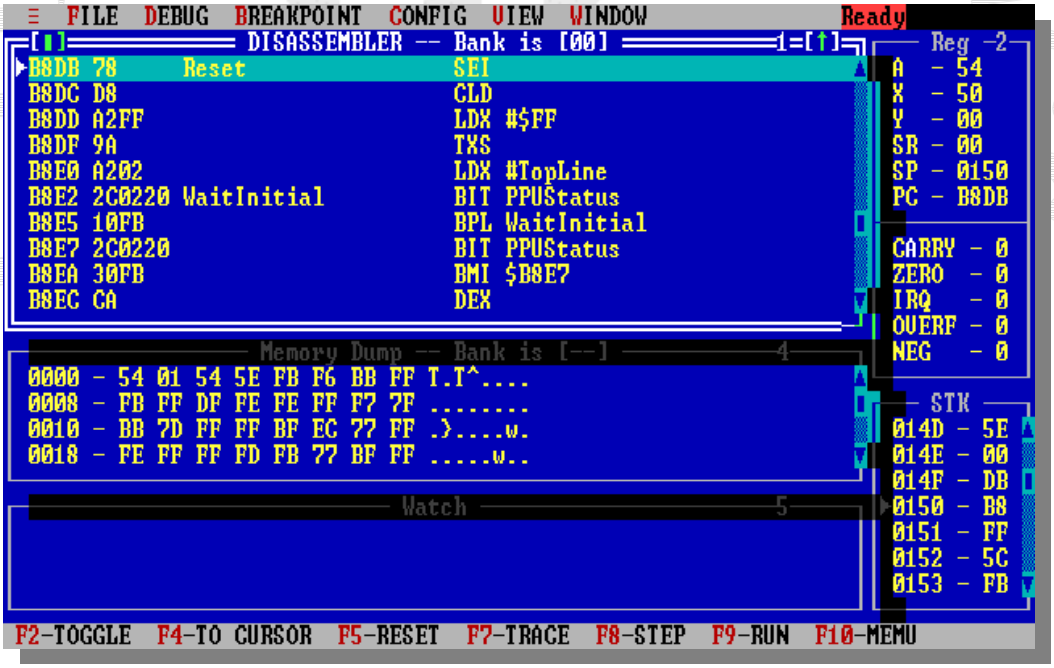
# *Symbol Debugger (S.D.)*

# 1-1 Symbol Debugger 軟體簡介

## *為何需要 Symbol Debugger 軟體*

軟體工程師在軟體開發的過程中，除了需要絞盡腦汁構思程式架構之外，最大的夢魘莫過於程式完成之後的測試與除錯(debug)工作。此時軟體工程師除了需要一套完整的發展系統硬體，還需要搭配一套好軟體，才能幫助 Debug 工作順利進行。可是一般的 Debug 軟體，大多僅具備模擬 ROM 的硬體功能，有限的程式斷點控制功能，以及將 ROM 內的資料予以反組譯後，將程式以助憶碼的型態呈現給軟體工程師。但是，這樣簡單的功能在程式不大或是程式內部含有許多副程式及資料結構的情況下還好，一旦程式變複雜，或是程式內含有許多公用副程式，或是程式內帶有大量的結構化資料庫時，這種簡單的軟體反而變成 Debug 動作的最大困擾，因為所有的資料都變成助憶碼，反而使程式更難看懂，也跟原始程式相差更遠。

為了解決這種 Debug 工作上的困擾，讓軟體工程師能夠更輕易的開發 NT6576/NT6578 系列微處機專用的軟體，並充分發揮該系列微處理機所具備的強大功能，NT657X 發展系統特別開發一套新的軟體提供給軟體工程師使用→ Symbol Debugger 軟體。




Symbol Debugger – 顧名思義就是能夠提供以 Symbol 方式將使用者的程式完整的表現在發展系統上。包含使用者自己定義的變數名(Variable Name)，副程式名稱(Subroutine Name)，供識別用的標籤(Label)等等，讓軟體工程師在 NT657X 系列的發展系統上作 Debug 動作時，就像是以原始程式在作 Debug 動作一般，不但親切也容易識別，待找出問題後也容易對照原始程式進行修改。

# *Symbol Debugger 軟體的特點*

為了配合 NT6576/NT6578 系列微處理機強大的圖形處理與周邊緣件控制能力，並且幫助軟體開發者順利發展 NT6576/NT6578 系列微處理機專用的軟體，所以 NT657X 系列發展系統提供的 Symbol Debugger 軟體具備以下的多項特色。

## 以 Symbol 作反組譯(Disassembly)

SD 軟體在作 Disassembly 動作時，除了能將存在於 NT657X ICE 記憶體內的軟體資料以助憶碼的型態呈現出來，還能接收由 Cross Assembler 軟體所產生的 Symbol 資料。使用者只要將這些 Symbol 資料 Load 進 SD，SD 就會根據各個 Symbol 的位址，自動與 NT657X ICE 內的資料作比對，只要位址或是數值相符，不論是副程式的 Label，變數的 Variable Name，甚至常數的名稱(Constant Name)，都可以一一轉換出來，讓使用者就像在對 Source Program 作 Debug 動作一般。

注意：**SD** 所使用的 **Symbol** 格式，只接受 **Zax** 格式的 **Symbol File**。

除了反組譯功能會以 Symbol 來顯示之外，SD 還接受使用者以 Symbol 型態來作各種輸入或是 Search 的動作，例如依據 Subroutine Name 將程式移到某個副程式，或是依據 Variable Name 找到某個變數等。

## On Line Assembly 能力

SD 同時還提供立即的 Assembly 功能。當使用者在 Debug 階段並發現某些可疑的錯誤，希望修改某些指令以測試推論正確與否時，SD 接受使用者以助憶碼的型態輸入 6502 的指令，以及用 Symbol 來輸入 Variable Name 或是 Subroutine Name，然後 SD 會自動將這些指令轉換成正確的機械碼(6502 Machine Code)，因此使用者不必再去記那些煩人 6502 Machine Code。

## 多重視窗(Multi Window)

為了讓使用者在 Debug 動作時能夠更輕易的掌握軟體的執行狀態，以及 NT6576/NT6578 微處機內部的情形，SD 提供了多達十種以上的 Window，以提供充分的資訊供使用者參考或控制。這些 Window 依功能不同而分別有：Disassembly Window，Memory Dump Window，Watch Variable Window，CPU Status Window，CPU Stack Window，VRAM Dump Window，PPU Register Window，Palette Window，Peripheral I/O Register Window，Break Point List Window，Joystick Status Window 等。

上述這些 Window，都可以任由使用者控制調整大小或是關閉，或移動位置，或是多個 Window 同時存在。

## 中斷點(Break Point)控制

NT657X ICE 本身提供一組 Hardware Break Point 的控制功能，能夠讓使用者設定各種複雜的觸發(Trigger)條件，以決定在何種情況下 ICE 需暫停執行狀況，讓使用者進行 Debug 動作。

除了硬體提供的 Break Point 控制，SD 另外提控由軟體控制的 Software Break Point 功能。與 Hardware Break Point 作比較，Software Break Point 雖然會影響 NT657X ICE 執行使用者軟體的速度，但是這種影響很輕微，相對的 Software Break Point 能提供更多的中斷點設定位置。以 SD 而言，她所能提供的 Software Break Point，幾乎沒有限制(至少可以同時設定 20 組以上)。而且 Software Break Point 的設定條件遠比 Hardware Break Point 簡單，設定方式也更方便。

## 程式追蹤執行(Trace)能力

SD 提供了三種程式追蹤執行模式。

"Trace"模式：一般的單部執行(Single Step)，每次僅執行一個指令；

"Step"模式：具有自動執行 Subroutine 能力的 Single Step;

"Goto"模式：自動由 6502 的 Program Counter 執行到 Cursor 所在位置；

有了以上三種追蹤執行模式，再配合 Break Point 的設置功能，相信 SD 將能幫助使用者很容易的進行 Debug 工作。

## 立即更新的變數與 CPU 狀態

當 SD 處於追蹤執行的監控模式時，只要軟體的執行狀況一停止，SD 就會自動將各種資訊予以更新。例如 CPU 的狀態，CPU Stack 的使用狀況，各種 NT6576/NT6578 IC 內暫存器的狀況等等，甚至由使用者設定希望觀察的變數的現況，都能即時的加以更新，令使用者能完整的掌握整個程式在各個階段下的執行結果。

# *Symbol Debugger 軟體的環境需求*

為求符合當前電腦系統的演進，NT657X 發展系統所提供的 Symbol Debugger 軟體，雖然是屬於 MS DOS 模式的軟體，但是她仍然能在 Windows95 之下以 DOS Window 的方式直接執行而不會有任何問題。

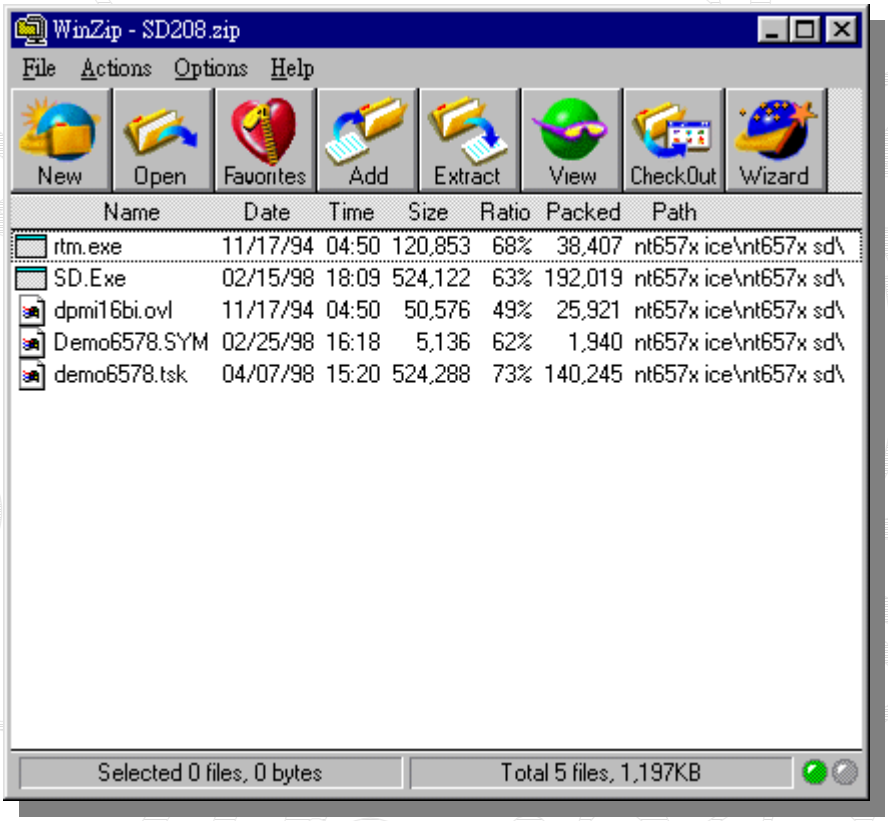
不論是 DOS 模式或是 DOS Window 模式，Symbol Debugger 軟體都僅需要 640K Bytes 的基本記憶體就可以執行。但是在 Symbol Debugger 軟體執行的時候，她會在 Symbol Debugger 程式所在的目錄，自行產生一個容量大約為?M Bytes 的暫存檔，因此用來執行 Symbol Debugger 所使用的硬碟，至少需要留下比?M Bytes 大的空間，才能讓 Symbol Debugger 順利的執行。當使用者正常結束 Symbol Debugger 的同時，Symbol Debugger 會自行刪除該暫存檔。

# *Symbol Debugger 基本操作說明*

## Symbol Debugger 軟體

Symbol Debugger 軟體隨 NT657X 發展系統交給使用者時，是以 WinZip 壓縮工具作成壓縮檔，使用者必須利用 WinZip for Windows95 的解壓縮程式才能解開。若使用者缺少 WinZip 的解壓縮工具，可至 Internet 上的WWW.WinZip.com站台 Download WinZip 解壓縮軟體。

利用 WinZip 解壓縮工具將 Symbol Debugger 解壓縮之後，其內容應該至少包含五個軟體。




SD.Exe – Symbol Debugger 軟體的主要執行程式。
DPMI16bi.ovl – Symbol Debugger 軟體內的各項副程式集。
RTM.Exe – Symbol Debugger 軟體所使用的 Run Time Utility。
Demo6578.Tsk – NT657X 系列微處理機中的 NT6578 示範程式。
Demo6578.Sym – NT6578 示範程式所使用的 Symbol 資料。

使用者要啓動 Symbol Debugger 軟體時，僅需在 Windows95 的視窗中，以 Mouse 的左鍵連續按二下 SD.Exe，即可開始執行 Symbol Debugger 軟體。

## 輸入工具

Mouse 與 Keyboard 同樣是 Symbol Debugger 的基本輸入工具，Mouse 雖然是選用配備，不過最好還是有配備 Mouse，對軟體的操作會比較方便。

整個 Symbol Debugger 軟體雖然是以 DOS 模式爲基礎，但是她的操作環境仍是以 Mouse 爲基本的輸入工具，而且 Symbol Debugger 還盡量學習 Window 操作的優點，每一種資料都獨立成一個 Window，而該 Window 不但可以 Open 或 Close，還可以 Move 或是改變 Window Size。所有操作指令也都做成 Pull Down 式的選單，使用者不但能以 Mouse 來選取，也可以利用 Hot Key 來加快操作速度。

在整個 Symbol Debugger 軟體之中，在 Mouse 的運作上會依循下列原則。

**I.** Mouse 的左鍵代表選取或是執行的功能。

按一下 Mouse 左鍵，將選取相對的功能或視窗，若是連按二下，則代表執行所選定的功能，或是進入資料輸入模式。

**II.** Mouse 的右鍵代表開啓各 Window 的附屬附屬功能選單。

每一個 Window 都擁有各自的的附屬附屬功能選單，在開啓附屬功能選單的選單之後，需移動 Mouse 到所想要的功能上，再按一次 Mouse 左鍵，才會開始執行該項功能。

# Symbol Debugger 軟體與 NT657X 發展系統

在 Symbol Debugger 的環境下，NT657X ICE 會受 Symbol Debugger 的控制，此時 NT657X ICE 面板上的 Reset 鍵會暫時失去作用。若想 Reset NT657X ICE，必須由 Symbol Debugger 軟體上下 Reset 命令(F5 鍵)才會使 NT657X ICE Reset。但是當 Symbol Debugger 軟體結束(Alt + X 鍵)執行時，則會自動 Reset NT657X ICE 一次，接著就使 NT657X ICE 處於自由運行(Free Run)狀態，此時 NT657X ICE 面板上的 Reset 又會恢復功能。

啓動 Symbol Debugger 軟體時，S.D.軟體會透過 PC 的 Parallel Port 查詢 NT657X ICE 的狀態，並檢查介於二者之間的 Security Key 是否存在，若是一切正常，則會進入 S.D.軟體的開機畫面並在右上角顯示 Ready 字樣。若否，則自動結束 S.D.的軟體執行並回到 DOS Prompt 狀態。

Symbol Debugger 軟體運作過程中，會不停的查詢 NT657X ICE 的狀態，若是在執行過程中發現通訊有問題，或是 NT657X ICE 發生異常狀況，則 S.D.會在右上角顯示 Error 字樣。

注意：當 **Error** 發生時，請不要再嘗試執行任何指令，應該以 **Reset** 命令**(F5** 鍵**)**使 **NT657X ICE** 重置，並讓 **S.D.**軟體與 **NT657X ICE** 重新建立連線。

## 資料輸入

在 Symbol Debugger 軟體運作時，有許多機會需要使用者輸入一些數據，例如 Address、Data 或是 Symbol Name 等。爲了讓使用者在輸入數據時，不必刻意區分 16 進制數據或是 Symbol Name，因此 Symbol Debugger 採用下列準則來判斷使用者輸入的資料。

a. Symbol Debugger 只接受使用者輸入 16 進制(Hex)的數據。

b. 若是輸入的數據是以 "$" 作引導，則 Symbol Debugger 會將此數據當成是 16 進制的數值來處理。

Example： $ABCD = $ABCDh

c. 如果輸入的數據並沒有 "$" 作引導，但是開頭的第一個字是介於 0~9 之間的數字，則 Symbol Debugger 仍會當成 16 進制的數據來處理。

Example： 0ABCD = $ABCDh
1234 = $1234h

d. 如果輸入的數據並沒有 "$" 作引導，但是開頭的第一個字不是介於 0~9 之間的數字時，Symbol Debugger 會將輸入的資料當成 Symbol Name 來處理。

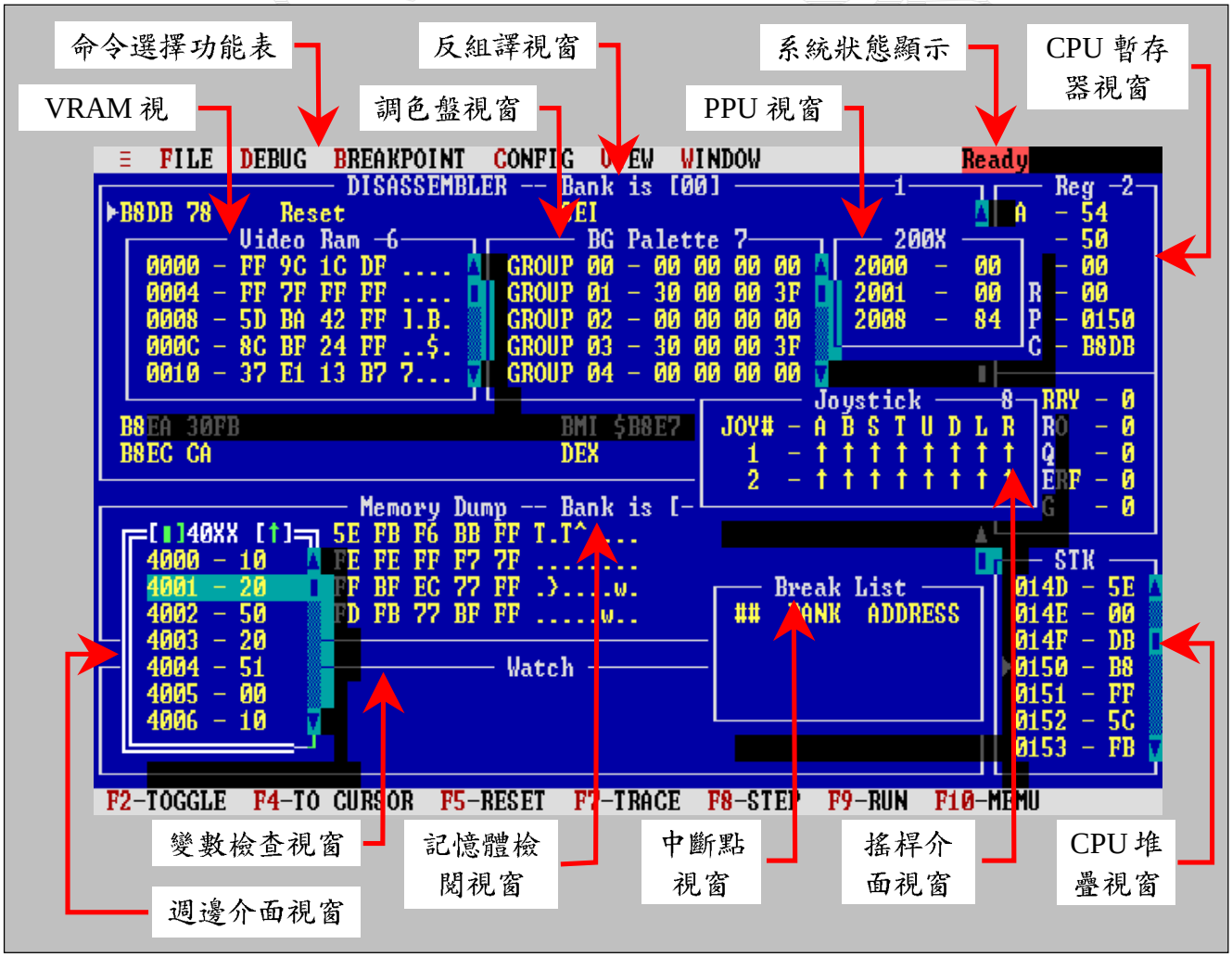
Example： ABCD = "ABCD" (Symbol or String)

e. 如果輸入的資料被當成 16 進制數據，但是資料中含有不是 0~9 的數字同時也不介於 A~F 之間的英文字，則 Symbol Debugger 會認為是使用者資料輸入錯誤。如果輸入的資料被當成 Symbol 處理，則沒有輸入字元使用上的限制。

# 1-2 視窗之功能與解釋

## 視窗種類

在順利進入 Symbol Debugger 軟體之後，可看到 SD 的一般操作畫面，其中包含了八個主要的操作單元，以及六個次要單元需要使用者下令之後才會開啓。本節就先對這些視窗作一個簡要說明，以便使用者能夠了解何種狀況下該使用哪一種功能，至於詳細的使用說明，則請使用者參閱後續的各節。




## 命令選擇功能表 (Function Selection Bar)

該 Function Selection Bar 的功能就與 Windows 內的下拉式選單(Pull Down Menu)一樣，只要使用者利用 Mouse 將 Cursor 移到所要使用的功能，然後按一下 Mouse 上的左鍵，該功能就會自動打開更進一步的選單供使用者選擇。當然，若使用者是以按熱鍵(Hot Key)的方式，一樣也能打開 Pull Down Menu。

整個 Function Selection Bar 共分成六大類，相關的詳細說明請參閱 Page 17 – “1-3 命令選擇功能表(Function Selection Bar)”一節。

# 系統狀態顯示(System Status Indicator)

System Status Indicator 所顯示的是 NT657X ICE 的狀況，以及 SD 軟體透過電腦的 Parallel Port 與 NT657X ICE 通訊的狀態。

當通訊一切正常時，該處會顯示 Ready 字樣。若是通訊有問題時，則會顯示 Error 字樣。當使用者使用 Trace 功能或 Step 功能時，該處也會有相對的字樣顯示，以便使用者能確實掌握 NT657X ICE 當時的操作狀態。相關的詳細操作說明請參閱 Page 45 - “1-4 系統狀態顯示(System Status Indicator)” 一節。

# 反組譯視窗(Disassembly Window)

Disassembly Window 可以說是整個 Symbol Debugger 軟體最重要的部分，她負責將儲存在 NT657X ICE 內的資料，以助憶碼加 Symbol 的形式反組譯在使用者的面前。

Disassembly Window 除了顯示功能之外，使用者也可以利用這個 Window 來設立或刪除 Software Break Point，使 Debug 的工作更容易完成。同時使用者也可以利用這個 Window 直接輸入新的指令。相關的詳細操作說明請參閱 Page 47 - “1-5 反組譯視窗(Disassembly Window)”一節。

# CPU 暫存器視窗(CPU Register Window)

CPU Register Window 所顯示的是 NT6576/NT6578 內所包含的 6502 CPU 的運作狀態，其內容包含了 CPU 的 A, X 及 Y 三個暫存器，以及 Program Counter(PC)，CPU Status Flag 等重要資訊。該 Window 另有更進一步功能，其使用說明請參閱 Page 55 – “1-6 CPU 狀態視窗(CPU Status Window)”一節。

# 變數檢查視窗(Watch Variable Window)

變數檢查視窗(Watch Variable Window)是 NT657X 發展系統所提供的一項特殊的功能 – 隨時檢查使用者變數(User Variable)的狀態，該功能可以讓使用者更輕易的掌握程式中所用的的重要變數的變化情形。

藉由該項功能，使用者可以設定所想要觀察的 Variable，不論是 1 Byte, 1 Word 或是連續數個 Bytes 的記憶體變數，甚至連 NT6576/NT6578 IC 的 I/O 暫存器，都可以在設定之後由 Watch Variable Window 中加以觀察，或是由該處直接設定變數的內含值。有關更詳細的使用說明，請參閱 Page 59 – “1-7 變數檢查視窗(Watch Variable Window)”一節。

# 記憶體檢閱視窗(Memory Dump Window)

Memory Dump Window 乃是用來將存在於 NT657X ICE 內的資料，以 16 進制碼(Hex)及轉換成 ASC II 格式之後的符號，呈現給使用者觀察。該 Window 最主要的作用，在於幫助使用者檢查那些不屬於程式指令的資料，例如圖形的 PGT 或是 PNT Data，或是一些程式中需要用來顯示訊息的 ASC II 字串(String)，也或者可能是一些 Hex 的數據資料庫等等。

透過 Memory Dump Window，使用者可以觀察 NT657X ICE 的任一記憶體位址(由 $0000h~$FFFFh 以及 Bank 0~Bank 31)，但是唯獨 VRAM 例外，因爲 VRAM 部分另外有 VRAM 的 Dump Window。相關的使用細節請參閱 Page 63 – “1-8 記憶體檢查視窗(Memory Dump Window)”一節。

# 堆疊視窗(CPU Stack Window)

CPU Stack Window 乃是專門設計，並用來顯示 6502 CPU 所使用的 Stack(位址在 $0100h~$01FFh)的狀況及其內含值，好幫助使用者在遇到程式有 Bug 時，能夠了解 CPU 的運作歷史，以及 Interrupt 的執行情形。

CPU Stack Window 是一個特殊的 Window，她僅能顯示位於$0100h~$01FFh 之間的 256 Bytes 空間，其餘部分的資料，都無法透過該 Window 來檢查。使用細節請參閱 Page 68 – “1-9 CPU 堆疊暫存器視窗(CPU Stack Window)”一節。

# 視訊記憶體視窗(VRAM Dump Window)

VRAM Dump Window 的功能與 Memory Dump Window 類似，所能操作的指令也相同，資料的顯示方法也一樣。唯一的差別在於 Memory Dump Window 所顯示的記憶體範圍在 CPU 所控制的 64K Bytes 的區域內(包含 32 個 Bank)，而 VRAM Dump Window 所顯示的記憶體，則是由 PPU 所控制的 64K Bytes 區域。相關的使用細節，請參閱 Page 71 – “1-10 視訊記憶體視窗(VRAM Dump Window)”一節。

# 調色盤資料視窗(Color Palette Window)

Color Palette Window 所顯示的內容，就是 NT6576/NT6578 微處理機內的調色盤。由於調色盤的內容會直接影響到電視畫面上輸出的色彩，爲了讓使用者能夠很輕易的透過發展系統調整電視的輸出效果，以求取最佳的視覺效果，因此特別在 S.D.內附加此一獨立功能，讓使用者能夠更直接的調整調色盤。

Color Palette Window 顯示資料的方式，僅以 Hex 資料格式顯示出來，而且是以每 4 個顏色爲一行，總共以 16 行來顯示出全部 64 個顏色的暫存器。更進一步的使用方法，請參閱 Page 76 – “1-11 調色盤資料視窗(Color Palette Window)”一節。

# PPU 暫存器視窗(PPU Register Window)

NT6576/NT6578 系列微處理機最重要的部份就在於圖形顯示的處理能力，而這其中又以 PPU 的一系列控制暫存器的作用最爲顯著。無論是背景的捲繞(Background Scrolling)、或是動畫顯示與處理(Sprite Process)都需透過 PPU 的 Control Register 才能辦到。

但是很不幸的，軟體在作圖形處理時，最容易出問題的部分也在於 PPU Control Register 的設定控制。不論是填值的時機不對(例如在 NMI 以外的時間改變 Sprite)，或是所填的值錯誤(例如將 16 色模式設成 4 色模式)等，都會使畫面輸出受到大的影響，但是程式的運作卻又看來正常。

因此在 S.D.軟體中，特別增加了 PPU Register Window，它可以讓使用者更輕易的知道 PPU 的設定狀況，或是藉由 PPU Register Window 直接調整 PPU Register 的內含值，使 Debug 的工作更容易進行。

PPU Register Window 僅顯示$2000h~$2008h 這段範圍內的 Register，而且其中有些 Write Only 的 Register 也不會出現在 PPU Register Window 之中。有關更詳細的操作說明，請參考 Page 77 – “1-12 PPU 暫存器視窗(PPU Register Window)”一節。

# 週邊介面視窗(Peripheral I/O Window)

Peripheral I/O Window 的作用與 PPU Register Window 相同，只不過顯示範圍是界定在$4000h~$43FFh 之間的 Super I/O 介面上。而其目的也是爲了讓使用者在作 Super I/O 的介面溝通時，能夠有一個方便且獨立的視窗供使用者利用。相關的細節操作說明，請參考 Page 79 – “1-13 週邊介面視窗(Peripheral I/O Window)”一節。

# 中斷點視窗(Break Point List Window)

Break Point List Window 會將使用者設定過的所有軟體中斷點(Software Break Point)全部列表在本 Window 之中，以供使用者透過本 Window 來決定某一 Software Break Point 是否要繼續使用或是加以刪除。

考慮到使用者在 Debug 的過程中，經常會利用 Break Point 的功能。但是當 Software Breakpoint 設多了之後，使用者本身也很容易就會忘記設了哪些 Software Breakpoint。此時若想刪除某一特定的 Breakpoint 時，又該如何做呢？有鑑於此種需求，Breakpoint List Window 特別列出所有 Software Breakpoint，供使用者檢查，也可以將不再需要的 Software Break Point 予以刪除，讓 Debug 過程更爲順暢與容易操作。詳細的操作說明，請參考 Page 81 – “1-14 中斷點視窗(Break Point List Window)”一節。

<Break Point List 的功能尚未符合需要的規格，暫時僅能供顯示用，不具備修改功能>

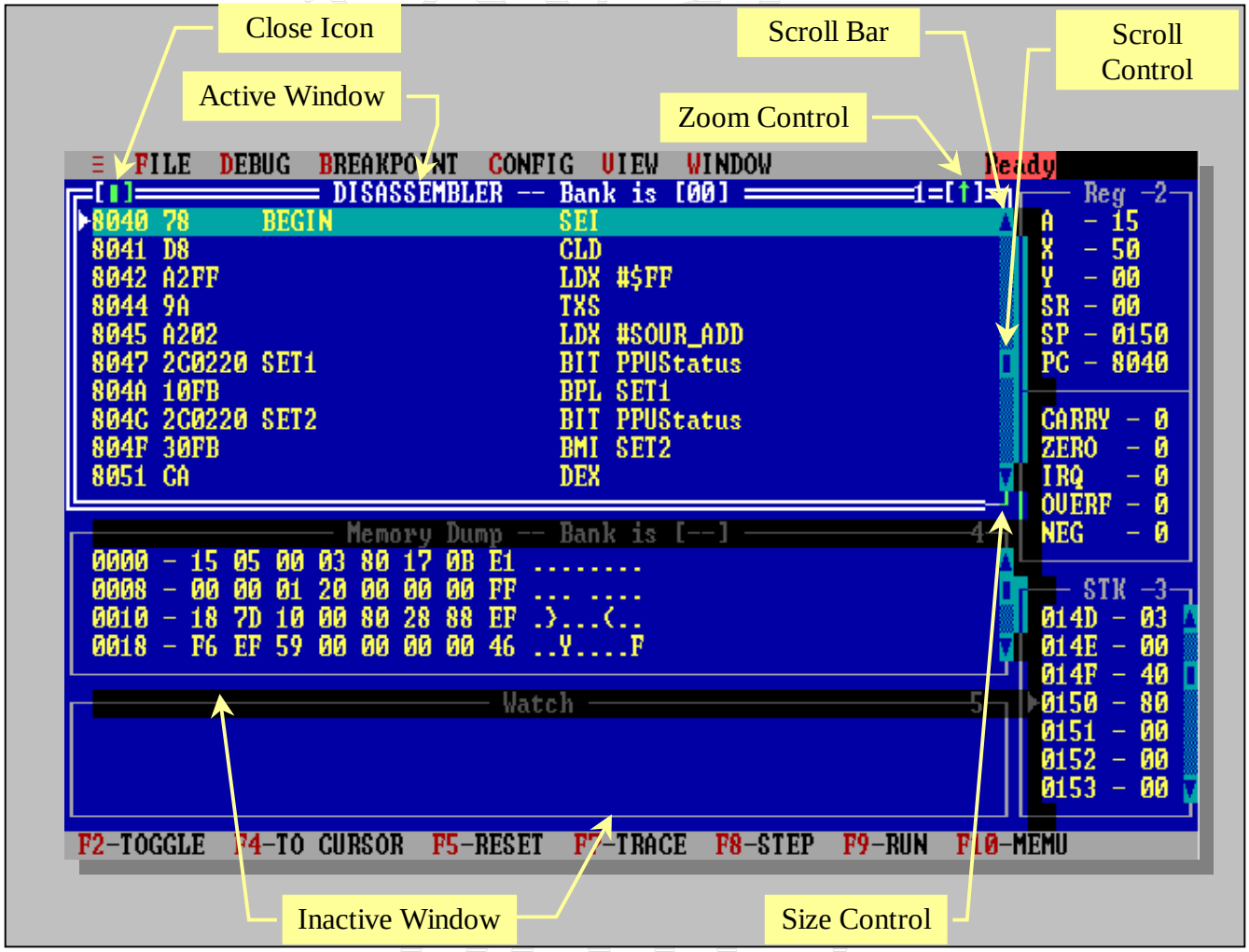
# 搖桿介面視窗(Joystick Register Window)

Joystick Register Window 也是基於讓使用者更方便使用的目的而設計的，雖然她的內容僅有 2 Bytes，但是在使用上，仍然能幫助使用者更輕易了解 NT6576/NT6578 微處理機上的 Joystick 的運作狀況。相關細節請參考 Page 84 – “1-15搖桿狀態視窗(Joystick Status Window)”一節。

<Joystick Window 雖然有顯示資料，但是無法反應 Joystick 的狀態，請暫勿使用>

# 視窗的操作

進入 Symbol Debugger 軟體之後，可以很輕易的發現 Symbol Debugger 裡面都是以類似 Windows 的操作介面，來提供使用者更容易接受的環境。其中包含了各 Window 的尺寸控制(Size Control)、捲動控制(Scroll Bar)、移動控制(Move)等，各種控制功能的使用基本上與 Windows 相同，以下則對使用方法加以說明。




## Active Window

所謂 Active Window 就是當時可以產生動作反應的視窗，由於 Symbol Debugger 允許顯示多視窗，但是同一時間又僅有一各視窗能夠產生反應，因此才會有 Active Window 及 Inactive Window 的區別。

被使用者選爲 Active 的 Window，其 Window 的邊框會有明顯且亮的雙白線，同時會出現 Close Icon、Size Icon 及 Zoom Icon 等記號，代表此 Window 此時可以接受這些調整控制。

# Inactive Window

在 Symbol Debug 中，除了唯一的一個 Active Window 之外，其他的都是 Inactive Window。Inactive Window 在 Symbol Debug 之中就像是背景一般，除了必要的資料更新動作以外，其他的各種 Window 調整動作都不會反應。

要使 Inactive Window 變成 Active Window，可以利用 Mouse 或是由 “View” 功能表中來改變。

# Zoom Control

“Zoom” 功能可以使 Window 由當時的尺寸變成最大，或是由最大變回先前設的大小。“Zoom” 功能僅有二種變化互相交替，但是不能任意調整尺寸大小。

# Size Control

“Size Control” 允許使用者藉著 Mouse 點選之後的 “拖-放”功能，來任意調整 Window 的尺寸。當使用者點選 “Size Icon”之後，只要 Mouse 上的按鍵不放開，整個 Window 的邊框會變成淺藍色，此時就可以拖著 Mouse 來決定該 Window 要變成何種尺寸。

“Size Control” 僅存在於 Window 的右下角，其他三各角落都不具備 “Size Control” 的功能。

# Move Control

要使用 “Move Control” 只要使用者以 Mouse 點選 Window 的最上緣，待 Window 的邊框變成淺藍色之後，再用 “拖-放” 的方式即可移動 Window 到任意位置。

# Scroll Bar and Scroll Control

Scroll Bar 與 Scroll Control 是一體的，使用者可以藉著 Scroll Bar 來控制捲繞動作，以便顯示更多的資料，而 Scroll Bar 的使用更可分成三種方式。

**I.** 以 Mouse 點選上/下箭號，可以讓 Window 內的資料以 Line Scrolling 的方式上下捲動，每次一行；

**II.** 以 Mouse 點選介於上/下箭號與 Scroll Control 之間的位置，可以讓 Window 內的資料以 Page Scrolling 的方式上下捲動，每次一個 Window Page；

**III.** 以 Mouse 直接點選 Scroll Control，並 “拖” 著 Scroll Control 上下移動，則可以是 Window 內的資料以及快的速度配合 Scroll Control 的相對位置來移動；

# Close Icon

以 Mouse 直接點選該 Icon，將使相對的 Window 被關閉，若要再打開此一 Window，則必須透過 “View” 功能表才行。

# 1-3 命令選擇功能表(Function Selection Bar)

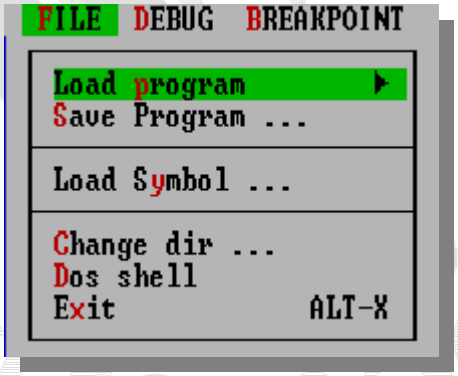
命令選擇功能表(Function Selection Bar)位於螢幕的最上面一行。SD 系統內的各個主要項目中都顯示在這表中，幫助使用者在不記任何命令的情況下仍舊可以很輕易的操作 Symbol Debugger 軟體。


Function Selection Bar 的命令一共分爲六大項目，其功能與相對的 Hot Key 定義列於下表。

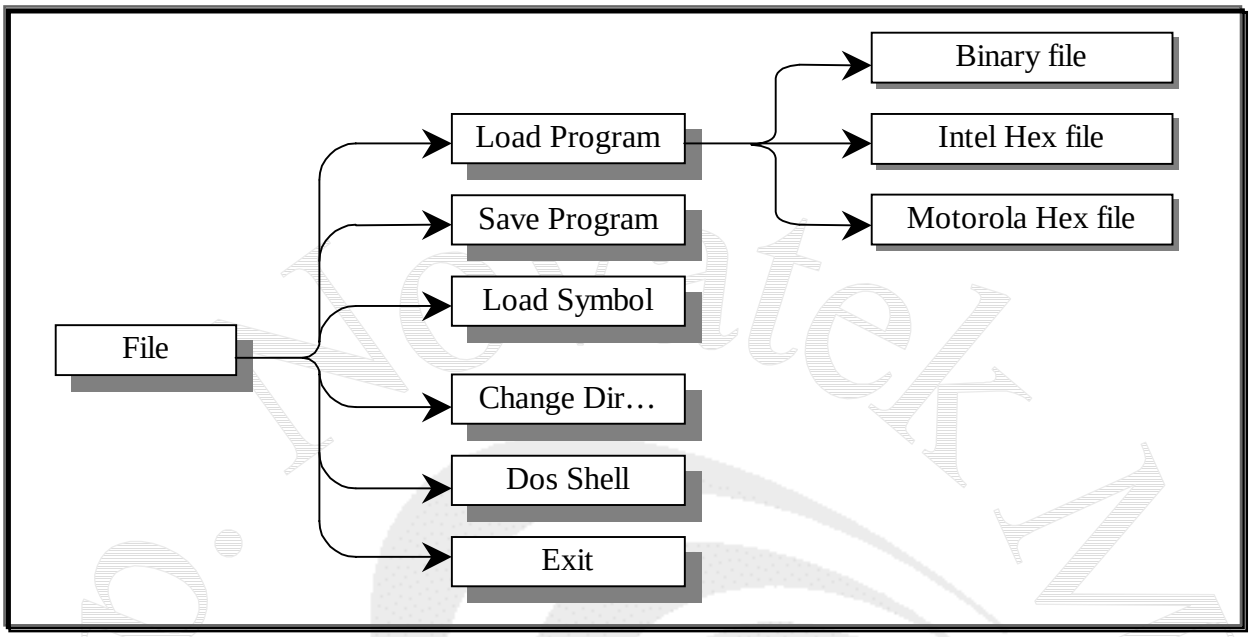
| 符號 | 功能定義 | Hot Key |
|---|---|---|
| ≡ | 系統版本訊息。 | |
| File | 檔案存取的命令組 | Alt + F |
| Debug | 控制 CPU 的執行模式 | Alt + D |
| Breakpoint | 軟/硬體中斷點的設定與解除 | Alt + B |
| Config | 容量設定選項 | Alt + C |
| View | 各視窗監視開關 | Alt + V |
| Window | 各視窗縮放選項 | Alt + W |

以下爲各功能的詳細解說：

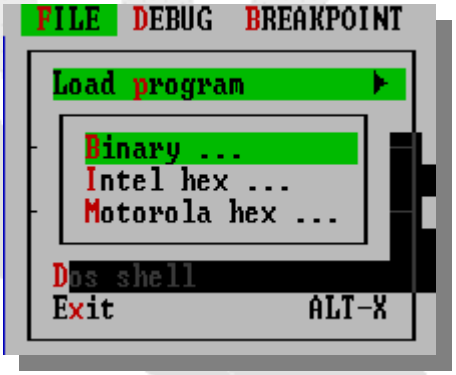
## *File*




File 功能內的動作都是與使用者程式的存取有關，整個 File 功能又細分成六個次要命令，其中的 Load Program 命令又依使用者編譯程式的習慣而再分成三種子命令。整個 File 功能的關係如下頁的附圖所示。




# Load Program

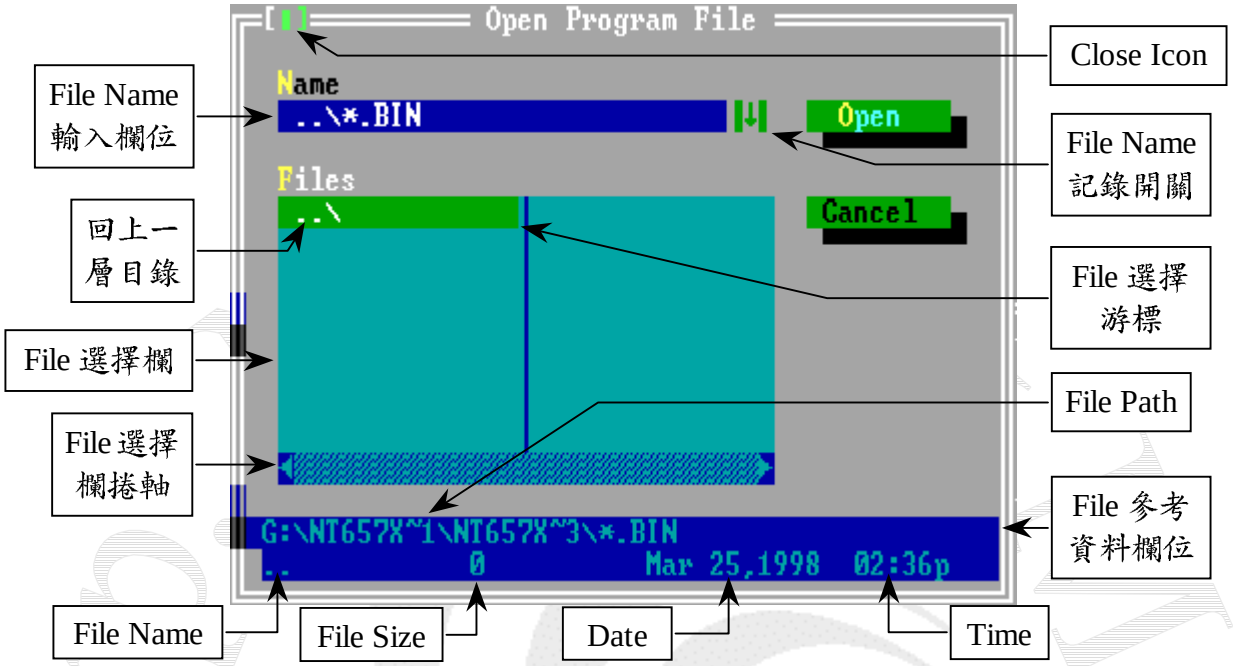



使用者可利用 Load Program 功能，將使用者所設計的程式 Down Load 到 NT657X ICE 的 ROM 模擬器(ROM Emulator)內，並讓系統執行使用者的程式或作 Debug 動作。當使用者選擇這個功能時，當使用者選擇 Load Program 功能之後，整個 Load Program 的動作又會分成二個階段 - File Open 與 Load Address 設定。

**I.** File Open 階段

在 File Open 階段，S.D.會再開啓 File Open Window，供使用者以選單來選擇所要 Down Load 的檔案。不論使用者是選擇 Load Binary、Load Intel Hex 或是 Load Motorola Hex，都是用同一個 Window，只不過預設的副檔名不同而已。選擇 Load Binary 時，預設的副檔名是*.Bin。選擇 Load Intel Hex 時，預設的副檔名是*.Hex。選擇 Load Motorola Hex 時，預設的副檔名則是*.S28。

當 File Open 的 Window 出現後，該 Window 中的各項功能都如上圖所標示。使用者可以利用鍵盤上的 Tab 鍵來切換，或是以 Mouse 控制 Cursor 到所要使用的功能，然後按一下 Mouse 左鍵來選擇所要使用的功能。至於各項功能的定義及使用說明則敘述如下。

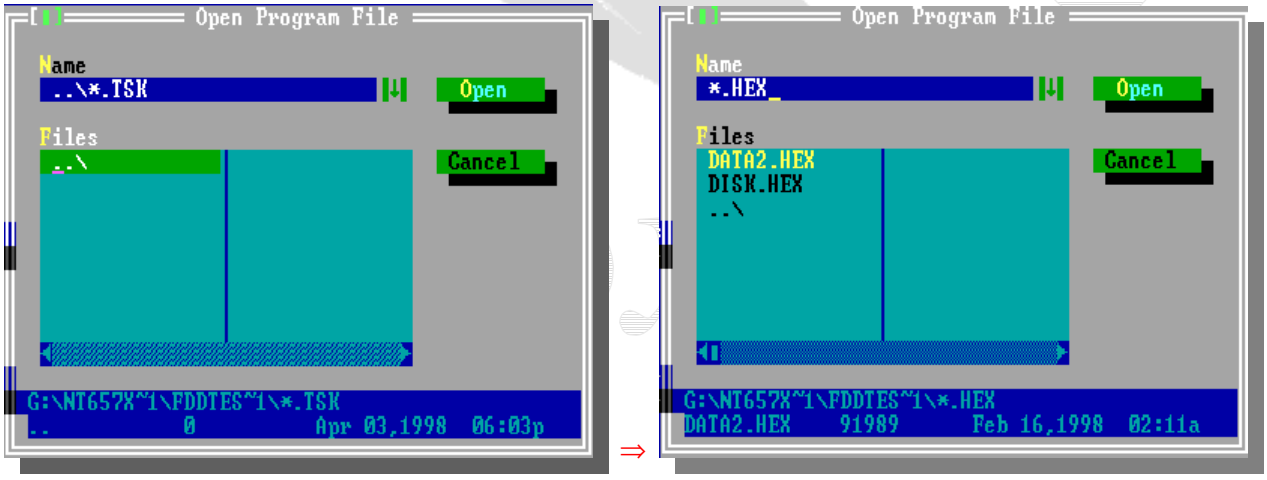



a. Close Icon

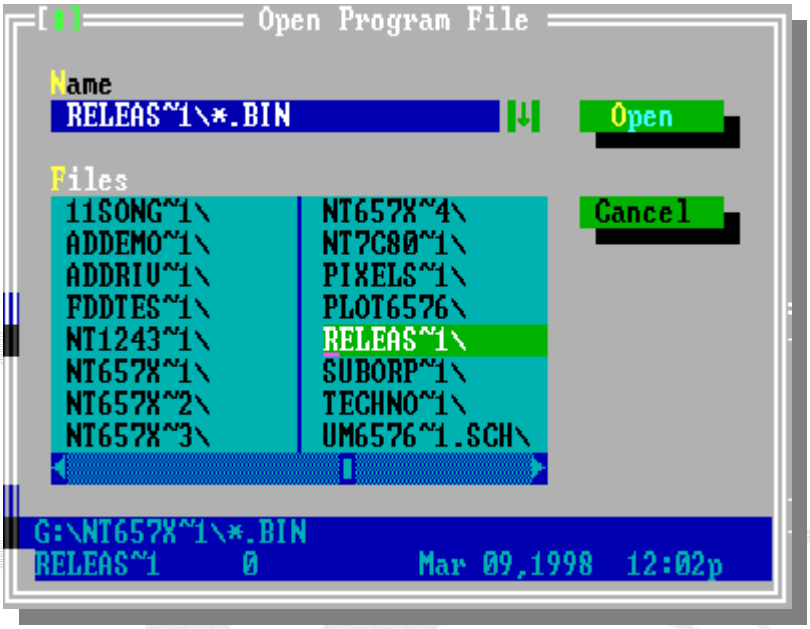
當使用者決定放棄此一子視窗而暫時不 Load File 時，能以 Mouse 按這個 Close Icon，而使該 Window 關閉。除了 Close Icon 之外，使用者按 Cancel 鍵或是按鍵盤上的 Esc 鍵也都可以放棄 Load File 的動作。

b. File Name 輸入欄位

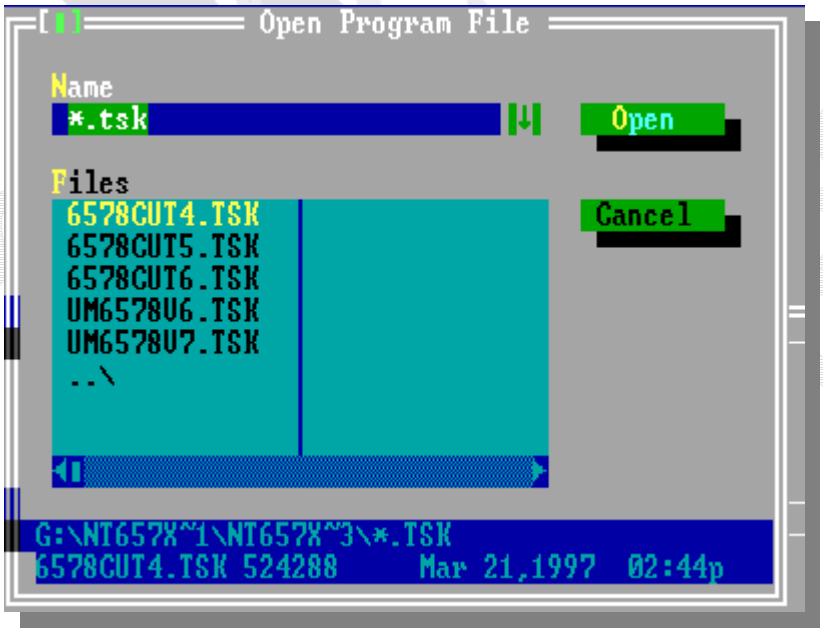
File Name 輸入欄位具有多種用途，可以用來讓使用者直接輸入已知的檔名，也可以用來改變目錄，更可以用來決定下方的 File 選擇欄在列出檔案名稱時，該以何種副檔名作爲選擇的依據。




或者使用者可以在此欄位輸入目錄的路徑名稱而不輸入檔案名稱，則 S.D.會將工作目錄直接移到使用者指定的目錄，並將該目錄內合乎副檔名要求的檔案名稱，全部顯示在 File 選擇欄內共使用者參考與選擇。




由於使用不同的 Cross Compiler 所編譯出的副檔名名稱不一定相同，例如某些 Compiler 會將 Binary File 的副檔名設成 *.BIN，而 2500AD 的 6502 Compiler 則會將編譯完的執行檔副檔名設爲 *.TSK。此時由於 S.D.所提供的預設副檔名與使用者所使用的副檔名不同，因此下方的 File Selection List 將不會出現任何婦檔名爲 *.TSK 的檔案供使用者選擇，在此種情況下使用者可以在此處將 *.BIN 改成 *.TSK，然後按下 Enter 鍵，就可以使 File Selection List 的內容改變。



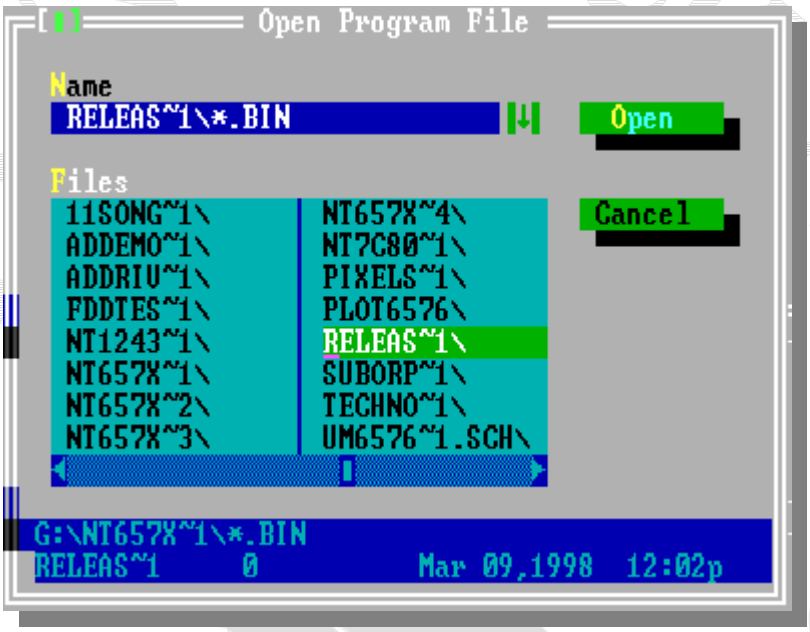

注意：在按下 **Enter** 鍵之後，**File Selection List** 有可能會回到上一層的目錄，此時使用者只要移動游標並改變目錄即可回到先前正在使用的工作目錄。

c. 回上一層目錄

任何時刻，使用者都可以透過此一記號，使 File 選擇欄內的工作目錄，往上移一層，以便使用者可以利用 Mouse 很輕易的改變檔案讀取目錄。

d. File 選擇游標

位在 File 選擇欄位內的游標，用來供使用者標示出想要選擇的檔案。使用者以 Mouse 的游標點選某一檔案之後，File 選擇標會立刻移到該檔案。或者使用者也可以透過 Keyboard 的方向鍵移動 File 選擇游標，到使用者想要選擇的檔案。在任何時刻，被 File 選擇游標選到的檔案，其相關資訊都會立刻出現在下方的 File 參考資料欄內。

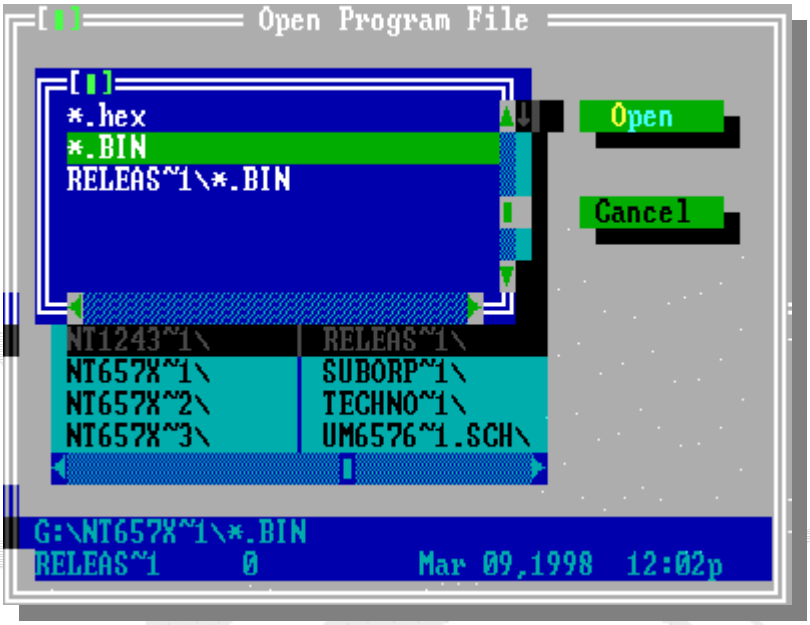



e. File 選擇欄

File Selection List 中會將指定目錄中，所有符合 File Name 輸入欄位中所規定條件的檔案全部顯示出來，使用者可以在這個 List 中，可以利用鍵盤上的方向鍵移動 Cursor 到所要選擇的檔案，然後按一下 Enter 鍵來完成選擇的動作。或是以 Mouse 控制 Cursor，使 Cursor 移到要選擇的檔案之後，雙擊 Mouse 的左鍵，也可以完成選擇的動作。

f. File Name 紀錄開關

使用者在 File Name 輸入欄位所輸入的所有動作，只要是有效而且也改變 S.D. 的反應，例如改變副檔名的名稱，或是改變工作目錄的路徑名稱等，都會被記錄在此處。因此使用者只要未退出並重新進入 S.D.，則可以利用此一輸入紀錄，將過去曾經用過的指令叫出來重新使用，而省去一再輸入相同命令的困擾。




File Name 輸入紀錄可以利用 Mouse 控制，將 Cursor 移到控制 Icon 上按一下左鍵之後開啓，然後以 Mouse 來雙擊(Double Click)所要選取的命令，或是以 Keyboard 的上/下方向鍵移動 Cursor 至所要的命令，接著按 Enter 鍵來選取需要的命令皆可。

注意：**File Name** 輸入紀錄內的資料，在退出 **S.D.**之後就會自動清除，因此每次重新啓動 **S.D.**之後，這部分的內容都是空的。

g. File 參考資料欄

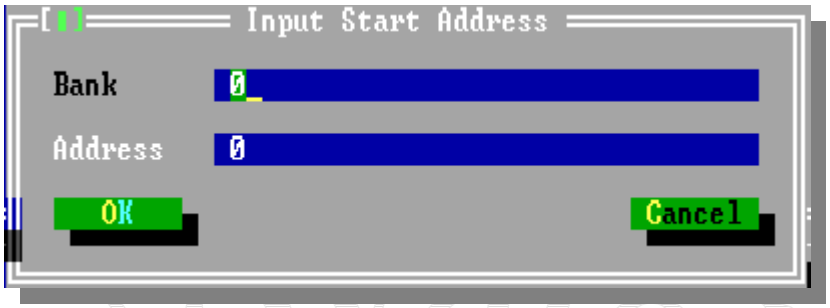
File 參考資料欄內所出現的是當時位在 File 選擇欄內，File 選擇游標所在位置的檔案的相關資料，其中包括了該 File 的目錄名稱(File Path)、檔案名稱(File Name)、檔案容量(File Size)、建檔日期(Date)及建檔時間(Time)等相關資料，提供使用者在選擇檔案時有參考依據。

**II.** File Load Parameter

由於 NT657X ICE 擁有高達 1M Bytes 的 Emulation ROM 空間，可以模擬 NT6576/NT6578 系列微處機的最大定址範圍，因此 S.D.在 Load Program 時，需要使用者輸入一些記憶體位址方面的資料，以便 S.D.能順利的將 Program 存到 NT657X ICE 內使用者指定的地址。至於所要 Load 的程式的 File Size 資料，則會由 S.D.自行計算得出，並在 Load 過程完畢之後顯示給使用者看。

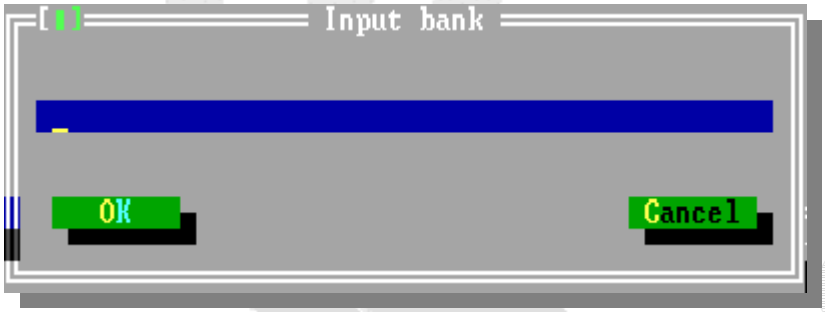
不過由於 Binary File 與 Hex File 的資料結構並不相同，因此在輸入 Parameter 時的做法也不盡相同。

Binary File 除了 Program Code 本身以外，不帶其他任何資料，因此使用者必須輸入該 File 在 Load 進 NT657X ICE 時，所要存放的 Bank 區域，以及在該 Bank 內的 Start Address(每個 Bank 以 32K Bytes 爲單位，$0000h~$7FFFh)。



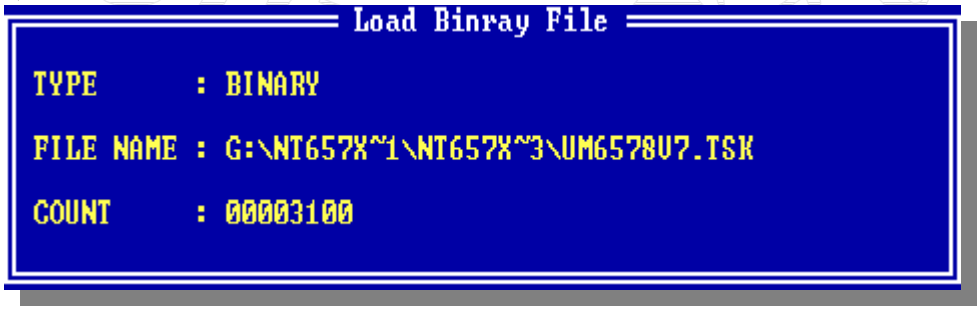

注意：**Program Down Load** 過程開始之後，即使所要 **Load** 的 **File Size** 超過一個 **Bank** 的大小，**S.D.**仍會自動計算並由下一個 **Bank** 繼續存放未完的資料。例如 **Down Load** 一個 **512K Bytes** 的 **Binary File** 到 **NT657X ICE** 的 **Bank 0** 的**$8000** 位址時，就在 **Bank** 一欄輸入 **0**， **Address** 也輸入 **0**，就可以將整個 **512K Bytes** 資料都 **Load** 進 **NT6576/NT6578** 系列微處理機的 **Bank 0** 至 **Bank 7.**

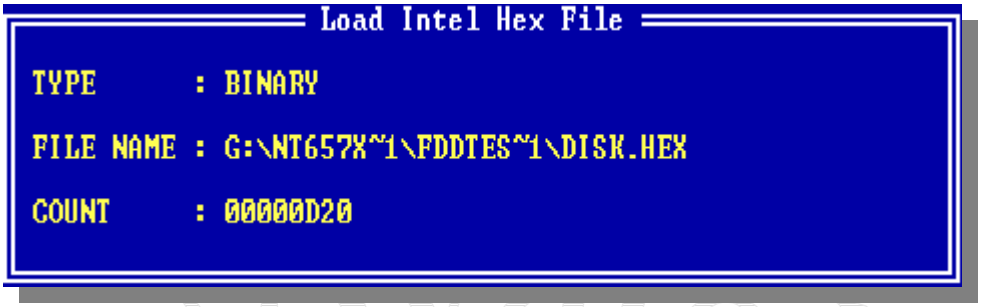
由於 Hex 格式的檔案內帶有該檔案的 File Size 及 Memory Allocation 等參考資訊，因此在輸入位址參數時，僅需輸入 Bank 值就好，Start Address 則由 S.D.自行從 Hex File 的資料中取得。




**III.** File Downloading

在使用者輸入參數完畢之後，S.D.會開始 Down Load 程式。而在 Down Load 的過程中，S.D.會自動爲傳送的每一個 Byte 作傳送正確與否的比對，並記錄 Check Sum，以免資料傳送的過程中出現任何錯誤。同時在畫面上也會有一 Window，用來顯示所傳送資料的進度。



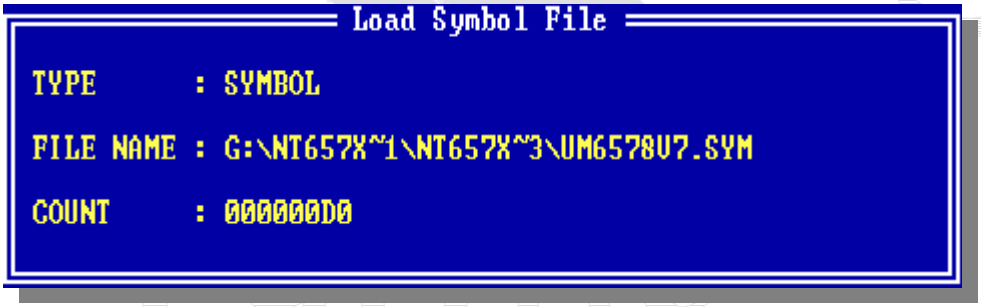



注意：由於傳送過程中，需要作資料正確與否的判斷，因此傳送速度會略受影響。如果使用者是在 **Windows** 的 **DOS Window** 下執行 **S.D.**，則建議使用者最好將該 **DOS Window** 設為全螢幕模式，以便幫助提昇 **Down Load** 過程的速度。

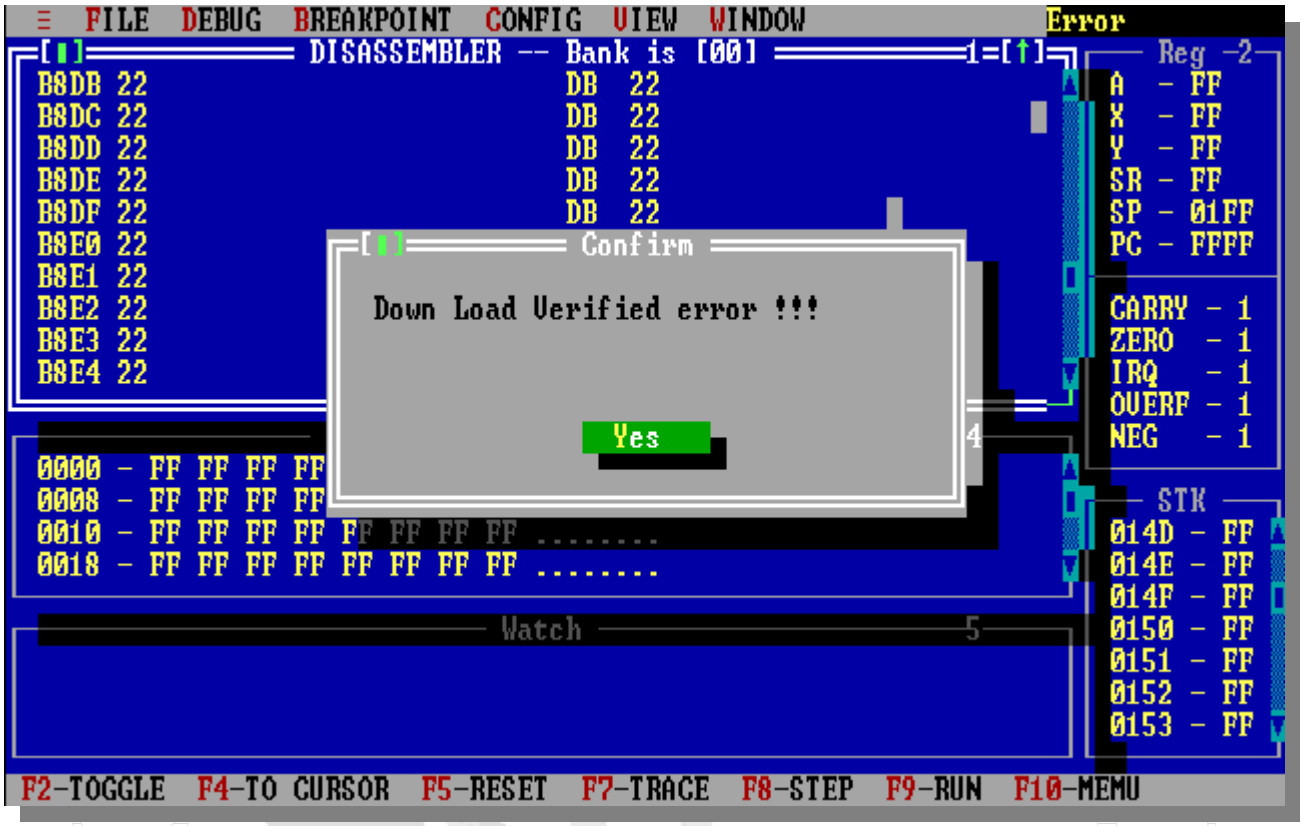
**IV.** File Down Load Complete

如果 Down Load 過程一切順利，則 Down Load 完畢之後，會有一個訊息提醒使用者 Down Load 工作順利完成。


在 Down Load 過程順利結束後，如果 S.D.發現同一目錄中有同檔名之 symbol file，則 S.D.會自動載入該 Symbol File。




如果 Down Load 過程中發現通訊錯誤，或者資料傳送結束後發現 Check Sum 有問題，則 S.D.會提出另一個錯誤訊息提醒使用者。

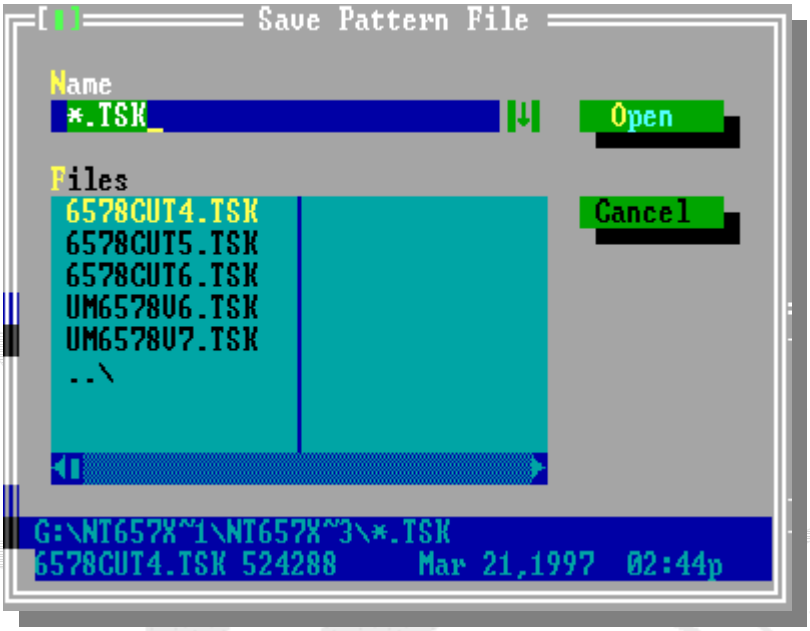



注意：如果 **Down Load** 錯誤，則 **S.D.**不會再繼續 **Down Load** 該檔案的 **Symbol File**。

# Save Program

Save 功能可以讓使用者將存在於 NT657X ICE 內的資料，尤其是經過 Debug 後確定沒問題的程式，以 Binary 的資料格式存到硬碟或是磁碟片等儲存媒體上。

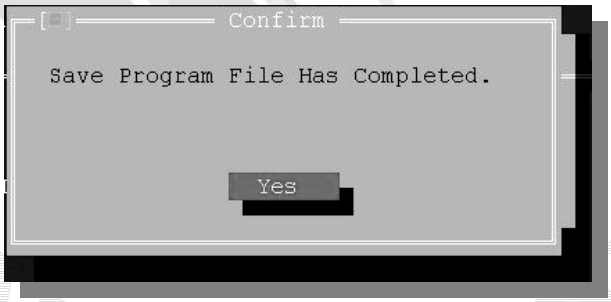
選擇 Save 功能之後，S.D.會提供一個與 Load Program 功能類似的 Save Pattern File Window。此一 Save Pattern File Window 與 Open File Window 的操作方式完全相同，因此操作說明部分，請參考 Page18 – “File Open 階段”一節中相關的介紹。




等使用者設完將 Save 的檔案名稱之後 (可以是新檔案或是副寫已存在的檔案)，S.D.會出現另依各視窗要求使用者輸入所要 Save 資料的相關參數，包括資料所在的 Bank、Start Address 及 Data Length 等。

<目前的 SD 在 File Save 時，並不會要求輸入 Parameter，須待改版之後才會有上述功能>

待相關參數設定完畢之後，S.D.會開始進行 Save 的動作，若一切正常，則 Save 完畢之後會出現 Save Ok 的訊息。



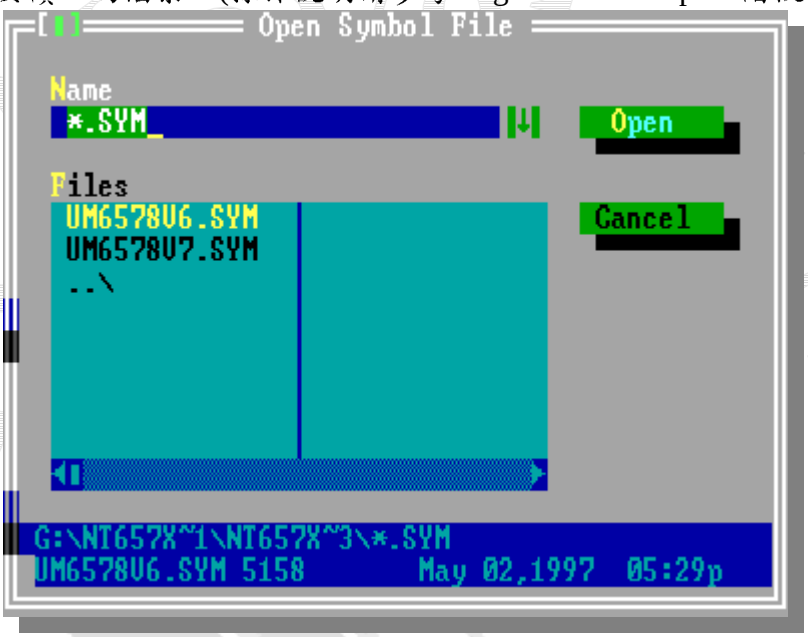

# Load Symbol

本系統之反組譯(Disassembly)視窗與變數(Watch Variable)視窗是可以用 Symbol 來代替地址與資料，讓使用者擁有一個類似 Symbol Debug 的環境，幫助軟體工程師更容易進行 Software Debug 的工作。所以使用者若能在進行 Software Compile 的時候一併產生該程式的 Symbol File，則本系統在 Download Symbol 之後，就可以將所有的 Variable 及 Subroutine，以 Symbol 的形式顯示給使用者。

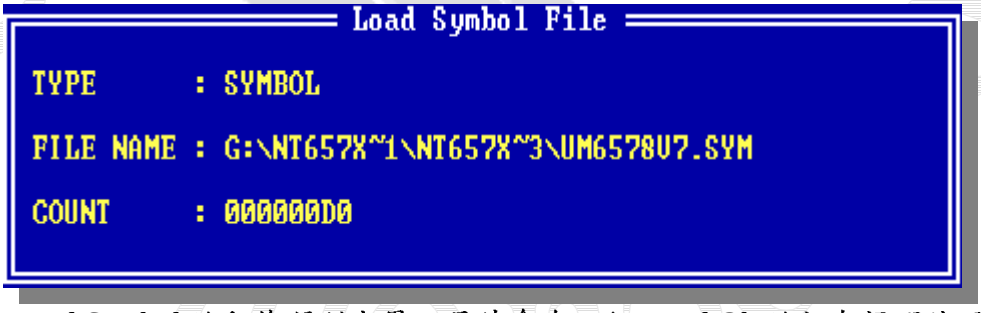
Symbol 檔案可由市面上多種 Cross Assembler 所產生(例如 2500AD 的 6502 Compiler 及 Link 程式)，不過 Symbol 的格式種類繁多，因此 S.D.採用一般最常用 Zax 格式，所以使用者在利用 Cross Assembly 產生 Symbol File 時，請指定以 Zax 格式產生。

雖然在 Download Program 的時候，S.D.會自動尋找並 Load 相同檔名的 Symbol File，可是爲了使用者習慣考慮起見，S.D.還是設了一個獨立的 Load Symbol 功能，以便當使用者的 Symbol File 放在與 Program File 不同一個目錄，或是使用者爲同一個程式建有多個 Symbol File 時，仍有一個 Load 工具可以使用。

使用者在選擇 Load Symbol 功能之後，S.D.一樣會開啓一個 Open Symbol File Window 讓使用者選擇所要讀入的檔案。(操作說明請參考 Page18 – “File Open 階段”)




Symbol 在 Load 的過程中，同樣會有一個 Message 顯示，讓使用者掌握 Load Symbol 的狀態。




待 Load Symbol 的動作順利完畢，另外會有一個 Load Ok 的訊息提醒使用者。

# Change Dir…

若使用的所要讀取的檔案與 S.D.所在的工作目錄不同，爲了避免讓使用者每次在 Open File 時都要更換目錄而造成不方便，S.D.特別提供了 Change Directory 功能，使用者可利用此功能更改工作目錄，如此就可以省掉每次要存取檔案時皆要跳至該目錄的麻煩。

使用者開啓 Change Directory 功能之後，會出現如下的 Window，使用者僅需參照 Open File Window 類似的操作方式，就可以任意切換 S.D.的工作目錄。

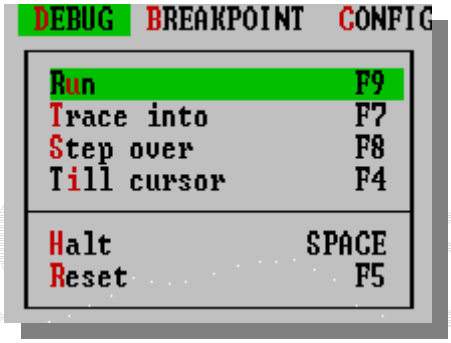

# DOS Shell

當使用者在執行或除錯時，可能會需要用到ＤＯＳ下的一些命令，因此本系統提供 DOS Shell 的功能，讓使用者回到 DOS 中去執行使用者的命令，待執行完畢後再以 “Exit” 命令離開 DOS Shell 回到 S.D.系統中。

# Exit

此功能爲跳離本系統的裝置，當使用者執行此命令後，SD 軟體就會完全退出而不再控制 NT657X ICE，而在 SD 軟體退出的同時，NT657X ICE 也會被自動 Reset，然後執行當時存在於 NT657X ICE 內的軟體。

# *Debug*



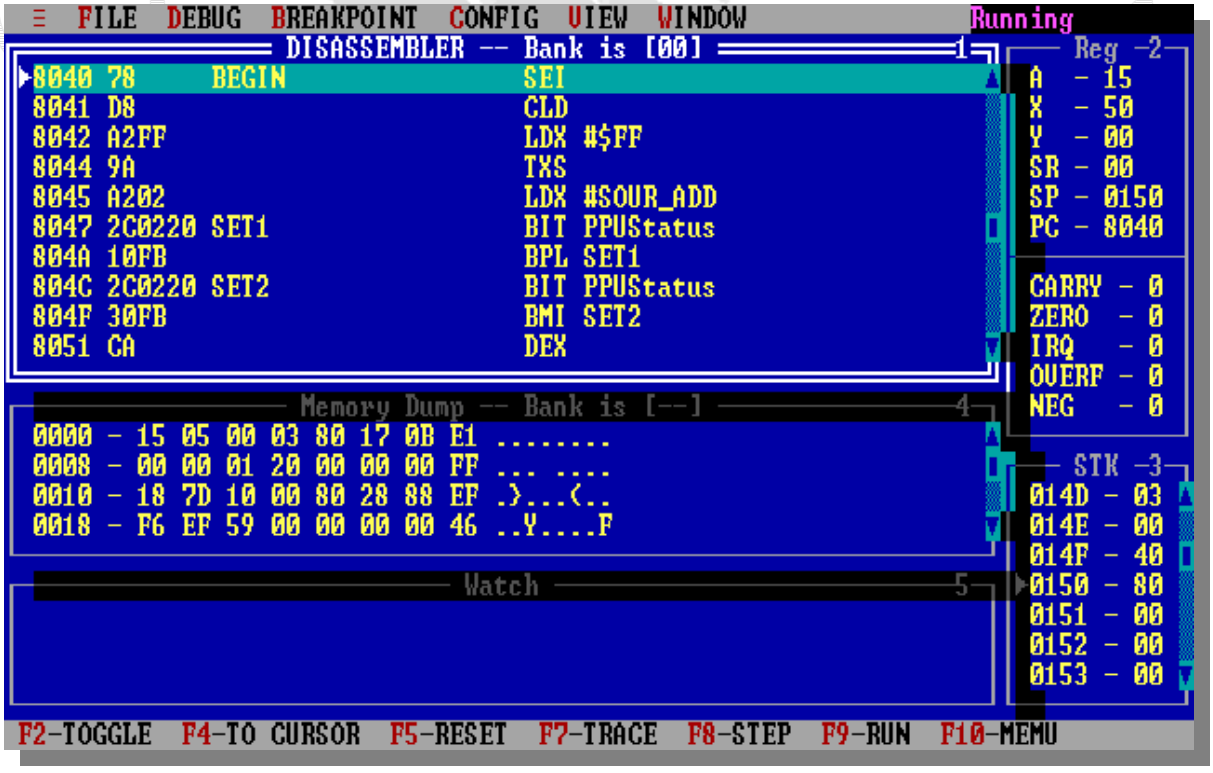

這一組皆是與如何控制 NT657X ICE 執行程式有關，SD 共將控制的模式區分爲六種方式，分別屬於四種執行指令 - 執行(Run)、執行至游標所在位置後即停止(Till Cursor)、一般單步追蹤(Trace)及自動執行副程式的單步追蹤(Step)。及二個強制中斷執行狀態的命令-暫停(Halt)與重置(Reset)。

Debug 視窗內的指令，無論執行條件或是停止條件，都是由 Disassembly Window 所決定。其他的 Window 都與 Debug 功能無關。

## Run

Run 功能會讓發展系統直接執行使用者的程式，但是在使用者的程式執行的同時，SD 軟體同時會監控整個執行過程。

當使用者下了 Run 指令之後，螢幕的右上角會顯示出 “Running” 的訊息，這表示 SD 正在執行使用者的程式。



當系統以此 Run 指令執行程式時，使用者程式的執行優先權會高於 S.D.的監控權，因此一旦使用 Run 指令，S.D.就不會在接受一般的指令，以免干擾使用者程式的執行狀況。若要停止 Run 指令繼續執行，則有以下三種方式：

a。系統執行時遇到一個中斷點，不論是 Software 或 Hardware 的 Breakpoint。

b。使用 Halt 的命令來使系統暫時停止執行使用者的程式。

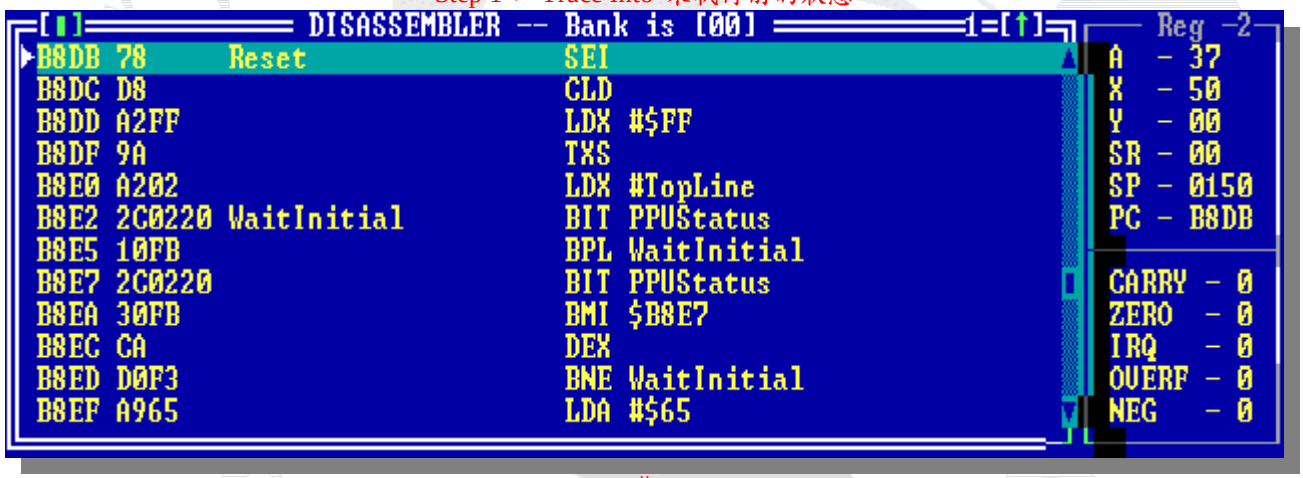
c。使用 Reset 命令來停止系統執行並重置整個系統。

d。使用者改變 NT657X ICE 的 Chip Type 選項。

# Trace Into

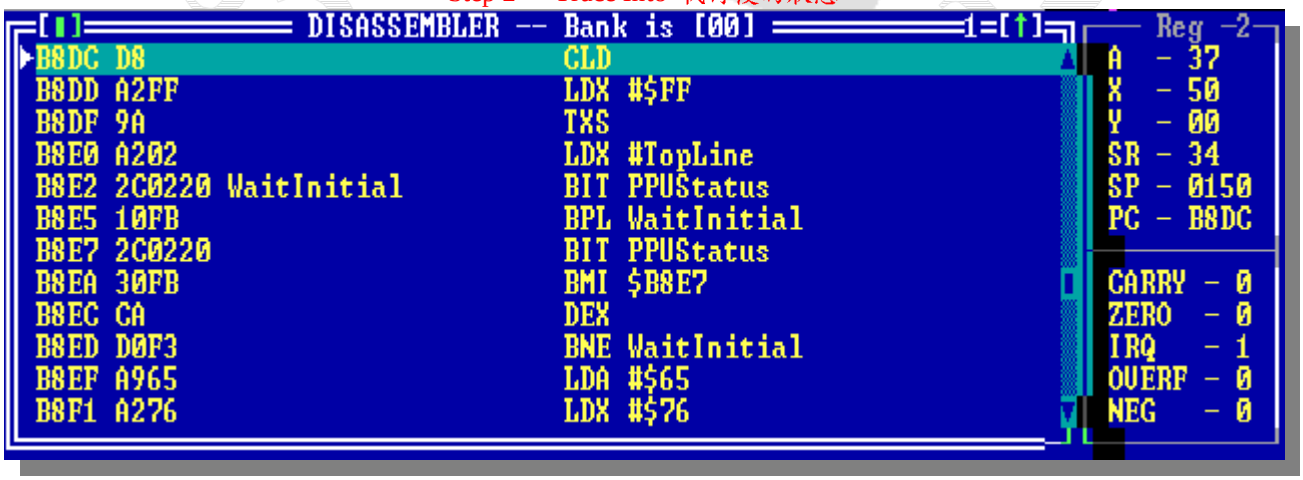
這是一個單步執行的控制命令，使用者每下一次 Trace 命令，SD 就會執行一個指令，然後將整個 SD 畫面內的各個視窗內容更新之後，再回到 SD 的監控狀態下。要使用 “Trace Into” 的功能，除了可以利用 Debug 選單內的 “Trace Into” 選項來執行之外，也可以利用 F7 鍵來執行。

Step 1：“Trace Into”未執行前的狀態




⇓

Step 2：“Trace Into”執行後的狀態




請注意，在 “Trace Into” 執行過之後，Step 1 與 Step 2 有下列幾項變化。

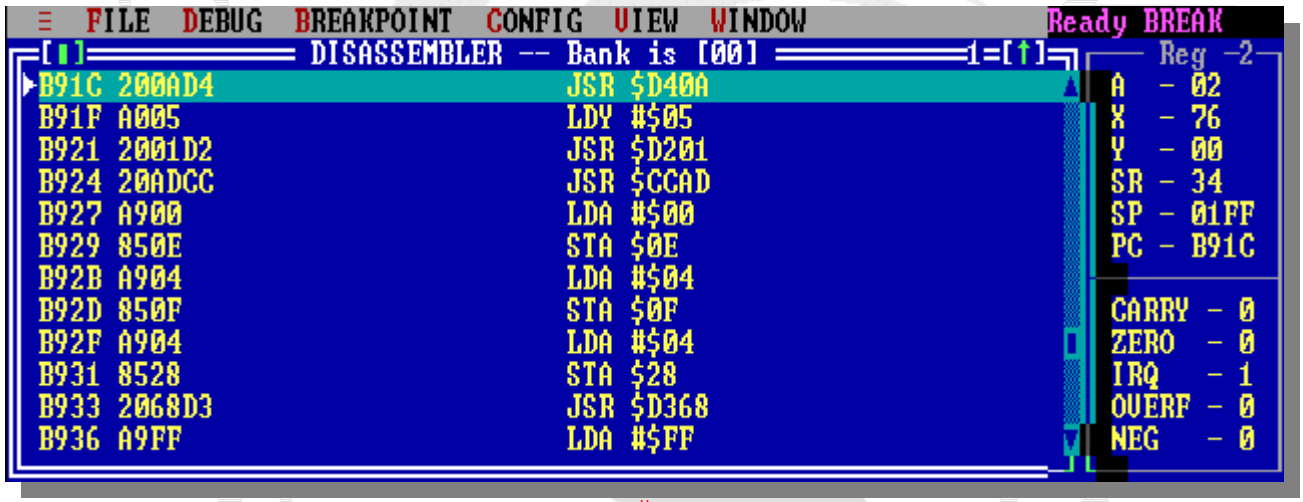
1. Disassembly Window 內的程式已經向下移動一列；
2. CPU Register Window 內的 Program Counter (PC) 已經由“$B8DBh”變成“$B8DCh”；
3. CPU Register Window 中的 IRQ Flag 已經變成 1 (因爲 SEI 指令的關係)；

# Step Over

它的功能與 “Trace into” 類似，都是會在接受命令後執行指令，然後自動停止。它們唯一的不同是，當使用者使用 “Step Over” 功能時，如果要被指令是 JSR，則 SD 會自動爲使用者執行 JSR 以後的副程式，待整段副程式都執行完畢後，才會停止程式的執行，除此以外皆和 Trace into 相同。

“Step Over”功能主要是爲了使用者在 Debug 階段，對於某些不想或是不需要重複驗證的副程式，都可以利用此功能跳過整個副程式，並將整個結果顯示出來。Step Over 功能可以利用 F8 鍵來直接執行。

Step 1：”Step Over”功能執行前




⇓

Step 2：”Step Over”功能執行後的狀態

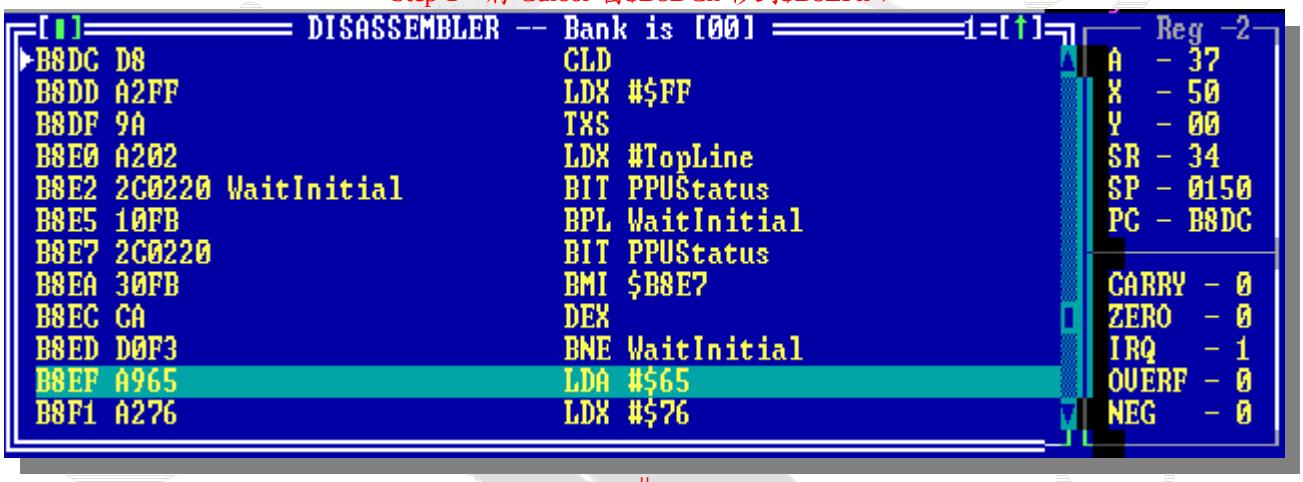
# Till Cursor

此命令下達之後，SD 會由目前 CPU 的 Program Counter (PC)所在的位置開始執行使用者的程式，一直到 Cursor 所在位置之處，然後停止執行。在這中間不論有多少指令或是副程式，SD 都會自動執行。

“Till Cursor”功能提供一個 Debug 環境中彈性最大的指令。“Step Over” 指令僅能跳過一段副程式，而 “Till Cursor” 指令則可以讓使用者跳過數個連在一起的副程式，這種功能允許使用者更有彈性的進行 Debug 工作。在程式中，僅有出問題的部分才使用 “Trace Into” 功能來找出 Bug，懷疑有問題的部分則可以用 “Step Over” 功能來驗證，至於確定沒問題的部分，就可以直接用 “Till Cursor” 功能來略過。

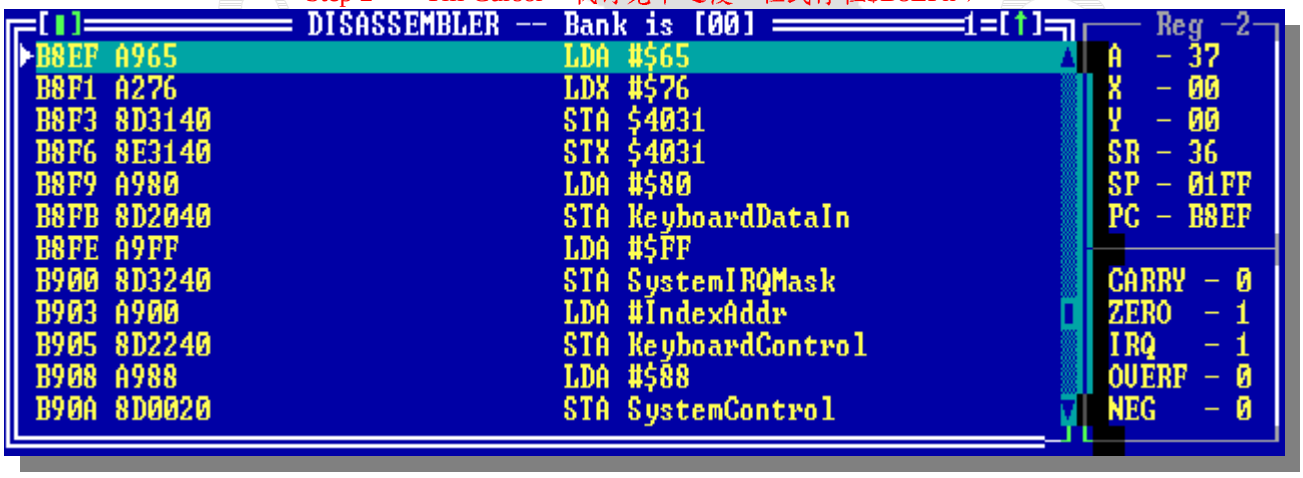
“Till Cursor”功能除了可以由 Debug 功能表中選擇之外，也可以利用 F4 鍵來直接執行，但是在執行 “Till Cursor” 之前，必須先移動 Cursor 到要停止的位址。

Step 1：將 Cursor 由$B8DCh 移到$B8EFh；



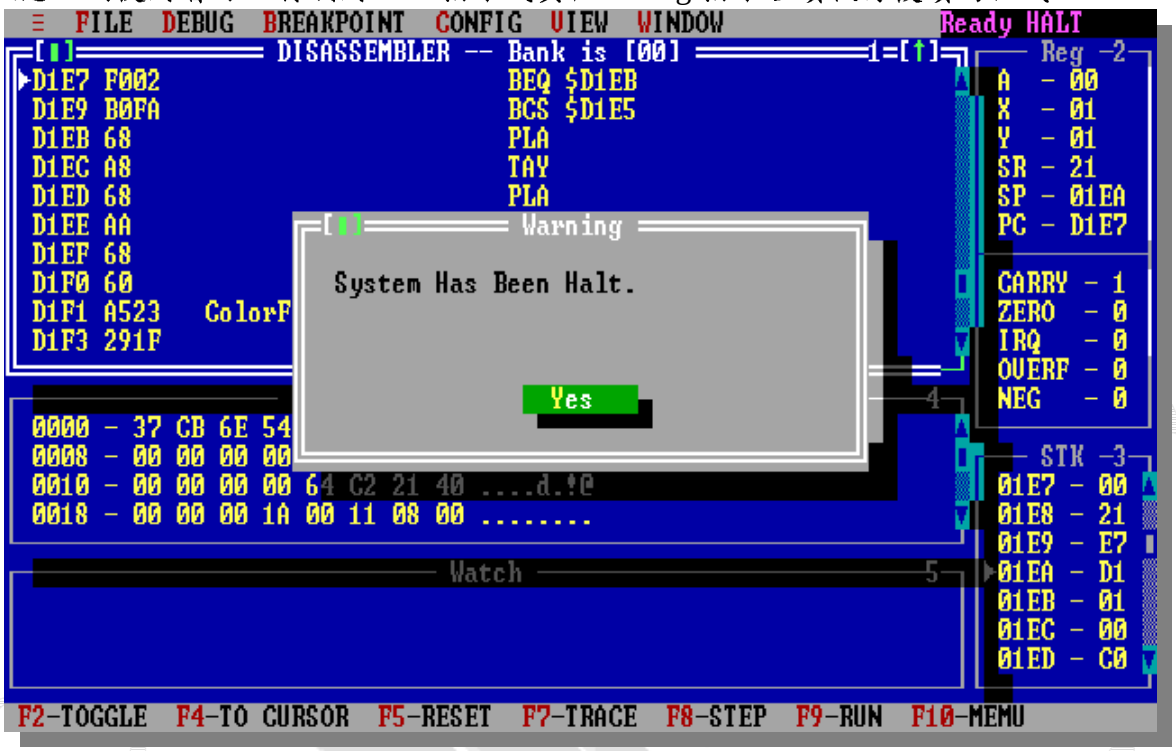

⇓

Step 2： “Till Cursor” 執行完畢之後，程式停在$B8EFh；

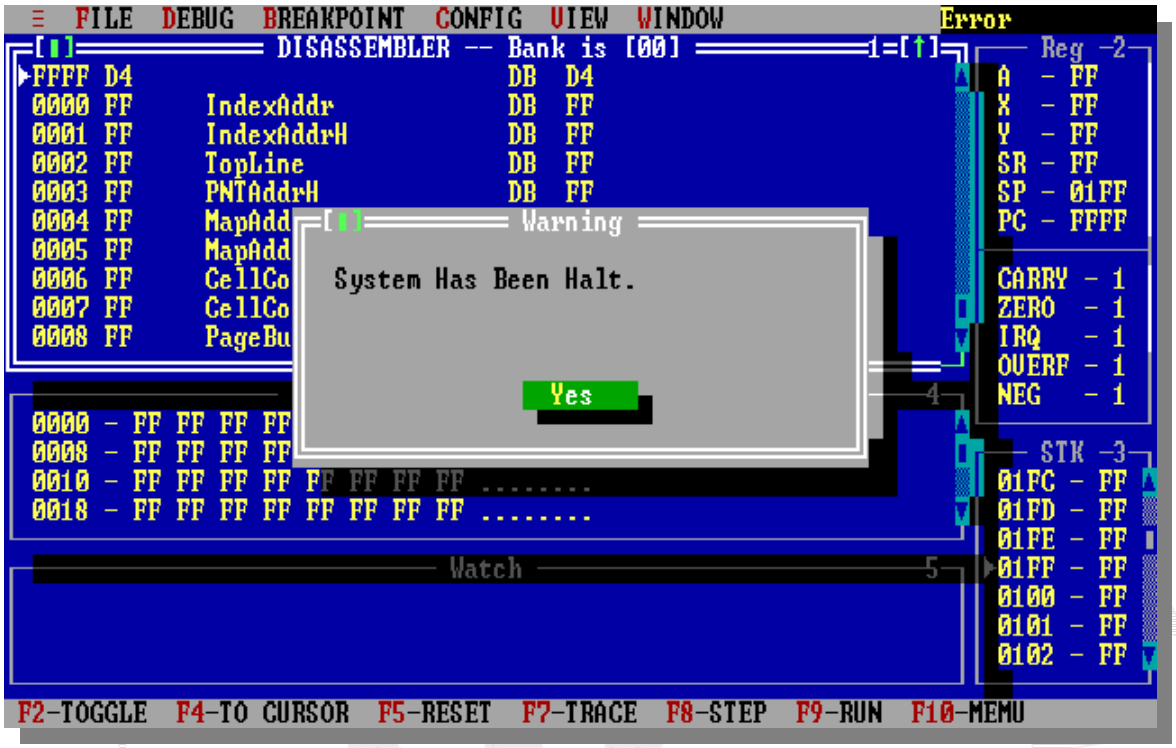


# Halt

當 SD 正在執行使用者的程式時，若想暫停使用者程式的執行狀態，則可以按鍵盤上的 Space Bar，使隨時讓 SD 暫時停下來。當 SD 執行 Halt 命令之後，同時畫面上也出現 Ready 的訊息，則使用者可以再利用 Run 指令或其他 Debug 指令繼續執行後續的程式。




由於 6502 CPU 是一款設計較簡單的 CPU，並不直接支援 Halt 功能，所以如果當使用者下達 Halt 指令時，程式正在執行 DMA 的動作或是正好在處理 Interrupt 或 NMI，則 Halt 功能雖然會讓 S.D.停下來，但是由於 6502 CPU 無法同步停下來，因此 S.D.的程式追蹤狀態有可能會出錯，同時右上角也會顯示 Error。




當 SD 在 Halt 之後出現 Error 狀態，SD 就不保證後續指令能再繼續正確的執行，所以若在 Halt 後發現 Error，則使用者最好下 Reset 命令來使 SD 恢復正常。如果使用者的程式中大量使用 DMA 或是 Interrupt 功能，而使程式不容易正確 Halt 時，建議使用 Breakpoint 會必較容易也穩定。

# Reset

這個功能是強迫發展系統放棄目前所執行的程式，而與 PC 上的 SD 監控軟體重新建立通訊連接。因此若使用者的程式內有邏幾幸的錯誤，而使軟體陷入無止境的迴圈內、或是當機、或是系統因 Halt 不正確而令 SD 發生“Error”的狀態時，使用者就可使用 Reset 功能，讓 S.D.回到正常狀態。

“Reset”命令的作用，就有如對 NT6576/NT6578 IC 上的 Reset 信號施加一個 Low 的信號，因此 NT6576/NT6578 IC 內的一切狀態都會回到原來的啓始狀態(Initial State)。

當使用者由 Config 功能表中選擇 Chip Type 選項，並且改變 NT6576/NT6578 的選項之後，Symbol Debugger 也會自動命令 NT657X ICE 執行 Reset 動作。

# *Breakpoint*

Breakpoint 功能包含如何設定／解除中斷點(包含軟體中段與硬體中斷)。由於在驗證軟體的過程中，尚未通過測試的軟體本身藏有邏輯性的錯誤，而使軟體在執行時陷入無止境的迴圈或是發生當機的狀況。因此必須藉由中斷點的設定，使軟體在執行到某些特定位址，或是在某些特定條件下，強制停止程式的執行，並將整個 CPU 當時的運作狀況顯示在 Symbol Debugger 的畫面上，幫助使用者來研判問題所在。

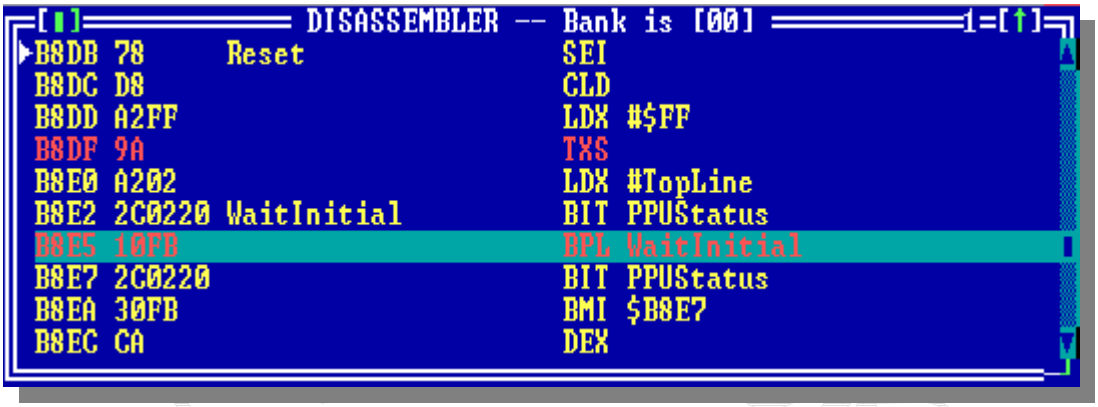
Symbol Debugger 在執行使用者軟體的過程中，會在執行每一個指令之前，檢查該指令的 Address，看該位址是否已被標示爲 Breakpoint。當 Symbol Debugger 軟體在執行中遇到 Breakpoint 時，就會停止執行該指令，然後將控制權交還 Symbol Debugger。而 Symbol Debugger 接著就將剛才軟體停止執行前的狀態顯示到各個相關的視窗之中，包括 CPU Register，Stack，Variable 等，同時 Disassembly Window 也會更新程式的 Disassembly 狀態，然後 Symbol Debugger 就會回到 Ready 狀態等待使用者輸入新的指令。

Symbol Debugger 軟體提供二種中斷方式，Software Breakpoint 與 Hardware Breakpoint。Software Breakpoint 是在 Symbol Debugger 執行使用者軟體的過程中，不斷檢查每個指令的位址來決定是否需要中斷程式的執行，因此可以擁有許多組的中斷點設定。而 Hardware Breakpoint 則需透過硬體的幫助，直接檢查 NT657X ICE 內各電器信號的變化，包括 Address Bus、Data Bus、R/W Control 等，只要條件符合，不管該位址是否爲程式或是指令，都可以強制中斷。由於構造較複雜，因此僅有一組。

注意：**Software Breakpoint** 僅針對每個指令的位址進行比對，因此不能設定在非程式區域，例如資料區或是變數區。而 **Hardware Breakpoint** 則無此限制。


## Toggle

這個功能是用來將 Cursor 所在位置的指令，設定或解除一個 Software Breakpoint，這個功能可以利用 F2 鍵來直接執行。若游標所在處原本不是一個 Software Breakpoint，則在按下 F2 鍵之後，這個位址會設定一個新的 Software Breakpoint。若游標處已經存在一個 Software Breakpoint，則按下 F2 鍵之後，會將此 Software Breakpoint 解除。



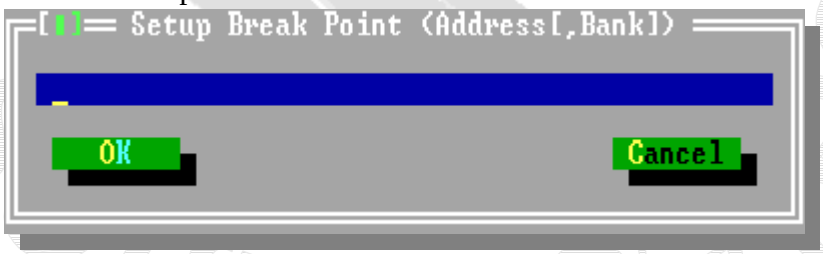

注意：紅色字體即代表 **Software Breakpoint**。

若是 Software Breakpoint 與 Hardware Breakpoint 同時存在於一個位址，基本上 Hardware Breakpoint 的優先權會高於 Software Breakpoint。但是由於 Hardware Breakpoint 會檢查中斷的條件是否相符合，而 Software Breakpoint 則只要 Address 相符合就 Break，因此最後可能還是由 Software Breakpoint 所 Break 下來。

以 F2 鍵設定 Software Breakpoint 時，S.D.會自動參考目前 Disassembly Window 中的程式所在的 Bank 值，並設定到 Software Breakpoint 的觸發條件。

# Break at

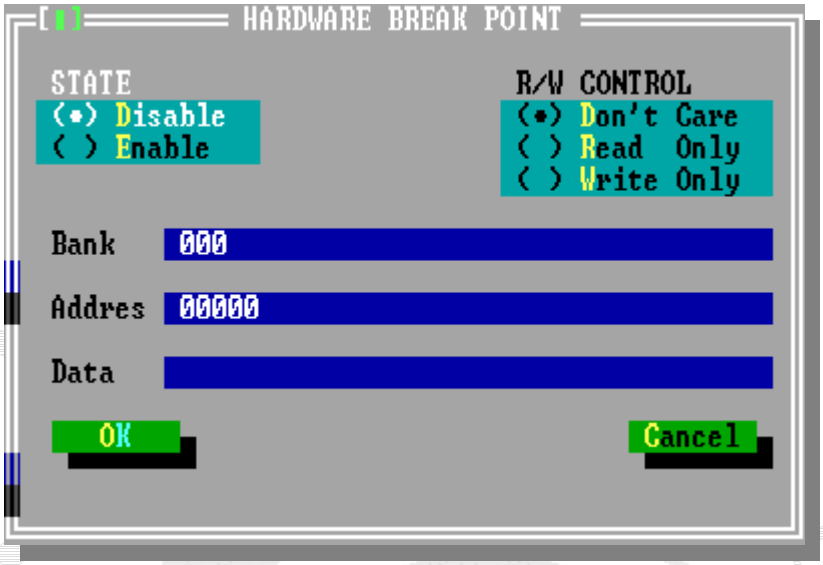
“Break at” 功能允許使用者直接輸入要設定的 Software Breakpoint 的位址，並且可以設定 Bank 值，使 Break 的觸發條件更爲精確。因此使用者不必先將 Cursor 移到將設定 Software Breakpoint 的位址之後，在案 F2 檢來設定，而是利用 “Break at” 的功能就可以直接設定所需要 Software Breakpoint。




選用 “Break at” 功能之後，畫面會出現如上圖的視窗供使用者輸入 Breakpoint 的 Address，此時使用者可以輸入 16 進制的 Address (例如 0A000) 以及 Bank 值，如果未指定 Bank 值，則 Symbol Debugger 預設的 Bank 值爲 0。

# H/W Break

這是一個硬體中斷點的設定功能，它可以提供較 Software Breakpoint 更複雜的中斷控制條件，對使用者而言是一個如虎添翼除錯工具。當使用者開啓本功能時，系統會有如下圖示，使用者可依所需的條件來設定硬體中斷點。




**I.** State：用來供使用者決定 Enable 或是 Disable Hardware Breakpoint 的功能。

**II.** R/W Control ：用來決定是否將 CPU 的 R/W 狀態加入 Hardware Breakpoint 的觸發條件。

a. Don't Care – CPU 的 R/W 狀態不列入 Hardware Breakpoint 的觸發條件。其代表的意義是只要下方的 Bank Λ Address Λ Data 的條件符合，Hardware Breakpoint 的功能就會被觸發。

b. Read Only – 僅有在 CPU 為 Read 狀態，而且 Bank Λ Address Λ Data 的條件也符合時，Hardware Breakpoint 的功能才會被觸發。

c. Write Only - 僅有在 CPU 為 Write 狀態，而且 Bank Λ Address Λ Data 的條件也符合時，Hardware Breakpoint 的功能才會被觸發。

**III.** Bank：供使用者選擇是否將 Bank 值列入 Hardware Point 列入觸發條件。一般使用時，若 Bank 值是設為 0，則代表不將 Bank 值列入觸發條件之一。如果 Bank 值不是 0，則 Symbol Debugger 只有在 Bank 值符合 Hardware Breakpoint 所設定的 Bank 值時，才會觸發 Hardware Breakpoint 功能。

**IV.** Address：供使用者選擇是否將 Address 列入 Hardware Point 的觸發條件。若 Address 值是設為 0，則 Address 不會列入 Hardware Breakpoint 的觸發條件。如果 address 值不是 0，則 Symbol Debugger 只有在 CPU 的 Address 值符合 Hardware Breakpoint 中所設定的 Address 時，才會觸發 Hardware Breakpoint 功能。

**V.** Data：Symbol Debugger 除了比對 Address 及 Bank 之外，還可以考慮是否將 Data 的內容也列入 Hardware Breakpoint 的觸發條件。一般使用時，Data 值是設為 0，如果 Data 值不是 0，則 Symbol Debugger 只有在 Bank Address 及 Data 值也符合時，才會觸發 Hardware Breakpoint 功能。

注意：使用 **Hardware Breakpoint** 時，請勿將 **Breakpoint** 位址設在 **Zero Page** 的 **$0000h~$0003h**，以及 **Stack** 的區域**($0100h~$01FFh)**。若使用者將 **Hardware Breakpoint** 設到上述這些區域，則 **Symbol Debugger** 在執行 **Break** 動作時，將會有發生錯誤的可能。

## Enable all

將所有預先設立的中斷點(Breakpoint)恢復成正常的動作狀態。在正常狀況下，這個選項是處於不能操作的狀態(Inactive)，只有在使用者選擇了 "Disable All" 功能之後，"Enable All" 這個選項才會變成可以操作的狀態(Active)。

## Disable all

將所有中斷點(Breakpoint)暫時取消。一旦選擇 "Disable All" 功能，不管原來在程式中設下多少 Software 或是 Hardware 的 Breakpoint，都會變成失效(Disable)的狀態，因此程式在執行時不會再因為 Breakpoint 而被停下來。此一功能主要是用來供使用者在 Debug 完之後，在不必刪除任何 Breakpoint 的前提下，可以利用此功能暫時避開 Breakpoint 的干擾，而進行軟體測試。

Disable All 功能僅是讓 Symbol Debugger 暫時忽略任何 Breakpoint 的存在，實際上 Breakpoint 的動作條件仍然沒有改變。在選擇 "Disable All" 之後，使用者只要再選擇 "Enable All" 就可以使每一個 Breakpoint 恢復正常。

## Delete all

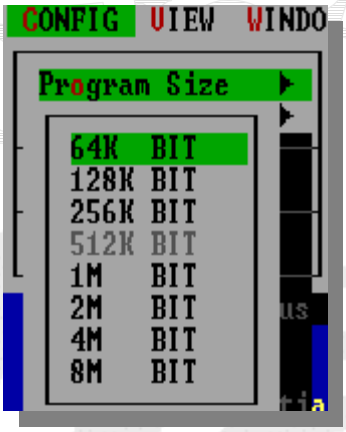
將所有中斷設定(Breakpoint)全部清除。一選用此功能，所有的 Breakpoint 的動作條件都會被清除，原來可以在 Breakpoint List 中看到的 Breakpoint 都會被消掉，Hardware Breakpoint 也會被設成 Disable 狀態，而 Disassemble Window 中的所有 Breakpoint 記號也會被清除。

# *Config*

由於 NT6576/NT6578 系列微處理機現階段共有二種不同功能的晶片，而 NT657X ICE 內部又提供高達 8 Mbits 的模擬記憶體。為了讓使用者能夠選擇晶片型式以及設定 NT657X ICE 內部的 ROM Emulator 的容量大小，因此提供這個選項供使用者設定適合的環境。

# Program Size

此一選項用來設定 NT657X ICE 內的 ROM Emulator 的容量大小，最小爲 64 Kbits，最大則爲 8 MBits。設定 Program Size 的意義就有如決定 NT6576/NT6578 IC 外加的 EPROM (或是 Mask ROM)的容量大小，例如設定 64 KBits，就有如選用 27C64 的 EPROM，而設爲 8 Mbits 時，則有如選擇 27C080 的 EPROM 一般。

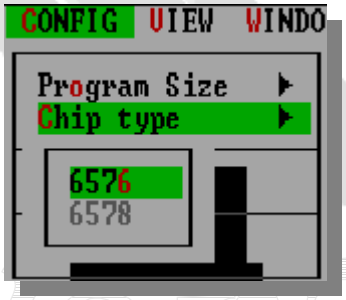



打開 Program Size 的選單時，裡面共有 8 個選項，呈現黑色的字體代表可供選擇的項目，而呈現灰色的字體則代表目前的設定值。

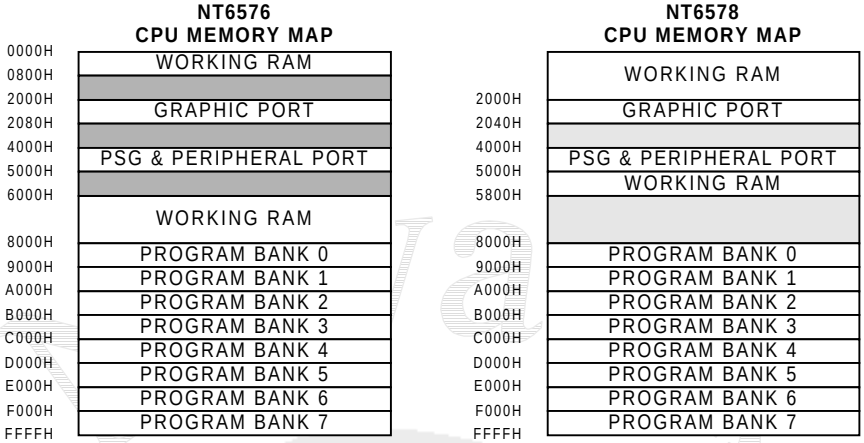
<在 S.D.未改版修改 File Save 功能之前，設定 Program Size 會改變 File Save 時的 Save Length 參數>

# Chip Type

該選項用來設定 NT657X ICE 所要模擬的 IC 型態，裡面共有二個選項，黑色字體代表可供選擇的項目，而灰色則是目前 NT657X ICE 的設定值。




改變 “Chip Type” 最大的影響在於 Working RAM 的分布方式，將隨 NT6576 及 NT6578 二種微處理機而有所不同。



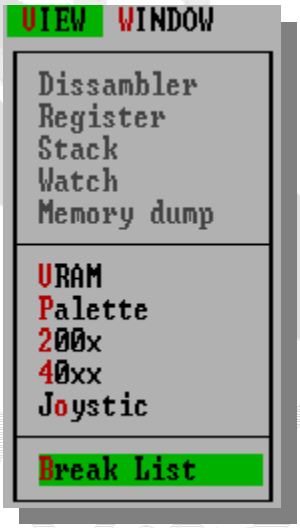

## Load Last

用來載入最近曾 Save 過的設定值。

## Save Config

用來將目前設定好的數據存入儲存媒體之中。

# *View*

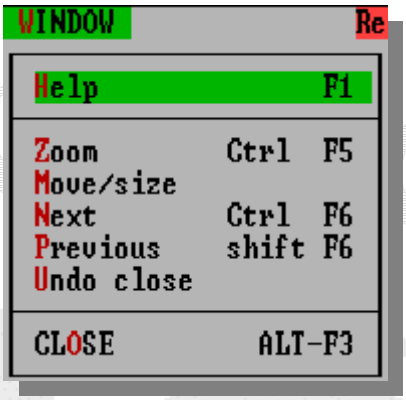



Symbol Debugger 軟體提供 11 種不同的視窗，以便將 NT657X ICE 內的所有狀況及時提供給使用者。但是這 11 個視窗並不是全部都開啓，在進入 Symbol Debugger 軟體時，僅有五個視窗是固定開啓的，其餘六個則是依使用者的需要來決定是否開啓，因此 S.D.提供此一 “View” 功能，供使用者決定該開啓哪些視窗。

打開 “View” 功能表之後，裡面會有 11 個選項可供選擇，其中灰色字體的選項，代表該視窗已經被開啓，所以不能再開啓。而帶有紅色字體的選項，則代表該視窗尚未開啓，使用者可以利用 Cursor、Mouse 或是代表該選項的紅字來開啓該視窗。

"View"功能內的各個 Window，將在後續個節中介紹，相關的詳細資料請參考各節的說明。

# *Window*




"Window"功能表內提供使用者控制各個 Window 大小、移動、切換等功能，以及一個簡單的線上命令說明(On Line Help)。

## Help


Help 選項會出現另一個 Window，裡面列出 S.D.中各種命令及相對的按鍵，以提供使用者一個簡單的使用提示。

## Zoom

"Zoom" 功能僅針對 Active Window 作調整，使用 "Zoom" 功能可以讓使用者將 Window 由當時尺寸放大成最大，或是由最大縮成一般大小，至於是放大或是縮小則由 Window 當時的尺寸所決定。


"Zoom" 功能可以用 Ctrl-F5 鍵直接呼叫。由於每個 Window 的特性並不相同，因此某些 Window 在作 Zoom 功能時，僅能進行高度的調整，而有些雖然可以調整高度與寬度，但限於資料顯示的關係，最大寬度會有所限制，以下就列出各 Window 的調整限制。

| Window 名稱 | Window 調整的限制 |
|---|---|
| Disassembly Window | 可以調整 Window 的高度，但是 Window 的寬度無法調整。 |
| Palette Window | |
| Stack Window | |
| Periphal Window | |
| Watch Variable Window | |
| Breakpoint List Window | |
| Memory Dump Window | 配合 Data Dump 的需要，高度可以任意調整，但是最大寬度有限制。 |
| VRAM Dump Window | |
| CPU Register Window | 由於資料量不多，因此 Window 的尺寸不需要也不能調整。 |
| PPU Register Window | |
| Joystick Status Window | |

# Move/Size


"Move" 功能讓使用者可以隨意調整個 Window 在 S.D.畫面上的位置，除了利用 Window 功能表內的 "Move" 指令之外，使用者也可以利用 Mouse 控制 Cursor，直接在所要移動的 Window 的上不有標題的部位，按住 Mouse 的左鍵，然後移動 Mouse 也可以達到相同的結果。

各 Window 在作移動時，基本上可以任意移動，但是必須注意 Window 的大小，因爲 S.D.並不允許任何 Window 的邊界移出畫面。

註一、 **"Move"** 指令與 **"Zoom"** 指令都是針對 **Active Window** 作動作，如果動作對象當時是 **Inactive Window**，則必須先讓該 **Window** 變成 **Active** 狀態之後，才能利用 **"Move"** 或 **"Zoom"** 指令作調整；

註二、不論使用 **"Move"** 或是 **"Zoom"** 指令，若是在調整的過程中，與其他 **Window** 發生重疊的現象，則 **Active Window** 永遠都會蓋過 **Inactive Window**；

# Next

將 Active Window 由目前的 Window，切換到下一個 Window，而切換順序則是依照 "View" 功能表中各 Window 的排列順序而定。要使用這項功能，可以利用 Ctrl+F6 二個鍵來切換。請注意，與一般 Windows 所使用的 Tab 鍵並不相同。

# Previous

將 Active Window 由目前的 Window，切換到上一個 Window，而切換順序則是依照 “View” 功能表中各 Window 的排列順序而定。要使用這項功能，可以利用 Shift+F6 二個鍵來切換。請注意，與一般 Windows 所使用的 Shift+Tab 鍵並不相同。

# Undo Close

此命令可以將最近被關掉的 Window 重新打開。

# Close

關閉目前是 Active 狀態的 Window，使用者可以利用 Alt+F3 鍵來完成，也可以利用各 Window 左上角的 Close Icon 來達到相同的目的。

被關閉的 Window，其在 “View” 功能表中的相對選項，會被改成待紅字的狀態，以便使用者在需要的時候，可以隨時重新開啓這個 Window。

# 1-4 系統狀態顯示(System Status Indicator)

系統狀態顯示(System Status Indicator)是一個單純狀態顯示功能，能告訴使用者現在NT657X ICE 系統本身之狀態，以及 Symbol Debugger 軟體與 NT657X ICE 連線的狀態。它位於螢幕的右上角，它的內容大致分爲七種，各狀態的涵意將解釋如下。

## Ready

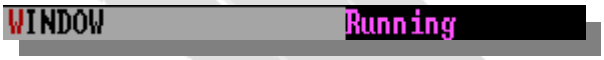

表示 Symbol Debugger 軟體與 NT657X ICE 的硬體連線狀況良好，NT657X ICE 硬體也正處於 Symbol Debugger 的監控模式下，等待使用者下操作命令。而 NT657X ICE 在這種狀況下，也不會執行任何使用者的的程式。

在 Ready 狀態下，使用者可已透過各種 Symbol Debugger 所提共的命令去存取或修改 NT657X ICE 內的資料，也可以將各種資料透過不同功能的 Window 顯示在 Symbol Debugger 畫面上，或是將使用者的程式 Download 到 NT657X ICE 內。

Ready 狀態僅是一種基本型態，隨使用者不同的操作方式，在其後方可能會加上不同的操作狀態，如 Halt、Break、Trace、Step 等。

> 注意：當 **NT657X ICE** 處於 **Ready** 狀態下時，**NT657X ICE** 上的 **Reset** 開關是被 **Disable** 的狀態，因此使用者若是按下 **NT657X ICE** 上的 **Reset** 開關將不會有反應。但是當使用者是處於 **Running** 狀態或是退出 **Symbol Debugger** 軟體之後，此一 **Reset** 開關就會恢復正常的工作狀態。

## Running




當使用者在 Ready 狀態下使用 Run(F9 鍵)命令之後， NT657X ICE 硬體將開始執行使用者的軟體，由於 NT657X ICE 是將整個發展系統用來執行使用者的程式，所以此時使用者無法透過 Symbol Debugger 執行其它功能，也就是無法對發展系統作任何修改或是顯示更新的動作。

若要在 Running 狀態下讓 NT657X ICE 回到 Ready 狀態，則可以透過系統重置命令(Reset)或暫停執行命令(Halt)來令 NT657X ICE 進入 Ready 模式，或是利用預先設定的中斷點來使 NT657X 在特定條件下停下來。

> 注意：前述所指的系統重置命令**(Reset)**，是指 **Symbol Debugger** 所提供的 **Reset** 命令**(F5** 鍵**)**，而不是 **NT657X ICE** 上所具備的 **Reset** 開關。如果使用者在 **Running** 的狀況下，按下 **NT657X ICE** 上的 **Reset** 開關，會使 **NT657X ICE** 產生硬體的重置動作**(Hardware Reset)**，並重新執行使用者的程式，可是 **Symbol Debugger** 軟體的 **Running** 狀態卻不會被改變。

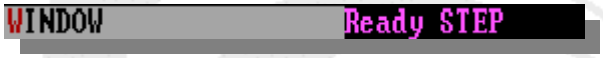
## Halt


當 NT657X ICE 在 Running 狀態下被使用者以 Halt 命令停下來之後，在右上角就會出現 Halt 字樣。請注意 Halt 之前所顯示的是 Ready 或是 Error。如果是 Ready，則使用者可以繼續使用並下達其他命令。如果是 Error，則請使用者下 Reset(F5 鍵)命令，使 Symbol Debugger 回到 Ready 狀態。

## Trace


當使用者在 Ready 狀態下使用 Trace(F7 鍵)命令或是 Goto Cursor(F4 鍵)之後，Symbol Debugger 的右上角就會出現 Trace 字樣，以提醒使用者目前正在使用的功能。

## Step

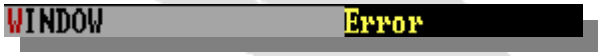



當使用者在 Ready 狀態下使用 Step(F8 鍵)命令之後，Symbol Debugger 的右上角就會出現 Step 字樣，以提醒使用者目前正在使用的功能。

## Break


當使用者的程式在執行過程中，遇到中斷點時，會自動停止程式的執行狀態，並將控制權交給 Symbol Debugger 軟體，然後在右上角顯示 Break 字樣，以便讓使用者了解程式停止執行的原因。

## Error




代表當時電腦上的 Symbol Debugger 軟體與 NT657X ICE 系統的連結出現問題。當系統顯示這個訊息時，使用者可以用 Reset(F5 鍵)命令來讓這兩個部分重新連接。

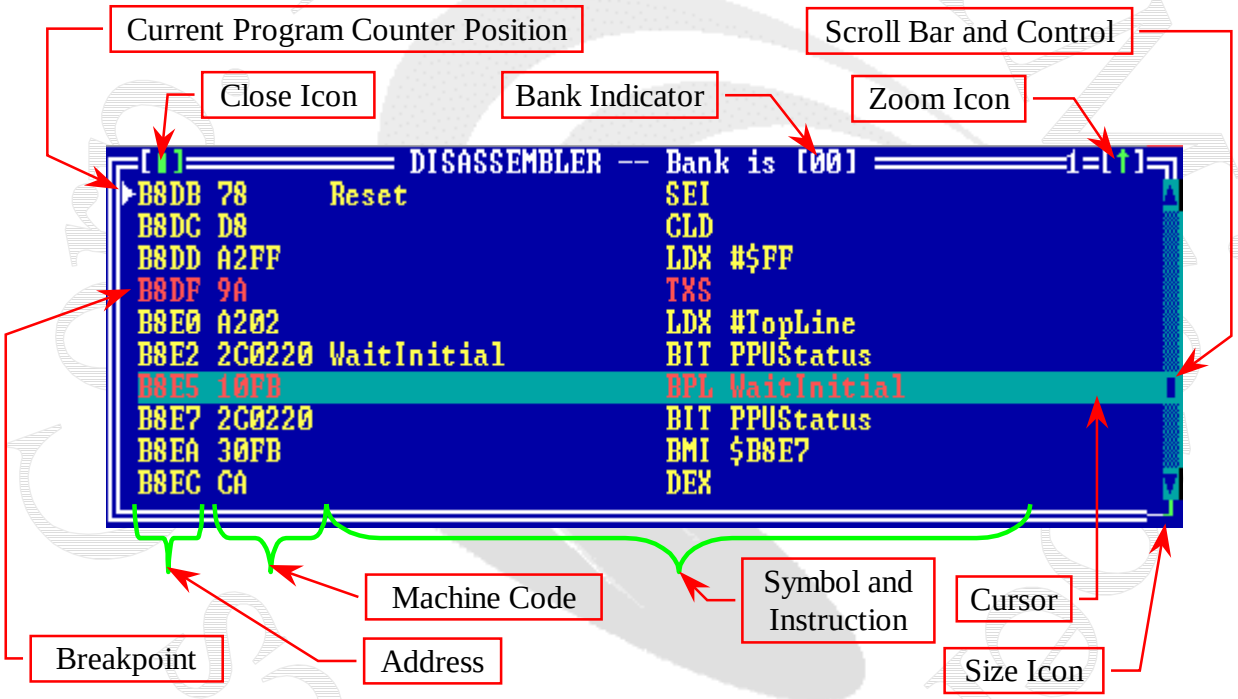
會使 Symbol Debugger 出現 Error 的原因有幾種，例如 PC 與 NT657X ICE 通訊不良、使用者下令 Halt 時剛好遇到 NT657X ICE 正在執行 DMA 等，但是不論何種原因，除非是 NT657X ICE 的硬體有問題，否則只要下 Reset(F5 鍵)命令，都可以使 Error 狀況回到 Ready 狀態。

# 1-5 反組譯視窗(Disassembly Window)

反組譯視窗是將使用者放在 NT657X ICE 內的資料以組合語言反組譯的形式顯示出來，使用者可以利用這個視窗來觀看使用者的程式，進行 Run、Trace、Step by Step、或是設定中斷點等 Debug 動作、甚至直接修改程式以驗證測試結果。

Disassembly Window 在顯示時，會對各種不同的資料屬性有不同的顯示處理。以下即為 Disassembly Window 的顯示方式、操作方法以及附屬功能選單命令使用的詳細解說。

## 顯示方式




在 Disassembly Window 中，除了將程式以反組譯方式顯示出來，以提供 Debug 動作的需求之外，她還提供設 Breakpoint，資料輸入，修改指令，單部執行等功能。以下就對 Disassembly Window 中各部分符號的意義加以說明。

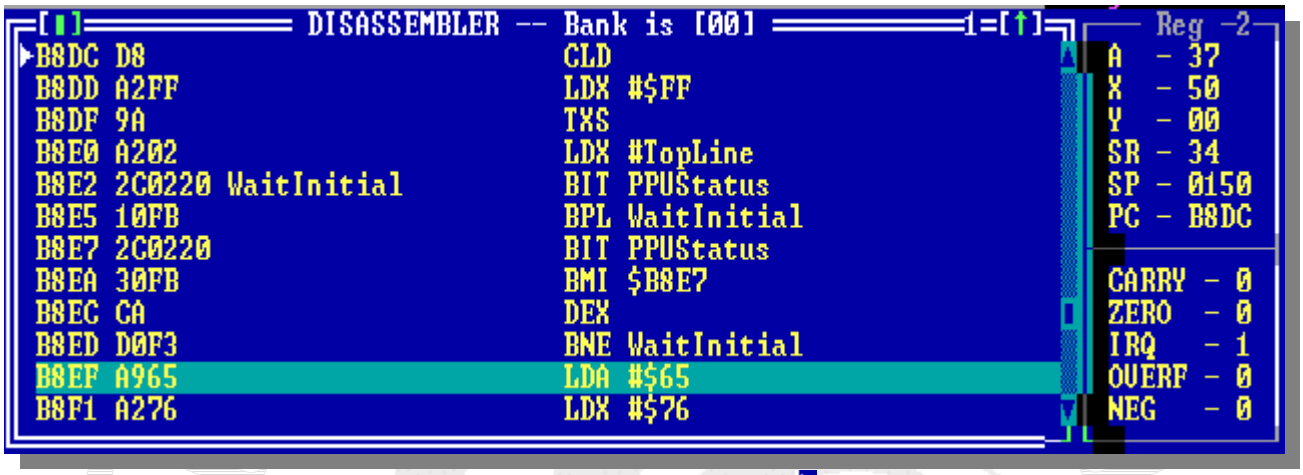
### Bank 指示(Bank Indicator)

為配合 NT6576/NT6578 系列微處理機，將位於$8000h~$FFFFh 這段 32 Kbytes 的空間分配成 8 個 Bank 的特性，特別在此一 Bank 指示欄位中，將當時 Disassembly Window 中 Cursor 所在位置的資料的 Bank 值顯示出來，讓使用者知道他所檢查的指令，是位在那一個 Bank 之中。

由於 NT6576/NT6578 系列微處理機僅對$8000h~$FFFFh 這段空間加上區分 Bank 的功能，而對於$0000h~$7FFFh 則不作任何 Bank 處理，因此如果使用者將 Cursor 移到$0000h~$7FFFh 這段範圍時，則 Bank 指示將顯示 "Bank is [--]"。

# 程式計數器指示(Program Counter Indicator)

這個 "▶" 符號用來提醒使用者，NT6576/NT6578 系列微處理機內的 Program Counter 當時所指定的位址，此一位址也就是 NT6576/NT6578 系列微處理機在下一次處理指令時所要讀取資料的位址。




例如上圖所顯示，在 Disassembly Window 中，"▶"記號所指的位址是$B8DCh，因此右邊 CPU Register Window 中的 PC(Program Counter)所顯示的數值就是$B8DCh。同理，如果使用者改變 PC 的內含值，則 "▶" 記號所顯示的位置也會改變。

基本上，在 Reset 過後、執行 Trace/Step 指令之後，或是 Break 之後，"▶"記號會與 Cursor 位在同一個指令上，但是使用者可以單獨移動 Cursor 至其他位置，以便執行 "Till Cursor"或是設立 Breakpoint 等功能。

# 游標(Cursor)

Cursor 是一個供使用者參考的指標，使用者可以透過移動 Cursor 至需要的地址，然後設立 Breakpoint。或者在需要修改資料時，利用移動 Cursor 至希望位址，然後開啓附屬功能選單表(Sub-Function Menu)來進行修改資料的動作。當然，若使用者要執行 "Till Cursor" 之類的 Debug 動作時，Cursor 更是不可缺少的工具。

Cursor 與 Program Counter Indicator 的功能並不相同，Cursor 所提供的僅是參考功能，當 6502 CPU 開始執行指令時，所使用的位址則是 Program Counter Indicator 指示的位址。

# 中斷點(Breakpoint)

變成紅色字體的指令，代表該指令已被設立中斷點。NT657X ICE 在執行任一指令之前，會先檢查該指令是否已被設立 Breakpoint，若未設立，則該指令繼續執行，若該指令已被設爲 Breakpoint，則會在執行該指令之前，就停止軟體的繼續執行，並將控制權交棄 Symbol Debugger。

有關 Breakpoint 的詳細用法，請參考 Page 35 "Breakpoint" 一節的說明。

# 程式位址(Program Address)

該區域所顯示的是使用者程式的指令在整個 NT6576/NT6578 系列微處理機中的記憶體位置。位址的顯示是以 16 進制的數值方式來顯示，以 Word 爲單位，顯示範圍爲$0000h~$FFFFh，不包含 Bank。

程式位址在顯示時，是以指令長度爲單位，而且也只針對指令顯示。例如一個指令的長度爲 3 Bytes 時，則下一個指令的位址就是目前位址再加三。而其區分的方法則是依照 6502 組合語言的標準。

# 機械碼(Machine Code)

該區域所顯示的是使用者程式指令的機械碼。位址的顯示是以 16 進制的數值方式來顯示，以 Byte 爲單位，顯示範圍爲$00h~$FFh。

程式機械碼在顯示時雖然會以 Byte 爲單位，但是也會根據 6502 指令的編碼規則，來調整每一行所顯示的 Byte 數。例如機械碼 $4Ch 是 JMP 指令，而 JMP 指令是以 3 Byte 爲一個指令，因此該行就會以 $4Ch 爲開始，然後接著 2 Bytes 的資料。

當 Disassembly Window 遇到一個不屬於 6502 指令集的數值，而且該數值是每一行的第一個 Byte 時，Disassembly Window 會將該數值以獨立的一行顯示，其後不會再跟隨其他資料。

# 助憶碼(Symbol and Instruction)

該區域所顯示的是使用者程式指令的 6502 助憶碼，以及使用者自訂的 Symbol，以及程式所使用的 Label。顯示方式完全是以 ASC II 碼的字元來表示，因此看起來最像 Source Program。

Disassembly Window 在顯示 Symbol 時，會依循下列幾個原則。

**I.** 當 Disassembly Window 遇到一個不屬於 6502 指令集的數值，而且該數值是每一行的第一個 Byte 時，Disassembly Window 會將該數值以獨立的一行顯示，同時在指令部分是以 “DB” 來代表，然後跟著顯示該數值；

Example:

DB FF

**II.** 如果被顯示的指令隨後所跟隨的是一個 Address，則 Disassembly 會到使用者提供的 Symbol Table 中尋找與該 Address 相符合的 Symbol，如果找到則顯示符合的 Symbol，如果找不到，則直接顯示 16 進制的 Address；

**III.** 如果被顯示的指令是一個相對跳躍指令(Related Jump)，例如 BNE 或是 BEQ 指令，則 Disassembly 會根據隨後的 Offset 值來算出實際要跳躍的絕對位址，然後再到 Symbol Table 中找尋與該絕對位址相符合的 Symbol。如果找到則顯示符合的 Symbol，如果找不到則顯示由 Disassembly 所計算出來的絕對位址，以免顯示相對位址讓使用者看不懂；

**IV.** 如果被顯示的指令是一個讀取立即值或是利用立即值作運算的指令時，例如 ADC #01h，則 Disassembly 仍會將該值與 Symbol Table 中的值作比較，如果找到則將 Symbol 取代該立即值，如果找不到，則以 16 進制顯示該立即值；

Example:

```
Test        Equ #07h

            Lda #01h
            Ldy #07h
            :
經過 Disassembly 後會變成
            Lda #01h
            Ldy #Test
            :
```

# *Disassembly Window 在作 Scroll 時的注意事項*

**I.** Disassembly Window 的 Scroll Bar 控制依然有用，但是只能利用 "▲" 及 "▼" 二個控制鈕作每次往上或往下移動一行的控制，或是利用 Mouse 單擊 Scroll Control 的上方或下方作每次一個 Page 的捲動，除此之外，以 Mouse 直接拖曳 Scroll Control 的捲動方式將不能使用。

**II.** 在利用 "▲" 及 "▼" 二個控制鈕作往上或往下移動一行的控制時，是先移動 Disassembly Window 中的 Cursor，待 Cursor 移到 Disassembly Window 的最上方或最下方之後，若還繼續往上或下移動，才會真正捲動畫面中的資料。但是作 Page 捲動時，則沒有此種限制。

**III.** Disassembly Window 在 Reset 之後，會自動依據使用者程式中的 Reset 向量，找出程式的起始位址。因此 Reset 後的第一個畫面，Disassembly Window 的顯示一定很正確。

**IV.** 由 Reset 後的第一個畫面開始，Disassembly Window 在往下捲繞的時候，無論是一行一行的移動，或是以 Page 為單位的移動，由於都能以 Reset 向量為依據，因此移動的時候都能依據指令為單位，而正確移動並顯示正確結果。

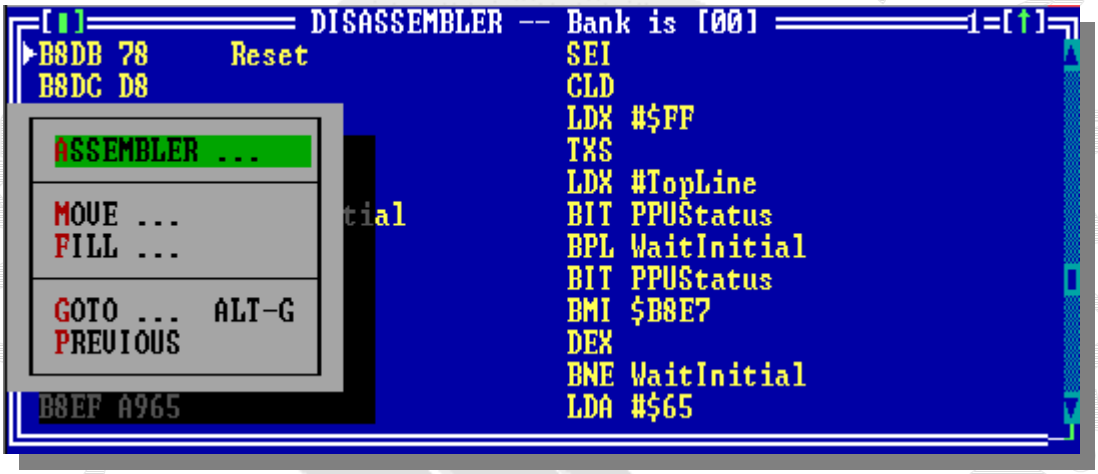
**V.** Disassembly Window 在作往下捲繞時，會自動紀錄每次移動的過程，因此在往下捲繞之後，若是再改成往上捲繞直到 Reset 後的程式開始處為止，Disassembly Window 的顯示狀況仍能維持正確。

**VI.** 如果往上捲繞的位置超過 Reset 後的程式進入點，由於無法猜測該往上移動 1 Byte 或是 2 Bytes、甚至 3 Bytes 才是正確的 OP Code 位置，因此 Disassembly Widow 會改成每次往上移動 1 Byte，然後由該 Address 開始作反組譯的顯示。但是由於該處的資料可能並不是一個指令的 OP Code，所以反組譯出來的結果有可能使整個畫面的程式都是錯誤的結果。

**VII.** 針對上述這種情形，使用者只要繼續向上移動，Disassembly Window 就會再往上移動 1 Byte，並顯示新的反組譯結果，所以最多只要向上移動三次(因為 6502 的指令最長只有 3 Bytes)，Disassembly Window 必能抓出正確的 OP Code，而反組譯結果也就會恢復正常。

# 附屬功能選單

Disassembly Window 除了有優秀的反組譯顯示功能以外，還具備立即的指令輸入能力、資料搬移、資料修改的功能。另外還有直接跳躍到某個位址、或是直接跳到某個 Symbol 後，在繼續顯示使用者程式的能力。這些功能都放在 Disassembly Window 的“附屬功能選單”之中，使用者能以滑鼠右鍵或是 Alt-F10 叫出這個“附屬功能選單”，而選單中的功能及使用方法則介紹如下。




## Assembler

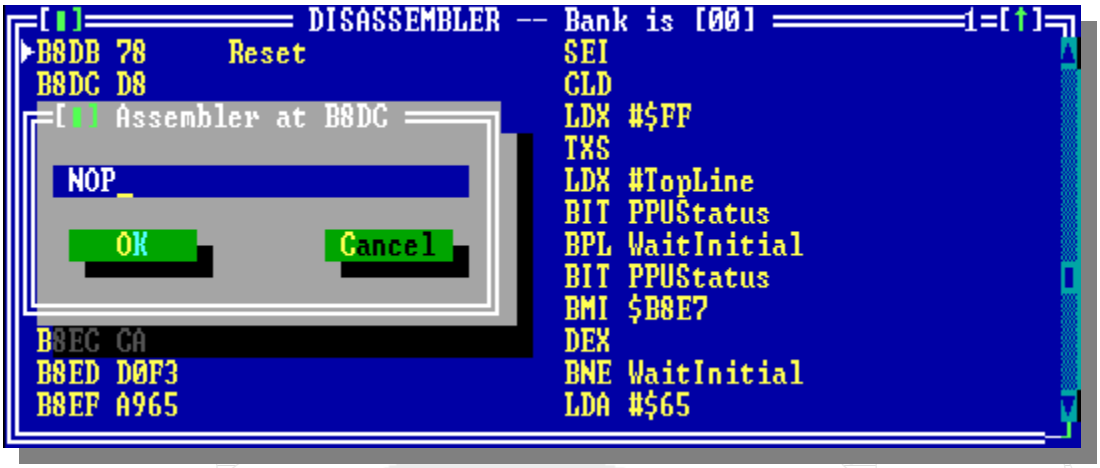
Assembler 功能允許使用者直接輸入 6502 的指令(注意，不是輸入 16 進制機械碼！)，來取代原先存在於 NT657X ICE 的程式。

要使用 Assembler 輸入組合語言，使用者可以有三種方式來開啓 Assembler 輸入功能。

**I.** 將 Cursor 移到所要修改的指令，然後按下 Space 鍵，即可進入 Assembler 輸入模式；

**II.** 以 Mouse 直接點選所要修改的指令二下，也可以進入 Assembler 輸入模式；

**III.** 將 Cursor 移到所要修改的指令，然後按下 F10。或者以 Mouse 的右鍵點選所要修改的指令一下，都可以開啓“附屬功能選單”，然後再選擇“Assembler”功能，一樣可以進入 Assembler 輸入模式；

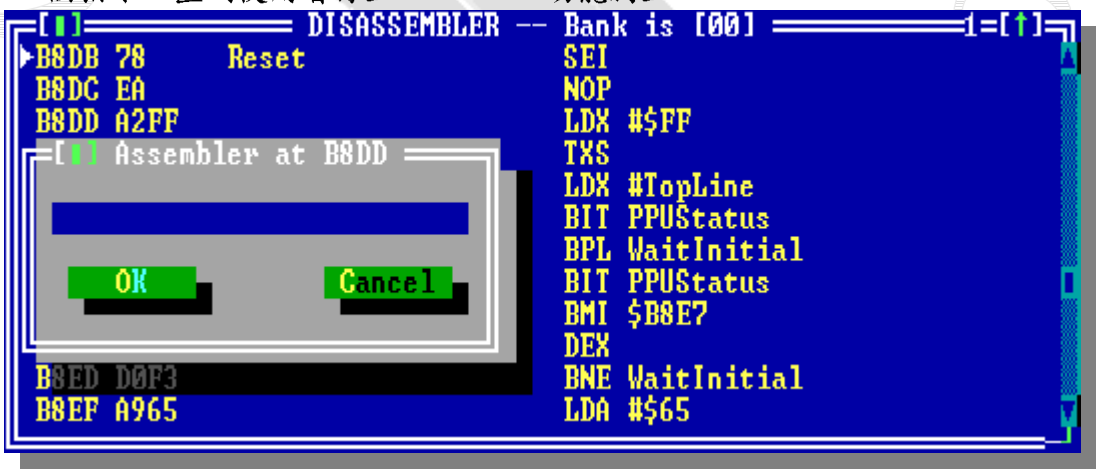



在進入 Assembler 輸入模式之後，Assembler 輸入視窗會出現在所要修改的指令的下方，同時視窗上會出現所要修改的 Address，接著使用者就可以輸入新的指令。

Symbol Debugger 的 Assembler 輸入功能可以接受合乎標準 6502 指令集語法的指令，同時指令中也允許輸入 Symbol Name 作爲變數名稱，或是副程式名稱。如果使用者輸入不合法的指令，或是指令中帶有不存在的 Symbol，則 Assembler 都會拒絕接受輸入。

在利用 Assembler 輸入新的指令時，必須注意新舊指令之間的指令長度是否有差異，以免因爲指令長度不同，而使 Assembler 出來的結果反而使程式遭到破壞。例如舊指令只有 2 Byte 長度，但是新指令卻有 3 Byte 長度，則 Assembler 在接受新指令之後，就會因爲新指令多出 1 Byte，而將下一個指令的頭一個 Byte 也挪過來供新指令使用，結果反而造成下一個指令被破壞掉。因此指令長度問題必須特別注意。

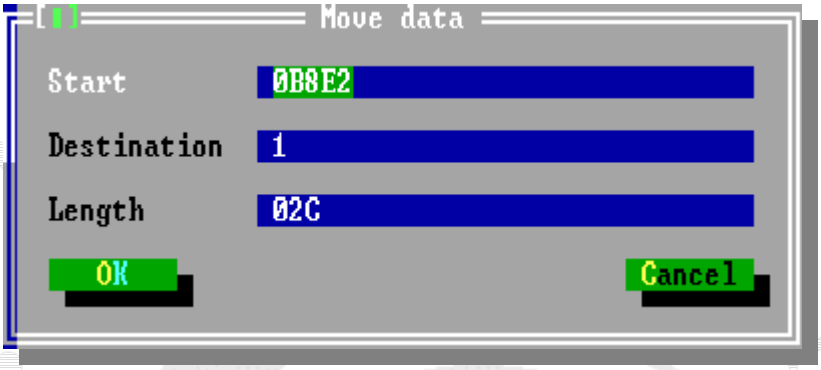
Assembler 在接受新指令之後，會自動將整個 Input Window 往下挪一行，以便使用者編輯下一個指令，直到使用者停止 Assembler 功能爲止。




若要結束輸入狀態，使用者只要按下 ESC 鍵、或者利用 Close Icon、或是以 Mouse 按下 Cancel 鍵，也直接按下 “OK” 或是鍵盤上的 Enter 鍵來輸入一個空的指令，都可以結束 Assembler 輸入狀態。

# Move

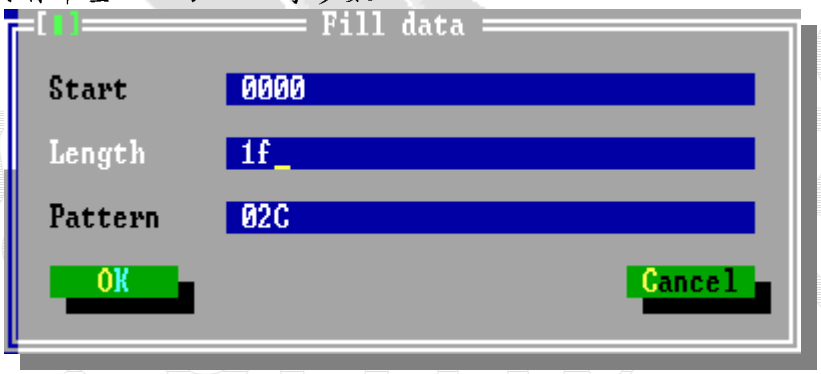
Move 功能是用來搬動存放於 NT657X ICE 的資料，此時會出現另一個供使用者輸入 Start Address、Data Length、及 Distention Address 的視窗。待相關參數設定完畢之後，Symbol Debugger 軟體就會開始搬移資料。




使用 Move 指令之前，以 Mouse 的右鍵點選所要搬移的指令並打開“附屬功能選單”之後，然後選用 Move 功能，此時被點選的指令的 Address，將自動出現在 Start Address 一欄，使用者只要再設定 Destination 及 Length 二項參數即可使用 Move 功能。

# Fill

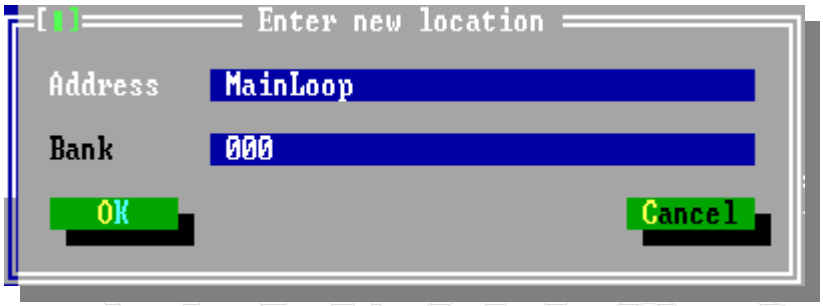
Fill 指令允許使用者將某一段區間都填上一個使用者設定的 16 進制數值。使用 Fill 功能時，Disassembly Window 會出現另一個供使用者輸入參數的視窗。在該視窗內，使用者必須輸入所要 Fill Data 區域的 Start Address、Fill Data 的 Length、以及使用者希望 Fill 的 Data 等參數。




使用 Fill 指令之前，以 Mouse 的右鍵點選所要 Fill 的指令並打開“附屬功能選單”之後，然後選用 Fill 功能，此時被點選的指令的 Address，將自動出現在 Start Address 一欄，使用者只要再設定 Length 及 Pattern 二項參數即可使用 Fill 功能。

# Goto

Goto 指令會使 Disassembly Window 的程式顯示位址受到影響。使用 Goto 功能會使 Disassembly Window 在顯示程式時，跳到各使用者所指定的位址上開始顯示程式。如果使用者在設定 Goto 參數時輸入錯誤的地址，則整個 Disassembly Window 的內容都會受影響而顯示錯誤。




使用 Goto 指令時，在 Disassembly Window 中的任何位置以 Mouse 右鍵開啓“附屬功能選單”之後，再選用 Goto 功能，然後再 Address 一欄輸入新的 Address 或是 Symbol Name，然後再輸入 Bank，而使用者在輸入時，可以輸入 Symbol 或是 16 進制的數值。

Goto 指令可以在打開“附屬功能選單”之後再選用，或是直接按 Alt+G 鍵來直接執行。

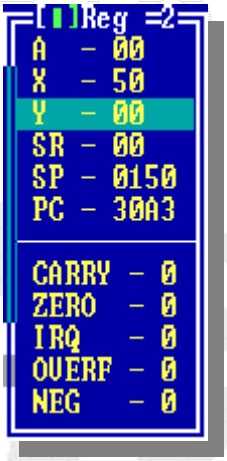
# Previous

<此功能有問題，請不要使用>

Previous 指令與 “Goto” 指令是相反的功能，“Goto”指令會使 Disassembly Window 跳到新的位旨趣顯示資料。而 Previous 功能則允許使用者跳回先前位址。

# 1-6 CPU 狀態視窗(CPU Status Window)

CPU 狀態視窗(CPU Status Window)會將 6502 CPU 之三個暫存器(A、X 及 Y Register)、一個 CPU 狀態暫存器(Status Register，SR)、一個堆疊指標暫存器(Stack Point，SP)以及 CPU 程式計數器(Program Counter，PC)的內容直接顯示在畫面上。




任何時刻，只要 Symbol Debugger 軟體取得 NT657X 發展系統的控制權，例如中斷之後，或是程式停止執行之後，Symbol Debugger 都會自動去讀取 6502 CPU 內的狀態，並將資料更新到 CPU 狀態視窗之內，以便使用者隨時掌握 6502 CPU 的反應。

Symbol Debugger 除了會將 6502 CPU 的狀態顯示出來之外，也允許使用者透過 CPU 狀態視窗更改個暫存器內的值。

## 顯示方式

基本上 A、X、Y 及 SR 都是以 Byte 為單位的 16 進制數值來顯示，而 SP 及 PC 則是以 Word 為單位的 16 進制數值顯示。但是考慮到 SR 內的資料都是以 bit 為單位的旗號(Flag)，因此在 CPU 狀態視窗的下方另有一半的畫面，將 SR 內的 Carry、Zero、Interrupt、Overflow 及 Negative Flag 等，以 bit 為單位單獨顯示。

> 注意：在 **NT6576/6578** 系列微處理機之內的 **6502 CPU**，並不具備 **10** 進制運算**(Decimal Operation)**的能力，因此原來 **6502 CPU** 內具有的 **Decimal Flag**，也因為沒有作用而沒有顯示出來。

由於 CPU Status Window 中需要顯示的資料有限，而且資料長度都是固定的，因此沒有調整 Window Size 的必要，所以 CPU Status Window 並沒有 Zoom Icon，Size Icon 及 Scroll Bar 等控制項目。

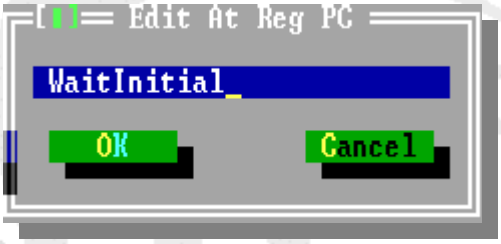
# 資料修改方式

在修改 CPU Status Window 中任一 Register 的內容之前，都必須先將 Cursor 移到將被修改的 Register。至於移動 Cursor 的方式可以用鍵盤上的上/下方向鍵控制，或者是直接以 Mouse 在要被修改的 Register 上以左鍵 Click 一下都可以。

修改資料時，CPU Status Window 提供三種修改方式，分別是直接輸入數值、以 Hot Key 增減數值、或是以 Toggle 方式改變旗號狀態等，詳細使用方法說明如下。

**I.** 直接輸入數值

當使用者要修改整個 Byte 或 Word 的資料時，可以在將 Cursor 移到被修改的 Register 之後，再按下 Space 鍵、或是 0~9 的數字鍵、或是以 Mouse 在要被修改的 Register 上以左鍵連續按二下、然後 CPU Status Window 就會出現一個 Input Window 來要求使用者要輸入數值。




使用者在輸入數值的時候，必須依照 Page 8 – “0 April 22, 1998” 一節中的 “資料輸入” 部分的規則來輸入 16 進制的數值或是 Symbol Name。如果使用者輸入的資料是錯誤的數值或是不存在的 Symbol Name，則 CPU Status Window 中的資料不會作任何改變。

若使用者未輸入任何值而直接按 Enter Key 或是選擇 Cancel、或是按下 ESC 鍵，則該 Register 原來的數值不變。

**II.** 以 Hot Key 增減暫存器內的數值

考慮到某些狀況下，使用者需要修改的數值與現有的數值差距不見得很大，此時若要使用者重新輸入一組完整的數值反而變得很麻煩，因此 Symbol Debugger 位這種情況提供一種直接增減數值的方法。使用者可以利用鍵盤右方，屬於數字鍵部分的灰色 “+” 和 “-“ 二個鍵，直接增減 Cursor 所對應到的 Register 內的數值。

灰色的 “+” 鍵，每一次可使 Register 內的數值加一，而 “-“ 鍵則會使 Register 內的數值每次減一。

**III.** 以 Toggle 方式修改 Flag 狀態

由於 Flag 部分都是以 bit 為單位，因此用一般的數值修改方法對他並不適用，所以 CPU Status Window 另外提供了利用 Space Bar 鍵來改變 Flag 的狀態。

在要改變 Flag 狀態之前，先將 Cursor 移到要修改的 Flag 上，然後按下 Space Bar 鍵，則該 Flag 的狀態就會一原來的狀態，作反向的變化。例如原來 Flag 的狀態是 1 ，使用者按下 Space Bar 後，該 Flag 就會變成 0 。

# 附屬功能選單

基本上 CPU Status Window 的操作功能並不複雜，僅有一些簡單的數值輸入功能而已，所以相對的附屬功能選單內的項目也很簡單，幾乎與前面所述的輸入方法一樣。此處仍需要附屬功能選單的目的，僅是爲了提供完全用 Mouse 來操作的使用者，能擁有與使用鍵盤來操作的使用者類似的功能。

在開啓附屬功能選單之前，使用者必須用 Mouse 將 Cursor 移到要修該資料的 Register 或是 Flag，然後在該 Register 或是 Flag 上按下 Mouse 的右鍵，則 CPU Status Window 就會開啓相對應的附屬功能選單。

因應 Register 與 Flag 輸入要求的差別，附屬功能選單也區分爲二種。

## Register 的附屬功能選單


**I.** Increase
此功能與前述所提到的鍵盤上右側數字鍵的灰色 “+” 鍵功能相同，都會將相對應之 Register 內的數值加 1 。

**II.** Decrease
此功能與前述所提到的鍵盤上右側數字鍵的灰色 “-” 鍵功能相同，都會將相對應之 Register 內的數值減 1 。

**III.** Zero
此功能會將相對應之 Register 內的數值設爲 $00h 或是 $0000h。

**IV.** SET
此功能會將相對應 Register 內的數值設爲 $FFh 或是 $FFFFh。

**V.** CHANGE
選用此功能會另外開啓 Input Window，讓使用者自行輸入數值，其輸入方法與輸入規則與前述相同(請參考 Page 56 “直接輸入數值” 一節)。

## Flag 的附屬功能選單


**I.** ZERO
此功能會將相對應的 Flag 設爲 0。

**II.** SET

此功能會將相對應的 Flag 設爲 1。

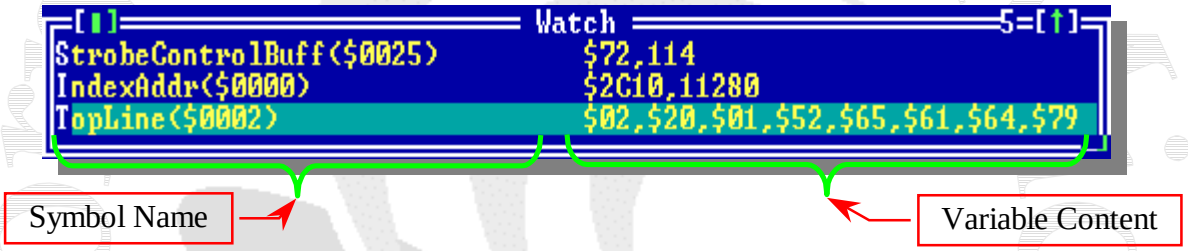
**III.** TOGGLE

此功能與前述的用 Space Bar 鍵的用法相同，選用此功能會使相對的 Flag 的內容產生 0 變 1 或是由 1 變 0 的作用。

# 1-7 變數檢查視窗(Watch Variable Window)

由於任何軟體在開發的時候，一定會用到很多使用者自行設定的變數，而在 Debug 過程中，這些使用者變數的變化情形，往往是軟體工程師最關切的，因此 Symbol Debugger 軟體特別提供此一變數檢查視窗(Watch Variable Window)，以便使用者自行設定所要檢查的變數，以及變數的資料長度。然後 Symbol Debugger 軟體會再每次取得控制權之後，自動將這些變數的最新資料內容予以更新顯示在 Watch Variable Window 之中，讓使用者在檢查自行設定的變數時，就像在檢查 CPU 的暫存器一般的容易。

在 Watch Variable Window 中，允許使用者設定的資料顯示長度有三種，分別是 Byte、Word 及 Array。Byte 是大家最熟悉也常用的變數長度，而 Word 則大部分適用於 Index Address 型態的變數，Array 則是資料已經預先設定爲 8 Bytes 長度，最適合用來檢查一塊當作 Buffer 的資料區。




## 顯示方式

Watch Variable Window 在顯示時，是以一行爲一個變數單位，不論該變數的資料長度是 Byte、Word 或是 Area，都是以一行爲一個單位。每一行又區分成二大部分，這二部分說明如下。

### Symbol Name

資料在顯示時，最左邊會顯示該變數的 Address 及 Symbol Name，其中 Symbol Name 是由 Symbol Debugger 軟體自動由 Symbol Table 中取出 Address 相符合者來顯示，至於該 Variable 的眞正 Address 則顯示在括弧中。

### Variable Content

在顯示內容時，Watch Variable Window 會同時顯示兩種數值，前面帶有 “$” 記號的是 16 進位數值，跟在後方的是 10 進制資料數值。請注意，當 Variable 採用 Array 方式時，爲了顯示 8 Bytes 的資料，因此 10 進至的資料就不再顯示。

### Cursor

在 Watch Variable Window 中的 Cursor，代表使用者當時可以對該 Variable 產生動作，而能夠使用的動作有 Insert、Delete 及 Modify 等，相關的動作說明，請參照後續的 Page 60 “資料修改” 一節。

Watch Variable Window 具有調整 Window Size 的能力，但是能調整的只有 Window 的上下尺寸，至於左右的寬度則不能調整，是固定不變的。

Watch Variable Window 在顯示資料時，並不會自動依 Variable 的 Address 或是 Symbol Name 作排序的動作，這是爲了考慮使用者在觀察 Variable 時，可能有順序上的要求，若是 Watch Variable Window 自行位使用者作排序處理，有可能破壞了使用者的觀察順序而造成不便。

注意：基本上，**Watch Variable Window** 除了可以檢查使用者的自訂變數之外，也可以用來檢查 **NT6576/NT6578** 系列微處理機內部的各項控制暫存器。但是在 **NT6576/NT6578** 系列微處理機內部的暫存器，有很多是設定成 **Write Only**，例如**$2000h** 及**$2001h** 等，都是僅能寫而不能讀的。像這類暫存器，若是利用 **Watch Variable Window** 來讀出資料內容，則將得到錯誤的結果。所以對這類 **Write Only** 的 **Register**，請利用 **Symbol Debugger** 軟體爲她們特別設計的 **Window** 來讀取(例如 **PPU Window)**，才能得到正確的結果。

# 資料修改

Watch Variable Window 除了顯示使用者自訂的變數與內容之外，也允許讓使用者直接修改變數的資料內容。同時使用者也可以利用 Watch Variable Window 自行決定增加或減少所要觀察的 Variable 數量，或是改變 Variable 的資料長度等。

再對所要觀察的 Variable 作任何改變之前，當然要先讓 Watch Variable Window 變成 Active Window，然後使用者必須先移動 Cursor 到任一 Variable 之後，才能進行資料修改的動作。

## 加入一個新的 Variable


使用者可以利用鍵盤上的 Ins 或 Insert 鍵，來告訴 Watch Variable Window 增加一個要被觀察的 Variable。待 Insert Variable Window 出現之後，使用者可以輸入變數的 16 進制 Address 或是變數的 Symbol Name，然後選擇想要的顯示長度。資料顯示長度的預設值爲 Byte，若使用者所需要的資料型態爲 Word 或是 Array，則可以透過資料長度的選擇項來修改。

使用者新加入的變數會顯示在 Cursor 的所在位置上，若當時 Watch Variable Window 不是空的，則原先在 Cursor 位置上的 Variable 會自動移到新加入的 Variable 之後。如果使用者希望將新加入的 Variable 放在整個 Watch Variable Window 的最後方，則必須先將 Cursor 移到 Watch Variable Window 的最後方之後，再使用 Insert 功能。

在增加一個新的 Variable 時，若使用者輸入的 Address 或是 Symbol Name 有誤，則該次輸入不會被接受。

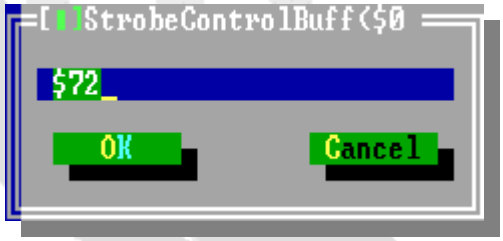
使用者新增一個 Variable 時，可以選擇所要顯示的資料長度，以及變數位址等，一旦輸入完畢，這些參數就不能再修改。使用者若是發現選擇錯誤，或是想要改變資料的顯示長度，則只有將該變數自 Watch Variable Window 中刪除，然後重新加入一個 Variable。

## 刪除一個 Variable

使用者如果想將 Watch Variable Window 中的某一個 Variable 刪除，則只要將 Cursor 移到該 Variable，然後按下鍵盤的 Del 或是 Delete 鍵，該變數就會被刪除，但是請注意，執行刪除 Variable 動作時，並不會有任何警告。

當 Variable 被刪除之後，原本排在該 Variable 後方的其他 Variable，會自動照順序遞補上來，而不會留下任何空缺。

## 修改 Variable 的資料




在使用者將 Cursor 移到任一 Variable 之後，使用者若按下 Space Bar 鍵或是以 Mouse 的左鍵雙擊該 Variable，則 Watch Variable Window 都會顯示另一個 Input Window 來讓使用者輸入新值。若使用者輸入的新值有錯，則被修改的 Variable 並不會改變其內含值。

## 附屬功能選單


使用者可以用 Mouse 的右鍵，直接點選已經存在於 Watch Variable Window 中的任一個 Variable，以便開啓附屬功能選單，執行修該變數的功能，或是插入一個新的 Variable。而附屬功能選單中的各項功能，則敘述如下。

## Ins Watch

此功能與鍵盤上的 Inset 鍵一樣，都是插入一個新的 Variable，同樣也會開啓另一個視窗，供使用者輸入相關參數。

## Del Watch

此功能與鍵盤上的 Delete 鍵一樣，都會將當時 Cursor 所在的 Variable 自 Watch Variable Window 中刪除。

## Inc

此功能用來改變所選定的 Variable 的內含值，每選一次即可讓該 Variable 的內含值加一。

## Dec

此功能用來改變所選定的 Variable 的內含值，每選一次即可讓該 Variable 的內含值減一。

## Zero

此功能用來使所選定的 Variable 的內含值變爲 0。

## Change…

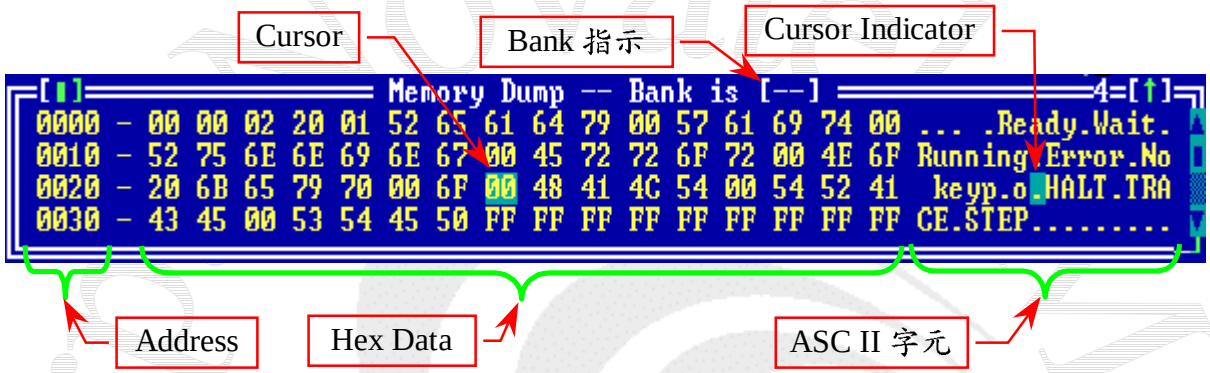
使用此功能就與按下 Space Bar 鍵一樣，都會出現另一個 Input Window 供使用者輸入新的值。

在 Watch Variable Window 中使用 Change 功能時，如果 Cursor 所在位置上並沒有設定要被 Watch 的 Variable，則 Change 命令將無法使用。

注意：**Inc**、**Dec**、**Zero**、**Change…**等四項功能，無法應用在屬於 **Array Type** 的 **Variable** 上，對 **Array** 型態的 **Variable** 使用上述四種指令，都不會有任何反應。

# 1-8 記憶體檢查視窗(Memory Dump Window)

Memory Dump Window 可以將 NT657X ICE 內，屬於 6502 CPU 所能控制的記憶體範圍($0000h~$FFFFh，其中$8000h~$FFFFh 又有 8 個 Bank)中的任一段記憶體，以 Byte 爲單位的 16 進制數值顯示在畫面上，同時畫面的右方還依相對的位置，以 ASC II 的字元型態來顯示，以便使用者能清楚了解資料的內容。




## 顯示方式

Memory Dump Window 在顯示資料時，會將整個資料分成三個部份，分別是 Address、Hex Data 及 ASCII 字元，再加上 Bank 指示、Cursor 等單元，這幾個單元分別解說如下。

### Bank 指示

此處 Bank 指示的作用，與 Disassembly Window 中的 Bank Indicator 相同，都是用來告訴使用者，當時 Cursor 所在位置的資料，是屬於哪一個 Bank。但是該 Bank 指示，僅在資料是位於 $8000h~$FFFFh 之間時才有作用，如果資料是位於 $0000h~$7FFFh 這塊區域時，則該 Bank 指示將顯示“Bank is [--]”。請參考 Page 47 第 1-5 節 “反組譯視窗(Disassembly Window)” 中的 “Bank 指示(Bank Indicator)”部份所述。

### Cursor

在 Memory Dump Window 中的 Cursor 是以 Byte 爲單位，使用者可以利用鍵盤上的方向鍵來移動 Cursor，或是以 Mouse 的左鍵直接點選任一 Byte，Cursor 就會直接移到此一 Byte 上。

如果以鍵盤移動 Cursor 時，使 Cursor 向上移動超過 Window 的邊界，或是在 Window 最左上角的 Byte 繼續向左移，都會使 Memory Dump Window 的內容向上捲動一行。同樣的，使 Cursor 向下移動超過 Window 的邊界，或是在 Window 最右下角的 Byte 繼續向右移，都會使 Memory Dump Window 的內容向下捲動一行。

同時使用者也能利用 Page Up 及 Page Down 二個鍵，來使整個 Window 內的資料每次上捲或下捲一個螢幕畫面的資料量。或者是以 Mouse 來控制 Scroll Bar，也能達到每次捲動一行或是捲動一個畫面的控制。

只有當 Cursor 移動到想要修改資料內容的 Byte 時，該 Byte 才具有資料變更的能力。同時 Cursor 也只能再 Hex Data 的區域內移動，不論如何捲繞(Scrolling)，Cursor 永遠都是在 Hex Data 的區域內。

# Cursor Indicator

Cursor Indicator 僅是一個象徵性的指示器，用來讓使用者了解目前 Cursor 所在位置的 Hex Data，所對應之 ASC II 字元資料的位置，以便使用者能夠方便對照 Hex Data 與 ASC II 字元之間的關係。

Cursor Indicator 無法由使用者直接控制，它只會依照 Cursor 的相對位置移動。

# Address

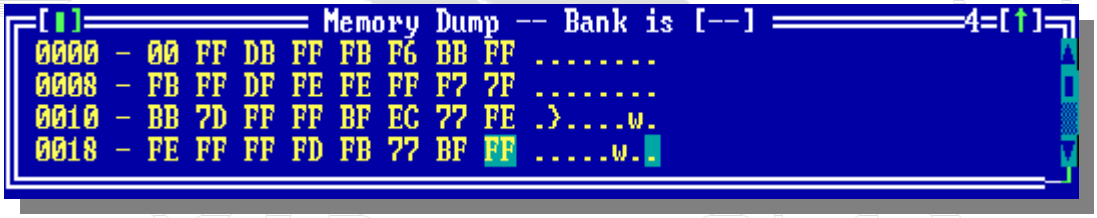
在這部份顯示的資料永遠都是以 Word 爲單位的 Address 資料，用來告知使用者目前所看的資料在記憶體中的位置，其範圍是$0000h~$FFFFh。

# Hex Data

Hex Data 是眞正的資料顯示部份，以 Byte 爲單位，每個 Bte 以二位數來顯示，資料範圍是$00h~$FFh，Byte 與 Byte 之間空一格以資區別。

Hex Data 的顯示寬度會依視窗的寬度而自動調整，但是基本上僅有三種變化，即最少每行 4 Bytes、或是每行 8 Bytes、最多每行 16 Bytes。

若是視窗的寬度所能顯示的 Byte 數介於二種規格之間，則 Memory Dump Window 自動取較少的資料量來顯示。例如當視窗的寬度僅夠顯示每行 15 Bytes 時，Memory Dump Window 眞正顯示的是每行 8 Bytes。




受每行最少顯示 4 Bytes 的影響，Memory Dump Window 再調整寬度時，會自動控制 Window 寬度不小於每行 4 Bytes。

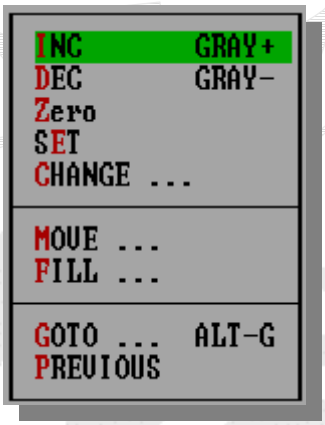
Hex Data 可以接受使用者的命令來改變內容，但是必續在將 Cursor 移到要改變的 Byte 之後，再以鍵盤右側的灰色 “+” 與 “-“ 二個鍵，或是開啓附屬功能選單後，才能改變 Hex Data 的內容。

# ASCII 字元

ASCII 字元顯示的資料，是直接對照左邊 Hex Data 的內容，然後將該內容以 ASCII 字元的型態表示在右邊的相對位置上。如果 Hex Data 的內容改變，ASCII 字元的顯示也自動改變。

ASCII 字元與左邊的 Hex Data 之間的對應，並不是每一個 Hex Data 都可以轉換成 ASC II 字元，僅有介於 $20h~$7Eh 之間數值的 Hex Data 會被轉變成 ASC II 字元，其他的數值，一律都以 “■” 符號代替。

## 附屬功能選單




### Inc

此功能會使 Cursor 所在之 Hex Data 的內含值加一，使用者也可以直接利用鍵盤右邊灰色部份的 “+” 鍵，來達到相同的功能。

### Dec

此功能會使 Cursor 所在之 Hex Data 的內含值減一，使用者也可以直接利用鍵盤右邊灰色部份的 “-” 鍵，來達到相同的功能。

### Zero

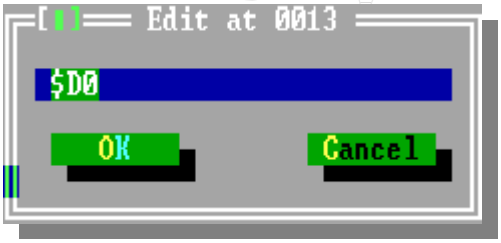
此功能會使 Cursor 所在之 Hex Data 的內含值變為 $00h。

### Set

此功能會使 Cursor 所在之 Hex Data 的內含值變為 $FFh。

### Change

此功能允許使用者修改 Cursor 所在之 Hex Data 的內含值，使用此功能之後，Memory Dump Window 會再出現一個 Input Window 供使用者輸入新的數值，該數值必須介於 $00h~$0FFh 之間。

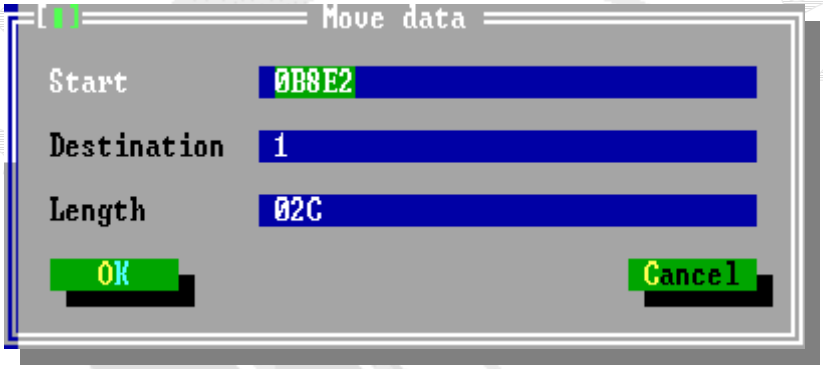


除了使用附屬選單中的 Change 功能之外，使用者也可以在將 Cursor 移到要修改的 Byte 之後，直接按 Space Bar 來進入 Input Window，或者是以 Mouse 的左鍵連續按所要修改的 Byte 二下，一樣可以進入 Input Window。

使用者在 Input Window 中輸入一個 Byte 數值並按了 Enter 鍵之後，Memory Dump Window 會自動將資料更新到 Hex Data 中相對應的 Byte 上，同時 Input Window 會自動移向下一個 Byte，以便使用者需要連續輸入數個 Byte。若是使用者想結束 Input Window，僅需按下鍵盤上的 ESC 鍵，或是以 Mouse 點選 Cancel 即可。如果使用者輸入錯誤，則 Cursor 所在位置的數值將不會有任何改變。

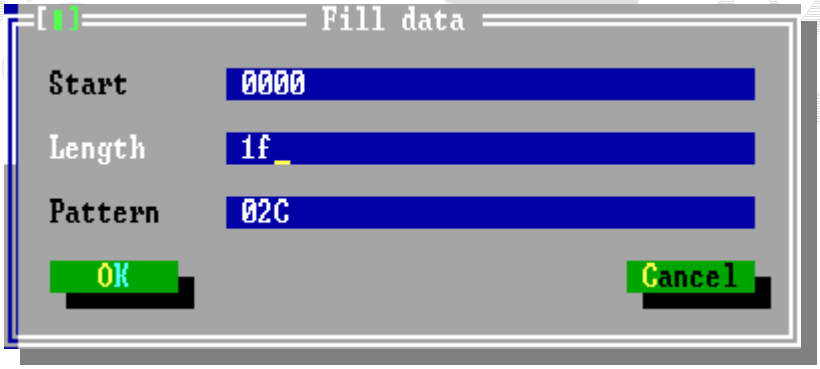
# Move

Move 功能是用來搬動存放於 NT657X ICE 的資料，此時會出現另一個供使用者輸入 Start Address、Data Length、及 Distention Address 的視窗。待相關參數設定完畢之後，Symbol Debugger 軟體就會開始搬移資料。



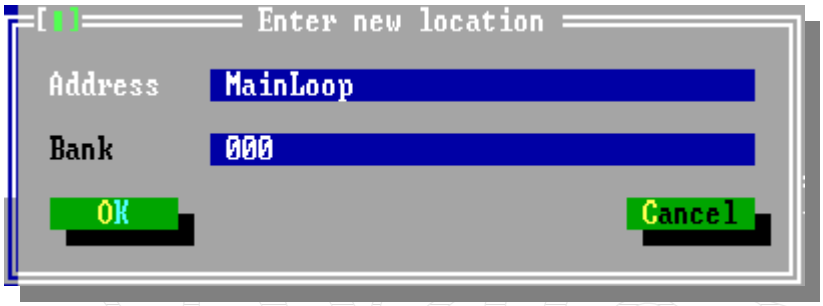

# Fill

Fill 指令允許使用者將某一段區間都填上一個使用者設定的 16 進制數值。使用 Fill 功能時，Memory Dump Window 會出現另一個供使用者輸入參數的視窗。在該視窗內，使用者必須輸入所要 Fill Data 區域的 Start Address、Fill Data 的 Length、以及使用者希望 Fill 的 Data 等參數。




# Goto

此處的 Goto 指令會使 Memory Dump Window 跳到使用者所指定的位址上開始顯示資料。如果使用者在設定 Goto 參數時輸入錯誤的地址，則整個 Memory Dump Window 的內容不會受影響。

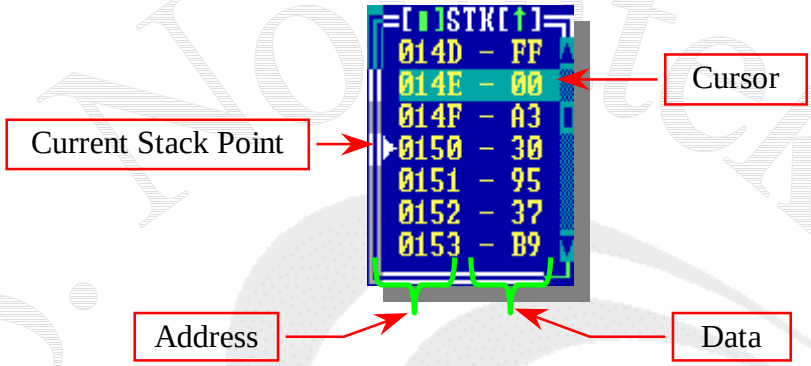



# Previous

<此功能在 Memory Dump Window 使用時會出問題，待測，請勿使用>

# 1-9 CPU 堆疊暫存器視窗(CPU Stack Window)

CPU Stack Window 可以將 NT657X ICE 內，屬於 6502 CPU 所使用的 Stack ($0100h~$01FFh)記憶體，以 Byte 爲單位的 16 進制數值顯示在視窗中，以便使用者能清楚了解 CPU 使用 Stack 的狀況。




## 顯示方式

CPU Stack Window 內的資料是以每行一個 Byte 爲單位來顯示，個部份代表的意義解說如下。

### Cursor

在 CPU Stack Window 中的 Cursor，是用來指示目前使用者所檢查之 Byte 的位置，同時該 Byte 也允許使用者下指令修改其內容。

### Current Stack Point

Current Stack Point 所代表的就是當時 6502 CPU 的 Stack Point，此一指標的位置會隨 CPU 的執行狀況而改變。由於 6502 CPU 在使用 Stack 的時候，是由高位址處開始，然後依序往低位址處 Push 資料。而 Pop 時，則是由低位址往高位址區移動，因此在 Current Stack Point 指標以下的資料，代表已被使用者用掉的 Stack，而 Current Stack Point 指標以上的，則使尚未使用的 Stack。

基本上，當 NT657X ICE 被 Reset 過之後或是 Symbol Debugger 軟體剛啓動時，此一指標的位置是指在$150h。但是在執行 Trace、Step 等 Debug 指令時，該指標將隨使用者的設定及 CPU 的執行狀況而改變。

### Address

此處所顯示的位址資料，用來幫助使用者在作 Debug 動作時，能夠輕易了解 CPU 對 Stack 的使用狀態，並且幫助使用者檢查 Stack 的內容。而所顯示的範圍，限定在 $0100h~$01FFh 之間。

## Data

此處所顯示的是 Stack 的資料內容，由於 Symbol Debugger 軟體也是利用 NT657X ICE 內的 6502 CPU 來執行通訊及 Monitor 的工作，因此當 S.D.在監控使用者的程式時，它也會需要使用 6502 的 Stack，止不過用的是使用者程式未使用的部份。因此當使用者在作 Debug 的過程中，將常會發現 CPU Stack Window 內非使用者使用的部份，其內含值經常改變，此乃正常現象，使用者不必在意。

# 附屬功能選單


## Inc

此功能會使 Cursor 所在之 Stack Data 內含值加一，使用者也可以直接利用鍵盤右邊灰色部份的“+”鍵，來達到相同的功能。

## Dec

此功能會使 Cursor 所在之 Stack Data 的內含值減一，使用者也可以直接利用鍵盤右邊灰色部份的“-”鍵，來達到相同的功能。

## Zero

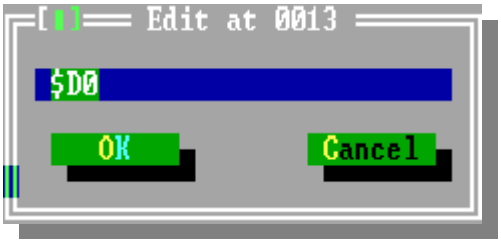
此功能會使 Cursor 所在之 Stack Data 的內含值變爲 $00h。

## Set

此功能會使 Cursor 所在之 Stack Data 的內含值變爲 $FFh。

## Change

此功能允許使用者修改 Cursor 所在之 Stack Data 的內含值，使用此功能之後，CPU Stack Window 會再出現一個 Input Window 供使用者輸入新的數值，該數值必須介於 $00h~$0FFh 之間。



除了使用附屬選單中的 Change 功能之外，使用者也可以在將 Cursor 移到要修改的 Byte 之後，直接按 Space Bar 來進入 Input Window，或者是以 Mouse 的左鍵連續按所要修改的 Byte 二下，一樣可以進入 Input Window。
若是使用者想結束 Input Window，僅需按下鍵盤上的 ESC 鍵，或是以 Mouse 點選 Cancel 即可。
如果使用者輸入錯誤，則 Cursor 所在位置的數值將不會有任何改變。

## Goto

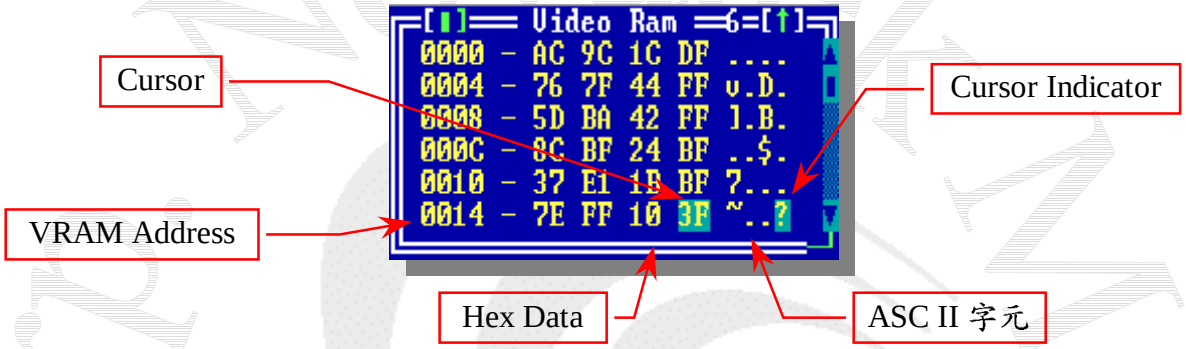
<此功能在 CPU Stack Window 沒有作用>

## Previous

<此功能在 CPU Stack Window 沒有作用>

# 1-10視訊記憶體視窗(VRAM Dump Window)

VRAM Dump Window 的功能就與 Memory Dump Window 的作用相同，只不過 VRAM Dump Window 所顯示的記憶體是屬於 PPU 所控制的圖形記憶體範圍($0000h~$FFFFh)。資料的顯示方法也與 Memory Dump Window 相同，都是以 Byte 爲單位的 16 進制數值顯示在畫面上，同時畫面的右方還依相對的位置，以 ASC II 的字元型態來顯示，以便使用者能清楚了解資料的內容。




## 顯示方式

VRAM Dump Window 在顯示資料時，也是一樣將資料分成三個部份，分別是 Address、Hex Data 及 ASCII 字元，以及 Cursor、Cursor Indicator 等單元，但是沒有 Bank 指示這部份(因爲 NT657X 系列微處理機的 VRAM 並不分 Bank)。

### Cursor

在 VRAM Dump Window 中的 Cursor 也是以 Byte 爲單位，使用者可以利用鍵盤上的方向鍵來移動 Cursor，或是以 Mouse 的左鍵直接點選任一 Byte，Cursor 就會直接移到此一 Byte 上。

如果以鍵盤移動 Cursor 時，使 Cursor 向上移動超過 Window 的邊界，或是在 Window 最左上角的 Byte 繼續向左移，都會使 Memory Dump Window 的內容向上捲動一行。同樣的，使 Cursor 向下移動超過 Window 的邊界，或是在 Window 最右下角的 Byte 繼續向右移，都會使 Memory Dump Window 的內容向下捲動一行。

同時使用者也能利用 Page Up 及 Page Down 二個鍵，來使整個 Window 內的資料每次上捲或下捲一個螢幕畫面的資料量。或者是以 Mouse 來控制 Scroll Bar，也能達到每次捲動一行或是捲動一個畫面的控制。

只有當 Cursor 移動到想要修改資料內容的 Byte 時，該 Byte 才具有資料變更的能力。同時 Cursor 也只能再 Hex Data 的區域內移動，不論如何捲繞(Scrolling)，Cursor 永遠都是在 Hex Data 的區域內。

# Cursor Indicator

Cursor Indicator 僅是一個象徵性的指示器，用來讓使用者了解目前 Cursor 所在位置的 Hex Data，所對應之 ASC II 字元資料的位置，以便使用者能夠方便對照 Hex Data 與 ASC II 字元之間的關係。

Cursor Indicator 無法由使用者直接控制，它只會依照 Cursor 的相對位置移動。

# Address

在這部份顯示的資料永遠都是以 Word 爲單位的 Address 資料，用來告知使用者目前所看的資料在 VRAM 中的位置，其範圍是$0000h~$FFFFh。

# Hex Data

Hex Data 是眞正的資料顯示部份，以 Byte 爲單位，每個 Bte 以二位數來顯示，資料範圍是$00h~$FFh，Byte 與 Byte 之間空一格以資區別。

Hex Data 的顯示寬度會依視窗的寬度而自動調整，但是基本上僅有三種變化，即最少每行 4 Bytes、或是每行 8 Bytes、最多每行 16 Bytes。

若是視窗的寬度所能顯示的 Byte 數介於二種規格之間，則 Memory Dump Window 自動取較少的資料量來顯示。例如當視窗的寬度僅夠顯示每行 15 Bytes 時，Memory Dump Window 眞正顯示的是每行 8 Bytes。

受每行最少顯示 4 Bytes 的影響，Memory Dump Window 再調整寬度時，會自動控制 Window 寬度不小於每行 4 Bytes。

Hex Data 可以接受使用者的命令來改變內容，但是必續在將 Cursor 移到要改變的 Byte 之後，再以鍵盤右側的灰色 “+” 與 “-“ 二個鍵，或是開啓附屬功能選單後，才能改變 Hex Data 的內容。

# ASCII 字元

ASCII 字元顯示的資料，是直接對照左邊 Hex Data 的內容，然後將該內容以 ASCII 字元的型態表示在右邊的相對位置上。如果 Hex Data 的內容改變，ASCII 字元的顯示也自動改變。

ASCII 字元與左邊的 Hex Data 之間的對應，並不是每一個 Hex Data 都可以轉換成 ASC II 字元，僅有介於 $20h~$7Eh 之間數值的 Hex Data 會被轉變成 ASC II 字元，其他的數值，一律都以 “.” 符號代替。

# 附屬功能選單


## Inc

此功能會使 Cursor 所在之 Hex Data 的內含值加一，使用者也可以直接利用鍵盤右邊灰色部份的 “+” 鍵，來達到相同的功能。

## Dec

此功能會使 Cursor 所在之 Hex Data 的內含值減一，使用者也可以直接利用鍵盤右邊灰色部份的 “-” 鍵，來達到相同的功能。

## Zero

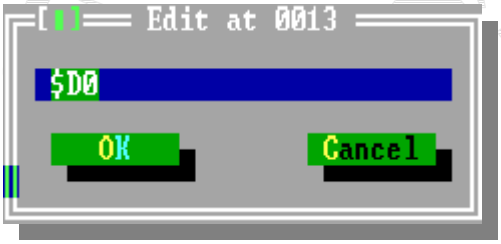
此功能會使 Cursor 所在之 Hex Data 的內含值變為 $00h。

## Set

此功能會使 Cursor 所在之 Hex Data 的內含值變為 $FFh。

## Change

此功能允許使用者修改 Cursor 所在之 Hex Data 的內含值，使用此功能之後，VRAM Dump Window 會再出現一個 Input Window 供使用者輸入新的數值，該數值必須介於 $00h~$0FFh 之間。

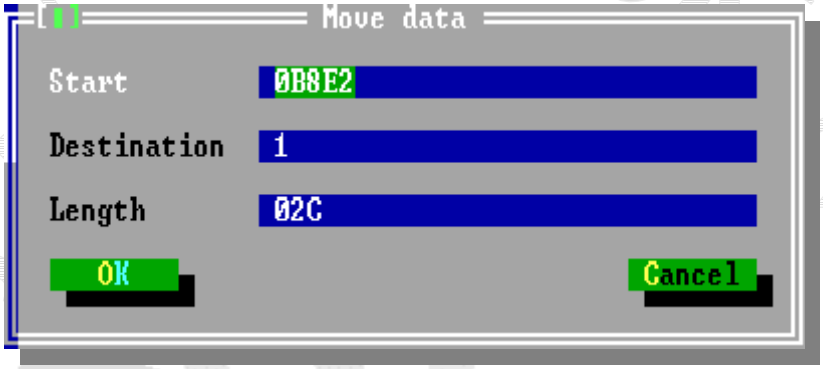



除了使用附屬選單中的 Change 功能之外，使用者也可以在將 Cursor 移到要修改的 Byte 之後，直接按 Space Bar 來進入 Input Window，或者是以 Mouse 的左鍵連續按所要修改的 Byte 二下，一樣可以進入 Input Window。

使用者在 Input Window 中輸入一個 Byte 數值並按了 Enter 鍵之後，VRAM Dump Window 會自動將資料更新到 Hex Data 中相對應的 Byte 上，同時 Input Window 會自動移向下一個 Byte，以便使用者需要連續輸入數個 Byte。若是使用者想結束 Input Window，僅需按下鍵盤上的 ESC 鍵，或是以 Mouse 點選 Cancel 即可。如果使用者輸入錯誤，則 Cursor 所在位置的數值將不會有任何改變。

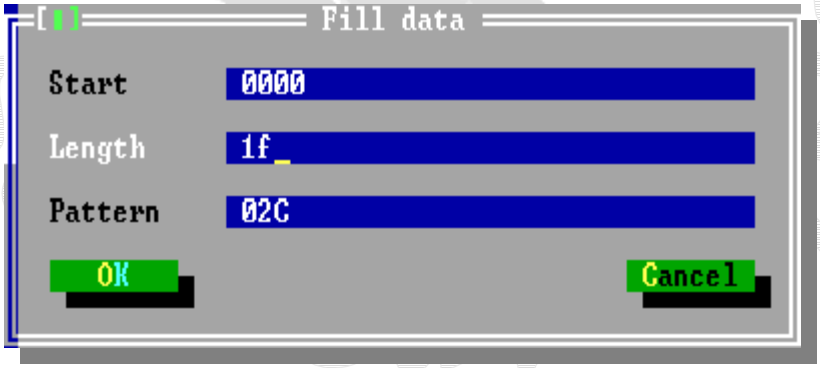
# Move

Move 功能是用來搬動存放於 VRAM 的資料，此時會出現另一個供使用者輸入 Start Address、Data Length、及 Distention Address 的視窗。待相關參數設定完畢之後，Symbol Debugger 軟體就會開始搬移資料。




# Fill

Fill 指令允許使用者將某一段區間都填上一個使用者設定的 16 進制數值。使用 Fill 功能時，VRAM Dump Window 會出現另一個供使用者輸入參數的視窗。在該視窗內，使用者必須輸入所要 Fill Data 區域的 Start Address、Fill Data 的 Length、以及使用者希望 Fill 的 Data 等參數。




# Goto

此處的 Goto 指令會使 VRAM Dump Window 的所顯示的資料，跳到使用者所指定的位址上開始顯示。如果使用者在設定 Goto 參數時輸入錯誤的地址，則整個 Disassembly Window 的內容都會受影響而顯示錯誤。

**注意：Goto 功能在 VRAM Window 使用時，請勿輸入 Symbol，若是使用 Symbol，有可能會造成當機；**

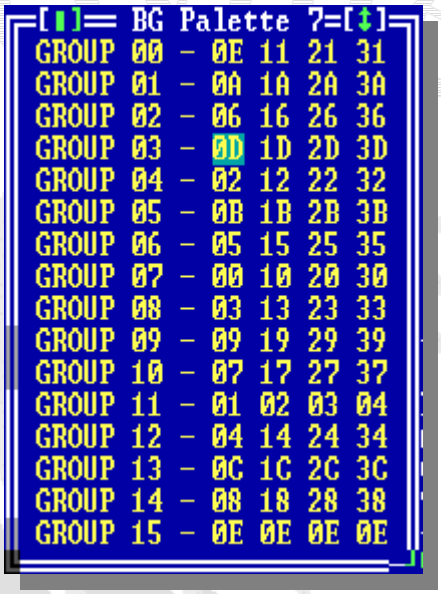
# Previous

<此功能在 VRAM Dump Window 使用時會出問題，待測，請勿使用>

# 1-11調色盤資料視窗(Color Palette Window)

Color Palette Window 可以將 NT6576/NT6578 系列微處理機內的調色盤資料顯示出來，它是以 16 進制的 Byte 格式來顯示資料。

## 顯示方式




基本上 Palette Window 在顯示調色盤資料時，是以 NT6576/NT6578 系列微處理機的 4-Color 模式爲基礎，將整個 64 Bytes 的 Palette Data 區分成 16 個 Group (Group 0~Group 15)，每個 Group 各有 4 Bytes 的 Palette Data，然後以 Group 爲單位，每行顯示一個 Group 的資料。

在 Palette Window 中也有 Cursor，此處 Cursor 的作用也是在於標示允許作資料修改的 Palette Data。

由於 Palette Window 最大僅有 16 個 Group，因此如上圖所示，在所有的 Group 都顯示出來之後，Palette Window 的高低尺寸就不能再調整。而左右寬度也是由於每行 4 Bytes 的關係，因此左右寬度是固定不變的。

由於在眞正的 IC 中，Palette 內的 Data 在 Reset 過之後，是呈現 Unknow 狀態，因此 Palette Window 內的資料在 NT657X ICE 執行 Reset 命令之後，是呈現不穩定的狀態，此時若是試圖修改 Palette Window 內的資料，將得到不穩定的輸出。

等 NT657X ICE 執行使用者的程式而且 Break 下來之後，Palette Window 內的資料才會隨使用者的設定而改變，在此時才允許使用者眞正改變 Palette Window 內的資料。但是如果使用者下令要 NT657X ICE 執行 Reset 動作，則 Palette Window 內的資料又會變成不穩定狀態。

# 附屬功能選單


## Inc

此功能會使 Cursor 所在之 Palette Data 的內含值加一，使用者也可以直接利用鍵盤右邊灰色部份的“+”鍵，來達到相同的功能。

## Dec

此功能會使 Cursor 所在之 Palette Data 的內含值減一，使用者也可以直接利用鍵盤右邊灰色部份的“-”鍵，來達到相同的功能。

## Zero

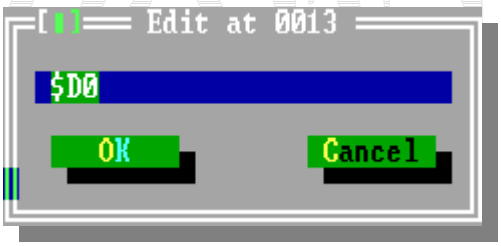
此功能會使 Cursor 所在之 Palette Data 的內含值變為 $00h。

## Set

此功能會使 Cursor 所在之 Palette Data 的內含值變為 $3Fh。

## Change

此功能允許使用者修改 Cursor 所在之 Palette Data 的內含值，使用此功能之後，Palette Window 會再出現一個 Input Window 供使用者輸入新的數值，該數值必須介於 $00h~$3Fh 之間，若數值超過$3Fh，則 Palette Window 會自動將數值轉成 $3Fh。




除了使用附屬選單中的 Change 功能之外，使用者也可以在將 Cursor 移到要修改的 Palette Data 之後，直接按 Space Bar 來進入 Input Window，或者是以 Mouse 的左鍵連續按所要修改的 Byte 二下，一樣可以進入 Input Window。

使用者在Input Window中輸入一個Byte數值並按了Enter鍵之後，Palette Window會自動將資料更新到Palette Data中相對應的Byte上，同時Input Window會自動移向下一個Byte，以便使用者需要連續輸入數個Byte。若是使用者想結束Input Window，僅需按下鍵盤上的ESC鍵，或是以Mouse點選Cancel即可。
如果使用者輸入錯誤，則Cursor所在位置的數值將不會有任何改變。

# Move

Move功能是用來搬動存放於Color Palette內的資料，此時會出現另一個供使用者輸入Start Address、Data Length、及Distention Address的視窗。待相關參數設定完畢之後，Symbol Debugger軟體就會開始搬移資料。

# Fill

Fill指令允許使用者將某一段區間都填上一個使用者設定的16進制數值。使用Fill功能時，Palette Dump Window會出現另一個供使用者輸入參數的視窗。在該視窗內，使用者必須輸入所要Fill Data區域的Start Address、Fill Data的Length、以及使用者希望Fill的Data等參數。

# Goto

<此功能在CPU Stack Window沒有作用>

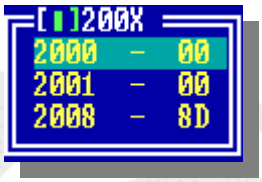
# Previous

<此功能在CPU Stack Window沒有作用>

# 1-12PPU 暫存器視窗(PPU Register Window)

由於 NT6576/NT6578 系列微處理機在圖形處理部份(PPU)的設計較爲特殊，因此某些控制暫存器(如 $2000h、$2001 等)，都是設計成 Write Only 的模式，如果使用者在 Debug 過程中想讀回這些暫存器內的資料，所得到的將是 Unknow 的數值。

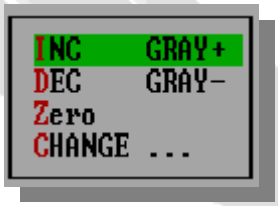
爲了解決使用者在 Debug 時遇到的此種困擾，NT657X ICE 特別在硬體線路上設計一組特別線路，專門讀回這些 Write Only 的 PPU Register，並提供這一個視窗讓使用者容易操作。




## 顯示方式

PPU Register Window 在顯示時，是以每一行顯示一個 Register，而 Register 的內容則是以 Byte 爲單位的 Hex 格式來顯示。整個 PPU Register Window 僅列出三個最常用到的 Register，其餘的 Register 不是可以 Read 回來，就是用的次數並不多，所以未出現在這個視窗中。

## 附屬功能選單




### Inc

此功能會使 Cursor 所在之 Palette Data 的內含值加一，使用者也可以直接利用鍵盤右邊灰色部份的“+”鍵，來達到相同的功能。

### Dec

此功能會使 Cursor 所在之 Palette Data 的內含值減一，使用者也可以直接利用鍵盤右邊灰色部份的“-”鍵，來達到相同的功能。

### Zero

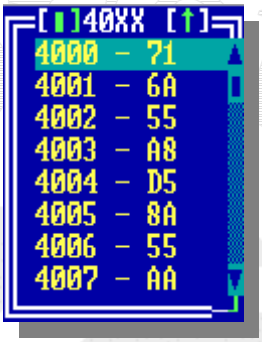
此功能會使 Cursor 所在之 Palette Data 的內含值變爲 $00h。

# Change

此功能允許使用者修改 Cursor 所在之 Data 的內含值，使用者輸入介於 $00h~$FFh 之間的數值。

# 1-13週邊介面視窗(Peripheral I/O Window)

Periphal I/O Window 所顯示的是位於 $40xx~$43XX 之間的資料，在 NT657X 系列微處理機的設計中，這段範圍是用來與 PC 的 Super I/O 晶片作聯繫用的，以便使用者連接 FDD 或是 MPEG Decoder 等 PC 所使用的元件。

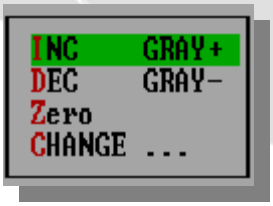



## 顯示方式

Periphal I/O Window 在顯示時，是以每一行顯示一個 Byte 的 Hex 格式來顯示。若是使用者有在 $40XX~$43XX 這各區域連接 Super I/O 晶片(例如 ITE8661 Super I/O Chip)或是 MPEG Decoder，則本視窗將顯示出該 Super I/O 晶片上相對位址之 Register 的內容，但是如果使用者位在上述區域加裝任何原件，則本視窗所顯示的值都是無意義的。

與其他視窗一樣，僅有 Cursor 所在的資料，可以進行資料修改的動作。

## 附屬功能選單




### Inc

此功能會使 Cursor 所在之 Periphal I/O Data 的內含值加一，使用者也可以直接利用鍵盤右邊灰色部份的 “+” 鍵，來達到相同的功能。

### Dec

此功能會使 Cursor 所在之 Periphal I/O Data 的內含值減一，使用者也可以直接利用鍵盤右邊灰色部份的 “-” 鍵，來達到相同的功能。

### Zero

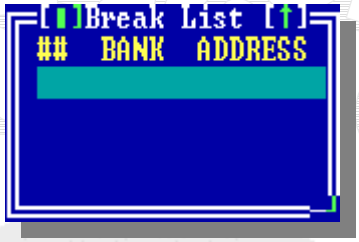
此功能會使 Cursor 所在之 Periphal I/O Data 的內含值變為 $00h。

# Change

此功能允許使用者修改 Cursor 所在之 Periphal I/O Data 的內含值，使用者輸入介於 $00h~$FFh 之間的數值。

# 1-14 中斷點視窗 (Break Point List Window)

Break Point List Window 會將使用者在 Disassembly Window 所設定的所有 Break Point，集中列出顯示在此一視窗中顯示出來，但是顯示出來的 Break Point 僅供參考之用，使用者無法透過 Break Point List Window 作任何修改。不能 Delete，也不能改變 Break 條件。



# 1-15搖桿狀態視窗(Joystick Status Window)

<本視窗雖然有顯示資料，但是無法反應 Joystick 的狀態，請勿使用>