T4x6N User Manual

May.27.2014

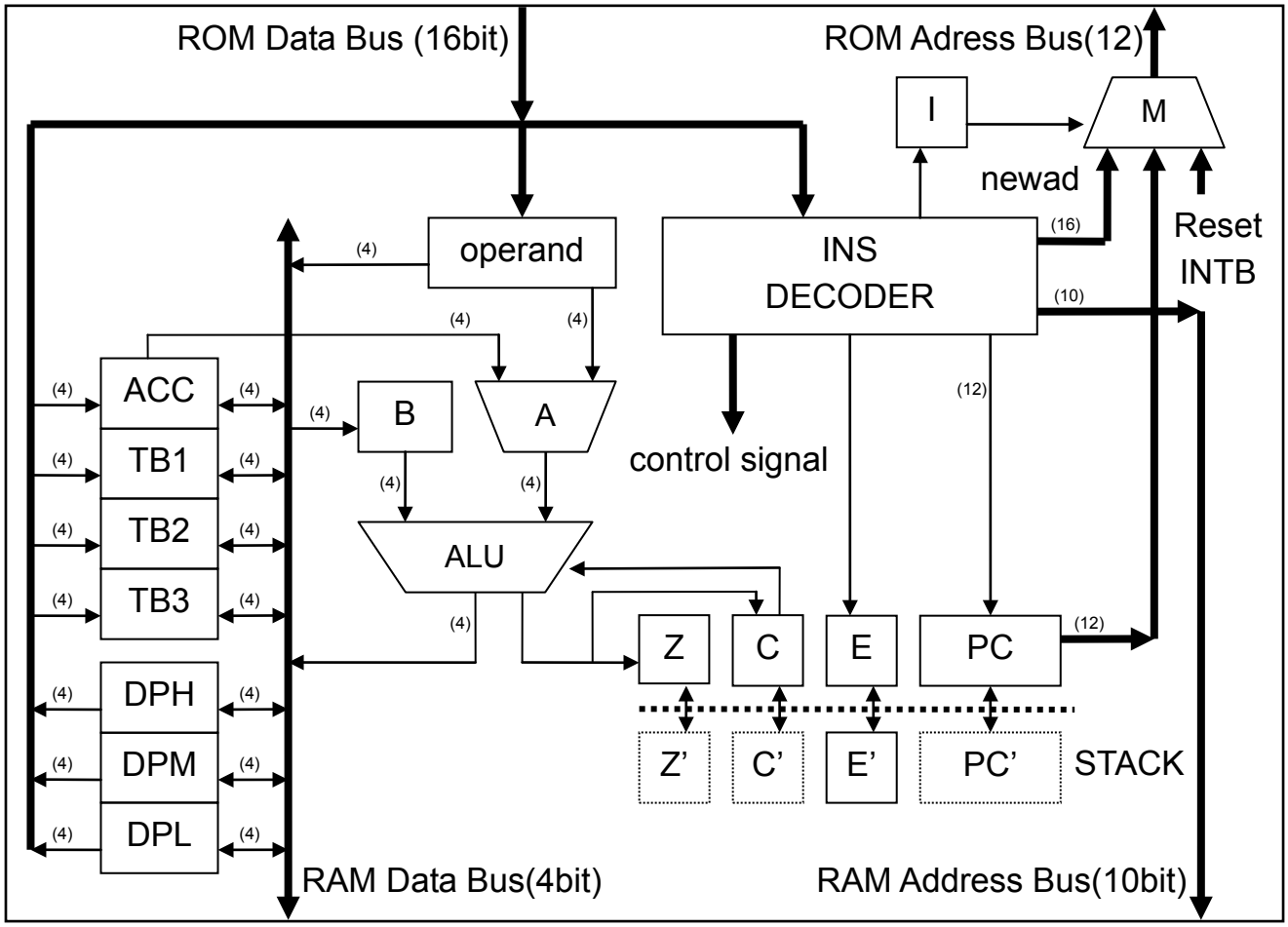
目　　錄

∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞

7. 副程式的介紹

∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞

1. T4x6N CPU 的硬體架構



T4x6N CPU 是一個架構簡單,功能完全之強而有力的 4 位元 CPU,內部結構如上圖所示,包含有程式計數器(PC),指令解碼器(IR),算數邏輯運算單元(ALU),堆棧(Stack),旗標及 7 個工作暫存器(ACC,TB1,TB2,TB3,DPL,DPM,DPH)等.

註:T4x6N 中的 x 為正整數用來表示堆棧(stack)的層數

程式計數器--是作為程式執行順序的指標,每一條指令執行完,程式計數器會自動遞增,但遇到跳躍,呼叫副程式或中斷產生時會改變程式計數器的內容,使執行程序的順序更改.

堆棧--當呼叫副程式或中斷產生時CPU會跳至副程式執行,此時程式計數器會被更改,但是執行完副程式後仍要返回主程式原來的地方,堆棧就暫存了返回的位址及自動增量(E)進位(C)及零(Z)旗標,返回時CPU會從堆棧中將位址及旗標取回,使程式能夠依照既定流程繼續執行程式.

指令解碼器--當程式計數器送出位址到程式記憶體,從程式記憶體讀取的指令進入指令解碼器,指令解碼器依指令解碼後,送出控制信號控制整個CPU動作.

算數邏輯運算單元--則會將A及B二暫存器內的值做加減算術運算或做AND,OR,XOR等邏輯運算,提供給CPU需要的所有運算,A及B二暫存器僅在運算時暫存之用,指令是無法直接對此二暫存器Access的.

T4x6N CPU 將程式記憶體(ROM)與可讀寫記憶體(RAM)的位址線及資料線分開處理,配合指令的規劃使程式記憶體最大可達 4K Word 供存放程式用,程式是由一群指令所組成,每一條指令為 16 位元寬,由運算碼(OP Code)及運算元(OPR)所組成;可讀寫記憶體最大可達到 1K Nibble,

工作暫存器佔用了前 8 個位址,不論是可讀寫記憶體或是工作暫存器都是 4 位元寬,在指令的執行中存放著處理的資料及控制資訊用,
T4x6N 的指令執行時間,除 RTB 指令為 4 個系統時脈外,其餘的每一條指令都在 2 個系統時脈執行完畢,此點說明了 T4x6N 是一個效率極佳的 CPU

2. T4x6N CPU 的工作暫存器

在 T4x6N CPU 中 RAM Mapping 的前八個位址$000~$007 已被 CPU 工作暫存器使用,所以在設計電路時的 I/O Port, Control Register 或 RAM 的 Memory Mapping 都是從 $008 開始使用,其前八個位址(CPU 工作暫存器)的用途分別說明如下:

$000 : Index == 虛擬指標暫存器

這不是一個實際的暫存器,只用來當做間接定址的指標,當對此位址讀寫或運算或讀取 ROM TABLE 時,CPU 實際是以 DPH,DPM,DPL 所組成的 12 位元為地址線,讀取 ROM 的資料或是對 RAM 做運算或讀寫資料.

$001 : Acc == 累積器

除了當成一般暫存器使用外;當需要將暫存器或 RAM 的資料轉換到其他 RAM 或暫存器時累積器是一個重要的中繼站,先使用 LDA 指令讀取暫存器或 RAM 的資料暫存在累積器,再用 STX 指令存到想要存放的暫存器或 RAM 中,來達成暫存器或 RAM 的資料轉換;此外在做邏輯或算數運算時,若是選擇結果放到 Acc,則運算完的結果將會存到累積器中;在使用直接指令模式運算,累積器是二個運算來源之一;若是使用 RTB 指令讀取 ROM TABLE 時,讀取到的資料是 16 位元,其中最低的 4 位元(D3~D0)也會暫存在累積器中.

$002 : TB1 == 讀表暫存器 1

此暫存器提供我們在使用 RTB 指令讀取 ROM TABLE 時,暫時存放讀取 16 位元資料中的次低 4 位元(D7~D4)用;在不使用讀表功能時,此暫存器可當作一般暫存器使用.

$003 : TB2 == 讀表暫存器 2

與 TB1 同樣功能,此暫存器提供我們在使用 RTB 指令讀取 ROM TABLE 時,暫時存放讀取 16 位元資料中的次高 4 位元(D11~D8)用;在不使用讀表功能時,此暫存器可當作一般暫存器使用.

$004 : TB3 == 讀表暫存器 3

同樣的此暫存器提供我們在使用 RTB 指令讀取 ROM TABLE 時,暫時存放讀取 16 位元資料中的最高 4 位元(D15~D12)用;在不使用讀表功能時,此暫存器可當作一般暫存器使用.

$005 : DPL == 低位指標暫存器,

使用間接方式對 RAM 位址讀寫或運算時,此暫存器是指標(DP)的 10 條地址線中最低的 4 位元(A3~A0);同時亦是使用 RTB 指令讀取 ROM TABLE 資料的 12 位元地址線中最低的 4 位元(A3~A0),所以運用在上述二種方式使用之前必須先設定此暫存器.若是與上述的使用不衝突時可當作一般暫存器使用.

$006 : DPM == 中位指標暫存器,

與 DPL 功能相同,在使用間接方式對 RAM 位址讀寫或運算時,此暫存器是指標(DP)的 10 條地址線之中間的 4 位元(A7~A4);同時亦是使用 RTB 指令讀取 ROM TABLE 資料的 12 位元地址線之中間的 4 位元(A7~A4),所以運用在上述二種方式使用之前必須先設定此暫存器.若是與上述的使用不衝突時可當作一般暫存器使用.

$007 : DPH == 高位指標暫存器,

與上述二暫存器功能相同,在使用間接方式對 RAM 位址讀寫或運算時,此暫存器是指標(DP)的 10 條地址線中最高的 2 位元(A9~A8),A10 及 A11 會被忽略掉一般設定為”0”;同時亦是使用 RTB 指令讀取 ROM TABLE 資料的 12 位元地址線中最高的 4 位元(A11~A8),所以運用在上述二種方式使用之前必須先設定此暫存器.若是與上述的使用不衝突時可當作一般暫存器使用.

3. 系統復位(Reset)及中斷產生

當復位端口輸入一個低電平會產生復位信號(Reset),此時 CPU 將進入復位狀態,進入復位狀態程式計數器會設定為$000,也就是說當復位端口回復到高電平,CPU 會從位址$000 開始執行.

當中斷端口輸入一個低電平會產生中斷信號(INTB),通知 CPU 將要進入中斷程式,在進入中斷程式之前,CPU 會先將目前正在執行的指令執行完畢,再將程式計數器及旗標壓入(Push)堆棧,同時將中斷允許旗標設定為 Disable,以避免中斷重複產生而造成堆棧爆掉,使 CPU 無法返回主程式的正確位址,接著程式計數器會設定為$001,讓中斷程式從$001 開始執行,待中斷程式執行完畢以 RTI 指令結束,CPU 執行 RTI 指令會先取回(Pop)堆棧中的返回位址及旗標放回到程式計數器及旗標,使程式能夠繼續依照程序執行,同時將中斷允許旗標清除為 Enable,讓中斷能夠再度產生,中斷副程式中若使用呼叫副程式的用法則需注意堆棧的使用,若使用超過規定的層數,將使 CPU 返回位址與旗標錯誤,而使程序亂掉。.

4. T4x6N 的虛擬指令

在 T4x6N 的編譯的程式中,可接受以下數種的虛擬指令,以協助我們程式的撰寫:

.ORG : 起始位址--設定下一條指令開始編譯的位址,之後的指令編譯時會依序增量.

```
[EX]            .ORG    $100
```

.DW : 資料定義--定義其後接的是 16 bit 的資料,主要是建表(TABLE)之用

```
[EX]            .DW     $1234,$6677
```

.EQ : 標籤定義--定義位址標籤或變數標籤(Label)為其後的數值取代.

```
[EX]  .EQ       SECBUF  $020
```

定義的數值不論有沒有加#，皆視為相同的數值，無法做為後續指令編譯時，判斷是 RAM 的位址或是常數的依據。

定義完成的常數在程式中使用,前面不需要加”#”

```
[EX]  .EQ       DIFF    $3
                SUB     DIFF,SECBUF,M   ;SECBUF-3
```

.END : 程式結尾--表示程式編寫已結束,其後的指令不再編譯.

```
[EX]            .END
```

Label : 位址或變數標籤--必須是英文字或數字組成的字串,最多不可超過 16 個字元.

[EX]　SELF:　　JMP　　SELF

;　　：註記符號—分號之後的文字會被忽略不會編輯,主要作為程式註記之用.
特別注意分號之後不可立即接”*”號

另外在撰寫程式及下面指令的介紹中,以下的符號皆有其特別的定義,也請特別注意

#:代表是立即值

$:代表是十六進制

例如: #$12 與#12 前者表立即值 18(十進制)　後者表立即值 12(十進制)

rr　: A(5:0) --- 以 6 位元表示的位址,表示範圍在$000~$03F 之間.

rrr : A(9:0) --- 以 10 位元表示的位址,表示範圍在$000~$3FF 之間.

aaa : A(11:0) -- 以 12 位元表示的位址,表示範圍在$000~$FFF 之間.

C　: 進位旗標

Z　: 零旗標

E　: 指標暫存器自動增量旗標

I　: 中斷允許旗標

+　: 算術加運算

-　: 算術減運算

|　: 邏輯 OR 運算

&　: 邏輯 AND 運算

⊕　: 邏輯 XOR 運算

5. T4x6N 的定址方式

T4x6N 的定址方式可分為下列四種,將分別敘述如下:

(1) 隱含定址模式

隱含定址方式的指令,在編寫程式時僅會看見助記碼部分,不會有參數接在後面(如下表助記碼欄),也就是指令工作皆在 CPU 內部處理完畢,此類指令包含有旗標設定指令 CDP,SDP,SEC,CLC 副程式返回指令 RTS 及中斷返回指令 RTI 及 NOP 指令.

助記碼	運算碼	功能	Z C E I
NOP	1000 000000000001	No operation	- - - -
RTS	1000 000000000000	PC←SC	- - - -
RTI	1011 111111111111	PC←SC,C←C’,Z←Z’,E←E’,I←0	x x x 0
CDP	1100 111111111111	E←0 (Enable DP auto increasement)	- - 0 -
SDP	1101 111111111111	E←1 (Disable DP auto increasement)	- - 1 -
SEC	1110 111111111111	C←1	- 1 - -
CLC	1111 111111111111	C←0	- 0 - -

(2) 立即定址模式

立即定址是將接在助記碼之後的立即值與接在其後的可讀寫記憶體或暫存器參數做處理,再將結果依最後的參數指示存回累積器或可讀寫記憶體,所以除了助記碼以外尚包含了立即值,可讀寫記憶體及儲存指示三個參數,此類指令包含有 ADC ADD SBC

SUB ORI XOR AND 算數邏輯運算指令,還有 CMP TST 的比較測試指令及 STX 儲存指令,其中 CMP TST 二指令不需將結果儲存起來,STX 指令一定存回記憶體,不需特別指示儲存方向,所以此三個指令沒有第三項參數(請參閱下表助記碼欄).

由於此類指令可讀寫記憶體僅有 6 位元表示,限制了可直接讀寫記憶體的範圍,也就是說此類指令只可以對位址$000~$03F 的記憶體使用,超出此一範圍的記憶體就必須經過累積器的轉換及配合直接定址指令的使用方能達到需求,在使用時要特別注意.

助記碼	運算碼	功能	Z C E I
ADC #$n,$rr,A	0000 00 nnnn rrrrrr	Acc←RAM[rr]+n+C	x x - -
ADC #$n,$rr,M	0000 10 nnnn rrrrrr	RAM[rr]←RAM[rr]+n+C	x x - -
ADD #$n,$rr,A	0001 00 nnnn rrrrrr	Acc←RAM[r]+n	x x - -
ADD #$n,$rr,M	0001 10 nnnn rrrrrr	RAM[rr]←RAM[rr]+n	x x - -
SBC #$n,$rr,A	0010 00 nnnn rrrrrr	Acc←RAM[rr]-n-C	x x - -
SBC #$n,$rr,M	0010 10 nnnn rrrrrr	RAM[rr]←RAM[rr]-n-C	x x - -
SUB #$n,$rr,A	0011 00 nnnn rrrrrr	Acc←RAM[rr]-n	x x - -
SUB #$n,$rr,M	0011 10 nnnn rrrrrr	RAM[rr]←RAM[rr]-n	x x - -
ORI #$n,$rr,A	0100 00 nnnn rrrrrr	Acc←RAM[rr]|n	x - - -
ORI #$n,$rr,M	0100 10 nnnn rrrrrr	RAM[rr]←RAM[rr]|n	x - - -
XOR #$n,$rr,A	0101 00 nnnn rrrrrr	Acc←RAM[rr]⊕n	x - - -
XOR #$n,$rr,M	0101 10 nnnn rrrrrr	RAM[rr]←RAM[rr]⊕n	x - - -
AND #$n,$rr,A	0110 00 nnnn rrrrrr	Acc←RAM[rr]&n	x - - -
AND #$n,$rr,M	0110 10 nnnn rrrrrr	RAM[rr]←RAM[rr]&n	x - - -
CMP #$n,$rr	0111 00 nnnn rrrrrr	RAM[rr]-n	x x - -
TST #$n,$rr	0111 10 nnnn rrrrrr	RAM[rr]&n	x - - -
STX #$n,$rr	1000 10 nnnn rrrrrr	RAM[rr]←n	- - - -

(3) 直接定址模式

直接定址在助記碼之後接二個參數,第一個是要處理的可讀寫記憶體之位置,第二個為處理完的結果要儲存方向的指示,此類指令包含有 ADC ADD SBC SUB ORI XOR AND 算數邏輯運算指令,還有 RLC RRC 旋轉指令,CMP TST 的比較測試指令及 LDA 讀取記憶體指令和 STX 儲存指令,與立即定值相同 CMP TST 二指令不需將結果儲存起來,STX 指令一定存回記憶體,LDA 一定將值取出放到從累積器,故而不需特別指示儲存方向,所以此四個指令沒有第二項參數(請參閱下表助記碼欄).

助記碼	運算碼	功能	Z C E I
ADC $rrr,A	0000 01 rrrrrrrrrr	Acc←Acc+RAM[rrr]+C	x x - -
ADC $rrr,M	0000 11 rrrrrrrrrr	RAM[rrr]←Acc+RAM[rrr]+C	x x - -
ADD $rrr,A	0001 01 rrrrrrrrrr	Acc←Acc+RAM[rrr]	x x - -
ADD $rrr,M	0001 11 rrrrrrrrrr	RAM[rrr]←Acc+RAM[rrr]	x x - -

SBC $rrr,A	0010 01 rrrrrrrrrr	Acc←RAM[rrr]-Acc-C	x x - -
SBC $rrr,M	0010 11 rrrrrrrrrr	RAM[rrr]←RAM[rrr]-Acc-C	x x - -
SUB $rrr,A	0011 01 rrrrrrrrrr	Acc←RAM[rrr]-Acc	x x - -
SUB $rrr,M	0011 11 rrrrrrrrrr	RAM[rrr]←RAM[rrr]-Acc	x x - -
ORI $rrr,A	0100 01 rrrrrrrrrr	Acc←RAM[rrr]|Acc	x - - -
ORI $rrr,M	0100 11 rrrrrrrrrr	RAM[rrr]←RAM[rrr]|Acc	x - - -
XOR $rrr,A	0101 01 rrrrrrrrrr	Acc←RAM[rrr]⊕Acc	x - - -
XOR $rrr,M	0101 11 rrrrrrrrrr	RAM[rrr]←RAM[rrr]⊕Acc	x - - -
AND $rrr,A	0110 01 rrrrrrrrrr	Acc←RAM[rrr]&Acc	x - - -
AND $rrr,M	0110 11 rrrrrrrrrr	RAM[rrr]←RAM[rrr]&Acc	x - - -
CMP $rrr	0111 01 rrrrrrrrrr	RAM[rrr]-Acc	x x - -
TST $rrr	0111 11 rrrrrrrrrr	RAM[rrr]&Acc	x - - -
LDA $rrr	1000 01 rrrrrrrrrr	Acc←RAM[rrr]	x - - -
STX $rrr	1000 11 rrrrrrrrrr	RAM[rrr]←Acc	- - - -
RLC $rrr,A	1001 00 rrrrrrrrrr	C←RAM[rrr]←C,Acc←result	x x - -
RLC $rrr,M	1001 10 rrrrrrrrrr	C←RAM[rrr]←C,RAM[rrr]←result	x x - -
RRC $rrr,A	1001 01 rrrrrrrrrr	C→RAM[rrr]→C,Acc←result	x x - -
RRC $rrr,M	1001 11 rrrrrrrrrr	C→RAM[rrr]→C,RAM[rrr]←result	x x - -

(4) 絕對位址定址模式

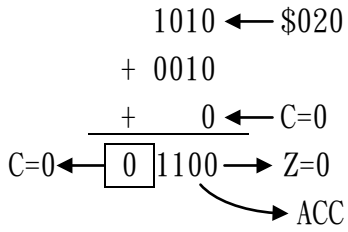
絕對位址定址是將接在助記碼之後的參數,設定程式計數器(PC)或是指標暫存器(DP),參數所表示的是一個 12 位元位址,JMP JPC JPZ 及 CAL 指令都會影響程式計數器的值,這會改變程式的依序執行,LDP 指令則可一次快速的設定指標暫存器(DP),不需要分別設定 DPH DPM DPL 三個暫存器,RTB 指令是特別為讀取程式記憶體中的表(Table)而設的指令,可將程式記憶體中設定值讀回到 TB3 TB2,TB1 及 Acc 暫存器中再讓 CPU 使用.其亦是 T4x6N 中唯一需要 2 個機械周期(4 個系統時脈)的指令,

助記碼	運算碼	功能	Z C E I
LDP $xyz	1010 xxxxyyyyzzzz	DPH←x,DPM←y,DPL←z	- - - -
RTB $aaa	1011 aaaaaaaaaaaa	TB3,TB2,TB1,Acc←ROM[aaa]	- - - -
JMP $aaa	1100 aaaaaaaaaaaa	PC←aaa	- - - -
JPC $aaa	1101 aaaaaaaaaaaa	If C=1 PC←aaa else PC←PC+1	- - - -
JPZ $aaa	1110 aaaaaaaaaaaa	If Z=1 PC←aaa else PC←PC+1	- - - -
CAL $aaa	1111 aaaaaaaaaaaa	SC←PC+1,C' ←C,Z' ←Z,E' ←E,PC←aaa	- - - -

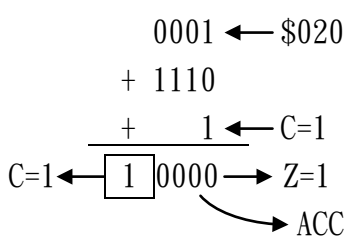
6. T4x6N 的指令介紹

****
* 指令型式: ADC #$n,$rr,A
* 指令說明: 記憶體$rr 的值加上立即值 n 再加上 C,結果存入累積器及 C 旗標
* 運算情形: Acc←RAM[rr] + n + C
* 影響旗標: Z,C
****

[EX]　　ADC #$2,$20,A
執行前 :　[$020]=$A,ACC=$5,C=0,Z=0
執行後 :　[$020]=$A,ACC=$C,C=0,Z=0

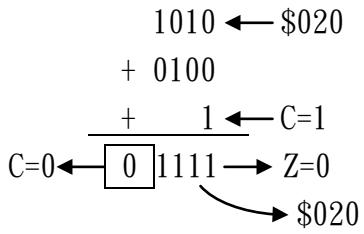


[EX]　　ADC #$E,$20,A
執行前 :　[$020]=$1,ACC=$5,C=1,Z=0
執行後 :　[$020]=$1,ACC=$0,C=1,Z=1

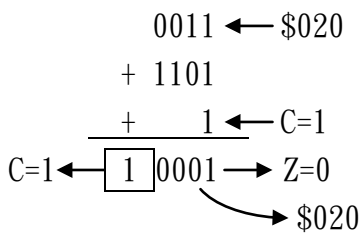


****
* 指令型式: ADC #$n,$rr,M
* 指令說明: 記憶體$rr 的值加上立即值 n 再加上 C,結果存入記憶體$rr 及 C 旗標
* 運算情形: RAM[rr]←RAM[rr] + n + C
* 影響旗標: Z,C
****

[EX]　　ADC #$4,$20,M
執行前　[$020]=$A,ACC=$5,C=1,Z=0
執行後　[$020]=$F,ACC=$5,C=0,Z=0

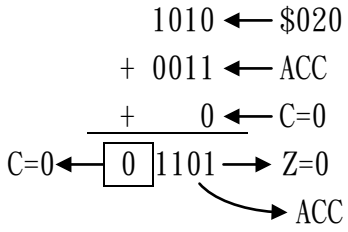


[EX]　　ADC #$D,$20,M
執行前　[$020]=$3,ACC=$5,C=1,Z=0
執行後　[$020]=$1,ACC=$5,C=1,Z=0

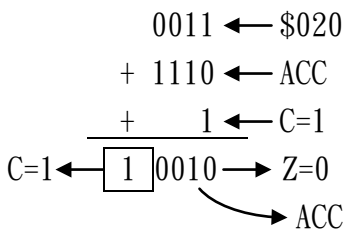


****
* 指令型式: ADC $rrr,A
* 指令說明: 記憶體$rrr 的值加上累積器的值再加上 C,結果存入累積器及 C 旗標
* 運算情形: ACC←RAM[rrr] + ACC + C
* 影響旗標: Z,C
****

[EX]　　ADC $020,A
執行前　[$020]=$A,ACC=$3,C=0,Z=0
執行後　[$020]=$A,ACC=$D,C=0,Z=0

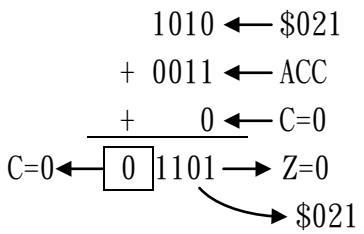


[EX]　　ADC $020,A
執行前　[$020]=$3,ACC=$E,C=1,Z=0
執行後　[$020]=$3,ACC=$2,C=1,Z=0

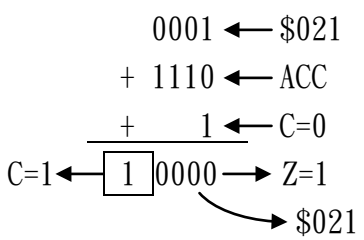


****
* 指令型式: ADC $rrr,M
* 指令說明: 記憶體$rrr 的值加上累積器的值再加上 C,結果存入記憶體$rrr 及 C 旗標
* 運算情形: RAM[rrr]←RAM[rrr] + ACC + C
* 影響旗標: Z,C
****

[EX]　　ADC $021,M
執行前　[$021]=$A,ACC=$3,C=0,Z=0
執行後　[$021]=$D,ACC=$3,C=0,Z=0

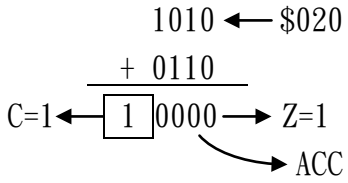


[EX]　　ADC $021,M
執行前　[$021]=$1,ACC=$E,C=1,Z=0
執行後　[$021]=$0,ACC=$E,C=1,Z=1

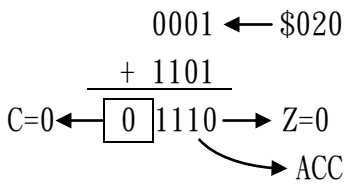


****
* 指令型式: ADD #$n,$rr,A
* 指令說明: 記憶體$rr 的值加上立即值 n,結果存入累積器及 C 旗標
* 運算情形: Acc←RAM[rr] + n
* 影響旗標: Z,C
****

[EX]　　ADD #$6,$20,A
執行前　[$020]=$A,ACC=$3,C=0,Z=0
執行後　[$020]=$A,ACC=$0,C=1,Z=1

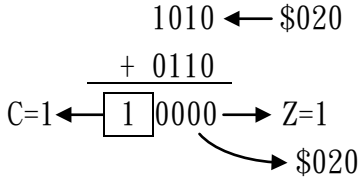


[EX]　　ADD #$D,$20,A
執行前　[$020]=$1,ACC=$3,C=1,Z=0
執行後　[$020]=$1,ACC=$E,C=0,Z=0

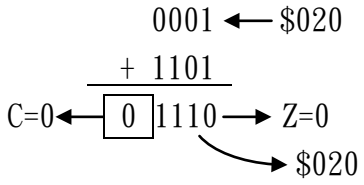


****
* 指令型式: ADD #$n,$rr,M
* 指令說明: 記憶體$rr 的值加上立即值 n,結果存入記憶體$rr 及 C 旗標
* 運算情形: RAM[rr]←RAM[rr] + n
* 影響旗標: Z,C
****

[EX]　　ADD #$6,$20,M
執行前　[$020]=$A,ACC=$3,C=0,Z=0
執行後　[$020]=$0,ACC=$3,C=1,Z=1



[EX]　　ADD #$D,$20,M
執行前　[$020]=$1,ACC=$3,C=1,Z=0
執行後　[$020]=$E,ACC=$3,C=0,Z=0



****
* 指令型式: ADD $rrr,A
* 指令說明: 記憶體$rrr 的值加上累積器的值,結果存入累積器及 C 旗標
* 運算情形: Acc←RAM[rrr] + ACC
* 影響旗標: Z,C
****

```
[EX]      ADD $020,A                          1010 ←$020
執行前    [$020]=$A,ACC=$3,C=0,Z=0          + 0011 ←ACC
執行後    [$020]=$A,ACC=$D,C=0,Z=0    C=0← 0 1101 → Z=0
                                              → ACC
```

```
[EX]      ADD $020,A                          0001 ←$020
執行前    [$020]=$1,ACC=$F,C=1,Z=0          + 1111 ←ACC
執行後    [$020]=$1,ACC=$0,C=1,Z=1    C=1← 1 0000 → Z=1
                                              → ACC
```

```
*************************************************************************
* 指令型式: ADD $rrr,M
* 指令說明: 記憶體$rrr 的值加上累積器的值,結果存入記憶體$rrr 及 C 旗標
* 運算情形: RAM[rrr]←RAM[rrr] + ACC
* 影響旗標: Z,C
*************************************************************************
```

```
[EX]      ADD $020,M                          1010 ←$020
執行前    [$020]=$A,ACC=$3,C=0,Z=0          + 0011 ←ACC
執行後    [$020]=$D,ACC=$3,C=0,Z=0    C=0← 0 1101 → Z=0
                                              → $020
```

```
[EX]      ADD $020,M                          0001 ←$020
執行前    [$020]=$1,ACC=$E,C=1,Z=0          + 1110 ←ACC
執行後    [$020]=$F,ACC=$E,C=0,Z=0    C=0← 0 1111 → Z=0
                                              → $020
```

```
*************************************************************************
* 指令型式: SBC #$n,$rr,A
* 指令說明: 記憶體$rr 的值減去立即值 n 再減去 C,結果存入累積器及 C 旗標
* 運算情形: ACC←RAM[rr] - n - C
* 影響旗標: Z,C
*************************************************************************
```

```
[EX]      SBC #$3,$20,A                       1010 ←$020
執行前    [$020]=$A,ACC=$4,C=0,Z=0            - 0011
                                              -    0 ←C=0
執行後    [$020]=$A,ACC=$7,C=0,Z=0    C=0← 0 0111 → Z=0
                                              → ACC
```

[EX]　　SBC #$E,$20,A　　0001 ← $020

執行前　[$020]=$1,ACC=$3,C=1,Z=0　　- 1110

　　- 1 ← C=1

執行後　[$020]=$1,ACC=$2,C=1,Z=0　　C=1 ← [1] 0010 → Z=0

　　→ ACC

[EX]　　SBC #$E,$20,A　　1110 ← $020

執行前　[$020]=$E,ACC=$3,C=0,Z=0　　- 1110

　　- 0 ← C=0

執行後　[$020]=$E,ACC=$0,C=0,Z=1　　C=0 ← [0] 0000 → Z=1

　　→ ACC

[EX]　　SBC #$E,$20,A　　1101 ← $020

執行前　[$020]=$D,ACC=$3,C=1,Z=0　　- 1110

　　- 1 ← C=1

執行後　[$020]=$D,ACC=$E,C=1,Z=0　　C=1 ← [1] 1110 → Z=0

　　→ ACC

****

* 指令型式: SBC #$n,$rr,M

* 指令說明: 記憶體$rr 的值減去立即值 n 再減去 C,結果存入記憶體$rr 及 C 旗標

* 運算情形: RAM[rr]←RAM[rr] - n - C

* 影響旗標: Z,C

****

[EX]　　SBC #$3,$20,M　　1010 ← $020

執行前　[$020]=$A,ACC=4,C=0,Z=0　　- 0011

　　- 0 ← C=0

執行後　[$020]=$7,ACC=4,C=0,Z=0　　C=0 ← [0] 0111 → Z=0

　　→ $020

[EX]　　SBC #$E,$20,M　　0001 ← $020

執行前　[$020]=$1,ACC=4,C=1,Z=0　　- 1110

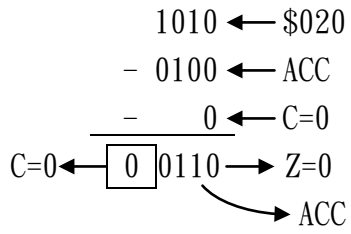
　　- 1 ← C=1

執行後　[$020]=$2,ACC=4,C=1,Z=0　　C=1 ← [1] 0010 → Z=0

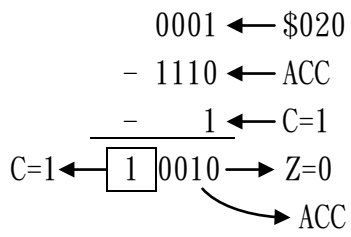
　　→ $020

****
* 指令型式: SBC $rrr,A
* 指令說明: 記憶體$rrr 的值減去累積器的值再減去 C,結果存入累積器及 C 旗標
* 運算情形: ACC←RAM[rrr] - ACC - C
* 影響旗標: Z,C
****

[EX] SBC $020,A
執行前 [$020]=$A,ACC=$4,C=0,Z=0
執行後 [$020]=$A,ACC=$6,C=0,Z=0

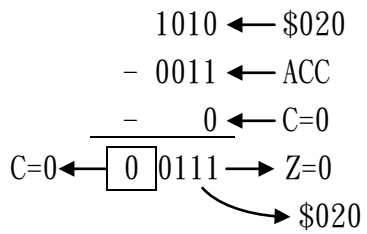


[EX] SBC $020,A
執行前 [$020]=$1,ACC=$E,C=1,Z=0
執行後 [$020]=$1,ACC=$2,C=1,Z=0

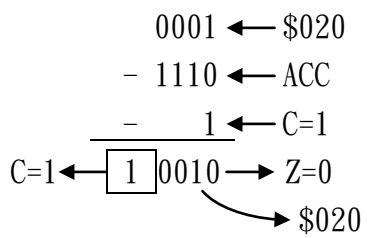


****
* 指令型式: SBC $rrr,M
* 指令說明: 記憶體$rrr 的值減去累積器的值再減去 C,結果存入記憶體$rrr 及 C 旗標
* 運算情形: RAM[rrr]←RAM[rrr] - ACC - C
* 影響旗標: Z,C
****

[EX] SBC $020,M
執行前 [$020]=$A,ACC=$3,C=0,Z=0
執行後 [$020]=$7,ACC=$3,C=0,Z=0

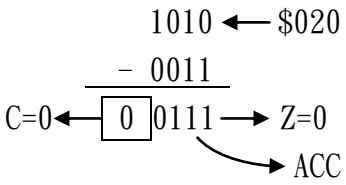


[EX] SBC $020,M
執行前 [$020]=$1,ACC=$E,C=1,Z=0
執行後 [$020]=$2,ACC=$E,C=1,Z=0

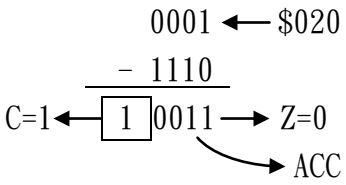


****
* 指令型式: SUB #$n,$rr,A
* 指令說明: 記憶體$rr 的值減去立即值 n,結果存入累積器及 C 旗標
* 運算情形: ACC←RAM[rr] - n
* 影響旗標: Z,C
****

[EX]　　SUB #$3,$20,A
執行前　[$020]=$A,ACC=$4,C=0,Z=0
執行後　[$020]=$A,ACC=$7,C=0,Z=0

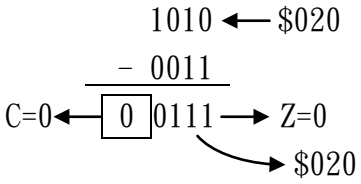


[EX]　　SUB #$E,$20,A
執行前　[$020]=$1,ACC=$4,C=1,Z=0
執行後　[$020]=$1,ACC=$3,C=1,Z=0

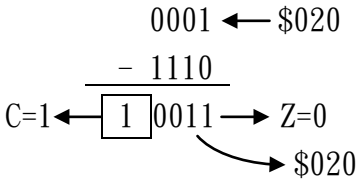


****
* 指令型式: SUB #$n,$rr,M
* 指令說明: 記憶體$rr 的值減去立即值 n,結果存入記憶體$rr 及 C 旗標
* 運算情形: RAM[rr]←RAM[rr] - n
* 影響旗標: Z,C
****

[EX]　　SUB #$3,$20,M
執行前　[$020]=$A,ACC=$4,C=0,Z=0
執行後　[$020]=$7,ACC=$4,C=0,Z=0



[EX]　　SUB #$E,$20,M
執行前　[$020]=$1,ACC=$4,C=1,Z=0
執行後　[$020]=$3,ACC=$4,C=1,Z=0



****
* 指令型式: SUB $rrr,A
* 指令說明: 記憶體$rrr 的值減去累積器的值,結果存入累積器及 C 旗標
* 運算情形: ACC←RAM[rrr] - ACC
* 影響旗標: Z,C
****

[EX]　　SUB $020,A　　1010 ← $020

執行前　[$020]=$A,ACC=$3,C=0,Z=0　　- 0011 ← ACC

執行後　[$020]=$A,ACC=$7,C=0,Z=0　　C=0 ← [0] 0111 → Z=0

ACC

[EX]　　SUB $020,A　　0001 ← $020

執行前　[$020]=$1,ACC=$E,C=1,Z=0　　- 1110 ← ACC

執行後　[$020]=$1,ACC=$3,C=1,Z=0　　C=1 ← [1] 0011 → Z=0

ACC

****

* 指令型式: SUB $rrr,M

* 指令說明: 記憶體$rrr 的值減去累積器的值,結果存入記憶體$rrr 及 C 旗標

* 運算情形: RAM[rrr]←RAM[rrr] - ACC

* 影響旗標: Z,C

****

[EX]　　SUB $020,M　　1010 ← $020

執行前　[$020]=$A,ACC=$3,C=0,Z=0　　- 0011 ← ACC

執行後　[$020]=$7,ACC=$3,C=0,Z=0　　C=0 ← [0] 0111 → Z=0

$020

[EX]　　SUB $020,M　　0001 ← $020

執行前　[$020]=$1,ACC=$E,C=1,Z=0　　- 1110 ← ACC

執行後　[$020]=$3,ACC=$E,C=1,Z=0　　C=1 ← [1] 0011 → Z=0

$020

****

* 指令型式: ORI #$n,$rr,A

* 指令說明: 記憶體$rr 的值與立即值 n 做邏輯 OR,結果存入累積器

* 運算情形: ACC←RAM[rr] | n

* 影響旗標: Z

****

[EX]　　ORI #$6,$20,A　　1001 ← $020

執行前　[$020]=$9,ACC=$5,Z=0　　or 0110

執行後　[$020]=$9,ACC=$F,Z=0　　1111 → Z=0

ACC

```
[EX]      ORI #$0,$20,A                    0000 ←— $020
執行前    [$020]=$0,ACC=$5,Z=0            or 0000
執行後    [$020]=$0,ACC=$0,Z=1               0000 —→ Z=1
                                                  ↘→ ACC
```

****

* 指令型式: ORI #$n,$rr,M

* 指令說明: 記憶體$rr 的值與立即值 n 做邏輯 OR,結果存入記憶體$rr

* 運算情形: RAM[rr]←RAM[rr] | n

* 影響旗標: Z

****

```
[EX]      ORI #$6,$20,M                    1001 ←— $020
執行前    [$020]=$9,ACC=$5,Z=0            or 0110
執行後    [$020]=$F,ACC=$5,Z=0               0000 —→ Z=0
                                                  ↘→ $020
```

```
[EX]      ORI #$0,$20,M                    0000 ←— $020
執行前    [$020]=$0,ACC=$5,Z=0            or 0000
執行後    [$020]=$0,ACC=$5,Z=1               0000 —→ Z=1
                                                  ↘→ $020
```

****

* 指令型式: ORI $rrr,A

* 指令說明: 記憶體$rrr 的值與累積器的值做邏輯 OR,結果存入累積器

* 運算情形: ACC←RAM[rrr] | n

* 影響旗標: Z

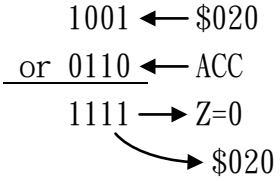
****

```
[EX]      ORI $020,A                       1001 ←— $020
執行前    [$020]=$9,ACC=$6,Z=0            or 0110 ←— ACC
執行後    [$020]=$9,ACC=$F,Z=0               1111 —→ Z=0
                                                  ↘→ ACC
```

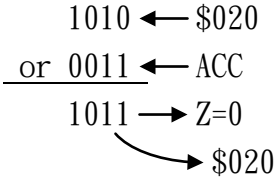
```
[EX]      ORI $020,A                       0000 ←— $020
執行前    [$020]=$0,ACC=$0,Z=0            or 0000 ←— ACC
執行後    [$020]=$0,ACC=$0,Z=1               0000 —→ Z=1
                                                  ↘→ ACC
```

****
* 指令型式: ORI $rrr,M
* 指令說明: 記憶體$rrr 的值與累積器做邏輯 OR,結果存入記憶體$rrr
* 運算情形: RAM[rrr]←RAM[rrr] | ACC
* 影響旗標: Z
****

[EX]　　ORI $020,M
執行前　[$020]=$9,ACC=$6,Z=0
執行後　[$020]=$F,ACC=$6,Z=0

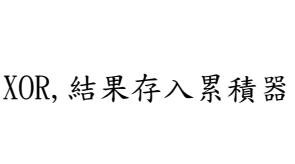


[EX]　　ORI $020,M
執行前　[$020]=$A,ACC=$3,Z=0
執行後　[$020]=$B,ACC=$3,Z=0

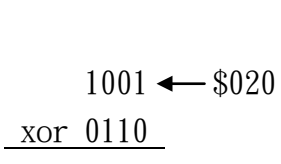


****
* 指令型式: XOR #$n,$rr,A
* 指令說明: 記憶體$rr 的值與立即值 n 做邏輯 XOR,結果存入累積器
* 運算情形: ACC←RAM[rr] ⊕ n
* 影響旗標: Z
****

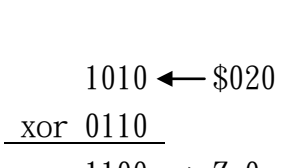
[EX]　　XOR #$6,$20,A
執行前　[$020]=$9,ACC=$3,Z=0
執行後　[$020]=$9,ACC=$F,Z=0



[EX]　　XOR #$6,$20,A
執行前　[$020]=$A,ACC=$3,Z=0
執行後　[$020]=$A,ACC=$C,Z=0



[EX]　　XOR #$A,$20,A
執行前　[$020]=$A,ACC=$3,Z=0
執行後　[$020]=$A,ACC=$0,Z=1



****
* 指令型式: XOR #$n,$rr,M
* 指令說明: 記憶體$rr 的值與立即值 n 做邏輯 XOR,結果存入記憶體$rr
* 運算情形: RAM[rr]←RAM[rr] ⊕ n
* 影響旗標: Z
****

[EX]　　XOR #$6,$20,M　　1001 ←$020
執行前　[$020]=$9 ACC=$3,Z=0　　xor 0110
執行後　[$020]=$F ACC=$3,Z=0　　1111 →Z=0
　　→$020

[EX]　　XOR #$6,$20,M　　1010 ←$020
執行前　[$020]=$A,ACC=$3,Z=0　　xor 0110
執行後　[$020]=$C,ACC=$3,Z=0　　1100 →Z=0
　　→$020

****
* 指令型式: XOR $rrr,A
* 指令說明: 記憶體$rrr 的值與累積器做邏輯 XOR,結果存入累積器
* 運算情形: ACC←RAM[rrr] ⊕ ACC
* 影響旗標: Z
****

[EX]　　XOR $20,A　　1001 ←$020
執行前　[$020]=$9,ACC=$6,Z=0　　xor 0110 ←ACC
執行後　[$020]=$9,ACC=$F,Z=0　　1111 →Z=0
　　→ACC

[EX]　　XOR $20,A　　1010 ←$020
執行前　[$020]=$A,ACC=$6,Z=0　　xor 0110 ←ACC
執行後　[$020]=$A,ACC=$C,Z=0　　1100 →Z=0
　　→ACC

****
* 指令型式: XOR $rrr,M
* 指令說明: 記憶體$rrr 的值與累積器做邏輯 XOR,結果存入記憶體$rrr
* 運算情形: RAM[rrr]←RAM[rrr] ⊕ ACC
* 影響旗標: Z
****

```
[EX]      XOR $20,M                          1001 ←— $020
執行前    [$020]=$9,ACC=$6,Z=0          xor 0110 ←— ACC
執行後    [$020]=$F,ACC=$6,Z=0               1111 —→ Z=0
                                                    → $020

[EX]      XOR $20,M                          1010 ←— $020
執行前    [$020]=$A,ACC=$6,Z=0          xor 0110 ←— ACC
執行後    [$020]=$C,ACC=$6,Z=0               1100 —→ Z=0
                                                    → $020
```

****

* 指令型式: AND #$n,$rr,A
* 指令說明: 記憶體$rr 的值與立即值 n 做邏輯 AND,結果存入累積器
* 運算情形: ACC←RAM[rr] & n
* 影響旗標: Z

****

```
[EX]      AND #$6,$20,A                      1001 ←— $020
執行前    [$020]=$9,ACC=$5,Z=0          and 0110
執行後    [$020]=$9,ACC=$0,Z=1               0000 —→ Z=1
                                                    → ACC

[EX]      AND #$6,$20,A                      1010 ←— $020
執行前    [$020]=$A,ACC=$F,Z=0          and 0110
執行後    [$020]=$A,ACC=$2,Z=0               0010 —→ Z=0
                                                    → ACC
```

****

* 指令型式: AND #$n,$rr,M
* 指令說明: 記憶體$rr 的值與立即值 n 做邏輯 AND,結果存入記憶體$rr
* 運算情形: RAM[rr]←RAM[rr] & n
* 影響旗標: Z

****

```
[EX]      AND #$6,$20,M                      1001 ←— $020
執行前    [$020]=$9,ACC=$F,Z=0          and 0110
執行後    [$020]=$0,ACC=$F,Z=1               0000 —→ Z=1
                                                    → $020
```

```
[EX]      AND #$6,$20,M                    1010 ←— $020
執行前    [$020]=$A,ACC=$7,Z=0            and 0110
執行後    [$020]=$2,ACC=$7,Z=0                0010 —→ Z=0
                                                  ↘→ $020
```

```
**************************************************************************
* 指令型式: AND $rrr,A
* 指令說明: 記憶體$rrr 的值與累積器做邏輯 AND,結果存入累積器
* 運算情形: ACC←RAM[rrr] & ACC
* 影響旗標: Z
**************************************************************************
```

```
[EX]      AND $020,A                       1001 ←— $020
執行前    [$020]=$9,ACC=$6,Z=0            and 0110 ←— ACC
執行後    [$020]=$9,ACC=$0,Z=1                0000 —→ Z=1
                                                  ↘→ ACC
```

```
[EX]      AND $020,A                       1010 ←— $020
執行前    [$020]=$A,ACC=$6,Z=0            and 0110 ←— ACC
執行後    [$020]=$A,ACC=$2,Z=0                0010 —→ Z=0
                                                  ↘→ ACC
```

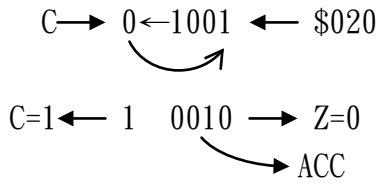
```
**************************************************************************
* 指令型式: AND $rrr,M
* 指令說明: 記憶體$rrr 的值與累積器做邏輯 AND,結果存入記憶體$rrr
* 運算情形: RAM[rrr]←RAM[rrr] & ACC
* 影響旗標: Z
**************************************************************************
```

```
[EX]      AND $020,M                       1001 ←— $020
執行前    [$020]=$9,ACC=$6,Z=0            and 0110 ←— ACC
執行後    [$020]=$0,ACC=$6,Z=1                0000 —→ Z=1
                                                  ↘→ $020
```

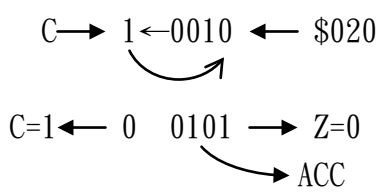
```
[EX]      AND $020,M                       1010 ←— $020
執行前    [$020]=$A,ACC=$6,Z=0            and 0110 ←— ACC
執行後    [$020]=$2,ACC=$6,Z=0                0010 —→ Z=0
                                                  ↘→ $020
```

****
* 指令型式: RLC $rrr,A
* 指令說明: 記憶體$rrr 的值與進位旗標向左旋轉,結果存入累積器
* 運算情形: C←RAM[rrr]←C , Acc←result
* 影響旗標: Z,C
****

[EX]　　RLC $020,A
執行前　[$020]=$9,ACC=$6,C=0,Z=0
執行後　[$020]=$9,ACC=$2,C=1,Z=0

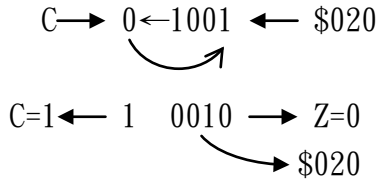


[EX]　　RLC $020,A
執行前　[$020]=$2,ACC=$6,C=1,Z=0
執行後　[$020]=$2,ACC=$5,C=0,Z=0

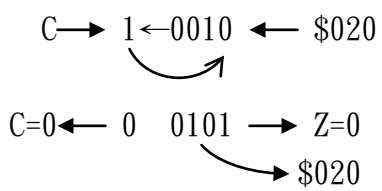


****
* 指令型式: RLC $rrr,M
* 指令說明: 記憶體$rrr 的值與進位旗標向左旋轉,結果存入記憶體$rrr
* 運算情形: C←RAM[rrr]←C , RAM[rrr]←result
* 影響旗標: Z,C
****

[EX]　　RLC $20,M
執行前　[$020]=$9,ACC=$6,C=0,Z=0
執行後　[$020]=$2,ACC=$6,C=1,Z=0

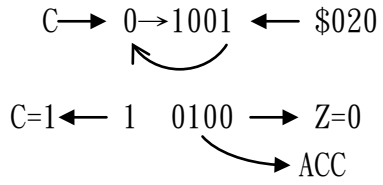


[EX]　　RLC $20,M
執行前　[$020]=$2,ACC=$6,C=1,Z=0
執行後　[$020]=$5,ACC=$6,C=0,Z=0

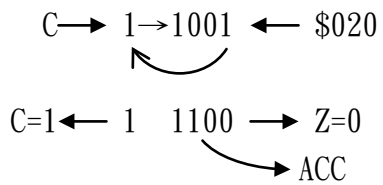


****
* 指令型式: RRC $rrr,A
* 指令說明: 記憶體$rrr 的值與進位旗標向右旋轉,結果存入累積器
* 運算情形: C→RAM[rrr]→C , ACC←result
* 影響旗標: Z,C
****

[EX]　　RRC $20,A
執行前　[$020]=$9,ACC=$6,C=0,Z=0
執行後　[$020]=$9,ACC=$4,C=1,Z=0

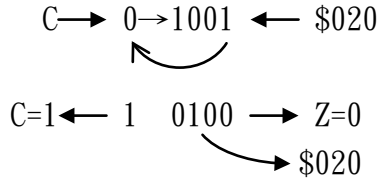


[EX]　　RRC $20,A
執行前　[$020]=$9,ACC=$4,C=1,Z=0
執行後　[$020]=$9,ACC=$C,C=1,Z=0

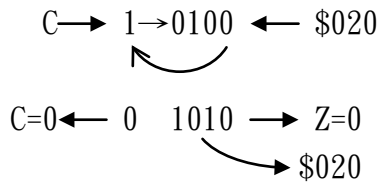


****
* 指令型式: RRC $rrr,M
* 指令說明: 記憶體$rrr 的值與進位旗標向右旋轉,結果存入記憶體$rrr
* 運算情形: C→RAM[rrr]→C , RAM[rrr]←result
* 影響旗標: Z,C
****

[EX]　　RRC $20,M
執行前　[$020]=$9,ACC=$6,C=0,Z=0
執行後　[$020]=$4,ACC=$6,C=1,Z=0



[EX]　　RRC $20,M
執行前　[$020]=$4,ACC=$6,C=1,Z=0
執行後　[$020]=$A,ACC=$6,C=0,Z=0



****
* 指令型式: LDA $rrr
* 指令說明: 記憶體$rrr 的值存入累積器
* 運算情形: ACC←RAM[rrr]
* 影響旗標: Z
****

[EX]　　LDA $020
執行前　[$020]=$9,ACC=$0,Z=0
執行後　[$020]=$9,ACC=$9,Z=0

[EX]　　LDA $020
執行前　[$020]=$0,ACC=$7,Z=0
執行後　[$020]=$0,ACC=$0,Z=1

****
* 指令型式: STX #$n,$rr
* 指令說明: 立即值 n 存入記憶體$rr
* 運算情形: RAM[rr]←n
* 影響旗標: 無
****
[EX]　　STX #$0,$20
執行前　[$020]=$9
執行後　[$020]=$0

[EX]　　STX #$6,$20
執行前　[$020]=$A
執行後　[$020]=$6

****
* 指令型式: STX $rrr
* 指令說明: 累積器的值存入記憶體$rrr
* 運算情形: RAM[rrr]←ACC
* 影響旗標: 無
****
[EX]　　STX $020
執行前　[$020]=$9,ACC=$6
執行後　[$020]=$6,ACC=$6

****
* 指令型式: TST #$n,$rr
* 指令說明: 記憶體$rr 的值與立即值 n 做邏輯 AND,結果不存起來,僅影響零旗標
* 運算情形: RAM[rr] & n
* 影響旗標: Z
****
[EX]　　TST #$8,$20　　1010 ←— $020
執行前　[$020]=$A,ACC=$0,Z=0　　and 1000
執行後　[$020]=$A,ACC=$0,Z=0　　1000 —→ Z=0

****
* 指令型式: TST $rrr
* 指令說明: 記憶體$rrr 的值與累積器做邏輯 AND,結果不存起來,僅影響零旗標
* 運算情形: RAM[rrr] & ACC
* 影響旗標: Z
****

[EX]　　TST $020　　1010 ←$020

執行前　　[$020]=$A,ACC=$4,Z=0　　and 0100

執行後　　[$020]=$A,ACC=$4,Z=1　　0000 →Z=1

****

* 指令型式: CMP #$n,$rr

* 指令說明: 記憶體$rr 的值減去立即值 n,結果不存起來,僅影響進位及零旗標狀況

* 運算情形: RAM[rr] - n

* 影響旗標: Z,C

****

[EX]　　CMP #$6,$20　　0110 ←$020

執行前　　[$020]=$6,ACC=7,C=1,Z=0　　- 0110

執行後　　[$020]=$6,ACC=7,C=0,Z=1　　C=0← 0 0000 → Z=1

[EX]　　CMP #$6,$20　　0101 ←$020

執行前　　[$020]=$5,ACC=7,C=0,Z=0　　- 0110

執行後　　[$020]=$5,ACC=7,C=1,Z=0　　C=1← 1 1111 → Z=0

****

* 指令型式: CMP $rrr

* 指令說明: 記憶體$rrr 的值減去累積器,結果不存起來,僅影響進位及零旗標狀況

* 運算情形: RAM[rrr] - ACC

* 影響旗標: Z,C

****

[EX]　　CMP $020　　0110 ←$020

執行前　　[$020]=$6,ACC=$6,C=1,Z=0　　- 0110 ←ACC

執行後　　[$020]=$6,ACC=$6,C=0,Z=1　　C=0← 0 0000 → Z=0

[EX]　　CMP $020　　0101 ←$020

執行前　　[$020]=$5,ACC=$6,C=0,Z=0　　- 0110 ←ACC

執行後　　[$020]=$5,ACC=$6,C=1,Z=0　　C=1← 1 1111 → Z=0

****

* 指令型式: CLC

* 指令說明: 清除進位旗標為 0

* 運算情形: C←0

* 影響旗標: C

****

[EX]　　CLC

執行前　　C=1　　執行後　　C=0

```
*************************************************************************
* 指令型式: SEC
* 指令說明: 設定進位旗標為 1
* 運算情形: C←1
* 影響旗標: C
*************************************************************************
[EX]      SEC
執行前    C=0
執行後    C=1

*************************************************************************
* 指令型式: LDP $xyz
* 指令說明: 立即位址值 xyz 存入指標暫存器
* 運算情形: DPH←x,DPM←y,DPL←z
* 影響旗標: 無
*************************************************************************
[EX]      LDP $1FC
執行前    [$07]=$0,[$06]=$1,[$05]=$3
執行後    [$07]=$1,[$06]=$F,[$05]=$C

*************************************************************************
* 指令型式: RTB $aaa
* 指令說明: 讀取程式記憶體位址 aaa 的 16 位元資料置入 TB3/TB2/TB1/ACC
* 運算情形: TB3/TB2/TB1/ACC←ROM[$aaa]
* 影響旗標: 無
*************************************************************************
[EX]      RTB $100
          ROM[$100]=$1234
執行前    TB3=$5,TB2=$6,TB1=$7,ACC=$8
執行後    TB3=$1,TB2=$2,TB1=$3,ACC=$4

*************************************************************************
* 指令型式: CDP
* 指令說明: 清除指標暫存器為無自動遞增功能
* 運算情形: E←0
* 影響旗標: E
*************************************************************************
[EX]      CDP
執行前    E=1
執行後    E=0
```

```
*************************************************************************
* 指令型式: SDP
* 指令說明: 設定指標暫存器為有自動遞增功能
* 運算情形: E←1
* 影響旗標: E
*************************************************************************
[EX]      SDP
執行前    E=0
執行後    E=1

*************************************************************************
* 指令型式: CAL $aaa
* 指令說明: 呼叫位址在 aaa 的副程式
* 運算情形: SC←PC+1,C' ←C,Z' ←Z,E' ←E,PC←aaa
* 影響旗標: 無
*************************************************************************
[EX]      CAL $1FF
執行前    PC=$030  STACK=$07F
執行後    PC=$1FF  STACK=$031

*************************************************************************
* 指令型式: RTI(取代 RET 指令，特性完全相同)
* 指令說明: 中斷副程式返回呼叫程式
* 運算情形: PC←SC,C←C',Z←Z',E←E',I←0
* 影響旗標: Z,C,E,I
*************************************************************************
[EX]      RTI
執行前    STACK=$200,PC=$030
執行後    STACK=$200,PC=$200

*************************************************************************
* 指令型式: RTS
* 指令說明: 副程式返回呼叫程式
* 運算情形: PC←SC
* 影響旗標: 無
*************************************************************************
[EX]      RTS
執行前    STACK=$200,PC=$030
執行後    STACK=$200,PC=$200
```

****
* 指令型式: JMP $aaa
* 指令說明: 跳至 aaa 執行程式
* 運算情形: PC←aaa
* 影響旗標: 無
****
[EX]　　JMP $1FC
執行前　PC=$000
執行後　PC=$1FC

****
* 指令型式: JPC $aaa
* 指令說明: 如果進位旗標=1 跳至 aaa 執行程式,進位旗標=0 則執行下一條指令
* 運算情形: If C=1 then PC←aaa else PC←PC+1
* 影響旗標: 無
****
[EX]　　JPC $1FF
執行前　PC=$020,C=0
執行後　PC=$021,C=0

[EX]　　JPC $1FF
執行前　PC=$020,C=1
執行後　PC=$1FF,C=1

****
* 指令型式: JPZ $aaa
* 指令說明: 如果零旗標=1 跳至 aaa 執行程式,零旗標=0 則執行下一條指令
* 運算情形: If Z=1 then PC←aaa else PC←PC+1
* 影響旗標: 無
****
[EX]　　JPZ $1FF
執行前　PC=$020,Z=0
執行後　PC=$021,Z=0

[EX]　　JPZ $1FF
執行前　PC=$020,Z=1
執行後　PC=$1FF,Z=1

7. 副程式的介紹

● Clear User RAM

我們撰寫程式在進入程式正式執行之前,都會先將 RAM 的值都清除為 0 確保 RAM 中的值,避免程式因使用未經過初始化的 RAM 而發生程式執行錯誤,此一原因是單片機在上電復位(Power On Reset)後不會以硬件去清除 RAM,所以無法確知 RAM 的值為何(一般以亂數稱之),下面程式使用 T4x6N 的指標定址方式將 RAM 位址$013 到$04F 都清除為 0.

```
CLRRAM:   SDP                ;enable DP auto increase
          LDP    $013        ;initial DP=013
CLRAM1:   STX    #$0,DP      ;(DP)←0
          CMP    #$5,DPM     ;DPM-5
          JPC    CLRAM1      ;if DPM<5 (DP<050)jump to CLRAM1
          CDP                ;disable DP auto increase
          RTS                ;return to Main Program
```

在 T4x6N CPU 位址 $000 為一個虛擬的指標暫存器,也就是說此一位址沒有暫存器,而是利用 DPH,DPM,DPL 組合而成的 DP 當作間接位址的指標,當對位址 $000 讀寫實際上是對 DP 中所指的位址讀寫,所以在使用指標間接定址的方式做運算時,必須先設定 DP 的值,要設定 DP 可以使用 STX #$n,DPH STX #$m,DPM 及 STX #$r,DPL 來分別設定 DP 中的 DPH,DPM,DPL 暫存器,亦可使用 LDP $nmr 指令將 DPH,DPM,DPL 三個暫存器一次設定完成.

此外對 DP 讀寫 T4x6N 提供一個自動增量功能,若在對 DP 讀寫之前先下達 SDP 指令,即設定 DP 為有自動增量功能,爾後對 $000 讀寫或運算完畢時 DP 都會自動加一,此一功能讓我們在對連續位址的資料處理更為方便,要停止自動增量功能只需下達 CDP 指令即可.

● Fill User RAM

在許多地方需要將連續位址的RAM設定為某一個特定值,例如開機時要將所有 LCD 顯示的RAM設定為 F,讓所有的 LCD 點全部點亮,以檢測 LCD 的連接是否正確,下面的程式將 RAM 位址$013到$01F 都設定為 F 來達到此一目的.

```
FILLCD:   LDP    $013        ;initial DP=013
          SDP                ;enable DP auto increase
          STX    #$F,ACC     ;Acc←F
FILCD1:   STX    DP          ;(DP)←Acc
          CMP    #$2,DPM     ;DPM-2
          JPC    FILCD1      ;if DPM<2 (DP<020)jump to FILCD1
          CDP                ;disable DP auto increase
          RTS                ;return to Main Program
```

● Clock Increase

在程式中做時鐘的計時及顯示是常有的情況,一般單片機內部電路設計會使用外掛 32768Hz 晶振(Crystal)產生準確的時間基本時脈,提供固定時基(Time Base)或可程式計數器(Timer/Counter)來產生所需要的時間,讓程式可以準確的計算時間,達到計時顯示的目的.

以下為一般 12 小時的時鐘計時程式,此程式每一秒呼叫一次,其中秒及分的變數為十進制表示,時變數為十六進制表示,做法提供讀者參考:

```
CLKINC:   ADD     #$1,SECLBF,M   ;SECLBF←SECLBF+1
          CMP     #$A,SECLBF     ;SECLBF-A
          JPC     INCRET         ;if SECLBF<$A jump to INCRET
          STX     #$0,SECLBF     ;SECLBF←0
          ADD     #$1,SECHBF,M   ;SECHBF←SECHBF+1
          CMP     #$6,SECHBF     ;SECHBF-6
          JPC     INCRET         ;if SECHBF<$6 jump to INCRET
          STX     #$0,SECHBF     ;SECHBF←0
          ADD     #$1,MINLBF,M   ;MINLBF←MINLBF+1
          CMP     #$A,MINLBF     ;MINLBF-A
          JPC     INCRET         ;if MINLBF<$A jump to INCRET
          STX     #$0,MINLBF     ;MINLBF←0
          ADD     #$1,MINHBF,M   ;MINHBF←MINHBF+1
          CMP     #$6,MINHBF     ;MINHBF-6
          JPC     INCRET         ;if MINHBF<$6 jump to INCRET
          STX     #$0,MINHBF     ;MINHBF←0
          ADD     #$1,HURLBF,M   ;HURLBF←HURLBF+1
          CMP     #$D,HURLBF     ;0<HURLBF<D (1~12)
          JPC     INCRET         ;if HURLBF<D jump to INCRET
          STX     #$1,HURLBF     ;HURLBF←1
INCRET:   RTS                    ;return to Main Program
```

在 T4x6N CPU 中的 CMP 指令就是做減法動作,其結果不會存到 Acc 或是記憶體中,但是會影響進位旗標(C)及零旗標(Z)用以提供給程式來做比較判斷用,請注意減法動作中的減數及被減數二者的關係以避免錯誤,例如上述程式中的 CMP　#$A,SECHBF 其動作為 SECHBF-A 若是改為使用另一個比較指令 CMP　SECHBF 其動作則為 SECHBF-Acc 要特別注意!

比較結果影響旗標的狀況列表如下

比較結果	進位旗標(C)	零旗標(Z)
A-B<0	1	0
A-B=0	0	1
A-B>0	0	0

● Count Up/Count Down Timer

在撰寫程式中另外一個時常用到的程式為上下數計數器,下列的上下數計數器副程式中,暫存位址必須是連續的,計數位數最大可達到 16 位數,計數方式是以十進制來表示.

在 RAM 中每一位數有 4bit 可以表示 0~F 十六個數目,但是十進制只使用了前段 0~9 的十個數,當上數(增量)到 A 時要加上偏量(offset)6 來補償,也就是說當增量到 A(十六進制)時,實際表示應該為 10(十進制),加上偏量 6 就可以使數值都在 0~9 範圍內表示,做法為先將計數位元加 6,若沒有進位表示未超過 A,必須再減掉偏量 6 還原,有進位表示已超過 A 就不需將已加的偏量減回,所以跳過減 6 指令.

下數(減量)則使用加二的補數-1 的方式來達成,即將每一個位數加上 F,當進位旗標為 0 時,表示此一位數向上一位數借 1,從上一位借位到下一位時為 16,但實際應該為 10,所以當有借位時要減掉偏量 6 來補償回來,當進位旗標為 1 時,表示不需向上位借位,亦就是數值仍在 0~9 之間,所以要跳過減 6 指令.

由上述上下數中判斷進位旗標來決定減 6 與否有一致性,所以可將上下數計數寫成同一段程式,由設定 TB1 來處理加 6(上數)或加 F(下數),執行時將每一位的計數位元與 TB1 相加,判斷進位旗標決定是否減 6 調整結果,來達到上下數的動作,在進入此一副程式之前必須先設定下列暫存器及旗標,來定義計數器的起始位址與位元數及上下數的偏量方可正確計數.

暫存器及旗標	上數計數器	下數計數器
Carry Flag	1	0
TB1	6	F
TB2	計數器的計算總位數	
DP	計數器的起始位址	

```
CNTIN:    LDA     TB1           ;Acc←TB1
          ADC     DP,A          ;Acc←(DP)+Acc+CY
          JPC     CNTLP         ;Acc>F  jump to CNTLP
          SUB     #$6,ACC,A     ;Acc←Acc-6
CNTLP:    SDP                   ;enable DP auto increase
          STX     DP            ;(DP)←Acc
          CDP                   ;disable DP auto increase
          SUB     #$1,TB2,M     ;TB2←TB2-1
          JPZ     CNTRET        ;TB3=0  jump to CNTRET
          JMP     CNTIN         ;TB3<>0  jump to CNTIN
CNTRET:   RTS                   ;return to Main Program
```

- 修改紀錄：
  2014/05/27 – Page 6 .EQ 定義的數值不論有沒有加#，皆視為相同的數值，無法做為後續指令編譯時，判斷是 RAM 的位址或是常數的依據。