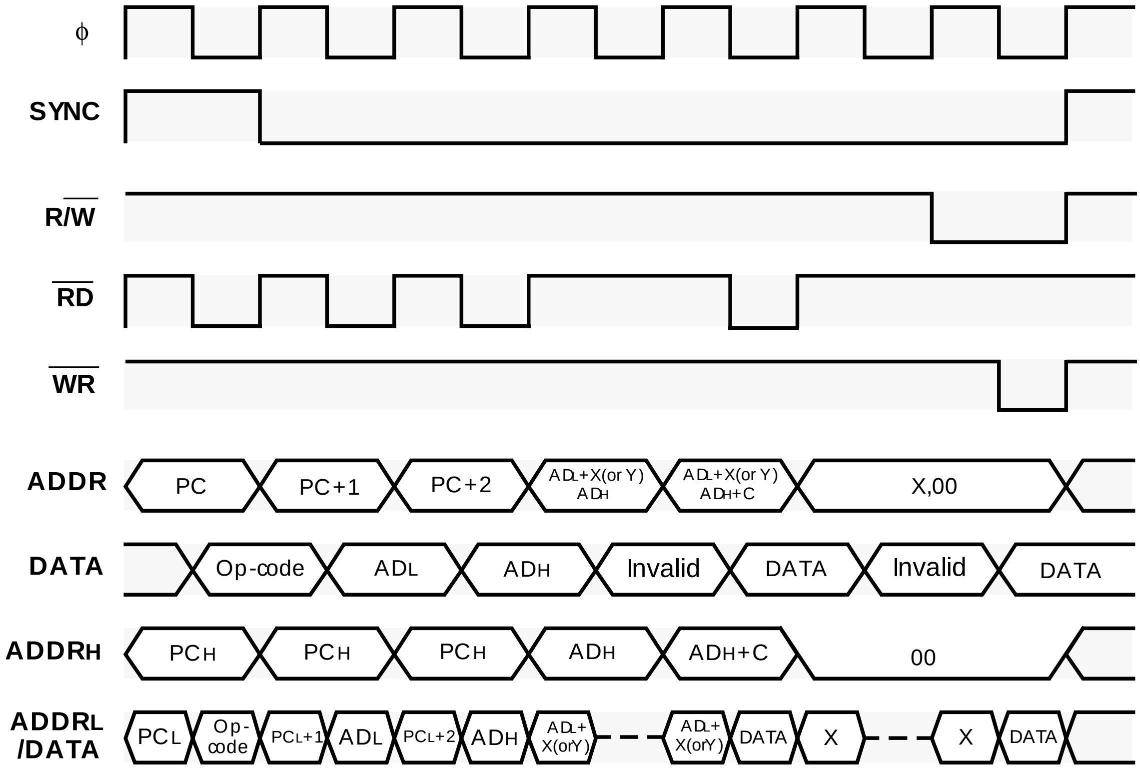


C : Carry of ADL+X or Y

ABSOLUTE X, ABSOLUTE Y

Instruction : **ΔLDAΔ$hhll,X or Y (T=1)**
Byte length : **3**
Cycle number : **7**

Timing :



C : Carry of ADL+X or Y

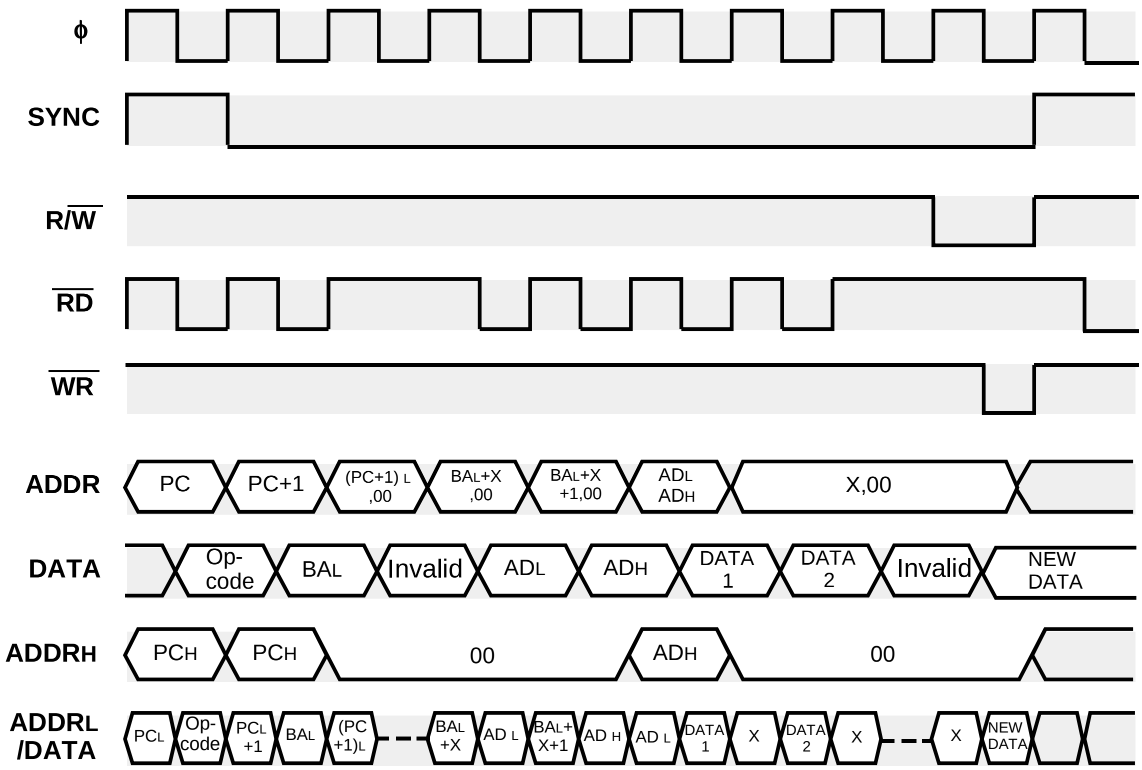
INDIRECT X

Instructions : ∆**ADC**∆**($zz,X)** **(T=1)**
∆**AND**∆**($zz,X)** **(T=1)**
∆**EOR**∆**($zz,X)** **(T=1)**
∆**ORA**∆**($zz,X)** **(T=1)**
∆**SBC**∆**($zz,X)** **(T=1)**

Byte length : **2**

Cycle number : **9**

Timing :

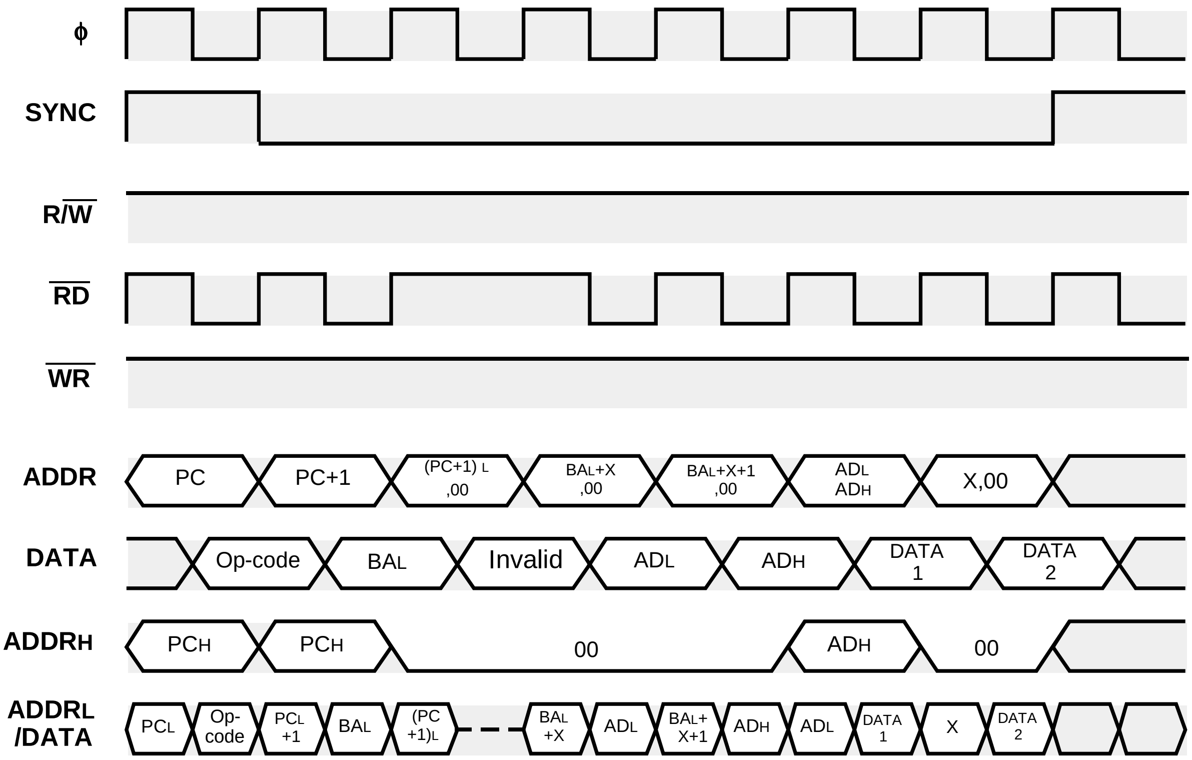


BA : Basic address

INDIRECT X

Instruction : ∆CMP∆($zz,X) (T=1)
Byte length : 2
Cycle number : 7

Timing :

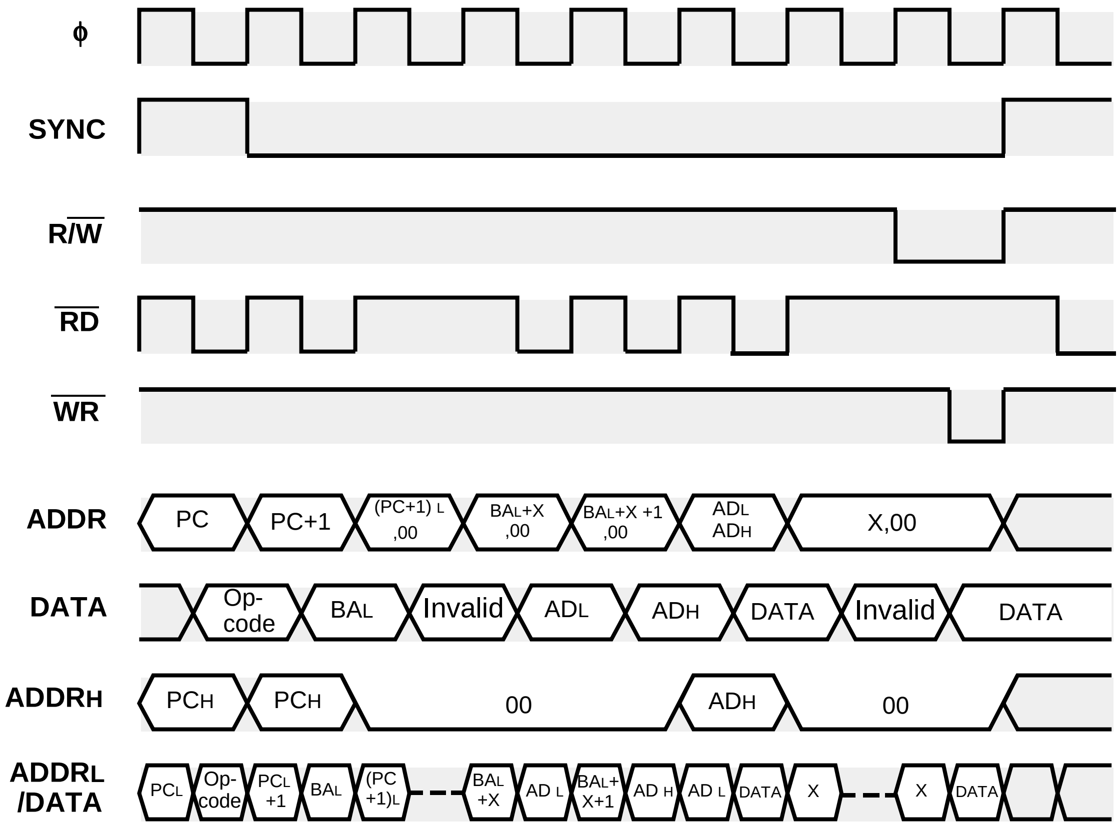


BA : Basic address

INDIRECT X

Instruction : ΔLDAΔ($zz,X) (T=1)
Byte length : 2
Cycle number : 8

Timing :



BA : Basic address

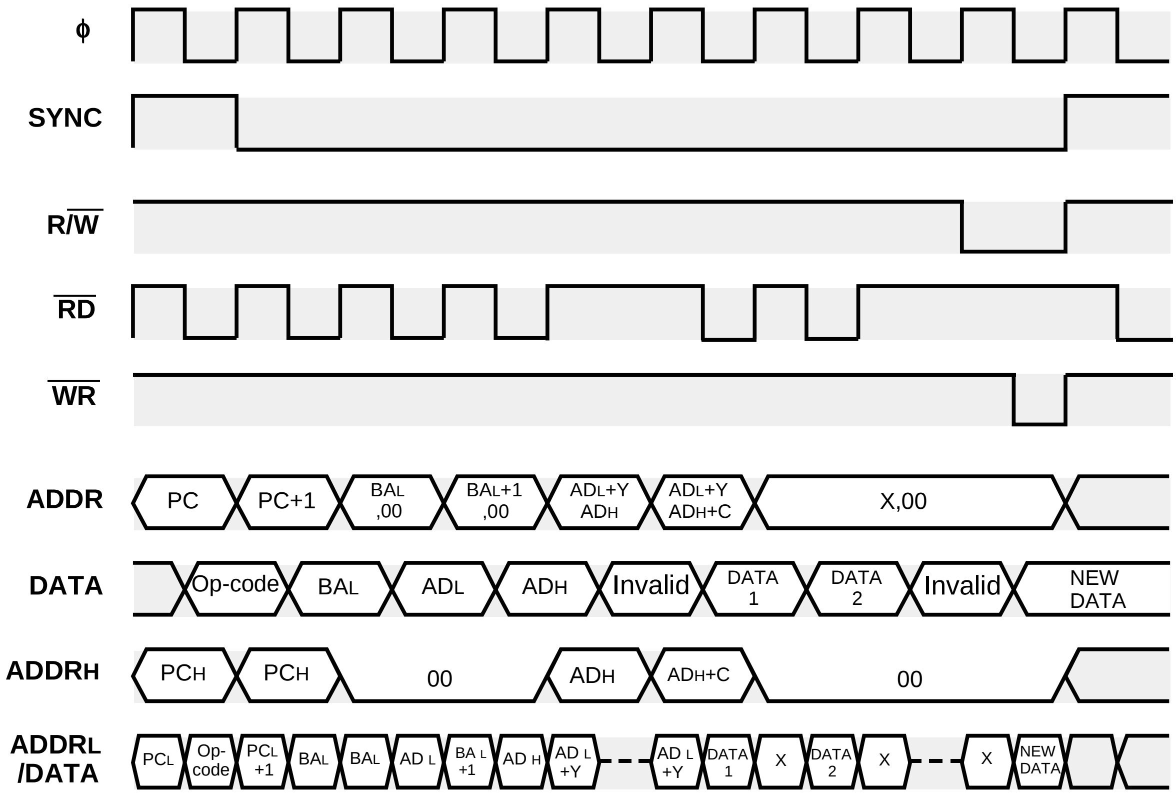
INDIRECT Y

Instructions : **∆ADC∆($zz),Y (T=1)**
∆AND∆($zz),Y (T=1)
∆EOR∆($zz),Y (T=1)
∆ORA∆($zz),Y (T=1)
∆SBC∆($zz),Y (T=1)

Byte length : **2**

Cycle number : **9**

Timing :



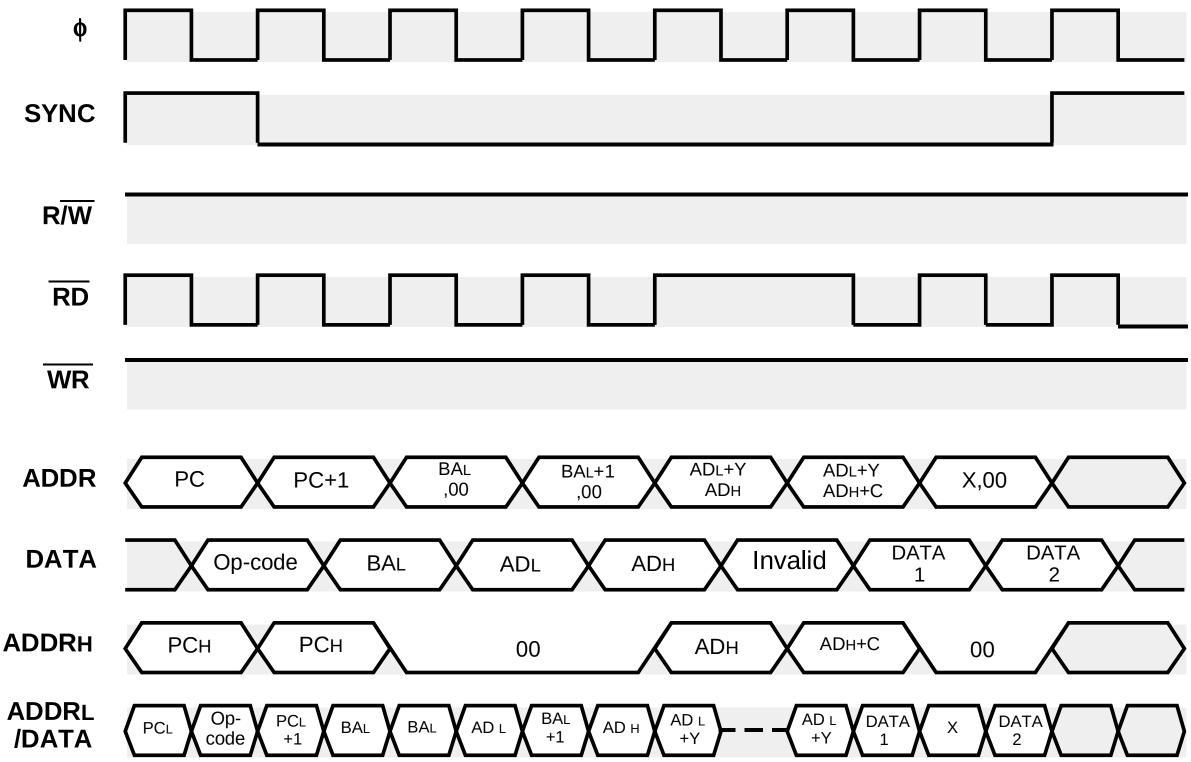
BA : Basic address

C : Carry of ADL+Y

INDIRECT Y

Instruction : ΔCMPΔ($zz),Y (T=1)
Byte length : **2**
Cycle number : **7**

Timing :



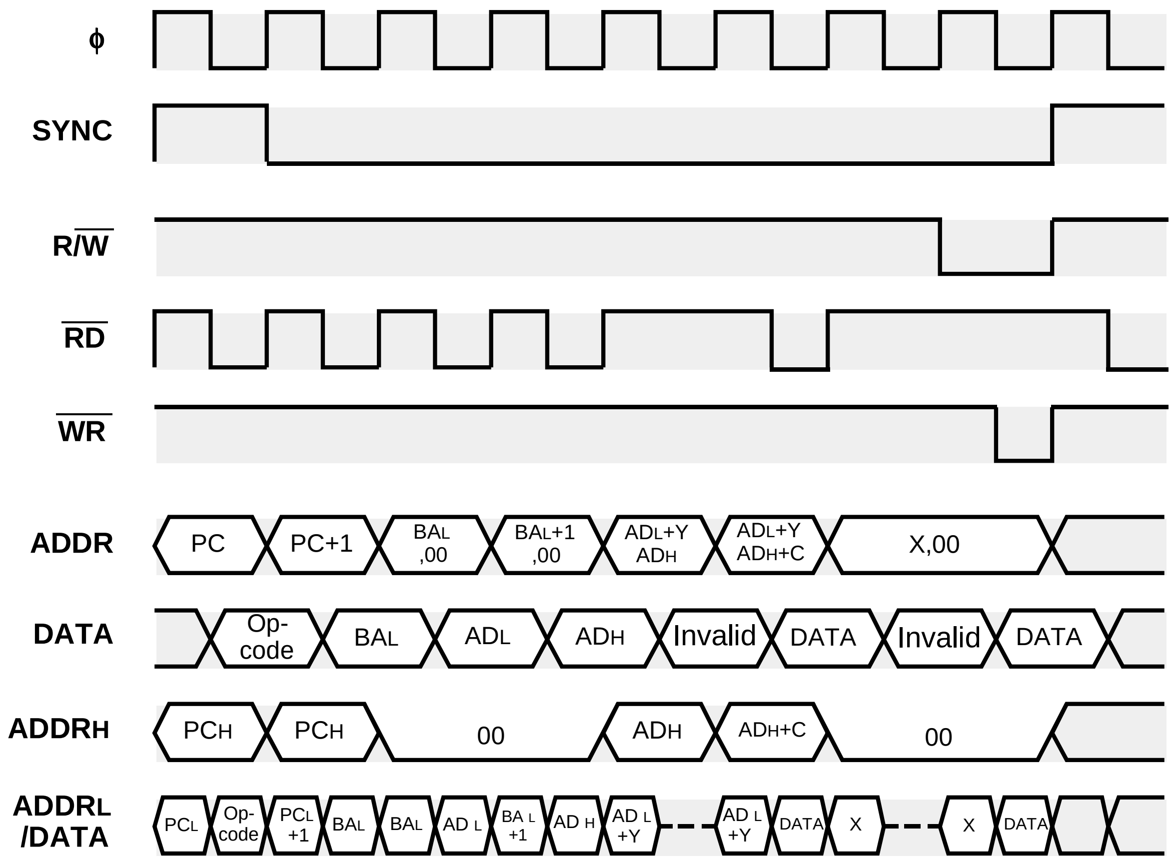
BA : Basic address

C : Carry of ADL+Y

INDIRECT Y

Instruction : ΔLDAΔ($zz),Y (T=1)
Byte length : 2
Cycle number : 8

Timing :



BA : Basic address

C : Carry of ADL+Y

APPENDIX 2. 740 Family Machine Language Instruction Table

Parameter / Classification		SYMBOL	FUNCTION	FLAG N V T B D I Z C	INSTRUCTION CODE $D_7D_6D_5D_4$ $D_3D_2D_1D_0$	HEX	BYTE NUMBER	CYCLE NUMBER	NOTE
Data Transfer	Load	LDA #$nn	(A)←nn	○×××××○×	1010 1001 <B2>	A9	2	2	2
		LDA $zz	(A)←(M) where M=(zz)	○×××××○×	1010 0101 <B2>	A5	2	3	2
		LDA $zz, X	(A)←(M) where M=(zz+(X))	○×××××○×	1011 0101 <B2>	B5	2	4	2
		LDA $hhll	(A)←(M) where M=(hhll)	○×××××○×	1010 1101 <B2> <B3>	AD	3	4	2
		LDA $hhll, X	(A)←(M) where M=(hhll+(X))	○×××××○×	1011 1101 <B2> <B3>	BD	3	5	2
		LDA $hhll, Y	(A)←(M) where M=(hhll+(Y))	○×××××○×	1011 1001 <B2> <B3>	B9	3	5	2
		LDA ($zz, X)	(A)←(M) where M=((zz+(X)+1)(zz+(X)))	○×××××○×	1010 0001 <B2>	A1	2	6	2
		LDA ($zz), Y	(A)←(M) where M=((zz+1)(zz)+(Y))	○×××××○×	1011 0001 <B2>	B1	2	6	2
		LDX #$nn	(X)←nn	○×××××○×	1010 0010 <B2>	A2	2	2	
		LDX $zz	(X)←(M) where M=(zz)	○×××××○×	1010 0110 <B2>	A6	2	3	
		LDX $zz, Y	(X)←(M) where M=(zz+(Y))	○×××××○×	1011 0110 <B2>	B6	2	4	
		LDX $hhll	(X)←(M) where M=(hhll)	○×××××○×	1010 1110 <B2> <B3>	AE	3	4	
		LDX $hhll, Y	(X)←(M) where M=(hhll+(Y))	○×××××○×	1011 1110 <B2> <B3>	BE	3	5	
		LDY #$nn	(Y)←nn	○×××××○×	1010 0000 <B2>	A0	2	2	
		LDY $zz	(Y)←(M) where M=(zz)	○×××××○×	1010 0100 <B2>	A4	2	3	
		LDY $zz, X	(Y)←(M) where M=(zz+(X))	○×××××○×	1011 0100 <B2>	B4	2	4	
		LDY $hhll	(Y)←(M) where M=(hhll)	○×××××○×	1010 1100 <B2> <B3>	AC	3	4	
		LDY $hhll, X	(Y)←(M) where M=(hhll+(X))	○×××××○×	1011 1100 <B2> <B3>	BC	3	5	
		LDM #$nn, $zz	(M)←nn where M=(zz)	××××××××	0011 1100 <B2> <B3>	3C	3	4	
	Store	STA $zz	(M)←(A) where M=(zz)	××××××××	1000 0101 <B2>	85	2	4	
		STA $zz, X	(M)←(A) where M=(zz+(X))	××××××××	1001 0101 <B2>	95	2	5	
		STA $hhll	(M)←(A) where M=(hhll)	××××××××	1000 1101 <B2> <B3>	8D	3	5	
		STA $hhll, X	(M)←(A) where M=(hhll+(X))	××××××××	1001 1101 <B2> <B3>	9D	3	6	
		STA $hhll, Y	(M)←(A) where M=(hhll+(Y))	××××××××	1001 1001 <B2> <B3>	99	3	6	
		STA ($zz, X)	(M)←(A) where M=((zz+(X)+1)(zz+(X)))	××××××××	1000 0001 <B2>	81	2	7	
		STA ($zz), Y	(M)←(A) where M=((zz+1)(zz)+(Y))	××××××××	1001 0001 <B2>	91	2	7	
		STX $zz	(M)←(X) where M=(zz)	××××××××	1000 0110 <B2>	86	2	4	
		STX $zz, Y	(M)←(X) where M=(zz+(Y))	××××××××	1001 0110 <B2>	96	2	5	
		STX $hhll	(M)←(X) where M=(hhll)	××××××××	1000 1110 <B2> <B3>	8E	3	5	
		STY $zz	(M)←(Y) where M=(zz)	××××××××	1000 0100 <B2>	84	2	4	
		STY $zz, X	(M)←(Y) where M=(zz+(X))	××××××××	1001 0100 <B2>	94	2	5	
		STY $hhll	(M)←(Y) where M=(hhll)	××××××××	1000 1100 <B2> <B3>	8C	3	6	
	Transfer	TAX	(X)←(A)	○×××××○×	1010 1010	AA	1	2	
		TXA	(A)←(X)	○×××××○×	1000 1010	8A	1	2	
		TAY	(Y)←(A)	○×××××○×	1010 1000	A8	1	2	
		TYA	(A)←(Y)	○×××××○×	1001 1000	98	1	2	
		TSX	(X)←(S)	○×××××○×	1011 1010	BA	1	2	
		TXS	(S)←(X)	××××××××	1001 1010	9A	1	2	
	Stack Operation	PHA	(M(S))←(A), (S)←(S)−1	××××××××	0100 1000	48	1	3	
		PHP	(M(S))←(PS), (S)←(S)−1	××××××××	0000 1000	08	1	3	
		PLA	(S)←(S)+1, (A)←(M(S))	○×××××○×	0110 1000	68	1	4	
		PLP	(S)←(S)+1, (PS)←(M(S))	Previous status in stack	0010 1000	28	1	4	

Parameter / Classification		SYMBOL	FUNCTION	FLAG N V T B D I Z C	INSTRUCTION CODE D7D6D5D4	D3D2D1D0	HEX	BYTE NUMBER	CYCLE NUMBER	NOTE
Operation	Add and Sabstruct	ADC #$nn	(A)←(A)+nn+(C)	○○××××○○	0 1 1 0	1 0 0 1 <B2>	69	2	2	1
		ADC $zz	(A)←(A)+(M)+(C) where M=(zz)	○○××××○○	0 1 1 0	0 1 0 1 <B2>	65	2	3	1
		ADC $zz, X	(A)←(A)+(M)+(C) where M=(zz+(X))	○○××××○○	0 1 1 1	0 1 0 1 <B2>	75	2	4	1
		ADC $hhll	(A)←(A)+(M)+(C) where M=(hhll)	○○××××○○	0 1 1 0	1 1 0 1 <B2> <B3>	6D	3	4	1
		ADC $hhll, X	(A)←(A)+(M)+(C) where M=(hhll+(X))	○○××××○○	0 1 1 1	1 1 0 1 <B2> <B3>	7D	3	5	1
		ADC $hhll, Y	(A)←(A)+(M)+(C) where M=(hhll+(Y))	○○××××○○	0 1 1 1	1 0 0 1 <B2> <B3>	79	3	5	1
		ADC ($zz, X)	(A)←(A)+(M)+(C) where M=((zz+(X)+1)(zz+(X)))	○○××××○○	0 1 1 0	0 0 0 1 <B2>	61	2	6	1
		ADC ($zz), Y	(A)←(A)+(M)+(C) where M=((zz+1)(zz)+(Y))	○○××××○○	0 1 1 1	0 0 0 1 <B2>	71	2	6	1
		SBC #$nn	(A)←(A)–nn–($\overline{C}$)	○○××××○○	1 1 1 0	1 0 0 1 <B2>	E9	2	2	1
		SBC $zz	(A)←(A)–(M)–($\overline{C}$) where M=(zz)	○○××××○○	1 1 1 0	0 1 0 1 <B2>	E5	2	3	1
		SBC $zz, X	(A)←(A)–(M)–($\overline{C}$) where M=(zz+(X))	○○××××○○	1 1 1 1	0 1 0 1 <B2>	F5	2	4	1
		SBC $hhll	(A)←(A)–(M)–($\overline{C}$) where M=(hhll)	○○××××○○	1 1 1 0	1 1 0 1 <B2> <B3>	ED	3	4	1
		SBC $hhll, X	(A)←(A)–(M)–($\overline{C}$) where M=(hhll+(X))	○○××××○○	1 1 1 1	1 1 0 1 <B2> <B3>	FD	3	5	1
		SBC $hhll, Y	(A)←(A)–(M)–($\overline{C}$) where M=(hhll+(Y))	○○××××○○	1 1 1 1	1 0 0 1 <B2> <B3>	F9	3	5	1
		SBC ($zz, X)	(A)←(A)–(M)–($\overline{C}$) where M=((zz+(X)+1)(zz+(X)))	○○××××○○	1 1 1 0	0 0 0 1 <B2>	E1	2	6	1
		SBC ($zz), Y	(A)←(A)–(M)–($\overline{C}$) where M=((zz+1)(zz)+(Y))	○○××××○○	1 1 1 1	0 0 0 1 <B2>	F1	2	6	1
		INC A	(A)←(A)+1	○×××××○×	0 0 1 1	1 0 1 0	3A	1	2	
		INC $zz	(M)←(M)+1 where M=(zz)	○×××××○×	1 1 1 0	0 1 1 0 <B2>	E6	2	5	
		INC $zz, X	(M)←(M)+1 where M=(zz+(X))	○×××××○×	1 1 1 1	0 1 1 0 <B2>	F6	2	6	
		INC $hhll	(M)←(M)+1 where M=(hhll)	○×××××○×	1 1 1 0	1 1 1 0 <B2> <B3>	EE	3	6	
		INC $hhll, X	(M)←(M)+1 where M=(hhll+(X))	○×××××○×	1 1 1 1	1 1 1 0 <B2> <B3>	FE	3	7	
		DEC A	(A)←(A)–1	○×××××○×	0 0 0 1	1 0 1 0	1A	1	2	
		DEC $zz	(M)←(M)–1 where M=(zz)	○×××××○×	1 1 0 0	0 1 1 0 <B2>	C6	2	5	
		DEC $zz, X	(M)←(M)–1 where M=(zz+(X))	○×××××○×	1 1 0 1	0 1 1 0 <B2>	D6	2	6	
		DEC $hhll	(M)←(M)–1 where M=(hhll)	○×××××○×	1 1 0 0	1 1 1 0 <B2> <B3>	CE	3	6	
		DEC $hhll, X	(M)←(M)–1 where M=(hhll+(X))	○×××××○×	1 1 0 1	1 1 1 0 <B2> <B3>	DE	3	7	
		INX	(X)←(X)+1	○×××××○×	1 1 1 0	1 0 0 0	E8	1	2	
		DEX	(X)←(X)–1	○×××××○×	1 1 0 0	1 0 1 0	CA	1	2	
		INY	(Y)←(Y)+1	○×××××○×	1 1 0 0	1 0 0 0	C8	1	2	
		DEY	(Y)←(Y)–1	○×××××○×	1 0 0 0	1 0 0 0	88	1	2	
	Multiply / Divide	MUL $zz, X	M(S), (A)←(A)×M(zz+(X)) (S)←(S)–1	××××××××	0 1 1 0	0 0 1 0	62	2	15	
		DIV $zz, X	(A)←(M(zz+(X)+1), M(zz+(X))÷(A) M(S)←One's complement of remainder (S)←(S)–1	××××××××	1 1 1 0	0 0 1 0	E2	2	16	

Parameter / Classification		SYMBOL	FUNCTION	FLAG N V T B D I Z C	INSTRUCTION CODE D7D6D5D4	D3D2D1D0	HEX	BYTE NUMBER	CYCLE NUMBER	NOTE
Operation	Logic Operation	AND # $ nn	(A)←(A)∧nn	○×××××○×	0 0 1 0	1 0 0 1 <B2>	29	2	2	1
		AND $ zz	(A)←(A)∧(M) where M=(zz)	○×××××○×	0 0 1 0	0 1 0 1 <B2>	25	2	3	1
		AND $ zz, X	(A)←(A)∧(M) where M=(zz+(X))	○×××××○×	0 0 1 1	0 1 0 1 <B2>	35	2	4	1
		AND $ hhll	(A)←(A)∧(M) where M=(hhll)	○×××××○×	0 0 1 0	1 1 0 1 <B2> <B3>	2D	3	4	1
		AND $ hhll, X	(A)←(A)∧(M) where M=(hhll+(X))	○×××××○×	0 0 1 1	1 1 0 1 <B2> <B3>	3D	3	5	1
		AND $ hhll, Y	(A)←(A)∧(M) where M=(hhll+(Y))	○×××××○×	0 0 1 1	1 0 0 1 <B2> <B3>	39	3	5	1
		AND ($ zz, X)	(A)←(A)∧(M) where M=((zz+(X)+1)(zz+(X)))	○×××××○×	0 0 1 0	0 0 0 1 <B2>	21	2	6	1
		AND ($ zz), Y	(A)←(A)∧(M) where M=((zz+1)(zz)+(Y))	○×××××○×	0 0 1 1	0 0 0 1 <B2>	31	2	6	1
		ORA # $ nn	(A)←(A)∨nn	○×××××○×	0 0 0 0	1 0 0 1 <B2>	09	2	2	1
		ORA $ zz	(A)←(A)∨(M) where M=(zz)	○×××××○×	0 0 0 0	0 1 0 1 <B2>	05	2	3	1
		ORA $ zz, X	(A)←(A)∨(M) where M=(zz+(X))	○×××××○×	0 0 0 1	0 1 0 1 <B2>	15	2	4	1
		ORA $ hhll	(A)←(A)∨(M) where M=(hhll)	○×××××○×	0 0 0 0	1 1 0 1 <B2> <B3>	0D	3	4	1
		ORA $ hhll, X	(A)←(A)∨(M) where M=(hhll+(X))	○×××××○×	0 0 0 1	1 1 0 1 <B2> <B3>	1D	3	5	1
		ORA $ hhll, Y	(A)←(A)∨(M) where M=(hhll+(Y))	○×××××○×	0 0 0 1	1 0 0 1 <B2> <B3>	19	3	5	1
		ORA ($ zz, X)	(A)←(A)∨(M) where M=((zz+(X)+1)(zz+(X)))	○×××××○×	0 0 0 0	0 0 0 1 <B2>	01	2	6	1
		ORA ($ zz), Y	(A)←(A)∨(M) where M=((zz+1)(zz)+(Y))	○×××××○×	0 0 0 1	0 0 0 1 <B2>	11	2	6	1
		EOR # $ nn	(A)←(A)∀nn	○×××××○×	0 1 0 0	1 0 0 1 <B2>	49	2	2	1
		EOR $ zz	(A)←(A)∀(M) where M=(zz)	○×××××○×	0 1 0 0	0 1 0 1 <B2>	45	2	3	1
		EOR $ zz, X	(A)←(A)∀(M) where M=(zz+(X))	○×××××○×	0 1 0 1	0 1 0 1 <B2>	55	2	4	1
		EOR $ hhll	(A)←(A)∀(M) where M=(hhll)	○×××××○×	0 1 0 0	1 1 0 1 <B2> <B3>	4D	3	4	1
		EOR $ hhll, X	(A)←(A)∀(M) where M=(hhll+(X))	○×××××○×	0 1 0 1	1 1 0 1 <B2> <B3>	5D	3	5	1
		EOR $ hhll, Y	(A)←(A)∀(M) where M=(hhll+(Y))	○×××××○×	0 1 0 1	1 0 0 1 <B2> <B3>	59	3	5	1
		EOR ($ zz, X)	(A)←(A)∀(M) where M=((zz+(X)+1)(zz+(X)))	○×××××○×	0 1 0 0	0 0 0 1 <B2>	41	2	6	1
		EOR ($ zz), Y	(A)←(A)∀(M) where M=((zz+1)(zz)+(Y))	○×××××○×	0 1 0 1	0 0 0 1 <B2>	51	2	6	1
		COM $ zz	(M)←($\overline{M}$) where M=(zz)	○×××××○×	0 1 0 0	0 1 0 0 <B2>	44	2	5	
		BIT $ zz	(A) ∧ (M) where M=(zz)	M7M6××××○×	0 0 1 0	0 1 0 0 <B2>	24	2	3	
		BIT $ hhll	(A) ∧ (M) where M=(hhll)	M7M6××××○×	0 0 1 0	1 1 0 0 <B2> <B3>	2C	3	4	
		TST $ zz	(M)=0? where M=(zz)	○×××××○×	0 1 1 0	0 1 0 0 <B2>	64	2	3	
	Comparison	CMP # $ nn	(A)–nn (Comparison in size)	○×××××○○	1 1 0 0	1 0 0 1 <B2>	C9	2	2	3
		CMP $ zz	(A)–(M) where M=(zz)	○×××××○○	1 1 0 0	0 1 0 1 <B2>	C5	2	3	3
		CMP $ zz, X	(A)–(M) where M=(zz+(X))	○×××××○○	1 1 0 1	0 1 0 1 <B2>	D5	2	4	3
		CMP $ hhll	(A)–(M) where M=(hhll)	○×××××○○	1 1 0 0	1 1 0 1 <B2> <B3>	CD	3	4	3
		CMP $ hhll, X	(A)–(M) where M=(hhll+(X))	○×××××○○	1 1 0 1	1 1 0 1 <B2> <B3>	DD	3	5	3
		CMP $ hhll, Y	(A)–(M) where M=(hhll+(Y))	○×××××○○	1 1 0 1	1 0 0 1 <B2> <B3>	D9	3	5	3
		CMP ($ zz, X)	(A)–(M) where M=((zz+(X)+1)(zz+(X)))	○×××××○○	1 1 0 0	0 0 0 1 <B2>	C1	2	6	3
		CMP ($ zz), Y	(A)–(M) where M=((zz+1)(zz)+(Y))	○×××××○○	1 1 0 1	0 0 0 1 <B2>	D1	2	6	3
		CPX # $ nn	(X)–nn (Comparison in size)	○×××××○○	1 1 1 0	0 0 0 0 <B2>	E0	2	2	
		CPX $ zz	(X)–(M) where M=(zz)	○×××××○○	1 1 1 0	0 1 0 0 <B2>	E4	2	3	
		CPX $ hhll	(X)–(M) where M=(hhll)	○×××××○○	1 1 1 0	1 1 0 0 <B2> <B3>	EC	3	4	
		CPY # $ nn	(Y)–nn (Comparison in size)	○×××××○○	1 1 0 0	0 0 0 0 <B2>	C0	2	2	
		CPY $ zz	(Y)–(M) where M=(zz)	○×××××○○	1 1 0 0	0 1 0 0 <B2>	C4	2	3	
		CPY $ hhll	(Y)–(M) where M=(hhll)	○×××××○○	1 1 0 0	1 1 0 0 <B2> <B3>	CC	3	4	

Parameter / Classification	SYMBOL	FUNCTION	FLAG (N V T B D I Z C)	INSTRUCTION CODE D7D6D5D4	D3D2D1D0	HEX	BYTE NUMBER	CYCLE NUMBER	NOTE
Operation / Rotate and Shift	ASL A	Left Shift C←A7A6 A1A0←0	○×××××○○	0 0 0 0	1 0 1 0	0A	1	2	
	ASL $ zz	where M=(zz)	○×××××○○	0 0 0 0	0 1 1 0 <B2>	06	2	5	
	ASL $ zz, X	where M=(zz+(X))	○×××××○○	0 0 0 1	0 1 1 0 <B2>	16	2	6	
	ASL $ hhll	Left Shift C←M7M6 M1M0←0 where M=(hhll)	○×××××○○	0 0 0 0	1 1 1 0 <B2> <B3>	0E	3	6	
	ASL $ hhll, X	where M=(hhll+(X))	○×××××○○	0 0 0 1	1 1 1 0 <B2> <B3>	1E	3	7	
	LSR A	Right Shift 0→A7A6 A1A0→C	0×××××○○	0 1 0 0	1 0 1 0	4A	1	2	
	LSR $ zz	where M=(zz)	0×××××○○	0 1 0 0	0 1 1 0 <B2>	46	2	5	
	LSR $ zz, X	where M=(zz+(X))	0×××××○○	0 1 0 1	0 1 1 0 <B2>	56	2	6	
	LSR $ hhll	Right Shift 0→M7M6 M1M0→C where M=(hhll)	0×××××○○	0 1 0 0	1 1 1 0 <B2> <B3>	4E	3	6	
	LSR $ hhll, X	where M=(hhll+(X))	0×××××○○	0 1 0 1	1 1 1 0 <B2> <B2>	5E	3	7	
	ROL A	Left Shift ←A7A6 A1A0←C←	○×××××○○	0 0 1 0	1 0 1 0	2A	1	2	
	ROL $ zz	where M=(zz)	○×××××○○	0 0 1 0	0 1 1 0 <B2>	26	2	5	
	ROL $ zz, X	where M(zz+(X))	○×××××○○	0 0 1 1	0 1 1 0 <B2>	36	2	6	
	ROL $ hhll	Left Shift ←M7M6 M1M0←C← where M(hhll)	○×××××○○	0 0 1 0	1 1 1 0 <B2> <B3>	2E	3	6	
	ROL $ hhll, X	where M(hhll+(X))	○×××××○○	0 0 1 1	1 1 1 0 <B2> <B3>	3E	3	7	
	ROR A	Right Shift →C→A7A6 A1A0→	○×××××○○	0 1 1 0	1 0 1 0	6A	1	2	
	ROR $ zz	where M=(zz)	○×××××○○	0 1 1 0	0 1 1 0 <B2>	66	2	5	
	ROR $ zz, X	where M=(zz+(X))	○×××××○○	0 1 1 1	0 1 1 0 <B2>	76	2	6	
	ROR $ hhll	Right Shift →C→M7M6 M1M0→ where M=(hhll)	○×××××○○	0 1 1 0	1 1 1 0 <B2> <B3>	6E	3	6	
	ROR $ hhll, X	where M=(hhll+(X))	○×××××○○	0 1 1 1	1 1 1 0 <B2> <B2>	7E	3	7	
	RRF $ zz	M7 M4 M3 M0 where M=(zz)	××××××××	1 0 0 0	0 0 1 0 <B2>	82	2	8	
Bit Management	CLB i, A	(Ai) ← 0 where i=0—7	××××××××	i i i 1	1 0 1 1	(1+2i)×10 +B	1	2	
	CLB i, $ zz	(Mi) ← 0 where i=0—7, M=(zz)	××××××××	i i i 1	1 1 1 1 <B2>	(1+2i)×10 +F	2	5	
	SEB i, A	(Ai) ←1 where i=0—7	××××××××	i i i 0	1 0 1 1	2i×10 +B	1	2	
	SEB i, $ zz	(Mi) ← 1 where i=0—7, M=(zz)	××××××××	i i i 0	1 1 1 1 <B2>	2i×10 +F	2	5	
Flag setting	CLC	(C) ← 0	×××××××0	0 0 0 1	1 0 0 0	18	1	2	
	SEC	(C) ← 1	×××××××1	0 0 1 1	1 0 0 0	38	1	2	
	CLD	(D) ← 0	××××0×××	1 1 0 1	1 0 0 0	D8	1	2	
	SED	(D) ← 1	××××1×××	1 1 1 1	1 0 0 0	F8	1	2	
	CLI	(I) ← 0	×××××0××	0 1 0 1	1 0 0 0	58	1	2	
	SEI	(I) ← 1	×××××1××	0 1 1 1	1 0 0 0	78	1	2	
	CLT	(T) ← 0	××0×××××	0 0 0 1	0 0 1 0	12	1	2	
	SET	(T) ← 1	××1×××××	0 0 1 1	0 0 1 0	32	1	2	
	CLV	(V) ← 0	×0××××××	1 0 1 1	1 0 0 0	B8	1	2	

Parameter / Classification		SYMBOL	FUNCTION	FLAG N V T B D I Z C	INSTRUCTION CODE $D_7D_6D_5D_4$	$D_3D_2D_1D_0$	HEX	BYTE NUMBER	CYCLE NUMBER	NOTE
Branch and Return	Jump	BRA $ hhll	(PC) ← (PC)+2+Rel	× × × × × × × ×	1 0 0 0	0 0 0 0 <B2>	80	2	4	4
		JMP $ hhll	(PC) ← hhll	× × × × × × × ×	0 1 0 0	1 1 0 0 <B2> <B3>	4C	3	3	
		JMP ($ hhll)	(PC_L) ← (hhll), (PC_H) ← (hhll+1)	× × × × × × × ×	0 1 1 0	1 1 0 0 <B2> <B3>	6C	3	5	
		JMP ($ zz)	(PC_L) ←(zz), (PC_H) ← (zz+1)	× × × × × × × ×	1 0 1 1	0 0 1 0 <B2>	B2	2	4	
		JSR $ hhll	(M(S))←(PC_H), (S)←(S) –1, (M(S)) ← (PC_L), (S)←(S) –1, and (PC)←hhll	× × × × × × × ×	0 0 1 0	0 0 0 0 <B2> <B3>	20	3	6	
		JSR ($ zz)	(M(S))←(PC_H), (S)←(S) –1, (M(S))←(PC_L), (S)←(S) –1, (PC_L)←(zz), and (PC_H)←(zz+1)	× × × × × × × ×	0 0 0 0	0 0 1 0 <B2>	02	2	7	
		JSR \ $ hhll	(M(S))←(PC_H), (S)←(S) –1, (M(S))←(PC_L), (S)←(S)–1, (PC_L)←ll, and (PC_H)←FF	× × × × × × × ×	0 0 1 0	0 0 1 0 <B2>	22	2	5	
	Branch	BBC i, A, $ hhll	When (Ai)=0 (PC) ←(PC)+2+Rel; When (Ai)=1 (PC) ← (PC)+2; Where i=0—7	× × × × × × × ×	i i i 1	0 0 1 1 <B2>	(1+2i)x10 +3	2	4	4
		BBC i, $ zz, $ hhll	When (Mi)=0 (PC) ← (PC)+3+Rel; When (Mi)=1 (PC) ← (PC)+3; Where i=0—7	× × × × × × × ×	i i i 1	0 1 1 1 <B2> <B3>	(1+2i)x10 +7	3	5	4
		BBS i, A, $ hhll	When (Ai)=1 (PC) ← (PC)+2+Rel; When (Ai)=0 (PC) ← (PC)+2; Where i=0—7	× × × × × × × ×	i i i 0	0 0 1 1 <B2>	2ix10 +3	2	4	4
		BBS i, $ zz, $ hhll	When (Mi)=1 (PC) ← (PC)+3+Rel; When (Mi)=0 (PC) ← (PC)+3; Where i=0—7	× × × × × × × ×	i i i 0	0 1 1 1 <B2> <B3>	2ix10 +7	3	5	4
		BCC $ hhll	When (C)=0 (PC) ← (PC)+2+Rel; When (C)=1 (PC) ← (PC)+2	× × × × × × × ×	1 0 0 1	0 0 0 0 <B2>	90	2	2	4
		BCS $ hhll	When (C)=1 (PC) ← (PC)+2+Rel; When (C)=0 (PC) ← (PC)+2	× × × × × × × ×	1 0 1 1	0 0 0 0 <B2>	B0	2	2	4
		BNE $ hhll	When (Z)=0 (PC) ← (PC)+2+Rel; When (Z)=1 (PC) ← (PC)+2	× × × × × × × ×	1 1 0 1	0 0 0 0 <B2>	D0	2	2	4
		BEQ $ hhll	When (Z)=1 (PC) ←(PC)+2+Rel; When (Z)=0 (PC) ← (PC)+2	× × × × × × × ×	1 1 1 1	0 0 0 0 <B2>	F0	2	2	4
		BPL $ hhll	When (N)=0 (PC) ← (PC)+2+Rel; When (N)=1 (PC) ← (PC)+2	× × × × × × × ×	0 0 0 1	0 0 0 0 <B2>	10	2	2	4
		BMI $ hhll	When (N)=1 (PC) ← (PC)+2+Rel; When (N)=0 (PC) ← (PC)+2	× × × × × × × ×	0 0 1 1	0 0 0 0 <B2>	30	2	2	4
		BVC $ hhll	When (V)=0 (PC) ← (PC)+2+Rel; When (V)=1 (PC) ← (PC)+2	× × × × × × × ×	0 1 0 1	0 0 0 0 <B2>	50	2	2	4
		BVS $ hhll	When (V)=1 (PC) ← (PC)+2+Rel; When (V)=0 (PC) ← (PC)+2	× × × × × × × ×	0 1 1 1	0 0 0 0 <B2>	70	2	2	4
	Return	RTI	(S)←(S)+1, (PS)←(M(S)), (S)←(S)+1, (PC_L)←(M(S)), (S)←(S)+1, and (PC_H)←(M(S))	Previous status in stack	0 1 0 0	0 0 0 0	40	1	6	
		RTS	(S)←(S)+1, (PC_L)←(M(S)), (S)←(S)+1, (PC_H)←(M(S)), and (PC)←(PC)+1	× × × × × × × ×	0 1 1 0	0 0 0 0	60	1	6	
Interrupt		BRK	(B)←1, (PC)←(PC)+2, (M(S))←(PC_H), (S)←(S)–1, (M(S))←(PC_L), (S)←(S)–1, (M(S))←(PS), (S)←(S)–1, (I)←1, (PC)←BADRS	× × × 1 × 1 × ×	0 0 0 0	0 0 0 0	00	1	7	
Other		NOP	(PC) ← (PC)+1	× × × × × × × ×	1 1 1 0	1 0 1 0	EA	1	2	
Special		WIT	Internal clock source is stopped.	× × × × × × × ×	1 1 0 0	0 0 1 0	C2	1	2	
		STP	Oscillation is stopped.	× × × × × × × ×	0 1 0 0	0 0 1 0	42	1	2	5

Symbol	Means	Symbol	Means
A	Accumulator	hh	High-order byte of address (0—255)
Ai	Bit i of accumulator	ll	Low-order byte of address (0—255)
X	Index register X	zz	Zero page address (0—255)
Y	Index register Y	nn	Date at (0—255)
M	Memory	i	Data at (0—7)
Mi	Bit i of memory	iii	Data at (0—7)
PS	Processor status register	<B2>	Second byte of instruction
S	Stack Pointer	<B3>	Third byte of instruction
PC	Program counter	Rel	Relative address
PCL	Low-order byte of program counter	BADRS	Break address
PCH	High-order byte of program counter	←	Direction of data transfer
N	Negative flag	( )	Contents of register of memory
V	Overflow flag	+	Add
T	X modified operation mode flag	—	Subtract
B	Break flag	*	Multiplication
D	Decimal mode flag	÷	Division
I	Interrupt disable flag	∨	Logical OR
Z	Zero flag	∧	Logical AND
C	Carry flag	∀	Logical Exclusive OR
#	Immediate mode	—	Negative
$	Hexadecimal	×	Stable flag after execution
\	Special page mode	○	Variable flag after execution

Notes 1: Listed function is when (T) = 0.
When (T) = 1, (M(X)) is entered instead of (A) and the cycle number is increased by 3.
2: Ditto. The cycle number is increased by 2.
3: Ditto. The cycle number is increased by 1.
4: The cycle number is increased by 2 when a branch is occurred.
5: If the STP instruction is disabled the cycle number will be 2 (same in operation as two NOPs).

APPENDIX 3. 740 Family list of Instruction Codes

D7 – D4 \ D3 – D0	Hexadecimal notation	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0	BRK	ORA IND, X	JSR ZP, IND	BBS 0, A	—	ORA ZP	ASL ZP	BBS 0, ZP	PHP	ORA IMM	ASL A	SEB 0, A	—	ORA ABS	ASL ABS	SEB 0, ZP
0001	1	BPL	ORA IND, Y	CLT	BBC 0, A	—	ORA ZP, X	ASL ZP, X	BBC 0, ZP	CLC	ORA ABS, Y	DEC A	CLB 0, A	—	ORA ABS, X	ASL ABS, X	CLB 0, ZP
0010	2	JSR ABS	AND IND, X	JSR SP	BBS 1, A	BIT ZP	AND ZP	ROL ZP	BBS 1, ZP	PLP	AND IMM	ROL A	SEB 1, A	BIT ABS	AND ABS	ROL ABS	SEB 1, ZP
0011	3	BMI	AND IND, Y	SET	BBC 1, A	—	AND ZP, X	ROL ZP, X	BBC 1, ZP	SEC	AND ABS, Y	INC A	CLB 1, A	LDM ZP	AND ABS, X	ROL ABS, X	CLB 1, ZP
0100	4	RTI	EOR IND, X	STP (Note)	BBS 2, A	COM ZP	EOR ZP	LSR ZP	BBS 2, ZP	PHA	EOR IMM	LSR A	SEB 2, A	JMP ABS	EOR ABS	LSR ABS	SEB 2, ZP
0101	5	BVC	EOR IND, Y	—	BBC 2, A	—	EOR ZP, X	LSR ZP, X	BBC 2, ZP	CLI	EOR ABS, Y	—	CLB 2, A	—	EOR ABS, X	LSR ABS, X	CLB 2, ZP
0110	6	RTS	ADC IND, X	MUL ZP, X (Note)	BBS 3, A	TST ZP	ADC ZP	ROR ZP	BBS 3, ZP	PLA	ADC IMM	ROR A	SEB 3, A	JMP IND	ADC ABS	ROR ABS	SEB 3, ZP
0111	7	BVS	ADC IND, Y	—	BBC 3, A	—	ADC ZP, X	ROR ZP, X	BBC 3, ZP	SEI	ADC ABS, Y	—	CLB 3, A	—	ADC ABS, X	ROR ABS, X	CLB 3, ZP
1000	8	BRA	STA IND, X	RRF ZP	BBS 4, A	STY ZP	STA ZP	STX ZP	BBS 4, ZP	DEY	—	TXA	SEB 4, A	STY ABS	STA ABS	STX ABS	SEB 4, ZP
1001	9	BCC	STA IND, Y	—	BBC 4, A	STY ZP, X	STA ZP, X	STX ZP, Y	BBC 4, ZP	TYA	STA ABS, Y	TXS	CLB 4, A	—	STA ABS, X	—	CLB 4, ZP
1010	A	LDY IMM	LDA IND, X	LDX IMM	BBS 5, A	LDY ZP	LDA ZP	LDX ZP	BBS 5, ZP	TAY	LDA IMM	TAX	SEB 5, A	LDY ABS	LDA ABS	LDX ABS	SEB 5, ZP
1011	B	BCS	LDA IND, Y	JMP ZP, IND	BBC 5, A	LDY ZP, X	LDA ZP, X	LDX ZP, Y	BBC 5, ZP	CLV	LDA ABS, Y	TSX	CLB 5, A	LDY ABS, X	LDA ABS, X	LDX ABS, Y	CLB 5, ZP
1100	C	CPY IMM	CMP IND, X	WIT	BBS 6, A	CPY ZP	CMP ZP	DEC ZP	BBS 6, ZP	INY	CMP IMM	DEX	SEB 6, A	CPY ABS	CMP ABS	DEC ABS	SEB 6, ZP
1101	D	BNE	CMP IND, Y	—	BBC 6, A	—	CMP ZP, X	DEC ZP, X	BBC 6, ZP	CLD	CMP ABS, Y	—	CLB 6, A	—	CMP ABS, X	DEC ABS, X	CLB 6, ZP
1110	E	CPX IMM	SBC IND, X	DIV ZP, X (Note)	BBS 7, A	CPX ZP	SBC ZP	INC ZP	BBS 7, ZP	INX	SBC IMM	NOP	SEB 7, A	CPX ABS	SBC ABS	INC ABS	SEB 7, ZP
1111	F	BEQ	SBC IND, Y	—	BBC 7, A	—	SBC ZP, X	INC ZP, X	BBC 7, ZP	SED	SBC ABS, Y	—	CLB 7, A	—	SBC ABS, X	INC ABS, X	CLB 7, ZP

Note: Some products unuse these instructions.

- 3-byte instruction
- 2-byte instruction
- 1-byte instruction

Refer to the related section because the clock control instruction and multiplication and division instruction depend on products.

MEMORANDUM

740 Family Software Manual

Publication Data : Rev.1.00 Aug 29, 1997
Rev.2.00 Nov 14, 2006
Published by : Sales Strategic Planning Div.
Renesas Technology Corp.

740 Family
Software Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ09B0322-0200