keep safety first in your circuit designs!

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

Notes regarding these materials

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

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

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

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

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

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

- Please contact Mitsubishi Electric Corporation or an authorized Mitsubishi Semiconductor product distributor for further details on these materials or the products contained therein.
<table>
<thead>
<tr>
<th>Rev. No.</th>
<th>Revision Description</th>
<th>Rev. date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>First Edition</td>
<td>970829</td>
</tr>
</tbody>
</table>

(1/1)
Preface

This software manual is for users of the 740 Family. Register structures, addressing modes and instructions are introduced in each section.

The enhanced instruction set with enhanced data and memory operations enable efficient programming.

Please refer to the “USER’S MANUAL” appropriate for the hardware device or the development support tools used.
# Table of contents

**CHAPTER 1. OVERVIEW** ............................................................................................ 1

**CHAPTER 2. CENTRAL PROCESSING UNIT (CPU)** ............................................... 2

2.1 Accumulator (A) ........................................................................................................................ 2
2.2 Index Register X (X), Index Register Y (Y) ........................................................................ 2
2.3 Stack Pointer (S) ....................................................................................................................... 3
2.4 Program Counter (PC) ............................................................................................................. 4
2.5 Processor Status Register (PS) ............................................................................................. 4

**CHAPTER 3. INSTRUCTIONS** .................................................................................... 6

3.1 Addressing Mode ...................................................................................................................... 6
3.2 Instruction Set ......................................................................................................................... 26
     3.2.1 Data transfer instructions ................................................................................................ 26
     3.2.2 Operating instruction ...................................................................................................... 27
     3.2.3 Bit managing instructions ............................................................................................... 28
     3.2.4 Flag setting instructions .................................................................................................. 28
     3.2.5 Jump, Branch and Return instructions ......................................................................... 28
     3.2.6 Interrupt instruction (Break instruction) ........................................................................ 29
     3.2.7 Special instructions .......................................................................................................... 29
     3.2.8 Other instruction .............................................................................................................. 29
     3.3 Description of instructions ................................................................................................... 30

**CHAPTER 4. NOTES ON USE** ............................................................................... 102

4.1 Notes on interrupts .............................................................................................................. 102
     4.1.1 Setting for interrupt request bit and interrupt enable bit ......................................... 102
     4.1.2 Switching of detection edge ........................................................................................ 102
     4.1.3 Distinction of interrupt request bit .............................................................................. 103
4.2 Notes on programming ........................................................................................................104
     4.2.1 Processor Status Register ........................................................................................... 104
     4.2.2 BRK instruction .............................................................................................................. 105
     4.2.3 Decimal calculations ...................................................................................................... 105
     4.2.4 JMP instruction .............................................................................................................. 106

**APPENDIX 1. Instruction Cycles in each Addressing Mode** .................................... 107

**APPENDIX 2. 740 Family Machine Language Instruction Table** .......................... 173

**APPENDIX 3. 740 Family list of Instruction Codes** ............................................. 179
<Addressing Mode>

Immediate ....................... 7
Accumulator ..................... 8
Zero Page ......................... 9
Zero Page X ...................... 10
Zero Page Y ...................... 11
Absolute ........................ 12
Absolute X ........................ 13
Absolute Y ........................ 14
Implied .......................... 15
Relative .......................... 16
Indirect X ........................ 17
Indirect Y ........................ 18
Indirect Absolute ............... 19
Zero Page Indirect ............. 20

<Instructions>

ADC ......................... 31  CLI ..................... 50  LDX .................. 69  SEI ................... 88
AND ........................ 32  CLT ........................ 51  LDY .................. 70  SET ................... 89
ASL .......................... 33  CLV ........................ 52  LSR .................. 71  STA ................... 90
BBC .......................... 34  CMP ........................ 53  MUL .................. 72  STP ................... 91
BBS .......................... 35  COM ........................ 54  NOP .................. 73  STX ................... 92
BCC .......................... 36  DEC ........................ 55  ORA .................. 74  STY ................... 93
BCS .......................... 37  CPY ........................ 56  PHA .................. 75  TAX ................... 94
BEQ .......................... 38  DEY ........................ 57  PHP .................. 76  TAY ................... 95
BIT .......................... 39  DEX ........................ 58  PLA .................. 77  TST ................... 96
BMI .......................... 40  DEY ........................ 59  PLP .................. 78  TXS ................... 97
BNE .......................... 41  DEX ........................ 60  ROL .................. 79  TXA ................... 98
BPL .......................... 42  EOR ........................ 61  ROR .................. 80  TXS ................... 99
BRA .......................... 43  INC ........................ 62  RRF .................. 81  TYA ................... 100
BRK .......................... 44  INX ........................ 63  RTI .................. 82  WIT .................... 101
BVC .......................... 45  INY ........................ 64  RTS .................. 83
BVS .......................... 46  JMP ........................ 65  SBC .................. 84
CLB .......................... 47  JSR ........................ 66  SEB .................. 85
CLC .......................... 48  LDA ........................ 67  SEC .................. 86
CLD .......................... 49  LDM ........................ 68  SED .................. 87
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 (PCH) 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 (PCL) 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 (PCL).
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 (PCH).

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 be 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 contents 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.

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. 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
    \[ A \leftarrow A \times M2 \]
    * : 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
    \[ M1 \leftarrow M1 \times M2 \]
    * : 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 (7F16) or
−128 (8016) 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

<table>
<thead>
<tr>
<th>Flag</th>
<th>Set instruction</th>
<th>Clear instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>SEC</td>
<td>CLC</td>
</tr>
<tr>
<td>Z</td>
<td>SEI</td>
<td>CLI</td>
</tr>
<tr>
<td>I</td>
<td>SED</td>
<td>CLD</td>
</tr>
<tr>
<td>D</td>
<td>SET</td>
<td>CLT</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 affect 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.
**INSTRUCTIONS**

**Immediate** Addressing mode

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 `ADC#$A5` Machine code $69_{16} A5_{16}$

This symbol(#) indicates the Immediate addressing mode.

```
Memory

Op-code (69_{16})

Operand (A5_{16})

(A) ← (A) + (C) + A5_{16}
```
INSTRUCTIONS

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
\[ \text{ΔROLΔA} \]
Machine code
\[ 2A_{16} \]

Diagram:
- Carry flag
- Accumulator

C

bit 7

bit 0
INSTRUCTIONS

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 00₁₆ 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

\[ \text{ADC} \quad \text{$\Delta$40} \]

Machine code

\[ 6_5 \quad 40_16 \]

Memory diagram:

- Zero page designation
- 00₁₆
- 40₁₆
- FF₁₆
- Operand (40₁₆)
- Op-code (65₁₆)
- Data (XX₁₆)

\[(A) \leftarrow (A) + (C) + XX₁₆ \]
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 $\Delta$ADC$\Delta$S$E$,X

Machine code $75_{16}$ 5$E_{16}$

The diagram illustrates the operation:

\[(A) \leftarrow (A) + (C) + XX_{16}\]

Memory diagram:
- Zero page
- Data $XX_{16}$
- Op-code $75_{16}$
- Operand $5E_{16}$

- Zero page X designation
- Contents of Index Register X
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 0016 as the high-order byte.

Instructions: **LDX, STX**

Example: Mnemonic

\[ \text{LDX} \Delta \text{\$62, Y} \]

Machine code

\[ \text{B6}_{16} \ 62_{16} \]
INSTRUCTIONS

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

\[ \Delta \text{ADC} \Delta \$AD12 \]

Machine code

6D_{16} \ 12_{16} \ AD_{16}

![Diagram of Absolute addressing mode]

\( (A) \leftarrow (A) + (C) + XX_{16} \)
INSTRUCTIONS

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 △ADC△$AD12, X

Machine code 7D₁₆ 12₁₆ AD₁₆

Memory

Op-code (7D₁₆)
Operand I (12₁₆)
Operand II (AD₁₆)

Contetns of Index Register X

(A) ← (A) + (C) + XX₁₆

Data(XX₁₆)

AE00₁₆

Absolute X designation

+ EE₁₆ = AE00₁₆
INSTRUCTIONS

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.


Example: Mnemonics

\[ \text{\textcopyright ADC \textcopyright SAD12, Y} \]

Machine code

\[ 79_{16} \ 12_{16} \ 16_{16} \]

Diagram:

Memory

- Opcode (79\textsubscript{16})
- Operand I (12\textsubscript{16})
- Operand II (AD\textsubscript{16})

Contents of Index Register Y

\[ + \ EE_{16} = AE00_{16} \]

Absolute Y designation

\[ (A) \leftarrow (A) + (C) + XX_{16} \]

Data(XX\textsubscript{16})
**INSTRUCTIONS**

**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

△CLC

**Machine code**

1816

---

**Processor status register**

<table>
<thead>
<tr>
<th>bit 7</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>△</td>
</tr>
</tbody>
</table>

Carry flag

Carry flag is cleared to “0.”

<table>
<thead>
<tr>
<th>bit 7</th>
<th>bit 6</th>
<th>bit 5</th>
<th>bit 4</th>
<th>bit 3</th>
<th>bit 2</th>
<th>bit 1</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
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**

\[ \Delta \text{BCC} \Delta^* -12 \]

**Machine code**

\[ 90_{16} \quad F2_{16} \]

When the carry flag is cleared, jumps to address \(^* -12\).

When the carry flag is set, goes to address \(^* +2\).
**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 0016 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: \( \Delta \text{ADC}(\$1E,X) \)

Machine code: \( 61_{16} 1E_{16} \)

- **Assuming that “0016” for Data I, and “1416” for Data II are stored in advance.**
## INSTRUCTIONS

### 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 00₁₆ 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 $\triangle ADC \triangle (\$1E)_Y$

**Machine code**: 71₁₆ 1E₁₆

Assuming that “01₁₆” for Data I, and “12₁₆” for Data II are stored in advance.
Indirect Absolute  Addressing mode

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
\( \Delta \text{JMP}\Delta(\$1400) \)  

Machine code: \( 6C_{16} \ 00_{16} \ 14_{16} \)

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

Note: The page’s last address (address \( XX_{16} \)) can not be specified for the indirect designation address; in other words, JMP (\( \$XX_{16} \)) can not be executed.
INSTRUCTIONS

Zero Page Indirect

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 00₁₆ 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
ΔJMPΔ($45)  

Machine code
B₂₁₆ 4₅₁₆

Assuming that “FF₁₆” for Data I, and “1E₁₆” for Data II are stored in advance.
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: 

```
JSR $FFC0
```

Machine code: 22$_{16}$ C0$_{16}$

This symbol indicates the Special page mode.
**INSTRUCTIONS**

**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 00₁₆ 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**

ΔCLBΔ5,$44

**Machine code**

BF₁₆ 44₁₆
INSTRUCTIONS

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

ΔCLBΔ5,A

Machine code

BB₁₆

Accumulator

bit 5

Memory

Bit designation

Op-code(BB₁₆)

1 0 1 1 1 0 1 1

Accumulator

bit 5

0
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

\[ \Delta {\text{BBC}}5, A, * \rightarrow 12 \]

**Machine code**

\[ B3_{16} \quad F2_{16} \]

**When the bit 5 of the Accumulator is cleared**

- **Accumulator:**
  - bit 5: 0

- **Memory:**
  - Address to be executed next
  - Bit designation: \( * -12 \)
  - Operand (F2\text{16})

**When the bit 5 of the Accumulator is set**

- **Accumulator:**
  - bit 5: 1

- **Memory:**
  - Address to be executed next
  - Bit designation: \( * +2 \)
INSTRUCTIONS

Zero Page Bit Relative

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 0016 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

\[ \Delta \text{BBC} 5, \$04, \#-12 \]

Machine language

B7 16 04 16 F1 16

When the bit 5 at address 04 16 is cleared, jumps to address *-12.

When the bit 5 at address 04 16 is set, goes to address *+3.
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 any 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.

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

### 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:

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Addition &amp; Subtraction</strong></td>
<td></td>
</tr>
<tr>
<td>ADC</td>
<td>Add memory contents and C flag to Accumulator or memory where is indicated by Index Register X</td>
</tr>
<tr>
<td>SBC</td>
<td>Subtracts memory contents and C flag’s complement from Accumulator or memory where is indicated by Index Register X</td>
</tr>
<tr>
<td>INC</td>
<td>Increment Accumulator or memory contents by 1</td>
</tr>
<tr>
<td>DEC</td>
<td>Decrement Accumulator or memory contents by 1</td>
</tr>
<tr>
<td>INX</td>
<td>Increment Index Register X by 1</td>
</tr>
<tr>
<td>DEX</td>
<td>Decrement Index Register X by 1</td>
</tr>
<tr>
<td>INY</td>
<td>Increment Index Register Y by 1</td>
</tr>
<tr>
<td>DEY</td>
<td>Decrement Index Register Y by 1</td>
</tr>
<tr>
<td><strong>Multiplication &amp; Division</strong></td>
<td></td>
</tr>
<tr>
<td>MUL (Note)</td>
<td>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</td>
</tr>
<tr>
<td>DIV (Note)</td>
<td>Quotient is stored in Accumulator and one’s complement of remainder is pushed onto stack</td>
</tr>
<tr>
<td><strong>Logical Operation</strong></td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>“AND” memory with Accumulator or memory where is indicated by Index Register X</td>
</tr>
<tr>
<td>ORA</td>
<td>“OR” memory with Accumulator or memory where is indicated by Index Register X</td>
</tr>
<tr>
<td>EOR</td>
<td>“Exclusive-OR” memory with Accumulator or memory where is indicated by Index Register X</td>
</tr>
<tr>
<td>COM</td>
<td>Store one’s complement of memory contents to memory</td>
</tr>
<tr>
<td>BIT</td>
<td>“AND” memory with Accumulator (The result is not stored into anywhere.)</td>
</tr>
<tr>
<td>TST</td>
<td>Test whether memory content is “0” or not</td>
</tr>
<tr>
<td><strong>Comparison</strong></td>
<td></td>
</tr>
<tr>
<td>CMP</td>
<td>Compare memory contents and Accumulator or memory where is indicated by Index Register X</td>
</tr>
<tr>
<td>CPX</td>
<td>Compare memory contents and Index Register X</td>
</tr>
<tr>
<td>CPY</td>
<td>Compare memory contents and Index Register Y</td>
</tr>
<tr>
<td><strong>Shift &amp; Rotate</strong></td>
<td></td>
</tr>
<tr>
<td>ASL</td>
<td>Shift left one bit (memory contents or Accumulator)</td>
</tr>
<tr>
<td>LSR</td>
<td>Shift right one bit (memory contents or Accumulator)</td>
</tr>
<tr>
<td>ROL</td>
<td>Rotate one bit left with carry (memory contents or Accumulator)</td>
</tr>
<tr>
<td>ROR</td>
<td>Rotate one bit right with carry (memory contents or Accumulator)</td>
</tr>
<tr>
<td>RRF</td>
<td>Rotate four bits right without carry (memory)</td>
</tr>
</tbody>
</table>

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

3.2.3 Bit managing instructions
The bit managing instructions clear “0” or set “1” designated bits of the Accumulator or memory.

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLB</td>
<td>Clear designated bit in the Accumulator or memory</td>
</tr>
<tr>
<td>SEB</td>
<td>Set designated bit in the Accumulator or memory</td>
</tr>
</tbody>
</table>

3.2.4 Flag setting instructions
The flag setting instructions clear “0” or set “1” C, D, I, T and V flags.

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLC</td>
<td>Clear C flag</td>
</tr>
<tr>
<td>SEC</td>
<td>Set C flag</td>
</tr>
<tr>
<td>CLD</td>
<td>Clear D flag</td>
</tr>
<tr>
<td>SED</td>
<td>Set D flag</td>
</tr>
<tr>
<td>CLI</td>
<td>Clear I flag</td>
</tr>
<tr>
<td>SEI</td>
<td>Set I flag</td>
</tr>
<tr>
<td>CLT</td>
<td>Clear T flag</td>
</tr>
<tr>
<td>SET</td>
<td>Set T flag</td>
</tr>
<tr>
<td>CLV</td>
<td>Clear V flag</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP</td>
<td>Jump to new location</td>
</tr>
<tr>
<td>BRA</td>
<td>Jump to new location</td>
</tr>
<tr>
<td>JSR</td>
<td>Jump to new location saving the current address</td>
</tr>
<tr>
<td>BBC</td>
<td>Branch when the designated bit in the Accumulator or memory is “0”</td>
</tr>
<tr>
<td>BBS</td>
<td>Branch when the designated bit in the Accumulator or memory is “1”</td>
</tr>
<tr>
<td>BCC</td>
<td>Branch when the C Flag is “0”</td>
</tr>
<tr>
<td>BCS</td>
<td>Branch when the C Flag is “1”</td>
</tr>
<tr>
<td>BNE</td>
<td>Branch when the Z Flag is “0”</td>
</tr>
<tr>
<td>BEQ</td>
<td>Branch when the Z Flag is “1”</td>
</tr>
<tr>
<td>BPL</td>
<td>Branch when the N Flag is “0”</td>
</tr>
<tr>
<td>BMI</td>
<td>Branch when the N Flag is “1”</td>
</tr>
<tr>
<td>BVC</td>
<td>Branch when the V Flag is “0”</td>
</tr>
<tr>
<td>BVS</td>
<td>Branch when the V Flag is “1”</td>
</tr>
<tr>
<td>RTI</td>
<td>Return from interrupt</td>
</tr>
<tr>
<td>RTS</td>
<td>Return from subroutine</td>
</tr>
</tbody>
</table>
### INSTRUCTIONS

**Instruction Set**

#### 3.2.6 Interrupt instruction (Break instruction)
This instruction causes a software interrupt.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt</td>
<td>BRK</td>
</tr>
<tr>
<td></td>
<td>Executes a software interrupt.</td>
</tr>
</tbody>
</table>

#### 3.2.7 Special instructions
These special instructions control the oscillation and the internal clock.

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Special</td>
<td>WIT</td>
</tr>
<tr>
<td></td>
<td>STP</td>
</tr>
<tr>
<td></td>
<td>Stops the internal clock.</td>
</tr>
<tr>
<td></td>
<td>Stops the oscillation of oscillator.</td>
</tr>
</tbody>
</table>

#### 3.2.8 Other instruction

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other</td>
<td>NOP</td>
</tr>
<tr>
<td></td>
<td>Only advances the program counter.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Accumulator</td>
<td>hh</td>
<td>Address high-order byte data in 0 to 255</td>
</tr>
<tr>
<td>Ai</td>
<td>Bit i of Accumulator</td>
<td>ll</td>
<td>Address low-order byte data in 0 to 255</td>
</tr>
<tr>
<td>PC</td>
<td>Program Counter</td>
<td>zz</td>
<td>Zero page address data in 0 to 255</td>
</tr>
<tr>
<td>PCL</td>
<td>Low-order byte of Program Counter</td>
<td>nn</td>
<td>Data in 0 to 255</td>
</tr>
<tr>
<td>PCH</td>
<td>High-order byte of Program Counter</td>
<td>*</td>
<td>Contents of the Program Counter</td>
</tr>
<tr>
<td>PS</td>
<td>Processor Status Register</td>
<td>i</td>
<td>Data in 0 to 7</td>
</tr>
<tr>
<td>S</td>
<td>Stack Pointer</td>
<td>#</td>
<td>Immediate mode</td>
</tr>
<tr>
<td>X</td>
<td>Index Register X</td>
<td>\</td>
<td>Special page mode</td>
</tr>
<tr>
<td>Y</td>
<td>Index Register Y</td>
<td>$</td>
<td>Hexadecimal symbol</td>
</tr>
<tr>
<td>M</td>
<td>Memory</td>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>Mi</td>
<td>Bit i of memory</td>
<td>–</td>
<td>Subtraction</td>
</tr>
<tr>
<td>C</td>
<td>Carry Flag</td>
<td>×</td>
<td>Multiplication</td>
</tr>
<tr>
<td>Z</td>
<td>Zero Flag</td>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>I</td>
<td>Interrupt Disable Flag</td>
<td>^</td>
<td>Logical AND</td>
</tr>
<tr>
<td>D</td>
<td>Decimal Operation Mode Flag</td>
<td>◊</td>
<td>Logical OR</td>
</tr>
<tr>
<td>B</td>
<td>Break Flag</td>
<td>∀</td>
<td>Logical exclusive OR</td>
</tr>
<tr>
<td>T</td>
<td>X Modified Operations Mode Flag</td>
<td>( )</td>
<td>Contents of register, memory, etc.</td>
</tr>
<tr>
<td>V</td>
<td>Overflow Flag</td>
<td>←</td>
<td>Direction of data transfer</td>
</tr>
<tr>
<td>N</td>
<td>Negative Flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REL</td>
<td>Relative address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BADRS</td>
<td>Break address</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Operation:** When \( (T) = 0 \), \((A) \leftarrow (A) + (M) + (C) \)

\( (T) = 1 \), \((M(X)) \leftarrow (M(X)) + (M) + (C) \)

**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.

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>ΔADCΔ#$nn</td>
<td>69₁₆, nn₁₆</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>ΔADCΔ$zz</td>
<td>65₁₆, zz₁₆</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Zero page X</td>
<td>ΔADCΔ$zz,X</td>
<td>75₁₆, zz₁₆</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Absolute</td>
<td>ΔADCΔ$hhl</td>
<td>6D₁₆, l₁₆, h₁₆</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Absolute X</td>
<td>ΔADCΔ$hhl,X</td>
<td>7D₁₆, l₁₆, h₁₆</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Absolute Y</td>
<td>ΔADCΔ$hhl,Y</td>
<td>79₁₆, l₁₆, h₁₆</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>(Indirect X)</td>
<td>ΔADCΔ($zz,X)</td>
<td>61₁₆, zz₁₆</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>(Indirect Y)</td>
<td>ΔADCΔ($zz),Y</td>
<td>71₁₆, zz₁₆</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Notes 1: When \( T=1 \), add 3 to the cycle number.
2: When ADC instruction is executed in the decimal operation mode (\( D = 1 \)), decision of \( C \) is delayed. Accordingly, do not execute the instruction which operates \( C \) such as SEC, CLC, etc.
**LOGICAL AND**

**Operation:** When \((T) = 0\), \((A) \leftarrow (A) \land (M)\)

\((T) = 1\), \((M(X)) \leftarrow (M(X)) \land (M)\)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>△AND△#$nn</td>
<td>29₁₆, nn₁₆</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>△AND△$zz</td>
<td>25₁₆, zz₁₆</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Zero page X</td>
<td>△AND△$zz,X</td>
<td>35₁₆, zz₁₆</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Absolute</td>
<td>△AND△$llll</td>
<td>2D₁₆, ll₁₆, hh₁₆</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Absolute X</td>
<td>△AND△$llll,X</td>
<td>3D₁₆, ll₁₆, hh₁₆</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Absolute Y</td>
<td>△AND△$llll,Y</td>
<td>39₁₆, ll₁₆, hh₁₆</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>(Indirect X)</td>
<td>△AND△($ zz,X)</td>
<td>21₁₆, zz₁₆</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>(Indirect Y)</td>
<td>△AND△($ zz),Y</td>
<td>31₁₆, zz₁₆</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Note: When \(T = 1\), add 3 to a cycle number.
Operation: $C \leftarrow b7 \quad b0 \leftarrow 0$

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.

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>ΔASLΔA</td>
<td>0A16</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>ΔASLΔ$zz$</td>
<td>0616, zz16</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Zero page X</td>
<td>ΔASLΔ$zz,X$</td>
<td>1616, zz16</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Absolute</td>
<td>ΔASLΔ$hll$</td>
<td>0E16, ll16, hh16</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Absolute X</td>
<td>ΔASLΔ$hll,X$</td>
<td>1E16, ll16, hh16</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>
**BBC**

**BRANCH ON BIT CLEAR**

**Operation:** When \( (Mi) \) or \( (Ai) = 0 \), \( (PC) \leftarrow (PC) + n + REL \)

\( (Mi) \) or \( (Ai) = 1 \), \( (PC) \leftarrow (PC) + n \)

\( n \): 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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator bit</td>
<td>( \Delta \text{BBC} ), ( A ), ( $hhll )</td>
<td>((20i+13)<em>{16}, r</em>{16})</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Relative</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zero page bit</td>
<td>( \Delta \text{BBC} ), ( $zz, $hhll )</td>
<td>((20i+17)<em>{16}, z</em>{16}, r_{16})</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Relative</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes:
1: \( r_{16} = $hhll - *(\times + n) \). The \( r_{16} \) 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, \((PC) \leftarrow (PC) + n + REL\)

\((Mi)\) or \((Ai)\) = 0, \((PC) \leftarrow (PC) + n\)

\(n\) : 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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator bit Relative</td>
<td>(\Delta\text{BBS}\Delta i,A,$hh$ll)</td>
<td>((20i+3)<em>{16}, \text{rr}</em>{16})</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Zero page bit Relative</td>
<td>(\Delta\text{BBS}\Delta i,$zz,$hh$ll)</td>
<td>((20i+7)<em>{16}, \text{zz}</em>{16}, \text{rr}_{16})</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Notes:
1: \(\text{rr}_{16}=\$hh\$ll-(\#+n)\). The \(\text{rr}_{16}\) 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**

**Operation**: When \((C) = 0\), \((PC) \leftarrow (PC) + 2 + REL\)
\((C) = 1\), \((PC) \leftarrow (PC) + 2\)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative</td>
<td>ΔBCCΔ$hll$</td>
<td>$90_{16}$, $rr_{16}$</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Notes:
1: $rr_{16}=$$hll$–($*$+2$). 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**

**Operation**: When \((C) = 1\), \((PC) \leftarrow (PC) + 2 + REL\)

\((C) = 0\), \((PC) \leftarrow (PC) + 2\)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative</td>
<td>ΔBCSΔ$hhll</td>
<td>B016, rr16</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Notes 1: \(rr16=$hhll–(*)+2\). The \(rr16\) is a value in a range of \(-128\) to \(+127\).

2: When a branch is executed, add 2 to the cycle number.
**BEQ**  
**BRANCH ON EQUAL**

**Operation**: When \((Z) = 1\), \((PC) \leftarrow (PC) + 2 + \text{REL}\)  
\((Z) = 0\), \((PC) \leftarrow (PC) + 2\)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative</td>
<td>(\Delta\text{BEQ}\A$\text{hhll}$</td>
<td>F016,rr16</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Notes 1: \(rr16=hhll-(\star+2)\). The \(rr16\) 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**

**Operation**: \((A) \land (M)\)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero page</td>
<td>ΔBITΔ$zz</td>
<td>24₁₀, zz₁₆</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Absolute</td>
<td>ΔBITΔ$hll</td>
<td>2C₁₀, ll₁₆, hh₁₆</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
**Operation**: When \( (N) = 1 \), \((PC) \leftarrow (PC) + 2 + \text{REL}\)
\( (N) = 0, (PC) \leftarrow (PC) + 2 \)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative</td>
<td>ΔBMIΔ$hll$</td>
<td>30₁₆, $rr₁₆$</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Notes 1: \( rr₁₆ = $hhll – (\ast + 2) \). The \( rr₁₆ \) 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 \((Z) = 0\), \((PC) \leftarrow (PC) + 2 + REL\)
\((Z) = 1\), \((PC) \leftarrow (PC) + 2\)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative</td>
<td>(\Delta\text{BNE}\Delta$h&lt;hl])</td>
<td>D016, (rr_{16})</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Notes 1: \(rr_{16}=$hl\−(\ast+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 \((N) = 0\), \((PC) \leftarrow (PC) + 2 + REL\)
\((N) = 1\), \((PC) \leftarrow (PC) + 2\)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative</td>
<td>ΔBPLΔ$hll$</td>
<td>1016, rr16</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Notes:
1: \(rr16 = hll - (*+2)\). The \(rr16\) is a value in a range of \(-128\) to \(+127\).
2: When a branch is executed, add 2 to the cycle number.
**BRA**

**BRANCH ALWAYS**

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative</td>
<td>ΔBRAΔ$h\text{ll}$</td>
<td>8016, $rr_{16}$</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Note: $rr_{16}=$ $h\text{ll}$–($*$+2). The $rr_{16}$ is a value in a range of –128 to +127.
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

### 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 \((V) = 0\), \((PC) \leftarrow (PC) + 2 + \text{REL}\)  
\((V) = 1\), \((PC) \leftarrow (PC) + 2\)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative</td>
<td>(\Delta\text{BVC}\Delta$hll)</td>
<td>50_{16}, $rr_{16}$</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

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.
BVS  
BRANCH ON OVERFLOW SET

**Operation**: When \((V) = 1\), \((PC) \leftarrow (PC) + 2 + \text{REL}\)
\((V) = 0\), \((PC) \leftarrow (PC) + 2\)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative</td>
<td>ΔBVSΔ$hll$</td>
<td>70(<em>{16}), (rr(</em>{16})</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

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**

**Operation**: \((Ai) \leftarrow 0, \text{ or } (Mi) \leftarrow 0\)

**Function**: This instruction clears the designated bit \(i\) of A or M.

**Status flag**: No change

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator bit</td>
<td>(\Delta \text{CLB}\Delta i,A)</td>
<td>((20i+1B)_{16})</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>((20i+1F)<em>{16}, ZZ</em>{16})</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Zero page bit</td>
<td>(\Delta \text{CLB}\Delta i,$zz)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CLC
CLEAR CARRY FLAG

Operation: \((C) \leftarrow 0\)

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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔCLC</td>
<td>18_{16}</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
**CLD**

**CLEAR DECIMAL MODE**

**Operation:** \((D) \leftarrow 0\)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔCLD</td>
<td>D8_{16}</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
CLI

CLEAR INTERRUPT DISABLE STATUS

Operation: \((I) \leftarrow 0\)

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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔCLI</td>
<td>58\text{16}</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

This instruction clears I.
Operation: \((T) \leftarrow 0\)

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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>(\Delta CLT)</td>
<td>(12_{16})</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
**CLV**

**CLEAR OVERFLOW FLAG**

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔCLV</td>
<td>B8&lt;sub&gt;16&lt;/sub&gt;</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
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 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.

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>ΔCMPΔ#$nn</td>
<td>C916, nn16</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>ΔCMPΔ$zz</td>
<td>C516, zz16</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Zero page X</td>
<td>ΔCMPΔ$zz,X</td>
<td>D516, zz16</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Absolute</td>
<td>ΔCMPΔ$hll</td>
<td>CD16, ll16, hh16</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Absolute X</td>
<td>ΔCMPΔ$hll,X</td>
<td>DD16, ll16, hh16</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Absolute Y</td>
<td>ΔCMPΔ$hll,Y</td>
<td>D916, ll16, hh16</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>(Indirect X)</td>
<td>ΔCMPΔ($zz,X)</td>
<td>C116, zz16</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>(Indirect Y)</td>
<td>ΔCMPΔ($zz),Y</td>
<td>D116, zz16</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Note: When T=1, add 1 to the cycle number.
Operation: \((M) \leftarrow \overline{(M)}\)

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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero page</td>
<td>(\Delta\text{COM}\mathbf{A}$zz)</td>
<td>44_{16}, zz_{16}</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
**CPX**

**COMPARE MEMORY AND INDEX REGISTER X**

**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.

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>(\Delta CPX\Delta #nn)</td>
<td>(E0_{16}, nn_{16})</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>(\Delta CPX\Delta #zz)</td>
<td>(E4_{16}, zz_{16})</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Absolute</td>
<td>(\Delta CPX\Delta #hhll)</td>
<td>(EC_{16}, ll_{16}, hh_{16})</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
**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.

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>(\triangle CPY#nn)</td>
<td>C016, nn16</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>(\triangle CPY$zz)</td>
<td>C416, zz16</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Absolute</td>
<td>(\triangle CPY$hhll)</td>
<td>CC16, ll16, hh16</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Operation: \((A) \leftarrow (A) - 1, \text{ or } (M) \leftarrow (M) - 1\)

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
- \(Z\): \(Z\) is 1 when the operation result is 0; otherwise \(Z\) is 0.
- \(C\): No change

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>(\Delta\text{DEC}\Delta A)</td>
<td>1A_{16}</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>(\Delta\text{DEC}\Delta $zz)</td>
<td>C6_{16}, zz_{16}</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Zero page X</td>
<td>(\Delta\text{DEC}\Delta $zz,X)</td>
<td>D6_{16}, zz_{16}</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Absolute</td>
<td>(\Delta\text{DEC}\Delta $hhll)</td>
<td>CE_{16}, ll_{16}, hh_{16}</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Absolute X</td>
<td>(\Delta\text{DEC}\Delta $hhll,X)</td>
<td>DE_{16}, ll_{16}, hh_{16}</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>
Operation : \((X) \leftarrow (X) - 1\)

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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔDEX</td>
<td>CA16</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
**DEY**

DECREMENT INDEX REGISTER Y BY ONE

**Operation:** \((Y) \leftarrow (Y) - 1\)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔDEY</td>
<td>8816</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
DIVIDE MEMORY BY ACCUMULATOR

**Operation**:\( (A) \leftarrow (M(zz+(X)+1), M(zz+(X))) / (A) \)
\( M(S) \leftarrow \text{one's complement of Remainder} \)
\( (S) \leftarrow (S) - 1 \)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero page X</td>
<td>ΔDIVΔ$zz,X</td>
<td>E2₁₆, zz₁₆</td>
<td>2</td>
<td>16</td>
</tr>
</tbody>
</table>

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 any products.
**EOR**

**EXCLUSIVE OR MEMORY WITH ACCUMULATOR**

**Operation:** When \(T = 0\), \((A) \leftarrow (A) \lor (M)\)

\((T) = 1\), \((M(X)) \leftarrow (M(X)) \lor (M)\)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>ΔEORΔ#$nn</td>
<td>4916, nn16</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>ΔEORΔ$zz</td>
<td>4516, zz16</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Zero page X</td>
<td>ΔEORΔ$zz,X</td>
<td>5516, zz16</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Absolute X</td>
<td>ΔEORΔ$hhll</td>
<td>4D16, ll16, hh16</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Absolute X</td>
<td>ΔEORΔ$hhll,X</td>
<td>5D16, ll16, hh16</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Absolute Y</td>
<td>ΔEORΔ$hhll,Y</td>
<td>5916, ll16, hh16</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>(Indirect X)</td>
<td>ΔEORΔ($zz,X)</td>
<td>4116, zz16</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>(Indirect Y)</td>
<td>ΔEORΔ($zz),Y</td>
<td>5116, zz16</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Note: When \(T=1\), add 3 to the cycle number.
**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>ΔINCΔA</td>
<td>3A16</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>ΔINCΔ$zz</td>
<td>E616, zz16</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Zero page X</td>
<td>ΔINCΔ$zz,X</td>
<td>F616, zz16</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Absolute</td>
<td>ΔINCΔ$hll</td>
<td>EE16, ll16, hh16</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Absolute X</td>
<td>ΔINCΔ$hll,X</td>
<td>FE16, ll16, hh16</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

(A) ← (A) + 1, or  
(M) ← (M) + 1  

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

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
INX

INCREMENT INDEX REGISTER X BY ONE

Operation: \((X) \leftarrow (X) + 1\)

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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>(\Delta\text{INX})</td>
<td>E816</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
**INY**

**INCREMENT INDEX REGISTER Y BY ONE**

**Operation**: \((Y) \leftarrow (Y) + 1\)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>(\Delta)INY</td>
<td>C816</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
**JMP**

**Operation:** When addressing mode is
(a) Absolute, then
   (PC) ← hhll
(b) Indirect Absolute, then
   (PC)<sub>L</sub> ← (hhll)
   (PC)<sub>H</sub> ← (hhll+1)
(c) Zero page Indirect Absolute, then
   (PC)<sub>L</sub> ← (zz)
   (PC)<sub>H</sub> ← (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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute</td>
<td>ΔJMPΔ$hhll$</td>
<td>4C&lt;sub&gt;16&lt;/sub&gt;,ll&lt;sub&gt;16&lt;/sub&gt;,hh&lt;sub&gt;16&lt;/sub&gt;</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Indirect Absolute</td>
<td>ΔJMPΔ($hhll$)</td>
<td>6C&lt;sub&gt;16&lt;/sub&gt;,ll&lt;sub&gt;16&lt;/sub&gt;,hh&lt;sub&gt;16&lt;/sub&gt;</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Zero Page Indirect</td>
<td>ΔJMPΔ($zz$)</td>
<td>B2&lt;sub&gt;16&lt;/sub&gt;,zz&lt;sub&gt;16&lt;/sub&gt;</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
**JSR**

**Operation:**

\[
\begin{align*}
(M(S)) & \leftarrow (PCH) \\
(S) & \leftarrow (S) - 1 \\
(M(S)) & \leftarrow (PC_L) \\
(S) & \leftarrow (S) - 1
\end{align*}
\]

After the above operations, if the addressing mode is

(a) Absolute, then

\[
(\text{PC}) \leftarrow \text{hhll}
\]

(b) Special page, then

\[
(\text{PC}_L) \leftarrow \text{ll} \\
(\text{PC}_H) \leftarrow \text{FF}_{16}
\]

(c) Zero page Indirect, then

\[
(\text{PC}_L) \leftarrow (\text{zz}) \\
(\text{PC}_H) \leftarrow (\text{zz+1})
\]

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute</td>
<td>ΔJSRΔ$h$ll</td>
<td>$20_{16}$, $ll_{16}$, $hh_{16}$</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Special page</td>
<td>ΔJSRΔ$h$ll</td>
<td>$22_{16}$, $ll_{16}$</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Zero page Indirect</td>
<td>ΔJSRΔ($zz$)</td>
<td>$02_{16}$, $zz_{16}$</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

(Note) “\"” (5C₁₆ of the ASCII code) denotes special page. $hh_{16}$ must be $FF_{16}$ in the special page addressing mode.

(Note)
LDA
LOAD ACCUMULATOR WITH MEMORY

**Operation:**
When \((T) = 0\), \((A) \leftarrow (M)\)
\((T) = 1\), \((M(X)) \leftarrow (M)\)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>(\Delta\text{LDA}\Delta#$nn)</td>
<td>A9(<em>{16}), $nn(</em>{16})</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>(\Delta\text{LDA}\Delta$zz)</td>
<td>A5(<em>{16}), $zz(</em>{16})</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Zero page X</td>
<td>(\Delta\text{LDA}\Delta$zz,X)</td>
<td>B5(<em>{16}), $zz(</em>{16})</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Absolute</td>
<td>(\Delta\text{LDA}\Delta$hhll)</td>
<td>AD(<em>{16}), ll(</em>{16}), hh(_{16})</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Absolute X</td>
<td>(\Delta\text{LDA}\Delta$hll,X)</td>
<td>BD(<em>{16}), ll(</em>{16}), hh(_{16})</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Absolute Y</td>
<td>(\Delta\text{LDA}\Delta$hll,Y)</td>
<td>B9(<em>{16}), ll(</em>{16}), hh(_{16})</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>(Indirect X)</td>
<td>(\Delta\text{LDA}\Delta($zz,X))</td>
<td>A1(<em>{16}), zz(</em>{16})</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>(Indirect Y)</td>
<td>(\Delta\text{LDA}\Delta($zz),Y)</td>
<td>B1(<em>{16}), zz(</em>{16})</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Note: When \(T = 1\), add 2 to the cycle number.
LDM
LOAD IMMEDIATE DATA TO MEMORY

Operation: \((M) \leftarrow nn\)

Function: This instruction loads the immediate value in M.

Status flag: No change

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero page</td>
<td>\texttt{\textDelta LDM\textDelta #nn,$zz}</td>
<td>3C\textsubscript{16}, nn\textsubscript{16}, zz\textsubscript{16}</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
**LDX**

**LOAD INDEX REGISTER X FROM MEMORY**

**Operation**: \((X) \leftarrow (M)\)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>ΔLDXΔ#$nn</td>
<td>A2(<em>{16}), nn(</em>{16})</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>ΔLDXΔ$zz</td>
<td>A6(<em>{16}), zz(</em>{16})</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Zero page Y</td>
<td>ΔLDXΔ$zz,Y</td>
<td>B6(<em>{16}), zz(</em>{16})</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Absolute</td>
<td>ΔLDXΔ$hhll</td>
<td>AE(<em>{16}), ll(</em>{16}), hh(_{16})</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Absolute Y</td>
<td>ΔLDXΔ$hhll,Y</td>
<td>BE(<em>{16}), ll(</em>{16}), hh(_{16})</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
**LDY**

**LOAD INDEX REGISTER Y FROM MEMORY**

**Operation**: \( (Y) \leftarrow (M) \)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>( \Delta \text{LDY} #nn )</td>
<td>A0\text{16}, nn\text{16}</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>( \Delta \text{LDY} $zz )</td>
<td>A4\text{16}, zz\text{16}</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Zero page X</td>
<td>( \Delta \text{LDY} $zz,X )</td>
<td>B4\text{16}, zz\text{16}</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Absolute</td>
<td>( \Delta \text{LDY} $hhll )</td>
<td>AC\text{16}, ll\text{16}, hh\text{16}</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Absolute X</td>
<td>( \Delta \text{LDY} $hhll,X )</td>
<td>BC\text{16}, ll\text{16}, hh\text{16}</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
**LSR**  
**LOGICAL SHIFT RIGHT**

**Operation:**  
0 → \[b_7\] [ ] [ ] [ ] [ ] [ ] [ ] [ ] b_0 → 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.

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>∆LSRAA</td>
<td>4A_{16}</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>∆LSRA$zz$</td>
<td>4616, ZZ16</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Zero page X</td>
<td>∆LSRA$zz,X</td>
<td>5616, ZZ16</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Absolute</td>
<td>∆LSRA$hll$</td>
<td>4E16, l16, hh16</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Absolute X</td>
<td>∆LSRA$hll,X$</td>
<td>5E16, l16, hh16</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>
**MUL**

**MULTIPLY ACCUMULATOR AND MEMORY**

**Operation**: 

\[ M(S) \cdot (A) \leftarrow (A) \times M(zz+(X)) \]

\[ (S) \leftarrow (S) - 1 \]

**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.
**NOP**

**NO OPERATION**

**Operation**: \((PC) \leftarrow (PC) + 1\)

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

**Status flag**: No change

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>(\Delta\text{NOP})</td>
<td>(\text{EA}_{16})</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
ORA

OR MEMORY WITH ACCUMULATOR

**Operation:** When \( T = 0 \), \( (A) \leftarrow (A) \lor (M) \)
\[ (T) = 1, \ (M(X)) \leftarrow (M(X)) \lor (M) \]

**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 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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>( \text{ORA} \Delta#nn )</td>
<td>0916, nn16</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>( \text{ORA} \Delta$zz )</td>
<td>0516, zz16</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Zero page X</td>
<td>( \text{ORA} \Delta$zz,X )</td>
<td>1516, zz16</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Absolute</td>
<td>( \text{ORA} \Delta$hll )</td>
<td>0D16, ll16, hh16</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Absolute X</td>
<td>( \text{ORA} \Delta$hll,X )</td>
<td>1D16, ll16, hh16</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Absolute Y</td>
<td>( \text{ORA} \Delta$hll,Y )</td>
<td>1916, ll16, hh16</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>(Indirect X)</td>
<td>( \text{ORA} \Delta($zz,X) )</td>
<td>0116, zz16</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>(Indirect Y)</td>
<td>( \text{ORA} \Delta($zz),Y )</td>
<td>1116, zz16</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Note: When \( T=1 \), add 3 to the cycle number.
**Operation**: 
\[(M(S)) \leftarrow (A)
(S) \leftarrow (S) - 1\]

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔPHA</td>
<td>48₁₆</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
**PHP**

**PUSH** PROCESSOR STATUS ON STACK

**Operation**: 
\[(M(S)) \leftarrow (PS)\]
\[(S) \leftarrow (S) - 1\]

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔPHP</td>
<td>08₁₆</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
PLA
PULL ACCUMULATOR FROM STACK

Operation: \((S) \leftarrow (S) + 1\)
\((A) \leftarrow (M(S))\)

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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔPLA</td>
<td>6816</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

77
**PLP**

**Pull Processor Status from Stack**

**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.

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔPLP</td>
<td>28_{16}</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
**ROL**

**ROTATE ONE BIT LEFT**

**Operation:**

```
 b7   b6   b5   b4   b3   b2   b1   b0  
    →   →   →   →   →   →   →   →   →   C
```

**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.

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>ΔROLΔA</td>
<td>2A₁₆</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>ΔROLΔ$zz</td>
<td>2₆₁₆, ZZ₁₆</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Zero page X</td>
<td>ΔROLΔ$zz,X</td>
<td>3₆₁₆, ZZ₁₆</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Absolute</td>
<td>ΔROLΔ$hhll</td>
<td>2E₁₆, ll₁₆, hh₁₆</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Absolute X</td>
<td>ΔROLΔ$hhll,X</td>
<td>3E₁₆, ll₁₆, hh₁₆</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>
Operation:

```
  C  b7                   b0
```

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.

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>ΔRORΔA</td>
<td>6A₁₆</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>ΔRORΔ$zz</td>
<td>66₁₆, ZZ₁₆</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Zero page X</td>
<td>ΔRORΔ$zz,X</td>
<td>76₁₆, ZZ₁₆</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Absolute</td>
<td>ΔRORΔ$hll</td>
<td>6E₁₆, ll₁₆, hh₁₆</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Absolute X</td>
<td>ΔRORΔ$hll,X</td>
<td>7E₁₆, ll₁₆, hh₁₆</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>
**RRF**

**ROTATE RIGHT OF FOUR BITS**

**Operation:**

```
    b7   b4   b3   b0
```

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

**Status flag:** No change

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero page</td>
<td>ΔRRFΔ$zz</td>
<td>82_16, zz_16</td>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>
RETURN FROM INTERRUPT

**Operation**: 
(S) ← (S) + 1
(PS) ← (M(S))
(S) ← (S) + 1
(PCL) ← (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 PCL. S is again incremented by one and stores the contents of memory location designated by S in PC_H.

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

**Status flag**: (S) ← (S) + 1

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔRTI</td>
<td>4016</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>
Operation: \[(PC_L) \leftarrow (M(S))\]
\[(S) \leftarrow (S) + 1\]
\[(PC_H) \leftarrow (M(S))\]
\[(PC) \leftarrow (PC) + 1\]

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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔRTS</td>
<td>6016</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>
SBC

Subtract with Carry

Operation: When \( T = 0 \), \( (A) \leftarrow (A) - (M) - (\overline{C}) \)
\( (T) = 1, (M(X)) \leftarrow (M(X)) - (M) - (\overline{C}) \)

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.

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>( \triangle ) SBC ( \Delta ) #nn</td>
<td>E9\text{16}, nn\text{16}</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Zero page</td>
<td>( \triangle ) SBC ( \Delta ) $zz</td>
<td>E5\text{16}, zz\text{16}</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Zero page X</td>
<td>( \triangle ) SBC ( \Delta ) $zz,X</td>
<td>F5\text{16}, zz\text{16}</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Absolute</td>
<td>( \triangle ) SBC ( \Delta ) $hhll</td>
<td>E\text{D16}, ll\text{16}, hh\text{16}</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Absolute X</td>
<td>( \triangle ) SBC ( \Delta ) $hhll,X</td>
<td>F\text{D16}, ll\text{16}, hh\text{16}</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Absolute Y</td>
<td>( \triangle ) SBC ( \Delta ) $hhll,Y</td>
<td>F\text{916}, ll\text{16}, hh\text{16}</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>(Indirect X)</td>
<td>( \triangle ) SBC ( \Delta ) $(zz,X)$</td>
<td>E\text{116}, zz\text{16}</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>(Indirect Y)</td>
<td>( \triangle ) SBC ( \Delta ) $(zz),Y$</td>
<td>F\text{116}, zz\text{16}</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Notes 1: When \( T=1 \), add 3 to the cycle number.
2: When SBC instruction is executed in the decimal operation mode
\((D = 1)\), decision of \( C \) is delayed. Accordingly, do not execute the
instruction which operates \( C \) such as SEC, CLC, etc.
**SEB**

**Operation:** \((Ai) \leftarrow 1\), or
\((Mi) \leftarrow 1\)

**Function:** This instruction sets the designated bit \(i\) of \(A\) or \(M\).

**Status flag:** No change

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator bit</td>
<td>(\triangle SEB \hat{i}, A)</td>
<td>((20i+B)_{16})</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Zero page bit</td>
<td>(\triangle SEB \hat{i}, $zz)</td>
<td>((20i+F)<em>{16}, zz</em>{16})</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
Operation: \((C) \leftarrow 1\)

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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine code</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔSEC</td>
<td>(38_{16})</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
**Operation:** \((D) \leftarrow 1\)

**Function:** This instruction sets 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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>(\Delta\text{SED})</td>
<td>F8_{16}</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Operation: \((l) \leftarrow 1\)

Function: This instruction sets \(l\).

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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔSEI</td>
<td>7816</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Operation: \((T) \leftarrow 1\)

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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>(\Delta)SET</td>
<td>3216</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
STA

STORE ACCUMULATOR IN MEMORY

**Operation:** \((M) \leftarrow (A)\)

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

**Status flag:** No change

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero page</td>
<td>(\Delta \text{STA}\Delta $zz)</td>
<td>8516, ZZ16</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Zero page X</td>
<td>(\Delta \text{STA}\Delta $zz,X)</td>
<td>9516, ZZ16</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Absolute</td>
<td>(\Delta \text{STA}\Delta $hhll)</td>
<td>8D16, lI16, hh16</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Absolute X</td>
<td>(\Delta \text{STA}\Delta $hhll,X)</td>
<td>9D16, lI16, hh16</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Absolute Y</td>
<td>(\Delta \text{STA}\Delta $hhll,Y)</td>
<td>9916, lI16, hh16</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>(Indirect X)</td>
<td>(\Delta \text{STA}\Delta ($zz,X))</td>
<td>8116, ZZ16</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>(Indirect Y)</td>
<td>(\Delta \text{STA}\Delta ($zz),Y)</td>
<td>9116, ZZ16</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>
**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔSTP</td>
<td>4216</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

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: \((M) \leftarrow (X)\)

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

Status flag: No change

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero page</td>
<td>(\Delta STX\Delta $)zz</td>
<td>86\text{16}, ZZ\text{16}</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Zero page Y</td>
<td>(\Delta STX\Delta $)zz,Y</td>
<td>96\text{16}, ZZ\text{16}</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Absolute</td>
<td>(\Delta STX\Delta $hh ll)</td>
<td>8E\text{16}, ll\text{16}, hh\text{16}</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
**Operation**: \((M) \leftarrow (Y)\)

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

**Status flag**: No change

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero page</td>
<td>(\Delta STY \Delta $zz)</td>
<td>(84_{16}, $z16)</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Zero page X</td>
<td>(\Delta STY \Delta $zz.X)</td>
<td>(94_{16}, $z16)</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Absolute</td>
<td>(\Delta STY \Delta $hhll)</td>
<td>(8C_{16}, ll_{16}, hh_{16})</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
**TAX**

**TRANSFER ACCUMULATOR TO INDEX REGISTER X**

**Operation:** \((X) \leftarrow (A)\)

**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔTAX</td>
<td>AA(_{16})</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Operation : \((Y) \leftarrow (A)\)

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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>(\Delta TAY)</td>
<td>A8(_{16})</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
**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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero page</td>
<td>( \Delta \text{TST} \Delta $zz )</td>
<td>64_{16}, zz_{16}</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
**TSX**  
**TRANSFER STACK 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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔTSX</td>
<td>BA₁₆</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
TXA
TRANSFER INDEX REGISTER X TO ACCUMULATOR

Operation: \((A) \leftarrow (X)\)

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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔTXA</td>
<td>8A₁₆</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
**Operation:** \((S) \leftarrow (X)\)

**Function:** This instruction stores the contents of \(X\) in \(S\).

**Status flag** No change

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>(\Delta TXS)</td>
<td>9A(_{16})</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
TYA
TRANSFER INDEX REGISTER Y TO ACCUMULATOR

Operation: \( (A) \leftarrow (Y) \)

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

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>ΔTYA</td>
<td>(98_{16})</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
**Operation**: CPU $\leftarrow$ Wait state

**Function**: The WIT instruction stops the internal clock but not the oscillation of the oscillation circuit is not stopped. CPU starts its function after the Timer X over flows (comes to the terminal count). All registers or internal memory contents except Timer X will not change during this mode. (Of course needs VDD).

**Status flag**: No change

<table>
<thead>
<tr>
<th>Addressing mode</th>
<th>Statement</th>
<th>Machine codes</th>
<th>Byte number</th>
<th>Cycle number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied</td>
<td>$\Delta$WIT</td>
<td>C2₁₆</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
4. NOTES ON USE

4.1 Notes on interrupts

4.1.1 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.”

4.1.2 Switching of detection edge
For the products able to switch the external interrupt detection edge, switch it as the following sequence.

Fig. 4.1.1 Switching sequence of detection edge

●Reason
The interrupt circuit recognizes the switching of the detection edge as the change of external input signals. This may cause to execute an unnecessary interrupt processing routine.
4.1.3 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” by using a data transfer instruction*, execute one or more instructions before executing the BBC or BBS instruction.

* Data transfer instruction: LDM, LDA, STA, STX, STY

**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.
4.2 Notes on programming

4.2.1 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. 4.2.1 Initialization of flags in Processor Status Register](image)

(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. 4.2.2 PLP instruction execution sequence](image)

![Fig. 4.2.3 Stack memory contents after PHP instruction execution](image)
4.2.2 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. 4.2.4 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.

4.2.3 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.
NOTES ON USE

(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. 4.2.5 Status flags in decimal mode

4.2.4 JMP instruction
When using the JMP instruction in indirect addressing mode, do not specify the last address on a page as an indirect address.
APPENDIX 1

Instruction Cycles in each Addressing Mode

APPENDIX 1. Instruction Cycles in each Addressing Mode

Clock $\phi$ 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 $\phi$. 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.

In these figures, $\phi$, SYNC, R/W (RD, WR), ADDR (ADDR L, ADDR H), and DATA are internal signals of the single-chip microcomputer; therefore, these signals can be investigated only in the 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

<table>
<thead>
<tr>
<th>Type</th>
<th>$\phi$</th>
<th>SYNC</th>
<th>R/W</th>
<th>R</th>
<th>W</th>
<th>ADDR</th>
<th>DATA</th>
<th>ADDR H</th>
<th>ADDR L/DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>M507XX</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>M509XX</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>M374XX</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>(Except M37451)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>M38XXX</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>M375XX</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>M37451</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>M50734</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>
Instructions:

- ΔCLC
- ΔCLD
- ΔCLI
- ΔCLT
- ΔCLV
- ΔDEX
- ΔDEY
- ΔINX
- ΔINY
- ΔNOP
- ΔSEC
- ΔSED
- ΔSEI
- ΔSET
- ΔTAX
- ΔTAY
- ΔTSX
- ΔTXA
- ΔTXS
- ΔTYA

Byte length: 1
Cycle number: 2

Timing:

\[
\begin{array}{c}
\phi \\
\text{SYNC} \\
\text{R/W} \\
\text{RD} \\
\text{WR} \\
\text{ADDR} \\
\text{DATA} \\
\text{ADDRH} \\
\text{ADDRL} / \text{DATA}
\end{array}
\]

Invalid
### IMPLIED

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

<table>
<thead>
<tr>
<th>Timing Event</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi )</td>
<td><img src="image" alt="Waveform" /></td>
</tr>
<tr>
<td>SYNC</td>
<td><img src="image" alt="Waveform" /></td>
</tr>
<tr>
<td>R/W</td>
<td><img src="image" alt="Waveform" /></td>
</tr>
<tr>
<td>RD</td>
<td><img src="image" alt="Waveform" /></td>
</tr>
<tr>
<td>WR</td>
<td><img src="image" alt="Waveform" /></td>
</tr>
</tbody>
</table>

#### DATA

<table>
<thead>
<tr>
<th>Address</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDR</td>
<td><img src="image" alt="Address Diagram" /></td>
</tr>
<tr>
<td>DATA</td>
<td><img src="image" alt="Data Diagram" /></td>
</tr>
<tr>
<td>ADDRH</td>
<td><img src="image" alt="Address High Diagram" /></td>
</tr>
<tr>
<td>ADDRDL</td>
<td><img src="image" alt="Address Low Diagram" /></td>
</tr>
</tbody>
</table>

**Notes:**
1. Some products are "01" or content of SPS flag.  
2. Some products differ the address.
Instructions: \texttt{\textasciitilde STP}, \texttt{\textasciitilde WIT}

Byte length: 1

Timing:

Return from standby state is executed by external interrupt.

Return from wait state is executed by internal or external interrupt.
### RTI Instruction

- **Instruction:** RTI
- **Byte length:** 1
- **Cycle number:** 6
- **Timing:**

<table>
<thead>
<tr>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi )</td>
<td>( \text{SYNC} )</td>
</tr>
<tr>
<td>( \overline{R/W} )</td>
<td>( \overline{RD} )</td>
</tr>
<tr>
<td>( \overline{WR} )</td>
<td></td>
</tr>
</tbody>
</table>

#### Timing Diagram

- \( \phi \): Timing phases
- \( \text{SYNC} \): Synchronization phase
- \( \overline{R/W} \): Read/Write phase
- \( \overline{RD} \): Data Read phase
- \( \overline{WR} \): Data Write phase

#### Address and Data Registers

<table>
<thead>
<tr>
<th>ADDR</th>
<th>PC</th>
<th>PC+1</th>
<th>S,00 ( \text{(Note)} )</th>
<th>S+1,00 ( \text{(Note)} )</th>
<th>S+2,00 ( \text{(Note)} )</th>
<th>S+3,00 ( \text{(Note)} )</th>
<th>PCL</th>
<th>PCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>Op-code</td>
<td>Invalid</td>
<td>Invalid</td>
<td>PS ( \text{(Back)} )</td>
<td>PCL ( \text{(Back)} )</td>
<td>PCH ( \text{(Back)} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDRH</td>
<td>PCH</td>
<td>PCH</td>
<td>00 ( \text{(Note)} )</td>
<td>PCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDR/L</td>
<td>PCL</td>
<td>Op-code</td>
<td>PCL+1</td>
<td>Invalid</td>
<td>S</td>
<td>S+1</td>
<td>PS</td>
<td>S+2</td>
</tr>
</tbody>
</table>

**Note:** Some products are "01" or content of SPS flag.
Instruction: 
Byte length: 1
Cycle number: 6

Timing:

SYNC

R/W
RD
WR

ADDR

DATA

ADDRH

ADDRL /DATA

Note: Some products are "01" or content of SPS flag.
Instructions: ΔPHA
         ΔPHP
Byte length: 1
Cycle number: 3
Timing:

Φ

SYNC

R/W

RD

WR

ADDR

DATA

ADDRH

ADDRL

/DATA

Note: Some products are "01" or content of SPS flag.
Instructions: ΔPLA
ΔPLP
Byte length: 1
Cycle number: 4
Timing:

(note)

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

**Instructions**
- ΔADC Δ#$nn (T=0)
- ΔAND Δ#$nn (T=0)
- ΔCMP Δ#$nn (T=0)
- ΔCPX Δ#$nn
- ΔCPY Δ#$nn
- ΔEOR Δ#$nn (T=0)
- ΔLDA Δ#$nn (T=0)
- ΔLDX Δ#$nn
- ΔLDY Δ#$nn
- ΔORA Δ#$nn (T=0)
- ΔSBC Δ#$nn (T=0)

**Byte length**: 2

**Cycle number**: 2

**Timing**

<table>
<thead>
<tr>
<th>φ</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SYNC</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R/W</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RD</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WR</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDR</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DATA</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDRH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDRL</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**ACCUMULATOR**

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

Byte length: 1
Cycle number: 2

Timing:

![Timing Diagram]

ADDR  
- PC  
- PC +1

DATA  
- Op-code  
- Invalid

ADDRH  
- PCH
- PCH
- PCH

ADDRL /DATA  
- PCL  
- Op-code  
- PCL +1  
- Invalid  
- PCL +1
Instructions: \( \Delta \text{BBC} \overline{A_i}, A, \$\text{shll} \)
\( \Delta \text{BBS} \overline{A_i}, A, \$\text{shll} \)

Byte length: 2

(1) With no branch
Cycle number: 4

Timing:

- \( \phi \)
- \( \text{SYNC} \)
- \( \text{RW} \)
- \( \text{RD} \)
- \( \text{WR} \)

**ADDR**

- PC
- PC+1

**DATA**

- Op-code
- Invalid

**ADDRH**

- PCH
- PCH

**ADDRL / DATA**

- PCL
- PCL+1
- Invalid
- PCL+1
- Invalid
- PCL+1
- Invalid
**ACCUMULATOR BIT RELATIVE**

Instructions:  
\[ \Delta B B C \Delta i, A, $h h l l \]  
\[ \Delta B B S \Delta i, A, $h h l l \]  

Byte length: 2

(2) With branch
Cycle number: 6

Timing:

\[ \phi \]

\[ \text{SYNC} \]

\[ \text{R/W} \]

\[ \text{RD} \]

\[ \text{WR} \]

\[ \text{ADDR} \]

\[ \text{DATA} \]

\[ \text{ADDRH} \]

\[ \text{ADDRL} / \text{DATA} \]

RR: Offset address
*1: \((PC + 1)_L\)
*2: \(((PC + 2) \pm RR)_L\)
ACCUMULATOR BIT

Instructions:
- $\Delta \text{CLB} \Delta_i, A$
- $\Delta \text{SEB} \Delta_i, A$

Byte length: 1
Cycle number: 2

Timing:

- $\phi$
- $\text{SYNC}$
- $\text{R/W}$
- $\text{RD}$
- $\text{WR}$

- $\text{ADDR}$
  - $\text{PC}$
  - $\text{PC+1}$

- $\text{DATA}$
  - $\text{Op-code}$
  - $\text{Invalid}$

- $\text{ADDRH}$
  - $\text{PCH}$

- $\text{ADDRL} / \text{DATA}$
  - $\text{PCL}$
  - $\text{PCL+1}$
  - $\text{Invalid}$
BIT RELATIVE

Instructions: \( \Delta B B C \Delta i, z z, z h l l \)
\( \Delta B B S \Delta i, z z, z h h l l \)

Byte length: 3

(1) With no branch
Cycle number: 5

Timing:

\[ \Phi \]

SYNC

\[ \overline{R/W} \]

RD

WR

ADDR

\[ PC \quad PC+1 \quad ADL,00 \quad PC+2 \]

DATA

\[ \text{Op-code} \quad ADL \quad DATA \quad \text{Invalid} \]

ADDRH

\[ \text{PCH} \quad \text{PCH} \quad 00 \quad \text{PCH} \]

ADDRL /DATA

\[ \text{PCL} \quad \text{Op-code} \quad \text{PCL+1} \quad ADL \quad ADL \quad \text{DATA} \quad \text{PCL+2} \quad \text{Invalid} \quad \text{PCL+2} \quad \text{Invalid} \]
BIT RELATIVE

Instructions: △BBS△i,△z,△z,△hll
△BBS△i,△z,△z,△hll

Byte length: 3

(2) With branch
Cycle number: 7

Timing:

<table>
<thead>
<tr>
<th>φ</th>
<th>SYNC</th>
<th>R/W</th>
<th>RD</th>
<th>WR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDR</th>
<th>DATA</th>
<th>ADDR</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>Op-code</td>
<td>ADL</td>
<td>DATA</td>
</tr>
<tr>
<td>PC+1</td>
<td>ADDL.00</td>
<td>PC+2</td>
<td>(PC+3)H</td>
</tr>
<tr>
<td>PC+2</td>
<td>(PC+3)H</td>
<td>(PC+3)±RR</td>
<td>(PC+3)+RR</td>
</tr>
</tbody>
</table>

RR: Offset address

*1: (PC+3)L
*2: ((PC+3)±RR)L

Invalid

Invalid

Invalid

Invalid

Invalid

Invalid

Invalid

Invalid

Invalid
# ZERO PAGE BIT

**Instructions:**
- \( \Delta \text{CLB} \\Delta i, \$zz \)
- \( \Delta \text{SEB} \\Delta i, \$zz \)

**Byte length:** 2

**Cycle number:** 5

**Timing:**

```
\[ \varphi \]
```

<table>
<thead>
<tr>
<th>ADDR</th>
<th>PC</th>
<th>PC +1</th>
<th>ADL, 00</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>Op-code</td>
<td>ADL</td>
<td>DATA</td>
</tr>
<tr>
<td>ADDRH</td>
<td>PCH</td>
<td>PCH</td>
<td>00</td>
</tr>
<tr>
<td>ADDRL /DATA</td>
<td>PC L</td>
<td>Op-code</td>
<td>PC L +1</td>
</tr>
</tbody>
</table>

---

**Figure:**

- \( \varphi \)
- \( \text{SYNC} \)
- \( \text{R/W} \)
- \( \text{RD} \)
- \( \text{WR} \)
Instructions:
- ΔADC Δ$zz$ (T=0)
- ΔAND Δ$zz$ (T=0)
- ΔBIT Δ$zz$
- ΔCMP Δ$zz$ (T=0)
- ΔCPX Δ$zz$
- ΔCPY Δ$zz$
- ΔEOR Δ$zz$ (T=0)
- ΔLDA Δ$zz$ (T=0)
- ΔLDX Δ$zz$
- ΔLDY Δ$zz$
- ΔORA Δ$zz$ (T=0)
- ΔSBC Δ$zz$ (T=0)
- ΔTST Δ$zz$

Byte length: 2
Cycle number: 3

Timing:

\[
\begin{array}{c|c|c|c|c|c}
\phi & | & | & | & | \\
SYNC & | & | & | & | \\
R/W & | & | & | & | \\
RD & | & | & | & | \\
WR & | & | & | & |
\end{array}
\]

 ADDR  
| PC | PC+1 | ADL.00 |

 DATA  
| Op-code | ADL | DATA |

 ADDRH  
| PCH | PCH | 00 |

 ADDRL /DATA  
| PCL | Op-code | PCL+1 | ADL | ADL | DATA |
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:

- φ
- SYNC
- R/W
- RD
- WR

ADDR:
- PC
- PC+1
- ADL.00

DATA:
- Op-code
- ADL
- DATA
- Invalid
- NEW DATA

ADDRH:
- PCH
- PCH
- 00

ADDRL/DATA:
- PCL
- Op-code
- PCL+1
- ADL
- ADL
- ADL
- ADL
- NEW DATA
Instruction: $\Delta$RRF$\Delta$szz
Byte length: 2
Cycle number: 8
Timing:
Instruction: \( \Delta \text{LDM}\#nn,\$zz \)
Byte length: 3
Cycle number: 4

Timing:

\[ \phi \]

\[ \text{SYNC} \]

\[ \text{R/W} \]

\[ \text{RD} \]

\[ \text{WR} \]

\[ \text{ADDR} \]

\[ \text{DATA} \]

\[ \text{ADDRH} \]

\[ \text{ADDRL} / \text{DATA} \]
## Zero Page X

**Instruction**: △MUL$z\$,X  (Note)
**Byte length**: 2
**Cycle number**: 15

**Timing**:

<table>
<thead>
<tr>
<th>φ</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNC</td>
<td></td>
</tr>
<tr>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>RD</td>
<td></td>
</tr>
<tr>
<td>WR</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDR</th>
<th>PC</th>
<th>PC+1</th>
<th>ADL+X,00</th>
<th>S,SPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>Opcode</td>
<td>ADL</td>
<td>DATA</td>
<td>Invalid</td>
</tr>
</tbody>
</table>

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

Note: This instruction cannot be used for any products.
Zero Page X

Instruction : \( \Delta \text{DIV}\Delta \$zz, X \) (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 any 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:
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$zz,X (T=0)
- ΔLDX Δ$zz,Y
- ΔLDY Δ$zz,X
- ΔORA Δ$zz,X (T=0)
- ΔSBC Δ$zz,X (T=0)

Byte length: 2
Cycle number: 4

Timing:

<table>
<thead>
<tr>
<th>φ</th>
<th>SYNC</th>
<th>R/W</th>
<th>RD</th>
<th>WR</th>
</tr>
</thead>
</table>

Address and Data Registers:

- ADDR: PC, PC+1, (PC+1) L, AD. + X (or Y)
- DATA: Op-code, ADL, Invalid, DATA
- ADDRH: PCH, PCH, 00
- ADDRL / DATA: PCL, Op-code, PCL+1, ADL, (PC+1) L, AD. + X (or Y), DATA
Instructions:

- ΔSTAΔ$zz,X
- ΔSTXΔ$zz,Y
- ΔSTYΔ$zz,X

Byte length: 2
Cycle number: 5

Timing:
Instructions:

- $\triangle$ADC $\triangle$Shhll (T=0)
- $\triangle$AND $\triangle$Shhll (T=0)
- $\triangle$BIT $\triangle$Shhll
- $\triangle$CMP $\triangle$Shhll (T=0)
- $\triangle$CPX $\triangle$Shhll
- $\triangle$CPY $\triangle$Shhll
- $\triangle$EOR $\triangle$Shhll (T=0)
- $\triangle$LDA $\triangle$Shhll (T=0)
- $\triangle$LDX $\triangle$Shhll
- $\triangle$LDY $\triangle$Shhll
- $\triangle$ORA $\triangle$Shhll (T=0)
- $\triangle$SBC $\triangle$Shhll (T=0)

Byte length: 3
Cycle number: 4

Timing:

\[
\phi \quad \text{SYNC} \quad \text{R/W} \quad \text{RD} \quad \text{WR}
\]

\[
\text{ADDR} \quad \text{DATA} \quad \text{ADDRH} \quad \text{ADDRL/DATA}
\]

- PC
- PC+1
- PC+2
- ADL
- ADH

- Op-code
- ADL
- ADH
- DATA

- PCH
- PCH
- PCH
- ADH

- PCL
- PCL+1
- PCL+2
- ADL
- ADL
- DATA
Instructions: \( \Delta ASL \ \Delta SHHLL \)
- \( \Delta DEC \ \Delta SHHLL \)
- \( \Delta INC \ \Delta SHHLL \)
- \( \Delta LSR \ \Delta SHHLL \)
- \( \Delta ROL \ \Delta SHHLL \)
- \( \Delta ROR \ \Delta SHHLL \)

Byte length: 3
Cycle number: 6

Timing:

\[ \phi \]
\[ \text{SYNC} \]
\[ \text{R/W} \]
\[ \text{RD} \]
\[ \text{WR} \]

ADDR:
- \( \text{PC} \)
- \( \text{PC} + 1 \)
- \( \text{PC} + 2 \)
- \( \text{ADL}, \text{ADH} \)

DATA:
- Op-code
- \( \text{ADL} \)
- \( \text{ADH} \)
- \( \text{DATA} \)
- Invalid
- NEW DATA

ADDRH:
- \( \text{PCH} \)
- \( \text{PCH} \)
- \( \text{PCH} \)
- \( \text{ADH} \)

ADDRL/DATA:
- \( \text{PCL} \)
- \( \text{Op-code} \)
- \( \text{PCL} + 1 \)
- \( \text{ADL} \)
- \( \text{PCL} + 2 \)
- \( \text{ADH} \)
- \( \text{ADL} \)
- \( \text{DATA} \)
- \( \text{ADL} \)
- \( \text{NEW DATA} \)
**ABSOLUTE**

Instruction: ΔJMP Δ$hll
Byte length: 3
Cycle number: 3

Timing:

- φ
- SYNC
- R/W
- RD
- WR

**ADDR**
- PC
- PC+1
- PC+2
- PCL, PCH

**DATA**
- Opcode
- PCL
- PCH

**ADDRH**
- PCH
- PCH
- PCH
- PCH

**ADDRL /DATA**
- PCL
- Opcode
- PCL+1
- PCL+2
- PCH
- PCL
**ABSOLUTE**

Instruction: JSR $hhll

Byte length: 3

Cycle number: 6

**Timing:**

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

Instructions:
- ΔSTA Δshll
- ΔSTX Δshll
- ΔSTY Δshll

Byte length: 3
Cycle number: 5

Timing:
ABSOLUTE X, ABSOLUTE Y

Instructions:
- ΔADC Δ$hhll,X or Y (T=0)
- ΔAND Δ$hhll,X or Y (T=0)
- ΔCMP Δ$hhll,X or Y (T=0)
- ΔEOR Δ$hhll,X or Y (T=0)
- ΔLDA Δ$hhll,X or Y (T=0)
- ΔDX Δ$hhll,Y
- ΔDHY Δ$hhll,X
- ΔORA Δ$hhll,X or Y (T=0)
- ΔSBC Δ$hhll,X or Y (T=0)

Byte length: 3
Cycle number: 5

Timing:
- φ
- SYNC
- R/W
- RD
- WR

ADDR
- PC
- PC+1
- PC+2
- ADL+X or Y
- ADH+X or Y

DATA
- Op-code
- ADL
- ADH
- Invalid
- DATA

ADDRH
- PCH
- PCH
- PCH
- ADH
- ADH+C

ADDRL
- PCL
- PCL+1
- PCL+2
- ADL
- ADL+X or Y
- ADL+X or Y
- ADL+X or Y
- DATA

C: Carry of ADL+X or Y
ABSOLUTE X

Instructions:
- ΔASL Δ$h_{hh},X$
- ΔDEC Δ$h_{hh},X$
- ΔINC Δ$h_{hh},X$
- ΔLSR Δ$h_{hh},X$
- ΔROL Δ$h_{hh},X$
- ΔROR Δ$h_{hh},X$

Byte length: 3
Cycle number: 7

Timing:

```
SYNC
R/W
RD
WR
ADDR
DATA
ADDRH
ADDRL/DATA
```

C: Carry of ADL+X
ABSOLUTE X, ABSOLUTE Y

Instruction: \( \text{STA} \text{SHH}, X \) or \( Y \)
Byte length: 3
Cycle number: 6
Timing:

\[
\begin{align*}
\phi & \quad & \\
\text{SYNC} & \quad & \\
\text{R/W} & \quad & \\
\text{RD} & \quad & \\
\text{WR} & \quad & \\
\text{ADDR} & \quad & \begin{array}{c}
\text{PC} \\
\text{PC+1} \\
\text{PC+2} \\
\text{ADL+X (or Y)} \text{ or } \text{ADH} \\
\text{ADL+X (or Y)} \text{ or } \text{ADH+C} \\
\end{array} \\
\text{DATA} & \quad & \begin{array}{c}
\text{Op-code} \\
\text{ADL} \\
\text{ADH} \\
\text{Invalid} \\
\text{Invalid} \\
\text{DATA} \\
\end{array} \\
\text{ADDRh} & \quad & \begin{array}{c}
PCH \\
PCH \\
PCH \\
\text{ADH} \\
\text{ADH+C} \\
\end{array} \\
\text{ADDRL/DATA} & \quad & \begin{array}{c}
PCL \\
PCL+1 \\
\text{ADL} \\
PCL+2 \\
\text{ADH} \\
\text{ADL+X (or Y)} \\
\text{ADL+X (or Y)} \\
\text{DATA} \\
\end{array} \\
\end{align*}
\]

C: Carry of ADL+X or Y
**INDIRECT**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>( \text{\textasciitilde JMP \textasciitilde} (\text{\textsf{shl}}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte length</td>
<td>3</td>
</tr>
<tr>
<td>Cycle number</td>
<td>5</td>
</tr>
</tbody>
</table>

**Timing**

<table>
<thead>
<tr>
<th>( \phi )</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNC</td>
<td><img src="image" alt="SYNC Waveform" /></td>
</tr>
<tr>
<td>( \overline{R/W} )</td>
<td><img src="image" alt="Read/Write Waveform" /></td>
</tr>
<tr>
<td>( \overline{RD} )</td>
<td><img src="image" alt="Read Waveform" /></td>
</tr>
<tr>
<td>( \overline{WR} )</td>
<td><img src="image" alt="Write Waveform" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDR</th>
<th>PC</th>
<th>PC+1</th>
<th>PC+2</th>
<th>BA</th>
<th>BA+1</th>
<th>ADL</th>
<th>ADH</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>Op-code</td>
<td>BAL</td>
<td>BAH</td>
<td>ADL</td>
<td>ADH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDRH</td>
<td>PC</td>
<td>PC+1</td>
<td>PC+2</td>
<td>BA</td>
<td>BA+1</td>
<td>ADL</td>
<td>ADH</td>
</tr>
<tr>
<td>ADDRL /DATA</td>
<td>PCL</td>
<td>PCL+1</td>
<td>PCL+2</td>
<td>BAH</td>
<td>BAL</td>
<td>ADL</td>
<td>ADL+1</td>
</tr>
</tbody>
</table>

BA : Basic address
Instruction : \( \Delta \text{IMP} \Delta (\$zz) \)
Byte length : 2
Cycle number : 4

Timing :

\[
\phi \\
\text{SYNC} \\
\text{R/W} \\
\text{RD} \\
\text{WR}
\]

ADDR

\[
\begin{array}{cccc}
\text{PC} & \text{PC+1} & \text{BAL},00 & \text{BAL+1},00 \\
\text{ADDR} & \text{ADDR H} & \text{ADDR L} & \text{DATA} & \text{DATA H} & \text{DATA L}
\end{array}
\]

ADDRH

\[
\begin{array}{cccc}
PCH & PCH & 00 & \text{ADH}
\end{array}
\]

ADDRL /DATA

\[
\begin{array}{cccc}
PCL & PCL+1 & \text{BAL} & \text{BAL+1}
\end{array}
\]

BA : Basic address
ZER0 PAGE INDICRFT

Instruction: ΔJSRΔ($zz)
Byte length: 2
Cycle number: 7

Timing:

φ

SYNC

R/W

RD

WR

ADDR

DATA

ADDRH

ADDRL /DATA

BA: Basic address

Note: Some kinds types are “01” or content of SPS flag.
INDIRECT X

Instructions:
- \( \Delta \text{ADC} \Delta (\$zz,X) \) (T=0)
- \( \Delta \text{AND} \Delta (\$zz,X) \) (T=0)
- \( \Delta \text{CMP} \Delta (\$zz,X) \) (T=0)
- \( \Delta \text{EOR} \Delta (\$zz,X) \) (T=0)
- \( \Delta \text{LDA} \Delta (\$zz,X) \) (T=0)
- \( \Delta \text{ORA} \Delta (\$zz,X) \) (T=0)
- \( \Delta \text{SBC} \Delta (\$zz,X) \) (T=0)

Byte length: 2
Cycle number: 6

Timing:

\( \phi \)

SYNC

R/W

RD

WR

ADDR

DATA

ADDRH

ADDRL /DATA

BA : Basic address
INDIRECT X

Instruction: ΔSTAΔ($zz,X)
Byte length: 2
Cycle number: 7

Timing:

1. φ
2. SYNC
3. R/W
4. RD
5. WR

LAB: Basic address
INDIRECT Y

Instructions:

- ΔADC Δ($zz),Y (T=0)
- ΔAND Δ($zz),Y (T=0)
- ΔCMP Δ($zz),Y (T=0)
- ΔEOR Δ($zz),Y (T=0)
- ΔLDA Δ($zz),Y (T=0)
- ΔORA Δ($zz),Y (T=0)
- ΔSBC Δ($zz),Y (T=0)

Byte length: 2
Cycle number: 6

Timing:

\[ \phi \]

\[ \text{SYNC} \]

\[ \text{R/W} \]

\[ \text{RD} \]

\[ \text{WR} \]

\[ \text{ADDR} \]

\[ \text{DATA} \]

\[ \text{ADDRh} \]

\[ \text{ADDRL / DATA} \]

BA: Basic address
C: Carry of ADL+Y
INDIRECT Y

Instruction: \( \triangle STAD\$(zz),Y \)
Byte length: 2
Cycle number: 7

Timing:

\[
\begin{align*}
\phi & \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \q
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:

- φ
- SYNC
- R/W
- RD
- WR

ADDR

- PC
- PC+1

DATA

- Op-code
- Invalid

ADDRH

- PCH
- PCH

ADDRL

/ DATA

- PCL
- Op-code
- PCL+1
- Invalid
RELATIVE

Instructions:
- ΔBCC ΔShhll
- ΔBCS ΔShhll
- ΔBEQ ΔShhll
- ΔBMI ΔShhll
- ΔBNE ΔShhll
- ΔBPL ΔShhll
- ΔBVC ΔShhll
- ΔBVS ΔShhll

Byte length: 2

(2) With branch
Cycle number: 4

Timing:

<table>
<thead>
<tr>
<th>PC</th>
<th>PC+1</th>
<th>(PC+2) ± RR</th>
<th>(PC+2) ± RR</th>
<th>(PC+2) ± RR</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDR</td>
<td>Op-code</td>
<td>± RR</td>
<td>Invalid</td>
<td>Invalid</td>
</tr>
<tr>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDRH</td>
<td>PCH</td>
<td>PCH</td>
<td>(PC+1)H</td>
<td>(PC+2)H</td>
</tr>
<tr>
<td>ADDRL/DATA</td>
<td>PCH</td>
<td>Op-code</td>
<td>PCH+1</td>
<td>± RR</td>
</tr>
</tbody>
</table>

RR: Offset value
Instruction: ΔBRA Δshll
Byte length: 2
Cycle number: 4
Timing:

RR: Offset value
### SPECIAL PAGE

**Instruction**: \(\triangle J SR \$ h h l\)

**Byte length**: 2

**Cycle number**: 5

**Timing**:

- \(\phi\)
- \(SYNC\)
- \(R/W\)
- \(RD\)
- \(WR\)

**ADDR**

- PC
- PC + 1
- S,00 (Note)
- S-1,00 (Note)
- BAL, FF

**DATA**

- Op-code
- BAL
- Invalid
- (PC + 1)H
- (PC + 1) L

**ADDRh**

- PCH
- 00 (Note)
- FF

**ADDRL/DATA**

- PCL
- Op-code
- PCL + 1
- BAL
- S
- (PC + 1) W
- S-1
- (PC + 1) L
- BAL

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

**BA**: Basic address
**Immediate**

Instructions:
- $\Delta ADC \#nn$ (T=1)
- $\Delta AND \#nn$ (T=1)
- $\Delta EOR \#nn$ (T=1)
- $\Delta ORA \#nn$ (T=1)
- $\Delta SBC \#nn$ (T=1)

Byte length: 2
Cycle number: 5

Timing:

- $\phi$
- $SYNC$
- $R/W$
- $RD$
- $WR$

 ADDR

- $PC$
- $PC+1$
- $X,00$

 DATA

- Op-code
- $DATA_1$
- $DATA_2$
- Invalid
- NEW DATA

 ADDRx

- $PCH$
- $PCH$
- $00$

 ADDRL / DATA

- $PCL$
- $PCL+1$
- $DATA_1$
- $DATA_2$
- $X$
- $X$
- $X$
- NEW DATA

152
IMMEDIATE

Instruction: \( \Delta \text{CMP} \Delta \#nn \quad (T=1) \)

Byte length: 2

Cycle number: 3

Timing:

\[ \phi \quad \text{SYNC} \quad \text{R/W} \quad \text{RD} \quad \text{WR} \]

ADDR: \( PC \quad PC+1 \quad X,00 \)

DATA: Op-code \( DATA_1 \quad DATA_2 \)

ADDRH: \( PCH \quad PCH \quad 00 \)

ADDRL /DATA: \( PCL \quad \text{Op-code} \quad PCL+1 \quad DATA_1 \quad X \quad DATA_2 \)
## Immediate

**Instruction**: \( \Delta \text{LDA} \Delta \#nn \) (T=1)

**Byte length**: 2

**Cycle number**: 4

### Timing

<table>
<thead>
<tr>
<th>Event</th>
<th>Timing Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi )</td>
<td><img src="image1" alt="Timing Diagram" /></td>
</tr>
<tr>
<td><strong>SYNC</strong></td>
<td><img src="image2" alt="Timing Diagram" /></td>
</tr>
<tr>
<td><strong>R/W</strong></td>
<td><img src="image3" alt="Timing Diagram" /></td>
</tr>
<tr>
<td><strong>RD</strong></td>
<td><img src="image4" alt="Timing Diagram" /></td>
</tr>
<tr>
<td><strong>WR</strong></td>
<td><img src="image5" alt="Timing Diagram" /></td>
</tr>
</tbody>
</table>

### ADDR

<table>
<thead>
<tr>
<th>ADDR</th>
<th>PC</th>
<th>PC+1</th>
<th>X,00</th>
</tr>
</thead>
</table>

### DATA

<table>
<thead>
<tr>
<th>DATA</th>
<th>Opcode</th>
<th>DATA</th>
<th>Invalid</th>
<th>DATA</th>
</tr>
</thead>
</table>

### ADDRH

<table>
<thead>
<tr>
<th>ADDRH</th>
<th>PCH</th>
<th>PCH</th>
<th>00</th>
</tr>
</thead>
</table>

### ADDRL /DATA

<table>
<thead>
<tr>
<th>ADDRL /DATA</th>
<th>PCL</th>
<th>Opcode</th>
<th>PC+1</th>
<th>DATA</th>
<th>X</th>
<th>X</th>
<th>DATA</th>
</tr>
</thead>
</table>
### [T=1] ZERO PAGE

<table>
<thead>
<tr>
<th>Instructions</th>
<th>ΔADCΔ$zz (T=1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ΔANDΔ$zz (T=1)</td>
</tr>
<tr>
<td></td>
<td>ΔEORΔ$zz (T=1)</td>
</tr>
<tr>
<td></td>
<td>ΔORAΔ$zz (T=1)</td>
</tr>
<tr>
<td></td>
<td>ΔSBCΔ$zz (T=1)</td>
</tr>
<tr>
<td>Byte length</td>
<td>2</td>
</tr>
<tr>
<td>Cycle number</td>
<td>6</td>
</tr>
<tr>
<td>Timing</td>
<td>:</td>
</tr>
</tbody>
</table>

#### Timing Diagram

![Timing Diagram](image-url)

- **ADDR**
  - PC
  - PC+1
  - ADL,00
  - X,00

- **DATA**
  - Op-code
  - ADL
  - DATA 1
  - DATA 2
  - Invalid
  - NEW DATA

- **ADD Rh**
  - PCH
  - 00

- **ADD R L /DATA**
  - PC
  - Op-code
  - PC+1
  - ADL
  - DATA 1
  - X
  - DATA 2
  - NEW DATA

155
**Instruction:** ΔCMP Δ$zz$ (T=1)  
**Byte length:** 2  
**Cycle number:** 4  

**Timing:**

<table>
<thead>
<tr>
<th>PC</th>
<th>PC+1</th>
<th>ADL,00</th>
<th>X,00</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCH</td>
<td>PCH</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>PCL</td>
<td>PCL+1</td>
<td>ADL</td>
<td>ADL</td>
</tr>
</tbody>
</table>

---

**ZERO PAGE**
Instruction: `LDA $zz` (T=1)
Byte length: 2
Cycle number: 5

Timing:

- φ
- Sync
- R/W
- RD
- WR

ADDRESS:
- PC
- PC+1
- ADL,00
- X,00

DATA:
- Op-code
- ADL
- DATA
- Invalid
- DATA

ADDRESS:
- PCH
- PCH
- 00

ADDRESS:
- PCL
- Op-code
- PCL+1
- ADL
- ADL
- DATA
- X
- X
- DATA
### ZERO PAGE 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** : 7

**Timing** :

<table>
<thead>
<tr>
<th>0</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sync</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDR</td>
<td>PC</td>
<td>PC +1</td>
<td>(PC+1) L</td>
<td>ADL, X</td>
<td>X, 00</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>Op-code</td>
<td>ADL</td>
<td>Invalid</td>
<td>DATA 1</td>
<td>DATA 2</td>
<td>Invalid</td>
</tr>
<tr>
<td>ADDRH</td>
<td>PCH</td>
<td>PCH</td>
<td>00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDRL/DATA</td>
<td>PCL</td>
<td>Op-code</td>
<td>PC+1</td>
<td>ADL</td>
<td>PC+1</td>
<td>ADL+X</td>
</tr>
</tbody>
</table>
ZERO PAGE X

Instruction: ΔCMPΔ$zz,X (T=1)

Byte length: 2
Cycle number: 5

Timing:

<table>
<thead>
<tr>
<th>φ</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SYNC</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R/W</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RD</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WR</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDR</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DATA</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDRH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDRL /DATA</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Op-code Invalid
ZERO PAGE X

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

Timing:

- **SYNC**
- **R/W**
- **RD**
- **WR**

**ADDR**
- PC
- PC +1
- (PC +1) L .00
- ADL +X .00
- X,00

**DATA**
- Opcode
- ADL
- Invalid
- DATA
- Invalid
- DATA

**ADDRH**
- PCH
- PCH
- 00

**ADDRL /DATA**
- PCL
- Op-code
- PCL +1
- ADL
- (PC +1) L
- ADL +X
- DATA
- X
- X
- DATA
Instructions:

- △ADC ΔShll (T=1)
- △AND ΔShll (T=1)
- △EOR ΔShll (T=1)
- △ORA ΔShll (T=1)
- △SBC ΔShll (T=1)

Byte length: 3
Cycle number: 7

Timing:

Δ ΔΔ $hhll (T=1)
Δ ΔΔ $hhll (T=1)
Δ ΔΔ $hhll (T=1)
Δ ΔΔ $hhll (T=1)
Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)

Δ ΔΔ $hhll (T=1)
**ABSOLUTE**

Instruction: \[\Delta\text{CMP} \Delta\text{shll} \quad (T=1)\]

Byte length: 3
Cycle number: 5

Timing:

- \(\phi\)
- SYNC
- R/W
- RD
- WR

<table>
<thead>
<tr>
<th>ADDR</th>
<th>PC</th>
<th>PC+1</th>
<th>PC+2</th>
<th>ADL</th>
<th>ADH</th>
<th>X,00</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>Op-code</td>
<td>ADL</td>
<td>ADH</td>
<td>DATA 1</td>
<td>DATA 2</td>
<td></td>
</tr>
<tr>
<td>ADDRH</td>
<td>PCH</td>
<td>PCH</td>
<td>PCH</td>
<td>ADH</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>ADDRL /DATA</td>
<td>PCL Op-code</td>
<td>PCL+1</td>
<td>ADL</td>
<td>PCL+2 ADH</td>
<td>ADL DATA 1</td>
<td>X DATA 2</td>
</tr>
</tbody>
</table>
Instruction: \( \text{LDA} \text{\$hhll} \) (T=1)
Byte length: 3
Cycle number: 6

Timing:

\[ \phi \]

\[ \text{SYNC} \]

\[ \text{R/W} \]

\[ \text{RD} \]

\[ \text{WR} \]

\[ \text{ADDR} \]

\[ \text{DATA} \]

\[ \text{ADDRH} \]

\[ \text{ADDRL/DATA} \]
Instructions:

\[
\begin{align*}
\text{ΔADC} & \text{Δshll,X or Y} \quad (T=1) \\
\text{ΔAND} & \text{Δshll,X or Y} \quad (T=1) \\
\text{ΔEOR} & \text{Δshll,X or Y} \quad (T=1) \\
\text{ΔORA} & \text{Δshll,X or Y} \quad (T=1) \\
\text{ΔSBC} & \text{Δshll,X or Y} \quad (T=1)
\end{align*}
\]

Byte length: 3
Cycle number: 8
Timing:

\[\text{SYNC}\]
\[\text{R/W}\]
\[\text{RD}\]
\[\text{WR}\]

\[\text{ADDR}\]
\[\text{DATA}\]
\[\text{ADDRH}\]
\[\text{ADDR/L /DATA}\]

C : Carry of ADL+X or Y
Instruction: ΔCMPΔ$hll,X or Y (T=1)
Byte length: 3
Cycle number: 6

Timing:

\[ \phi \]

\[ \text{SYNC} \]

\[ \text{R/W} \]

\[ \text{RD} \]

\[ \text{WR} \]

\[ \text{ADDR} \]

\[ \text{DATA} \]

\[ \text{ADDRL/DATA} \]

C: Carry of ADL+X or Y
ABSOLUTE X, ABSOLUTE Y

Instruction: \( LDA \$hhll, X \) or \( Y \) (T=1)

Byte length: 3

Cycle number: 7

Timing:

\[
\begin{array}{c}
\phi \\
SYNC \\
R/W \\
RD \\
WR \\
ADDR \\
DATA \\
ADDRh \\
ADDRL/DATA \\
\end{array}
\]

C : Carry of ADL+X or Y
IN DIRECT X

Instructions:

- $\Delta\text{ADC}(\$zz,X)$ (T=1)
- $\Delta\text{AND}(\$zz,X)$ (T=1)
- $\Delta\text{EOR}(\$zz,X)$ (T=1)
- $\Delta\text{ORA}(\$zz,X)$ (T=1)
- $\Delta\text{SBC}(\$zz,X)$ (T=1)

Byte length: 2
Cycle number: 9
Timing:

Sync

R/W
RD
WR

ADDR

DATA

ADDRH

ADDRL

DATA

BA : Basic address
INDIRECT X

Instruction : \( \triangle \text{CMP} \triangle (\$zz, X) \) (T=1)

Byte length : 2
Cycle number : 7

Timing:

\[ \begin{array}{|c|}
\hline
\phi \ \\
\hline
\text{SYNC} \ \\
\hline
\text{R/W} \ \\
\hline
\text{RD} \ \\
\hline
\text{WR} \ \\
\hline
\end{array} \]

ADDR

\[ \begin{array}{|c|c|c|c|c|c|}
\hline
\text{PC} & \text{PC} + 1 & (\text{PC} + 1)_L & \text{BAL} + X & \text{BAL} + X + 1 & \text{ADL} & \text{ADH} & X,00 \\
\hline
\end{array} \]

DATA

\[ \begin{array}{|c|c|c|c|c|c|c|}
\hline
\text{Op-code} & \text{BAL} & \text{Invalid} & \text{ADL} & \text{ADH} & \text{DATA 1} & \text{DATA 2} \\
\hline
\end{array} \]

ADDRH

\[ \begin{array}{|c|c|c|c|}
\hline
\text{PCH} & \text{PCH} & 00 & \text{ADH} & 00 \\
\hline
\end{array} \]

ADDRL/DATA

\[ \begin{array}{|c|c|c|c|c|c|c|c|}
\hline
\text{PCL} & \text{Op-code} & \text{PCL} + 1 & \text{BAL} & (\text{PC} + 1)_L & \text{BAL} + X & \text{BAL} + X + 1 & \text{ADL} & \text{ADL} & \text{ADL} & \text{ADL} & \text{DATA 1} & \text{X} & \text{DATA 2} \\
\hline
\end{array} \]

BA : Basic address
**INDIRECT X**

Instruction: \( \Delta \text{LDA} \Delta (\#zz, X) \)  \((T=1)\)

Byte length: 2

Cycle number: 8

Timing:

<table>
<thead>
<tr>
<th>Time</th>
<th>φ</th>
<th>SYNC</th>
<th>R/W</th>
<th>RD</th>
<th>WR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ADDR**

- PC
- PC + 1
- \( (PC+1)_L \)
- BAL \( +X \)
- ADL \( +X +1 \)
- ADH
- X, 00

**DATA**

- Opcode
- BAL
- Invalid
- ADL
- ADH
- DATA
- Invalid
- DATA

**ADDRH**

- PCH
- PCH
- 00
- ADH
- 00

**ADDRL**

- PCL
- PCL + 1
- BAL
- \( (PC+1)_H \)
- ADL
- ADH
- X
- X

**DATA**

- PCL
- PCL + 1
- BAL
- \( (PC+1)_H \)
- ADL
- ADH
- DATA
- X

BA : Basic address
INDIRECT Y

Instructions:

- \( \triangleADC \Delta($zz),Y \) (T=1)
- \( \triangleAND \Delta($zz),Y \) (T=1)
- \( \triangleEOR \Delta($zz),Y \) (T=1)
- \( \triangleORA \Delta($zz),Y \) (T=1)
- \( \triangleSBC \Delta($zz),Y \) (T=1)

Byte length: 2
Cycle number: 9

Timing:

- Op-code Invalid
- NEW DATA
- C: Carry of ADL+Y
- BA: Basic address
**INDIRECT Y**

**Instruction:** \( \Delta \text{CMP}(\$zz),Y \) (T=1)

**Byte length:** 2

**Cycle number:** 7

**Timing:**

![Timing Diagram]

**ADDR**
- PC
- PC +1
- BAL
- BAL +1
- ADL + Y
- ADL + Y
- ADH
- ADL + C
- X,00

**DATA**
- Op-code
- BAL
- ADL
- ADH
- Invalid
- DATA 1
- DATA 2

**ADDRH**
- PCH
- PCH
- 00
- ADH
- ADL + C
- 00

**ADDRL / DATA**
- PCL
- PCL +1
- BAL
- BAL +1
- ADL
- ADL + Y
- ADL + Y
- ADL +1
- ADL + C
- ADL + Y
- DATA 1
- DATA 2

**BA:** Basic address

**C:** Carry of ADL + Y

---

Insertions of new content for better understanding:

- \( \Delta \text{CMP}(\$zz),Y \) (T=1)
- Byte length: 2
- Cycle number: 7
- Timing:
  - φ
  - SYNC
  - R/W
  - RD
  - WR

Diagram showing the timing with various stages labeled accordingly.

---

**Sync:**

**R/W:**

**RD:**

**WR:**

---

171
**INDIRECT Y**

Instruction : \( \Delta \text{LDA} \Delta(\text{\$zz}),Y \) \( (T=1) \)

Byte length : 2

Cycle number : 8

Timing :

- **\( \phi \)**
- **SYNC**
- **R/W**
- **RD**
- **WR**

---

**ADDR**

- \( \text{PC} \)
- \( \text{PC}+1 \)
- \( \text{BAL} \) \( .00 \)
- \( \text{BAL}+1 \) \( .00 \)
- \( \text{ADL}+Y \)
- \( \text{ADH} \)
- \( \text{ADL}+Y \) \( \text{ADH}+C \)
- \( X,00 \)

**DATA**

- **Op-code**
- \( \text{BAL} \)
- \( \text{ADL} \)
- \( \text{ADH} \)
- \( \text{Invalid} \)
- \( \text{DATA} \)
- \( \text{Invalid} \)
- \( \text{DATA} \)

**ADDRH**

- \( \text{PCH} \)
- \( \text{PCH} \)
- \( .00 \)
- \( \text{ADH} \)
- \( \text{ADH}+C \)
- \( .00 \)

**ADDRL/DATA**

- \( \text{PC} \) \( \text{Op-code} \)
- \( \text{PC}+1 \) \( \text{Op-code} \)
- \( \text{BAL} \)
- \( \text{BAL}+1 \)
- \( \text{ADL} \)
- \( \text{ADL}+Y \)
- \( \text{ADL}+Y \) \( \text{DATA} \)
- \( X \)
- \( X \)
- \( \text{DATA} \)

**Notes:**

- BA : Basic address
- C : C any of ADL+Y
### APPENDIX 2

#### 740 Family Machine Language Instruction Table

<table>
<thead>
<tr>
<th>Parameter Classification</th>
<th>SYMBOL</th>
<th>FUNCTION</th>
<th>FLAG</th>
<th>INSTRUCTION CODE</th>
<th>BYTE NUMBER</th>
<th>CYCLE NUMBER</th>
<th>NOTE</th>
</tr>
</thead>
</table>
|                          | LDA $ nn    | (A)→nn where M=(zz)                                                     | $ \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \time...
<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>FUNCTION</th>
<th>FLAG</th>
<th>INSTRUCTION CODE</th>
<th>BYTE</th>
<th>CYCLE</th>
<th>NUMBER</th>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC # $ nn</td>
<td>(A)←(A)+nn+(C)</td>
<td>0 1 1 0 1 0 0 1</td>
<td>69</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ADC $ zz</td>
<td>(A)←(A)+(M)+(C) where M=(zz)</td>
<td>0 1 1 0 0 1 0 1</td>
<td>65</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ADC $ zz, X</td>
<td>(A)←(A)+(M)+(C) where M=(zz+(X))</td>
<td>0 1 1 1 0 1 0 1</td>
<td>75</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ADC $ hhII</td>
<td>(A)←(A)+(M)+(C) where M=(hhII)</td>
<td>0 1 1 0 1 1 0 1</td>
<td>6D</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ADC $ hhII, X</td>
<td>(A)←(A)+(M)+(C) where M=(hhII+(X))</td>
<td>0 1 1 1 1 1 1 0 1</td>
<td>7D</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ADC $ hhII, Y</td>
<td>(A)←(A)+(M)+(C) where M=(hhII+(Y))</td>
<td>0 1 1 1 0 0 1 0 1</td>
<td>79</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ADC ($ zz, X)</td>
<td>(A)←(A)+(M)+(C) where M=((zz+(X)+1)((zz+(X)))</td>
<td>0 1 1 0 0 0 1 0 1</td>
<td>61</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ADC ($ zz, Y)</td>
<td>(A)←(A)+(M)+(C) where M=((zz+1)(zz)+(Y))</td>
<td>0 1 1 1 0 1 0 1 0 1</td>
<td>71</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SBC # $ nn</td>
<td>(A)←(A)-(nn)-(C)</td>
<td>0 1 1 0 1 0 1</td>
<td>69</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SBC $ zz</td>
<td>(A)←(A)-(M)-(C) where M=(zz)</td>
<td>0 1 1 0 0 1 0 1</td>
<td>65</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SBC $ zz, X</td>
<td>(A)←(A)-(M)-(C) where M=(zz+(X))</td>
<td>0 1 1 1 0 1 0 1</td>
<td>75</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SBC $ hhII</td>
<td>(A)←(A)-(M)-(C) where M=(hhII)</td>
<td>0 1 1 0 1 1 0 1</td>
<td>6D</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SBC $ hhII, X</td>
<td>(A)←(A)-(M)-(C) where M=(hhII+(X))</td>
<td>0 1 1 1 1 1 1 0 1</td>
<td>7D</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SBC $ hhII, Y</td>
<td>(A)←(A)-(M)-(C) where M=(hhII+(Y))</td>
<td>0 1 1 1 0 0 1 0 1</td>
<td>79</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SBC ($ zz, X)</td>
<td>(A)←(A)-(M)-(C) where M=((zz+(X)+1)((zz+(X)))</td>
<td>0 1 1 0 0 0 1 0 1</td>
<td>61</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SBC ($ zz, Y)</td>
<td>(A)←(A)-(M)-(C) where M=((zz+1)(zz)+(Y))</td>
<td>0 1 1 1 0 1 0 1 0 1</td>
<td>71</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INC A</td>
<td>(A)←(A)+1</td>
<td>0 0 1 0 1 0</td>
<td>3A</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC $ zz</td>
<td>(M)←(M)+1 where M=(zz)</td>
<td>0 1 1 0 0 1 0 1</td>
<td>65</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC $ zz, X</td>
<td>(M)←(M)+1 where M=(zz+(X))</td>
<td>0 1 1 1 0 1 0 1</td>
<td>75</td>
<td>2</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC $ hhII</td>
<td>(M)←(M)+1 where M=(hhII)</td>
<td>0 1 1 0 1 1 0 1</td>
<td>6D</td>
<td>3</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC $ hhII, X</td>
<td>(M)←(M)+1 where M=(hhII+(X))</td>
<td>0 1 1 1 1 1 1 0 1</td>
<td>7D</td>
<td>3</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC A</td>
<td>(A)←(A)-1</td>
<td>0 0 1 0 1 0</td>
<td>3A</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC $ zz</td>
<td>(M)←(M)-1 where M=(zz)</td>
<td>0 1 1 0 0 1 0 1</td>
<td>65</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC $ zz, X</td>
<td>(M)←(M)-1 where M=(zz+(X))</td>
<td>0 1 1 1 0 1 0 1</td>
<td>75</td>
<td>2</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC $ hhII</td>
<td>(M)←(M)-1 where M=(hhII)</td>
<td>0 1 1 0 1 1 0 1</td>
<td>6D</td>
<td>3</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC $ hhII, X</td>
<td>(M)←(M)-1 where M=(hhII+(X))</td>
<td>0 1 1 1 1 1 1 0 1</td>
<td>7D</td>
<td>3</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INX</td>
<td>(X)←(X)+1</td>
<td>0 1 1 0 0 0 0</td>
<td>8</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEX</td>
<td>(X)←(X)-1</td>
<td>0 1 1 0 0 1 0</td>
<td>8</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INY</td>
<td>(Y)←(Y)+1</td>
<td>0 1 1 0 0 0 0</td>
<td>8</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEY</td>
<td>(Y)←(Y)-1</td>
<td>0 1 1 0 0 1 0</td>
<td>8</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL $ zz, X</td>
<td>M((S)←(A)+(M)(zz+(X))</td>
<td>0 1 1 0 0 1 0 1 0</td>
<td>62</td>
<td>2</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIV $ zz, X</td>
<td>(A)←(M)(zz+(X)+1), M(zz+(X))+(A)</td>
<td>0 1 1 0 0 1 0 1 0</td>
<td>62</td>
<td>2</td>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# 740 Family Machine Language Instruction Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>SYMBOL</th>
<th>FUNCTION</th>
<th>FLAG</th>
<th>INSTRUCTION CODE</th>
<th>BYTE NUMBER</th>
<th>CYCLE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NVTBDIZC</td>
<td>D/DiDiDo</td>
<td>D/DiDiDo</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

### Logic Operation

| ORA $ $ nn | (A)←(A)+nn | M²(zz) | 0 0 0 0 1 0 0 1 | 09 | 2 | 2 |
| ORA $ zz  | (A)←(A)+M   | M²(zz) | 0 0 0 0 0 1 0 1 | 05 | 2 | 3 |
| ORA $ zz, X| (A)←(A)+M   | M²(zz+X) | 0 0 0 1 0 1 0 1 | 15 | 2 | 4 |
| ORA $ hll  | (A)←(A)+M   | M²(hill) | 0 0 0 0 1 1 0 1 | 0D | 3 | 4 |
| ORA $ hll, X| (A)←(A)+M   | M²(hill+X) | 0 0 0 1 1 0 1 0 | 1D | 3 | 5 |
| ORA $ hll, Y| (A)←(A)+M   | M²(hill+Y) | 0 0 0 1 1 0 1 1 | 19 | 3 | 5 |
| ORA $(zz, X)| (A)←(zz+X) | M²(zz+X+1) | 0 0 0 0 0 0 1 0 | 01 | 2 | 6 |
| ORA $(zz, Y)| (A)←(zz+Y) | M²(zz+Y) | 0 0 0 0 1 0 0 1 | 11 | 2 | 6 |

### Comparison

| TST $ zz | (M)←(M) | M²(zz) | 0 1 1 0 0 1 0 0 | 64 | 2 | 3 |
|          | (M)←(M) | M²(zz) | 0 1 1 0 0 1 0 0 | 64 | 2 | 3 |

### Conclusion

- The table provides a comprehensive overview of various instructions in the 740 Family Machine Language, including their flags, instruction codes, byte numbers, cycle numbers, and major operation types.
- Each instruction is categorized under different sections such as Logic Operation and Comparison, allowing for easy identification and understanding.
- The table uses a standardized format to ensure clarity and accessibility for users familiar with machine language programming.
### Classification and Shift

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>FUNCTION</th>
<th>FLAG</th>
<th>INSTRUCTION CODE</th>
<th>BYTE NUMBER</th>
<th>CYCLE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASL A</td>
<td>Left Shift $A_1 : A_{16}$ where $M = (z z)$</td>
<td>$M$</td>
<td>$1 0 0 0 1 0 1 0$</td>
<td>$0 6$</td>
<td>$2 5$</td>
</tr>
<tr>
<td>ASL $$zz</td>
<td>Left Shift $M = (z z + (X))$</td>
<td>$M$</td>
<td>$0 0 1 0 0 1 1 0$</td>
<td>$1 6$</td>
<td>$2 6$</td>
</tr>
<tr>
<td>ASL $$zz, X</td>
<td>Left Shift $M = (z z + (X))$</td>
<td>$M$</td>
<td>$0 0 1 0 0 1 1 0$</td>
<td>$0 E$</td>
<td>$3 6$</td>
</tr>
<tr>
<td>ASL $$hhl</td>
<td>Left Shift $M = (h h l + (X))$</td>
<td>$M$</td>
<td>$0 0 1 0 0 1 1 0$</td>
<td>$1 E$</td>
<td>$3 7$</td>
</tr>
<tr>
<td>LSR A</td>
<td>Right Shift 0 $A_1 : A_{16}$</td>
<td>$M$</td>
<td>$1 0 0 0 1 0 1 0$</td>
<td>$4 A$</td>
<td>$1 2$</td>
</tr>
<tr>
<td>LSR $$zz</td>
<td>Right Shift $M = (z z + (X))$</td>
<td>$M$</td>
<td>$0 1 0 1 0 1 1 0$</td>
<td>$4 6$</td>
<td>$2 5$</td>
</tr>
<tr>
<td>LSR $$zz, X</td>
<td>Right Shift $M = (z z + (X))$</td>
<td>$M$</td>
<td>$0 1 0 1 0 1 1 0$</td>
<td>$5 6$</td>
<td>$2 6$</td>
</tr>
<tr>
<td>LSR $$hhl</td>
<td>Right Shift $M = (h h l + (X))$</td>
<td>$M$</td>
<td>$0 1 0 1 0 1 1 0$</td>
<td>$0 E$</td>
<td>$3 6$</td>
</tr>
<tr>
<td>LSR $$hhl, X</td>
<td>Right Shift $M = (h h l + (X))$</td>
<td>$M$</td>
<td>$0 1 0 1 0 1 1 0$</td>
<td>$1 E$</td>
<td>$3 7$</td>
</tr>
<tr>
<td>ROL A</td>
<td>Left Shift $A_1 : A_{16}$ where $M = (z z)$</td>
<td>$M$</td>
<td>$0 1 0 0 1 1 1 0$</td>
<td>$2 A$</td>
<td>$1 2$</td>
</tr>
<tr>
<td>ROL $$zz</td>
<td>Left Shift $M = (z z + (X))$</td>
<td>$M$</td>
<td>$0 1 0 0 1 1 1 0$</td>
<td>$2 6$</td>
<td>$2 5$</td>
</tr>
<tr>
<td>ROL $$zz, X</td>
<td>Left Shift $M = (z z + (X))$</td>
<td>$M$</td>
<td>$0 1 0 0 1 1 1 0$</td>
<td>$3 6$</td>
<td>$2 6$</td>
</tr>
<tr>
<td>ROL $$hhl</td>
<td>Left Shift $M = (h h l + (X))$</td>
<td>$M$</td>
<td>$0 1 0 0 1 1 1 0$</td>
<td>$2 E$</td>
<td>$3 6$</td>
</tr>
<tr>
<td>ROL $$hhl, X</td>
<td>Left Shift $M = (h h l + (X))$</td>
<td>$M$</td>
<td>$0 1 0 0 1 1 1 0$</td>
<td>$1 E$</td>
<td>$3 7$</td>
</tr>
<tr>
<td>ROR A</td>
<td>Right Shift $A_1 : A_{16}$</td>
<td>$M$</td>
<td>$1 0 1 0 1 0 1 0$</td>
<td>$6 A$</td>
<td>$1 2$</td>
</tr>
<tr>
<td>ROR $$zz</td>
<td>Right Shift $M = (z z + (X))$</td>
<td>$M$</td>
<td>$1 0 1 0 1 0 1 0$</td>
<td>$6 6$</td>
<td>$2 5$</td>
</tr>
<tr>
<td>ROR $$zz, X</td>
<td>Right Shift $M = (z z + (X))$</td>
<td>$M$</td>
<td>$1 0 1 0 1 0 1 0$</td>
<td>$7 6$</td>
<td>$2 6$</td>
</tr>
<tr>
<td>ROR $$hhl</td>
<td>Right Shift $M = (h h l + (X))$</td>
<td>$M$</td>
<td>$1 0 1 0 1 0 1 0$</td>
<td>$6 E$</td>
<td>$3 6$</td>
</tr>
<tr>
<td>ROR $$hhl, X</td>
<td>Right Shift $M = (h h l + (X))$</td>
<td>$M$</td>
<td>$1 0 1 0 1 0 1 0$</td>
<td>$7 E$</td>
<td>$3 7$</td>
</tr>
<tr>
<td>RRF $$zz</td>
<td>Right Shift $A_1 : A_{16}$ where $M = (z z)$</td>
<td>$M$</td>
<td>$1 0 0 0 0 1 0 1 0$</td>
<td>$8 2$</td>
<td>$2 8$</td>
</tr>
</tbody>
</table>

### Bit Management

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>FUNCTION</th>
<th>FLAG</th>
<th>INSTRUCTION CODE</th>
<th>BYTE NUMBER</th>
<th>CYCLE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLB i, A</td>
<td>(Ai) ← 0 where i=0—7</td>
<td>$M$</td>
<td>$i i i 1 0 1 1 1$</td>
<td>$(1+2)$</td>
<td>$1 2$</td>
</tr>
<tr>
<td>CLB i, $$zz</td>
<td>(Mi) ← 0 where i=0—7, M=(zz)</td>
<td>$M$</td>
<td>$i i i 1 1 1 1 1 1$</td>
<td>$(1+2)$</td>
<td>$2 5$</td>
</tr>
<tr>
<td>SEB i, A</td>
<td>(Ai) ← 1 where i=0—7</td>
<td>$M$</td>
<td>$i i i 1 0 1 1 1 1$</td>
<td>$2 X 0$</td>
<td>$1 2$</td>
</tr>
<tr>
<td>SEB i, $$zz</td>
<td>(Mi) ← 1 where i=0—7, M=(zz)</td>
<td>$M$</td>
<td>$i i i 1 1 1 1 1 1$</td>
<td>$2 2 X 0$</td>
<td>$2 5$</td>
</tr>
</tbody>
</table>

### Flag setting

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>FUNCTION</th>
<th>FLAG</th>
<th>INSTRUCTION CODE</th>
<th>BYTE NUMBER</th>
<th>CYCLE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLC (C) ← 0</td>
<td>$M$</td>
<td>$0 0 0 1 1 0 0 0$</td>
<td>$1 8$</td>
<td>$1 2$</td>
<td></td>
</tr>
<tr>
<td>SEC (C) ← 1</td>
<td>$M$</td>
<td>$0 1 1 1 1 0 0 0$</td>
<td>$3 8$</td>
<td>$1 2$</td>
<td></td>
</tr>
<tr>
<td>CLD (D) ← 0</td>
<td>$M$</td>
<td>$1 1 0 0 1 0 0 0$</td>
<td>$D 8$</td>
<td>$1 2$</td>
<td></td>
</tr>
<tr>
<td>SED (D) ← 1</td>
<td>$M$</td>
<td>$1 1 1 1 1 0 0 0$</td>
<td>$F 8$</td>
<td>$1 2$</td>
<td></td>
</tr>
<tr>
<td>CLI (I) ← 0</td>
<td>$M$</td>
<td>$0 1 0 1 1 0 0 0$</td>
<td>$5 8$</td>
<td>$1 2$</td>
<td></td>
</tr>
<tr>
<td>SEI (I) ← 1</td>
<td>$M$</td>
<td>$0 1 1 1 1 0 0 0$</td>
<td>$7 8$</td>
<td>$1 2$</td>
<td></td>
</tr>
<tr>
<td>CLT (T) ← 0</td>
<td>$M$</td>
<td>$0 0 0 1 0 0 0 0$</td>
<td>$1 2$</td>
<td>$1 2$</td>
<td></td>
</tr>
<tr>
<td>SET (T) ← 1</td>
<td>$M$</td>
<td>$0 1 1 1 0 0 0 0$</td>
<td>$3 2$</td>
<td>$1 2$</td>
<td></td>
</tr>
<tr>
<td>CLV (V) ← 0</td>
<td>$M$</td>
<td>$1 0 1 1 1 0 0 0$</td>
<td>$B 8$</td>
<td>$1 2$</td>
<td></td>
</tr>
<tr>
<td>SYMBOL</td>
<td>FUNCTION</td>
<td>FLAG</td>
<td>INSTRUCTION CODE</td>
<td>BYTE NUMBER</td>
<td>CYCLE NUMBER</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>------</td>
<td>-----------------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>BJP $hh</td>
<td>(PC) ← (PC) + rel</td>
<td>N V T B D I Z C</td>
<td>1 0 0 0 0 0 0 0</td>
<td>4C</td>
<td>3 3</td>
</tr>
<tr>
<td>JMP $hh</td>
<td>(PC) ← hll</td>
<td></td>
<td>0 1 0 0 1 1 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP ($hh)</td>
<td>(PC) ← (hhh), (PCR) ← (hhh+1)</td>
<td></td>
<td>0 1 1 0 1 1 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP ($zz)</td>
<td>(PC) ← (zzz), (PCR) ← (zzz+1)</td>
<td></td>
<td>1 0 1 1 0 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSR $hh</td>
<td>(M(S)) ← (PC), (S) ← (S)−1, (M(S)) ← (PC), (S) ← (S)−1, (PC) ← hll</td>
<td></td>
<td>0 0 1 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSR ($zz)</td>
<td>(M(S)) ← (PC), (S) ← (S)−1, (M(S)) ← (PC), (S) ← (S)−1, (PC) ← hll</td>
<td></td>
<td>0 0 1 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCC $hh</td>
<td>When (A) = 0 (PC) ← (PC) + rel</td>
<td>N V T B D I Z C</td>
<td>1 0 1 0 0 0 0 0</td>
<td>90</td>
<td>2 2 4</td>
</tr>
<tr>
<td>BCS $hh</td>
<td>When (A) = 1 (PC) ← (PC) + rel</td>
<td></td>
<td>1 0 1 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNE $hh</td>
<td>When (Z) = 0 (PC) ← (PC) + rel</td>
<td></td>
<td>1 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEQ $hh</td>
<td>When (Z) = 1 (PC) ← (PC) + rel</td>
<td></td>
<td>1 1 1 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BPL $hh</td>
<td>When (N) = 0 (PC) ← (PC) + rel</td>
<td></td>
<td>0 0 1 0 0 0 0 0</td>
<td>10</td>
<td>2 2 4</td>
</tr>
<tr>
<td>BMI $hh</td>
<td>When (N) = 1 (PC) ← (PC) + rel</td>
<td></td>
<td>0 1 1 0 0 0 0 0</td>
<td>30</td>
<td>2 2 4</td>
</tr>
<tr>
<td>BVC $hh</td>
<td>When (V) = 0 (PC) ← (PC) + rel</td>
<td></td>
<td>0 1 0 0 0 0 0 0</td>
<td>50</td>
<td>2 2 4</td>
</tr>
<tr>
<td>BVS $hh</td>
<td>When (V) = 1 (PC) ← (PC) + rel</td>
<td></td>
<td>0 1 1 0 0 0 0 0</td>
<td>70</td>
<td>2 2 4</td>
</tr>
<tr>
<td>RTI</td>
<td>Previous status in stack</td>
<td></td>
<td>0 1 0 0 0 0 0 0</td>
<td>40</td>
<td>1 6</td>
</tr>
<tr>
<td>RTS</td>
<td></td>
<td></td>
<td>0 1 1 0 0 0 0 0</td>
<td>60</td>
<td>1 6</td>
</tr>
<tr>
<td>BRK</td>
<td>(BRK) ← (PC+2), (M(S)) ← (PC), (S) ← (S)−1, (M(S)) ← (PC), (S) ← (S)−1, (PC) ← BADDR</td>
<td></td>
<td>0 0 0 0 0 0 0 0</td>
<td>60</td>
<td>1 7</td>
</tr>
<tr>
<td>NOP</td>
<td>(PC) ← (PC)+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WIT</td>
<td>Internal clock source is stopped.</td>
<td></td>
<td></td>
<td></td>
<td>1 2</td>
</tr>
<tr>
<td>STP</td>
<td>Oscillation is stopped.</td>
<td></td>
<td></td>
<td></td>
<td>1 2</td>
</tr>
</tbody>
</table>
# 740 Family Machine Language Instruction Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Means</th>
<th>Symbol</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Accumulator</td>
<td>hh</td>
<td>High-order byte of address (0—255)</td>
</tr>
<tr>
<td>Al</td>
<td>Bit i of accumulator</td>
<td>ii</td>
<td>Low-order byte of address (0—255)</td>
</tr>
<tr>
<td>X</td>
<td>Index register X</td>
<td>zz</td>
<td>Zero page address (0—255)</td>
</tr>
<tr>
<td>Y</td>
<td>Index register Y</td>
<td>nn</td>
<td>Data at (0—255)</td>
</tr>
<tr>
<td>M</td>
<td>Memory</td>
<td>i</td>
<td>Data at (0—7)</td>
</tr>
<tr>
<td>Mi</td>
<td>Bit i of memory</td>
<td>iii</td>
<td>Data at (0—7)</td>
</tr>
<tr>
<td>PS</td>
<td>Processor status register</td>
<td></td>
<td>Second byte of instruction</td>
</tr>
<tr>
<td>S</td>
<td>Stack Pointer</td>
<td>&lt;B2&gt;</td>
<td>Third byte of instruction</td>
</tr>
<tr>
<td>PC</td>
<td>Program counter</td>
<td>&lt;B3&gt;</td>
<td>Relative address</td>
</tr>
<tr>
<td>PC L</td>
<td>Low-order byte of program counter</td>
<td>BADRS</td>
<td>Break address</td>
</tr>
<tr>
<td>PC H</td>
<td>High-order byte of program counter</td>
<td></td>
<td>Direction of data transfer</td>
</tr>
<tr>
<td>N</td>
<td>Negative flag</td>
<td>←</td>
<td>Contents of register of memory</td>
</tr>
<tr>
<td>V</td>
<td>Overflow flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>Direction of data transfer</td>
<td></td>
<td>Subtract</td>
</tr>
<tr>
<td>i</td>
<td>Logical mode flag</td>
<td></td>
<td>Multiplication</td>
</tr>
<tr>
<td>B</td>
<td>Break flag</td>
<td></td>
<td>Division</td>
</tr>
<tr>
<td>D</td>
<td>Decimal mode flag</td>
<td></td>
<td>Logical OR</td>
</tr>
<tr>
<td>I</td>
<td>Interrupt disable flag</td>
<td></td>
<td>Logical AND</td>
</tr>
<tr>
<td>Z</td>
<td>Zero flag</td>
<td></td>
<td>Logical Exclusive OR</td>
</tr>
<tr>
<td>C</td>
<td>Carry flag</td>
<td></td>
<td>Negative</td>
</tr>
<tr>
<td>s</td>
<td>Immediate mode</td>
<td>X</td>
<td>Stable flag after execution</td>
</tr>
<tr>
<td>s</td>
<td>Hexadecimal</td>
<td></td>
<td>Variable flag after execution</td>
</tr>
<tr>
<td>S</td>
<td>Special page mode</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Products which unuse these instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>STP</td>
<td>M37424,M37524</td>
</tr>
<tr>
<td>MUL</td>
<td>M507XX,M509XX,M37408,M37409,M37410</td>
</tr>
<tr>
<td></td>
<td>M37412,M37413,M37414,M37415,M37416,M37417</td>
</tr>
<tr>
<td></td>
<td>M37418,M37420,M37421</td>
</tr>
</tbody>
</table>

Note: Some products unuse these instructions.

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