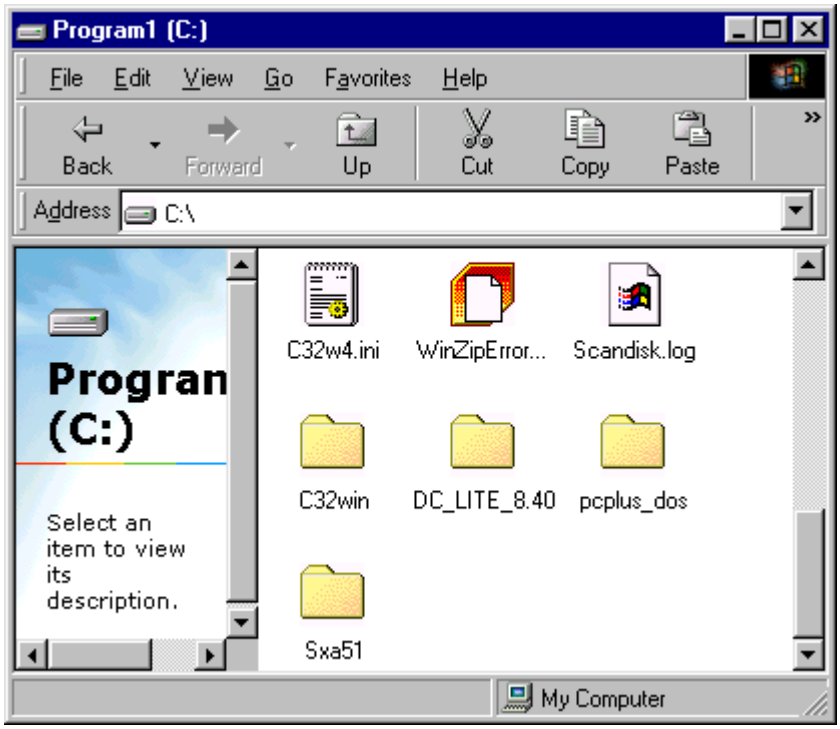
คู่มือการใช้งานโปรแกรม SXA51

โปรแกรม SXA51.exe คือ โปรแกรมแอสเซมเบลอร์ชนิดหนึ่งที่ทำหน้าที่แปลภาษาแอสเซมบลีให้เป็นรหัสคำสั่ง (Opcode) แบบ “ Intel Hex Format ” หรือ HEX file นั้นเอง ซึ่งรองรับไมโครคอนโทรลเลอร์ตระกูล MCS-51

เนื้อหาภายในคู่มือฉบับนี้จะแบ่งออกเป็น 4 ส่วน คือ 1. การติดตั้งโปรแกรม SXA51.exe, 2. การใช้งานโปรแกรม SXA51.exe ในโหมด Comman Line , 3. การใช้งานโปรแกรม SXA51.exe บน Windows ผ่านโปรแกรม EditPlus ซึ่งผู้อ่านสามารถดาว์นโหลดได้จากเวปไซด์ www.editplus.com หรือดาว์นโหลดจาก www.thaiware.com ซึ่งเป็นเวอร์ชั่น Shareware, 4. ไวยากรณ์ใน SXA51.EXE ,เพื่อไม่ให้เป็นการเสียเวลามาเริ่มต้นกันเลย!!!

การติดตั้งโปรแกรม SXA51.exe

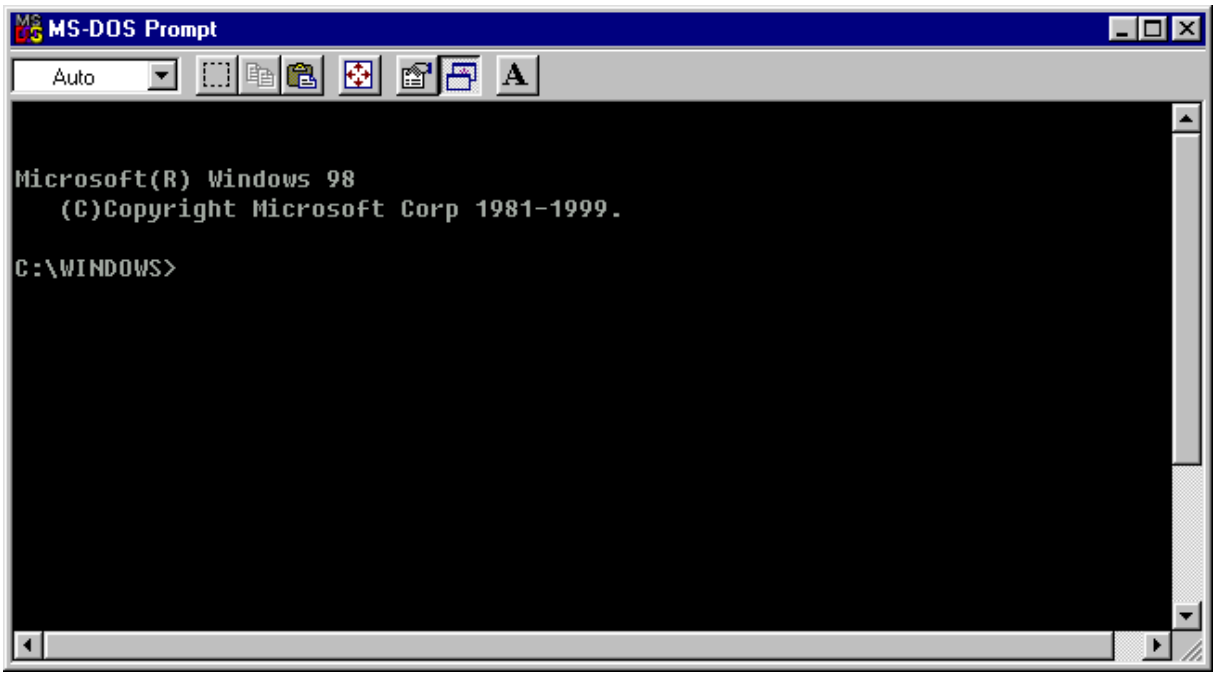
การติดตั้งโปรแกรม SXA51.EXE นั้น ไม่มีอะไรยุ่งยากครับ เริ่มแรกให้ผู้อ่าน Copy ไฟล์โปรแกรมซึ่งอยู่ใน CD-ROM ที่ทาง ETT แถมไปกับบอร์ดทดลองซึ่งอยู่ในโฟล์เดอร์ ...\Tools\sxa51 โดยให้ Copy มาทั้งโฟล์เดอร์เลยจากนั้นนำไปวางไว้ที่ไดร์ C:\ (จริงๆ แล้วจะวางไว้ที่ไหนก็ได้ครับ) เท่านี้เป็นอันเสร็จพิธีครับ..!



รูปแสดงตำแหน่งไฟล์ที่ติดตั้ง

การใช้งานโปรแกรม SXA51.exe ในโหมด Comman Line

การแปลภาษาแอสเซมบลีด้วยวิธีนี้ผู้อ่านจะต้องทำงานอยู่บนโหมด DOS ซึ่งวิธีการเข้าโหมด DOS นี้ทำได้โดย คลิ๊กที่ปุ่ม Start => Programs => MS-DOS Prompt



รูปแสดงการเข้าโหมด DOS

ขั้นที่ 1 : เมื่อผู้อ่านเข้าโหมด Dos เป็นที่เรียบร้อยแล้ว จากนั้นให้ใช้คำสั่ง “CD” เพื่อเข้าไปในโฟล์เดอร์ชื่อ C:\Sxa51 (ที่เราวางโปรแกรมไว้ตอนต้น)

C:WINDOWS>CD\ ◄..... ออกทุกห้อง

C:\>CD SXA51 ◄..... เข้าห้อง SXA51

ขั้นที่ 2 : เมื่อผู้อ่านเข้ามาในห้อง SXA51 ที่อยู่ในไดร์ C แล้ว จากนั้นผู้อ่านต้องแน่ใจว่าในห้องนี้ได้มีไฟล์โปรแกรมที่ผู้อ่านได้เขียนขึ้น และ ได้ Save เป็นนามสกุลจุด “ASM” เช่นในที่นี้ผู้เขียนมีไฟล์ชื่อ P0.ASM อยู่ในห้อง C:\SXA51 (ซึ่งอยู่ในห้องเดียวกันกับตัวโปรแกรม SXA51.EXE Compiler) จากนั้นผู้เขียนจะเริ่มแปลโปรแกรมเป็นไฟล์ “ Intel Hex Format ” หรือ HEX file โดยใช้คำสั่งดังนี้ คือ

เว้นวรรค

```
C:\SXA51>SXA51 <Option> FileName.ASM
```

ชื่อไฟล์ ความยามไม่เกิน 8 ตัวอักษร (ไม่รวมนามสกุล)

Option :

- L = Make a Listing file (filename.lst)
- D = Be verbose
- N = Do not make an object (filename.hex)
- C = Include a symbol cross reference listing

รายละเอียดอ่านหัวข้อถัดๆ ไป

Example : **กรณีที่ไฟล์ P0.asm อยู่ในห้องเดียวกับโปรแกรม SXA51.exe**

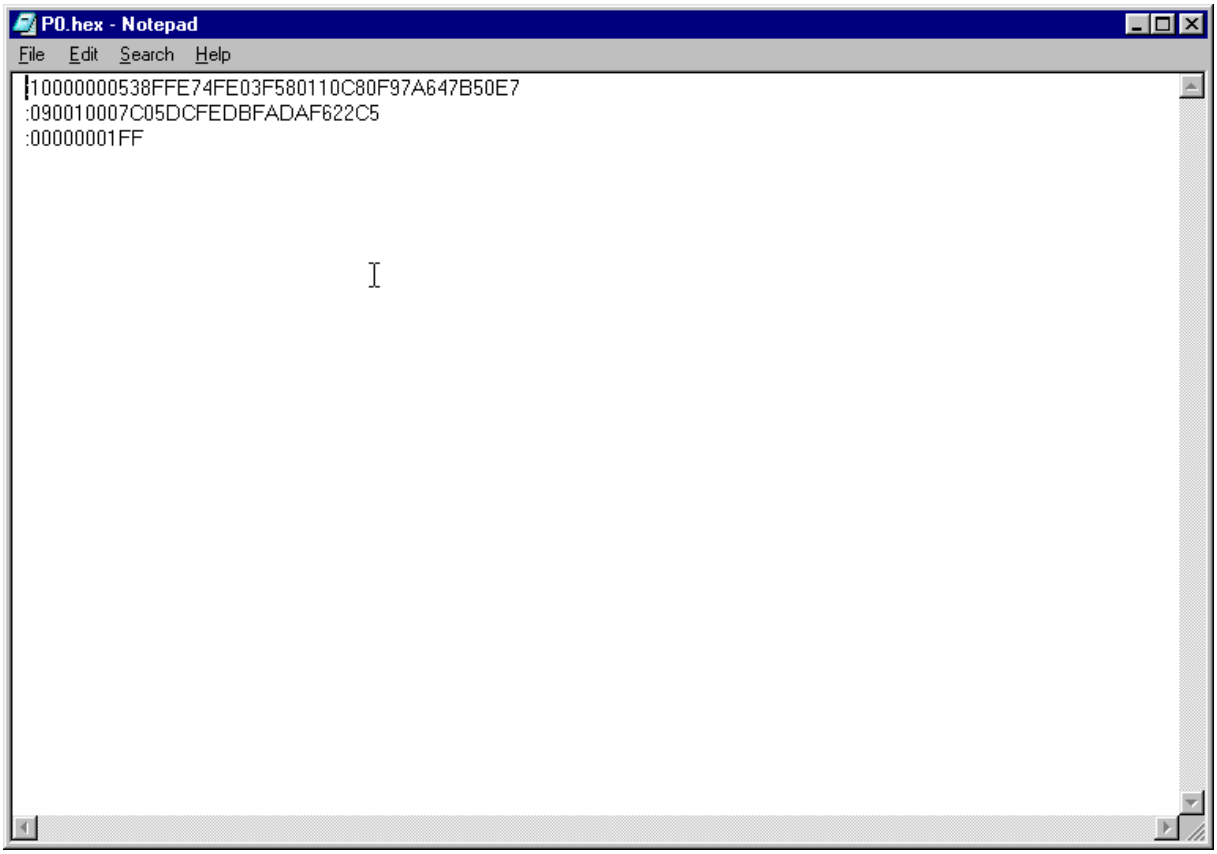
```
C:\SXA51>SXA51 P0.ASM
```

Example : **กรณีที่ไฟล์ P0.asm อยู่คนละห้องกับโปรแกรม SXA51.exe เช่นอยู่ใน D:\ EXAM**

```
C:\SXA51>SXA51 D:\EXAM\P0.ASM
```

ขั้นที่ 3 : เมื่อมาถึงตรงนี้ผู้อ่านจะได้ไฟล์นามสกุล “ . HEX “ แล้วครับ แต่ยังไม่สามารถนำไฟล์นี้ไปใช้ในการดาว์นโหลดโปรแกรมได้

ขั้นที่ 4 : ทำการเปิดไฟล์นามสกุล .HEX ที่สร้างขึ้นจากนั้นตอนก่อนหน้านี้ด้วยโปรแกรม Notepad จากนั้นให้ทำการเลื่อนข้อความในโปรแกรม Notepad ขึ้นมา 1 บรรทัด (เนื่องจากบรรทัดแรกเป็นบรรทัดว่าง)



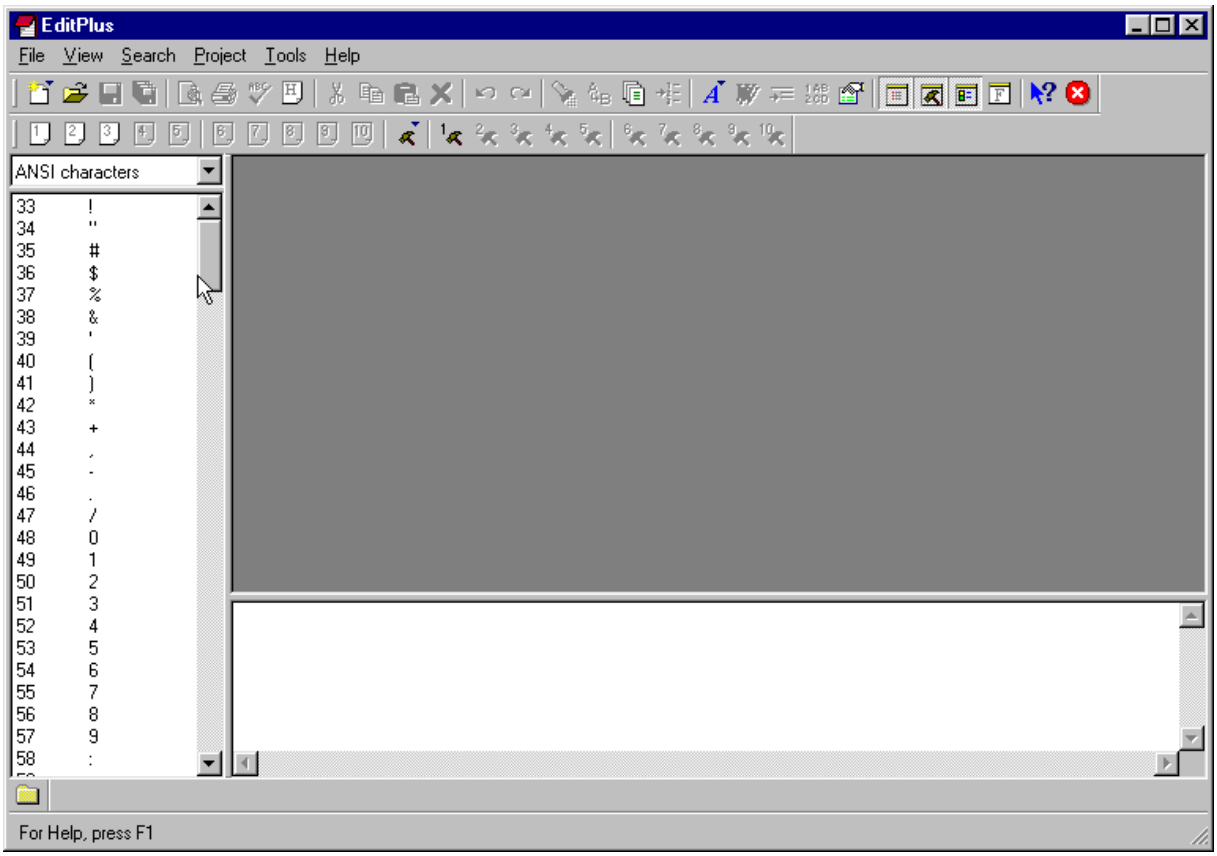
รูปแสดงการเลื่อนข้อความขึ้น 1 บรรทัด

ขั้นที่ 5 : ทำการ Save ไฟล์นี้ทับไฟล์เดิม, เท่านี้ก็เป็นอันเสร็จแล้วครับ จากนี้ผู้อ่านสามารถนำไฟล์ HEX นี้ไปใช้ในการ ดาว์นโหลดลง CPU ได้แล้ว....@@@

การใช้งานโปรแกรม SXA51.exe บน Windows

การแปลโปรแกรมด้วยวิธีนี้ถือว่าเป็นวิธีที่สะดวกที่สุด เนื่องจากว่าผู้อ่านไม่ต้องมานั่งพิมพ์คำสั่งทุกๆ ครั้งที่มีการ Compile โปรแกรมใหม่แต่อย่างใด โดยใช้โปรแกรม EditPlus ในการทำงาน

โปรแกรม Editplus จัดเป็นโปรแกรม Text Editer โปรแกรมหนึ่งที่ถือว่าเป็นโปรแกรมที่มีความสามารถสูง ในการที่ผู้อ่านจะใช้ในพัฒนาโปรแกรมใดๆ ไม่ว่าจะเป็นภาษา C, C++, Java, VHDL, HTML, ASP และ อื่นๆ แต่ในที่นี้ผู้เขียนจะนำมาใช้กับการแปลภาษา Assembly โดยเริ่มแรกผู้อ่านต้องไปดาว์นโหลดโปรแกรมนี้มาก่อน (ในที่นี้ทาง ETT ไม่มีให้นะครับ) จากเวปไซด์ www.editplus.com หรือดาว์โหลดจาก www.thaiware.com ซึ่งเวอร์ชั่นที่ได้มาจะเป็นตัวทดลองใช้ จากนั้นติดตั้งโปรแกรมให้เรียบร้อย และ เมื่อเปิดโปรแกรมขึ้นมาจะได้หน้าต่าง ดังภาพด้านล่าง

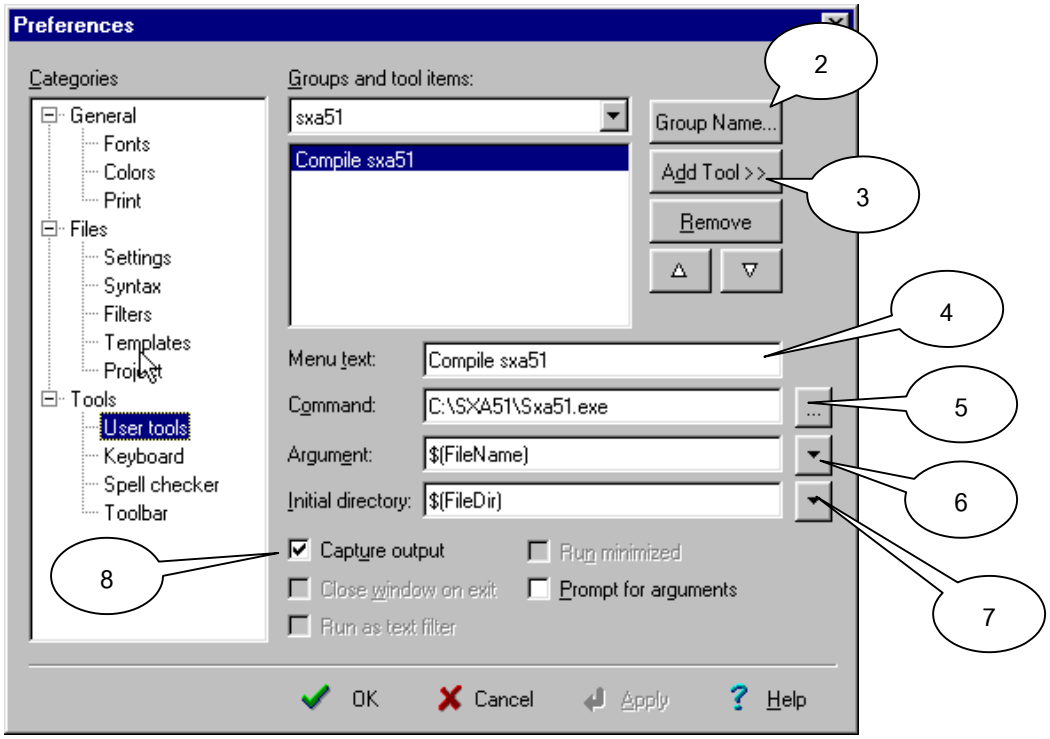


รูปแสดงโปรแกรม Editplus II

การปรับแต่งค่าบนโปรแกรม Editplus II

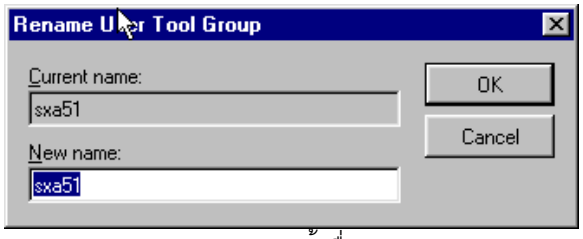
เมื่อติดตั้งโปรแกรม Editplus แล้ว จากนั้นผู้อ่านจะต้องทำการกำหนดค่าต่างๆ ให้ถูกต้องเสียก่อน ซึ่งการกำหนดค่านี้จะกำหนดเพียงครั้งเดียวเท่านั้น หลังจากนั้นโปรแกรมจะจดจำการเซ็ตอัพค่าต่างๆ ไว้

ขั้นที่ 1 : ไปที่เมนู Tools =>Configure User Tools…



แสดงหน้าต่าง Preferences ใน Editplus (ในภาพเป็นการกำหนดค่าสมบูรณ์แล้ว)

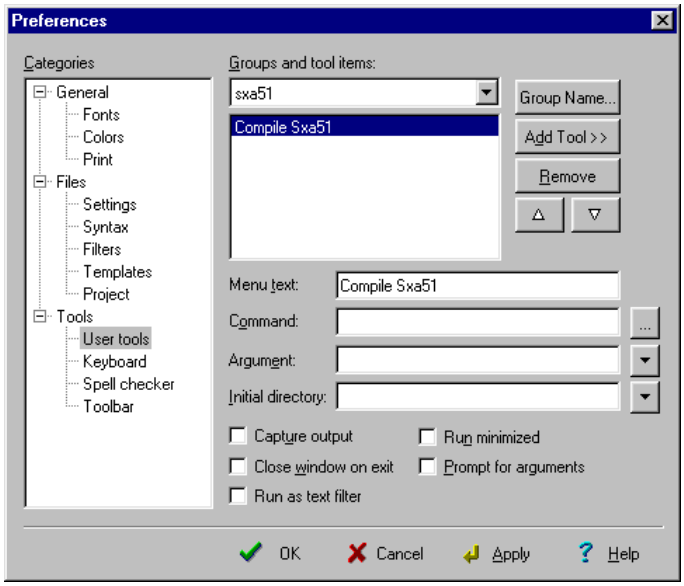
ขั้นที่ 2 : กดปุ่ม Group Name… จากนั้นจะมีหน้าต่าง “ Rename User Tool Group “ ขึ้นมาให้ใส่ชื่อ SXA51 ลงในช่อง New name: จากนั้นกดปุ่ม OK



รูปแสดงการตั้งชื่อกลุ่ม

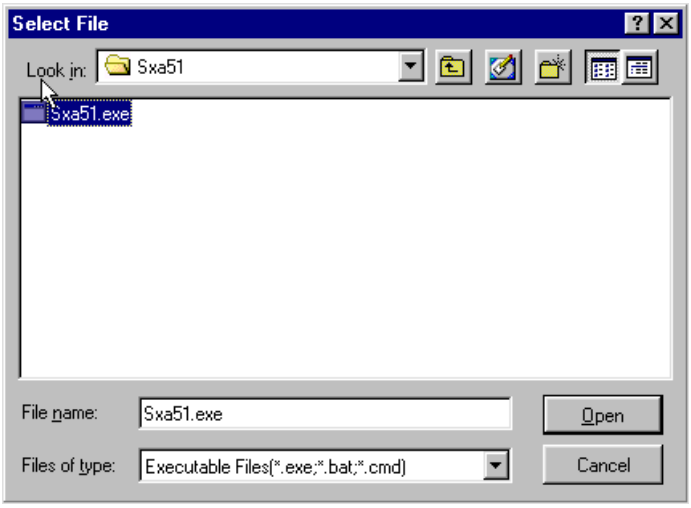
ขั้นที่ 3 : กดปุ่ม Add Tool >> จากนั้นเลือก “ Program “

ขั้นที่ 4 : ใส่ข้อความ “ Compile Sxa51 ” ในช่อง Menu text:



รูปแสดงการกำหนดค่าในช่อง Menu text

ขั้นที่ 5 : กดปุ่ม ... ตรงส่วนท้ายของช่อง Command: ซึ่งจะทำให้มีหน้าต่าง Select File ขึ้นมา ให้ผู้อ่านกำหนด Path ไปที่ C:\sxa51\sxa51.exe แล้วกดปุ่ม Open ตามลำดับ



รูปแสดงการเลือก Path สำหรับไฟล์ที่จะใช้ในการ Compile โปรแกรม .ASM

ขั้นที่ 6 : กดปุ่ม ตรงส่วนท้ายของช่อง Argument: จากนั้นเลือก “ File Name “ ซึ่งมีความหมายว่า การ Compile จะถึงชื่อไฟล์พร้อม นามสกุล

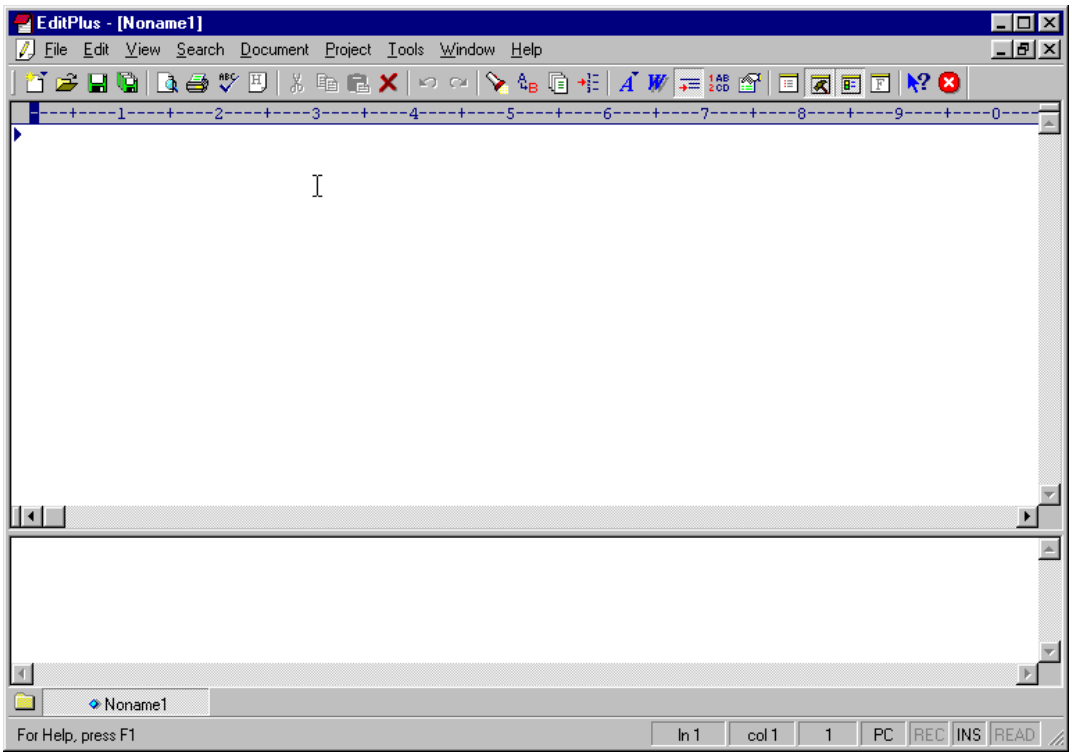
ขั้นที่ 7 : กดปุ่ม ตรงส่วนท้ายของช่อง Initial directory: จากนั้นเลือก “ File Directory “ ซึ่งมีความหมายว่า สามารถ Compile โปรแกรมจาก Folder ใดก็ได้ และ เมื่อ Compile แล้วจะได้ไฟล์นามสกุล . Hex อยู่ใน Folder นั้นด้วย

ขั้นที่ 8 : ติ๊กเครื่องหมายหน้าช่อง Capture output เพื่อให้การแสดงผลการ Compile ออกทางหน้าต่าง Output Window จากนั้นกดปุ่ม Apply และ ปุ่ม OK ตามลำดับ เท่านี้ก็เป็นอันเสร็จพิธี...!!!

ตัวอย่างการใช้งานโปรแกรม

เมื่อผู้อ่านติดตั้งโปรแกรม และ เซ็ตอัพ ค่าต่างๆ เป็นที่เรียบร้อยแล้ว, ในส่วนของการใช้งานนั้นง่ายมาก ซึ่งอธิบายเป็นขั้นตอนได้ดังนี้ คือ

ขั้นที่ 1 : เปิดโปรแกรม EditPlus II จากนั้นไปที่เมนู File => New => Normal Text จากนั้นจะปรากฏแผ่นกระดาษเปล่าๆ มาให้ 1 แผ่น ซึ่งให้ผู้อ่านเขียนโปรแกรมในส่วนนี้



รูปแสดงการเปิดหน้าต่างสำหรับเขียนโปรแกรม

ขั้นที่ 2 : จากรูปด้านบน ถ้าหน้าต่างโปรแกรมของผู้อ่านไม่มี Output Window ให้ไปเปิดที่เมนู View => ติ๊กเครื่องหมายถูกหน้า Output Window จากนั้นเขียนโปรแกรมของผู้อ่านได้ตามต้องการ เมื่อเสร็จแล้วให้ Save นามสกุล " . asm "

ขั้นที่ 3 : จากนั้นทำการ Compile โปรแกรมที่เขียนขึ้นโดยไปที่เมนู Tools => Compile Sxa51 หรือ จะใช้วิธีกดปุ่ม Ctrl + 1 ก็ได้ไม่ผิดกติกาครับ ซึ่งถ้าการ compile ถูกต้องผู้อ่านจะได้รับข้อความดังแสดงในรูปด้านล่าง

```
---------- Compile Sxa51 ----------

8051 Cross-Assembler (1.3)  Copyright (c) 1987, 1989
Binary Technology, Inc. Meriden, NH

No errors detected
Object file size: 25 bytes
Program entry address: 0000 (Hex)
Normal Termination
Output completed (0 sec consumed).
```

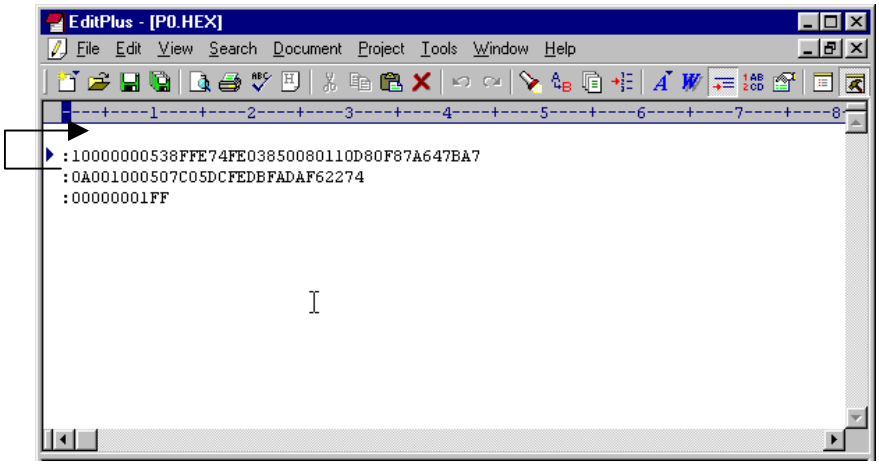
รูปแสดงการ Compile อย่างสมบูรณ์ในหน้าต่าง Output Window

ขั้นที่ 4 : ถ้าโปรแกรมของผู้อ่านผิดพลาด, หน้าต่าง Output Window นี้จะแสดงจำนวน Error และ จะแสดงคำสั่งที่ Error

ขั้นที่ 5 : เมื่อถึงตรงนี้ผู้อ่านจะได้ไฟล์เพิ่มขึ้นมา 1 ไฟล์ ซึ่งอยู่ในโฟลเดอร์เดียวกับไฟล์ . ASM ที่ผู้อ่านได้เขียนขึ้น ซึ่งจะมีนามสกุล . HEX

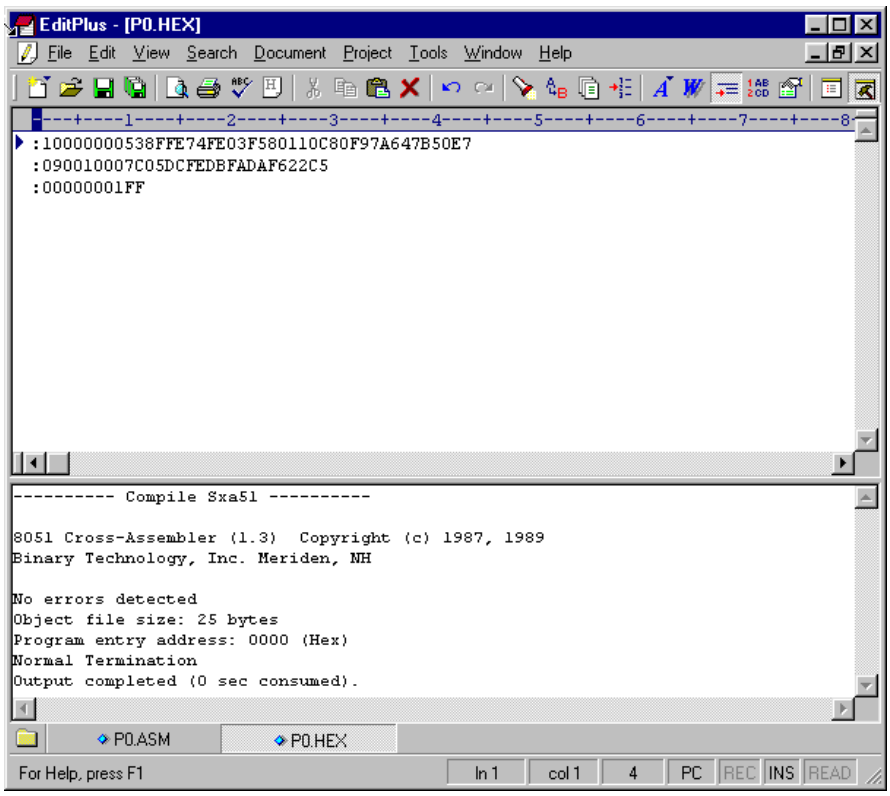
ขั้นที่ 6 : เปิดไฟล์นามสกุล . HEX นั้นขึ้นมาโดยใช้ Icon หรือ ไปที่เมนู File => Open

ขั้นที่ 7 : ผู้อ่านจะเห็นบรรทัดบนสุดของไฟล์นามสกุล . HEX นั้นว่าง 1 บรรทัด จากนั้นให้ผู้อ่านนำเคอร์เซอร์วางไว้หน้าสุดของข้อความ (หน้า :) จากนั้นเลื่อนทุกบรรทัดขึ้นมาด้วยปุ่ม Backspace



แสดงการเลือกทุกคำสั่ง 1 บรรทัด

ขั้นที่ 8 : จากนั้นกดปุ่ม Save



แสดงการเลือกทุกคำสั่ง 1 บรรทัด

ขั้นที่ 9 : เท่านี้เป็นอันเสร็จขั้นตอนทั้งหมดแล้วครับ จากนั้นผู้อ่านก็นำ Hex file ไปใช้ในการดาว์นโหลดโปรแกรมได้แล้ว

Note : ในกรณีที่ผู้อ่านต้องการเปิดไฟล์นามสกุล . asm ขึ้นมา, ผู้อ่านจะไม่เห็นไฟล์นามสกุล .asm ดังนั้น ให้ผู้อ่านเลือกช่อง Files of type : เป็น All files (*.*)

ไวยากรณ์ใน SXA51.EXE

การเขียนโปรแกรมภาษา Assembly ตามข้อกำหนดของโปรแกรม “SXA51”

การใช้งานโปรแกรม Assembler SXA51.EXE (MCS-51 Assemblers)

สำหรับโปรแกรม SXA51.EXE เป็นโปรแกรม Assembler ทำหน้าที่สำหรับแปลโปรแกรม ภาษา Assembly ที่อยู่ในรูปของ Text Files ให้เป็นไฟล์ แบบ "Intel Hex Format" (Files.HEX) เพื่อนำไปทำการ Download หรือโปรแกรมให้กับ CPU ตระกูล MCS51 อีกครั้งหนึ่ง ซึ่งโปรแกรม SXA51 นี้ จะมีรูปแบบ และข้อกำหนดเฉพาะบางประการ ในส่วนที่เป็นความสามารถเฉพาะของโปรแกรม Assembler เอง เช่น วิธีการกำหนดสัญลักษณ์ LABEL วิธีการกำหนด ค่าคงที่ของเลขฐานสิบหก เลขฐานสิบ ฯลฯ ดังนั้น ในการเขียนโปรแกรม Text File ภาษา Assembly เพื่อนำมาให้โปรแกรม SXA51 นี้ ทำการแปลให้นั้น จะต้องกำหนดรูปแบบต่างๆ ให้ตรงกับข้อกำหนดของโปรแกรม SXA51 ต้องการด้วย

ในการเขียนโปรแกรมที่เป็นภาษา Assembly นั้น ตามหลักการเบื้องต้นจริงๆแล้ว เมื่อเราต้องการ เรียกใช้งานหน่วยความจำหรือต้องการกระโดดไปทำงานในตำแหน่งใดหรือการเรียกใช้งานโปรแกรมย่อย ต่างๆนั้น จะต้องระบุตำแหน่ง Address ที่ต้องการให้โปรแกรมทราบ ซึ่งค่าของตำแหน่ง Address ต่างๆนั้น ก็จะเป็นค่าของ ตัวเลขฐานสิบหก ซึ่งลักษณะเช่นนี้จะเป็นอุปสรรคต่อการเขียนโปรแกรมและยากแก่การ ตรวจสอบโปรแกรมเป็นอย่างมาก เนื่องจากค่าของตัวเลขต่างๆที่รวมอยู่ในโปรแกรมนั้นอาจมีจำนวนมาก และยากต่อการจดจำและแก้ไข ดังนั้นโปรแกรม Assembler ต่างๆ จึงมีการพัฒนารูปแบบให้สามารถ แทนค่าตำแหน่ง Address ต่างๆ ด้วยสัญลักษณ์ที่เป็นข้อความได้ ซึ่งจะเป็นการอำนวยความสะดวกให้กับ ผู้เขียนโปรแกรมเป็นอย่างมากอีกทั้งยังสามารถกำหนดตำแหน่ง Address ต่างๆให้เป็นชื่อหรือข้อความอื่น ที่สามารถสื่อความหมายได้ดีขึ้น เช่น เมื่อต้องการเรียกใช้โปรแกรมย่อย ที่อยู่ในตำแหน่ง Address 1000H ซึ่งเป็นโปรแกรมย่อยทำหน้าที่ Scankeyboard แทนที่จะใช้คำสั่ง LCALL 1000H ก็สามารถใช้ LCALL SCANKEY แทนได้ เมื่อนำโปรแกรมไปแปลก็จะได้ค่าที่ถูกต้องเช่นเดิม เป็นต้น

สำหรับรูปแบบในการเขียนโปรแกรมภาษา Asembly ของ CPU นั้นจะมีลักษณะคล้ายกัน แต่ก็ยัง มีความแตกต่างกันอยู่บ้างขึ้นอยู่กับข้อกำหนดและความสามารถของโปรแกรมที่จะใช้ทำหน้าที่ในการสั่ง ให้แปลโปรแกรมคำสั่งภาษา Assembly ให้เป็นรหัสคำสั่ง ซึ่งนิยมเรียกว่าโปรแกรม Assembler ซึ่งมีอยู่มาก มายหลายโปรแกรมบางตัวอาจทำหน้าที่เป็น Compiler โดยมีทั้งส่วนที่ใช้เขียนโปรแกรม (Text Editor) และ Assembler ในตัวเดียวกันเลย สำหรับในการทดลองนี้ขอแนะนำให้ใช้โปรแกรม Assembler ชื่อ SXA51 ซึ่งใช้ทำหน้าที่แปลโปรแกรมได้อย่างเดียว การเขียนโปรแกรมต้องอาศัยโปรแกรม Text Editor ตัวอื่นช่วย เช่น EDIT.COM ของ DOS หรือ SK.COM โดยใช้โปรแกรมที่เป็น Text Editor ในการเขียนโปรแกรมแล้ว Save เป็นไฟล์ไว้ จากนั้นจึงนำไฟล์ภาษา Assembly ที่ได้แล้วมาให้โปรแกรม SXA51 แปลให้ใน ภายหลัง โดยรูปแบบในการเขียนโปรแกรมภาษา Assembly ภายใต้ข้อกำหนดของโปรแกรม SXA51 นั้นจะแบ่ง Field ของโปรแกรมออกเป็น 4 ส่วน หลักๆ ดังต่อไปนี้

Lable	Operation	Operand	Comment
(สัญลักษณ์)	**(คำสั่ง)**	**(ตัวกระทำ)**	**(คำอธิบาย)**

ซึ่งในการเขียนโปรแกรมแล้วบางบรรทัดอาจมีครบทั้ง 4 Field บางบรรทัดอาจมีเพียง Field เดียว ก็ได้ไม่แน่นอน ขึ้นอยู่กับลักษณะและรูปแบบของคำสั่งที่นำมาใช้เขียนในโปรแกรม

LABEL(ชื่อสัญลักษณ์) ในส่วนนี้จะมีหรือไม่มีก็ได้ แต่ถ้ามีจะต้องอยู่ใน Column แรกทางซ้าย และตัวอักขระตัวแรกของ LABEL นี้ควรต้องชิดขอบซ้ายของหน้ากระดาษเสมอ สำหรับชื่อของ LABEL นั้นสามารถตั้งได้ตามความต้องการ โดยใช้ตัวอักษรพิมพ์เล็กหรือใหญ่ก็ได้ ซึ่งจะมีความหมายเหมือนกัน ซึ่งอักขระที่จะสามารถนำมาใช้ตั้งเป็น ชื่อ LABEL นั้นตัวอักษรแรกต้องขึ้นต้นด้วย A-Z หรือ "_" หรือ "." หรือ "?" เท่านั้น ซึ่งตัวอักษรที่สามารถนำมาใช้ตั้งเป็นชื่อ LABEL ได้นั้นได้แก่ A-Z,0-9 และเครื่องหมายพิเศษอื่นๆ โดยความยาวของชื่อ LABEL นั้นสามารถตั้งได้สูงสุดไม่เกิน 15 ตัวอักษร และต้องไม่ตั้งชื่อ LABEL ให้ตรงกับชื่อคำสั่งหรือริจิสเตอร์ของ CPU ด้วย และชื่อของ LABEL ต้องเรียงติดกันห้ามเว้นวรรค โดยชื่อ LABEL นั้นต้องปิดท้ายด้วยเครื่องหมายโคลอน (:) ที่ท้ายชื่อด้วยเสมอ แต่โปรแกรมจะไม่ถือเอา เครื่องหมายโคลอน(:) นี้เป็นส่วนของชื่อด้วย แต่ต้องใส่กำกับไว้ เพื่อบ่งบอกให้โปรแกรม Assembler ทราบว่า ชื่อนั้นคือ LABEL ตัวอย่างเช่น

```
HERE:   NOP
        DJNZ    R2,HERE
หรือ
here:   NOP
        DJNZ    R2,HERE
```

ซึ่งในการเขียนโปรแกรมภาษา Assembly ส่วนมากนั้นจะนิยมกำหนด ชื่อ LABEL ไว้เฉพาะใน ตำแหน่งของโปรแกรมที่ต้องการจะกระโดดไปทำงานหรืออ้างอิงถึงโดยคำสั่งอื่นๆหรืออาจเป็นตำแหน่ง เริ่มต้นของโปรแกรมย่อยต่างๆ ที่ต้องเรียกใช้ เพราะจะทำให้ง่ายต่อการจดจำและสื่อความหมายได้มากกว่า วิธีการที่ใช้การกำหนดค่าตำแหน่ง Address เป็นแบบตัวเลขโดยตรง และประการสำคัญเมื่อมีการแทรก หรือตัดคำสั่งต่างๆในส่วนหนึ่งส่วนใดของโปรแกรมแล้ว สามารถที่จะนำโปรแกรมนั้นไปทำการแปลใหม่ ซึ่งโปรแกรม Assembler ก็จะสามารถทำการคำนวณหาค่าของตำแหน่ง Address ใหม่ที่ถูกต้องให้ได้ทันที

OPERATION(คำสั่ง)ในส่วนนี้จะต้องมีอยู่ในทุกบรรทัดที่เป็นโปรแกรมซึ่งใน Field นี้จะใช้เขียน คำสั่งภาษา Assembly ของ CPU (Mnemonic Code) เช่น MOV INC ADD SUBB เป็นต้น โดยคำสั่งที่ เขียนขึ้นนั้นในส่วนที่เป็น Mnemonic เดียวกัน จะต้องเขียนให้ติดกันด้วยห้ามเว้น ถ้าหากมีการเว้นช่องว่าง โปรแกรมจะถือว่าส่วนที่อยู่ด้านหลังช่องว่างของคำสั่ง (Menmonic) นั้นเป็นค่าของตัวกระทำ (Operand) ซึ่งจะทำให้เกิดความผิดพลาดขึ้นในการแปลโปรแกรม

OPERAND(ตัวกระทำ) ใน Field นี้อาจมีหรือไม่มีก็ได้ขึ้นอยู่กับรูปแบบของคำสั่ง (Mnemonic) ที่นำมาใช้เขียนโปรแกรมใน Field ของ Operation ซึ่งในส่วน Field ของ Operand นี้จะเป็นตัวบ่งบอก ให้ทราบถึงหน้าที่การกระทำของคำสั่ง(Menmonic)ว่าจะให้ข้อมูลหรือตัวกระทำ(Operand) ใดมากระทำกัน ในคำสั่งบ้าง ซึ่งบางคำสั่ง เช่น RET(Return Fron Subroutine) ก็จะไม่จำเป็นต้องมีค่าของ Operand เข้ามา เกี่ยวข้องในคำสั่ง แต่บางคำสั่งอาจมีตัวกระทำ Operand ในคำสั่ง 1 ตัว หรือมากกว่า ก็ได้ ถ้ามี Operand มากกว่า 1 ตัว จะแบ่งแยกแต่ละ Operand ด้วยเครื่องหมายคอมม่า (,) ตัวอย่างเช่น

CLR	C	CLR เป็นคำสั่ง(Mnemonic) ส่วน C เป็นตัวกระทำ (Operand)
MOV	A,R2	คำสั่งนี้จะมีตัวกระทำ 2 ตัวคือ A และ R2
RET		เป็นคำสั่งชนิดที่ไม่มีตัวกระทำ

COMMENT(**คำอธิบาย**) ในส่วนนี้จะมีหรือไม่มีก็ได้ไม่มีผลต่อการทำงานของโปรแกรมที่เขียน ขึ้นแต่อย่างใดและโปรแกรม Assembler จะไม่นำส่วนคำอธิบายหรือ Comment นี้ มาเกี่ยวข้องกับการแปล โปรแกรมด้วย ซึ่งส่วนคำอธิบายนี้จะนิยมเขียนกำกับไว้ในโปรแกรมเพื่อใช้เป็นคำอธิบายความหมายกันลืม เพื่อประโยชน์ในการกลับมาแก้ไขโปรแกรมเดิมที่เขียนไว้นานแล้วจะได้สามารถเข้าใจได้โดยง่าย โดยคำอธิบายนี้จะสามารถเขียนได้ยาวเท่าได้ก็ได้ไม่จำกัด เพียงแต่มีข้อแม้ว่า ถ้าเขียน Comment นี้อยู่ใน บรรทัดเดียวกับคำสั่ง จะต้องเขียนใน Column สุดท้ายหลังจากจบรูปแบบของคำสั่งที่เขียนขึ้นแล้ว

โดยก่อนเริ่มต้นเขียนส่วน Comment นี้ต้องใส่เครื่องหมายเซมิโคลอน (;) นำหน้าไว้ด้วยเสมอ และห้ามไม่ให้ ใส่ Comment ไว้หน้าคำสั่ง เพราะโปรแกรม Assembler จะถือว่าข้อความที่อยู่ตามหลัง เครื่องหมาย เซมิโคลอน (;) เป็น Comment ทั้งหมดและจะไม่นำข้อความในบรรทัดนั้นมาแปลอีก

สำหรับข้อกำหนดอื่น ๆที่ควรทราบมีดังนี้

1. การกำหนดค่าเลขฐาน 16 ให้ใช้อักขระ H ปิดท้ายค่านั้น เช่น 7FH และถ้าค่าเลขฐาน 16 นำหน้าด้วย ตัวอักษร A-F ต้องใส่เลขศูนย์นำหน้าด้วย เช่น 0A8H,0FFH เป็นต้น
2. การกำหนดค่าเลขฐาน 2 ให้ใช้อักขระ B ปิดท้าย เช่น 00000001B
3. การกำหนดค่าของรหัส ASCII ของตัวอักขระใดให้อยู่ในเครื่องหมาย (") เช่น "A"
4. การกำหนดค่าคงที่(Immediate Data) ให้ใช้เครื่องหมาย (#) นำหน้าเลขฐานนั้นๆ เช่น #12H
5. การไม่ระบุเครื่องหมายให้กับตัวเลขจะถือว่าเป็นค่าของเลขฐานสิบ เช่น 5 หรือ 200

ตัวอย่างเช่น

ต้องการกำหนดค่า 1234 ฐานสิบหก ให้ใช้	1234H	
ต้องการกำหนดค่า 1 ในฐานสอง ให้ใช้	00000001B	
ต้องการกำหนดรหัส ASCII ของตัว A ให้ใช้	"A"	(ค่ารหัส ASCII ของ A = 41H)

ตัวอย่างโปรแกรม ภาษา Assembly เช่น

```
Label       Operator    Operand          Comment

PORT_A      EQU         0E000H           ; Port A of 8255
PORT_CTL    EQU         0E003H           ; Port Control of 8255
            ORG         2200H
MAIN:       MOV         DPTR,#PORT_CTL
            MOV         A,#80H           ; All  Port = Output
            MOVX        @DPTR,A
LOOP1:      MOV         DPTR,#TAB_LED ; Table Data
            MOV         R2,#8
LOOP2:      CLR         A
            MOVC        A,@A+DPTR
            PUSH        DPH
            PUSH        DPL
            MOV         DPTR,#PORT_A
            MOVX        @DPTR,A
            POP         DPL
            POP         DPH
            LCALL       DELAY
            INC         DPTR
            DJNZ        R2,LOOP2
            SJMP        LOOP1
```

จากตัวอย่างโปรแกรมที่แสดงข้างต้น จะเห็นได้ว่าลักษณะการเขียนโปรแกรมภาษา Assembly นั้น ไม่จำเป็นต้องเขียนให้ครบทุก Field ส่วนจะมีกี่ Fieldในแต่ละบรรทัดคำสั่งนั้น ขึ้นอยู่กับข้อบังคับของคำสั่ง และความต้องการของผู้เขียนเป็นหลัก จากตัวอย่าง MAIN LOOP1 และ LOOP2 เป็น Label Address ใช้กำกับไว้ในโปรแกรมเพื่อการอ้างถึงตำแหน่งของคำสั่งในบรรทัดนั้นๆ ซึ่งจะเห็นว่า MAIN ไม่มีการอ้าง ตำแหน่งในโปรแกรมเพื่อใช้งานเลย แต่กำหนดไว้ให้ทราบว่าเป็นตำแหน่ง Address ของโปรแกรมหลัก สำหรับในส่วนของตัวกระทำ Operand นั้นบางคำสั่งจะไม่มี เนื่องจากเป็นข้อกำหนดของคำสั่งนั้นๆเอง และในส่วนของ Field Comment นั้นจะมีหรือไม่มีก็ได้แต่ในโปรแกรมจะใส่ไว้บางจุดในโปรแกรม เพื่อใช้ เป็นคำอธิบายความหมายบางอย่างกันลืม เช่น ในบรรทัดที่2 ของโปรแกรม(MOV A,#80H) จะใส่ Comment ไว้เพื่ออธิบายว่านำค่า #80H ที่กำหนดให้รีจิสเตอร์ ACC นั้นเป็นค่าสำหรับกำหนดให้ 8255 ทำหน้าที่เป็น Output Port ทั้งหมด เป็นต้น สำหรับคำสั่ง EQU และ ORG นั้นจะเป็นคำสั่งเทียมใช้กำหนดค่าให้กับ สัญลักษณ์ Label ต่างๆ เพื่อความสะดวกในการเขียนโปรแกรม โดยไม่จำเป็นต้องมาจดจำค่าเป็นตัวเลข ให้ยุ่งยาก ซึ่งจะกล่าวอธิบายถึงในลำดับต่อไป

คำสั่งเทียมในโปรแกรม Assembler SXA51

ในการเขียนโปรแกรมภาษา Assembly นั้นจะมีความยุ่งยากกว่าโปรแกรมภาษาสูงอื่นๆพอสมควร โดยเฉพาะอย่างยิ่งเมื่อต้องมีการใช้พื้นที่ของหน่วยความจำสำหรับเก็บข้อมูลชั่วคราวไว้ก่อน เมื่อต้องการ เรียกข้อมูลออกมาใช้จะเป็นการยุ่งยากอย่างมาก ว่าตำแหน่ง Address ใดถูกจัดสรรหน่วยความจำไว้สำหรับ เก็บค่าของอะไรบ้าง และยังอาจเกิดความสับสนเรียกใช้ผิดตำแหน่งได้ ทำให้เกิดความผิดพลาดขึ้นได้ง่าย ด้วยเหตุนี้โปรแกรมที่ทำหน้าที่เป็นตัว Assembler ต่างๆ จึงมีการคิดค้นสร้างคำสั่งอีกกลุ่มหนึ่งขึ้นมาใหม่ เพื่ออำนวยความสะดวกให้กับผู้เขียนโปรแกรม โดยกลุ่มคำสั่งดังกล่าวนี้นิยมเรียกกันทั่วไปว่า "คำสั่งเทียม" ซึ่งคำสั่งเทียม นี้จะไม่ใช่คำสั่งสำหรับสั่งงาน CPU แต่จะถูกนำมาใช้เพื่อบ่งบอกให้โปรแกรม Assembler ได้ทราบถึงสิ่งที่จะกำหนดขึ้น เพื่อนำไปใช้ในการแปลโปรแกรมเท่านั้น

ORG (Origin) เป็นคำสั่งเทียม ใช้ระบุค่าตำแหน่ง Address เพื่อบ่งบอกให้โปรแกรม Assembler ได้รับรู้ว่าสัญลักษณ์หรือคำสั่ง ที่เขียนต่อจากคำสั่ง ORG นั้นจะมีค่าตำแหน่ง Address เริ่มต้นตามค่า ที่กำหนดไว้จากคำสั่ง ORG นี้และค่า Address นั้นจะต่อเนื่องกันไปเรื่อยๆ เช่น

```
        ORG     0000H
MAIN:   MOV     SP,#128-32      ; Define Stack Internal RAM
        MOV     R2,#0FFH
        DJNZ    R2,$            ; Power-On Delay
```

EQU (Equate) เป็นคำสั่งเทียมใช้สำหรับกำหนดค่าให้กับ ตัวแปรสัญลักษณ์(Symbol) ที่กำหนด ไว้หน้าคำสั่ง EQU โดยโปรแกรม Assembler จะถือเอาค่า ของตัวแปรสัญลักษณ์ นั้นๆ ให้มีค่าตามที่ กำหนดไว้หลังคำสั่ง EQU ตลอดไปในการแปลคำสั่ง ดังนั้นทุกๆครั้งที่มีการอ้างถึง ตัวแปรสัญลักษณ์ ตัวนั้นๆเมื่อใด ก็จะมีความหมายเหมือนกับการอ้างถึงค่าที่กำหนดให้ทุกประการ ตัวอย่างเช่น

```
PORT_A EQU  0E000H
PORT_B EQU  PORT_A+1
PORT_C EQU  PORT_A+2
```

จากตัวอย่างการใช้คำสั่ง EQU ที่ผ่านมานั้น ในทุกๆตำแหน่งของโปรแกรมที่มีการอ้างถึง PORTA ก็จะมีความหมายเหมือนกับอ้างถึงค่า 0E000H เสมอ ส่วนค่าของ PORT_B และ PORT_C ก็จะมีค่าเป็น 0E001H และ 0E002H ตามลำดับ ถ้ามีการเปลี่ยนแปลงค่าของ PORT_A เป็นค่าอื่นก็จะส่งผลให้ PORT_B และ PORT_C ถูกเปลี่ยนแปลงค่าตามไปด้วย เราสามารถใช้คำสั่ง EQU กำหนดค่าให้กับ Symbol Name หนึ่งๆได้เพียงครั้งเดียว เท่านั้นในโปรแกรม ซึ่งลักษณะของคำสั่ง EQU จะเหมือนกับการกำหนดค่า Constant ในภาษาสูงนั่นเอง

DB(Define Byte) เป็นคำสั่งเทียมใช้สำหรับกำหนดค่าข้อมูลคงที่ขนาด 8 บิต(Byte) ไว้ในโปรแกรม โดยคำสั่งนี้จะถูกนำมาใช้ประโยชน์ในการกำหนด ตาราง TABLE ต่างๆ ที่จะนำมาใช้ใน โปรแกรมโดย จำนวนข้อมูลแต่ละค่าจะถูกแบ่งแยกออกจากกันด้วยเครื่องหมายคอมม่า (,) ตัวอย่างเช่น

```
TAB_LED:    DB    00000001B
            DB    02H,04H,08H
TAB_MSG:    DB    "ABC",00H
```

จากตัวอย่างที่ผ่านมาเมื่อสั่งให้โปรแกรม Assembler แปลโปรแกรมแล้ว ที่ตำแหน่ง Address ของ TAB_LED จะมีค่าคงที่อยู่ 4 ไบท์ คือ 01H,02H,04H และ 08H เรียงต่อเนื่องกันไป ส่วน TAB_MSG จะได้ค่าเป็นรหัส ASCII ของตัวอักษร ABC เรียงกันไป คือ 41H,42H,43H และปิดท้ายด้วย 00H

DW(Define Word) เป็นคำสั่งเทียมสำหรับสั่งกำหนด ค่าคงที่ ขนาด 16 บิต(Word) โดยต้องกำหนด ค่าของ Byteต่ำก่อนแล้วตามด้วย Byte สูง และแบ่งแยกข้อมูลแต่ละ Word ด้วยเครื่องหมาย คอมม่า(,)

DS (Define Storage) เป็นคำสั่งเทียมใช้สำหรับสงวน หรือ จองพื้นที่ของหน่วยความจำไว้ เท่ากับ จำนวนที่ระบุไว้หลังคำสั่ง DS เช่น

```
            ORG   0000H
DSP_BUFF:   DS    4
```

จะเป็นการบอกให้โปรแกรม Assembler ได้ทราบว่า Symbol ชื่อ DSP_BUFF นั้น ถูกกำหนดให้ มีขนาดของหน่วยความจำ 4ไบท์ โดยมีตำแหน่ง 0000H-0003H ซึ่งถ้ามีการอ้างถึง Symbol นี้ในโปรแกรม ก็จะมีความหมายเหมือนกับการอ้างถึงค่าตำแหน่ง Address 0000H-0003H ด้วย โดยถ้าใช้การอ้างแบบ 8 บิต เมื่อต้องการใช้ตำแหน่งที่ 2 ของ Symbol อาจใช้รูปแบบเป็น DSP_BUFF+1 แทนก็ได้ ซึ่งจะเป็นประโยชน์ มากในการเขียนโปรแกรมเพราะสามารถสื่อความหมายถึงหน้าที่ของหน่วยความจำได้ดีกว่าการอ้างเป็นค่า ตัวเลข และยังง่ายต่อการเปลี่ยนแปลงแก้ไขตำแหน่ง Address ในหน่วยความจำด้วย

END เป็นคำสั่งเทียมสำหรับบอกให้ Assembler ทราบว่าส่วนของโปรแกรมทั้งหมดสิ้นสุดลงก่อน หน้าตำแหน่งของคำสั่ง END นี้ ซึ่งต้องมีคำสั่งนี้ไว้ท้ายของโปรแกรมเสมอ และหากเขียนโปรแกรมต่อหลัง จากคำสั่ง END นี้แล้ว ASsembler จะไม่แปลส่วนที่ต่อจาก END นี้ให้เลย

การสั่งแปลโปรแกรมด้วย SXA51

สำหรับวิธีการเรียกใช้งาน โปรแกรม Assembler SXA51 เพื่อให้ทำหน้าที่แปลโปรแกรมภาษา Assembly ที่เขียนไว้แล้วให้เป็น File แบบ "Intel HEX Format" (HEX) มีรูปแบบดังนี้

```
SXA51  <Option> FILENAME.ASM
```

Option เป็นส่วนของข้อกำหนดในการสั่งแปลโปรแกรม โดยใช้เครื่องหมาย "-" นำหน้าตัวอักษร ที่แทนค่าของ Option โดยเครื่องหมาย "-" และตัวอักษรแสดง Option ต้องพิมพ์ติดกัน โดยสามารถใช้ค่า Option มากกว่า 1 ค่าได้ในเวลาเดียวกัน สำหรับค่าตัวอักษร ในการกำหนด Option มีดังนี้

-L หมายถึง ให้สร้าง Listing File ให้ด้วย โดยจะได้ชื่อ Listing File เหมือน Source File แต่จะมีสกุล เป็น FILE.LST แทน ซึ่ง Listing File จะมีประโยชน์มากในการตรวจสอบ ตำแหน่งที่เกิดการ Error จากการ Compiler และยังใช้อ้างอิงตำแหน่ง Address และ Code ของคำสั่ง ต่างๆได้อีกด้วย

-N หมายถึง ให้ทำการ Compiler โดยไม่สร้างไฟล์ใดๆ ใช้เพื่อทดสอบว่าโปรแกรมที่เขียนขึ้นมีส่วน ที่เกิดการ Error หรือไม่ ก่อนจะสั่ง Compiler จริง อีกครั้งหนึ่ง ซึ่งการ Compiler แบบนี้จะทำได้อย่าง รวดเร็วมาก โดยเฉพาะในกรณีที่ Source Code มีขนาดใหญ่มากๆ หากสั่ง Compiler แบบปรกติ จะใช้เวลานาน ทำให้เสียเวลา

-C หมายถึง ให้ทำการ Compiler พร้อมทั้งสร้าง Listing File เช่นเดียวกับ -L แต่แบบนี้จะได้ส่วน สัญลักษณ์ Symbol ต่อท้าย Listing ไฟล์เพิ่มให้ด้วย ซึ่ง Symbol นี้จะมีประโยชน์อย่างมาก ในการค้นหา ตำแหน่งต่างๆในโปรแกรมโดยอ้างอิงตาม Label

-D หมายถึง ให้แสดงขั้นตอนการแปลในขณะกำลังแปลให้ด้วย นิยมให้ในกรณีที่ Source Code มีขนาดใหญ่มากๆ และเครื่องคอมพิวเตอร์ที่ใช้ทำงานได้ช้า ซึ่งบางครั้งอาจต้องรอเป็นเวลานาน จนอาจ เข้าใจผิดว่าเครื่องคอมพิวเตอร์ Hang ไปแล้ว เมื่อใช้ Option นี้จะทำให้ทราบว่า โปรแกรมกำลัง ทำงาน อะไรอยู่ในขณะนั้นๆบ้าง

FILENAME.ASM หมายถึง ชื่อไฟล์ Source Code ภาษา Assembly ที่ต้องการให้โปรแกรม แปลให้ซึ่งถ้าหากเรียกใช้โปรแกรมโดยไม่มี Option ใดๆ จะได้ไฟล์ที่แปลเสร็จแล้วชื่อเดียวกับไฟล์ต้นฉบับ ทุกอย่างแต่จะมีสกุลเปลี่ยนเป็น HEX แทน โดยสถานะของการแปลโปรแกรมจะแสดงให้เห็นที่หน้าจอ PC ถ้าทุกอย่างถูกต้องจะแสดงสถานะเป็น "0 Error" หมายถึงไม่มีความผิดพลาด แต่ถ้ามีความผิดพลาดเกิดขึ้น จากการแปลแล้วจะแสดง ตำแหน่งที่ผิดพลาดพร้อมกับสาเหตุที่ตรวจพบรวมทั้งจำนวนที่ผิดให้ทราบทาง หน้าจอด้วย แต่ถ้าหากมี การ Error หลายๆที่จนไม่สามารถดูได้ทางหน้าจอแล้วก็สามารถสั่งแปลโปรแกรม โดยให้สร้างไฟล์ Listing ให้ด้วยก็ได้ ซึ่งจะได้ไฟล์เพิ่มขึ้นมาอีก 1 ไฟล์ โดยกำหนดใน Option คำสั่ง โดย ชื่อของไฟล์จะเหมือนต้นฉบับแต่มีสกุลเป็น LST แทน จากนั้น จึงใช้โปรแกรม Text Editor เปิด File ที่มีสกุล LST ที่เรากำหนดไว้แล้วตรวจสอบหาตำแหน่งและสาเหตุ ของความผิดพลาดที่แสดงไว้ใน Listing File เพื่อจะได้ทำการแก้ไขต้นฉบับให้ถูกต้องแล้วนำไปแปลใหม่ เช่น

```
C:\> SXA51  TEST.ASM
```

เป็นการสั่งให้โปรแกรม SXA51 แปลไฟล์ภาษา Assembly ชื่อ TEST.ASM ซึ่งถ้าไม่เกิดความ ผิดพลาดใดๆขึ้นจะได้ไฟล์ชื่อ TEST.HEX เพิ่มขึ้นมาอีก 1 ไฟล์ โดยไฟล์ที่ได้เพิ่มขึ้นมานี้จะเป็นไฟล์ที่ มีรูปแบบตรงตามข้อกำหนดของ "Intel HEX " สามารถนำไปทำการ Download ให้กับ CPU ได้ทันที

```
C:\> SXA51 -L  TEST.ASM
```

เป็นการสั่งให้โปรแกรม SXA51 ทำการแปลโปรแกรมภาษา Assembly ชื่อ TEST.ASM ให้ พร้อมทั้งสร้างไฟล์ Listing ชื่อ TEST.LST ให้ด้วย ซึ่งไฟล์ Listing นี้จะแสดงค่าตำแหน่ง Address และ รหัสคำสั่งเป็นตัวเลข ควบคู่ไปกับโปรแกรมที่เขียนขึ้นให้ทราบ หรือถ้าการแปลเกิด Error ก็จะแสดงสาเหตุ และตำแหน่งคำสั่งที่ Error ให้ทราบด้วย

สำหรับในการเขียนโปรแกรมเพื่อทำการทดลองกับ CPU MCS51 เบอร์ P89C51RD2 นั้น ตำแหน่งเริ่มต้น (ORG) ของโปรแกรมที่จะเขียนนั้นต้องเริ่มต้นที่ 0000H เท่านั้น ไม่สามารถเขียนให้อยู่นอกเหนือจากที่ กำหนดไว้ ได้เพราะหลังการรีเซ็ตทุกครั้ง CPU จะกระโดดมาทำงานตำแหน่งแรกที่ 0000H เสมอ โดยสามารถเขียน โปรแกรมได้ยาวทั้งหมด 8KByte ส่วนพื้นที่ของหน่วยความจำ RAM สำหรับใช้งานนั้นจะมีขนาด 256 ไบท์ โดยมีตำแหน่งใช้งานอยู่ระหว่าง 00H-FFH โดยในการเขียนโปรแกรมนั้น ต้องทำการจัดสรรหน่วยความจำ ส่วนนี้เองว่าจะใช้ส่วนใดสำหรับเก็บข้อมูล และให้ส่วนใดเป็นพื้นที่ของ Stack สำหรับการทำงานของ CPU และไม่สามารถใช้คำสั่ง MOVX ได้ เนื่องจากการต่อวงจรให้กับ CPU จะเป็นแบบ Single Chips ไม่มีหน่วย ความจำภายนอกมาต่อร่วมด้วย ขาของ CPU ทุกขาจะถูกนำไปใช้ทำหน้าที่เป็น I/O Port ทั้งหมด

ขอให้พึงระลึกไว้เสมอว่า คำสั่งเทียม นี้เป็นคำสั่งเพิ่มเติมที่โปรแกรม Assembler สร้างขึ้นมาเพื่อ อำนวยความสะดวกให้กับผู้เขียนโปรแกรม ซึ่งรูปแบบของคำสั่งเทียมต่างๆเหล่านี้ โปรแกรม Assembler แต่ละโปรแกรมอาจมีมากน้อยไม่เท่ากัน และรูปแบบในการใช้งานคำสั่งเทียมต่างๆ ก็จะไม่เหมือนกันด้วย ขึ้นอยู่กับข้อกำหนด ของโปรแกรม Assembler ที่จะนำมาใช้ในการแปลคำสั่งโปรแกรม ดังนั้นในการที่จะ เขียนโปรแกรมแต่ละครั้งควรศึกษาถึงรูปแบบและข้อกำหนดของคำสั่งเทียมต่างๆให้เข้าใจเสียก่อน เพื่อที่จะได้เขียนโปรแกรมให้ถูกต้องตามรูปแบบและข้อกำหนดที่โปรแกรม Assembler นั้นๆ กำหนดไว้ ตัวอย่างเช่น เมื่อเขียนโปรแกรมตามข้อกำหนดของโปรแกรม Assembler ชื่อ SXA51 แล้วนำโปรแกรม นั้นไปให้โปรแกรม Assembler ชื่อ CROSS-32 ทำการแปลให้แล้ว อาจเกิดการ Error ขึ้นเป็นจำนวนมาก เนื่องจากรูปแบบและข้อกำหนดต่างๆของโปรแกรมทั้งสองตัวนี้จะไม่เหมือนกัน โดยผู้ใช้สามารถศึกษาเพิ่มเติมได้จากตัวอย่างโปรแกรมต่างๆเพื่อประกอบความเข้าใจ