MA8000 Controller

Instruction Manual

V1.0

MosArt

SEMICONDUCTOR CORP.

Content

前言

本指令集將介紹如何編寫 MA8000 組合語言，描述 MA8000 組合語言中每個指令的各別用法，每個指令儘可能地舉解說並附上詳細的執行過程圖，以能夠容易且更深一層的了解 MA8000 組合語言。於介紹 MA8000 的指令前，先附上各個指令用法的簡表。表一列舉經常使用的指令，表二列舉偶而才用到的指令，表三則是較少使用的使令，後續則有所有指令的完整講解。

經常使用的指令

Instruction Code	Meaning
ADC	Add with Carry
AND	Logical AND
ASL	Arithmetic Shift Left
BCC	Branch if Carry Clear
BCS	Branch if Carry Set
BEQ	Branch if Equal to Zero (Z = 1)
BMI	Branch if Minus (S = 1)
BNE	Branch if Not Equal to Zero (Z = 0)
BPL	Branch if Plus (S = 0)
CMP	Compare Accumulator to Memory
DEC	Decrement (by 1)
DEX (DEY)	Decrement Index Register X (Y) by 1
INC	Increment (by 1)
INX (INY)	Increment Index Register X (Y) by 1
JMP	Jump to New Location
JSR	Jump to Subroutine
LDA	Load Accumulator
LDX (LDY)	Load Index Register X (Y)
LSR	Logical Shift Right
PHA	Push Accumulator onto Stack
PLA	Pull Accumulator from Stack
ROL	Rotate Left through Carry
ROR	Rotate Right Through Carry
RTS	Return from Subroutine
SBC	Subtract with Borrow
STA	Store Accumulator
STX (STY)	Store Index Register X (Y)

(表一)

偶而使用的指令

Instruction Code	Meaning
BIT	Bit Test
BRK	Break
CLC	Clear Carry
CLD	Clear Decimal Mode
CLI	Clear Interrupt Mask (Enable Interrupts)
CPX (CPY)	Compare with Index Register X (Y)
EOR	Logical Exclusive-OR
NOP	No Operation
ORA	Logical (Inclusive) OR
RTI	Return from Interrupt
SEC	Set Carry
SED	Set Decimal Mode
SEI	Set Interrupt Mask (Disable Interrupts)
TAX (TAY)	Transfer Accumulator to Index Register X (Y)
TXA (TYA)	Transfer Index Register X (Y) to Accumulator

(表二)

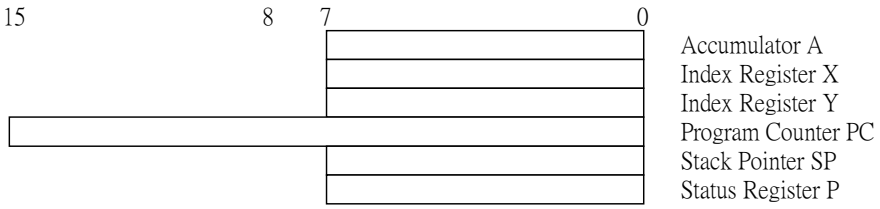
不常使用的指令

Instruction Code	Meaning
BVC	Branch if Overflow Clear
BVS	Branch if Overflow Set
CLV	Clear Overflow
PHP	Push Status Register onto Stack
PLP	Pull Status Register from Stack
TSX	Transfer Stack Pointer to Index Register X
TXS	Transfer Index Register X to Stack Pointer

(表三)

MA8000 之 CPU 暫存器以及狀態旗號

MA8000 微處理機包含有 ：1 個累加器(Accumulator, A)、1 個狀態暫存器(status register, P)、2 個索引暫存器(index register X, Y)、1 個堆疊指引器(Stack Pointer, SP)、以及 1 個程式計數器(Program Counter, PC)。其圖示如下 ：



MA8000 狀能暫存器包含了 6 個狀態旗號、一個插斷控制位元。這些狀態旗號位元分別列舉如下 :

進位狀態旗號 (C ; Carry)：

用來存放算術運算最高效位元的進位值。

零值狀態旗號 (Z ; ZERO)：

於任何算術或邏輯運算之後得到的結果若爲“0”，則旗號被置定爲“1”；若不爲 0，則被清除爲“0”。

溢位狀態旗號 (V ; OVERFLOW)：

MA8000 微處理機的溢位狀態旗號爲一典型的溢位旗號，但你也可將其視爲一般的控制輸入一樣，由外界邏輯線路置定其值。

符號狀態旗號 (S ; SIGN)：

此旗號爲算術或邏輯運算之後，所得結果的最高效位元。所以當其值爲“1”時則表示結果爲負值；反之，“0”則表示其結果爲正值。

十進位模式狀態旗號 (D ; DECIMAL MODE)：

當置定十進位模式狀態旗號爲“1”時，使得 ADC 與 SBC 執行二進位碼十進位數運算。因爲 MA8000 自動執行十進位加法與減法，所以不需像其他微處理機，需有一個輔助進位或半進位狀態旗號才可完成加減法。

中止狀態旗號 (B ; BREAK)：

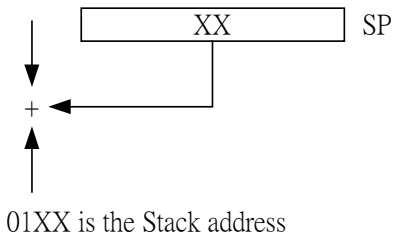
中止狀態旗號與軟體插斷有關。當執行軟體插斷指令，MA8000 自動將中止狀態旗號置定爲 1。

插斷狀態旗號 (I ; INTERRUPT)：

用以控制允許/禁止插斷；當 I=1 時禁止插斷，當 I=1 時允許插斷。

狀態暫存器是一 8 位元的暫存器，其各個旗號所應對應的位元如下圖所示：

在 MA8000 的記憶體內有一個已固定的堆疊區，此堆疊區由一個長度爲 8 個位元的疊指引器所指引，所以 MA8000 堆疊區最大長度爲 256 個位元組。CPU 固定在堆疊位址的高階位元組補上 01_{16}，所以記憶體位址從 0100_{16} 到 $01FF_{16}$ 永久指定爲堆疊區：



由於 MA8000 其堆疊位址是在記憶體的前頭，所以記憶體位址低的地方須設計爲讀/寫記憶體。爲避免與符號狀態旗號“S”混淆起見，本冊以 “SP”代表堆疊器。

進位狀態旗號“C”是用來存放算術運算後最高效位元的進位值，其有項不尋常的特性，即：在減法運算時，其進位的意義與一般狀況正好相反。執行 SBC 指令後，如果需要借位則進位旗號會被清除爲零;如果不需要借位則進位旗號會被置定爲 1。標準的零值狀態旗號“Z”，於任何算術或邏輯運算之後得到的結果若爲 0，則旗號被置定爲 1；若運算之後得到的結果不爲 0，則旗號被清除爲“0”。

MA8000 記憶體提供 11 種基本的定址模式：

1) 記憶體---立即定址法。
2) 記憶體---絕對或直接，非零頁定址法。
3) 記憶體---零頁定址法。
4) 隱含或固有定址法。
5) 累加器定址。
6) 先索引間接定址法。
7) 後索引間接定址法。
8) 零頁索引又稱基頁索引定址法。
9) 絕對索引定址法。
10) 相對定址法。
11) 間接定址法。

於下節中列舉出 MA8000 所有指令，並附上各個指令的定址方法、目的碼、所佔用位元組數、時序脈波、被影響的旗號位元、以及該指令所執行的運算。

ADC　　與進位值一起把記憶體內容加至累加器中

Operation:

Add contents of memory location, with carry, to those of Accumulator.

A←A+[addr]+C

A←A+[addr+X]+C

A←A+[[addr+X]]+C

A←A+[addr+1, addr]+Y]+C

A←A+[addr16]+C

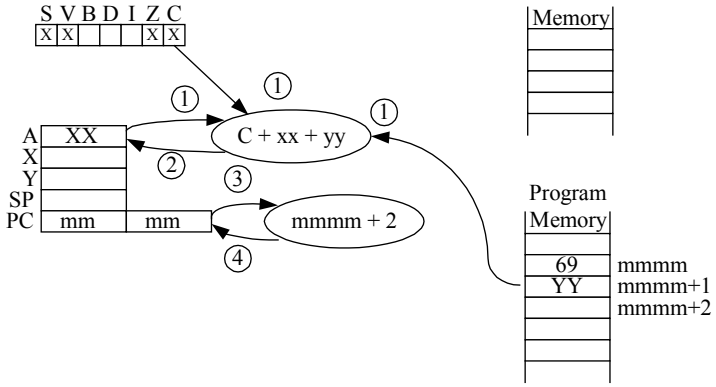
A←A+[addr16+X]+C or A←A+[addr+Y]+C

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
ADC #const	69	2	2
ADC addr	6D	3	4
ADC dp	65	2	3
ADC (dp)	72	2	5
ADC addr, X	7D	3	4
ADC addr, Y	79	3	4
ADC dp, X	75	2	4
ADC (dp, X)	61	2	6
ADC (dp), Y	71	2	5

Flags:

n	v	-	b	d	i	z	c
✓	✓	-	-	-	-	✓	✓



Example:

假設: xx = $3A_{16}$

yy = $7C_{16}$

C = 1

經過執行下列指令: ADC#$7C

則累加器的內容爲 $B7_{16}$

運算過程如下:

3A = 00111010

7C = 01111100

Carry = 0

10110111

Nonzero result sets Z to 0

No carry, Set C to 0

1 sets S to 1

0 V 1 = 1, Set V to 1

AND　記憶體及累加器內容之邏輯“且”(AND)

Operation:

AND contents of Accumulator with those of memory location.

A←AΛ[addr]+C

A←AΛ[addr+X]+C

A←AΛ[[addr+X]]+C

A←AΛ[addr+1, addr]+Y]+C

A←AΛ[addr16]+C

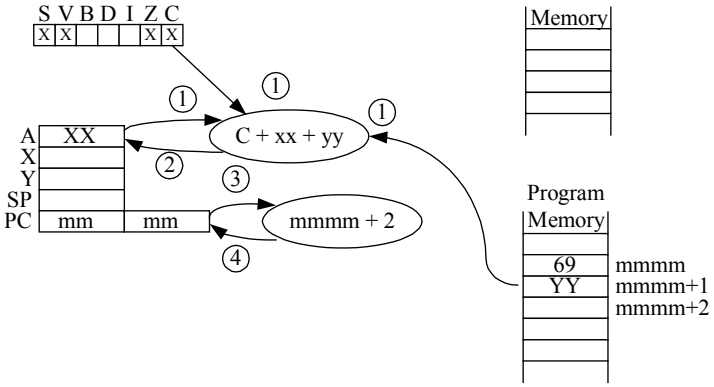
A←AΛ[addr16+X]+C or A←A+[addr+Y]+C

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
AND #const	29	2	2
AND addr	2D	3	4
AND dp	25	2	3
AND (dp)	32	2	5
AND addr, X	3D	3	4
AND addr, Y	39	3	4
AND dp, X	35	2	4
AND (dp, X)	21	2	6
AND (dp), Y	31	2	5

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-



Example:

假設: xx = FC_{16}

yy = 13_{16} == (0040_{16})

經過執行下列指令: AND $40

則累加器的內容爲 10_{16}

運算過程如下:

FC = 11111100

13 = 00010011

00010000

0 in bit 1 sets S to 0

Nonzero result sets Z to 0

ASL　累加器或記憶體位元組左移

Operation:

Arithmetic shift left contents of memory location. Index through Register X only.

[addr]

[addr+X]

[addr 16]

[addr 16+X]

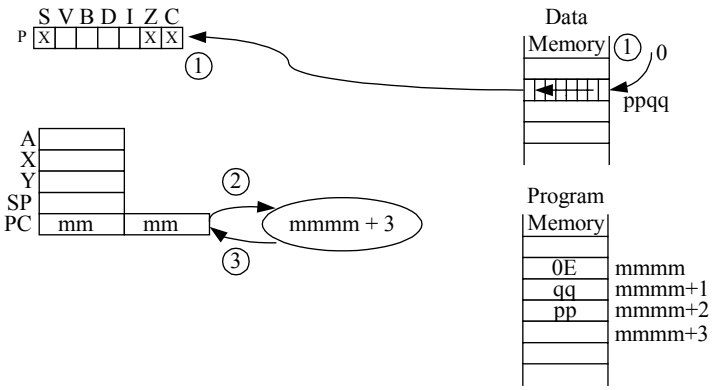
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
ASL A	0A	1	2
ASL addr	0E	3	6
ASL dp	06	2	5
ASL addr, X	1E	3	7
ASL dp, X	16	2	6

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	✓

ASL addr 動作



Example:

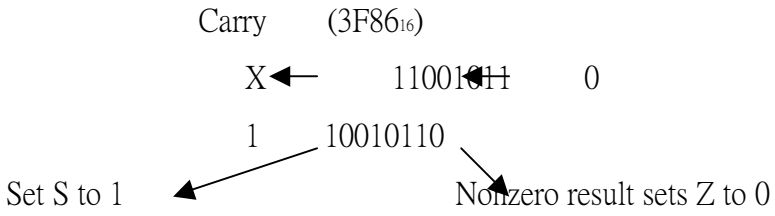
假設: ppqq = $3F86_{16}$

(ppqq) = CB_{16}

經過執行下列指令: ASL $3F86

則 3F86 位址的內容爲 96_{16}，且 C 旗號的值被設定爲 1

運算過程如下:



BCC　若進位旗號被清除則分支 (C = 0)

Operation:

Branch relative if Carry flag is cleared.

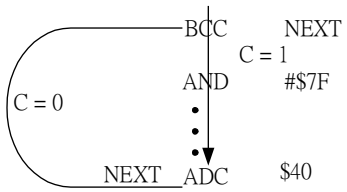
If C = 0, then PC←PC+disp

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
BCC nearlabel	90	2	2

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-



Example:

當指令執行至 BCC NEXT 時，即判別 C 旗號之值；若 C = 0 則執行 ADC　$40 指令；若 C = 1 則繼續執行 AND　#$7F

BCS　若進位旗號被置定則分支 (C = 1)

Operation:

Branch relative if Carry flag is set.

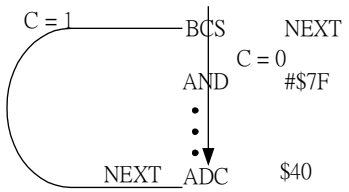
If C = 1, then PC←PC+disp

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
BCS nearlabel	B0	2	2

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-



Example:

當指令執行至 BCS NEXT 時，即判別 C 旗號之值；若 C = 1 則分至 ADC　$40 指令；若 C = 0 則繼續執行 AND　#$7F

BEQ　　若等於 0 則分支 (Z = 1)

Operation:

Branch relative if result is equal to zero.

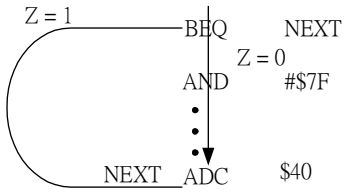
If Z = 1, then PC←PC+disp

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
BEQ nearlabel	F0	2	2

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-



Example:

當指令執行至 BEQ NEXT 時，若 Z = 1 則執行 ADC　$40 指令；若 Z = 0 則繼續執行 AND　#$7F

BIT　　位元測試

Operation:

AND contents of Accumulator with those of memory location, Only the status bits are affected.

A Λ [addr]

A Λ [addr 16]

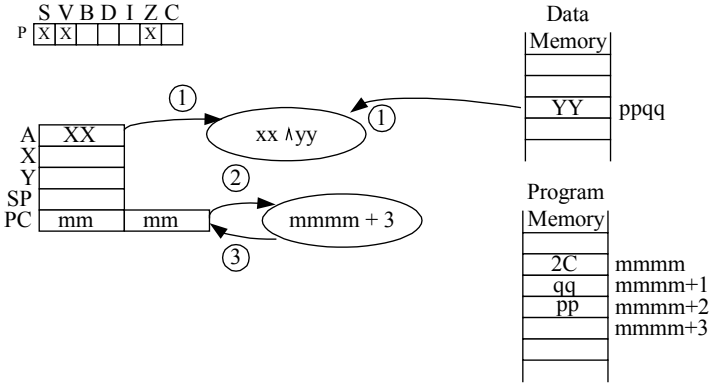
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
BIT #const	89	2	2
BIT addr	2C	3	4
BIT dp	24	2	3
BIT addr, X	3C	3	4
BIT dp, X	34	2	4

Flags:

n	v	-	b	d	i	z	c
✓	✓	-	-	-	-	✓	-

BIT addr 動作:



Example:

假設:　xx = $A6_{16}$

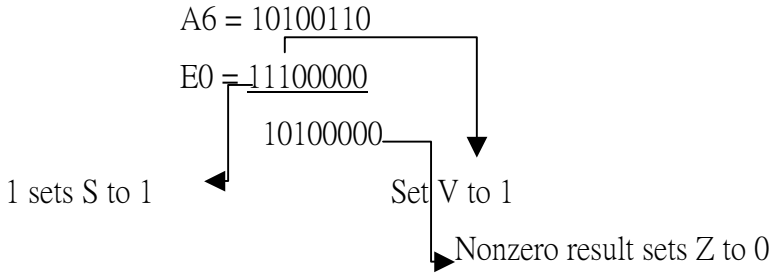
yy = $E0_{16}$

ppqq = 1641_{16}

經過執行下列指令:　　BIT　$1641

則累加器的內容爲 $A6_{16}$，記憶體 1641_{16} 之內仍爲 $E0_{16}$

狀態旗號則如下:



Note:　BIT 指令常出現於條件分支指令(如 BEQ、BCS…)的前面，而且 BIT 指令也被用來執行對 DATA 做遮罩(Masking)的功能。

BMI　若負值則分支 (S = 1)

Operation:

Branch relative if result is negative.

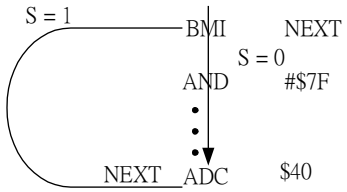
If S = 1, then PC←PC+disp

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
BMI nearlabel	30	2	2

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-



Example:

當指令執行至 BMI NEXT 時，若 S = 1 則執行 ADC　$40 指令；若 S =0 則繼續執行 AND　#$7F

BNE　若不等於零則分支 (Z = 0)

Operation:

Branch relative if result is not zero.

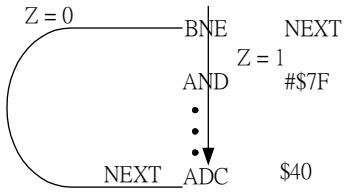
If Z = 0, then PC←PC+disp

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
BNE nearlabel	D0	2	2

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-



Example:

當指令執行至 BNE NEXT 時，若 Z = 0 則執行 ADC　$40 指令；若 Z = 1 則繼續執行 AND　#$7F

BPL　若正值則分支 (S = 0)

Operation:

Branch relative if result is positive.

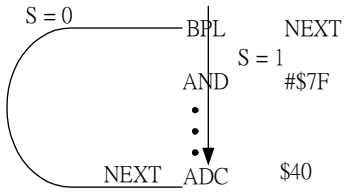
If S = 0, then PC←PC+disp

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
BPL nearlabel	10	2	2

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-



Example:

當指令執行至 BPL NEXT 時，若 S = 0 則執行 ADC　$40 指令；若 S = 1 則繼續執行 AND　#$7F

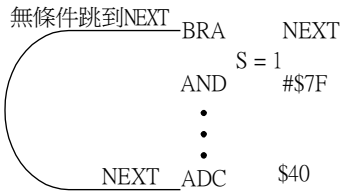
BRA　　直接位址跳越

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
BRA nearlabel	80	2	3

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-



BRK　強制中止(軟體插斷)

Operation:

Programmed interrupt. BRK cannot be disabled. The Program Counter is incremented twice before it is saved on the Stack.

[SP]←PC(HI)

[SP-1]←PC(LO)

[SP-2]←P

SP←SP-3

PC(HI)←[FFFF]

PC(LO)←[FFFE]

I←1

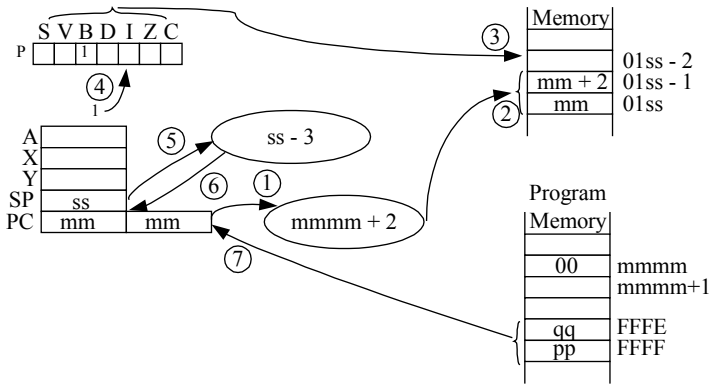
B←1

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
BRK	00	2	7

Flags:

n	v	-	b	d	i	z	c
-	-	-	✓	✓	✓	-	-



BRK 指令可爲程式提供一個中斷點(Break point)，以方便除錯(debug)，也可以將 CPU 的控制權適時地轉移給某些特別重要的軟体系統。

執行完 BRK 指令後，程式計數器的內容變爲 ppqq；其中 pp 爲記憶體位址 $FFFF_{16}$的內容，而 qq 則爲 $FFFE_{16}$的內容。

Note: 程式設計師必須加一段指令，以判別 BRK 指令或硬体 IRQ 中斷；方法爲檢查 B 旗號爲 0 或 1，程式如下：

```
PLA             讀取狀態暫存器
PHA             儲存於 STACK
AND #$10            是 BRK 引起中斷？
BNE BRK_PROG        是的，執行 BRK 中斷
```

BVC　若溢位旗號被清除則分支 (V = 0)

Operation:

Branch relative if Overflow flag is cleared.

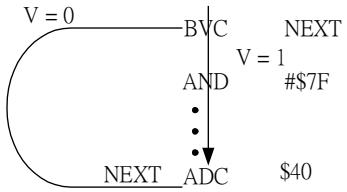
If V = 0, then PC←PC+disp

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
BVC nearlabel	50	2	2

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-



Example:

當指令執行至 BVC NEXT 時，若 V = 0 則執行 ADC　$40 指令；若 V = 1 則繼續執行 AND　#$7F

BVS　若溢位旗號被置定則分支 (V = 1)

Operation:

Branch relative if Overflow flag is set.

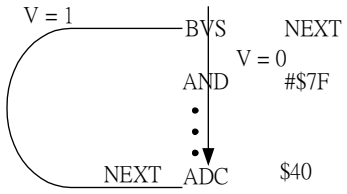
If V = 1, then PC←PC+disp

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
BVS nearlabel	70	2	2

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-



Example:

當指令執行至 BPL NEXT 時，若 V = 1 則執行 ADC　$40 指令；若 VS = 0 則繼續執行 AND　#$7F

CLC　　清除進位旗號

Operation:

Clear Carry flag

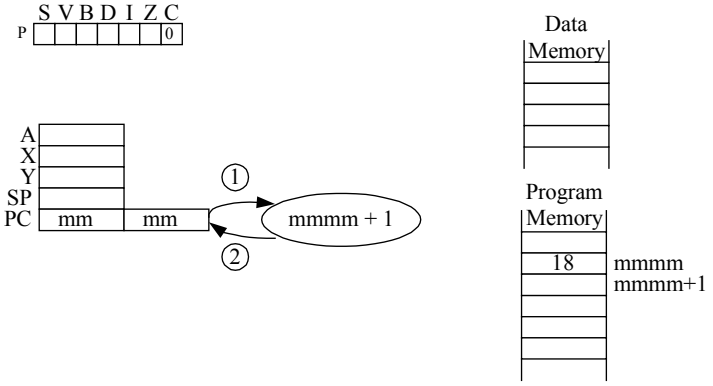
C←0

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
CLC	18	1	2

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	✓



CLD　清除十進位模式旗號

Operation:

Clear Decimal flag

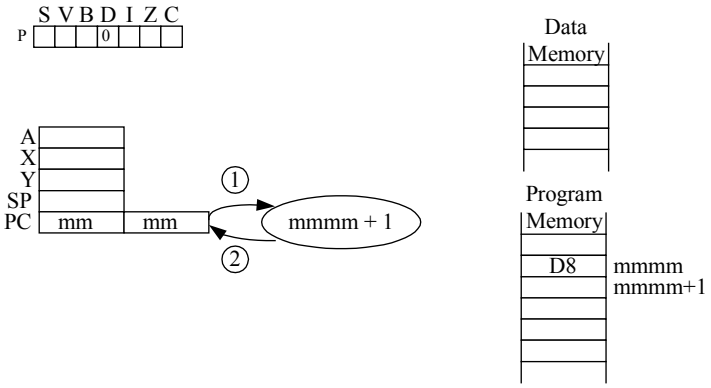
D←0

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
CLD	D8	1	2

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	✓	-	-	-



CLI　　清除插斷遮罩 (允許插斷)

Operation:

Enable interrupts by clearing　interrupt disable bit of Status register.

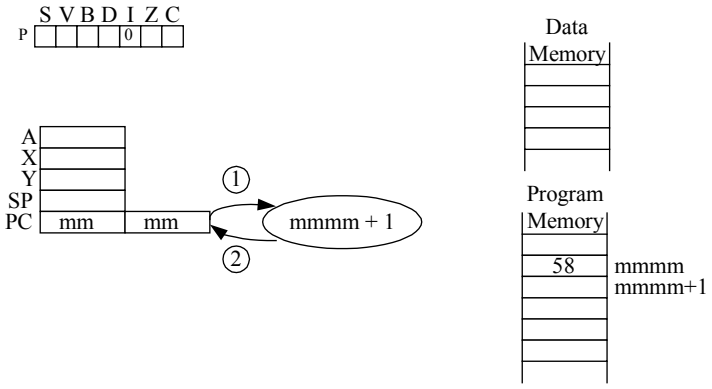
I←0

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
CLI	58	1	2

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	✓	-	-



CLV　清除溢位旗號

Operation:

Clear Overflow flag

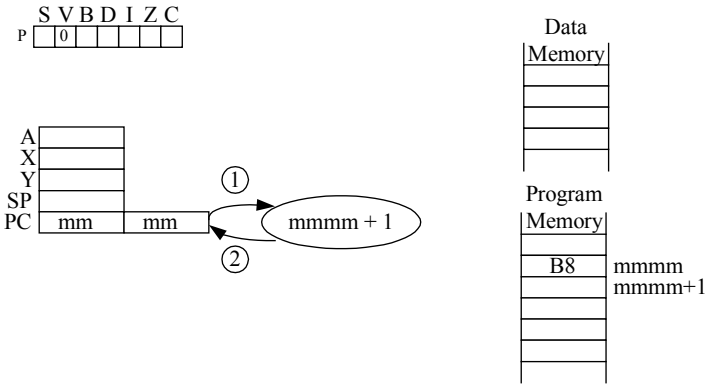
V←0

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
CLV	B8	1	2

Flags:

n	v	-	b	d	i	z	c
-	✓	-	-	-	-	-	-



CMP　累加器與記憶體內容相比

Operation:

Compare immediate with Accumulator. Only the status flags are affected.

A - data

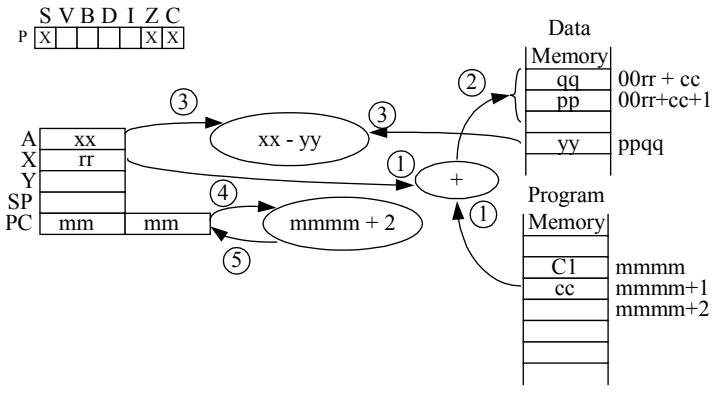
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
CMP #const	C9	2	2
CMP addr	CD	3	4
CMP dp	C5	2	3
CMP (dp)	D2	2	5
CMP addr, X	DD	3	4
CMP addr, Y	D9	3	4
CMP dp, X	D5	2	4
CMP (dp, X)	C1	2	6
CMP (dp), Y	D1	2	5

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	✓

CMP (dp, X)動作:



Example:

假設: $xx = FF_{16}$

$yy = 18_{16}$

$(0043_{16}) = 6D_{16}$

$(0044_{16}) = 15_{16}$

$rr = 20_{16}$

$cc = 23_{16}$

$0043_{16} = rr + cc$

$(156D_{16}) = yy = 18_{16}$

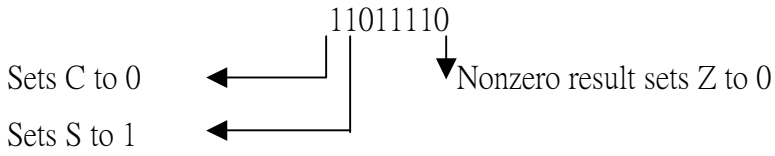
經過執行下列指令: CMP($23, X)

則累加器的內容爲 $F6_{16}$，且記憶體位址 $156D_{16}$的內容也仍然是 18_{16}

旗號則修改如下:

F6 = 11110110

Twos complement of 18 = 11101000



Note: 此指令最常出現在條件分支指令的前面，被用來設定各項狀態旗號。

CPX　X 暫存器與記憶體內容相比

Operation:

Compare immediate with Index Register X. Only the status flags are affected.

X - data

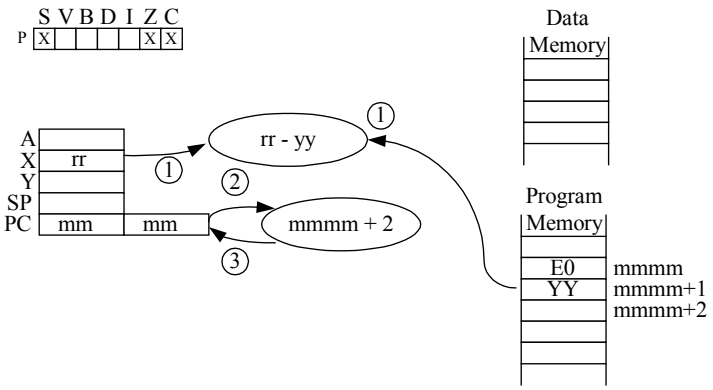
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
CPX #const	E0	2	2
CPX addr	EC	3	4
CPX dp	E4	2	3

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	✓

CPX　#const 動作:



CPY　Y 暫存器與記憶體內容相比

Operation:

Compare immediate with Index Register Y. Only the status flags are affected.

Y - data

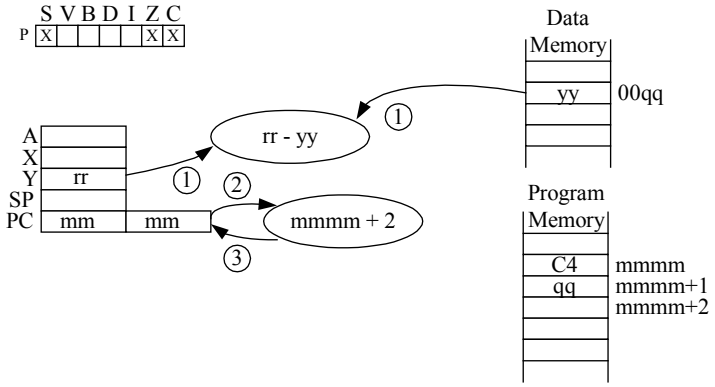
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
CPY #const	C0	2	2
CPY addr	CC	3	4
CPY dp	C4	2	3

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	✓

CPY dp 動作:



DEC　記憶體內容減 1

Operation:

Decrement contents of memory location. Index through Register X only.

[addr]←[addr]-1

[addr+X]←[addr+X]+1

[addr 16]←[addr 16]-1

[addr 16+X]←[addr 16+X]-1

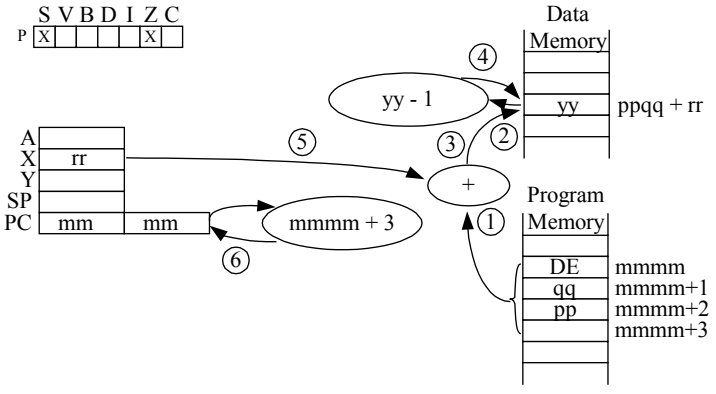
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
DEC A	3A	1	2
DEC addr	CE	3	6
DEC dp	C6	2	5
DEC addr, X	DE	3	7
DEC dp,X	D6	2	6

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-

DEC addr, X 動作:



Example:

假設: yy = $A5_{16}$

ppqq$_{16}$ = 0100_{16}

rr = $0A_{16}$

經過執行下列指令: DEC $0100, X

則記憶體位址 $010A_{16}$ 內容變爲 $A4_{16}$。

旗號改變如下:

A5 = 10100101

Ones complement of 1 = 11111111

Carry is not altered 10100100

Sets S to 1

Nonzero result sets Z to 0

Overflow (V) is not altered

DEX　X 暫存器內容減 1

Operation:

Decrement contents of Index Register X.

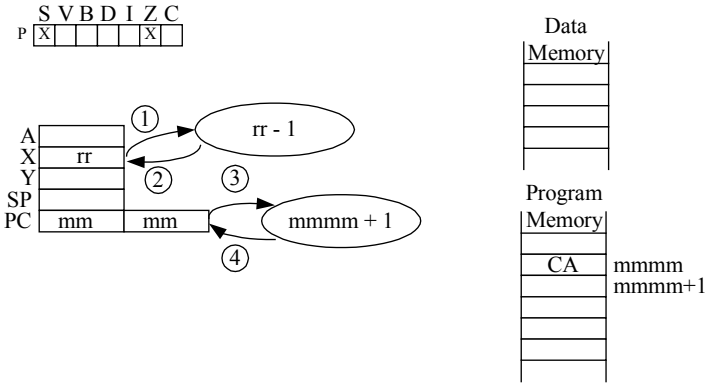
X←X-1

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
DEX	CA	1	2

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-



DEY　Y 暫存器內容減 1

Operation:

Decrement contents of Index Register Y.

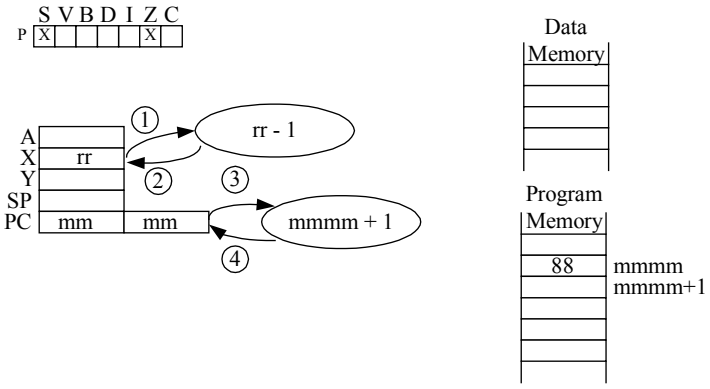
Y←Y-1

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
DEY	88	1	2

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-



EOR　累加器與記憶體內容邏輯互斥或 (EXCLUSIVE – OR)

Operation:

Exclusive-OR contents of Accumulator with those of memory location.

A←A V [addr]+C

A←A V [addr+X]+C

A←A V [[addr+X]]+C

A←A V [addr+1, addr]+Y]+C

A←A V [addr16]+C

A←A V [addr16+X]+C or A←A+[addr+Y]+C

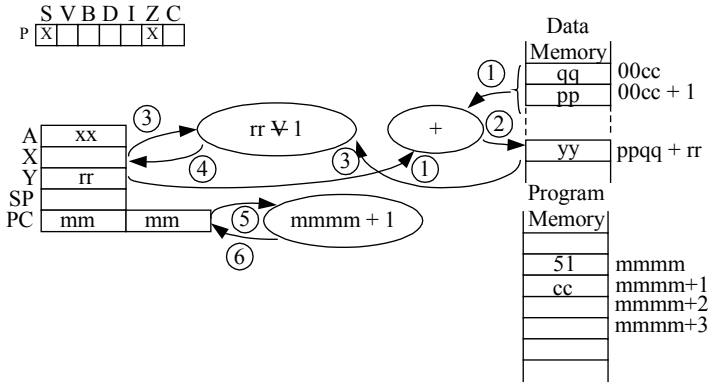
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
EOR #const	49	2	2
EOR addr	4D	3	4
EOR dp	45	2	3
EOR (dp)	41	2	5
EOR addr, X	5D	3	4
EOR addr, Y	59	3	4
EOR dp, X	55	2	4
EOR (dp, X)	41	2	6
EOR (dp), Y	51	2	5

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-

EOR (dp), Y 動作



Example:

假設: $xx = E3_{16}$

$yy = A0_{16}$

$rr = 10_{16}$

$qq = (40_{16}) = 1E_{16}$

$pp = (41_{16}) = 25_{16}$

$(252E_{16}) = yy = A0_{16}$

經過執行下列指令: EOR ($40), Y

則累加器內容變爲 43_{16}。

旗號改變如下:

E3 = 11100011

A0 = 10100000

01000011

0 Sets S to 0 Nonzero result sets Z to 0

EOR 指令常被用來檢查位元狀態的改變。

注意：此指令會將累加器內每一個爲 “1”的位元改爲 “0”，而每一個爲 “0”的位元爲 “1”。

INC　　記憶體內容增加 1

Operation:

Increment contents of memory location. Index through Register X only.

[addr]←[addr]+1

[addr+X]←[addr+X]+1

[addr 16]←[addr 16]+1

[addr 16+X]←[addr 16+X]+1

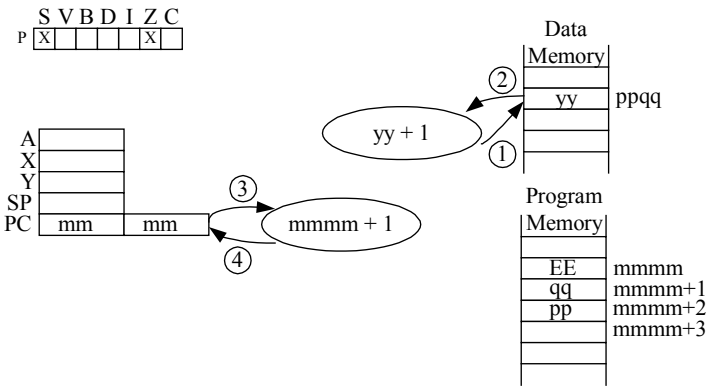
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
INC A	1A	1	2
INC addr	EE	3	6
INC dp	E6	2	5
INC addr, X	FE	3	7
INC dp, X	F6	2	6

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-

INC addr 動作:



Example:

假設: $pp = 01_{16}$

$qq = A2_{16}$

$(01A2_{16}) = yy = C0_{16}$

經過執行下列指令: INC $01A2

則位址 $01A2_{16}$ 之內容變爲 C1

旗號改變如下:

C0 = 11000000

1 = 00000001

11000001

Sets S to 1

Nonzero result sets Z to 0

Carry and Overflow are not altered

INX　　X 暫存器內容增加 1

Operation:

Increment contents of Index Register X.

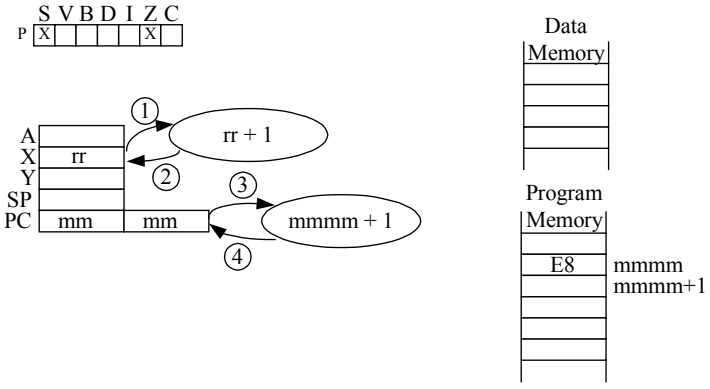
X←X+1

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
INX	E8	1	2

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-



INY　　Y 暫存器內容增加 1

Operation:

Increment contents of Index Register Y.

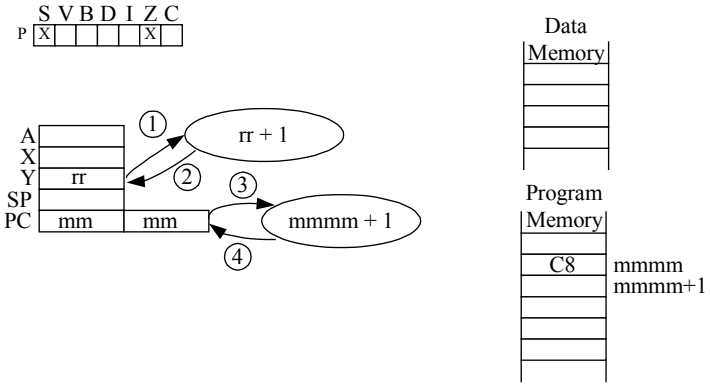
Y←Y+1

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
INY	C8	1	2

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-



JMP　直接或間接位址跳越

Operation:

Jump to new location, using extended or indirect addressing.

PC←label or PC←[label]

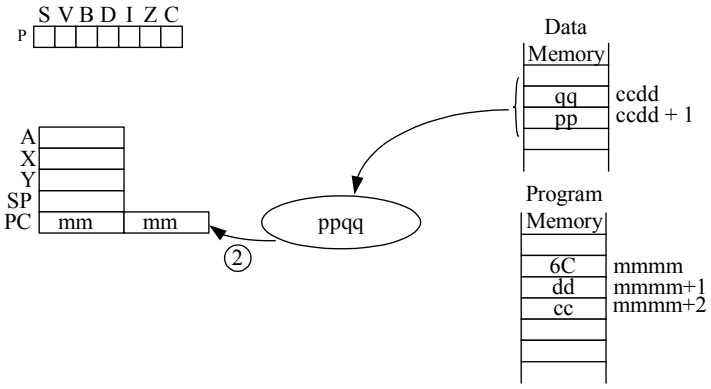
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
JMP addr	4C	3	3
JMP (addr)	6C	3	5
JMP (addr, X)	7C	3	6

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-

JMP (addr)動作:



JSR　　跳到副程式

Operation:

Jump to subroutine beginning at address given in bytes 2 and 3 of the instruction. Note that the stored Program Counter points to the last byte of the JSR instruction.

[SP]←PC(HI)

[SP-1]←PC(LO)

SP←SP-2

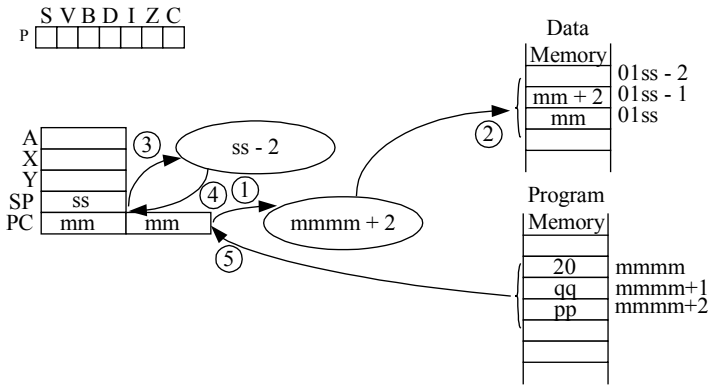
PC←label

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
JSR addr	20	3	6

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-



Example:

假設:　mmmm = $E34F_{16}$

ss = $E3_{16}$

($01A2_{16}$) = yy = $C0_{16}$

經過執行下列指令:　JSR $E100

則程式計數器內容變爲 $E100_{16}$，堆疊指引器內容變爲 $E1_{16}$(=E3-2)

LDA　記憶體資料載入累加器中

Operation:

Load Accumulator with immediate data.

A←data

Load Accumulator from memory.

A←[addr]

A←[addr+X]

A←[[addr+1, addr]+Y]

A←[addr 16]

A←[addr 16+X] or A←[addr 16+Y]

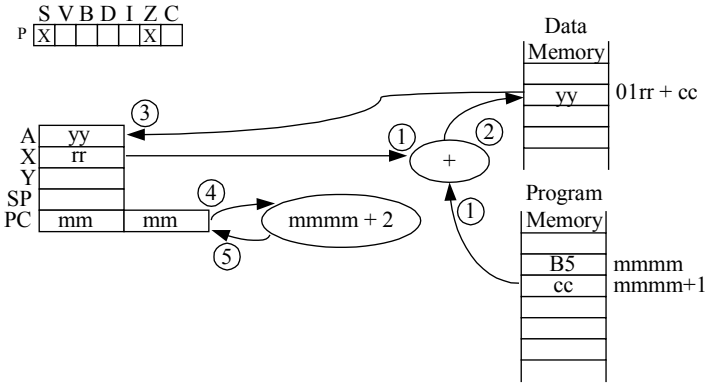
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
LDA #const	A9	2	2
LDA addr	AD	3	4
LDA dp	A5	2	3
LDA (dp)	B2	2	5
LDA addr, X	BD	3	4
LDA addr, Y	B9	3	4
LDA dp, X	B5	2	4
LDA (dp, X)	A1	2	6
LDA (dp), Y	B1	2	5

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-

LDA dp, X 動作:



Example:

假設: X 暫存器內容爲 10_{16}

且 cc = 43_{16}

(0053_{16}) = AA_{16}

經過執行下列指令: LDA$43, X

則累加器內容變爲 AA_{16}

旗號改變如下:

AA = 11000000

1 Sets S to 1

Nonzero result sets Z to 0

LDX　將記憶體內容載入 X 暫存器中

Operation:

Load Index Register X with immediate data.

X←data

Load index Register X from memory. Index through Register Y only.

X←[addr]

X←[addr+Y]

X←[addr 16]

X←[addr 16+Y]

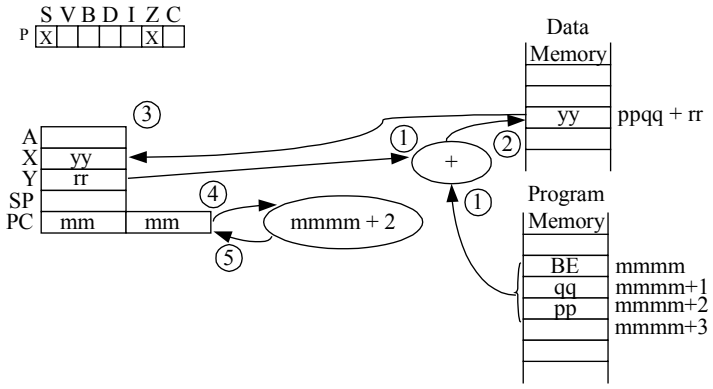
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
LDX #const	A2	2	2
LDX addr	AE	3	4
LDX dp	A6	2	3
LDX addr, Y	BE	3	4
LDX dp, Y	B6	2	4

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-

LDX addr, Y 動作:



Example:

假設: Y 暫存器內容爲 28_{16}

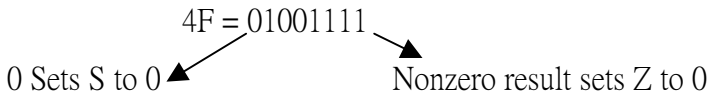
且 ppqq = $2E1A_{16}$

yy = ($2E42_{16}$) = $4F_{16}$

經過執行下列指令: LDX$2E1A, Y

則 X 暫存器內容爲 $4F_{16}$

旗號改變如下:



LDY　將記憶體內容載入 Y 暫存器中

Operation:

Load Index Register Y with immediate data.

Y←data

Load index Register Y from memory. Index through Register X only.

Y←[addr]

Y←[addr+X]

Y←[addr 16]

Y←[addr 16+X]

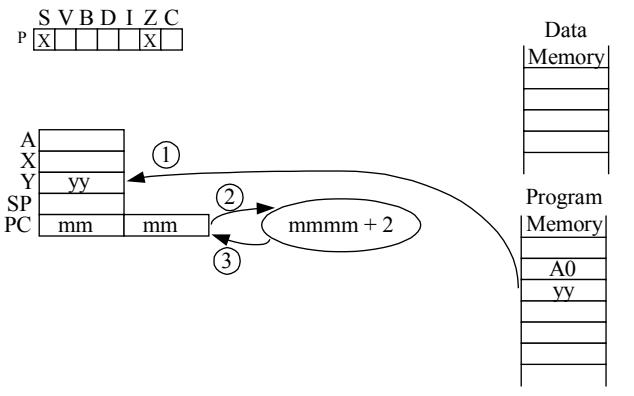
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
LDY #const	A0	2	2
LDY addr	AC	3	4
LDY dp	A4	2	3
LDY addr, X	BC	3	4
LDY dp, X	B4	2	4

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-

LDY #const 動作:



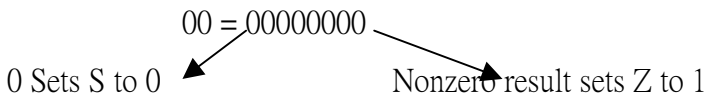
Example:

假設:　　yy = 00_{16}

經過執行下列指令:　　LDY#0

則不管 Y 暫存器原內容爲何，其結果變爲 00_{16}。

旗號改變如下:



LSR　累加器或記憶體內容邏輯右移

Operation:

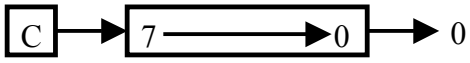
Logical shift right contents of memory location. Index through Register X only.

[addr]

[addr+X]

[addr 16]

[addr 16, X]



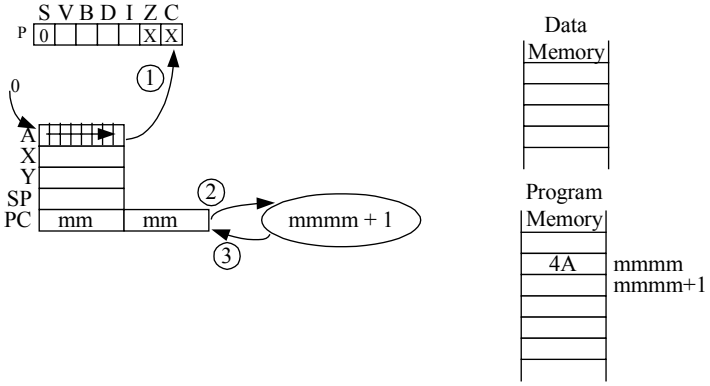
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
LSR A	4A	1	2
LSR addr	4E	3	6
LSR dp	46	2	5
LSR addr, X	5E	3	7
LSR dp, X	56	2	6

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	✓

LSR A 動作:



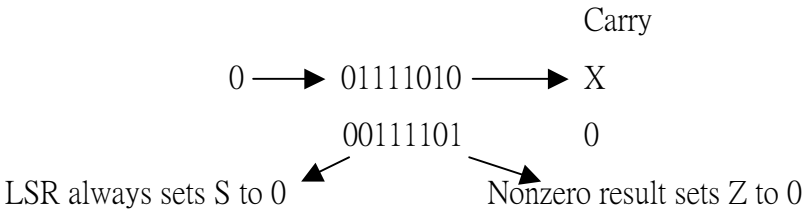
Example:

假設:　　累加器內容爲 $7A_{16}$

經過執行下列指令:　　LSR A

則累加器內容變爲 $3D_{16}$

旗號改變如下:



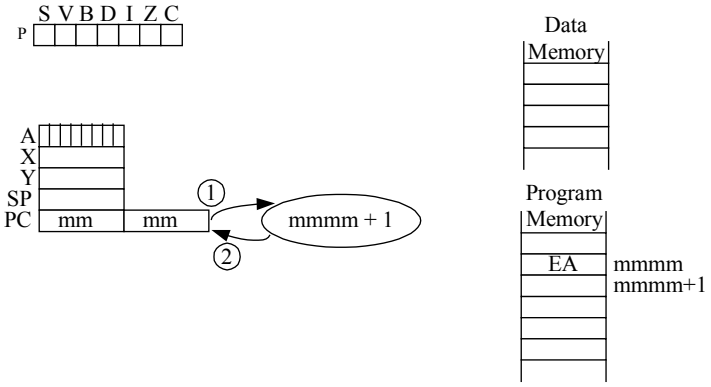
NOP　無運算

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
NOP	EA	1	2

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-



NOP 是個佔據一個位元組的指令，它除了令程式計數器加一外，並無其它動作產生。通常用來做時間延遲，每個 NOP 指令需 2 個時序脈波，在程式查錯時，程式會更正或改變，可利用 NOP 指令來替代不再使用的指令，並可用來替代某些您不想用的指令(例如 JSR)。在一個測試完成的程式裡，一般是很少有 NOP 指令；但 NOP 指令對查錯或測試常常很有用。

ORA　累加器與記憶體內容邏輯“或”(OR)運算

Operation:

OR contents of Accumulator with those of memory location.

A←A ∨ [addr]

A←A ∨ [addr+X]

A←A ∨ [[addr+X]]

A←A ∨ [addr+1, addr]+Y]

A←A ∨ [addr16]

A←A ∨ [addr16+X] or A←A+[addr+Y]

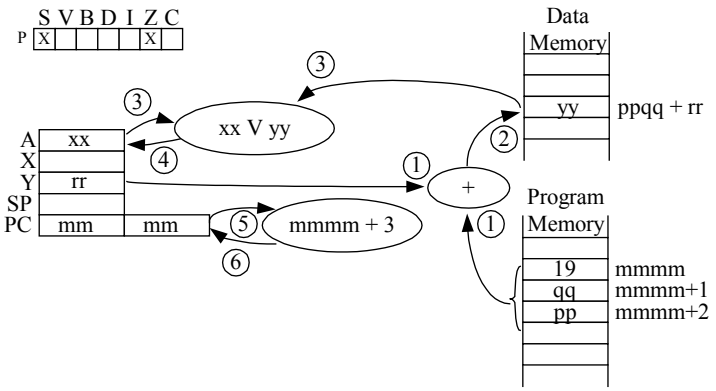
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
ORA #const	09	2	2
ORA addr	0D	3	4
ORA dp	05	2	3
ORA (dp)	12	2	5
ORA addr, X	1D	3	4
ORA addr, Y	19	3	4
ORA dp, X	15	2	4
ORA (dp, X)	01	2	6
ORA (dp), Y	11	2	5

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-

ORA addr, Y 動作:



假設: ppqq = 1623_{16}

rr = 10_{16}

xx = $E3_{16}$

(1623_{16}) = yy = AB_{16}

經過執行下列指令: ORA$1623, Y

則累加器內容爲 EB_{16}

旗號改變如下:

E3 = 11100011

AB = 10101011

11101011

Sets S to 1 Nonzero result sets Z to 0

ORA 爲一邏輯指令，常用來開啓位元(Turn bit “ON”)亦即使位元爲 “1”。

PHA　累加器的內容推入 (PUSH) 堆疊器上

Operation:

Push Accumulator contents onto Stack.

[SP] ←A

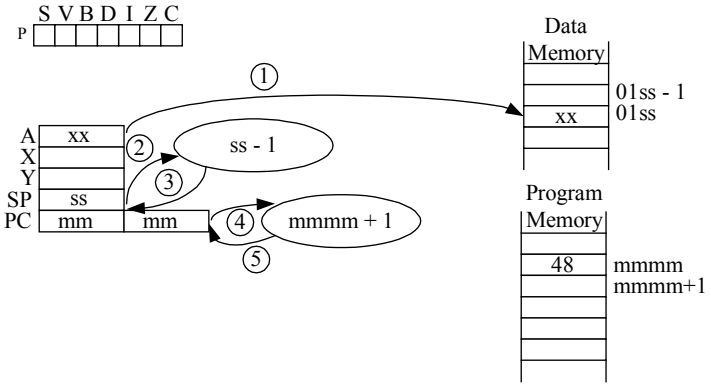
SP←SP-1

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
PHA	48	1	3

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-



PHA 指令作用爲將累加器內容推入至堆疊器頂端，將堆疊指引器值減 1。執行結果並不影響其他暫存器及所有旗號。注意：累加器的內容存至堆疊後再將堆疊指引器的內容減 1。

假設累加器內容爲 $3A_{16}$，而堆疊指引器的內容爲 $F7_{16}$，經過執行 PHA 指令後累加器的內容仍是 $3A_{16}$，堆疊指引器的值減 1 後變爲 $F6_{16}$，此時堆疊器最頂端的值爲 $3A_{16}$，即$(01F7_{16})=3A_{16}$。

PHA 指令最常用在呼用副程式，或做插斷服務之前儲存累加器的內容。

PHP　狀態暫存器的內容推入堆疊器上

Operation:

Push Status register contents onto Stack.

[SP] ←P

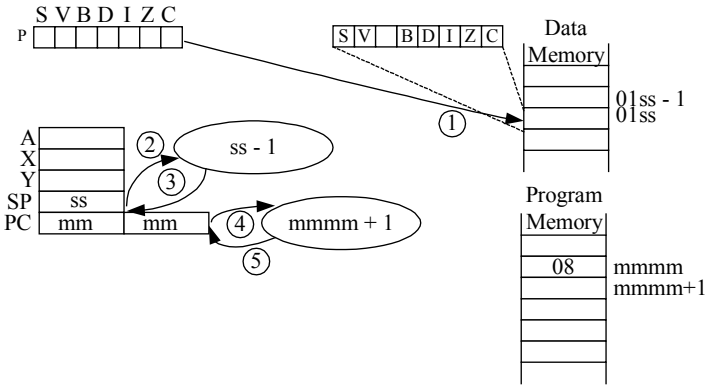
SP←SP-1

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
PHP	08	1	3

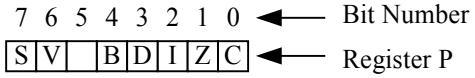
Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-



PHP 指令爲將狀態暫存器的內容堆入至堆疊器頂端，將堆疊指引器的值減 1，其執行結果並不影響其它暫存器及所有旗號。

在記憶體內的狀態暫存器組成如下所示：



其中第五位元組沒有用到，它的值爲 0 或 1 並不會有所影響。

PHP 指令通常使用在呼用一個副程式之前，將狀態暫存器的內容先儲存起來。應注意的是：在插斷服務或 BRK 指令之前，並不須執行 PHP 指令，因爲 6502 在插斷服務或中止(BRK)動作產生之前，會自動將狀態暫存器的內容推入至堆疊器的頂端。

PHX　　X 暫存器的內容推入(Push)堆疊器上

Operation:

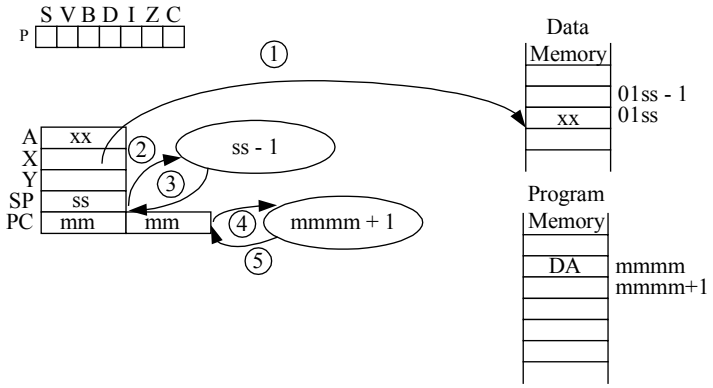
Push X register contents onto Stack.

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
PHX	DA	1	3

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-



PHY　　Y 暫存器的內容推入(Push)堆疊器上

Operation:

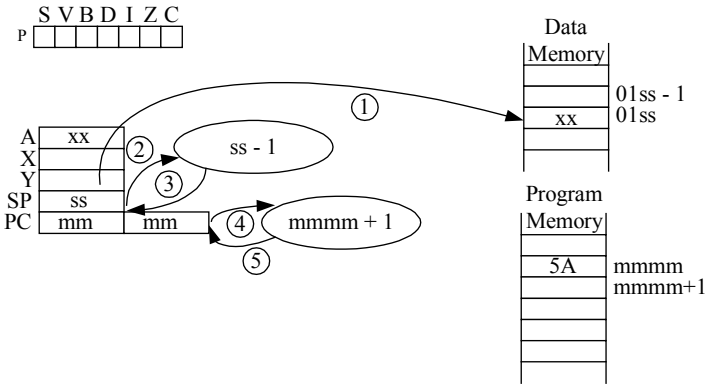
Push Y register contents onto Stack.

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
PHY	5A	1	3

Flags:

n	v	-	b	d	i	z	c
✓	✓	-	-	-	-	✓	✓



PLA　　提出 (PULL)堆疊器頂端的內容置入累加器中

Operation:

Load Accumulator from top of Stack (“Pull”).

A←[SP+1]

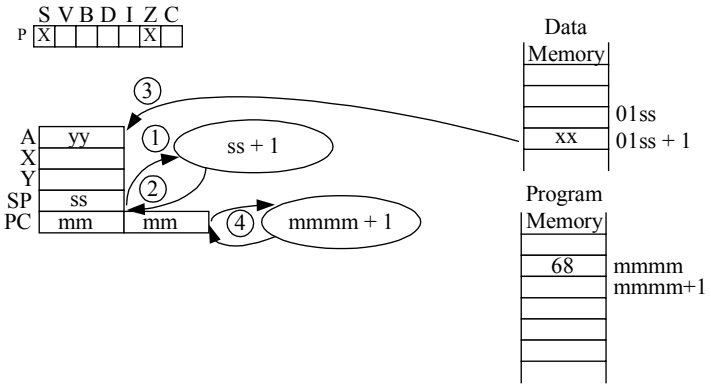
SP←SP+1

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
PLA	68	1	4

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-



Example:

假設:　　堆疊指引器的值爲 $F6_{16}$

且記憶體位址 $01f7_{16}$的內容爲 CE_{16}

經過執行指令:　　PLA

則累加器內容爲 CE_{16} ，堆疊指引器的值爲 $F7_{16}$

PLP　　提出 (PULL)堆疊器頂端的內容置於狀態暫存器中

Operation:

Load Status register from top of Stack (“Pull”).

P←[SP+1]

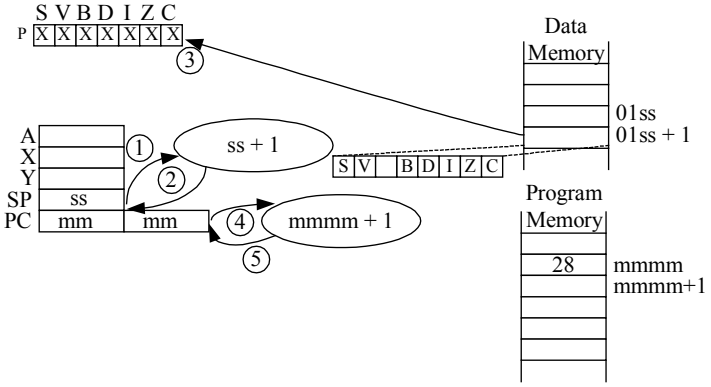
SP←SP+1

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
PLP	28	1	4

Flags:

n	v	-	b	d	i	z	c
✓	✓	-	✓	✓	✓	✓	✓



PLP 指令是先將堆疊指引器之值加 1，取出堆疊器頂端的內容置於狀態暫存器中，對其它暫存器並無影響；但狀態暫存器中的各個旗號可能會有所改變。

PLP 指令是用來提取先前暫存在堆疊器中的狀態暫存器之內容，它與 PHP 指令是相互爲用的；即在呼用副程式之前若有 PHP 動作，則完成副程式之後就以 PLP 指令來恢復呼用副程式之前的原始狀態。注意：當插斷服務完成後，並不須執行 PLP 指令，因 RTI 指令會自動的將堆疊器頂端之內容提出，並置入狀態暫存器內。

PLX　　提出(PULL)堆疊器頂端的內容置於 X 暫存器內

Operation:

$X \leftarrow [SP + 1]$

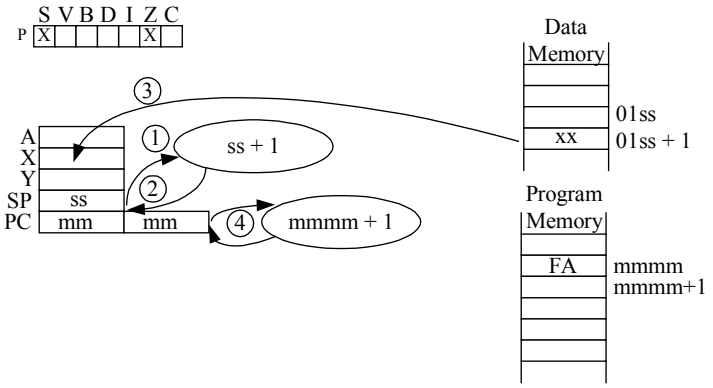
$SP \leftarrow SP + 1$

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
PLX	FA	1	4

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-



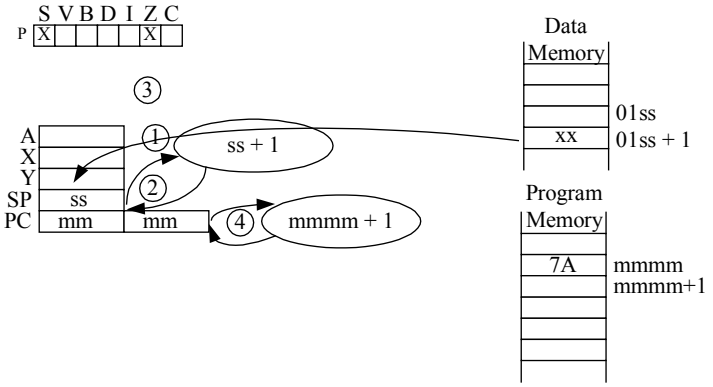
PLY　　提出(PULL)堆疊器頂端的內容置於 Y 暫存器內

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
PLY	7A	1	4

Flags:

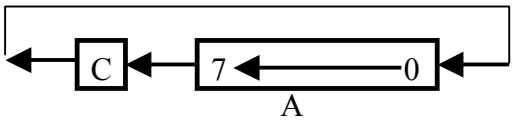
n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-



ROL　累加器記憶體內容經由進位旗號 (C)左旋轉

Operation:

Rotate contents of Accumulator left through Carry.

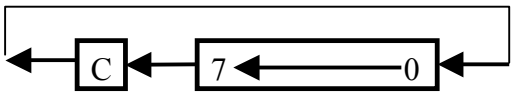


Rotate contents of memory location one bit left through Carry. Index through Register X only.

[addr]

[addr+X]

[addr 16]



[addr 16, X]

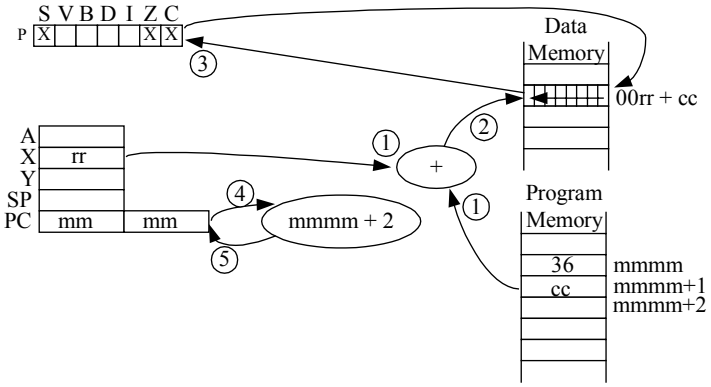
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
ROL A	2A	1	2
ROL addr	2E	3	6
ROL dp	26	2	5
ROL addr, X	3E	3	7
ROL dp, X	36	2	6

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	✓

ROL dp, X 動作:



Example:

假設: cc = 34_{16}

rr = 16_{16}

($004A_{16}$) = $2E_{16}$

進位旗號值(C) = 0

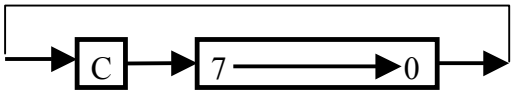
經過執行指令: ROL $34, X

則記憶體位址 $004A_{16}$ 內容變爲 $5C_{16}$

ROR　累加器或記憶體內容經由進位旗號 (C) 右旋轉

Operation:

Rotate contents of Accumulator right through Carry.



A

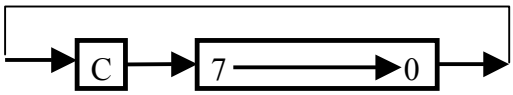
Rotate contents of memory location one bit right through Carry. Index through Register X only.

[addr]

[addr+X]

[addr 16]

[addr 16, X]



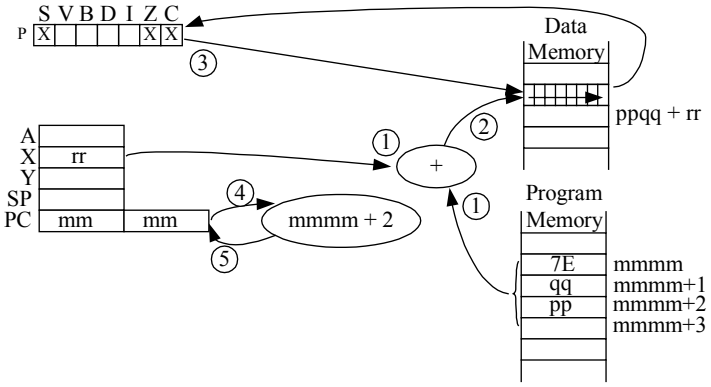
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
ROR A	6A	1	2
ROR addr	6E	3	6
ROR dp	66	2	5
ROR addr, X	7E	3	7
ROR dp, X	76	2	6

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	✓

ROR addr, X 動作:



Example:

假設: rr = 14_{16}

ppqq = 0100_{16}

(0114_{16}) = ED_{16}

進位旗號值爲 1

經過執行指令: ROR $0100, X

則記憶體位址 0114_{16} 內容變爲 $F6_{16}$

RTI　　從插斷返回

Operation:

Return from interrupt; restore Status

P←[SP+1]

PC(LO)←[SP+2]

PC(HI)←[SP+3]

SP←SP+3

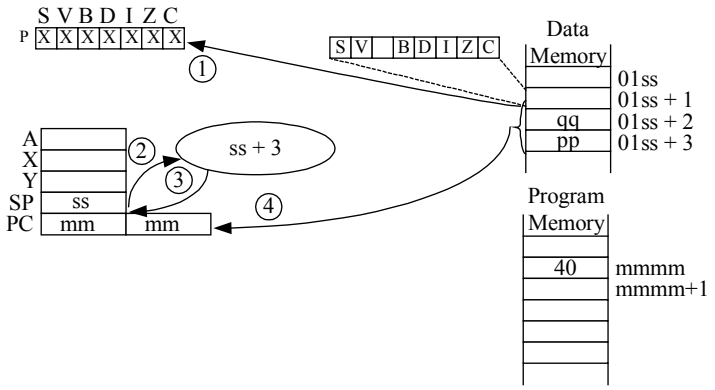
PC←PC+1

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
RTI	40	1	6

Flags:

n	v	-	b	d	i	z	c
✓	✓	-	-	✓	✓	✓	✓



Example:

假設:　堆疊指引器之值爲 $E8_{16}$

(01E9) = $C1_{16}$

($00EA_{16}$) = $3E_{16}$

($01EB_{16}$) = $D5_{16}$

經過執行指令:　RTI

則堆疊指引器之值變爲 EB_{16} ，程式計數器內容變爲 $D53E_{16}$，狀態暫存器則變爲 $C1_{16}$。如下所示：

C =

S	V		B	D	I	Z	C
1	1	0	0	0	0	0	1

注意：假設插斷服務常式沒有改變存於堆疊器裡的插斷遮罩位元，那麼插斷遮罩位元是以原先被儲存的狀態暫存器之內容而定的。

RTS　從副程式返回

Operation:

Return from subroutine, incrementing Program Counter to point to the instruction after the JSR which called the routine.

PC(LO)←[SP+1]

PC(HI)←[SP+2]

SP←SP+2

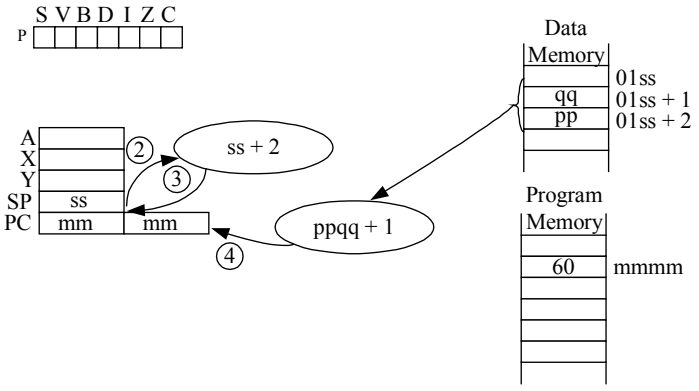
PC←PC+1

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
RTS	60	1	6

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-



Example:

假設:　堆疊指引器之值爲 DF_{16}

$(01E0_{16}) = 08_{16}$

$(01E1_{16}) = 7C_{16}$

經過執行指令:　RTS

則堆疊指引器之值變爲 $E1_{16}$ ，程式計數器內容變爲 $7C09_{16}$

SBC　從累加器中減去記憶體內容及進位旗號值的補數

Operation:

Subtract immediate, with borrow from Accumulator.

A←A-data-C

Subtract contents of memory location, with borrow, from contents of Accumultor.

A←A-[addr]-C

A←A-[addr+X]-C

A←A-[[addr+X]]-C

A←A-[[addr+1, addr]+Y]-C

A←[addr 16]-C

A←[addr 16+X]-C or A←A-[addr 16+Y]-C

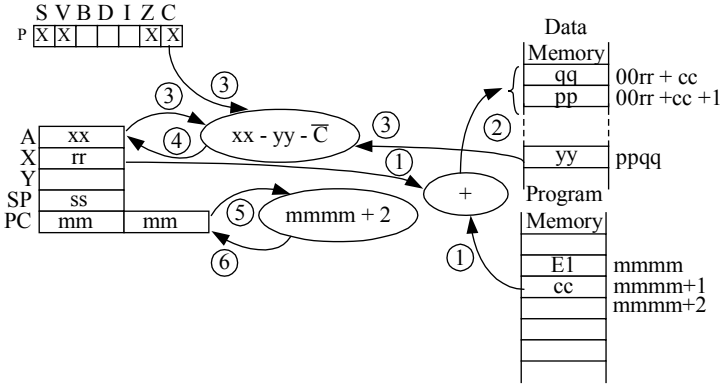
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
SBC #const	E9	2	2
SBC addr	ED	3	4
SBC dp	E5	2	3
SBC (dp)	F2	2	5
SBC addr, X	FD	3	4
SBC addr, Y	F9	3	4
SBC dp, X	F5	2	4
SBC (dp, X)	E1	2	6
SBC (dp), Y	F1	2	5

Flags:

n	v	-	b	d	i	z	c
✓	✓	-	-	-	-	✓	✓

SBC (dp, X)動作:



Example:

假設: $xx = 14_{16}$

$cc = 15_{16}$

$rr = 37_{16}$

$ppqq = 07E2_{16}$

$yy = (07E2_{16}) = 34_{16}$

$(004C_{16}) = E2_{16}$

$(004D_{16}) = 07_{16}$

經過執行指令: SBC ($15, X)

則累加器內容變爲 DF_{16}

注意：執行後的結果其進位旗號不是借位的意思，而其意義剛好相反；如不需借位，則進位旗號被置爲 1；如須借位，則進位旗號被清爲 0。

SBC 爲 6502 指令群中唯一的減法指令。欲利用其在單一位元組或兩個多位元組低階位元組的減法運算時，須利用 SEC 指令將 C 旗號置定爲 1，使其不影響運算結果。

SEC　置定進位旗號值爲 1 (C = 1)

Operation:

Set Carry flag

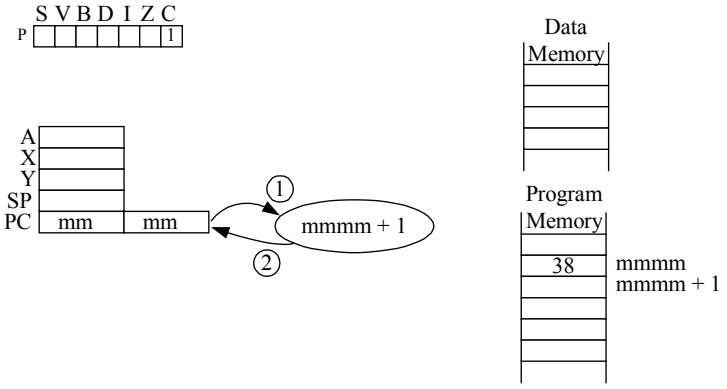
C←1

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
SEC	38	1	2

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	✓



SEC 指令將進位旗號 C 置定爲 1，C 以外各旗號以及其他暫存器的內容都無影響。SEC 可視爲 SBC 指令的一部份，因 SBC 爲 6502 所提供唯一的減法運算指令，因其尚須減去 C 旗號值的補數之故，故在執行 SBC 指令前，先執行 SEC 指令，以避免低位元組之減法產生錯誤之結果。

SED　置定十進位模式旗號值爲 1 (D = 1)

Operation:

Set Decimal Mode

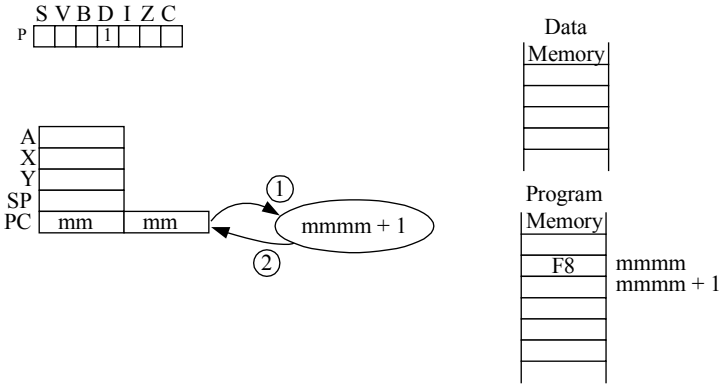
D←1

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
SED	F8	1	2

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	✓	-	-	-



SED 指令將十進位模式旗號 D 置爲 1，對 D 以外各旗號以及其它暫存器的內容都無影響。SED 指令是與 ADC 或 SBC 兩指令一起用的。

當 D=0 時，ADC 及 SBC 都是執行二進位運算；當 D=1 則執行的是十進位運算。注意：依據 D 旗號值的不同，相同的程式會產生不同的結果，且若不小心監督 D 旗號值，程式可能會產生令人困惑與錯誤百出的結果。

SEI　　置定插斷遮罩旗號爲 1 (I = 1, 禁止插斷)

Operation:

Disable interrupts

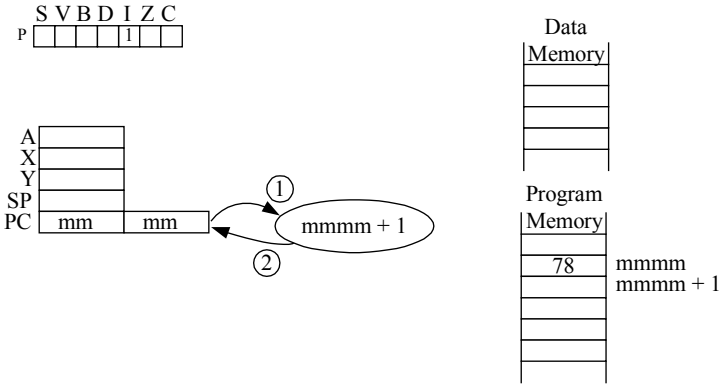
I←1

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
SEI	78	1	2

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	✓	-	-



SEI 指令將使 6502 失去接受插斷的處理能力，即 6502 對插斷要求控制線不做任何反應。

在一般情況下，若 I=0，當 IRQ 爲低電位時，6502 會接受插斷要求，而處理由 $FFFE_{16}$ 與 $FFFF_{16}$ 位址內所指的插斷服務程式。除 I 位元以外，其它各位元與暫存器皆不受 SEI 指令影響。

STA　累加器的內容儲存至記憶體中

Operation:

Store Accumulator to memory

[addr]←A

[addr+X]←A

[[addr+X]]←A

[[addr+1, addr]+Y]←A

[addr 16]←A

[addr 16+X]←A or [addr 16+Y]←A

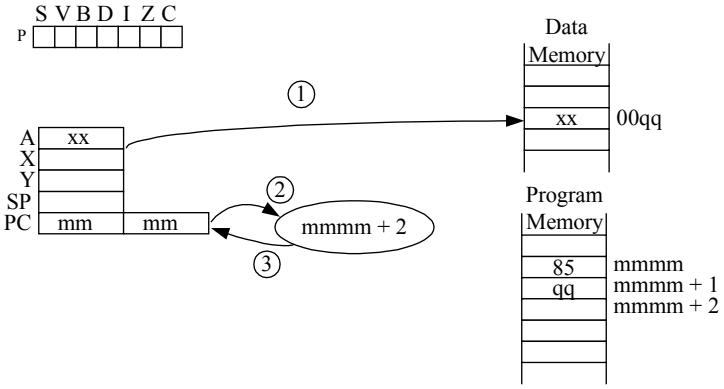
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
STA addr	8D	3	4
STA dp	85	2	3
STA (dp)	92	2	5
STA addr, X	9D	3	5
STA addr, Y	99	3	5
STA dp, X	95	2	4
STA (dp, X)	81	2	6
STA (dp), Y	91	2	6

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-

STA dp 動作:



Example:

假設: xx = 63_{16}

qq = $3A_{16}$

經過執行指令: STA $3A

則記憶體位址 $003A_{16}$ 的內容變爲 63_{16}，STA 指令對所有暫存器或是旗號值都沒影響。

STX　X 暫存器的內容儲存至記憶體中

Operation:

Store Index Register X to memory. Index through Register Y only.

[addr]←X

[addr+Y]←X

[addr 16]←X

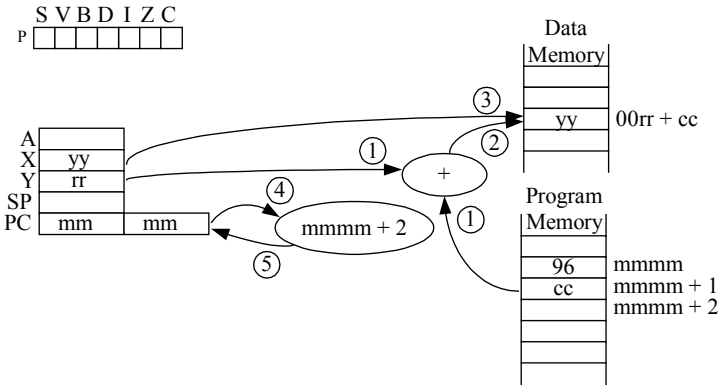
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
STX addr	8E	3	4
STX dp	86	2	3
STX dp, Y	96	2	4

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-

STX dp, Y 動作:



Example:

假設:　cc = 28_{16}

rr = 20_{16}

yy = $E9_{16}$

經過執行指令:　STX　$28, Y

則記憶體位址 0048_{16} 的內容變爲 $E9_{16}$，STX 指令對所有暫存器或是旗號值都沒影響。

STY　　Y 暫存器的內容儲存至記憶體中

Operation:

Store Index Register X to memory. Index through Register Y only.

[addr]←Y

[addr+X]←Y

[addr 16]←Y

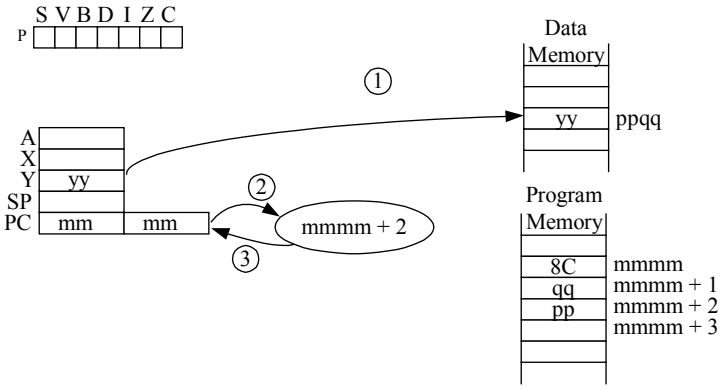
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
STY addr	8C	3	4
STY dp	84	2	3
STY dp, X	94	2	4

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-

STY addr 動作:



Example:

假設:　　yy = 01_{16}

　　ppqq = $08F3_{16}$

經過執行指令:　　STY　$08F3

則記憶體位址 $08F3_{16}$ 的內容變爲 01_{16}，STY 指令對所有暫存器或是旗號值都沒影響。

STZ　　將 0 寫入記憶体某一位址內

Operation:

M ← 00

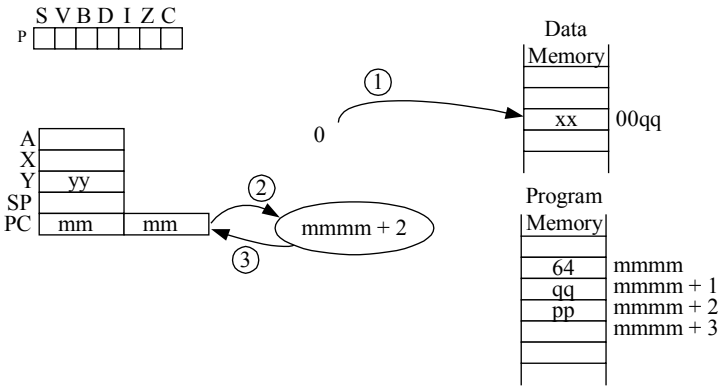
Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
STZ addr	9C	3	4
STZ dp	64	2	3
STZ addr, X	9E	3	5
STZ dp, X	74	2	4

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-

STZ dp 動作:



TAX　累加器的內容轉移到 X 暫存器中

Operation:

Move Accumulator contents to Index Register X

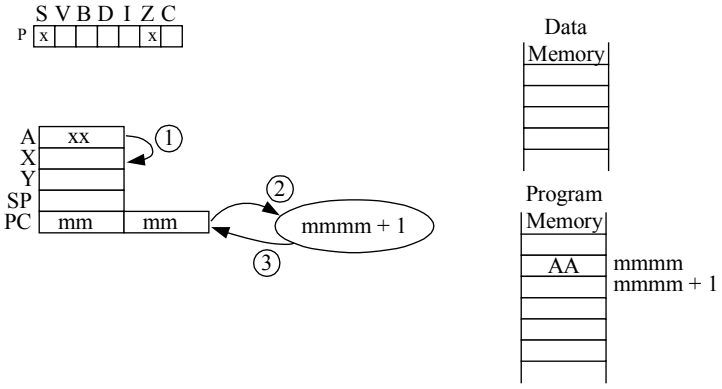
X←A

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
TAX	AA	1	2

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-



Example:

假設:　$XX = 00_{16}$

經過執行指令:　TAX

則累加器及 X 暫存器的內容皆爲 00_{16}，TAX 指令對所有暫存器或是旗號值都沒影響。

TAY　累加器的內容轉移到 Y 暫存器中

Operation:

Move Accumulator contents to Index Register Y

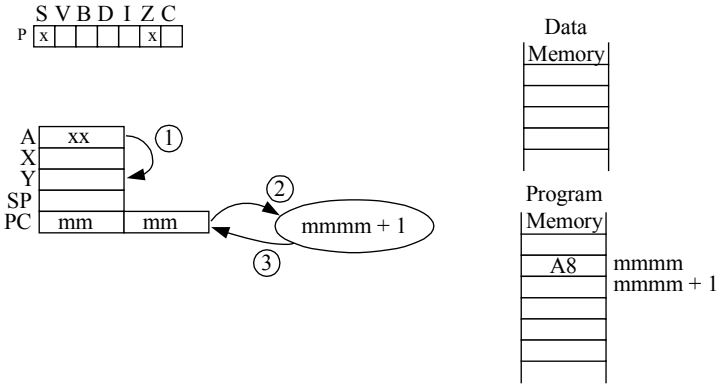
Y←A

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
TAY	A8	1	2

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-



Example:

假設:　rr = $F1_{16}$

經過執行指令:　TAY

則累加器及 Y 暫存器的內容皆爲 $F1_{16}$，TAY 指令對所有暫存器或是旗號值都沒影響。

TRB　累加器內容的補數與記憶體內容作邏輯“且”(AND 運算，結果存回記憶體

Operation:

Test and Reset Memory Bits Against Accumulator

$M \leftarrow \overline{A} \wedge M$

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
TRB addr	1C	3	6
TRB dp	14	2	5

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	✓	-

Example:

假設:　(A) = $5A_{16}$

(8765_{16}) = $7F_{16}$

經過執行指令:　TRB 8765

則　$\overline{5A} \wedge 7F_{16} = A5_{16} \wedge 7F_{16} = 25_{16}$

再將 25_{16} 寫入位址 8765_{16}

TSB　累加器與記憶體內容作邏輯“或”(OR)運算，結果存回記憶體

Operation:

Test and Set Memory Bits Against Accumulator

$M \leftarrow A \vee M$

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
TSB addr	0C	3	6
TSB dp	04	2	5

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	✓	-

Example:

假設:　(A) = $5A_{16}$

(8765_{16}) = $7F_{16}$

經過執行指令:　TSB 8765_{16}

則　5A　Λ　$25F_{16}$ = $7F_{16}$

再將 $7F_{16}$ 寫入位址 8765_{16}

TSX　堆疊指引器的內容轉移至 X 暫存器中

Operation:

Move contents of Stack Pointer to Index Register X

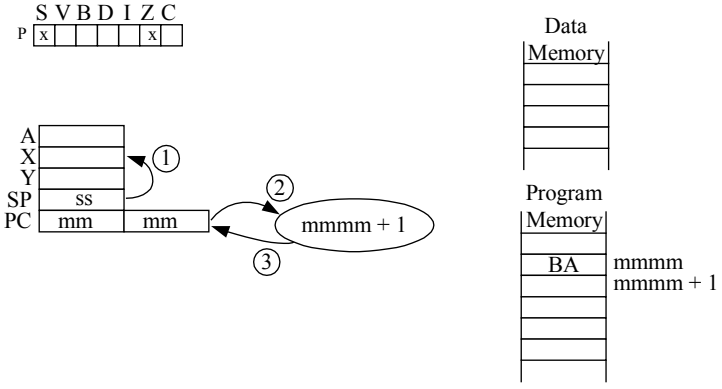
X←SP

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
TSX	BA	1	2

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-



Example:

假設:　ss = ED_{16}

經過執行指令:　TSX

則 X 暫存器及堆疊指引器的內容皆爲 ED_{16}，S 與 Z 旗號則改變如下：。

11101101

Sets S to 1　　Nonzero result sets Z to 0

TXA　X 暫存器之內容轉移至累加器中

Operation:

Move contents of Index Register X to Accumulator

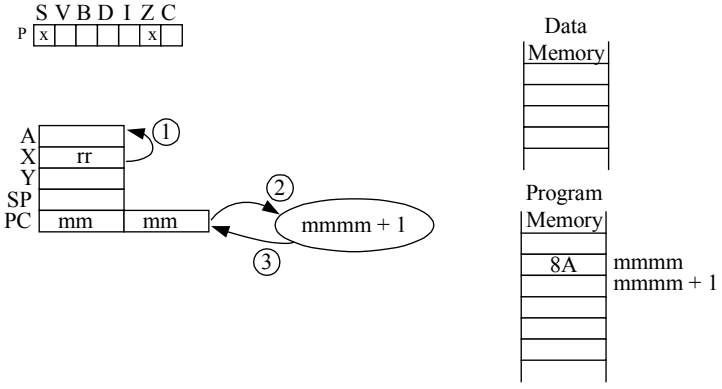
A←X

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
TXA	8A	1	2

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-



Example:

假設:　rr = $3B_{16}$

經過執行指令:　TXA

則 X 暫存器及累加器的內容皆爲 $3B_{16}$，S 與 Z 旗號則改變如下：。

00111011

Sets S to 1　　Nonzero result sets Z to 0

TXS　X 暫存器之內容轉移至堆疊指引器中

Operation:

Move contents of Index Register X to Stack Pointer

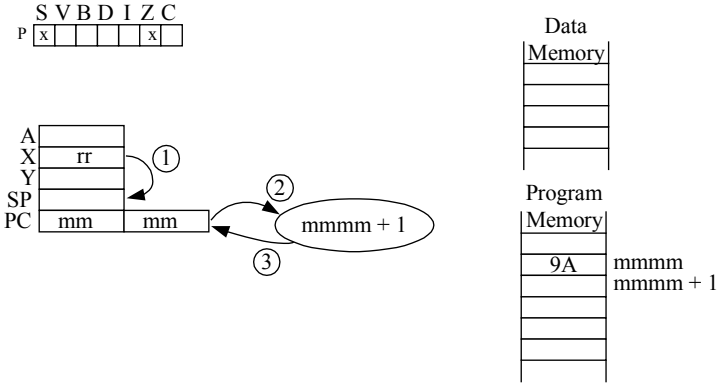
SP←X

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
TXS	9A	1	2

Flags:

n	v	-	b	d	i	z	c
-	-	-	-	-	-	-	-



Example:

假設:　rr = $F6_{16}$

經過執行指令:　TXS

則 X 暫存器及堆疊指引器的內容皆爲 $F6_{16}$，使得目前的堆疊指引器之內容變爲 $01F6_{16}$。

TYA　Y 暫存器之內容轉移至累加器中

Operation:

Move contents of Index Register Y to Accumulator

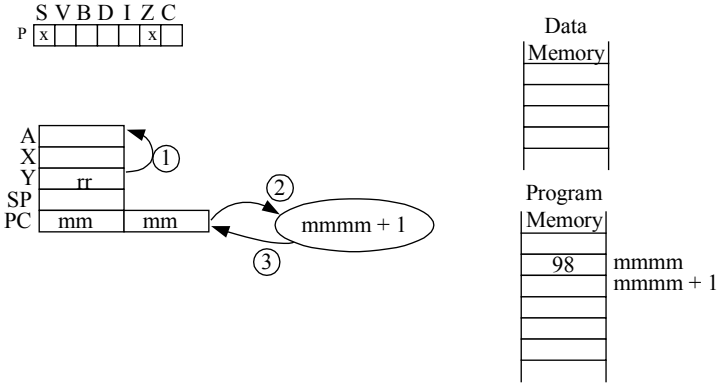
A←Y

Code:

Mnemonic	Object Code	# of Bytes	# of Cycles
TYA	98	1	2

Flags:

n	v	-	b	d	i	z	c
✓	-	-	-	-	-	✓	-



Example:

假設:　xx = AF_{16}

經過執行指令:　TYA

則累加器及 Y 暫存器的內容皆爲 AF_{16}，S 與 Z 旗號則改變如下：。

10101111

Sets S to 1　　Nonzero result sets Z to 0

範例

資料的總和 (8-bit Sum of Data)

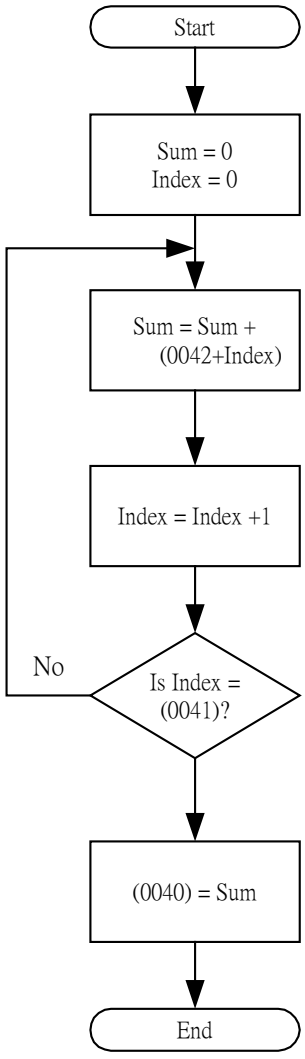
目的： 計算一序列數目之總和。序列的長度在記憶體位址 0041 中序列本身從記憶體位址 0042 開始，把總和存於記憶體位址 0040 內。假設總和爲一個 8 位元的數目，則不必考慮進位問題。

Sample Problem:

(0041) = 03
(0042) = 28
(0043) = 55
(0044) = 26

Result: (0040) = (0042) + (0043) +(0044)
= 28 + 55 + 26
= A3

因爲(0041) = 03 所以總和中包含三個單元。



注意:　(0042+INDEX)表示 0042 加上索引暫存器之值後該記憶體位址之內容。記住，在 6502 微處理機中，0042 爲一個 16 位元的位址，索引暫存器爲一個 8 位元的差距，而(0042+INDEX)爲一個 8 位元的資料。

解

```
        LDA   #0        ;SUM = 0
        TAX             ;INDEX = 0
SUMD    CLC             ;DO NOT INCLUDE CARRY
        ADC   $42, X    ;SUM = SUM + DATA
        INX             ;INCREMENT INDEX
        CPX   $41       ;HAVE ALL ELEMENTS BEEN
                          SUMMED ?
        BNE   SUMD          ;NO, CONTINUE SUMMATION
        STA   $40       ;YES, STORE SUM
        END
```

16 位元資料的和 (16-bit Sum of Data)

目的： 計算一序列數目之和。序列的個數在記憶體位址 0042 內，序列本身由記憶體位址 0043 開始，將求得之和存到記憶體位址 0040 及 0041 中。

Sample Problem:

(0042) = 03
(0043) = C8
(0044) = FA
(0045) = 96
Result = C8 + FA + 96 = 0258_{16}
(0040) = 58
(0041) = 02

解

```
CKSUM    =$40
LENG =$41
DATA =$42

         LDA   #0        ;SUM = ZERO
         TAX             ;INDEX = ZERO
         TAY             ;MSB'S OF SUM = ZERO
SUMD     CLC             ;DO NOT INCLUDE CARRY
         ADC   $43, X    ;SUM = SUM + DATA
         BCC   COUNT
         INY             ;ADD CARRY TO MSB'S OF SUM
COUNT    INX
         CPX   $42
         BNE   SUMD      ;CONTINUE UNTILL ALL ELEMENT
                           SUMMED
         STA   $40       ;STORE LSB'S OF SUM
         STY   $41       ;STORE LSB'S OF SUM
         END
```

尋找最大值 (Find Maximum Value)

目的: 在一段資料中尋找最大元素。此段資料的個數在記憶體位址 0041 中，且資料本身由記憶體位址 0042 開始，把最大元素儲存在記憶體位址 0040 內。假設此段資料內的數目皆爲無符號的 8 位元二進位數。

Sample Problem:

(0041) = 05
(0042) = 67
(0043) = 79
(0044) = 15
(0045) = E3
(0046) = 72

Result: (0040) = E3, since this is the largest of the five unsigned numbers

解

```
        LDX   #41       ;GET ELEMENT COUNT
        LDA   #6        ;MAXIMUM = ZERO (MINIMUM
                          POSSIBLE VALUE)
MAXM    CMP   $41, X    ;IS NEXT ELEMENT ABOVE
                          MAXIMUM ?
        BCS   NOCHG     ;NO, KEEP MAXIMUM
        LDA   $41, X    ;YES, REPLACE MAXIMUM WITH
                          ELEMENT
NOCHG   DEX
        BNE   MAXM      ;CONTINUE UNTIL ALL ELEMENTS
                          EXAMINED
        STA   $40       ;SAVE MAXIMUM
        END
```

資料的檢查和 (Checksum of Data)

目的: 計算一序列數目的檢查和。此序列的長度存於記憶體位址 0041 內，且序列本身由記憶體位址 0042 開始，檢查和是由對所有的數目做 “互斥或” (EOR)運算所得到的。把檢查和存於記憶體位址 0040 中。

Sample Problem:

(0041) = 03
(0042) = 28
(0043) = 55
(0044) = 26

Result: (0040) = (0042) ⊕ (0043) ⊕ (0044)
= 28 ⊕ 55 ⊕ 26
= 00101000
⊕01010101
01111101
⊕00100110
= 5B

解一

```
CKSUM   =$40
LENG    =$41
DATA    =$42

        LDA  #0          ;CHECKSUM = ZERO
        LDX  LENG        ;INDEX = LENGTH OF STRING
LOOP    EOR  DATA-1,X ;EXCLUSIVE-ORING WITH ELEMENT
        DEX              ;DECREMENT INDEX
        BNE  LOOP        ;UNITL ALL ELEMENTS ARE EOR
        STA  CKSUM   ;STORE RESULT
        END
```

解二

```
CKSUM   =$40
LENG    =$41
DATA    =$42

        LDA  #0          ;CHECKSUM = ZERO
        TAX              ;INDEX = LENGTH OF STRING
LOOP    EOR  DATA,X  ;EXCLUSIVE-ORING WITH ELEMENT
        INX              ;INCREMENT INDEX
        CPX  LENG        ;UNTIL ALL ELEMENTS ARE EOR
        BCC  LOOP
        STA  CKSUM   ;STORE RESULT
        END
```

零、正數與負數的個數 (Number of Zero, Positive and Negative Numbers)

目的: 決定一段數目中零、正數與負數的個數。此序列的個數在記憶體位址 0043 中，且序列本身由位址 0044 開始。把負數元素的個數存於記憶體位址 0040 中，零元素的個數存於記憶體位址 0041 中，正數元素的個數則存於記憶體位址 0042 內。

Sample Problem:

(0043) = 06
(0044) = 68
(0045) = F2
(0046) = 87
(0047) = 00
(0048) = 59
(0049) = 2A

Result: 2 negative, 1 zero, and 3 positive, so
(0040) = 02
(0041) = 01
(0042) = 03

解

```
NEC   =$40
ZER   =$41
POS   =$42
LENG  =$43
DATA  =$44
      LDA   #0          ;CLEAR COUNT TO START
      STA   NEG
      STA   ZER
      STA   POS
      TAX               ;INDEX = ZERO
CHECK INC   ZER         ;ASSUME NEXT NUMBER IS ZERO
      LDA   DATA, X     ;GET DATA
      TAY               ;STORE IN TEMPORARY REGISTER
      BEQ   NEXTE       ;YES, NEXT NUMBER
      DEC   ZER         ;NO, DECREMENT COUNT OF ZERO
      INC   POS         ;ASSUME THIS NUMBER IS POSITIVE
      TYA               ;RESTORE DATA
      BPL   NEXTE       ;YES, NEXT NUMBER
      DEC   POS         ;NO, DECREMENT COUNT OF POSITIVE
      INC   NEG         ;INCREMENT COUNT OF NEGATIVE
NEXTE INX               ;NEXT ELEMENT (BY INCREMENT
                          INDEX)
      CPX   LENG
      BNE   CHECK       ;UNTIL ALL NUMBERS ARE CHECKED
      END
```

找最小值 (Find Minimum)

目的: 在一段資料中找出最小的元素。此段資料的長度存於記憶體位址 0041 內，且此段資料本身由記憶體位址 0042 開始。把最小值儲存於記憶體位址 0040 內。假設此段資料內的數目皆爲 8 個位元的無符號二進位數。

Sample Problem:

(0041) = 05
(0042) = 67
(0043) = 79
(0044) = 15
(0045) = E3
(0046) = 70

Result: (0040) = 15, since this is the smallest of the five unsigned numbers.

解

```
        MIN   =$40
        LENG  =$41
        LIST  =$42

        LDX   LENG      ;GET LIST COUNT
        LDA   #$FF      ;MINIMUM =$FF (MAXIMUN POSSIBLE
                          VALUE)
SRCH    CMP   LIST-1,X  ;IS NEXT ELEMENT BELOW
                          MINIMUM?
        BCC   NOCHG     ;NO, KEEP MINIMUM
        BEQ   NOCHG
        LDA   LIST-1,X  ;YES, REPLACE MINMIUM WITH
                          ELEMENT
NOCHG   DEX             ;DECCREMENT COUNT
        BNE   SRCH      ;CONTINUE ALL ELEMENTS ARE
                          EXAMINED
        STA   MIN       ;STORE MINIMUM
        END
```

計算位元爲 1 的個數 (Count 1 Bits)

目的: 計算記憶體位址 0040 內位元爲 1 的個數有多少，將結果存於記憶體位址 0041 中。

Sample Problem:

(0040) = 3B = 00111011

Result: (0041) = 05

解

```
DATA    =$40
COUNT   =$41

        LDA   DATA        ;GET DATA
        LDY   #0          ;COUNT = ZERO
        LDX   #8          ;NUMBER OF SHIFTS = 8
LOOP    ASL   A           ;SHIFT LEFT ONE BIT
        BCC   NEXT
        INY
NEXT    DEX
        BNE   LOOP        ;CONTINUE UNTIL ALL BITS CHECKED
        STY   COUNT       ;STORE COUNT
        END
```

計算一串字元的長度(Length of a string of Characters)

目的: 求出一串 ASCII 碼字元的長度。這串字元所存放的開始位址是 0041，字串的末尾以一個印字頭回轉字元(“CR”即 $0D_{16}$)做爲結束記號。將此串字元的長度求出並存在 0040 這個位址。

Sample Problem:

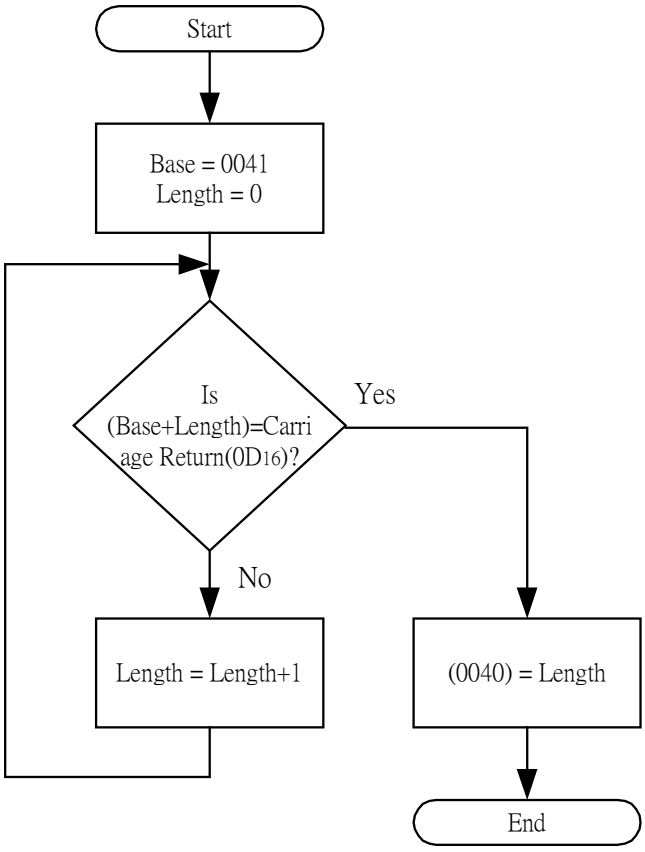
(0041) = 0D

Result: (0040) = 00 since the first character is a carriage return

(0041) = 52 ‘R’
(0042) = 41 ‘A’
(0043) = 54 ‘T’
(0044) = 48 ‘H’
(0045) = 45 ‘E’
(0046) = 52 ‘R’
(0047) = 0D CR

Result: (0040) = 06

流程圖:



解

```
        LDX   #0          ;STRING LENGTH = ZERO
        LDA   #$0D        ;GET ASCII CARRIAGE RETURN
                            TO COMPARE
CHKCR   CMP   $41, X      ;IS CHARACTER A CHRRIAGE
                            RETURN ?
        BEQ   DONE        ;YES, DONE
        INX               ;NO, ADD 1 TO STRING LENGTH
        JMP   CHKCR
DONE    STX   $40         ;SAVE STRING LENGTH
        BRK
```

就電腦而言，這個印字頭回轉字元 “CR”，只是另一個 ASCII 碼而已($0D_{16}$)，事實上，這個字元是用來指示輸出設備執行一個控制功能，而不只是印出一個不影響電腦的一個符號。

本程式中的比較指令，CMP 有如執行一個減法運算，並且設定 “旗號”，但仍將印字頭回轉字元留在累加器內，供以後之比較用。 “零旗號”受以下因素之影響

Z = 1　　若此字元是一個印字頭回轉字元

Z = 0　　若此字元不是一個印字頭回轉字元

INX 指令是將 1 加到做爲字元長度計數器的 X 暫存器內。LDX#0 則是在程式迴路開始之前，將字元長度計數器設定初值爲 0。記著:要在一個迴路裡使用一個變數之前，須將此數設定其起始值。

除非字元長度計數器一直遞減到 0 或是達到一個極大值，否則這個迴路是不會停止的;電腦會一直不斷的檢查此串字元，直至發現印字頭回轉字元。

試將此迴路限定一個最高循環次數，以避免所處理到是個沒有印字頭回轉字元的錯誤字元串。

注意:　將此程式的邏輯重新安排，並改變起始條件，將使程式縮短，並且減少其執行時間。使程式在檢查是否爲印字頭回轉字元之前將字元串長度加 1，則只需一個跳越指令. 程式如下

解

```
        LDX   #$FF        ;STRING LENGTH = -1
        LDA   #$0D        ;GET ASCII CARRIAGE RETURN
                            TO COMPARE
CHKCR   INX               ;ADD 1 TO STRING LENGTH
        CMP   $41, X      ;IS CHARACTER A CARRIAGE
                            RETURN ?
        BNE   CHKCR       ;NO, CHECK NEXT CHARACTER
        STX   $40         ;YES, SAVE STRING LENGTH
        END
```

此段新程式，不僅較短較快，且沒有絕對的目的位址，所以它能很容易的放置在記憶體的任何地方。

找出第一個非空白的字元 (Find First Non-blank Character)

目的: 在一串 ASCII 碼的字元中，找一個非空白的字元。此串字元開始位址是 0042。將此第一個非空白字元索引，存放到記憶體位址 0040 中。空白字元的 ASCII 碼是 20_{16}。

Sample Problem:

(0042) = 37 (ASCII 7)

Result: (0040) = 00 since memory location 0042 contains a non-blank character.

(0042) = 20 SP
(0043) = 20 SP
(0044) = 20 SP
(0045) = 46 'F'
(0046) = 20 SP

Result: (0040) = 03, since the three previous memory locations all contain blanks.

解

```
        LDX   #0          ;START WITH INDEX = ZERO
        LDA   #$20        ;GET ASCII SPACE FOR
                           COMPARISON
CHBLK   CMP   $42, X      ;IS CHARACTER AN ASCII
                           SPACE?
        BNE   DONE        ;NO, DONE
        INX               ;YES, EXAMINE NEXT
                           CHARACTER
        JMP   CHBLK
DONE    STX   $40         ;SAVE INDEX OF FIRST
                           NON-BLANK CHARACTER
        END
```

在一串字元尋找出空白是經常要做的工作。當空白只是用來增加易讀性或者是適合某種特殊形式時，空白是常常要被刪除的。尤其當您必須支付傳送量與儲存體需要量的費用時；顯然地，把開頭、結尾或額外的空白符號儲存或傳送是很浪費的然而，假如容許有空白符號存在，則資料或程式的鍵入會顯得簡單。

同樣的，如果更改起始條件，使程式的迴路控制段先前於處理段，則可減少程式的位元組數目，並且縮短迴路的執行時間，程式如下:

```
        LDX   #$FF          ;START WITH INDEX = -1
        LDA   #$20          ;GET ASCII SPACE FOR
                              COMPARISON
CHBLK   INX                 ;INCREMENT INDEX
        CMP   $42, X        ;IS CHRARCTER ANASCII SPACE?
        BEQ   CHBLK         ;YES, EXAMINE NEXT
                              CHARACTER
        STX   $40           ;NO, SAVE INDEX OF FIRST NON-
                              BLANK CHARACTER
        END
```

將前端的零替換成空白(Replace Leading Zeros with Blanks)

目的: 修改一串用 ASCII 表示的十進位字元,把所有前端的 0 替換成空白,此串字元開始位址是 0041,假設它全部是由 ASCII 碼的十進位數字所組.此串字元的長度在記憶體位址 0040 中

Sample Problem:

(0040) = 02
(0041) = 36 ASCII 6

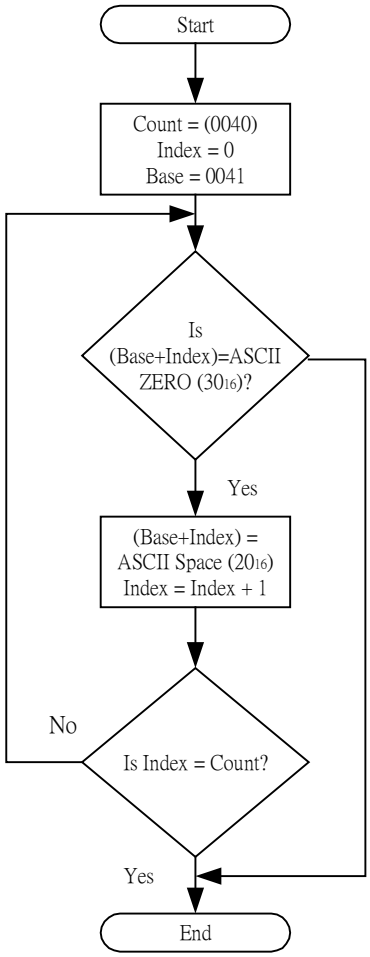
若前端數字不是 0，則此程式不會改變此串字元。

(0040) = 08
(0041) = 30 ASCII 0
(0042) = 30 ASCII 0
(0043) = 38 ASCII 8

Result: (0041) = 20 SP
(0042) = 20 SP

前端兩個 ASCII 碼“零”,已被 ASCII 碼的“空白”符號所替換。

流程圖:



解

```
        LDX   #0          ;INDEX = ZERO TO START
        LDY   #$20        ;GET ASCII SPACE FOR
                           REPLACEMENT
        LDA   #'0         ;GET ASCII ZERO FOR
                           COMPARISON
CHKZ    CMP   $41, X      ;IS LEADING DIGIT ZERO?
        BNE   DONE        ;NO, END REPLACEMENT
                           PROCESS
        STY   $41, X      ;YES, REPLACE ZERO WITH
                           BLANK
        INX
        CPX   $40
        BNE   CHKZ        ;EXAMING NEXT DIGIT IF ANY
DONE    END
```

NOTE: ‘0,在 0 字元前的單引號表示此運算元是一個 ASCII 碼

爲了在印出或顯示出時增進十進位字串之可讀性，在此之前您經常要去修改此十進位數字串。常見的修改工作括刪除前端的 0、將數字排成適當的長度、加上符號或辨識記號與四捨五入等。

此處迴路有兩個出口—一個是當處理機發現出非零的數字；另一個是當整串數字已仍部被檢查過了時。

STY $41, X 指令是將原來儲存著 ASCII 碼零的記憶體位址內擺進一個 ASCII 碼的空白字(20_{16})。注意：STY 只有有限數目的定址模式，所以暫存器 Y 沒有索引模式，沒有先索引，也沒有絕對索引，唯一的索引定址模式是以索引暫存器 X 的零頁模。

將每一個 ASCII 碼字元加上一個偶同位元 (Add Even Parity to ASCII Characters)

目的: 把偶同位元加到一串由 7 位元 ASCII 碼所組成字元串.此字元串的長度儲存在位址 0040 中,且此串字元之開始儲存位址是 0041.

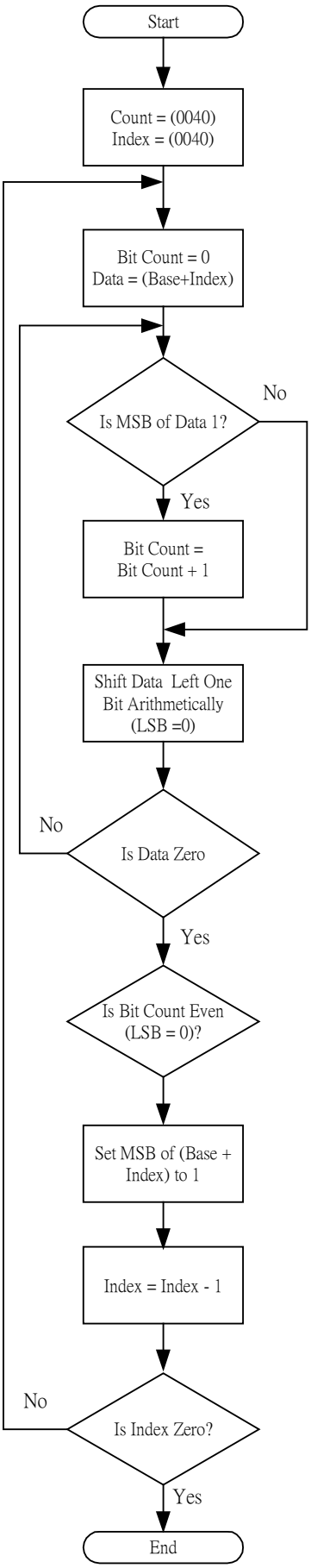
如果將每一個字元的最高效位元(MSB)設定爲 1 之後,能使該字元中 1 的個數偶數的話,就把該字元的最高效位元(MSB)設定爲 1,使此字元成爲偶同位.

Sample Problem:

(0040) = 06
(0041) = 31
(0042) = 32
(0043) = 33
(0044) = 34
(0045) = 35
(0046) = 36

Result: (0041) = B1
(0042) = B2
(0043) = 33
(0044) = B4
(0045) = 35
(0046) = 36

流程圖:


Start
Count = (0040)
Index = (0040)
Bit Count = 0
Data = (Base+Index)
Is MSB of Data 1?
No
Yes
Bit Count =
Bit Count + 1
Shift Data Left One
Bit Arithmetically
(LSB =0)
Is Data Zero
No
Yes
Is Bit Count Even
(LSB = 0)?
Set MSB of (Base +
Index) to 1
Index = Index - 1
Is Index Zero?
No
Yes
End

解:

```
            LDX   #40           ;INDEX = MAXIMUN COUNT
GTDATA      LDY   #0            ;BIT COUNT = ZERO FOR DATA
            LDA   $40, X        ;GET DATA FROM BLOCK
CHBIT       BPL   CHKZ                ;IS NEXT DATA BIT 1?
            INY                 ;YES, ADD 1 T0 BIT COUNT
CHKZ        ASL   A             ;EXAMINE NEXT BIT POSITION
            BNE   CHBIT         ;UNLESS ALL BITS ARE ZEROS
            TYA
            LSR   A             ;DID DATA HAVE ENEV NUMBER
                                  OF '1' BITS?
            BCC   MEXTE
            LDA   $40, X        ;NO, SET PARITY BIT
            ORA   #%10000000
            STA   $40, X
NEXTE       DEX
            BNE   GTDATA        ;CONTINUE TFROUGH DATA
                                  BLOCK
            END
```

爲了在一個有雜訊干擾的傳送線上提供簡單的偵誤能力，通常一個 ASCII 碼字元在被傳送之前，都加上同位元，同位元具有偵查出所有單一位元，也就是說：您只能偵查資料中的同位元而通知資料有錯誤發生，但是您無法指出是那一個位元接收有誤。接收器能夠做的只是要求再傳送一次而已。

計算同位元的步驟是：去計算資料字中 1 位元的數目，若爲奇數，則將此資料的字最高效位元設定爲 1，以指示此字爲偶同位元。

當一個數字被位移時，ASL 指令將最低效位元清成 0，因此一連串的執行 ASL 指令，無論原先資料爲何，將得一個爲零的結果。此程式中位元的計算部份，不僅不需要使用計數器，而且當所有剩下的位元全爲 0 時，也會停止檢查此資料。這種方式在很多情況下可以節省執行時間。

將此資料最高效位元設定爲 1 或 0 的方式，是用一個最高效位元爲 1 而其他位元爲 0 圖形與此資料做邏輯 “或”運算。不論原來值爲何，把一個位元與 “1”位元做邏輯 “或”運算，則結果必爲 1，然而把一個位元與 “0”做邏輯 “或”運算，則不影響此位元之原來值。

字形的比對(Pattern Match)

目的: 比較兩串 ASCII 碼的字元是否相同，若相同則將記憶體位址 0040 清爲 0，否則將其設定爲 FF_{16}(全爲 1)。字元串的長度放記憶體位址 0041 中。一個字元串由記憶體 0042 開始，另一字元串由記憶體位址 0052 開始。

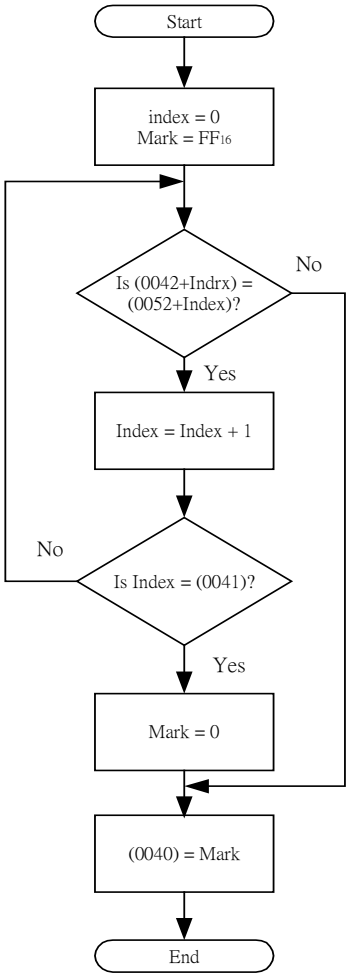
Sample Problem:

(0041) = 03
(0042) = 43 'C'
(0043) = 41 'A'
(0044) = 54 'T'
(0052) = 43 'C'
(0053) = 41 'A'
(0054) = 54 'T'

Result: (0040) = 00
(0041) = 03
(0042) = 52 'R'
(0043) = 41 'A'
(0044) = 54 'T'
(0052) = 43 'C'
(0053) = 41 'A'
(0054) = 54 'T'

Result: (0040) = FF, since the first characters in the strings differ

流程圖:



注意: 當 CPU 發現有一個不同時，則此比對處理結束一字元串的剩餘部份不需要再檢查下去了!

解:

```
        LDX   #0          ;START WITH FIRST ELEMENT IN
                            STRINGS
        LDY   #$FF        ;MARKER FOR NO MATCH
CHCAR   LDA   $42, X      ;GET CHARACTER FROM STRING 1
        CMP   $52, X      ;IS THEERE A MATCH WITH
                            STRING 2?
        BNE   DONE        ;NO, DONE
        INX
        CPX   $41
        BNE   CHCAR       ;CHECK NEXT PAIR IF ANY LEFT
        LDY   #0          ;IF NONE LEFT, MARK MATCH
DONE    STY   $40         ;SAVE MATCH MARKER
        END
```

此對 ASCII 碼的字元串是重要的工作。例如名稱或命令的辨識、組譯程式或編譯程式中變數或運算碼的辨別、檔案的尋找、及許多其他事務等。

索引暫存器 X 是利用來取用此兩串字元的，是有其基底位址不同。此種法允許字元串孝放於記憶體之任何地方。然而若字元串不是存放在第零頁，那麼我們必須使用絕對索引定址法。若有兩個不同的基底位址存放零頁的某處，也可使用後索引模式。

求出一串電傳打字機訊息的長度 (Length of a Teletypewriter Message)

目的: 所有字元是由 7 個位元的 ASCII 碼與最高效位元爲 0 所組成。訊息由記憶體位址 0041 開始。訊息本身是以一個 ASCII 碼 STX(02_{16})字元開始，並以 ETX(03_{16})字元結束。將此串訊息的長度存於記憶體位址 0040 中。

Sample Problem:

(0041) = 40
(0042) = 02 STX
(0043) = 47 'G'
(0044) = 4F '0'
(0045) = 03 ETX

Result: (0040) = 02, since there are two characters between the STX in location 0042 and ETX in location 0045.

解:

```
LENG =$40

DATA =$41

        LDA   #$02          ;GET ASCII STX CHARACTER
        LDX   #$FF          ;INDEX = -1
SRSTX   INX                 ;INCREMENT INDEX
        CMP   DATA, X       ;IS A STX CHARACTER
        BNE   SRSTX         ;NO, NEXT CHARACTER
        LDA   #$03          ;GET ASCII ETX CHARACTER
        LDY   #$FF          ;STRING LENGTH = -1
COUNT   INX                 ;INCREMENT INDEX
        INY                 ;ADD 1 TO STRING LENGTH
        CMP   DATA, X       ;IS A ETX CHARACTER
        BNE   COUNT         ;NO, NEXT CHARACTER
        STY   LENG          ;YES, SAVE STRING LENGTH
        END
```

出最後一個非空白的字元 (Find Last Non-Blank Character)

目的: 在一串 ASCII 碼的字元中，找出最後一個非空白的字元。此串字元由記憶體位址 0042 開始，且以一個印字頭回轉字元作爲結束。將最後一個非空白字元的索引放入記憶體位址 0040 中。

Sample Problem:

(0042) = 37 ASCII 7
(0043) = 0D CR
Result: (0040) = 00, since the last non-blank character is in memory location 0042
(0042) = 41 'A'
(0043) = 20 SP
(0044) = 48 'H'
(0045) = 41 'A'
(0046) = 54 'T'
(0047) = 20 SP
(0048) = 20 SP
(0049) = 0D CR
Result: (0040) = 04

解:

```
IDX   =$40
STRG  =$42

      LDA   #$0D        ;GET ASCII CARRIAGE RETURN
      LDX   #$FF        ;INDEX = -1
LOOP  INX               ;INCREMENT INDEX
      CMP   STRG, X     ;IS A CHARACTER RETURN
      BNE   LOOP        ;NO, NEXT CHARACTER
      LDA   #$20        ;GET ASCII SPACE
NEXT  DEX               ;DECREMENT INDEX
      CMP   STRG, X     ;IS A SPACE?
      BEQ   NEXT        ;YES, NEXT CHARACTER
      STX   IDX         ;NO, SAVE RESULT
      END
```

將十進位數字串的小數部份切除成整數形態 (Truncate Decimal String to Integer Form)

目的: 修改一串 ASCII 碼的十進位數字，將其小數點右邊的所有數字全部替換成 ASCII 碼的空白字元(20_{16})。此串數字之開始位址是 0041，同時假設此串數字全部由 ASCII 碼的十進位數字所組成，並且可能有一個小數點($2E_{16}$)。此串數字的長度存在記憶體位址 0040 中。若此串數字沒有小數點，則假設此十進位之小數點隱含在最右邊。

Sample Problem:

(0040) = 04
(0041) = 37 ASCII 7
(0042) = 2E ASCII
(0043) = 38 ASCII 8
(0044) = 31 ASCII 1

Result:
(0041) = 37 ASCII 7
(0042) = 2E ASCII
(0043) = 20 SP
(0044) = 20 SP

(0040) = 03
(0041) = 26 ASCII 6
(0042) = 37 ASCII 7
(0043) = 31 ASCII 1

Result: Unchanged as number is assumed to be 671

解:

```
LENG =$40
DATA =$41

        LDA   #$2E          ;GET ASCII DECIMAL POINT
        LDX   #$FF          ;INDEX = -1
NEXT    INX                 ;INCREMENT INDEX
        CMP   DATA, X       ;IS A DECIMAL POINT?
        BEQ   OK            ;YES, GO TO PHASE II
        CPX   LENG          ;CHECK NEXT CHARACTER IF
                              ANY LEFT
        BNE   NEXT
        BEQ   DONE          ;IF NONE LEFT, UNCHANGE
LOOP    LDA   #$20          ;GET ASCII SPACE
        STA   DATA, X       ;REPLACE WITH SPACE
OK      INX                 ;INCREMENT INDEX
        CPX   LENG          ;IS ANY LEFT?
        BCC   LOOP          ;YES, NEXT CHARACTER
DONE     END
```

檢查 ASCII 碼字元的偶同位元 (Check Even Parity in ASCII Characters)

目的: 檢查一串 ASCII 字元的偶同位。此串字元的長度存放在記憶位址 0041 中，且字元串本身的開始位址是 0042。如困此串字元中的每一字元都有正確的偶同位元，那麼將記憶體位址 0040 之內容清爲 0，否則將記憶體位址 0040 之內容設定爲 FF_{16}。

Sample Problem:

(0041) = 03
(0042) = B1
(0043) = B2
(0044) = 33

Result: (0040) = 00, since all the characters have even parity

(0041) = 03
(0042) = B1
(0043) = B6
(0044) = 33

Result: (0040) = FF, since the characters in memory location 0043 does not have even parity.

解:

```
            MARK    =$40
            LENG =$41
            DATA =$42
            LDX   LENG          ;GET MAXIMUN COUNT
            LDA   #$FF          ;MARKER FOR NO MATCH
            STA   MARK
NEXT        LDY   #0            ;BIT COUNT = ZERO
            LDA   DATA-1, X
LOOP        BPL   SHFT          ;IS NEXT DATA BIT 1?
            INY                 ;YES, ADD 1 TO BIT COUNT
SHFT        ASL   A             ;EXAMINE NEXT BIT?
            BNE   LOOP          ;UNTIL ALL BITS ARE ZERO
            TYA                 ;RESTORE COUNT
            LSR   A             ;DATA HAVE EVEN NUMBER OF '1'
                                  BITS?
            BCS   DONE          ;NO, ERROR AND EXIT
            DEX
            BNE   NEXT          ;NEXT ELEMENT IF ANY LEFT
            INC   MARK          ;IF NONE LEFT, SET MARKER TO
                                  ZERO
DONE        END
```

字串的比較 (String Comparison)

目的: 比較兩串 ASCII 碼字元，看何者爲大。字串的長度在記憶體位址 0041 中。一個字串由記憶體位址 0042 開始，而另一字串則由記憶體位址 0052 開始。如果在記憶體位址 0042 開始的字串大於或等於另一字串，則把記憶體位址 0040 清除爲 0；否則把記憶體位址 0040 設定爲 FF_{16}。

Sample Problem:

(0041) = 03
(0042) = 43 'C'
(0043) = 41 'A'
(0044) = 54 'T'
(0052) = 42 'B'
(0053) = 41 'A'
(0054) = 54 'T'

Result: (0040) = 00, since CAT is 'large' than BAT

(0041) = 03
(0042) = 43 'C'
(0043) = 41 'A'
(0044) = 54 'T'
(0052) = 43 'C'
(0053) = 41 'A'
(0054) = 54 'T'

Result: (0040) = 00, since the two strings are equal

(0041) = 03
(0042) = 43 'C'
(0043) = 41 'A'
(0044) = 54 'T'
(0052) = 43 'C'
(0053) = 55 'U'
(0054) = 54 'T'

Result: (0040) = FF, since CUT is 'large' than CAT

解:

```
            MARK    =$40
            LENG    =$41
            STR1    =$42
            STR2    =$52

            LDY  #$FF       ;MARKER FOR LESS THAN ($FF)
            LDX  #0         ;START WITH FIRST ELEMENT IN
                              SRTINGS
LOOP        LDA  SRT1, X    ;GET CHARCTER FROM STRING 1

            CMP  STR2, X    ;IS EQUAL TO STRING 2
            BEQ  NEXT       ;YES, NEXT ELEMENT
            BCS  GT         ;STRING 1 GREATER THAN
                              STRING 2
            BCC  LT         ;STRING 1 LESS THAN STRING 2
NEXT        INX             ;INCREMENT INDEX
            CPX  LENG
            BCC  LOOP       ;NEXT ELEMENT IF ANY LEFT
GT          INY             ;MARKER = ZERO
LT          STY  MARK       ;SET MARKTER
            END
```

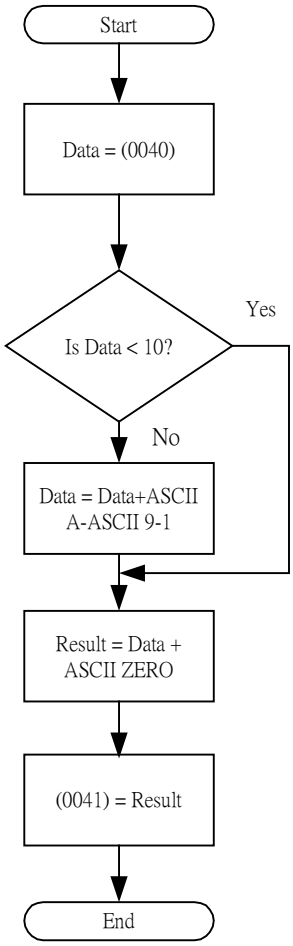
十六進位轉換成 ASCII 碼(Hex to ASCII)

目的： 把記憶體位址 0040 中的內容轉換成以 ASCII 碼表示的字元。記憶體位址內僅儲存內僅儲存一個 16 進位數字，把這個轉換後的 ASCII 碼文字儲存到記憶體位址 0041 內。

Sample Problem:

(0040) = 0C
Result = (0041) = 43 'C'
(0040) = 06
Result = (0041) = 36 '6

流程圖:



解:

```
        LDA   $40          ;GET DATA
        CMP   #10          ;IS DATA LESS THEN 10?
        BCC   ASCZ
        ADC   # ‘A-‘9- 2 ;NO, ADD OFFSET FOR LETTERS
                             (CARRY=1)
ASCZ    ADC   # ‘0         ;ADD OFFSET FOR ASCII
        STA   #41          ;STORE ASCII DIGIT
        END
```

此程式基本觀念是:把所有 16 進位數加上 ASCII 碼的零(30_{16}).雖然這加法正確地把十進位數轉換成 ASCII 碼.,但是在 ASCII 碼 9 和 ASCII 碼 A 間出現數值不連續現象,因此我們必須加以考慮.此不連續數值必須加至非十進位數字 A.B.C.D.E 和 F 中.程式中第一個 ADC 指令是累加器內容加上差距(’A - ‘ 9 - 2) 這個指令解決了上面的問題.

注意:ADC 指令總會加上進位位元.在 BCC 指令之後,進位位元爲 1,所以我們將加算因子減一.至於第二個 ADC 指令所加入進位位元爲 0,原因是若程式 CMP 指令之後發生分支或者累加器的內容爲一個正確的 16 進位 10 到 15 之數字,則進位位元被清爲 0.因此在任何合理的情況下,我們不須爲進位而擔心.

此常式可適用於各種程式.例如監督程式必須將以 16 進位方式儲存的記憶體內的容轉換爲 ASCII 碼以顯示一個 ASCII 列表機陰極射線顯示器上.

下列程式是另一種更快的轉換方法,不須任何條件式的跳越指令:

```
        SED                ;MAKE ADDITIONS DECIMAL
        CLC                ;CLEAR CARRY TO START
        LAD   #40          ;GET HEXADECIMAL DIGIT
        ADC   #$90         ;DEVELOP EXTRA 6 AND CARRY
        ADC   #$40         ;ADD IN CARRY, ASCII OFFSET
        STA   #41          ;STORE ASCII DIGIT
        CLD                ;CLEAR DECIMAL MODE BEFORE
                             ENDING
        END
```

ASCII 碼轉換成十進位 (ASCII to Decimal)

目的: 把記憶體位址 0040 中的 ASCII 碼換成十進位數,然後把結果存放到記憶體位址 0041 內。如果記憶體位址 0040 中的內容不是以 ASCII 型式來表示的十進位數字,則把記憶體位址 0041 中的內容設定爲 FF_{16}。

Sample Problem:

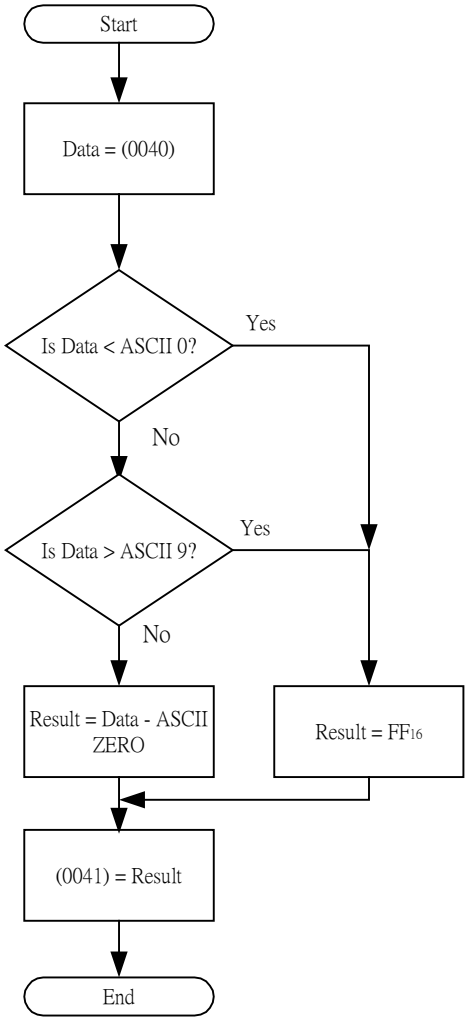
(0040) = 37 (ASCII 7)

Result: (0041) = 07

(0040) = 55 (an invalid code, it is not an ASCII decimal digit)

Result: (0041) = FF

流程圖:



解:

```
        LDX   #$FF        ;GET ERROR MESSACE
        LDA   $40         ;GET DATA
        SEC               ;IGNORE CARRY IN
                            SUBTRACTION
        SBC   #'0         ;IS DATA BELOW ASCII ZERO?
        BCC   DONE        ;YES, NOT A DIGIT
        CMP   #10         ;IS DATA ABOVE ASCII NINE?
        BCS   DONE        ;YES, NOT A DIGIT
        TAX               ;SAVE DIGIT IF VALID
DONE    STX   $41         ;SAVE DIGIT OR ERROR MARKER
        END
```

這個程式處理 ASCII 碼字元就如處理一般普通的數目一樣.

注意:　十進位數字和英文字母各別形成了一組連續的碼,所以一串英文字母能夠依它們的 ASCII 碼按字母的數值由小而大順序排列從任何 ASCII 的十進位數字減去 ASCII 碼 0，即爲此數字的 BCD 表示型式。因 SBC 指令所執行的是(A)=(A)-(M)-(1-C)，其中 M 代表記憶體位址，C 代表進位旗號值，所以在執行減法指令之前必須先將 C 旗號設定爲 1，才能得到所希望的結果，而另一方面，比較指令(CMP)於其含的減法中不包含進位旗號。當十進位是從 ASCII 碼裝置如電傳打字機或陰極射線管終端機等輸入的。則必須 ASCII 碼轉換爲十進位碼，此程式基本觀今是:先決此字元是否介於 ASCII 0 到 ASCII 9 之間如果是則此字元是一個 ASCII 十進位字,因這些數字形一個數字。

注意:　程式中以一個實際的減法(SBC#'0)取代比較指令，因將 ASCII 字元轉換爲十進位必執行減法運算;而另一比較運算是以一隱含減法(CMP#10)來完成，因爲如原來數目爲一正確的 ASCII 十進位數的話,則最後結果已存於累加器中了。

BCD 碼轉換成二進位 (BCD to Binary)

目的: 把記憶體位址 0040,0041 中的 BCD 碼換成 2 進位數,然後把結果存放到記憶體位址 0042 內。

Sample Problem:

(0040) = 02

(0041) = 09

Result = (0042) = $1D_{16}=29_{10}$

解

```
LDA   $40          ;GET MOST SIGNIFICANT
                     DIGIT(MSD)
ASL   A            ;MSD TIMES TWO
STA   $42          ;SAVE MSD TIMES TWO
ASL   A            ;MSD TIMES FOUR
ASL   A            ;MSD TIMES EIGHT
CLC
ADC   $42          ;MSD TIMES TEN (NO CARRY)
ADC   $41          ;ADD LEAST SIGNIFICANT
                     DIGIT(LSD)
STA   $42          ;STORE BINARY EQUIVALENT
END
```

二進位碼轉換成 ASCII 字串 (Convert Binary Number to ASCII String)

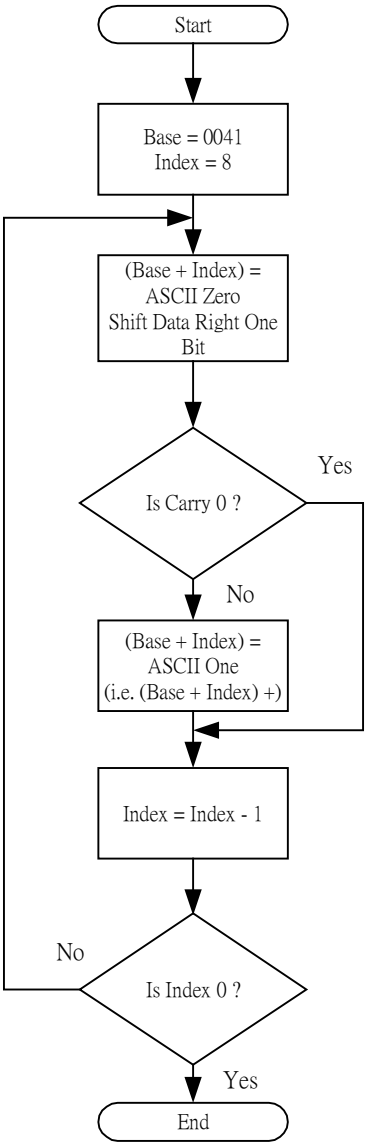
目的: 把記憶體位址 0041 中的 8 個 2 進位數轉換成 8 個 ASCII(0-9), 然後結果放置在記憶體位址 0042 到 0049 之中.

Sample Problem:

(0040) = D2 = 11010010

Result = (0042) = 31 ASCII 1
(0043) = 31 ASCII 1
(0044) = 30 ASCII 0
(0045) = 31 ASCII 1
(0046) = 30 ASCII 0
(0047) = 30 ASCII 0
(0048) = 31 ASCII 1
(0049) = 30 ASCII 0

流程圖:



解:

```
        LDA   $41          ;GET DATA
        LDX   #8           ;NUMBER OF BITS = 8
        LDY   # '0         ;GET ASCII ZERO TO STORE IN
                             STRING
CONV    STY   $41, X       ;STORE ASCII ZERO IN STRING
        LSR   A            ;IS NEXT BIT OF DATA ZERO?
        BCC   COUNT
        INC   $41, X       ;NO, MAKE STRING ELEMENT
                             ASCII ONE
COUNT   DEX                ;COUNT BITS
        BNE   CONV
        END
```

INC 指令通常被用來直接地將記憶體的內容增加 1，在此不須任何設(explicit)指令用來把資料從記憶體中取出，然後把結果存回記憶體中;而且也不會破壞任何的使用者暫存器。然而中央處理單必須實際地把資料從記憶體中取出，存放在暫時性的暫存器中，將資料加 1，然後把結果儲存回記憶體中，所有的資料理實際上都發生在 CPU 內部。

注意: INX 指令與如 INC $41, X 此類指令之間的不同處，INX 指令是把 X 暫存器的內容加 1;而 INC $41, X 指令將被索引的記憶體內容加 1。

當一數目以二位型式列印在 ASCII 的裝置上時，二進位對 ASCII 的轉換是必要的;而轉換的動作只要簡單地加上 ASCII 0(即 30_{16}即可。

十進位加法(Decimal Addition)

目的: 將兩個多字組的十進位(BCD)數目相加，此數目的長度在記憶體位址 0040 中，此兩數目分別由憶體位址 0041 與 0051 開始，把加總後的結果取代原來從記憶體位址 0041 開始數值。

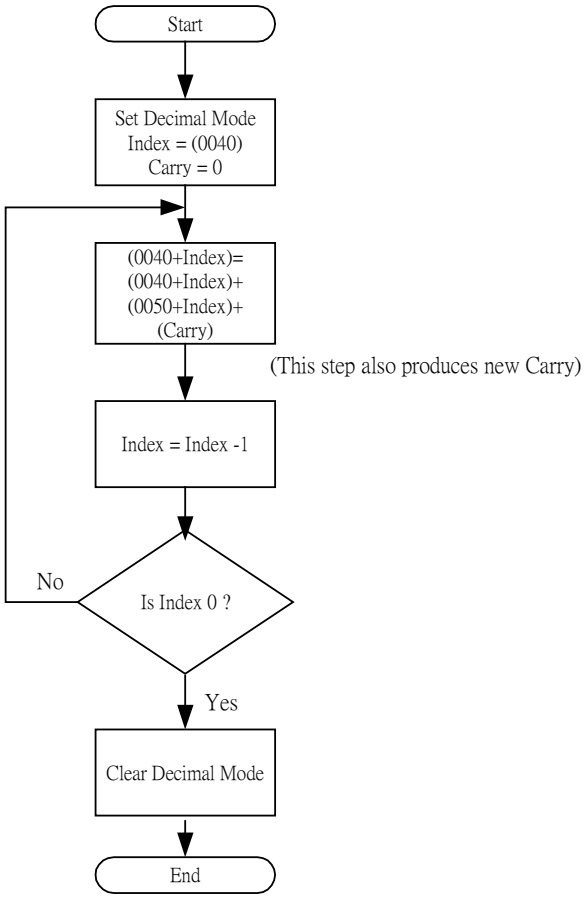
Sample Problem:

(0040) = 04
(0041) = 36
(0042) = 70
(0043) = 19
(0044) = 85
(0051) = 12
(0052) = 66
(0053) = 34
(0054) = 59

Result = (0041) = 49
(0042) = 36
(0043) = 54
(0044) = 44

that is:
```
  36701985
+12663459
  49365444
```

流程圖:



解:

```
        SED             ;MAKE ALL ARITHMETIC
                          DECIMAL
        LDX   $40       ;INDEX = LENGTH OF STRINGS
        CLC             ;CLEAR CARRY TO START
ADDW    LDA   $40, X    ;GET TWO DIGITS FROM STRING 1
        ADC   $50, X    ;ADD TWO DOGOTS FRPM
                          STRING 2
        STA   $40, X    ;STORE RESULT IN STRING 1
        DEX
        BNE   ADDW      ;CONTINUE UNTIL ALL DIGITS
                          ADDED
        CLD             ;RETURN TO BINARY MODE
        END
```

6502 的十進位模式運算，自動地處理下列二進位加法與十進位加法不同的情況:

1) 兩數字的和介於 10 與 15 之間
   此情況下,總和必須加 6 才能得到正確的結果
2) 兩數位的總和大於或等於 16.在這種情況之下，此結困是一個正確的 BCD 數字但是其值比實際的值少了 6。

當十位模式旗號被置定爲 1 之後，所有的算術運算都採十進位的型式,包括加法與減法，且不論是引用那種定址模式。
但即使十進位模式的旗號已被置定爲 1，遞增與遞減指令所產生的結果仍是二進位的型式。

十進位模式的旗號被置定爲 1 時，執行加法與減法運算之後，符號位元已不具任何意義了。符號位元只反應二進位運算的結果。而非十進位運算。

八位元二進位除法 (8-Bit Binary Division)

目的: 把在記憶體位址 0040 與 0041 中的 16 位元無符號數目除以在記憶體位址 0042 中的 8 位元無符號數目，兩數目已被標準化而滿足下列兩項規定:

1) 除數與被除數的最高效位元皆爲 0

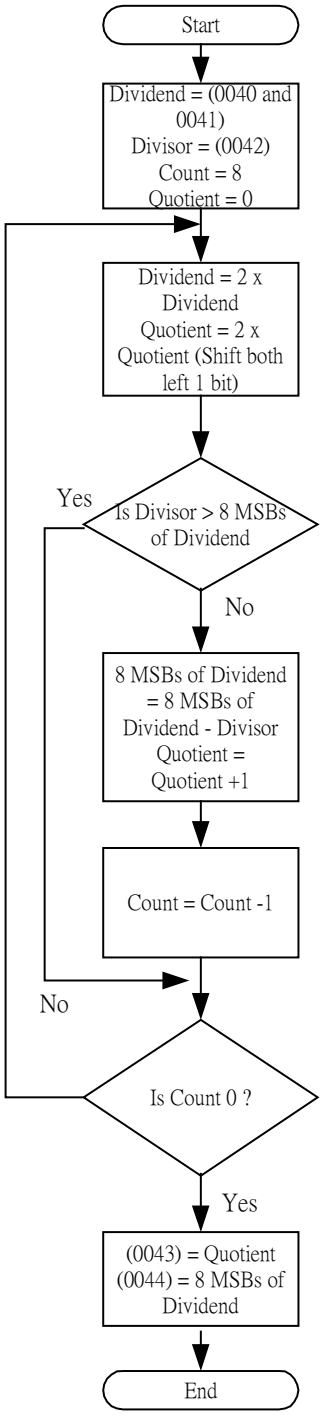
2) 在記憶體位址 0042 中的數目大於記憶體位址 0041 數目

Sample Problem:

(0040) = 40 (0040_{16}=64_{10} decimal)
(0041) = 00
(0042) = 08
Result = (0043) = 08
(0044) = 00
i. e. 64/8 = 8

(0040) = 6D ($326D_{16}$=$12{,}909_{10}$ decimal)
(0041) = 32
(0042) = 47 (47_{16}=71 decimal)
Result = (0043) = B5 ($B5_{16}$=181 decimal)
(0044) = 3A ($3A_{16}$=58_{10} decimal)

流程圖:


Start
Dividend = (0040 and 0041)
Divisor = (0042)
Count = 8
Quotient = 0
Dividend = 2 x Dividend
Quotient = 2 x Quotient (Shift both left 1 bit)
Yes
Is Divisor > 8 MSBs of Dividend
No
8 MSBs of Dividend = 8 MSBs of Dividend - Divisor
Quotient = Quotient +1
Count = Count -1
No
Is Count 0 ?
Yes
(0043) = Quotient
(0044) = 8 MSBs of Dividend
End

解:

```
        LDX   #08          ;NUMBER OF BITS IN DIVISOR = 8
        LDA   $40          ;START WITH LSB'S OF
                            DIVIDEND
        STA   $43
        LDA   $41          ;GET MSB'S OF DIVIDEND
DIVID   ASL   $43          ;SHIFT DIVIDEND, QUQTIENT
                            LEFT 1BIT
        ROL   A
        CMP   $42          ;CAN DIVISOR BE SUBTRACTED?
        BCC   CHCNT        ;NO, GO TO NEXT STEP
        SBC   $42          ;YES, SUBTRACT DIVISOR
                            (CARRY = 1)
        INC   $43          ;AND INCREMENT QUOTIENT BY 1
CHCNT   DEX                ;LOOP UNTIL ALL 8 BITS
                            HANDLED
        BNE   DIVID
        STA   $44          ;STORE REMAINDER
        END
```

除法常用在計算器、終端機、通訊核錯、控制演算法則與許多其他的應用上。

此除法演算法則在一個使用 1MHZ 振盪頻率的 6502 上，要花費 150 至 230 微秒,精確的時間則依商數 1 位元的數目而定，其他的演算法則可能略有減少平均執行時間，但是軟體的除法運算典型的執行時間約 200 微秒(uS)。

因沒有簡單的方法去執行 16 位元的減法比較運算，所以需要 8 位元的減法運算。

記憶體位址 0043 中與累加器中存有被除數與商數，被除數被左移時，商數替代了在記憶體位址 0043 中的被除數，故不必擔心 SBC 指令中進位元，它必爲 1 否則 BCC 指令將引起一分支動作，若進位元爲 1 並不影響 SBC 指令的結果,因爲此進位位元是個反借位。